

Ejercicios de Java resueltos

Los 153 ejercicios de Java de la web en una sola hoja: cada uno con su enunciado, los datos que necesitas (código de partida, ejemplos de entrada y salida o la base de datos) y, al final, la solución explicada.

Índice

Entornos de Desarrollo

- [1. Gráfico de trabajo pendiente \(burndown\) de un sprint](#)
- [3. Depura el resumen de notas](#)
- [5. Valores límite: casos que detectan errores](#)
- [7. Un analizador de código en miniatura](#)
- [9. Dobles de prueba escritos a mano](#)
- [11. Del diagrama de estados al código](#)
- [13. Depura el programa de notas](#)
- [15. Caja negra: el problema del triángulo](#)
- [17. Del diagrama de clases al código](#)

Programación

- [18. Del pseudocódigo a Java](#)
- [20. Conversor de bases sin atajos](#)
- [22. Desglose de segundos](#)
- [24. La cuota de un préstamo](#)
- [26. Calificación a partir de la nota](#)
- [28. Encuentra el fallo: la nota en palabras](#)
- [30. Validar una fecha y calcular el día siguiente](#)
- [32. Tabla de multiplicar](#)
- [34. Suma de los dígitos](#)
- [36. La conjetura de Collatz](#)
- [38. Números primos con una función](#)
- [40. Encuentra el fallo: el método que no cambia nada](#)
- [42. Una calculadora con descenso recursivo](#)
- [44. Encuentra el fallo: el máximo de temperaturas bajo cero](#)
- [46. Estadísticas completas de notas](#)
- [48. El juego de la vida de Conway](#)
- [50. Encuentra el fallo: la contraseña que nunca es correcta](#)
- [52. Encuentra el fallo: el palíndromo exigente](#)
- [54. Justificar un texto](#)
- [56. Una clase Fraccion inmutable](#)
- [58. Cuentas bancarias con herencia](#)
- [60. Expresiones matemáticas con polimorfismo](#)
- [62. Reglas de contraseñas componibles](#)
- [64. Comparar dos ficheros \(diff\)](#)
- [66. Encuentra el fallo: el contador de palabras olvidadizo](#)
- [68. Ruta más corta en el metro](#)

[2. Planificación con dependencias y camino crítico](#)

- [4. Depura una búsqueda binaria](#)
- [6. Matar mutantes con varias condiciones](#)
- [8. Un Git en miniatura: base de fusión](#)
- [10. Tu propio JUnit con reflexión](#)
- [12. Ingeniería inversa: de las clases al diagrama](#)
- [14. Valores límite: clasificar edades](#)
- [16. Refactoriza: extraer método](#)
- [19. Traza del algoritmo de Euclides](#)
- [21. Simula una máquina con su propio ensamblador](#)
- [23. Las trampas de los tipos numéricos](#)
- [25. Permisos de Unix con operadores de bits](#)
- [27. Encuentra el fallo: años bisiestos](#)
- [29. Año bisiesto](#)
- [31. IRPF por tramos \(simplificado\)](#)
- [33. Encuentra el fallo: la suma de 1 a n](#)
- [35. Encuentra el fallo: la media que pierde decimales](#)
- [37. El calendario de un mes](#)
- [39. Factorial recursivo](#)
- [41. Las torres de Hanói](#)
- [43. Máximo y mínimo de un array](#)
- [45. Invertir un array](#)
- [47. Suma de filas y columnas de una matriz](#)
- [49. Contar vocales](#)
- [51. Palíndromos](#)
- [53. El cifrado de Vigenère](#)
- [55. Encuentra el fallo: el alumno sin nombre](#)
- [57. Máquina expendedora](#)
- [59. Encuentra el fallo: el producto que no está en la lista](#)
- [61. Clasificación de una liga con Comparator](#)
- [63. Validar y resumir un CSV de ventas](#)
- [65. Encuentra el fallo: borrar mientras se recorre](#)
- [67. Agrupar anagramas](#)

Acceso a Datos

- [69. Media de notas desde un fichero CSV](#)
- [71. Leer un XML con DOM](#)
- [73. Un CSV de verdad: comillas, separadores y saltos de línea](#)
- [75. Sentencias preparadas frente a concatenación](#)
- [77. Transacciones con puntos de guardado y bloqueos](#)
- [79. Los estados de una entidad JPA](#)
- [81. Profundidad de activación en una base de objetos](#)
- [83. Propiedad ligada \(bound property\)](#)
- [85. Persistencia de un JavaBean en XML](#)
- [87. Un contenedor de inyección de dependencias](#)
- [89. Conversor de CSV a JSON](#)

Desarrollo de Interfaces

- [91. La validación de un formulario de registro](#)
- [93. Captura y burbuja de un evento](#)
- [95. Clasificar gestos táctiles](#)
- [97. Un componente de valoración por estrellas](#)
- [99. Un campo de texto con máscara](#)
- [101. El orden de tabulación de un formulario](#)
- [103. El CHANGELOG a partir de los commits](#)
- [105. Un revisor automático de manuales](#)
- [107. ¿Puedo distribuir mi aplicación con estas librerías?](#)
- [109. Casos de prueba por valores límite](#)
- [111. Pruebas de interfaz guionizadas](#)
- [113. Informe agrupado con subtotales](#)
- [115. Comparar versiones semánticas](#)

Programación Multimedia y Dispositivos Móviles

- [116. Densidades de pantalla y recursos de Android](#)
- [118. Elegir el minSdk de una app](#)
- [120. La pila de navegación de una app](#)
- [122. Física con paso fijo, acumulador e interpolación](#)
- [124. El efecto túnel: detección discreta frente a continua](#)
- [126. Una generación del juego de la vida](#)
- [128. Camino más corto en un laberinto](#)

Programación de Servicios y Procesos

- [129. Coordinador de procesos](#)
- [131. Suma en paralelo con hilos](#)
- [133. Servidor de un protocolo de texto](#)
- [135. Una API REST con HttpServer](#)
- [137. Guardar contraseñas con PBKDF2](#)

Ejercicios largos

- [139. Biblioteca: préstamos, devoluciones y multas](#)
- [141. Cajero automático con PIN, límite diario y billetes](#)
- [143. Cifrado de Vigenère con el alfabeto español](#)
- [145. Evaluador de expresiones con pilas y variables](#)

- [70. Serializar una lista de productos](#)
- [72. Acceso aleatorio a registros](#)
- [74. Leer un formato binario propio con versión y CRC](#)
- [76. Un pool de conexiones por dentro](#)
- [78. El problema N+1 y las estrategias de carga](#)
- [80. Arrays y tipos compuestos de PostgreSQL](#)
- [82. Herencia de tablas en PostgreSQL](#)
- [84. Propiedad restringida \(constrained\)](#)
- [86. Componente DAO con eventos](#)
- [88. Filtrar un fichero de log](#)
- [90. Media de precios por categoría \(DOM\)](#)

- [92. Repartir el ancho como un GridPane](#)
- [94. Un intérprete de órdenes de voz](#)
- [96. ¿Qué gesto hace la mano?](#)
- [98. Propiedades observables y bindings](#)
- [100. Contraste de colores según WCAG](#)
- [102. ¿Se entiende este texto?](#)
- [104. De Javadoc a una página de documentación](#)
- [106. Versiones semánticas y rangos de dependencias](#)
- [108. Un actualizador incremental con despliegue escalonado](#)
- [110. Un mini ejecutor de pruebas unitarias](#)
- [112. Validar un NIF en un formulario](#)
- [114. Paginación de una tabla](#)

- [117. El ciclo de vida de una Activity](#)
- [119. Las diferencias de una lista, como DiffUtil](#)
- [121. Sincronización sin conexión con conflictos](#)
- [123. Colisiones con rejilla espacial](#)
- [125. ¿Quién gana al tres en raya?](#)

- [127. Puntuación con combos](#)

- [130. Planificador de CPU: FCFS, SJF y Round Robin](#)
- [132. Productor-consumidor con un búfer acotado](#)
- [134. Mensajes fiables sobre UDP](#)
- [136. Disyuntor \(circuit breaker\) para un servicio inestable](#)
- [138. Tokens firmados con HMAC \(como un JWT\)](#)

- [140. Clasificación de una liga de fútbol](#)
- [142. Buscaminas: números, casilla más peligrosa y zonas despejadas](#)
- [144. Parking con tarifas por tipo de vehículo](#)
- [146. Conversor de CSV a JSON con comillas y tipos](#)

147. Nóminas de una plantilla con distintos contratos

149. Validador de DNI, NIE e IBAN diseñado con clases de equivalencia

151. Servidor de órdenes con sockets e hilos

153. Depura el programa de notas: cinco errores escondidos

148. Contador de palabras en paralelo con un pool de hilos

150. Fichero de productos con registros de tamaño fijo y acceso directo

152. Almacén de contraseñas seguro con PBKDF2 y sal

Entornos de Desarrollo

1. Gráfico de trabajo pendiente (burndown) de un sprint

Difícil · Desarrollo de Software · apuntesdam.com/subject/entornos/topic/desarrollo-software

En Scrum, el burndown muestra cada día cuántos puntos de historia quedan por terminar. La entrada es la duración del sprint en días y, después, cada historia con su identificador, sus puntos y el día en que se terminó (0 si no se terminó). Muestra la cabecera «Sprint de D días, T puntos comprometidos» y, para cada día de 0 a D, una línea con el formato «Día %2d | barra | %2d (ideal %.1f)», donde la barra son tantos # como puntos quedan, rellena con espacios hasta T caracteres, y el ideal es la recta que baja de T a 0 ($T - T \cdot d/D$). Si quedan más puntos que el ideal redondeado hacia arriba, añade « (retraso)». Una historia solo descuenta sus puntos al terminarse (no hay puntos a medias). Al final, la velocidad (puntos terminados) y las historias que vuelven al product backlog (o «ninguna»).

Código de partida (java)

```
import java.util.*;

public class Main {
    record Historia(String id, int puntos, int diaTerminada) { } // diaTerminada = 0 si no se terminó

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int dias = sc.nextInt();
        List<Historia> hs = new ArrayList<>();
        while (sc.hasNext()) hs.add(new Historia(sc.next(), sc.nextInt(), sc.nextInt()));
        // TODO: mostrar el burndown día a día, la velocidad y las historias pendientes
    }
}
```

Ejemplo: Sprint que acaba bien

ENTRADA

```
5
H1 3 2
H2 5 4
H3 2 5
```

SALIDA ESPERADA

```
Sprint de 5 días, 10 puntos comprometidos
Día 0 | ##### | 10 (ideal 10.0)
Día 1 | ##### | 10 (ideal 8.0) (retraso)
Día 2 | ##### | 7 (ideal 6.0) (retraso)
Día 3 | ##### | 7 (ideal 4.0) (retraso)
Día 4 | ## | 2 (ideal 2.0)
Día 5 | | 0 (ideal 0.0)
Velocidad: 10 puntos
Vuelven al product backlog: ninguna
```

Ejemplo: Sprint con retraso

ENTRADA

```
4
H1 5 3
H2 3 0
H3 2 4
H4 1 1
```

SALIDA ESPERADA

```
Sprint de 4 días, 11 puntos comprometidos
Día 0 | ##### | 11 (ideal 11.0)
Día 1 | ##### | 10 (ideal 8.3) (retraso)
Día 2 | ##### | 10 (ideal 5.5) (retraso)
Día 3 | ##### | 5 (ideal 2.8) (retraso)
Día 4 | ### | 3 (ideal 0.0) (retraso)
Velocidad: 8 puntos
Vuelven al product backlog: H2
```

2. Planificación con dependencias y camino crítico

Muy difícil · Desarrollo de Software · apuntesdam.com/subject/entornos/topic/desarrollo-software

Para planificar un proyecto se descompone en tareas con su duración y las tareas que deben terminar antes (como en un diagrama de Gantt o PERT). Cada línea de la entrada es «ID duración» y, opcionalmente, la lista de tareas previas separadas por comas. Si una tarea depende de otra que no existe, muestra «Error: X depende de Y, que no existe». Calcula un orden topológico con el algoritmo de Kahn: empieza por las tareas sin previas en el orden de la entrada y usa

una cola; cada tarea, al salir, libera a las que dependen de ella (en el orden en que se declararon). Si no se pueden ordenar todas, «Error: hay dependencias circulares». Después calcula el inicio más temprano (el máximo fin de sus previas) y el fin de cada tarea, la duración del proyecto, el fin más tardío de cada tarea sin retrasar el proyecto (recorriendo el orden al revés) y la holgura (fin tardío – fin temprano). Muestra el orden, una línea por tarea con «%-3s inicio %2d fin %2d holgura %d», la duración y el camino crítico: las tareas con holgura 0, en orden topológico, unidas por « → ».

Código de partida (java)

```
import java.util.*;

public class Main {
    record Tarea(String id, int duracion, List<String> previas) { }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, Tarea> tareas = new LinkedHashMap<>();
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (l.isEmpty()) continue;
            // Formato: ID duración [previa1,previa2,...]
            String[] p = l.split("\\s+");
            List<String> previas = p.length > 2 ? List.of(p[2].split(",")) : List.of();
            tareas.put(p[0], new Tarea(p[0], Integer.parseInt(p[1]), previas));
        }
        // TODO: orden topológico, inicio y fin más tempranos, duración del proyecto,
        //         holgura de cada tarea y camino crítico (o error si hay un ciclo)
    }
}
```

Ejemplo: Proyecto web

ENTRADA

```
A 3
B 2 A
C 4 A
D 1 B,C
E 2 D
```

SALIDA ESPERADA

```
Orden: A B C D E
A inicio 0 fin 3 holgura 0
B inicio 3 fin 5 holgura 2
C inicio 3 fin 7 holgura 0
D inicio 7 fin 8 holgura 0
E inicio 8 fin 10 holgura 0
Duración del proyecto: 10 días
Camino crítico: A → C → D → E
```

Ejemplo: Dos ramas

ENTRADA

```
Analisis 5
BD 3 Analisis
API 6 BD
UI 4 Analisis
Pruebas 2 API,UI
Manual 1 UI
```

SALIDA ESPERADA

```
Orden: Analisis BD UI API Manual Pruebas
Analisis inicio 0 fin 5 holgura 0
BD inicio 5 fin 8 holgura 0
UI inicio 5 fin 9 holgura 5
API inicio 8 fin 14 holgura 0
Manual inicio 9 fin 10 holgura 6
Pruebas inicio 14 fin 16 holgura 0
Duración del proyecto: 16 días
Camino crítico: Analisis → BD → API → Pruebas
```

3. Depura el resumen de notas

Difícil · Instalación y uso de IDEs · apuntesdam.com/subject/entornos/topic/ides

El método resumen debería calcular la media (con decimales), cuántas notas son aprobados (5 o más), la nota máxima y la mínima de un grupo de notas enteras de 0 a 10. Compila y parece funcionar, pero tiene cuatro errores. Usa el depurador de tu IDE (pon un punto de ruptura dentro del bucle y observa i, suma, max y min en cada vuelta) o añade trazas temporales, encuentra los cuatro y corrígelos sin reescribir el método desde cero.

Código de partida (java)

```
import java.util.*;

public class Main {
    // Este código compila pero tiene CUATRO errores. Encuéntralos con el depurador (o con trazas) y corrígelos.
    static String resumen(int[] notas) {
        int suma = 0, aprobados = 0, max = 0, min = 0;
        for (int i = 1; i < notas.length; i++) {
            suma += notas[i];
            if (notas[i] > 5) aprobados++;
            if (notas[i] > max) max = notas[i];
            if (notas[i] < min) min = notas[i];
        }
        double media = suma / notas.length;
        return String.format(Locale.US, "media %.2f, aprobados %d de %d, máxima %d, mínima %d",
            media, aprobados, notas.length, max, min);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (l.isEmpty()) continue;
            int[] notas = Arrays.stream(l.split("\\s+")).mapToInt(Integer::parseInt).toArray();
            System.out.println(Arrays.toString(notas) + " → " + resumen(notas));
        }
    }
}
```

Ejemplo: Grupo variado

ENTRADA

7 4 9 5 10

SALIDA ESPERADA

[7, 4, 9, 5, 10] → media 7.00, aprobados 4 de 5, máxima 10, mínima 4

Ejemplo: Todos suspensos

ENTRADA

3 1 4

SALIDA ESPERADA

[3, 1, 4] → media 2.67, aprobados 0 de 3, máxima 4, mínima 1

4. Depura una búsqueda binaria

Muy difícil · Instalación y uso de IDEs · apuntesdam.com/subject/entornos/topic/ides

`primeraMayorIgual` debe devolver la primera posición de un array ordenado cuyo valor es mayor o igual que `x`, o `a.length` si todos son menores (es el «lower bound» que usan `Arrays.binarySearch` y las bases de datos en sus índices). Con ella, contar calcula cuántas veces aparece `x`. La versión actual tiene varios errores: según el caso, se queda en un bucle infinito, devuelve una posición equivocada o nunca devuelve `a.length`. Depúrala con casos pequeños (arrays de 1 o 2 elementos), anotando lo, hi y medio en cada vuelta, y corrígela. Además, el cálculo de medio puede desbordarse con arrays de más de mil millones de elementos: cámbialo por una forma segura.

Código de partida (java)

```
import java.util.*;

public class Main {
    // Primera posición i tal que a[i] >= x (a.length si no hay ninguna). El array está ordenado.
    // Tiene errores que solo aparecen en algunos casos: encuéntralos.
    static int primeraMayorOIgual(int[] a, int x) {
        int lo = 0, hi = a.length - 1;
        int vueltas = 0; // protección para que no se cuelgue mientras lo depuras
        while (lo < hi && vueltas++ < 100) {
            int medio = (lo + hi) / 2;
            if (a[medio] < x) lo = medio;
            else hi = medio - 1;
        }
        return lo;
    }

    static int contar(int[] a, int x) {
        return primeraMayorOIgual(a, x + 1) - primeraMayorOIgual(a, x);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int[] a = Arrays.stream(sc.nextLine().trim().split("\\s+")).mapToInt(Integer::parseInt).toArray();
        while (sc.hasNextInt()) {
            int x = sc.nextInt();
            System.out.println("x=" + x + ": primera posición >= x → " + primeraMayorOIgual(a, x) + ",
                apariciones → " + contar(a, x));
        }
    }
}
```

Ejemplo: Con repetidos

ENTRADA

```
1 3 3 3 5 8
3
4
0
9
```

SALIDA ESPERADA

```
x=3: primera posición >= x → 1, apariciones → 3
x=4: primera posición >= x → 4, apariciones → 0
x=0: primera posición >= x → 0, apariciones → 0
x=9: primera posición >= x → 6, apariciones → 0
```

Ejemplo: Arrays pequeños

ENTRADA

```
7
7
8
6
```

SALIDA ESPERADA

```
x=7: primera posición >= x → 0, apariciones → 1
x=8: primera posición >= x → 1, apariciones → 0
x=6: primera posición >= x → 0, apariciones → 0
```

5. Valores límite: casos que detectan errores

Difícil · Diseño y realización de pruebas · apuntesdam.com/subject/entornos/topic/disenio-pruebas

La técnica de valores límite dice que los errores se esconden en las fronteras entre clases de equivalencia. El método tarifa ya está bien hecho; tu trabajo es escribir en CASOS las pruebas, cada una con un peso y la tarifa esperada. El programa comprueba que tus casos pasan con el método correcto y después los ejecuta contra seis versiones con un error cada una (mutantes), como los que comete cualquier programador: un < en vez de <=, un precio equivocado o un límite olvidado. Un mutante queda «detectado» si alguno de tus casos da un resultado distinto. Consigue detectar los seis con casos razonados, no al azar.

Código de partida (java)

```
import java.util.*;
import java.util.function.DoubleUnaryOperator;

public class Main {
    // Tarifa de envío según el peso en kg. -1 significa «no admitido».
    // peso <= 0      → -1
    // 0 < peso <= 1  → 4.5
    // 1 < peso <= 5  → 7
    // 5 < peso <= 20 → 12
    // peso > 20     → -1
    static double tarifa(double peso) {
        if (peso <= 0) return -1;
        if (peso <= 1) return 4.5;
        if (peso <= 5) return 7;
        if (peso <= 20) return 12;
        return -1;
    }

    // TODO: escribe aquí tus casos de prueba: {peso, resultado esperado}
    static final double[][] CASOS = {
        {3, 7},
    };

    // ---- No modifiques lo que sigue: versiones con un error cada una («mutantes»)
    static final Map<String, DoubleUnaryOperator> MUTANTES = new LinkedHashMap<>();
    static {
        MUTANTES.put("M1 (< 1 en vez de <= 1)", p -> p <= 0 ? -1 : p < 1 ? 4.5 : p <= 5 ? 7 : p <= 20 ? 12 :
            -1);
        MUTANTES.put("M2 (< 5 en vez de <= 5)", p -> p <= 0 ? -1 : p <= 1 ? 4.5 : p < 5 ? 7 : p <= 20 ? 12 :
            -1);
        MUTANTES.put("M3 (< 20 en vez de <= 20)", p -> p <= 0 ? -1 : p <= 1 ? 4.5 : p <= 5 ? 7 : p < 20 ? 12 :
            -1);
        MUTANTES.put("M4 (< 0 en vez de <= 0)", p -> p < 0 ? -1 : p <= 1 ? 4.5 : p <= 5 ? 7 : p <= 20 ? 12 :
            -1);
        MUTANTES.put("M5 (sin límite superior)", p -> p <= 0 ? -1 : p <= 1 ? 4.5 : p <= 5 ? 7 : 12);
        MUTANTES.put("M6 (7.5 en vez de 7)", p -> p <= 0 ? -1 : p <= 1 ? 4.5 : p <= 5 ? 7.5 : p <= 20 ? 12 :
            -1);
    }

    public static void main(String[] args) {
        String fallo = null;
        for (double[] c : CASOS)
            if (tarifa(c[0]) != c[1] && fallo == null) fallo = "tarifa(" + c[0] + ") da " + tarifa(c[0]) +
                ", no " + c[1];
        System.out.println("Casos correctos con el original: " + (fallo == null ? "sí" : "no (" + fallo +
            ")"));
        List<String> vivos = new ArrayList<>();
        for (var m : MUTANTES.entrySet()) {
            boolean detectado = false;
            for (double[] c : CASOS) if (m.getValue().applyAsDouble(c[0]) != c[1]) detectado = true;
            if (!detectado) vivos.add(m.getKey());
        }
        System.out.println("Mutantes detectados: " + (MUTANTES.size() - vivos.size()) + " de " +
            MUTANTES.size());
        System.out.println("Sobreviven: " + (vivos.isEmpty() ? "ninguno" : String.join(", ", vivos)));
    }
}
```

Ejemplo: Detección de mutantes

SALIDA ESPERADA

```
Casos correctos con el original: sí
Mutantes detectados: 6 de 6
Sobreviven: ninguno
```


6. Matar mutantes con varias condiciones

Muy difícil · Diseño y realización de pruebas · apuntesdam.com/subject/entornos/topic/disenio-pruebas

El precio de la entrada de un museo depende de la edad, de si se es socio y de si es fin de semana (las reglas están en el comentario de ORIGINAL). Escribe en CASOS las pruebas necesarias, cada una con la edad, si es socio (1/0), si es fin de semana (1/0) y el precio esperado, para detectar los mutantes: errores en los límites de edad, en el orden en que se aplican los descuentos, en el tope de 0 € o en cómo se combinan las condiciones. Atención: uno de los ocho es un mutante equivalente, que se comporta exactamente igual que el original con cualquier entrada posible, así que ningún caso puede detectarlo. Detecta los otros siete y averigua cuál es el equivalente y por qué. Con muchas condiciones, probar todas las combinaciones crece muy deprisa: busca un conjunto pequeño (con unos 12 casos basta) en el que cada condición y cada límite influyan en algún resultado.

```

import java.util.*;

public class Main {
    interface Precio { double calcular(int edad, boolean socio, boolean finDeSemana); }

    // Precio de la entrada a un museo:
    //  edad < 0 → -1 (error)
    //  base 10 €; los fines de semana, 12 €
    //  menores de 12: 50 % de descuento; desde 65: 40 % de descuento
    //  los socios tienen además 2 € menos sobre el precio ya descontado (nunca por debajo de 0)
    static final Precio ORIGINAL = (edad, socio, finde) -> {
        if (edad < 0) return -1;
        double p = finde ? 12 : 10;
        if (edad < 12) p *= 0.5;
        else if (edad >= 65) p *= 0.6;
        if (socio) p = Math.max(0, p - 2);
        return p;
    };

    // TODO: tus casos: {edad, socio (0/1), fin de semana (0/1), precio esperado}
    static final double[][] CASOS = {
        {30, 0, 0, 10},
    };

    // ---- No modifique lo que sigue
    static final Map<String, Precio> MUTANTES = new LinkedHashMap<>();
    static {
        MUTANTES.put("M1 (<= 12)", (e, s, f) -> e < 0 ? -1 : aplicar(f ? 12 : 10, e <= 12, e >= 65, s));
        MUTANTES.put("M2 (> 65)", (e, s, f) -> e < 0 ? -1 : aplicar(f ? 12 : 10, e < 12, e > 65, s));
        MUTANTES.put("M3 (<= 0 es error)", (e, s, f) -> e <= 0 ? -1 : aplicar(f ? 12 : 10, e < 12, e >= 65, s));
        MUTANTES.put("M4 (socio antes del %)", (e, s, f) -> {
            if (e < 0) return -1;
            double p = (f ? 12 : 10) - (s ? 2 : 0);
            return e < 12 ? p * 0.5 : e >= 65 ? p * 0.6 : p;
        });
        MUTANTES.put("M5 (sin tope en 0)", (e, s, f) -> {
            if (e < 0) return -1;
            double p = f ? 12 : 10;
            if (e < 12) p *= 0.5; else if (e >= 65) p *= 0.6;
            return s ? p - 2 : p;
        });
        MUTANTES.put("M6 (ignora el fin de semana con descuento)", (e, s, f) -> e < 0 ? -1 : aplicar((f && e >= 12 && e < 65) ? 12 : 10, e < 12, e >= 65, s));
        MUTANTES.put("M7 (el socio solo descuenta a adultos)", (e, s, f) -> e < 0 ? -1 : aplicar(f ? 12 : 10, e < 12, e >= 65, s && e >= 12 && e < 65));
        MUTANTES.put("M8 (65 o más: 50 %)", (e, s, f) -> {
            if (e < 0) return -1;
            double p = f ? 12 : 10;
            if (e < 12 || e >= 65) p *= 0.5;
            if (s) p = Math.max(0, p - 2);
            return p;
        });
    };

    static double aplicar(double p, boolean nino, boolean mayor, boolean socio) {
        if (nino) p *= 0.5; else if (mayor) p *= 0.6;
        if (socio) p = Math.max(0, p - 2);
        return p;
    }

    static boolean igual(double a, double b) { return Math.abs(a - b) < 1e-9; }

    public static void main(String[] args) {
        String fallo = null;
        for (double[] c : CASOS) {
            double r = ORIGINAL.calcular((int) c[0], c[1] == 1, c[2] == 1);
            if (!igual(r, c[3]) && fallo == null) fallo = Arrays.toString(c) + " da " + r;
        }
        System.out.println("Casos correctos con el original: " + (fallo == null ? "sí" : "no (" + fallo + ")"));
        List<String> vivos = new ArrayList<>();
    }
}

```

```

    for (var m : MUTANTES.entrySet()) {
        boolean detectado = false;
        for (double[] c : CASOS)
            if (!igual(m.getValue().calcular((int) c[0], c[1] == 1, c[2] == 1), c[3])) detectado = true;
        if (!detectado) vivos.add(m.getKey());
    }
    System.out.println("Mutantes detectados: " + (MUTANTES.size() - vivos.size()) + " de " +
        MUTANTES.size());
    System.out.println("Sobreviven: " + (vivos.isEmpty() ? "ninguno" : String.join(", ", vivos)));
}
}

```

Ejemplo: Detección de mutantes

SALIDA ESPERADA

```

Casos correctos con el original: sí
Mutantes detectados: 7 de 8
Sobreviven: M5 (sin tope en 0)

```

7. Un analizador de código en miniatura

Difícil · Optimización y Documentación · apuntesdam.com/subject/entornos/topic/optimizacion-documentacion

Herramientas como Checkstyle o PMD leen el código fuente y avisan de los «malos olores». Implementa analizar con cuatro reglas, recorriendo las líneas en orden. Detecta la cabecera de un método con la expresión regular METODO (el grupo 2 es el nombre y el 3, los parámetros). Regla 1: si el nombre no sigue lowerCamelCase (empieza por minúscula y solo tiene letras y cifras), «Línea n: el método «X» no sigue lowerCamelCase». Regla 2: si tiene más de MAX_PARAMETROS parámetros, «Línea n: «X» tiene p parámetros (máximo 3)». Regla 3: en cada línea que no declare una constante (que no contenga «final»), cada número distinto de 0 y 1 que no esté dentro de una cadena ni de un comentario // es «Línea n: número mágico N». Regla 4: un método que, desde su cabecera hasta la llave que lo cierra (cuenta { y }), mide más de MAX_LINEAS_METODO líneas da «Línea inicio: «X» mide L líneas (máximo 8)», y este aviso se añade al encontrar la llave de cierre. El main muestra los avisos y el total.

Código de partida (java)

```

import java.util.*;
import java.util.regex.*;

public class Main {
    static final int MAX_LINEAS_METODO = 8, MAX_PARAMETROS = 3;
    static final Pattern METODO = Pattern.compile(
        "^\\s*(public|private|protected|static|\\s)+[\\w<\\[\\]]+\\s+(\\w+)\\s*\\(((\\^)*\\))\\s*\\{";

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<String> lineas = new ArrayList<>();
        while (sc.hasNextLine()) lineas.add(sc.nextLine());
        List<String> avisos = analizar(lineas);
        avisos.forEach(System.out::println);
        System.out.println(avisos.size() + (avisos.size() == 1 ? " aviso" : " avisos"));
    }

    static List<String> analizar(List<String> lineas) {
        List<String> avisos = new ArrayList<>();
        // TODO: recorrer las líneas y aplicar las cuatro reglas
        return avisos;
    }
}

```

Ejemplo: Nombre y número mágico

ENTRADA

```
public class Factura {
    static final double IVA = 0.21;

    public double total(double base) {
        return base * (1 + IVA);
    }

    public double Descuento(double total, int
unidades) {
        if (unidades > 10) return total * 0.95;
        return total;
    }
}
```

SALIDA ESPERADA

```
Línea 8: el método «Descuento» no sigue
lowerCamelCase
Línea 9: número mágico 10
Línea 9: número mágico 0.95
3 avisos
```

Ejemplo: Método largo con muchos parámetros

ENTRADA

```
public class Pedidos {
    private double calcular(String cliente, int
unidades, double precio, boolean urgente, String
pais) {
        double total = unidades * precio;
        if (urgente) {
            total += 15;
        }
        if (pais.equals("ES")) {
            total *= 1.21;
        }
        String msg = "Pedido de 99 unidades";
        return total;
    }

    public static void main(String[] args) {
        System.out.println(new
Pedidos().calcular("ana", 3, 2.5, true, "ES"));
    }
}
```

SALIDA ESPERADA

```
Línea 2: «calcular» tiene 5 parámetros (máximo 3)
Línea 5: número mágico 15
Línea 8: número mágico 1.21
Línea 2: «calcular» mide 11 líneas (máximo 8)
Línea 15: número mágico 3
Línea 15: número mágico 2.5
6 avisos
```

8. Un Git en miniatura: base de fusión

Muy difícil · Optimización y Documentación · apuntesdam.com/subject/entornos/topic/optimizacion-documentacion

En Git, cada commit apunta a sus padres (uno normalmente, dos si es una fusión) y una rama es solo un nombre que apunta a un commit. Para fusionar, Git busca la base de fusión: el antepasado común más reciente de los dos commits. Implementa antepasados (el propio commit y todos los que se alcanzan siguiendo padres) y baseDeFusion: de los antepasados comunes, el que no es antepasado de ningún otro común (si hubiera varios, el primero en orden de creación; si no hay ninguno, null). Con eso, el main decide qué haría git merge: «ya actualizado» si el origen es la base, «fast-forward» si el destino es la base (basta con mover la rama) o «fusión de tres vías» en otro caso.

Código de partida (java)

```
import java.util.*;

public class Main {
    static final Map<String, List<String>> padres = new LinkedHashMap<>();    // commit → sus padres
    static final Map<String, String> ramas = new TreeMap<>();                // rama → commit al que apunta

    // Todos los antepasados de un commit, incluido él mismo
    static Set<String> antepasados(String c) {
        // TODO
        return new HashSet<>();
    }

    // Antepasado común «más reciente» (el que no es antepasado de otro antepasado común); null si no hay
    static String baseDeFusion(String a, String b) {
        // TODO
        return null;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] t = sc.nextLine().trim().split("\\s+");
            switch (t[0]) {
                case "COMMIT" -> padres.put(t[1], t.length > 2 ? List.of(t[2].split(",")) : List.of());
                case "RAMA" -> ramas.put(t[1], t[2]);
                case "BASE" -> System.out.println("merge-base " + t[1] + " " + t[2] + " → " +
                    baseDeFusion(ramas.get(t[1]), ramas.get(t[2])));
                case "FUSIONAR" -> { // FUSIONAR destino origen: ¿qué haría git merge origen estando
                    en destino?
                        String d = ramas.get(t[1]), o = ramas.get(t[2]);
                        String base = baseDeFusion(d, o);
                        String r;
                        if (o.equals(base)) r = "ya actualizado (nada que fusionar)";
                        else if (d.equals(base)) r = "fast-forward: " + t[1] + " pasa de " + d + " a " + o;
                        else r = "fusión de tres vías (base " + base + ")": se crea un commit de fusión";
                        System.out.println("git merge " + t[2] + " en " + t[1] + " → " + r);
                    }
            }
        }
    }
}
```

Ejemplo: Ramas divergentes

ENTRADA

```
COMMIT c1
COMMIT c2 c1
COMMIT c3 c2
COMMIT c4 c3
COMMIT c5 c2
COMMIT c6 c5
RAMA main c4
RAMA login c6
RAMA hotfix c3
BASE main login
FUSIONAR main login
FUSIONAR main hotfix
FUSIONAR hotfix main
```

SALIDA ESPERADA

```
merge-base main login → c2
git merge login en main → fusión de tres vías
(base c2): se crea un commit de fusión
git merge hotfix en main → ya actualizado (nada
que fusionar)
git merge main en hotfix → fast-forward: hotfix
pasa de c3 a c4
```

Ejemplo: Después de fusionar

ENTRADA

```
COMMIT c1
COMMIT c2 c1
COMMIT c3 c2
COMMIT c4 c3
COMMIT c5 c2
COMMIT c6 c5
RAMA main c4
RAMA login c6
RAMA hotfix c3
COMMIT c7 c4,c6
RAMA main c7
BASE main login
FUSIONAR main login
FUSIONAR login main
```

SALIDA ESPERADA

```
merge-base main login → c6
git merge login en main → ya actualizado (nada que
fusionar)
git merge main en login → fast-forward: login pasa
de c6 a c7
```

9. Dobles de prueba escritos a mano

Difícil · Pruebas con JUnit y Mockito · apuntesdam.com/subject/entornos/topic/pruebas-junit

Para probar ServicioPedidos sin una base de datos ni un servidor de correo reales se usan dobles de prueba: objetos que implementan las mismas interfaces. Completa InventarioFalso, cuyo constructor recibe el stock que devolverá stock() para cualquier producto y que anota cada llamada a descontar como «producto -unidades» en una lista que devuelve descuentos(), y CorreoEspia, que no envía nada: guarda cada envío como «destinatario | asunto» en una lista que devuelve enviados(). Es lo que hace Mockito automáticamente con when(...).thenReturn(...) y verify(...). El main ya prueba el servicio con distintos casos y muestra lo que han registrado tus dobles.

Código de partida (java)

```
import java.util.*;

interface Inventario {
    int stock(String producto);
    void descontar(String producto, int unidades);
}

interface Correo {
    void enviar(String destinatario, String asunto);
}

// Clase que se quiere probar: no sabe si habla con una base de datos real o con un doble
class ServicioPedidos {
    private final Inventario inventario;
    private final Correo correo;

    ServicioPedidos(Inventario inventario, Correo correo) {
        this.inventario = inventario;
        this.correo = correo;
    }

    boolean pedir(String cliente, String producto, int unidades) {
        if (unidades <= 0) return false;
        if (inventario.stock(producto) < unidades) {
            correo.enviar("almacen@tienda.es", "Sin stock: " + producto);
            return false;
        }
        inventario.descontar(producto, unidades);
        correo.enviar(cliente, "Pedido confirmado: " + unidades + " x " + producto);
        return true;
    }
}

// TODO: un doble de Inventario con un stock fijado por la prueba que anote las llamadas a descontar
class InventarioFalso implements Inventario {
    InventarioFalso(int stockFijo) { }
    public int stock(String producto) { return 0; }
    public void descontar(String producto, int unidades) { }
    List<String> descuentos() { return List.of(); }
}

// TODO: un doble de Correo que guarde cada envío como "destinatario | asunto"
class CorreoEspia implements Correo {
    public void enviar(String destinatario, String asunto) { }
    List<String> enviados() { return List.of(); }
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] t = sc.nextLine().trim().split(" "); // stock cliente producto unidades
            InventarioFalso inv = new InventarioFalso(Integer.parseInt(t[0]));
            CorreoEspia correo = new CorreoEspia();
            boolean ok = new ServicioPedidos(inv, correo).pedir(t[1], t[2], Integer.parseInt(t[3]));
            System.out.println("pedir(" + t[1] + ", " + t[2] + ", " + t[3] + ") con stock " + t[0] + " → " +
                ok);
            System.out.println("    descuentos: " + inv.descuentos());
            System.out.println("    correos: " + correo.enviados());
        }
    }
}
```

Ejemplo: Pedido correcto

ENTRADA

```
10 ana@mail.com teclado 3
```

SALIDA ESPERADA

```
pedir(ana@mail.com, teclado, 3) con stock 10 → true
descuentos: [teclado -3]
correos: [ana@mail.com | Pedido confirmado: 3 x teclado]
```

Ejemplo: Sin stock y unidades no válidas

ENTRADA

```
2 luis@mail.com raton 5
5 eva@mail.com monitor 0
```

SALIDA ESPERADA

```
pedir(luis@mail.com, raton, 5) con stock 2 → false
descuentos: []
correos: [almacen@tienda.es | Sin stock: raton]
pedir(eva@mail.com, monitor, 0) con stock 5 → false
descuentos: []
correos: []
```

10. Tu propio JUnit con reflexión

Muy difícil · Pruebas con JUnit y Mockito · apuntesdam.com/subject/entornos/topic/pruebas-junit

¿Cómo encuentra JUnit los métodos de prueba? Con reflexión: inspecciona la clase en tiempo de ejecución y busca los métodos con la anotación `@Test`. Implementa ejecutar: localiza los métodos anotados con `@Prueba` y `@Antes` de la clase, ordena las pruebas por nombre y, para cada una, crea una instancia nueva de la clase (así una prueba no afecta a otra), ejecuta los métodos `@Antes` y después la prueba. Si termina bien, «✓ nombre»; si lanza `FalloAsercion`, es un fallo: «X nombre: mensaje»; si lanza cualquier otra excepción, es un error: «!! nombre: NombreSimpleDeLaExcepción». Al final, «Pruebas: N, correctas: a, fallidas: b, con error: c». Los métodos sin anotación no se ejecutan.

Código de partida (java)

```
import java.lang.annotation.*;
import java.lang.reflect.*;
import java.util.*;

// Anotaciones propias, como @Test y @BeforeEach de JUnit
@Retention(RetentionPolicy.RUNTIME) @Target(ElementType.METHOD) @interface Prueba { }
@Retention(RetentionPolicy.RUNTIME) @Target(ElementType.METHOD) @interface Antes { }

class FalloAsercion extends RuntimeException {
    FalloAsercion(String m) { super(m); }
}

class Afirmar {
    static void iguales(Object esperado, Object real) {
        if (!Objects.equals(esperado, real)) throw new FalloAsercion("se esperaba <" + esperado + "> pero fue <" + real + ">");
    }
    static void cierto(boolean c, String mensaje) {
        if (!c) throw new FalloAsercion(mensaje);
    }
}

// ---- Las pruebas de ejemplo (no las modifique)
class PilaTest {
    private Deque<Integer> pila;

    @Antes void preparar() { pila = new ArrayDeque<>(); pila.push(1); pila.push(2); }

    @Prueba void cimaEsElUltimo() { Afirmar.iguales(2, pila.peek()); }
    @Prueba void sacarReduce() { pila.pop(); Afirmar.iguales(1, pila.size()); }
    @Prueba void cadaPruebaEmpiezaLimpia() { Afirmar.iguales(2, pila.size()); }
    @Prueba void fallaAProposito() { Afirmar.iguales(3, pila.size()); }
    @Prueba void lanzaExcepcion() { new ArrayDeque<Integer>().pop(); }
    void noEsUnaPrueba() { throw new IllegalStateException("no debería ejecutarse"); }
}

public class Main {
    // Ejecuta todos los métodos @Prueba de la clase, en orden alfabético, con una instancia nueva cada vez
    static void ejecutar(Class<?> clase) throws Exception {
        // TODO
    }

    public static void main(String[] args) throws Exception {
        ejecutar(PilaTest.class);
    }
}
```

Ejemplo: Pila de ejemplo

SALIDA ESPERADA

```
✓ cadaPruebaEmpiezaLimpia
✓ cimaEsElUltimo
X fallaAProposito: se esperaba <3> pero fue <2>
!! lanzaExcepcion: NoSuchElementException
✓ sacarReduce
Pruebas: 5, correctas: 3, fallidas: 1, con error: 1
```

11. Del diagrama de estados al código

Difícil · Diagramas UML · apuntesdam.com/subject/entornos/topic/diagramas-uml

El comentario de la clase describe el diagrama de estados de un pedido: sus estados, los eventos que provocan cada transición y las acciones asociadas. Implementa aplicar para que, según el estado actual y el evento, cambie de estado y devuelva el nuevo estado con su acción entre corchetes si la tiene («CANCELADO [devolver el dinero]», «PAGADO [reenviar]»). Un paquete perdido vuelve a PAGADO para reenviarlo, pero a la tercera pérdida el pedido pasa a CANCELADO con devolución. Cualquier evento que no tenga una transición desde el estado actual (incluidos todos los eventos en los estados finales) devuelve «transición no permitida (sigue en ESTADO)» sin cambiar nada.

Código de partida (java)

```
import java.util.*;

public class Main {
    // Diagrama de estados de un pedido (estado inicial: CARRITO)
    // CARRITO --pagar--> PAGADO
    // CARRITO --cancelar--> CANCELADO
    // PAGADO --enviar--> ENVIADO
    // PAGADO --cancelar--> CANCELADO [acción: devolver el dinero]
    // ENVIADO --entregar--> ENTREGADO
    // ENVIADO --perder--> PAGADO [acción: reenviar; como mucho 2 veces, después CANCELADO con devolución]
    // ENTREGADO y CANCELADO son estados finales
    enum Estado { CARRITO, PAGADO, ENVIADO, ENTREGADO, CANCELADO }

    static Estado estado = Estado.CARRITO;
    static int perdidas = 0;

    // Aplica un evento y devuelve el texto que describe lo ocurrido
    static String aplicar(String evento) {
        // TODO
        return "";
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNext()) {
            String ev = sc.next();
            Estado antes = estado;
            String r = aplicar(ev);
            System.out.println(antes + " --" + ev + "--> " + r);
        }
        System.out.println("Estado final: " + estado);
    }
}
```

Ejemplo: Camino feliz

ENTRADA

pagar enviar entregar

SALIDA ESPERADA

CARRITO --pagar--> PAGADO
PAGADO --enviar--> ENVIADO
ENVIADO --entregar--> ENTREGADO
Estado final: ENTREGADO

Ejemplo: Eventos no válidos y cancelación

ENTRADA

enviar pagar cancelar entregar

SALIDA ESPERADA

CARRITO --enviar--> transición no permitida (sigue en CARRITO)
CARRITO --pagar--> PAGADO
PAGADO --cancelar--> CANCELADO [devolver el dinero]
CANCELADO --entregar--> transición no permitida (sigue en CANCELADO)
Estado final: CANCELADO

12. Ingeniería inversa: de las clases al diagrama

Muy difícil · Diagramas UML · apuntesdam.com/subject/entornos/topic/diagramas-uml

Herramientas como IntelliJ o PlantUML pueden generar el diagrama de clases a partir del código. Hazlo tú con reflexión: para cada clase de la lista, en orden, escribe su bloque de PlantUML («interface X {», «abstract class X {» o «class X {»). Dentro, los atributos ordenados por nombre con su visibilidad (+ public, - private, # protected, ~ paquete) como «-dni : String», y después los métodos declarados en la clase, también por nombre, como «+matricular(Asignatura) : void». Un atributo cuyo tipo es otra clase del modelo no se escribe como atributo sino como asociación «Asignatura --> "1" Profesor : titular»; uno de tipo List<X> es una asociación con multiplicidad * («Alumno --> "*" Matricula : matriculas»). Al final,

después de todos los bloques y en el orden en que se descubrieron mientras se recorrían las clases (primero las asociaciones de cada clase y luego su herencia), las relaciones, incluidas la herencia «Persona <|-- Alumno» y la realización de interfaces «Evaluable <|.. Alumno».

Código de partida (java)

```
import java.lang.reflect.*;
import java.util.*;

// ---- Modelo de ejemplo (no lo modifiques)
abstract class Persona {
    protected String nombre;
    private String dni;
    public String getNombre() { return nombre; }
}

interface Evaluable { double nota(); }
class Alumno extends Persona implements Evaluable {
    private int curso;
    private List<Matricula> matriculas;
    public double nota() { return 0; }
    public void matricular(Asignatura a) { }
}

class Profesor extends Persona {
    private double salario;
    private List<Asignatura> imparte;
}

class Asignatura {
    private String codigo;
    int horas;
    private Profesor titular;
}

class Matricula {
    private Asignatura asignatura;
    private double nota;
}

public class Main {
    static final List<Class<?>> MODELO = List.of(Persona.class, Evaluable.class, Alumno.class,
        Profesor.class, Asignatura.class, Matricula.class);

    static String visibilidad(int mod) {
        return Modifier.isPublic(mod) ? "+" : Modifier.isPrivate(mod) ? "-" : Modifier.isProtected(mod) ?
            "#" : "~";
    }

    // Genera el diagrama de clases en PlantUML a partir de las clases compiladas
    static String plantUml(List<Class<?>> clases) {
        StringBuilder sb = new StringBuilder("@startuml\n");
        // TODO
        return sb.append("@enduml").toString();
    }

    public static void main(String[] args) {
        System.out.println(plantUml(MODELO));
    }
}
```

Ejemplo: Modelo de un instituto

SALIDA ESPERADA

```
@startuml
abstract class Persona {
    -dni : String
    #nombre : String
    +getNombre() : String
}
interface Evaluable {
    +nota() : double
}
class Alumno {
    -curso : int
    +matricular(Asignatura) : void
    +nota() : double
}
class Profesor {
    -salario : double
}
class Asignatura {
    -codigo : String
    ~horas : int
}
class Matricula {
    -nota : double
}
Alumno --> "*" Matricula : matriculas
Persona <|-- Alumno
Evaluable <|.. Alumno
Profesor --> "*" Asignatura : imparte
Persona <|-- Profesor
Asignatura --> "1" Profesor : titular
Matricula --> "1" Asignatura : asignatura
@enduml
```

13. Depura el programa de notas

Fácil · Ejercicios de Entornos de Desarrollo · apuntesdam.com/subject/entornos/topic/ejercicios-entornos

Este programa lee N notas y debe mostrar su media con dos decimales y la nota máxima, pero tiene tres errores. Usa el depurador (o trazas) para encontrarlos y corrígelos sin reescribir el programa. Entrada: N y después las N notas.

Código de partida (java)

```
import java.util.*;

public class Main {

    // Devuelve la media de las notas (con decimales)
    static double media(int[] notas) {
        int suma = 0;
        for (int i = 1; i <= notas.length - 1; i++) {
            suma += notas[i];
        }
        return suma / notas.length;
    }

    // Devuelve la nota más alta
    static int maxima(int[] notas) {
        int max = notas[0];
        for (int n : notas) {
            if (n < max) max = n;
        }
        return max;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] notas = new int[n];
        for (int i = 0; i < n; i++) notas[i] = sc.nextInt();
        System.out.printf(Locale.US, "Media: %.2f\n", media(notas));
        System.out.println("Máxima: " + maxima(notas));
    }
}
```

Ejemplo: Cuatro notas

ENTRADA

```
4
5 7 8 10
```

SALIDA ESPERADA

```
Media: 7.50
Máxima: 10
```

Ejemplo: Media con decimales

ENTRADA

```
3
6 6 7
```

SALIDA ESPERADA

```
Media: 6.33
Máxima: 7
```

14. Valores límite: clasificar edades

Fácil · Ejercicios de Entornos de Desarrollo · apuntesdam.com/subject/entornos/topic/ejercicios-entornos

Implementa clasificar(edad) según esta especificación: edad < 0 o > 120 → "Error"; 0–11 → "Niño"; 12–17 → "Adolescente"; 18–64 → "Adulto"; 65–120 → "Jubilado". Los tests usan justo los valores límite de cada clase de equivalencia, que es donde suelen esconderse los errores. Entrada: una edad por línea.

Código de partida (java)

```
import java.util.*;

public class Main {

    static String clasificar(int edad) {
        // TODO: implementa la especificación
        return "";
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextInt()) {
            System.out.println(clasificar(sc.nextInt()));
        }
    }
}
```

Ejemplo: Límites inferiores

ENTRADA

-1
0
11
12

SALIDA ESPERADA

Error
Niño
Niño
Adolescente

Ejemplo: Límites centrales

ENTRADA

17
18
64
65

SALIDA ESPERADA

Adolescente
Adulto
Adulto
Jubilado

15. Caja negra: el problema del triángulo

Medio · Ejercicios de Entornos de Desarrollo · apuntesdam.com/subject/entornos/topic/ejercicios-entornos

Clásico de las pruebas de caja negra. Implementa `tipo(a, b, c)`, que recibe las longitudes de tres lados enteros y devuelve "Equilátero", "Isósceles", "Escaleno" o "No es triángulo" (si algún lado es ≤ 0 o si un lado es mayor o igual que la suma de los otros dos). Entrada: tres enteros por línea.

Código de partida (java)

```
import java.util.*;

public class Main {

    static String tipo(int a, int b, int c) {
        // TODO: clasifica el triángulo
        return "";
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextInt()) {
            System.out.println(tipo(sc.nextInt(), sc.nextInt(), sc.nextInt()));
        }
    }
}
```

Ejemplo: Tipos válidos

ENTRADA

```
3 3 3
3 3 5
3 4 5
```

SALIDA ESPERADA

```
Equilátero
Isósceles
Escaleno
```

Ejemplo: Casos inválidos

ENTRADA

```
0 4 5
1 2 3
-3 4 5
```

SALIDA ESPERADA

```
No es triángulo
No es triángulo
No es triángulo
```

16. Refactoriza: extraer método

Medio · Ejercicios de Entornos de Desarrollo · apuntesdam.com/subject/entornos/topic/ejercicios-entornos

El cálculo original del precio estaba en un único bloque lleno de números mágicos:

$p = \text{base} * \text{unidades}$; si VIP $p -= 15\%$; si SOCIO $p -= 5\%$; si unidades ≥ 10 $p -= 10\%$; $p += 21\%$ de IVA.

Ya se ha extraído la estructura en métodos pequeños. Completa `descuentoCliente()`, `descuentoVolumen()` y el IVA de `precioFinal()` para que el comportamiento sea idéntico al original. Entrada por línea: precio base, unidades y tipo de cliente (NORMAL, SOCIO o VIP).

Código de partida (java)

```
import java.util.*;

public class Main {

    static final double IVA = 0.21;

    static double descuentoCliente(String tipo) {
        // TODO: 0.15 para VIP, 0.05 para SOCIO y 0 para el resto
        return 0;
    }

    static double descuentoVolumen(int unidades) {
        // TODO: 0.10 a partir de 10 unidades
        return 0;
    }

    static double aplicarDescuento(double importe, double descuento) {
        return importe * (1 - descuento);
    }

    static double precioFinal(double base, int unidades, String tipo) {
        double importe = base * unidades;
        importe = aplicarDescuento(importe, descuentoCliente(tipo));
        importe = aplicarDescuento(importe, descuentoVolumen(unidades));
        // TODO: aplica el IVA usando la constante
        return importe;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in).useLocale(Locale.US);
        while (sc.hasNext()) {
            double base = sc.nextDouble();
            int unidades = sc.nextInt();
            String tipo = sc.next();
            System.out.printf(Locale.US, "%.2f%n", precioFinal(base, unidades, tipo));
        }
    }
}
```

Ejemplo: Cliente normal

ENTRADA

10 2 NORMAL

SALIDA ESPERADA

24.20

Ejemplo: Socio con descuento por volumen

ENTRADA

20 12 SOCIO

SALIDA ESPERADA

248.29

17. Del diagrama de clases al código

Medio · Ejercicios de Entornos de Desarrollo · apuntesdam.com/subject/entornos/topic/ejercicios-entornos

Implementa el diagrama de clases: la clase abstracta Figura (nombre, area() abstracto y toString() ya hechos) y sus subclases Circulo (radio) y Rectangulo (base, altura), que heredan de ella. El programa lee figuras (C radio o R base altura), las muestra y suma sus áreas.

Figura ◁— Circulo (- radio: double)

Figura ◁— Rectangulo (- base: double, - altura: double)

Código de partida (java)

```
import java.util.*;

abstract class Figura {
    protected String nombre;

    public Figura(String nombre) { this.nombre = nombre; }

    public abstract double area();

    @Override
    public String toString() {
        return String.format(Locale.US, "%s: %.2f", nombre, area());
    }
}

class Circulo extends Figura {
    private double radio;

    public Circulo(double radio) {
        super("Círculo");
        // TODO: guarda el radio
    }

    @Override
    public double area() {
        return 0; // TODO:  $\pi \cdot r^2$ 
    }
}

class Rectangulo extends Figura {
    // TODO: atributos base y altura

    public Rectangulo(double base, double altura) {
        super("Rectángulo");
        // TODO: guarda base y altura
    }

    @Override
    public double area() {
        return 0; // TODO: base · altura
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in).useLocale(Locale.US);
        List<Figura> figuras = new ArrayList<>();
        while (sc.hasNext()) {
            String tipo = sc.next();
            if (tipo.equals("C")) figuras.add(new Circulo(sc.nextDouble()));
            else figuras.add(new Rectangulo(sc.nextDouble(), sc.nextDouble()));
        }
        double total = 0;
        for (Figura f : figuras) {
            System.out.println(f);
            total += f.area();
        }
        System.out.printf(Locale.US, "Total: %.2f\n", total);
    }
}
```

Ejemplo: Un círculo y un rectángulo

ENTRADA

```
C 1
R 3 4
```

SALIDA ESPERADA

```
Círculo: 3.14
Rectángulo: 12.00
Total: 15.14
```

Programación

18. Del pseudocódigo a Java

Fácil · Fundamentos de la Programación · apuntesdam.com/subject/programacion/topic/introduccion-programacion

Traduce a Java este pseudocódigo:

Leer n

Si n = 0 entonces

 Escribir «cero»

Si no

 Si n > 0 entonces signo ← «positivo» si no signo ← «negativo»

 Si n mod 2 = 0 entonces paridad ← «par» si no paridad ← «impar»

 Escribir signo, « », paridad

Fin si

Por ejemplo, para 7 muestra «positivo impar» y para -4, «negativo par».

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        // TODO: traduce el pseudocódigo del enunciado
    }
}
```

Ejemplo: Positivo impar

ENTRADA

7

SALIDA ESPERADA

positivo impar

Ejemplo: Negativo par

ENTRADA

-4

SALIDA ESPERADA

negativo par

Ejemplo: Cero

ENTRADA

0

SALIDA ESPERADA

cero

19. Traza del algoritmo de Euclides

Medio · Fundamentos de la Programación · apuntesdam.com/subject/programacion/topic/introduccion-programacion

El algoritmo de Euclides, de hace más de 2000 años, calcula el máximo común divisor de dos números: mientras b no sea 0, se divide a entre b, a pasa a valer b y b pasa a valer el resto. Lee a y b (enteros positivos) y muestra la traza: en cada paso, «Paso k: a = cociente · b + resto»; al final, «mcd = m en n pasos» («1 paso» en singular). Hacer trazas así es la forma de comprobar a mano que un algoritmo es correcto.

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int a = sc.nextInt(), b = sc.nextInt();
        // TODO: algoritmo de Euclides mostrando la traza
    }
}
```

Ejemplo: mcd(1071, 462)

ENTRADA

1071 462

SALIDA ESPERADA

Paso 1: $1071 = 2 \cdot 462 + 147$
Paso 2: $462 = 3 \cdot 147 + 21$
Paso 3: $147 = 7 \cdot 21 + 0$
mcd = 21 en 3 pasos

Ejemplo: Divisible

ENTRADA

48 12

SALIDA ESPERADA

Paso 1: $48 = 4 \cdot 12 + 0$
mcd = 12 en 1 paso

20. Conversor de bases sin atajos

Difícil · Fundamentos de la Programación · apuntesdam.com/subject/programacion/topic/introduccion-programacion

Los ordenadores trabajan en binario y los programadores leemos a menudo hexadecimal (colores #FF8800, direcciones de memoria). Implementa `aBase(n, base)`, que convierte un entero no negativo a cualquier base de 2 a 16 sin usar `Integer.toBinaryString` ni similares: se divide sucesivamente entre la base y los restos, leídos de abajo arriba, son las cifras ($10 \rightarrow A$, $11 \rightarrow B \dots 15 \rightarrow F$). El programa lee varios números y muestra cada uno en binario, octal y hexadecimal.

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    static String aBase(int n, int base) {
        // TODO: divisiones sucesivas entre la base
        return "";
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextInt()) {
            int n = sc.nextInt();
            System.out.println(n + " → binario " + aBase(n, 2) + " · octal " + aBase(n, 8) + " · hexadecimal " + aBase(n, 16));
        }
    }
}
```

Ejemplo: Varios números

ENTRADA

10
255
0
4096

SALIDA ESPERADA

10 → binario 1010 · octal 12 · hexadecimal A
255 → binario 11111111 · octal 377 · hexadecimal FF
0 → binario 0 · octal 0 · hexadecimal 0
4096 → binario 1000000000000 · octal 10000 · hexadecimal 1000

21. Simula una máquina con su propio ensamblador

Muy difícil · Fundamentos de la Programación · apuntesdam.com/subject/programacion/topic/introduccion-programacion

Un procesador repite sin parar el ciclo de captación y ejecución: lee la instrucción a la que apunta el contador de programa (PC), la ejecuta y pasa a la siguiente (o salta). Simula una máquina con un acumulador, 16 posiciones de memoria (a 0) y estas instrucciones, una por línea (las líneas vacías y los comentarios con ; se ignoran; el número de línea cuenta desde 0 sin ellas): **LOADI v** (acumulador = v), **LOAD d** (acumulador = memoria[d]), **STORE d** (memoria[d] = acumulador), **ADD d** y **SUB d** (suma o resta memoria[d] al acumulador), **JMP l** (salta a la instrucción l), **JZ l** (salta si el acumulador es 0), **JNEG l** (salta si es negativo), **PRINT** (muestra el acumulador) y **HALT** (muestra «HALT tras n instrucciones · acumulador = v» y termina). Errores: si el PC se sale del programa, «ERROR: el programa termina sin HALT»; si una instrucción no existe, «ERROR: instrucción desconocida X en la línea l»; y si se han ejecutado ya 1000 instrucciones sin terminar, «ERROR: demasiadas instrucciones (¿bucle infinito?)».

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<String[]> programa = new ArrayList<>();
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (!l.isEmpty() && !l.startsWith(";")) programa.add(l.split("\\s+"));
        }
        int[] memoria = new int[16];
        int acumulador = 0, pc = 0, ejecutadas = 0;
        // TODO: ciclo de captación y ejecución de instrucciones
    }
}
```

Ejemplo: Suma de 1 a 5

ENTRADA

```
; Suma de 1 a 5 con un bucle
LOADI 5
STORE 0      ; m[0] = contador
LOADI 0
STORE 1      ; m[1] = suma
LOADI 1
STORE 2      ; m[2] = 1 (constante)
LOAD 0       ; 6: bucle
JZ 15
LOAD 1
ADD 0
STORE 1
LOAD 0
SUB 2
STORE 0
JMP 6
LOAD 1       ; 15: fin
PRINT
HALT
```

SALIDA ESPERADA

```
15
HALT tras 56 instrucciones · acumulador = 15
```

Ejemplo: Cuenta atrás

ENTRADA

```
LOADI 3
STORE 0
LOADI 1
STORE 1
LOAD 0
PRINT
SUB 1
STORE 0
JZ 10
JMP 4
HALT
```

SALIDA ESPERADA

```
3
2
1
HALT tras 22 instrucciones · acumulador = 0
```

22. Desglose de segundos

Fácil · Introducción a Java · apuntesdam.com/subject/programacion/topic/introduccion-java

Lee un número de segundos y desglósalo en días, horas, minutos y segundos usando la división entera (/) y el resto (%). Muestra «N s = d d h h m min s s» y después «Formato reloj: hh:mm:ss (+d días)» («+1 día» en singular) con dos cifras en horas, minutos y segundos.

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        long total = sc.nextLong();
        // TODO: días, horas, minutos y segundos
    }
}
```

Ejemplo: Un día y pico

ENTRADA

90061

SALIDA ESPERADA

```
90061 s = 1 d 1 h 1 min 1 s
Formato reloj: 01:01:01 (+1 día)
```

Ejemplo: Menos de un minuto

ENTRADA

45

SALIDA ESPERADA

```
45 s = 0 d 0 h 0 min 45 s
Formato reloj: 00:00:45 (+0 días)
```

23. Las trampas de los tipos numéricos

Medio · Introducción a Java · apuntesdam.com/subject/programacion/topic/introduccion-java

Los tipos de Java tienen límites y reglas que causan errores muy habituales. Lee dos enteros a y b (b distinto de 0) y muestra, en este orden: «int: a * b = r» calculado con int (puede desbordarse); «long: a * b = r» convirtiendo antes a long; «división entera: a / b = r»; «división real: a / b = r» convirtiendo a double antes de dividir; «casting: (int) real = r», truncando esa división real; y «char: 'A' + k = L (código c)», donde k = |a| % 26 y L es la letra que está k posiciones después de la A.

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int a = sc.nextInt(), b = sc.nextInt();
        // TODO: muestra las seis líneas del enunciado
    }
}
```

Ejemplo: Sin desbordamiento

ENTRADA

7 2

SALIDA ESPERADA

```
int: 7 * 2 = 14
long: 7 * 2 = 14
división entera: 7 / 2 = 3
división real: 7 / 2 = 3.5
casting: (int) 3.5 = 3
char: 'A' + 7 = H (código 72)
```

Ejemplo: Con desbordamiento

ENTRADA

100000 30000

SALIDA ESPERADA

```
int: 100000 * 30000 = -1294967296
long: 100000 * 30000 = 3000000000
división entera: 100000 / 30000 = 3
división real: 100000 / 30000 = 3.3333333333333335
casting: (int) 3.3333333333333335 = 3
char: 'A' + 4 = E (código 69)
```

24. La cuota de un préstamo

Difícil · Introducción a Java · apuntesdam.com/subject/programacion/topic/introduccion-java

Lee el capital de un préstamo, el interés anual en % y los años. La cuota mensual (sistema francés) es $C \cdot i / (1 - (1 + i)^{-n})$, con i el interés mensual (anual / 100 / 12) y n el número de meses; si el interés es 0, la cuota es C / n . Muestra «Cuota mensual: X EUR (n cuotas)», «Total pagado: T EUR · intereses: I EUR» y el desglose de las tres primeras cuotas (o menos si hay menos): en cada una, los intereses son el capital pendiente por i , lo amortizado es la cuota menos los intereses y el pendiente baja en lo amortizado: «Cuota k: intereses a · amortiza b · pendiente c». Todos los importes con dos decimales y punto decimal.

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in).useLocale(Locale.US);
        double capital = sc.nextDouble(); // euros
        double interesAnual = sc.nextDouble(); // en %
        int años = sc.nextInt();
        // TODO: cuota mensual, intereses totales y las tres primeras cuotas
    }
}
```

Ejemplo: Hipoteca

ENTRADA

150000 3.5 25

SALIDA ESPERADA

Cuota mensual: 750.94 EUR (300 cuotas)
Total pagado: 225280.61 EUR · intereses: 75280.61 EUR
Cuota 1: intereses 437.50 · amortiza 313.44 · pendiente 149686.56
Cuota 2: intereses 436.59 · amortiza 314.35 · pendiente 149372.22
Cuota 3: intereses 435.67 · amortiza 315.27 · pendiente 149056.95

Ejemplo: Coche sin intereses

ENTRADA

12000 0 1

SALIDA ESPERADA

Cuota mensual: 1000.00 EUR (12 cuotas)
Total pagado: 12000.00 EUR · intereses: 0.00 EUR
Cuota 1: intereses 0.00 · amortiza 1000.00 · pendiente 11000.00
Cuota 2: intereses 0.00 · amortiza 1000.00 · pendiente 10000.00
Cuota 3: intereses 0.00 · amortiza 1000.00 · pendiente 9000.00

25. Permisos de Unix con operadores de bits

Muy difícil · Introducción a Java · apuntesdam.com/subject/programacion/topic/introduccion-java

Los permisos de un fichero en Linux son 9 bits: lectura, escritura y ejecución ($r = 4$, $w = 2$, $x = 1$) para el usuario (bits 8-6), el grupo (5-3) y los demás (2-0); por eso se escriben en octal, como 754. La entrada empieza con los permisos en octal y sigue con órdenes «+bits quién», «-bits quién» o «=bits quién» (bits: combinación de r , w , x ; quién: u , g , o o a para todos), como en `chmod`. Implementa `simbolico(permisos)`, que devuelve la forma `rwrx-xr--`, y aplica cada orden con operadores de bits: $+$ enciende los bits ($|$), $-$ los apaga ($\&$ con $\sim$) y $=$ deja en ese grupo exactamente esos bits. Tras cada orden se muestra «orden quién $\rightarrow$ ooo `rwrx...`» (en octal con tres cifras).

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    static String simbolico(int permisos) {
        // TODO: rwx por cada bit encendido, - por cada apagado
        return "";
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int permisos = Integer.parseInt(sc.next(), 8); // p. ej. 754 en octal
        System.out.println("Inicial: " + String.format("%03o", permisos) + " " + simbolico(permisos));
        while (sc.hasNext()) {
            String orden = sc.next(); // +x, -w, =r...
            String quien = sc.next(); // u, g, o o a
            // TODO: aplicar la orden con operadores de bits
            System.out.println(orden + " " + quien + "  $\rightarrow$  " + String.format("%03o", permisos) + " " +
                simbolico(permisos));
        }
    }
}
```

Ejemplo: Ficheros de una web

ENTRADA

```
644
+x u
-r o
=rw g
+w a
```

SALIDA ESPERADA

```
Inicial: 644 rw-r--r--
+x u → 744 rwxr--r--
-r o → 740 rwxr-----
=rw g → 760 rwxrw----
+w a → 762 rwxrw--w-
```

Ejemplo: Script

ENTRADA

```
700
+rx a
-w u
=x o
```

SALIDA ESPERADA

```
Inicial: 700 rwx-----
+rx a → 755 rwxr-xr-x
-w u → 555 r-xr-xr-x
=x o → 551 r-xr-x--x
```

26. Calificación a partir de la nota

Fácil · Estructuras Condicionales · apuntesdam.com/subject/programacion/topic/condicionales

Lee una nota entera por teclado y muestra su calificación: Suspenso (0-4), Aprobado (5), Bien (6), Notable (7-8) o Sobresaliente (9-10). Si la nota no está entre 0 y 10, muestra 'Nota no valida'.

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int nota = sc.nextInt();

        // TODO: muestra la calificación correspondiente
        System.out.println("?");
    }
}
```

Ejemplo: Nota 4 → Suspenso

ENTRADA

4

SALIDA ESPERADA

Suspenso

Ejemplo: Nota 6 → Bien

ENTRADA

6

SALIDA ESPERADA

Bien

Ejemplo: Nota 9 → Sobresaliente

ENTRADA

9

SALIDA ESPERADA

Sobresaliente

27. Encuentra el fallo: años bisiestos

Fácil · Estructuras Condicionales · apuntesdam.com/subject/programacion/topic/condicionales

El programa lee varios años y debe decir si cada uno es bisiesto: lo es si es divisible entre 4, salvo los divisibles entre 100 que no lo son entre 400 (1900 no es bisiesto; 2000 sí). Hay años que salen mal. Corrige la condición.

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextInt()) {
            int a = sc.nextInt();
            boolean bisiesto = a % 4 == 0 || a % 100 != 0 || a % 400 == 0;
            System.out.println(a + (bisiesto ? ": bisiesto" : ": no bisiesto"));
        }
    }
}
```

Ejemplo: 2024, 2023, 1900 y 2000

ENTRADA

2024
2023
1900
2000

SALIDA ESPERADA

2024: bisiesto
2023: no bisiesto
1900: no bisiesto
2000: bisiesto

28. Encuentra el fallo: la nota en palabras

Fácil · Estructuras Condicionales · apuntesdam.com/subject/programacion/topic/condicionales

El programa convierte una nota entera (0 a 10) en su calificación: 0-4 Insuficiente, 5 Suficiente, 6 Bien, 7-8 Notable, 9-10 Sobresaliente. Con un 6 escribe varias palabras en lugar de una. Corrígelo.

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        int nota = new Scanner(System.in).nextInt();
        switch (nota) {
            case 5:
                System.out.println("Suficiente");
                break;
            case 6:
                System.out.println("Bien");
            case 7:
            case 8:
                System.out.println("Notable");
                break;
            case 9:
            case 10:
                System.out.println("Sobresaliente");
                break;
            default:
                System.out.println("Insuficiente");
        }
    }
}
```

Ejemplo: Un 6

ENTRADA

6

SALIDA ESPERADA

Bien

Ejemplo: Un 8

ENTRADA

8

SALIDA ESPERADA

Notable

29. Año bisiesto

Medio · Estructuras Condicionales · apuntesdam.com/subject/programacion/topic/condicionales

Lee un año y di si es bisiesto. Un año es bisiesto si es divisible entre 4, excepto los divisibles entre 100, salvo que también lo sean entre 400. Formato: '2024 es bisiesto' o '2023 no es bisiesto'.

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int anio = sc.nextInt();

        boolean bisiesto = false; // TODO: calcula si es bisiesto

        if (bisiesto) {
            System.out.println(anio + " es bisiesto");
        } else {
            System.out.println(anio + " no es bisiesto");
        }
    }
}
```

Ejemplo: 2024 es bisiesto

ENTRADA

2024

SALIDA ESPERADA

2024 es bisiesto

Ejemplo: 1900 no es bisiesto (divisible entre 100)

ENTRADA

1900

SALIDA ESPERADA

1900 no es bisiesto

30. Validar una fecha y calcular el día siguiente

Difícil · Estructuras Condicionales · apuntesdam.com/subject/programacion/topic/condicionales

Lee fechas «día mes año» (una por línea) y, para cada una, muestra «dd/mm/aaaa: no válida (el mes debe estar entre 1 y 12)», «dd/mm/aaaa: no válida (ese mes tiene N días)» o «dd/mm/aaaa → siguiente: dd/mm/aaaa», añadiendo « (año bisiesto)» si el año de la fecha lo es. Un año es bisiesto si es divisible entre 4 y no entre 100, o si es divisible entre 400 (2024 y 2000 lo son; 1900 no). Abril, junio, septiembre y noviembre tienen 30 días y febrero 28 o 29.

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextInt()) {
            int d = sc.nextInt(), m = sc.nextInt(), a = sc.nextInt();
            // TODO: validar la fecha y calcular el día siguiente
        }
    }
}
```

Ejemplo: Fechas variadas

ENTRADA

```
15 3 2025
28 2 2024
29 2 2023
31 12 1999
31 4 2025
10 13 2025
```

SALIDA ESPERADA

```
15/03/2025 → siguiente: 16/03/2025
28/02/2024 → siguiente: 29/02/2024 (año bisiesto)
29/02/2023: no válida (ese mes tiene 28 días)
31/12/1999 → siguiente: 01/01/2000
31/04/2025: no válida (ese mes tiene 30 días)
10/13/2025: no válida (el mes debe estar entre 1 y 12)
```

31. IRPF por tramos (simplificado)

Muy difícil · Estructuras Condicionales · apuntesdam.com/subject/programacion/topic/condicionales

El IRPF es progresivo: cada tramo de la base se grava a un tipo distinto. Implementa escala(base) con la escala estatal y autonómica sumadas (simplificada): hasta 12 450 € al 19 %; de 12 450 a 20 200 al 24 %; hasta 35 200 al 30 %; hasta 60 000 al 37 %; hasta 300 000 al 45 %; y más al 47 %. Cada tramo se suma a lo acumulado en los anteriores (por ejemplo, $\text{escala}(20\,200) = 2\,365,50 + 0,24 \cdot 7\,750$). Lee el salario bruto anual, el número de hijos y la edad. Seguridad Social = 6,35 % del bruto; base = bruto – Seguridad Social – 2 000 (gastos), sin bajar de 0. Mínimo personal y familiar = 5 550, +1 150 si tiene más de 65 años, +2 400 por el primer hijo, +2 700 por el segundo, +4 000 por el tercero y +4 500 por cada uno a partir del cuarto. Cuota = $\text{escala}(\text{base}) - \text{escala}(\text{mínimo})$, o 0 si el mínimo no es menor que la base. Muestra «Base liquidable», «Mínimo personal y familiar», «Cuota de IRPF», «Tipo efectivo: x %» (cuota sobre el bruto) y «Neto anual: ... · al mes (14 pagas): ...», todo con dos decimales y punto decimal.

Código de partida (java)

```
import java.util.*;

public class Main {
    static double escala(double base) {
        // TODO: impuesto por tramos de una base
        return 0;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in).useLocale(Locale.US);
        double bruto = sc.nextDouble();
        int hijos = sc.nextInt();
        int edad = sc.nextInt();
        // TODO: base, mínimo personal y familiar, cuota, tipo efectivo y neto
    }
}
```

Ejemplo: Sueldo medio sin hijos

ENTRADA

```
28000 0 30
```

SALIDA ESPERADA

```
Base liquidable: 24222.00
Mínimo personal y familiar: 5550.00
Cuota de IRPF: 4377.60
Tipo efectivo: 15.63 %
Neto anual: 21844.40 · al mes (14 pagas): 1560.31
```

Ejemplo: Dos hijos

ENTRADA

```
45000 2 42
```

SALIDA ESPERADA

```
Base liquidable: 40142.50
Mínimo personal y familiar: 10650.00
Cuota de IRPF: 8530.73
Tipo efectivo: 18.96 %
Neto anual: 33611.78 · al mes (14 pagas): 2400.84
```

Ejemplo: Sueldo bajo

ENTRADA

14000 1 25

SALIDA ESPERADA

Base liquidable: 11111.00
Mínimo personal y familiar: 7950.00
Cuota de IRPF: 600.59
Tipo efectivo: 4.29 %
Neto anual: 12510.41 · al mes (14 pagas): 893.60

32. Tabla de multiplicar

Fácil · Bucles · apuntesdam.com/subject/programacion/topic/bucles

Lee un número n y muestra su tabla de multiplicar del 1 al 10 con el formato 'n x i = resultado', una línea por operación.

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        // TODO: muestra la tabla de multiplicar de n
    }
}
```

Ejemplo: Tabla del 7

ENTRADA

7

SALIDA ESPERADA

7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70

33. Encuentra el fallo: la suma de 1 a n

Fácil · Bucles · apuntesdam.com/subject/programacion/topic/bucles

El programa lee n y debe mostrar la suma 1 + 2 + ... + n. Para n = 4 debería salir 10, pero sale 6. Corrige el bucle.

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        int n = new Scanner(System.in).nextInt();
        int suma = 0;
        for (int i = 1; i < n; i++) {
            suma += i;
        }
        System.out.println("Suma: " + suma);
    }
}
```

Ejemplo: n = 4

ENTRADA

4

SALIDA ESPERADA

Suma: 10

34. Suma de los dígitos

Medio · Bucles · apuntesdam.com/subject/programacion/topic/bucles

Lee un número entero positivo y calcula la suma de sus dígitos usando un bucle while. Formato: 'Suma: X'.

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int suma = 0;

        // TODO: suma los dígitos de n con un bucle while

        System.out.println("Suma: " + suma);
    }
}
```

Ejemplo: 1234 → 10

ENTRADA

1234

SALIDA ESPERADA

Suma: 10

Ejemplo: 9 → 9

ENTRADA

9

SALIDA ESPERADA

Suma: 9

35. Encuentra el fallo: la media que pierde decimales

Medio · Bucles · apuntesdam.com/subject/programacion/topic/bucles

El programa lee números enteros hasta un 0 y muestra su media con dos decimales. Con 7, 8 y 8 debería salir 7,67 (en formato 7.67) pero sale 7.00. Corrígelo sin cambiar el tipo de los números leídos.

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int suma = 0, contador = 0;
        int n = sc.nextInt();
        while (n != 0) {
            suma += n;
            contador++;
            n = sc.nextInt();
        }
        double media = suma / contador;
        System.out.printf(java.util.Locale.US, "Media: %.2f%n", media);
    }
}
```

Ejemplo: 7, 8 y 8

ENTRADA

7 8 8 0

SALIDA ESPERADA

Media: 7.67

36. La conjetura de Collatz

Difícil · Bucles · apuntesdam.com/subject/programacion/topic/bucles

Toma un número: si es par, divídelo entre 2; si es impar, multiplícalo por 3 y súmale 1. Repite. La conjetura de Collatz dice que siempre se acaba llegando a 1, y nadie ha conseguido demostrarlo. Lee a y b ($1 \leq a \leq b$). Muestra la secuencia de a separada por « → » y después «Entre a y b: m necesita p pasos; el valor más alto alcanzado es t», donde m es el número de ese intervalo que más pasos necesita para llegar a 1 (el menor si hay empate) y t el mayor valor que aparece en cualquiera de esas secuencias. Usa long: los valores intermedios crecen mucho.

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int a = sc.nextInt(), b = sc.nextInt();
        // TODO: secuencia de a y número con más pasos entre a y b
    }
}
```

Ejemplo: Del 6 al 10

ENTRADA

6 10

SALIDA ESPERADA

6 → 3 → 10 → 5 → 16 → 8 → 4 → 2 → 1
Entre 6 y 10: 9 necesita 19 pasos; el valor más alto alcanzado es 52

Ejemplo: Del 27 al 30

ENTRADA

27 30

SALIDA ESPERADA

27 → 82 → 41 → 124 → 62 → 31 → 94 → 47 → 142 → 71 → 214 → 107 → 322 → 161 → 484 → 242 → 121 → 364 → 182 → 91 → 274 → 137 → 412 → 206 → 103 → 310 → 155 → 466 → 233 → 700 → 350 → 175 → 526 → 263 → 790 → 395 → 1186 → 593 → 1780 → 890 → 445 → 1336 → 668 → 334 → 167 → 502 → 251 → 754 → 377 → 1132 → 566 → 283 → 850 → 425 → 1276 → 638 → 319 → 958 → 479 → 1438 → 719 → 2158 → 1079 → 3238 → 1619 → 4858 → 2429 → 7288 → 3644 → 1822 → 911 → 2734 → 1367 → 4102 → 2051 → 6154 → 3077 → 9232 → 4616 → 2308 → 1154 → 577 → 1732 → 866 → 433 → 1300 → 650 → 325 → 976 → 488 → 244 → 122 → 61 → 184 → 92 → 46 → 23 → 70 → 35 → 106 → 53 → 160 → 80 → 40 → 20 → 10 → 5 → 16 → 8 → 4 → 2 → 1
Entre 27 y 30: 27 necesita 111 pasos; el valor más alto alcanzado es 9232

37. El calendario de un mes

Muy difícil · Bucles · apuntesdam.com/subject/programacion/topic/bucles

Lee un año y un mes y muestra su calendario como el de la orden cal de Linux, empezando la semana en lunes: el nombre del mes y el año; la cabecera «Lu Ma Mi Ju Vi Sá Do»; y las semanas, cada día con dos caracteres alineado a la derecha y separado por un espacio, dejando huecos de tres espacios antes del día 1 (sin espacios al final de las líneas). Termina con «N días · F de fin de semana · empieza en día». Para saber en qué día de la semana cae el día 1 usa la congruencia de Zeller: con m = mes e y = año, pero contando enero y febrero como los meses 13 y 14 del año anterior, $k = y \% 100$ y $j = y / 100$, $h = (1 + 13 \cdot (m+1)/5 + k + k/4 + j/4 + 5 \cdot j) \% 7$ da 0 = sábado, 1 = domingo, 2 = lunes... (divisiones enteras).

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int anio = sc.nextInt(), mes = sc.nextInt();
        String[] nombres = {"Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio", "Julio", "Agosto",
                            "Septiembre", "Octubre", "Noviembre", "Diciembre"};
        // TODO: días del mes, día de la semana del día 1 y la cuadrícula
    }
}
```

Ejemplo: Octubre de 2025

ENTRADA

2025 10

SALIDA ESPERADA

Octubre 2025
Lu Ma Mi Ju Vi Sá Do
1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
31 días · 8 de fin de semana · empieza en
miércoles

Ejemplo: Febrero bisiesto

ENTRADA

2024 2

SALIDA ESPERADA

Febrero 2024
Lu Ma Mi Ju Vi Sá Do
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29
29 días · 8 de fin de semana · empieza en jueves

Ejemplo: Septiembre de 2025

ENTRADA

2025 9

SALIDA ESPERADA

Septiembre 2025
Lu Ma Mi Ju Vi Sá Do
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30
30 días · 8 de fin de semana · empieza en lunes

38. Números primos con una función

Medio · Métodos · apuntesdam.com/subject/programacion/topic/metodos

Completa la función `esPrimo(int n)` y úsala para mostrar, separados por espacios, todos los números primos desde 2 hasta el número leído.

Código de partida (java)

```
import java.util.Scanner;

public class Main {

    static boolean esPrimo(int n) {
        // TODO: devuelve true si n es primo
        return false;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int limite = sc.nextInt();
        StringBuilder sb = new StringBuilder();
        for (int i = 2; i <= limite; i++) {
            if (esPrimo(i)) sb.append(i).append(" ");
        }
        System.out.println(sb.toString().trim());
    }
}
```

Ejemplo: Primos hasta 10

ENTRADA

10

SALIDA ESPERADA

2 3 5 7

Ejemplo: Primos hasta 20

ENTRADA

20

SALIDA ESPERADA

2 3 5 7 11 13 17 19

39. Factorial recursivo

Medio · Métodos · apuntesdam.com/subject/programacion/topic/metodos

Implementa la función recursiva `factorial(int n)` que devuelva $n!$ como long. El programa muestra ' $n!$ = resultado'. Recuerda que $0! = 1$.

Código de partida (java)

```
import java.util.Scanner;

public class Main {

    static long factorial(int n) {
        // TODO: implementa el factorial de forma recursiva
        return 0;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        System.out.println(n + "! = " + factorial(n));
    }
}
```

Ejemplo: $5! = 120$

ENTRADA

5

SALIDA ESPERADA

$5! = 120$

Ejemplo: $0! = 1$

ENTRADA

0

SALIDA ESPERADA

$0! = 1$

40. Encuentra el fallo: el método que no cambia nada

Medio · Métodos · apuntesdam.com/subject/programacion/topic/metodos

El método `aplicarDescuento` debería rebajar el precio un porcentaje, pero el programa sigue mostrando el precio original. Corrige el método y su llamada para que el precio cambie.

Código de partida (java)

```
public class Main {
    static void aplicarDescuento(double precio, int porcentaje) {
        precio = precio - precio * porcentaje / 100;
    }

    public static void main(String[] args) {
        double precio = 50;
        aplicarDescuento(precio, 20);
        System.out.println("Precio final: " + precio);
    }
}
```

Ejemplo: 50 € con un 20 %

SALIDA ESPERADA

Precio final: 40.0

41. Las torres de Hanói

Difícil · Métodos · apuntesdam.com/subject/programacion/topic/metodos

Hay tres postes (A, B y C) y n discos de distinto tamaño apilados en A, del mayor abajo al menor arriba. Hay que pasarlos todos a C moviendo un disco cada vez y sin poner nunca uno grande encima de uno pequeño. La solución recursiva es sorprendentemente corta: para mover n discos de origen a destino, se mueven los $n-1$ de encima al poste auxiliar, se mueve el disco n al destino y se vuelven a mover los $n-1$ encima de él. Completa `hanoi` para que cuente los movimientos y, si mostrar es cierto ($n \leq 4$), escriba cada uno como «Disco k: $X \rightarrow Y$ ». El main ya muestra el total y lo compara con $2^n - 1$.

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    static int movimientos = 0;

    static void hanoi(int n, char origen, char destino, char auxiliar, boolean mostrar) {
        // TODO: mover n discos de origen a destino usando auxiliar
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        hanoi(n, 'A', 'C', 'B', n <= 4);
        System.out.println(n + (n == 1 ? " disco: " : " discos: ") + movimientos + (movimientos == 1 ? " movimiento" : " movimientos") + " ( $2^n + n - 1 = " + ((1L << n) - 1) + "$ )");
    }
}
```

Ejemplo: Tres discos

ENTRADA

3

SALIDA ESPERADA

Disco 1: A → C
Disco 2: A → B
Disco 1: C → B
Disco 3: A → C
Disco 1: B → A
Disco 2: B → C
Disco 1: A → C
3 discos: 7 movimientos ($2^3 - 1 = 7$)

Ejemplo: Un disco

ENTRADA

1

SALIDA ESPERADA

Disco 1: A → C
1 disco: 1 movimiento ($2^1 - 1 = 1$)

42. Una calculadora con descenso recursivo

Muy difícil · Métodos · apuntesdam.com/subject/programacion/topic/metodos

Los compiladores leen las expresiones con analizadores recursivos. Implementa uno para expresiones de enteros (long) con +, -, *, / (división entera), %, paréntesis y el signo menos delante de un número o de un paréntesis, respetando la prioridad: primero paréntesis y signo, después * / %, y por último + -, de izquierda a derecha. Se usa una gramática con tres métodos que se llaman entre sí: expresion() lee términos separados por + o -; termino() lee factores separados por *, / o %; factor() lee un número, un - seguido de otro factor o una expresión entre paréntesis (llamando otra vez a expresion: ahí está la recursividad). texto es la línea sin espacios y pos la posición actual. Los errores se lanzan con throw new ErrorExpresion(mensaje): «división entre cero», «falta «)»», «la expresión termina antes de tiempo» y «se esperaba un número en la posición p». El main ya muestra «expresión = resultado» o «expresión → error: mensaje».

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    static String texto;
    static int pos;

    static class ErrorExpresion extends RuntimeException {
        ErrorExpresion(String mensaje) { super(mensaje); }
    }

    static long expresion() {
        // TODO: términos separados por + o -
        return 0;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String linea = sc.nextLine();
            if (linea.isBlank()) continue;
            texto = linea.replace(" ", "");
            pos = 0;
            try {
                long valor = expresion();
                if (pos < texto.length()) throw new ErrorExpresion("sobra «" + texto.charAt(pos) + "» en la posición " + pos);
                System.out.println(linea.trim() + " = " + valor);
            } catch (ErrorExpresion e) {
                System.out.println(linea.trim() + " → error: " + e.getMessage());
            }
        }
    }
}
```

Ejemplo: Expresiones variadas

ENTRADA

```
2 + 3 * 4
(2 + 3) * 4
100 / 7 % 3
-(8 - 10) * -3
2 * (3 + (4 - 1) * 5) - 6
7 / (3 - 3)
(1 + 2
4 + * 5
```

SALIDA ESPERADA

```
2 + 3 * 4 = 14
(2 + 3) * 4 = 20
100 / 7 % 3 = 2
-(8 - 10) * -3 = -6
2 * (3 + (4 - 1) * 5) - 6 = 30
7 / (3 - 3) → error: división entre cero
(1 + 2 → error: falta «)»
4 + * 5 → error: se esperaba un número en la
posición 2
```

43. Máximo y mínimo de un array

Fácil · Arrays · apuntesdam.com/subject/programacion/topic/arrays

Dado un array de enteros, encuentra el valor máximo, el valor mínimo y las posiciones donde se encuentran. Imprime cada resultado en una línea separada con el formato: 'Maximo: X (posicion Y)' y 'Minimo: X (posicion Y)'.

Código de partida (java)

```
public class Main {
    public static void main(String[] args) {
        int[] numeros = {34, 12, 78, 5, 90, 23, 67, 45};

        // TODO: encuentra el máximo y su posición
        int max = numeros[0];
        int posMax = 0;
        // Tu código aquí

        // TODO: encuentra el mínimo y su posición
        int min = numeros[0];
        int posMin = 0;
        // Tu código aquí

        System.out.println("Maximo: " + max + " (posicion " + posMax + ")");
        System.out.println("Minimo: " + min + " (posicion " + posMin + ")");
    }
}
```

Ejemplo: Array {34,12,78,5,90,23,67,45} → máximo 90 en pos 4, mínimo 5 en pos 3

SALIDA ESPERADA

```
Maximo: 90 (posicion 4)
Minimo: 5 (posicion 3)
```

44. Encuentra el fallo: el máximo de temperaturas bajo cero

Fácil · Arrays · apuntesdam.com/subject/programacion/topic/arrays

El programa lee varias temperaturas y muestra la más alta. Funciona en verano, pero en invierno, con todas bajo cero, dice que la máxima es 0. Corrígelo.

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] t = new int[n];
        for (int i = 0; i < n; i++) t[i] = sc.nextInt();
        int max = 0;
        for (int x : t) {
            if (x > max) max = x;
        }
        System.out.println("Máxima: " + max);
    }
}
```

Ejemplo: Verano

ENTRADA

4
31 28 35 30

SALIDA ESPERADA

Máxima: 35

Ejemplo: Invierno

ENTRADA

3
-5 -2 -9

SALIDA ESPERADA

Máxima: -2

45. Invertir un array

Medio · Arrays · apuntesdam.com/subject/programacion/topic/arrays

Invierte el contenido de un array sin usar un segundo array auxiliar. El programa debe imprimir el array original y el array invertido usando `Arrays.toString()`.

Código de partida (java)

```
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        int[] array = {1, 2, 3, 4, 5, 6, 7, 8};

        System.out.println("Original: " + Arrays.toString(array));

        // TODO: invierte el array en su lugar (sin array auxiliar)
        // Tu código aquí

        System.out.println("Invertido: " + Arrays.toString(array));
    }
}
```

Ejemplo: Invertir {1,2,3,4,5,6,7,8} → {8,7,6,5,4,3,2,1}

SALIDA ESPERADA

Original: [1, 2, 3, 4, 5, 6, 7, 8]
Invertido: [8, 7, 6, 5, 4, 3, 2, 1]

46. Estadísticas completas de notas

Medio · Arrays · apuntesdam.com/subject/programacion/topic/arrays

Dado un array de notas (valores entre 0 y 10), calcula y muestra: la media con dos decimales, cuántos alumnos han aprobado (nota ≥ 5), cuántos han suspendido, la nota más alta y la nota más baja.

Código de partida (java)

```
public class Main {
    public static void main(String[] args) {
        double[] notas = {8.5, 4.0, 6.5, 9.0, 3.5, 7.0, 5.0, 8.0, 2.5, 6.0};

        // TODO: calcula todas las estadísticas en un solo recorrido
        double suma = 0;
        int aprobados = 0;
        int suspensos = 0;
        double maxima = notas[0];
        double minima = notas[0];

        // Tu bucle aquí

        double media = suma / notas.length;

        System.out.printf("Media: %.2f\n", media);
        System.out.println("Aprobados: " + aprobados);
        System.out.println("Suspensos: " + suspensos);
        System.out.printf("Nota maxima: %.1f\n", maxima);
        System.out.printf("Nota minima: %.1f\n", minima);
    }
}
```

Ejemplo: Estadísticas correctas para {8.5,4.0,6.5,9.0,3.5,7.0,5.0,8.0,2.5,6.0}

SALIDA ESPERADA

```
Media: 6.00
Aprobados: 7
Suspensos: 3
Nota maxima: 9.0
Nota minima: 2.5
```

47. Suma de filas y columnas de una matriz

Difícil · Arrays · apuntesdam.com/subject/programacion/topic/arrays

Dada una matriz de enteros 3x3, calcula e imprime la suma de cada fila y la suma de cada columna. Formato: 'Fila 0: X', 'Fila 1: Y', 'Fila 2: Z', 'Col 0: A', 'Col 1: B', 'Col 2: C'.

Código de partida (java)

```
public class Main {
    public static void main(String[] args) {
        int[][] matriz = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };

        // TODO: imprime la suma de cada fila
        // Fila 0: 6, Fila 1: 15, Fila 2: 24
        for (int i = 0; i < matriz.length; i++) {
            int sumaFila = 0;
            // Tu código aquí
            System.out.println("Fila " + i + ": " + sumaFila);
        }

        // TODO: imprime la suma de cada columna
        // Col 0: 12, Col 1: 15, Col 2: 18
        for (int j = 0; j < matriz[0].length; j++) {
            int sumaCol = 0;
            // Tu código aquí
            System.out.println("Col " + j + ": " + sumaCol);
        }
    }
}
```

Ejemplo: Sumas de {{1,2,3},{4,5,6},{7,8,9}}

SALIDA ESPERADA

```
Fila 0: 6
Fila 1: 15
Fila 2: 24
Col 0: 12
Col 1: 15
Col 2: 18
```

48. El juego de la vida de Conway

Muy difícil · Arrays · apuntesdam.com/subject/programacion/topic/arrays

En una cuadrícula, cada célula está viva (#) o muerta (.). En cada generación, todas cambian a la vez según sus 8 vecinas: una viva sigue viva si tiene 2 o 3 vecinas vivas (si no, muere); una muerta nace si tiene exactamente 3. Fuera de la cuadrícula no hay células. La entrada es el número de generaciones y las filas de la cuadrícula inicial. Implementa siguiente(t), que devuelve una matriz nueva con la siguiente generación, y simula: si una generación es igual a la anterior, muestra «Estable desde la generación g» (g es la última que cambió, contando la inicial como 0) y para. Al final muestra la cuadrícula y «Células vivas: n».

Código de partida (java)

```
import java.util.*;

public class Main {
    static boolean[][] siguiente(boolean[][] t) {
        // TODO: calcular la siguiente generación
        return t;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int generaciones = sc.nextInt();
        List<String> filas = new ArrayList<>();
        while (sc.hasNext()) filas.add(sc.next());
        boolean[][] t = new boolean[filas.size()][filas.get(0).length()];
        for (int i = 0; i < t.length; i++)
            for (int j = 0; j < t[0].length; j++) t[i][j] = filas.get(i).charAt(j) == '#';
        // TODO: avanzar las generaciones, detectar si se estabiliza y mostrar el resultado
    }
}
```

Ejemplo: Planeador

ENTRADA

```
4
.....
..#...
...#..
.###..
.....
.....
```

SALIDA ESPERADA

```
.....
.....
...#..
....#.
.###.
.....
Células vivas: 5
```

Ejemplo: Parpadeador

ENTRADA

```
3
.....
..#..
...#..
...#..
.....
```

SALIDA ESPERADA

```
.....
.....
.###.
.....
.....
Células vivas: 3
```

Ejemplo: Bloque estable

ENTRADA

```
10
....
.##.
.##.
....
```

SALIDA ESPERADA

```
Estable desde la generación 0
....
.##.
.##.
....
Células vivas: 4
```

49. Contar vocales

Fácil · Strings · apuntesdam.com/subject/programacion/topic/strings

Lee una frase y cuenta cuántas vocales (a, e, i, o, u, en mayúscula o minúscula) contiene. Formato: 'Vocales: N'.

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String frase = sc.nextLine();
        int vocales = 0;

        // TODO: cuenta las vocales de la frase

        System.out.println("Vocales: " + vocales);
    }
}
```

Ejemplo: 'Hola Mundo'

ENTRADA

Hola Mundo

SALIDA ESPERADA

Vocales: 4

Ejemplo: 'programacion'

ENTRADA

programacion

SALIDA ESPERADA

Vocales: 5

50. Encuentra el fallo: la contraseña que nunca es correcta

Fácil · Strings · apuntesdam.com/subject/programacion/topic/strings

El programa lee una contraseña y debe decir «Acceso concedido» si es «java2025». Aunque se escriba bien, siempre dice «Acceso denegado». Corrígelo.

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        String clave = new Scanner(System.in).nextLine();
        if (clave == "java2025") {
            System.out.println("Acceso concedido");
        } else {
            System.out.println("Acceso denegado");
        }
    }
}
```

Ejemplo: Contraseña correcta

ENTRADA

java2025

SALIDA ESPERADA

Acceso concedido

Ejemplo: Contraseña incorrecta

ENTRADA

java2024

SALIDA ESPERADA

Acceso denegado

51. Palíndromos

Medio · Strings · apuntesdam.com/subject/programacion/topic/strings

Lee una frase y di si es palíndroma ignorando espacios y mayúsculas. Muestra 'Es palindromo' o 'No es palindromo'.

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String frase = sc.nextLine();

        boolean palindromo = false; // TODO

        System.out.println(palindromo ? "Es palindromo" : "No es palindromo");
    }
}
```

Ejemplo: 'Anita lava la tina'

ENTRADA

Anita lava la tina

SALIDA ESPERADA

Es palindromo

Ejemplo: 'Java'

ENTRADA

Java

SALIDA ESPERADA

No es palindromo

52. Encuentra el fallo: el palíndromo exigente

Medio · Strings · apuntesdam.com/subject/programacion/topic/strings

Una frase es palíndroma si se lee igual al revés sin contar mayúsculas ni espacios. El programa dice que «Anita lava la tina» no lo es. Corrígelo para que ignore mayúsculas y espacios.

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        String frase = new Scanner(System.in).nextLine();
        String limpia = frase;
        String inversa = new StringBuilder(limpia).reverse().toString();
        System.out.println(limpia.equals(inversa) ? "Es palíndromo" : "No es palíndromo");
    }
}
```


Ejemplo: Anita lava la tina

ENTRADA

Anita lava la tina

SALIDA ESPERADA

Es palíndromo

Ejemplo: No palíndromo

ENTRADA

Hola mundo

SALIDA ESPERADA

No es palíndromo

53. El cifrado de Vigenère

Difícil · Strings · apuntesdam.com/subject/programacion/topic/strings

El cifrado de Vigenère (siglo XVI) desplaza cada letra tantas posiciones como indica la letra de una clave que se repite: con la clave LIMON, la primera letra se desplaza 11 (L), la segunda 8 (I), etc. La entrada es «cifrar» o «descifrar», la clave (solo letras) y el texto. Implementa vigenere: las letras del alfabeto inglés (A-Z, a-z) se desplazan respetando mayúsculas y minúsculas y dando la vuelta al llegar a la Z; cualquier otro carácter (espacios, signos, ñ, letras con tilde) se copia sin cambios y no consume letra de la clave. Para descifrar se desplaza en sentido contrario.

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    static String vigenere(String texto, String clave, boolean cifrar) {
        // TODO: desplazar cada letra según la letra de la clave que le toca
        return texto;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String modo = sc.nextLine().trim(), clave = sc.nextLine().trim(), texto = sc.nextLine();
        System.out.println(vigenere(texto, clave, modo.equals("cifrar")));
    }
}
```

Ejemplo: Cifrar

ENTRADA

cifrar
LIMON
Ataque al amanecer

SALIDA ESPERADA

Lbmehp ix ozlvqqrc

Ejemplo: Descifrar

ENTRADA

descifrar
LIMON
Lbmehp ix ozlvqqrc

SALIDA ESPERADA

Ataque al amanecer

54. Justificar un texto

Muy difícil · Strings · apuntesdam.com/subject/programacion/topic/strings

Los procesadores de textos justifican los párrafos: reparten los espacios para que todas las líneas ocupen el mismo ancho. La primera línea de la entrada es el ancho y el resto, el texto (que puede venir en varias líneas). Reparte las palabras en líneas metiendo en cada una todas las que quepan con un espacio entre ellas. Justifica cada línea añadiendo espacios entre palabras hasta llegar al ancho; si los espacios no se reparten por igual, los primeros huecos reciben uno

más. La última línea y las que solo tienen una palabra van alineadas a la izquierda. Muestra cada línea entre barras «|...|» (rellenando con espacios hasta el ancho) y al final «n líneas de ancho caracteres». Si alguna palabra es más larga que el ancho, muestra «Error: la palabra «p» no cabe en ancho caracteres» y nada más.

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int ancho = Integer.parseInt(sc.nextLine().trim());
        StringBuilder todo = new StringBuilder();
        while (sc.hasNextLine()) todo.append(sc.nextLine()).append(' ');
        String[] palabras = todo.toString().trim().split("\\s+");
        // TODO: repartir en líneas y justificarlas
    }
}
```

Ejemplo: Ancho 30

ENTRADA

```
30
Un buen programa no es el que funciona, sino el
que
cualquier compañero puede leer y cambiar sin miedo
dentro de
seis meses.
```

SALIDA ESPERADA

```
|Un  buen programa no es el que|
|funciona,    sino    el    que|
|cualquier compañero puede leer|
|y  cambiar sin miedo dentro de|
|seis meses.                  |
5 líneas de 30 caracteres
```

Ejemplo: Ancho 16

ENTRADA

```
16
El veloz murciélago hindú comía feliz cardillo y
kiwi.
```

SALIDA ESPERADA

```
|El      veloz|
|murciélago hindú|
|comía    feliz|
|cardillo y kiwi.|
4 líneas de 16 caracteres
```

55. Encuentra el fallo: el alumno sin nombre

Fácil · Clases y Objetos · apuntesdam.com/subject/programacion/topic/clases

La clase Alumno guarda el nombre y la nota, pero al imprimir un alumno sale «null: 0.0». Corrige el constructor.

Código de partida (java)

```
public class Main {
    static class Alumno {
        private String nombre;
        private double nota;

        Alumno(String nombre, double nota) {
            nombre = nombre;
            nota = nota;
        }

        @Override
        public String toString() {
            return nombre + ": " + nota;
        }
    }

    public static void main(String[] args) {
        System.out.println(new Alumno("Lucía", 8.5));
        System.out.println(new Alumno("Mario", 6.0));
    }
}
```

Ejemplo: Dos alumnos

SALIDA ESPERADA

Lucía: 8.5
Mario: 6.0

56. Una clase Fraccion inmutable

Difícil · Clases y Objetos · apuntesdam.com/subject/programacion/topic/clases

Completa la clase Fraccion. El constructor debe lanzar `IllegalArgumentException("denominador 0")` si el denominador es 0, dejar el signo en el numerador ($1/-2$ pasa a $-1/2$) y simplificar con el máximo común divisor ($4/8$ pasa a $1/2$). Los métodos sumar, restar, multiplicar y dividir devuelven una fracción nueva sin modificar las originales; dividir entre una fracción que vale 0 lanza `ArithmeticException("división entre cero")`. `toString` muestra «num/den», o solo el numerador si el denominador es 1. El main ya lee las operaciones (por ejemplo « $1/2 + 1/3$ ») y muestra los resultados o los errores.

Código de partida (java)

```
import java.util.Scanner;

class Fraccion {
    private final long num;
    private final long den;

    Fraccion(long num, long den) {
        // TODO: rechazar denominador 0, simplificar y dejar el signo en el numerador
        this.num = num;
        this.den = den;
    }

    private static long mcd(long a, long b) {
        return b == 0 ? Math.abs(a) : mcd(b, a % b);
    }

    Fraccion sumar(Fraccion o) {
        // TODO
        return this;
    }

    Fraccion restar(Fraccion o) {
        // TODO
        return this;
    }

    Fraccion multiplicar(Fraccion o) {
        // TODO
        return this;
    }

    Fraccion dividir(Fraccion o) {
        // TODO: lanzar ArithmeticException("división entre cero") si o vale 0
        return this;
    }

    @Override
    public String toString() {
        // TODO: si el denominador es 1, solo el numerador
        return num + "/" + den;
    }
}

public class Main {
    static Fraccion leer(String s) {
        String[] p = s.split("/");
        return new Fraccion(Long.parseLong(p[0]), p.length > 1 ? Long.parseLong(p[1]) : 1);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = Integer.parseInt(sc.nextLine().trim());
        for (int i = 0; i < n; i++) {
            String linea = sc.nextLine().trim();
            String[] t = linea.split(" ");
            try {
                Fraccion a = leer(t[0]), b = leer(t[2]);
                Fraccion r = switch (t[1]) {
                    case "+" -> a.sumar(b);
                    case "-" -> a.restar(b);
                    case "*" -> a.multiplicar(b);
                    default -> a.dividir(b);
                };
                System.out.println(linea + " = " + r);
            } catch (ArithmeticException | IllegalArgumentException e) {
                System.out.println(linea + " → Error: " + e.getMessage());
            }
        }
    }
}
```

Ejemplo: Las cuatro operaciones

ENTRADA

```
4
1/2 + 1/3
3/4 - 1/4
2/3 * 9/4
1/2 / 3/4
```

SALIDA ESPERADA

```
1/2 + 1/3 = 5/6
3/4 - 1/4 = 1/2
2/3 * 9/4 = 3/2
1/2 / 3/4 = 2/3
```

Ejemplo: Signos y enteros

ENTRADA

```
4
1/-2 + 1/2
-3/-6 * 4
5/3 - 2/3
7 / -14
```

SALIDA ESPERADA

```
1/-2 + 1/2 = 0
-3/-6 * 4 = 2
5/3 - 2/3 = 1
7 / -14 = -1/2
```

57. Máquina expendedora

Muy difícil · Clases y Objetos · apuntesdam.com/subject/programacion/topic/clases

Completa la clase Maquina (los precios y las monedas van en céntimos). insertar acepta monedas de 5, 10, 20, 50, 100 y 200 y responde «Crédito: 1,50 €»; cualquier otra da «Moneda no válida: 3». comprar comprueba, en este orden, que el producto existe («No existe el producto X9»), que queda stock («Zumo agotado») y que el crédito llega («Saldo insuficiente: faltan 0,40 €»). Después calcula el cambio de mayor a menor moneda, usando las monedas de la caja más las que se acaban de insertar; si así no se puede dar exacto responde «No hay cambio: inserte el importe exacto» y no vende. Si vende: «Sale Agua. Cambio: 1×0,10 €» (o «Cambio: nada»), la caja se actualiza y el crédito vuelve a 0. devolver entrega las monedas insertadas («Devuelto: 1×2,00 €, 1×0,50 €» o «Devuelto: nada»). estado lista cada producto con su stock y el total de la caja. Usa el método euros que ya tienes para dar formato.

Código de partida (java)

```
import java.util.*;

class Producto {
    final String codigo, nombre;
    final int precio;           // en céntimos
    int stock;

    Producto(String codigo, String nombre, int precio, int stock) {
        this.codigo = codigo;
        this.nombre = nombre;
        this.precio = precio;
        this.stock = stock;
    }
}

class Maquina {
    static final int[] VALORES = {200, 100, 50, 20, 10, 5};
    private final Map<String, Producto> productos = new LinkedHashMap<>();
    private final Map<Integer, Integer> monedas = new TreeMap<>(Comparator.reverseOrder());
    private final List<Integer> insertadas = new ArrayList<>();

    Maquina() {
        for (int v : VALORES) monedas.put(v, 0);
    }

    static String euros(int centimos) {
        return centimos / 100 + "," + String.format("%02d", centimos % 100) + " €";
    }

    void cargarMonedas(int valor, int cantidad) {
        monedas.merge(valor, cantidad, Integer::sum);
    }

    void añadirProducto(Producto p) {
        productos.put(p.codigo, p);
    }

    String insertar(int valor) {
        // TODO
        return "";
    }

    String comprar(String codigo) {
        // TODO
        return "";
    }

    String devolver() {
        // TODO
        return "";
    }

    String estado() {
        // TODO
        return "";
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Maquina m = new Maquina();
        while (sc.hasNextLine()) {
            String[] t = sc.nextLine().trim().split(" ");
            switch (t[0]) {
                case "MONEDAS" -> m.cargarMonedas(Integer.parseInt(t[1]), Integer.parseInt(t[2]));
                case "PRODUCTO" -> m.añadirProducto(new Producto(t[1], t[2], Integer.parseInt(t[3]),
                    Integer.parseInt(t[4])));
                case "INSERTAR" -> System.out.println(m.insertar(Integer.parseInt(t[1])));
                case "COMPRAR" -> System.out.println(m.comprar(t[1]));
                case "DEVOLVER" -> System.out.println(m.devolver());
                case "ESTADO" -> System.out.print(m.estado());
            }
        }
    }
}
```

```

    }
    }
    }
    default -> { }
}

```

Ejemplo: Compra con cambio

ENTRADA

```

MONEDAS 100 2
MONEDAS 50 1
MONEDAS 20 3
MONEDAS 10 2
PRODUCTO A1 Agua 90 2
PRODUCTO B2 Galletas 135 1
PRODUCTO C3 Zumo 160 0
INSERTAR 100
COMPRAR A1
ESTADO

```

SALIDA ESPERADA

```

Crédito: 1,00 €
Sale Agua. Cambio: 1x0,10 €
A1 Agua 0,90 € (1)
B2 Galletas 1,35 € (1)
C3 Zumo 1,60 € (0)
Caja: 4,20 €

```

Ejemplo: Errores y devolución

ENTRADA

```

MONEDAS 100 2
MONEDAS 50 1
MONEDAS 20 3
MONEDAS 10 2
PRODUCTO A1 Agua 90 2
PRODUCTO B2 Galletas 135 1
PRODUCTO C3 Zumo 160 0
INSERTAR 3
INSERTAR 50
COMPRAR B2
COMPRAR C3
COMPRAR X9
INSERTAR 200
DEVOLVER
DEVOLVER

```

SALIDA ESPERADA

```

Moneda no válida: 3
Crédito: 0,50 €
Saldo insuficiente: faltan 0,85 €
Zumo agotado
No existe el producto X9
Crédito: 2,50 €
Devuelto: 1x2,00 €, 1x0,50 €
Devuelto: nada

```

58. Cuentas bancarias con herencia

Difícil · Herencia · apuntesdam.com/subject/programacion/topic/herencia

La clase abstracta Cuenta ya guarda el IBAN y el saldo y sabe ingresar. Completa sus dos subclases. CuentaCorriente permite quedarse en descubierto hasta -300 € (si una retirada lo supera lanza `IllegalStateException("supera el descubierto")`) y al cierre de mes cobra 2 € de comisión si el saldo es negativo. CuentaAhorro no admite descubierto (`IllegalStateException("saldo insuficiente")`), permite como mucho 3 retiradas al mes (`IllegalStateException("máximo de retiradas del mes")`) y al cierre suma un 0,5 % de interés y reinicia el contador de retiradas. En las dos, un importe menor o igual que 0 lanza `IllegalArgumentException("importe no válido")`. El main ya procesa las órdenes y trata las cuentas de forma polimórfica.

```

import java.util.*;

abstract class Cuenta {
    private final String iban;
    protected double saldo;

    Cuenta(String iban, double saldoInicial) {
        this.iban = iban;
        this.saldo = saldoInicial;
    }

    String getIban() { return iban; }
    double getSaldo() { return saldo; }

    void ingresar(double importe) {
        if (importe <= 0) throw new IllegalArgumentException("importe no válido");
        saldo += importe;
    }

    // Cada tipo de cuenta decide si se puede retirar
    abstract void retirar(double importe);

    // Se llama al final de cada mes
    abstract void cierreMes();

    abstract String tipo();

    @Override
    public String toString() {
        return String.format(Locale.US, "%s %s: %.2f", tipo(), iban, saldo);
    }
}

class CuentaCorriente extends Cuenta {
    // TODO: descubierto hasta -300; comisión de 2 € al cierre si el saldo es negativo
    CuentaCorriente(String iban, double saldo) { super(iban, saldo); }
    void retirar(double importe) { }
    void cierreMes() { }
    String tipo() { return "Corriente"; }
}

class CuentaAhorro extends Cuenta {
    // TODO: sin descubierto, máximo 3 retiradas al mes, 0,5 % de interés al cierre
    CuentaAhorro(String iban, double saldo) { super(iban, saldo); }
    void retirar(double importe) { }
    void cierreMes() { }
    String tipo() { return "Ahorro"; }
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, Cuenta> cuentas = new LinkedHashMap<>();
        while (sc.hasNextLine()) {
            String[] t = sc.nextLine().trim().split(" ");
            try {
                switch (t[0]) {
                    case "CORRIENTE" -> cuentas.put(t[1], new CuentaCorriente(t[1],
                        Double.parseDouble(t[2])));
                    case "AHORRO" -> cuentas.put(t[1], new CuentaAhorro(t[1], Double.parseDouble(t[2])));
                    case "INGRESAR" -> cuentas.get(t[1]).ingresar(Double.parseDouble(t[2]));
                    case "RETIRAR" -> cuentas.get(t[1]).retirar(Double.parseDouble(t[2]));
                    case "MES" -> {
                        for (Cuenta c : cuentas.values()) c.cierreMes();
                        System.out.println("-- cierre de mes --");
                    }
                    case "VER" -> cuentas.values().forEach(System.out::println);
                }
            } catch (IllegalArgumentException | IllegalStateException e) {
                System.out.println(String.join(" ", t) + " → " + e.getMessage());
            }
        }
    }
}

```



```
}  
}
```

Ejemplo: Descubierta y comisión

ENTRADA

```
CORRIENTE ES01 100  
RETIRAR ES01 350  
RETIRAR ES01 100  
MES  
VER
```

SALIDA ESPERADA

```
RETIRAR ES01 100 → supera el descubierto  
-- cierre de mes --  
Corriente ES01: -252.00
```

Ejemplo: Límite de retiradas e interés

ENTRADA

```
AHORRO ES02 1000  
RETIRAR ES02 100  
RETIRAR ES02 100  
RETIRAR ES02 100  
RETIRAR ES02 100  
MES  
RETIRAR ES02 50  
VER
```

SALIDA ESPERADA

```
RETIRAR ES02 100 → máximo de retiradas del mes  
-- cierre de mes --  
Ahorro ES02: 653.50
```

59. Encuentra el fallo: el producto que no está en la lista

Difícil · Herencia · apuntesdam.com/subject/programacion/topic/herencia

La clase Producto define equals para que dos productos con el mismo código sean iguales, pero lista.contains(...) dice que no encuentra uno que sí está. Corrige equals (y lo que haga falta) para que contains funcione.

Código de partida (java)

```
import java.util.*;  
  
public class Main {  
    static class Producto {  
        final String codigo;  
        final String nombre;  
        Producto(String codigo, String nombre) { this.codigo = codigo; this.nombre = nombre; }  
  
        public boolean equals(Producto otro) {  
            return codigo.equals(otro.codigo);  
        }  
    }  
  
    public static void main(String[] args) {  
        List<Producto> lista = new ArrayList<>(List.of(new Producto("A1", "Teclado"), new Producto("B2",  
            "Ratón")));  
        System.out.println("¿Está B2? " + lista.contains(new Producto("B2", "Ratón inalámbrico")));  
        System.out.println("¿Está C3? " + lista.contains(new Producto("C3", "Monitor")));  
    }  
}
```

Ejemplo: Buscar por código

SALIDA ESPERADA

```
¿Está B2? true  
¿Está C3? false
```

60. Expresiones matemáticas con polimorfismo

Muy difícil · Herencia · apuntesdam.com/subject/programacion/topic/herencia

Una expresión como $(2 \cdot x) + 3$ se puede representar como un árbol de objetos: una Suma cuyo operando izquierdo es un Producto de Num(2) y Var(x), y cuyo derecho es Num(3). Num, Var y la clase abstracta Binaria (que ya sabe escribirse entre paréntesis) están hechas. Completa Suma, Resta y Producto con tres métodos: evaluar (con los valores de las variables), derivar respecto de una variable (la suma y la resta derivan cada operando; el producto usa la regla $(a \cdot b)' = a' \cdot b + a \cdot b'$) y simplificar, que primero simplifica los operandos y luego aplica estas reglas: dos números se operan; $0 + e$ y $e + 0$ dan e; $e - 0$ da e; $e - e$ (mismo texto) da 0; $0 \cdot e$ y $e \cdot 0$ dan 0; $1 \cdot e$ y $e \cdot 1$ dan e. La entrada es la expresión en notación prefija y los valores de las variables.

```

import java.util.*;

abstract class Expr {
    abstract double evaluar(Map<String, Double> vars);
    abstract Expr derivar(String v);
    Expr simplificar() { return this; }

    static String num(double d) {
        return d == Math.floor(d) && !Double.isInfinite(d) ? String.valueOf((long) d) : String.valueOf(d);
    }
}

class Num extends Expr {
    final double valor;
    Num(double valor) { this.valor = valor; }
    double evaluar(Map<String, Double> vars) { return valor; }
    Expr derivar(String v) { return new Num(0); }
    public String toString() { return num(valor); }
}

class Var extends Expr {
    final String nombre;
    Var(String nombre) { this.nombre = nombre; }
    double evaluar(Map<String, Double> vars) { return vars.get(nombre); }
    Expr derivar(String v) { return new Num(nombre.equals(v) ? 1 : 0); }
    public String toString() { return nombre; }
}

// Clase base de las operaciones con dos operandos
abstract class Binaria extends Expr {
    final Expr izq, der;
    Binaria(Expr izq, Expr der) { this.izq = izq; this.der = der; }
    abstract String simbolo();
    public String toString() { return "(" + izq + " " + simbolo() + " " + der + ")"; }
}

class Suma extends Binaria {
    Suma(Expr a, Expr b) { super(a, b); }
    String simbolo() { return "+"; }
    // TODO: evaluar, derivar y simplificar
    double evaluar(Map<String, Double> vars) { return 0; }
    Expr derivar(String v) { return this; }
}

class Resta extends Binaria {
    Resta(Expr a, Expr b) { super(a, b); }
    String simbolo() { return "-"; }
    // TODO: evaluar, derivar y simplificar
    double evaluar(Map<String, Double> vars) { return 0; }
    Expr derivar(String v) { return this; }
}

class Producto extends Binaria {
    Producto(Expr a, Expr b) { super(a, b); }
    String simbolo() { return "*"; }
    // TODO: evaluar, derivar y simplificar
    double evaluar(Map<String, Double> vars) { return 0; }
    Expr derivar(String v) { return this; }
}

public class Main {
    // Lee una expresión en notación prefija: "+ * 2 x 3" es (2 * x) + 3
    static Expr leer(Iterator<String> it) {
        String t = it.next();
        return switch (t) {
            case "+" -> new Suma(leer(it), leer(it));
            case "-" -> new Resta(leer(it), leer(it));
            case "*" -> new Producto(leer(it), leer(it));
            default -> Character.isLetter(t.charAt(0)) ? new Var(t) : new Num(Double.parseDouble(t));
        };
    }
}

```

```

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    Expr e = leer(Arrays.asList(sc.nextLine().trim().split(" ")).iterator());
    Map<String, Double> vars = new TreeMap<>();
    while (sc.hasNext()) vars.put(sc.next(), Double.parseDouble(sc.next()));
    System.out.println("Expresión: " + e);
    System.out.println("Simplificada: " + e.simplificar());
    System.out.println("Valor: " + Expr.num(e.evaluar(vars)));
    Expr d = e.derivar("x");
    System.out.println("Derivada respecto de x: " + d);
    System.out.println("Derivada simplificada: " + d.simplificar());
}
}

```

Ejemplo: Recta

ENTRADA

```

+ * 2 x 3
x 4

```

SALIDA ESPERADA

```

Expresión: ((2 * x) + 3)
Simplificada: ((2 * x) + 3)
Valor: 11
Derivada respecto de x: (((0 * x) + (2 * 1)) + 0)
Derivada simplificada: 2

```

Ejemplo: Parábola

ENTRADA

```

- * x x * 3 x
x 5

```

SALIDA ESPERADA

```

Expresión: ((x * x) - (3 * x))
Simplificada: ((x * x) - (3 * x))
Valor: 10
Derivada respecto de x: (((1 * x) + (x * 1)) - ((0 * x) + (3 * 1)))
Derivada simplificada: ((x + x) - 3)

```

61. Clasificación de una liga con Comparator

Difícil · Interfaces · apuntesdam.com/subject/programacion/topic/interfaces

Cada línea de la entrada es un partido con el formato «Local 2-1 Visitante». Completa registrar, que actualiza las estadísticas de un equipo con los goles a favor y en contra de un partido (3 puntos por victoria y 1 por empate), y el Comparator CLASIFICACION, que ordena por puntos de mayor a menor; a igualdad de puntos, por diferencia de goles; después por goles a favor (ambos de mayor a menor) y, si todo coincide, por nombre en orden alfabético. El main ya lee los partidos y muestra la tabla.

Código de partida (java)

```
import java.util.*;

class Equipo {
    final String nombre;
    int jugados, ganados, empatados, perdidos, golesFavor, golesContra;

    Equipo(String nombre) { this.nombre = nombre; }

    int puntos() { return ganados * 3 + empatados; }
    int diferencia() { return golesFavor - golesContra; }

    void registrar(int favor, int contra) {
        // TODO: actualizar partidos, resultado y goles
    }
}

public class Main {
    // TODO: puntos (de más a menos), diferencia de goles, goles a favor y, por último, nombre (A-Z)
    static final Comparator<Equipo> CLASIFICACION = (a, b) -> 0;

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, Equipo> equipos = new HashMap<>();
        while (sc.hasNextLine()) {
            String linea = sc.nextLine().trim();
            if (linea.isEmpty()) continue;
            // Formato: Local 2-1 Visitante
            String[] t = linea.split(" ");
            String[] goles = t[1].split("-");
            int gl = Integer.parseInt(goles[0]), gv = Integer.parseInt(goles[1]);
            equipos.computeIfAbsent(t[0], Equipo::new).registrar(gl, gv);
            equipos.computeIfAbsent(t[2], Equipo::new).registrar(gv, gl);
        }
        List<Equipo> tabla = new ArrayList<>(equipos.values());
        tabla.sort(CLASIFICACION);
        System.out.println("Pos Equipo      PJ  G  E  P  GF  GC  DG Pts");
        int pos = 1;
        for (Equipo e : tabla)
            System.out.printf("%3d %-10s %3d %2d %2d %2d %3d %3d %+3d %3d%n", pos++, e.nombre, e.jugados,
                               e.ganados, e.empatados, e.perdidos, e.golesFavor, e.golesContra, e.diferencia(),
                               e.puntos());
    }
}
```

Ejemplo: Jornada simple

ENTRADA

Leones 2-1 Tigres
Osos 0-0 Lobos
Tigres 3-0 Osos
Lobos 1-1 Leones

SALIDA ESPERADA

Pos	Equipo	PJ	G	E	P	GF	GC	DG	Pts
1	Leones	2	1	1	0	3	2	+1	4
2	Tigres	2	1	0	1	4	2	+2	3
3	Lobos	2	0	2	0	1	1	+0	2
4	Osos	2	0	1	1	0	3	-3	1

Ejemplo: Desempates

ENTRADA

Alfa 3-0 Delta
Beta 2-0 Delta
Gamma 3-1 Delta
Epsilon 2-0 Delta

SALIDA ESPERADA

Pos	Equipo	PJ	G	E	P	GF	GC	DG	Pts
1	Alfa	1	1	0	0	3	0	+3	3
2	Gamma	1	1	0	0	3	1	+2	3
3	Beta	1	1	0	0	2	0	+2	3
4	Epsilon	1	1	0	0	2	0	+2	3
5	Delta	4	0	0	4	1	10	-9	0

62. Reglas de contraseñas componibles

Muy difícil · Interfaces · apuntesdam.com/subject/programacion/topic/interfaces

Regla es una interfaz funcional: su único método abstracto, comprobar, devuelve la lista de errores de un texto (vacía si lo cumple). Completa sus métodos default y static para poder combinar reglas como piezas. y(otra) devuelve una regla que exige las dos y junta los errores de ambas, en orden. o(otra) se cumple si se cumple cualquiera de las dos; si fallan ambas, da un único error con los dos mensajes unidos por « o ». minimo(n) falla con «mínimo n caracteres». contiene(clase, nombre) exige algún carácter de la clase de expresión regular [clase] y falla con «falta nombre». noContiene(palabra) falla, sin distinguir mayúsculas, con «no puede contener «palabra»». El main ya monta dos políticas con estas piezas y comprueba una contraseña por línea.

Código de partida (java)

```
import java.util.*;

@FunctionalInterface
interface Regla {
    // Lista de errores; vacía si el texto cumple la regla
    List<String> comprobar(String s);

    default Regla y(Regla otra) {
        // TODO: se deben cumplir las dos; errores de ambas
        return this;
    }

    default Regla o(Regla otra) {
        // TODO: basta con que se cumpla una; si fallan las dos, un único error «e1 o e2»
        return this;
    }

    static Regla minimo(int n) {
        // TODO: «mínimo n caracteres»
        return s -> List.of();
    }

    static Regla contiene(String clase, String nombre) {
        // TODO: el texto debe tener algún carácter de la clase [clase]; si no, «falta nombre»
        return s -> List.of();
    }

    static Regla noContiene(String palabra) {
        // TODO: sin distinguir mayúsculas; si la contiene, «no puede contener «palabra»»
        return s -> List.of();
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String politica = sc.nextLine().trim();
        Regla regla = switch (politica) {
            case "BASICA" -> Regla.minimo(6).y(Regla.contiene("0-9", "un número"));
            default -> Regla.minimo(10)
                .y(Regla.contiene("0-9", "un número"))
                .y(Regla.contiene("A-Z", "una mayúscula").o(Regla.contiene("^a-zA-Z0-9", "un símbolo")))
                .y(Regla.noContiene("password"))
                .y(Regla.noContiene("1234"));
        };
        while (sc.hasNextLine()) {
            String clave = sc.nextLine();
            List<String> errores = regla.comprobar(clave);
            System.out.println(errores.isEmpty() ? "✓ " + clave : "X " + clave + ": " + String.join("; ",
                errores));
        }
    }
}
```

Ejemplo: Política básica

ENTRADA

```
BASICA
abc
abcdef
abc123
```

SALIDA ESPERADA

```
X abc: mínimo 6 caracteres; falta un número
X abcdef: falta un número
✓ abc123
```

Ejemplo: Política estricta

ENTRADA

```
ESTRICTA
Corto1
MiClaveSegura7
miclavesegura7
miclave-segura7
MyPassword2025
```

SALIDA ESPERADA

```
X Corto1: mínimo 10 caracteres
✓ MiClaveSegura7
X miclavesegura7: falta una mayúscula o falta un
símbolo
✓ miclave-segura7
X MyPassword2025: no puede contener «password»
```

63. Validar y resumir un CSV de ventas

Difícil · Ficheros y Streams · apuntesdam.com/subject/programacion/topic/ficheros

La entrada es un fichero CSV (separado por «;», con decimales con coma) con la cabecera fecha;producto;unidades;precio y una venta por línea. Valida cada línea y muestra sus errores con su número de línea en el fichero (la cabecera es la 1; las líneas en blanco se saltan pero cuentan): «se esperaban 4 campos y hay N», «producto vacío», «unidades no válidas («dos»)», «precio no válido («2,50»)» o «valores negativos o cero» (unidades ≤ 0 o precio < 0). Solo el primer error de cada línea. Con las líneas correctas, acumula el importe (unidades $\times$ precio) por producto y muestra «--- N líneas correctas, M con errores ---», los productos de mayor a menor importe (a igualdad, por nombre) con formato «%-10s %9.2f» y una línea TOTAL.

Código de partida (java)

```
import java.io.*;
import java.util.*;

public class Main {
    public static void main(String[] args) throws IOException {
        // La entrada estándar hace de fichero: se lee igual que con un FileReader
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        Map<String, Double> ventas = new HashMap<>();
        int correctas = 0, erroneas = 0;
        String cabecera = br.readLine();
        String linea;
        int numLinea = 1;
        while ((linea = br.readLine()) != null) {
            numLinea++;
            if (linea.isBlank()) continue;
            // TODO: validar la línea; si es correcta, acumular unidades  $\times$  precio en el producto
        }
        // TODO: mostrar el resumen ordenado por importe (de mayor a menor)
    }
}
```

Ejemplo: Fichero correcto

ENTRADA

```
fecha;producto;unidades;precio
2025-03-01;Café;3;1,20
2025-03-01;Tostada;2;2,50
2025-03-02;Café;5;1,20
2025-03-02;Zumo;1;3
```

SALIDA ESPERADA

```
--- 4 líneas correctas, 0 con errores ---
Café          9.60
Tostada       5.00
Zumo          3.00
TOTAL        17.60
```

Ejemplo: Con errores

ENTRADA

```
fecha;producto;unidades;precio
2025-03-01;Café;3;1,20
2025-03-01;Tostada;dos;2,50
2025-03-01;Zumo;1

2025-03-02;Café;-1;1,20
2025-03-02;;1;2
2025-03-03;Tostada;4;2,50
2025-03-03;Tostada;4;2,50
```

SALIDA ESPERADA

```
Línea 3: unidades no válidas («dos»)
Línea 4: se esperaban 4 campos y hay 3
Línea 6: valores negativos o cero
Línea 7: producto vacío
Línea 8: precio no válido («2,50»)
--- 2 líneas correctas, 5 con errores ---
Tostada      10.00
Café         3.60
TOTAL        13.60
```

64. Comparar dos ficheros (diff)

Muy difícil · Ficheros y Streams · apuntesdam.com/subject/programacion/topic/ficheros

Herramientas como git diff muestran qué ha cambiado entre dos versiones de un fichero. La entrada contiene las líneas de la versión antigua, una línea «=====» y las de la nueva. Implementa diff: debe conservar el mayor número posible de líneas comunes en orden (la subsecuencia común más larga, LCS) y marcar el resto. Cada línea del resultado empieza por un espacio si está en las dos versiones, por «-» si solo está en la antigua y por «+» si solo está en la nueva. Recorre las dos versiones desde el principio: si las líneas actuales son iguales, se conservan; si no, se quita la de la antigua cuando eso no empeora la LCS ($\text{lcs}[i+1][j] \geq \text{lcs}[i][j+1]$) y, si no, se añade la de la nueva. El main ya cuenta los cambios.

Código de partida (java)

```
import java.io.*;
import java.util.*;

public class Main {
    // Devuelve las líneas del diff: " línea" (igual), "-línea" (solo en a), "+línea" (solo en b)
    static List<String> diff(List<String> a, List<String> b) {
        // TODO
        return new ArrayList<>();
    }

    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        List<String> a = new ArrayList<>(), b = new ArrayList<>();
        List<String> actual = a;
        String linea;
        while ((linea = br.readLine()) != null) {
            if (linea.equals("=====")) { actual = b; continue; }
            actual.add(linea);
        }
        List<String> d = diff(a, b);
        int quitadas = 0, añadidas = 0;
        for (String l : d) {
            System.out.println(l);
            if (l.startsWith("-")) quitadas++;
            if (l.startsWith("+")) añadidas++;
        }
        System.out.println("@@ " + quitadas + (quitadas == 1 ? " línea quitada, " : " líneas quitadas, ")
            + añadidas + (añadidas == 1 ? " añadida @" : " añadidas @"));
    }
}
```


Ejemplo: Un cambio

ENTRADA

```
uno
dos
tres
cuatro
=====
uno
DOS
tres
cuatro
```

SALIDA ESPERADA

```
uno
-dos
+DOS
tres
cuatro
@@ 1 línea quitada, 1 añadida @@
```

Ejemplo: Código

ENTRADA

```
public class A {
    int x;
    void f() {
        x++;
    }
}
=====
public class A {
    private int x;
    void f() {
        x++;
        log();
    }
}
```

SALIDA ESPERADA

```
public class A {
-   int x;
+   private int x;
    void f() {
        x++;
        log();
    }
}
@@ 1 línea quitada, 2 añadidas @@
```

65. Encuentra el fallo: borrar mientras se recorre

Medio · Colecciones · apuntesdam.com/subject/programacion/topic/colecciones

El programa debe quitar de la lista las tareas terminadas (las que empiezan por «x ») y mostrar las pendientes. En lugar de eso, termina con `ConcurrentModificationException`. Corrígelo.

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        List<String> tareas = new ArrayList<>(List.of("x comprar pan", "estudiar Java", "x llamar a Ana", "x pagar luz", "hacer la práctica"));
        for (String t : tareas) {
            if (t.startsWith("x ")) {
                tareas.remove(t);
            }
        }
        System.out.println("Pendientes: " + tareas);
    }
}
```

Ejemplo: Tres tareas terminadas

SALIDA ESPERADA

```
Pendientes: [estudiar Java, hacer la práctica]
```

66. Encuentra el fallo: el contador de palabras olvidadizo

Medio · Colecciones · apuntesdam.com/subject/programacion/topic/colecciones

El programa cuenta cuántas veces aparece cada palabra de una frase y las muestra en orden alfabético. Todas salen con 1 aunque se repitan. Corrígelo.

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        String frase = new Scanner(System.in).nextLine().toLowerCase();
        Map<String, Integer> contador = new TreeMap<>();
        for (String p : frase.split(" ")) {
            contador.put(p, 1);
        }
        contador.forEach((p, n) -> System.out.println(p + " " + n));
    }
}
```

Ejemplo: Frase con repeticiones

ENTRADA

el perro y el gato y el loro

SALIDA ESPERADA

el 3
gato 1
loro 1
perro 1
y 2

67. Agrupar anagramas

Difícil · Colecciones · apuntesdam.com/subject/programacion/topic/colecciones

Dos palabras son anagramas si tienen las mismas letras en distinto orden (amor, mora, Roma, ramo). La entrada es una lista de palabras. Implementa agrupar: trabaja en minúsculas (las letras con tilde cuentan como distintas: lámina no es anagrama de animal), ignora las palabras repetidas y devuelve solo los grupos con 2 o más palabras distintas. Cada grupo va ordenado alfabéticamente y los grupos se ordenan de más a menos palabras y, a igualdad, por su primera palabra. Si no hay ninguno, el main muestra «No hay anagramas».

Código de partida (java)

```
import java.util.*;

public class Main {
    // Devuelve los grupos de anagramas (2 o más palabras), cada uno ordenado alfabéticamente
    // y los grupos de más a menos palabras (a igualdad, por su primera palabra)
    static List<List<String>> agrupar(List<String> palabras) {
        // TODO
        return new ArrayList<>();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<String> palabras = new ArrayList<>();
        while (sc.hasNext()) palabras.add(sc.next());
        List<List<String>> grupos = agrupar(palabras);
        if (grupos.isEmpty()) System.out.println("No hay anagramas");
        for (List<String> g : grupos) System.out.println(g.size() + ": " + String.join(", ", g));
    }
}
```

Ejemplo: Ejemplo

ENTRADA

amor saco Roma cosa mora ramo casa ocas

SALIDA ESPERADA

4: amor, mora, ramo, roma
3: cosa, ocas, saco

Ejemplo: Repetidas y mayúsculas

ENTRADA

```
Lámina animal Manila listen silent enlist Listen  
tinsel perro
```

SALIDA ESPERADA

```
4: enlist, listen, silent, tinsel  
2: animal, manila
```

68. Ruta más corta en el metro

Muy difícil · Colecciones · apuntesdam.com/subject/programacion/topic/colecciones

Cada línea «LINEA nombre e1 e2 e3...» conecta estaciones consecutivas en los dos sentidos. Cada «RUTA origen destino» pide el camino con menos paradas. Implementa ruta con una búsqueda en anchura (BFS): una cola con las estaciones por explorar y un mapa que guarda, para cada estación descubierta, desde cuál se llegó a ella. Al sacar el destino de la cola, reconstruye el camino hacia atrás siguiendo ese mapa. Las vecinas se exploran en orden alfabético (ya vienen así en red), lo que decide el camino cuando hay empate. Si no se puede llegar, devuelve null. El main muestra «Sol → Opera → Callao (2 paradas)».

Código de partida (java)

```
import java.util.*;  
  
public class Main {  
    // Estación → estaciones vecinas (en orden alfabético)  
    static Map<String, Set<String>> red = new TreeMap<>();  
  
    static void conectar(String a, String b) {  
        red.computeIfAbsent(a, k -> new TreeSet<>()).add(b);  
        red.computeIfAbsent(b, k -> new TreeSet<>()).add(a);  
    }  
  
    // Camino con menos paradas de origen a destino (incluidos los dos), o null si no hay  
    static List<String> ruta(String origen, String destino) {  
        // TODO: búsqueda en anchura  
        return null;  
    }  
  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        while (sc.hasNextLine()) {  
            String linea = sc.nextLine().trim();  
            if (linea.isEmpty()) continue;  
            String[] t = linea.split(" ");  
            if (t[0].equals("LINEA")) {  
                for (int i = 2; i < t.length - 1; i++) conectar(t[i], t[i + 1]);  
            } else if (t[0].equals("RUTA")) {  
                if (!red.containsKey(t[1]) || !red.containsKey(t[2])) {  
                    System.out.println("Estación desconocida");  
                    continue;  
                }  
                List<String> r = ruta(t[1], t[2]);  
                if (r == null) System.out.println(t[1] + " → " + t[2] + ": sin conexión");  
                else System.out.println(String.join(" → ", r) + " (" + (r.size() - 1) + " paradas)");  
            }  
        }  
    }  
}
```

Ejemplo: Misma línea y transbordo

ENTRADA

```
LINEA L1 Sol Opera Callao Plaza Norte  
LINEA L2 Sol Retiro Goya Este  
LINEA L3 Callao Centro Goya  
LINEA L4 Isla Faro  
RUTA Sol Callao  
RUTA Opera Este
```

SALIDA ESPERADA

```
Sol → Opera → Callao (2 paradas)  
Opera → Callao → Centro → Goya → Este (4 paradas)
```

Ejemplo: Varios caminos

ENTRADA

LINEA L1 Sol Opera Callao Plaza Norte
LINEA L2 Sol Retiro Goya Este
LINEA L3 Callao Centro Goya
LINEA L4 Isla Faro
RUTA Plaza Retiro
RUTA Norte Sol
RUTA Goya Goya

SALIDA ESPERADA

Plaza → Callao → Centro → Goya → Retiro (4 paradas)
Norte → Plaza → Callao → Opera → Sol (4 paradas)
Goya (0 paradas)

Acceso a Datos

69. Media de notas desde un fichero CSV

Fácil · Manejo de Ficheros · apuntesdam.com/subject/acceso-datos/topic/gestion-ficheros

El programa guarda en notas.csv las líneas recibidas por teclado (formato nombre;nota). Completa leerMedia() para que lea el fichero línea a línea con BufferedReader y devuelva la media de las notas. Se imprime con dos decimales.

Código de partida (java)

```
import java.io.*;
import java.util.*;

public class Main {

    static double leerMedia(File fichero) throws IOException {
        double suma = 0;
        int contador = 0;
        // TODO: lee el fichero línea a línea y acumula las notas

        return contador == 0 ? 0 : suma / contador;
    }

    public static void main(String[] args) throws IOException {
        // Guarda en notas.csv las líneas recibidas por teclado
        File fichero = new File("notas.csv");
        Scanner sc = new Scanner(System.in);
        try (PrintWriter pw = new PrintWriter(new FileWriter(fichero))) {
            while (sc.hasNextLine()) pw.println(sc.nextLine());
        }

        System.out.printf(Locale.US, "Media: %.2f\n", leerMedia(fichero));
    }
}
```

Ejemplo: Tres alumnos

ENTRADA

Ana;8
Luis;6
Marta;9.5

SALIDA ESPERADA

Media: 7.83

Ejemplo: Un alumno

ENTRADA

Pedro;5

SALIDA ESPERADA

Media: 5.00

70. Serializar una lista de productos

Medio · Manejo de Ficheros · apuntesdam.com/subject/acceso-datos/topic/gestion-ficheros

Completa guardar() y cargar() para escribir la lista de productos en productos.obj con ObjectOutputStream y volver a leerla con ObjectInputStream. El programa lee los productos por teclado (nombre;precio), los guarda, los carga de nuevo y muestra cuántos hay y el importe total.

Código de partida (java)

```
import java.io.*;
import java.util.*;

public class Main {

    static class Producto implements Serializable {
        private static final long serialVersionUID = 1L;
        String nombre;
        double precio;
        Producto(String nombre, double precio) { this.nombre = nombre; this.precio = precio; }
    }

    static void guardar(List<Producto> productos, File fichero) throws IOException {
        // TODO: escribe la lista en el fichero
    }

    @SuppressWarnings("unchecked")
    static List<Producto> cargar(File fichero) throws IOException, ClassNotFoundException {
        // TODO: lee la lista desde el fichero
        return new ArrayList<>();
    }

    public static void main(String[] args) throws Exception {
        List<Producto> productos = new ArrayList<>();
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().split(";");
            if (p.length == 2) productos.add(new Producto(p[0], Double.parseDouble(p[1])));
        }

        File fichero = new File("productos.obj");
        guardar(productos, fichero);
        List<Producto> leidos = cargar(fichero);

        double total = 0;
        for (Producto p : leidos) total += p.precio;
        System.out.printf(Locale.US, "%d productos, total %.2f%n", leidos.size(), total);
    }
}
```

Ejemplo: Tres productos

ENTRADA

Teclado;49.99
Ratón;19.5
Monitor;189

SALIDA ESPERADA

3 productos, total 258.49

71. Leer un XML con DOM

Medio · Manejo de Ficheros · apuntesdam.com/subject/acceso-datos/topic/gestion-ficheros

El programa guarda en biblioteca.xml el documento recibido por teclado. Completa procesar() para cargarlo con DOM, recorrer los elementos <libro> y mostrar cuántos libros hay y cuántos están disponibles (atributo disponible="si").

Código de partida (java)

```
import java.io.*;
import java.nio.file.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;

public class Main {

    static void procesar(File fichero) throws Exception {
        int total = 0, disponibles = 0;
        // TODO: carga el XML con DOM y cuenta los libros

        System.out.println("Libros: " + total + ", disponibles: " + disponibles);
    }

    public static void main(String[] args) throws Exception {
        File fichero = new File("biblioteca.xml");
        Files.write(fichero.toPath(), System.in.readAllBytes());
        procesar(fichero);
    }
}
```

Ejemplo: Tres libros, dos disponibles

ENTRADA

```
<biblioteca><libro disponible="si">
<titulo>Java</titulo></libro><libro
disponible="no"><titulo>SQL</titulo></libro><libro
disponible="si"><titulo>XML</titulo></libro>
</biblioteca>
```

SALIDA ESPERADA

```
Libros: 3, disponibles: 2
```

72. Acceso aleatorio a registros

Difícil · Manejo de Ficheros · apuntesdam.com/subject/acceso-datos/topic/gestion-ficheros

El programa crea empleados.dat con registros de tamaño fijo: id (int, 4 bytes), nombre (10 caracteres = 20 bytes) y salario (double, 8 bytes). Completa leerRegistro() para que use seek() y lea directamente el registro número n (empezando en 1) sin recorrer los anteriores. Se lee n por teclado.

Código de partida (java)

```
import java.io.*;
import java.util.*;

public class Main {
    static final int TAM = 4 + 20 + 8; // id + nombre(10 chars) + salario

    static void escribir(RandomAccessFile raf, int id, String nombre, double salario) throws IOException {
        raf.writeInt(id);
        StringBuilder sb = new StringBuilder(nombre);
        sb.setLength(10); // rellena o recorta a 10 caracteres
        raf.writeChars(sb.toString());
        raf.writeDouble(salario);
    }

    static String leerRegistro(RandomAccessFile raf, int n) throws IOException {
        // TODO: sitúate en el registro n y lee sus tres campos
        return "";
    }

    public static void main(String[] args) throws IOException {
        try (RandomAccessFile raf = new RandomAccessFile("empleados.dat", "rw")) {
            raf.setLength(0);
            escribir(raf, 1, "Ana", 1850.5);
            escribir(raf, 2, "Luis", 2100);
            escribir(raf, 3, "Marta", 2475.25);
            escribir(raf, 4, "Jorge", 1990);

            int n = new Scanner(System.in).nextInt();
            System.out.println(leerRegistro(raf, n));
        }
    }
}
```

Ejemplo: Registro 1

ENTRADA

1

SALIDA ESPERADA

1 Ana 1850.5

Ejemplo: Registro 3

ENTRADA

3

SALIDA ESPERADA

3 Marta 2475.25

73. Un CSV de verdad: comillas, separadores y saltos de línea

Difícil · Manejo de Ficheros · apuntesdam.com/subject/acceso-datos/topic/gestion-ficheros

Partir cada línea por el separador falla con CSV reales: un campo entre comillas dobles puede contener el separador, saltos de línea y comillas (escritas dos veces: ""). Lee todo el texto (separador ; como el Excel en español) y analízalo carácter a carácter según el RFC 4180: fuera de comillas, ; separa campos y el salto de línea termina el registro (las líneas vacías se ignoran); una comilla al principio abre un campo entrecomillado, dentro del cual "" es una comilla y " cierra. La primera fila es la cabecera. Para cada registro de datos con tantos campos como la cabecera, muestra «Registro i:» y una línea « cabecera = valor» por campo (los saltos de línea dentro del valor se muestran como \n y un valor vacío como «(vacío)»); una comilla en mitad de un campo sin comillas es un carácter normal; si el número de campos no coincide, «Error en el registro i: tiene X campos y se esperaban M». Termina con «Registros válidos: V de T · Campos: M». Si el texto acaba con una comilla sin cerrar: «Error: comilla sin cerrar en el registro i» (i = registros completos antes de ese) y nada más.

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in).useDelimiter("\\A");
        String texto = sc.hasNext() ? sc.next().replace("\r", "") : "";
        List<List<String>> registros = new ArrayList<>();
        // TODO: analizar el CSV carácter a carácter (separador ; y comillas dobles)
    }
}
```

Ejemplo: Alumnos con campos difíciles

ENTRADA

```
nif;nombre;direccion;nota
12345678Z;"García; Ana";"Calle Mayor 3
2º B";8,5
87654321X;Luis "el rápido";Avenida Sol 1;7
X1234567L;"Pérez, "Lolo""";
00000000T;Falta;un campo
```

SALIDA ESPERADA

```
Registro 1:
  nif = 12345678Z
  nombre = García; Ana
  direccion = Calle Mayor 3
  2º B
  nota = 8,5
Registro 2:
  nif = 87654321X
  nombre = Luis "el rápido"
  direccion = Avenida Sol 1
  nota = 7
Registro 3:
  nif = X1234567L
  nombre = Pérez, "Lolo"
  direccion = (vacío)
  nota = (vacío)
Error en el registro 4: tiene 3 campos y se
esperaban 4
Registros válidos: 3 de 4 · Campos: 4
```

74. Leer un formato binario propio con versión y CRC

Muy difícil · Manejo de Ficheros · apuntesdam.com/subject/acceso-datos/topic/gestion-ficheros

Muchas aplicaciones guardan sus datos en un formato binario propio. Este guarda alumnos así (todo en big-endian, como `DataOutputStream`): 4 bytes ASCII «ALUM»; un short con la versión (1 o 2); un int con el número de registros; por cada registro, un int id, un short con la longitud en bytes del nombre seguido del nombre en UTF-8, en la versión 2 un byte con el curso, y un float con la nota; y al final un int con el CRC32 de todos los bytes anteriores. La entrada es el fichero en hexadecimal. Comprueba en este orden: la firma (si no es ALUM, «Formato desconocido»), la versión («Versión X no soportada»), que no se acaben los bytes antes de tiempo («Fichero truncado») y el CRC («CRC incorrecto: esperado XXXXXXXX, calculado YYYYYYYY», en hexadecimal con 8 cifras). Si todo es correcto, muestra cada registro como «id · nombre · nota» (en la versión 2, «id · nombre · curso N · nota»), con la nota con dos decimales, y «Registros: n · media M · versión v · CRC OK».

Código de partida (java)

```
import java.nio.*;
import java.util.*;
import java.util.zip.CRC32;

public class Main {
    static byte[] desdeHex(String h) {
        byte[] b = new byte[h.length() / 2];
        for (int i = 0; i < b.length; i++) b[i] = (byte) Integer.parseInt(h.substring(2 * i, 2 * i + 2),
            16);
        return b;
    }

    public static void main(String[] args) {
        byte[] datos = desdeHex(new Scanner(System.in).nextLine().trim());
        ByteBuffer buf = ByteBuffer.wrap(datos); // big-endian, como DataInputStream
        // TODO: leer cabecera, registros y CRC
    }
}
```

Ejemplo: Versión 1 correcta

ENTRADA

```
414C554D00010000000300000001000B416E612047617263C3
AD614108000000000002000B49C3B1616B69205275697A40C8
000000000003000C5A6FC3AB204D617274C3AD6E411C000054
FEC7DF
```

SALIDA ESPERADA

```
1 · Ana García · 8.50
2 · Iñaki Ruiz · 6.25
3 · Zoë Martín · 9.75
Registros: 3 · media 8.17 · versión 1 · CRC OK
```

Ejemplo: Versión 2 con curso

ENTRADA

```
414C554D000200000002000000A00064C7563C3AD610140F0
00000000000B000C4A6F73C3A9204D6172C3AD610240A00000
27FC88E5
```

SALIDA ESPERADA

```
10 · Lucía · curso 1 · 7.50
11 · José María · curso 2 · 5.00
Registros: 2 · media 6.25 · versión 2 · CRC OK
```

Ejemplo: CRC alterado

ENTRADA

```
414C554D00010000000300000001000B416E612047617263C3
AD614108000000000002000B49C3B1616B69205275697A40C8
000000000003000C5A6FC3AB204D617274C3AD6E411C000054
FEC7DE
```

SALIDA ESPERADA

```
CRC incorrecto: esperado 54FEC7DE, calculado
54FEC7DF
```

75. Sentencias preparadas frente a concatenación

Medio · Conectores JDBC · apuntesdam.com/subject/acceso-datos/topic/jdbc

Un PreparedStatement envía la sentencia y los datos por separado, así que un dato nunca puede convertirse en código SQL. Para verlo, simula lo que pasaría al concatenar. La primera línea es la plantilla con ? como marcadores; cada línea siguiente es un parámetro «tipo:valor» con tipo s (texto), i (entero), d (decimal con punto), b (booleano), f (fecha) o n (nulo). Los ? dentro de un literal entre comillas simples de la plantilla no son marcadores. Muestra «Parámetros: N» y, si no coincide con los datos, «Error: la sentencia tiene N parámetros y se han dado M». Valida que los enteros y decimales lo sean («Error: el parámetro k no es un entero: x» / «no es un decimal»). Después muestra «Concatenando: ...» (cada texto entre comillas tal cual) y «Preparada: ...» (con las comillas simples del texto duplicadas, que es como queda el dato escapado); los booleanos son TRUE/FALSE, las fechas DATE '...' y los nulos NULL. Termina con «¡Inyección! Con la concatenación el texto del usuario cambia la estructura de la consulta» si algún texto contiene una comilla simple, o «Las dos versiones son equivalentes con estos datos».

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String plantilla = sc.nextLine();
        List<String[]> params = new ArrayList<>(); // {tipo, valor}
        while (sc.hasNextLine()) {
            String l = sc.nextLine();
            if (l.isBlank()) continue;
            int dos = l.indexOf(':');
            params.add(new String[] { l.substring(0, dos), l.substring(dos + 1) });
        }
        // TODO: localizar los ? fuera de los literales y montar las dos versiones
    }
}
```

Ejemplo: Inicio de sesión atacado

ENTRADA

```
SELECT * FROM usuario WHERE login = ? AND clave = ?
s:admin
s:' OR '1'='1'
```

SALIDA ESPERADA

Parámetros: 2
Concatenando: SELECT * FROM usuario WHERE login = 'admin' AND clave = '' OR '1'='1'
Preparada: SELECT * FROM usuario WHERE login = 'admin' AND clave = '' OR '1'='1'
¡Inyección! Con la concatenación el texto del usuario cambia la estructura de la consulta

Ejemplo: Tipos y ? en un literal

ENTRADA

```
UPDATE alumno SET nota = ?, aprobado = ?,
nacimiento = ?, tutor = ? WHERE nombre = ? AND obs
<> '¿Qué?'
d:7.5
b:true
f:2005-03-14
n:
s:O'Brien
```

SALIDA ESPERADA

Parámetros: 5
Concatenando: UPDATE alumno SET nota = 7.5, aprobado = TRUE, nacimiento = DATE '2005-03-14', tutor = NULL WHERE nombre = 'O'Brien' AND obs <> '¿Qué?'
Preparada: UPDATE alumno SET nota = 7.5, aprobado = TRUE, nacimiento = DATE '2005-03-14', tutor = NULL WHERE nombre = 'O'Brien' AND obs <> '¿Qué?'
¡Inyección! Con la concatenación el texto del usuario cambia la estructura de la consulta

76. Un pool de conexiones por dentro

Difícil · Conectores JDBC · apuntesdam.com/subject/acceso-datos/topic/jdbc

Abrir una conexión a la base de datos cuesta decenas de milisegundos, así que las aplicaciones usan un pool (HikariCP, el de Tomcat...) que las reutiliza. La primera línea es «max M espera W» y las siguientes, ordenadas por tiempo, «t pedir cliente» o «t devolver cliente». Al pedir: si hay una conexión libre se entrega la que lleva más tiempo libre («t: X recibe la conexión #k (reutilizada)»); si no, y hay menos de M creadas en total, se crea una nueva numerada desde 1 («(nueva)»); si no, el cliente espera en una cola FIFO («t: X espera (pool lleno)»). Al devolver: si alguien espera, la conexión pasa al primero de la cola («t: X devuelve #k → pasa a Y (esperó d ms)»); si no, queda libre («t: X devuelve #k (queda libre)»); si el cliente no tenía conexión, «t: Error: X no tiene conexión». Quien lleva esperando más de W ms (su inicio + W es menor que el instante del siguiente evento) se va antes de procesar ese evento: «inicio+W: X agota la espera (timeout)»; al terminar los eventos se agotan los que queden. Termina con «Conexiones creadas: c · Reutilizadas: r · Esperas: e · Timeouts: o · Espera media: m ms» (media entera de las esperas que sí recibieron conexión; 0 si ninguna) y «Al final: c abiertas, l libres, u en uso».

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] conf = sc.nextLine().trim().split(" "); // «max M espera W»
        int max = Integer.parseInt(conf[1]), espera = Integer.parseInt(conf[3]);
        List<String[]> eventos = new ArrayList<>(); // {tiempo, accion, cliente}
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (!l.isEmpty()) eventos.add(l.split(" "));
        }
        // TODO: simular el pool (conexiones libres, en uso, cola de espera y tiempos límite)
    }
}
```

Ejemplo: Aplicación web

ENTRADA

```
max 2 espera 100
0 pedir web1
5 pedir web2
10 pedir web3
40 devolver web1
45 pedir web4
50 devolver web3
60 pedir web5
70 devolver web2
80 devolver web5
90 pedir web7
200 pedir web6
210 devolver web9
320 devolver web4
330 pedir web8
340 pedir web10
```

SALIDA ESPERADA

```
0: web1 recibe la conexión #1 (nueva)
5: web2 recibe la conexión #2 (nueva)
10: web3 espera (pool lleno)
40: web1 devuelve #1 → pasa a web3 (esperó 30 ms)
45: web4 espera (pool lleno)
50: web3 devuelve #1 → pasa a web4 (esperó 5 ms)
60: web5 espera (pool lleno)
70: web2 devuelve #2 → pasa a web5 (esperó 10 ms)
80: web5 devuelve #2 (queda libre)
90: web7 recibe la conexión #2 (reutilizada)
200: web6 espera (pool lleno)
210: Error: web9 no tiene conexión
300: web6 agota la espera (timeout)
320: web4 devuelve #1 (queda libre)
330: web8 recibe la conexión #1 (reutilizada)
340: web10 espera (pool lleno)
440: web10 agota la espera (timeout)
Conexiones creadas: 2 · Reutilizadas: 2 · Esperas:
5 · Timeouts: 2 · Espera media: 15 ms
Al final: 2 abiertas, 0 libres, 2 en uso
```

77. Transacciones con puntos de guardado y bloqueos

Muy difícil · Conectores JDBC · apuntesdam.com/subject/acceso-datos/topic/jdbc

Simula dos conexiones JDBC (sesiones A y B) sobre unas cuentas, como haría una base de datos con nivel de aislamiento READ COMMITTED. La primera línea es «cuentas C1=v1 C2=v2 ...» y cada línea siguiente «S orden»: BEGIN (abre una transacción: setAutoCommit(false); si ya hay una, «S: ya hay una transacción abierta»); SELECT c («S ve c = v»: el valor propio pendiente si la sesión lo ha cambiado en su transacción y, si no, el confirmado: nunca ve lo no confirmado de la otra); UPDATE c ±n (si la otra sesión tiene c cambiada en una transacción abierta, «S: UPDATE c bloqueado por O (se cancela la sentencia)»; si el saldo quedaría negativo, «S: UPDATE c rechazado: el saldo quedaría en v»; si no, sin transacción se confirma al momento «S: c = v (confirmado)» y dentro de una, «S: c = v»); SAVEPOINT nombre; ROLLBACK TO nombre (vuelve al estado del punto y borra los puntos posteriores; «S: vuelta a nombre» o «S: no existe el punto de guardado nombre»); COMMIT («S: COMMIT (n cambios)», n = cuentas distintas cambiadas) y ROLLBACK («S: ROLLBACK (n cambios descartados)»). COMMIT, ROLLBACK y SAVEPOINT sin transacción: «S: no hay transacción». Un UPDATE rechazado no deshace la transacción. Al final, por cada sesión con una transacción abierta (A y luego B), «S tenía una transacción abierta: se deshace al cerrar la conexión», y «Confirmado: C1=v1 ...» en el orden inicial.

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, Integer> confirmado = new LinkedHashMap<>(); // lo que ve cualquiera
        for (String par : sc.nextLine().trim().split(" ")) {
            if (!par.contains("=")) continue; // «cuentas A1=100 A2=50 ...»
            String[] kv = par.split("=");
            confirmado.put(kv[0], Integer.parseInt(kv[1]));
        }
        // Por sesión: cambios pendientes (null si no hay transacción) y puntos de guardado
        Map<String, Map<String, Integer>> pendientes = new HashMap<>();
        Map<String, LinkedHashMap<String, Map<String, Integer>>> puntos = new HashMap<>();
        // TODO: procesar las órdenes de las sesiones A y B
    }
}
```

Ejemplo: Transferencia con comisión

ENTRADA

```
cuentas A1=100 A2=50 A3=0
A BEGIN
A UPDATE A1 -30
A UPDATE A2 +30
B SELECT A1
A SELECT A1
B UPDATE A1 +5
B UPDATE A3 +10
A SAVEPOINT antes_comision
A UPDATE A2 -100
A UPDATE A3 -2
A ROLLBACK TO antes_comision
A UPDATE A2 -1
A COMMIT
B SELECT A1
B UPDATE A1 +5
B BEGIN
B UPDATE A2 -79
B SELECT A2
A SELECT A2
```

SALIDA ESPERADA

```
A: BEGIN
A: A1 = 70
A: A2 = 80
B ve A1 = 100
A ve A1 = 70
B: UPDATE A1 bloqueado por A (se cancela la
sentencia)
B: A3 = 10 (confirmado)
A: SAVEPOINT antes_comision
A: UPDATE A2 rechazado: el saldo quedaría en -20
A: A3 = 8
A: vuelta a antes_comision
A: A2 = 79
A: COMMIT (2 cambios)
B ve A1 = 70
B: A1 = 75 (confirmado)
B: BEGIN
B: A2 = 0
B ve A2 = 0
A ve A2 = 79
B tenía una transacción abierta: se deshace al
cerrar la conexión
Confirmado: A1=75 A2=79 A3=10
```

78. El problema N+1 y las estrategias de carga

Medio · Mapeo Objeto-Relacional (Hibernate/JPA) · apuntesdam.com/subject/acceso-datos/topic/hibernate

Con JPA, una consulta inocente como «SELECT a FROM Autor a» seguida de `a.getLibros()` en un bucle puede lanzar una consulta por autor: el problema N+1. Simula lo que hace Hibernate. La entrada tiene líneas «autor id nombre», «libro id idAutor título» y «estrategia E». Se ejecuta la consulta de autores (ordenados por id) y después, para cada autor, se accede a sus libros (ordenados por id) y se muestra «Nombre: n libros (Título, Título)» («1 libro», o «0 libros» sin paréntesis). Muestra cada SQL en el momento en que se lanza: «SQL: select * from autor order by id» al principio y, según la estrategia: LAZY, «SQL: select * from libro where autor_id = X» justo antes de usar los libros de cada autor; EAGER, esas mismas consultas pero todas seguidas tras la primera (con JPQL, EAGER no evita el N+1); JOIN FETCH, una única consulta «SQL: select * from autor a left join libro l on l.autor_id = a.id order by a.id» que lo trae todo; BATCH k, al necesitar los libros de un autor no cargado se cargan los de hasta k autores no cargados a partir de él: «SQL: select * from libro where autor_id in (1, 2)»; SUBSELECT, la primera vez se cargan los de todos: «SQL: select * from libro where autor_id in (select id from autor)». Termina con «Consultas: n (estrategia E)».

Código de partida (java)

```
import java.util.*;

public class Main {
    record Autor(int id, String nombre) { }
    record Libro(int id, int autorId, String titulo) { }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<Autor> autores = new ArrayList<>();
        List<Libro> libros = new ArrayList<>();
        String estrategia = "LAZY";
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (l.startsWith("autor ")) {
                String[] p = l.split(" ", 3);
                autores.add(new Autor(Integer.parseInt(p[1]), p[2]));
            } else if (l.startsWith("libro ")) {
                String[] p = l.split(" ", 4);
                libros.add(new Libro(Integer.parseInt(p[1]), Integer.parseInt(p[2]), p[3]));
            } else if (l.startsWith("estrategia ")) {
                estrategia = l.substring(11).trim();
            }
        }
        autores.sort(Comparator.comparingInt(Autor::id));
        libros.sort(Comparator.comparingInt(Libro::id));
        // TODO: simular «SELECT a FROM Autor a» y el acceso a a.getLibros() de cada autor
    }
}
```

Ejemplo: Estrategia LAZY

ENTRADA

```
autor 1 Cervantes
autor 2 Galdós
autor 3 Emilia Pardo Bazán
autor 4 Machado
autor 5 Lorca
libro 10 1 Don Quijote
libro 11 2 Fortunata y Jacinta
libro 12 1 Novelas ejemplares
libro 13 3 Los pazos de Ulloa
libro 14 5 Bodas de sangre
libro 15 5 Yerma
libro 16 2 Misericordia
estrategia LAZY
```

SALIDA ESPERADA

```
SQL: select * from autor order by id
SQL: select * from libro where autor_id = 1
Cervantes: 2 libros (Don Quijote, Novelas ejemplares)
SQL: select * from libro where autor_id = 2
Galdós: 2 libros (Fortunata y Jacinta, Misericordia)
SQL: select * from libro where autor_id = 3
Emilia Pardo Bazán: 1 libro (Los pazos de Ulloa)
SQL: select * from libro where autor_id = 4
Machado: 0 libros
SQL: select * from libro where autor_id = 5
Lorca: 2 libros (Bodas de sangre, Yerma)
Consultas: 6 (estrategia LAZY)
```

Ejemplo: Estrategia JOIN FETCH

ENTRADA

```
autor 1 Cervantes
autor 2 Galdós
autor 3 Emilia Pardo Bazán
autor 4 Machado
autor 5 Lorca
libro 10 1 Don Quijote
libro 11 2 Fortunata y Jacinta
libro 12 1 Novelas ejemplares
libro 13 3 Los pazos de Ulloa
libro 14 5 Bodas de sangre
libro 15 5 Yerma
libro 16 2 Misericordia
estrategia JOIN FETCH
```

SALIDA ESPERADA

```
SQL: select * from autor a left join libro l on l.autor_id = a.id order by a.id
Cervantes: 2 libros (Don Quijote, Novelas ejemplares)
Galdós: 2 libros (Fortunata y Jacinta, Misericordia)
Emilia Pardo Bazán: 1 libro (Los pazos de Ulloa)
Machado: 0 libros
Lorca: 2 libros (Bodas de sangre, Yerma)
Consultas: 1 (estrategia JOIN FETCH)
```

Ejemplo: Estrategia BATCH 2

ENTRADA

```
autor 1 Cervantes
autor 2 Galdós
autor 3 Emilia Pardo Bazán
autor 4 Machado
autor 5 Lorca
libro 10 1 Don Quijote
libro 11 2 Fortunata y Jacinta
libro 12 1 Novelas ejemplares
libro 13 3 Los pazos de Ulloa
libro 14 5 Bodas de sangre
libro 15 5 Yerma
libro 16 2 Misericordia
estrategia BATCH 2
```

SALIDA ESPERADA

```
SQL: select * from autor order by id
SQL: select * from libro where autor_id in (1, 2)
Cervantes: 2 libros (Don Quijote, Novelas
ejemplares)
Galdós: 2 libros (Fortunata y Jacinta,
Misericordia)
SQL: select * from libro where autor_id in (3, 4)
Emilia Pardo Bazán: 1 libro (Los pazos de Ulloa)
Machado: 0 libros
SQL: select * from libro where autor_id in (5)
Lorca: 2 libros (Bodas de sangre, Yerma)
Consultas: 4 (estrategia BATCH 2)
```

79. Los estados de una entidad JPA

Difícil · Mapeo Objeto-Relacional (Hibernate/JPA) · apuntesdam.com/subject/acceso-datos/topic/hibernate

En JPA una entidad puede estar NUEVA (transient), GESTIONADA (managed: el EntityManager vigila sus cambios), SEPARADA (detached) o ELIMINADA (removed), y el SQL no se lanza cuando llamas a los métodos sino en el flush. Simula un EntityManager con una única tabla por entidad e ids IDENTITY (1, 2, 3...). Órdenes: «new v Clase c1=v1,c2=v2» («v: NUEVA (transient)»); «persist v» (NUEVA → inserta al momento: «SQL: insert into clase (c1, c2) values ('v1', 'v2') → id N» y pasa a GESTIONADA; si ya lo está, «v ya está gestionada: no se hace nada»; si está separada, «PersistentObjectException: v está separada (usa merge)»); «set v campo=valor» (si no está gestionada, «v (estado): el cambio solo está en memoria»); «find v Clase id» (si esa fila ya está gestionada en el contexto, se devuelve la misma instancia sin SQL: «v es la misma instancia que w (caché de primer nivel, sin SQL)», con w la primera variable por orden alfabético que la apunta; si no, «SQL: select * from clase where id = N» y «v: GESTIONADA {campos}» o «v = null»); «detach v»; «merge v w» (copia el estado de w en la instancia gestionada con su id, cargándola con un select si no está en el contexto; w no cambia: «v: copia gestionada de w (w sigue separada)»); «remove v» (solo gestionadas: «v: ELIMINADA (se borrará en el flush)»; si no, «IllegalArgumentException: v no está gestionada»); «flush» y «commit» (recorren el contexto en orden: las eliminadas lanzan «SQL: delete ...» y dejan de estar en el contexto; las gestionadas con campos distintos de su copia del último flush lanzan «SQL: update clase set c = 'v', ... where id = N» solo con los campos cambiados; después «flush: n sentencias» o «COMMIT (n sentencias en el flush)», con «1 sentencia» en singular); «close» («Contexto cerrado: n entidades pasan a SEPARADA») y «estado v».

Código de partida (java)

```
import java.util.*;

public class Main {
    static class Entidad {
        Long id;
        String tabla;
        Map<String, String> campos = new LinkedHashMap<>();
        String estado = "NUEVA"; // NUEVA, GESTIONADA, SEPARADA, ELIMINADA
        Map<String, String> instantanea; // copia al pasar a gestionada o tras el último flush
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, Entidad> variables = new HashMap<>();
        Map<Long, Map<String, String>> baseDatos = new TreeMap<>();
        List<Entidad> contexto = new ArrayList<>(); // contexto de persistencia (en orden de entrada)
        long[] siguienteId = { 1 };
        while (sc.hasNextLine()) {
            String linea = sc.nextLine().trim();
            if (linea.isEmpty()) continue;
            String[] p = linea.split(" ", 3);
            // TODO: new, persist, set, find, detach, merge, remove, flush, commit, close y estado
        }
    }
}
```

Ejemplo: Ciclo de vida completo

ENTRADA

```
new e1 Alumno nombre=Ana,nota=6
persist e1
set e1 nota=8
set e1 nota=6
flush
set e1 nota=9
find e2 Alumno 1
commit
detach e1
set e1 nombre=Ana María
commit
persist e1
merge e3 e1
estado e3
commit
remove e1
remove e3
commit
find e4 Alumno 1
```

SALIDA ESPERADA

```
e1: NUEVA (transient)
SQL: insert into alumno (nombre, nota) values ('Ana', '6') → id 1
flush: 0 sentencias
e2 es la misma instancia que e1 (caché de primer nivel, sin SQL)
SQL: update alumno set nota = '9' where id = 1
COMMIT (1 sentencia en el flush)
e1: SEPARADA
e1 (separada): el cambio solo está en memoria
COMMIT (0 sentencias en el flush)
PersistentObjectException: e1 está separada (usa merge)
SQL: select * from alumno where id = 1
e3: copia gestionada de e1 (e1 sigue separada)
e3: GESTIONADA id=1 {nombre=Ana María, nota=9}
SQL: update alumno set nombre = 'Ana María' where id = 1
COMMIT (1 sentencia en el flush)
IllegalArgumentExcepion: e1 no está gestionada
e3: ELIMINADA (se borrará en el flush)
SQL: delete from alumno where id = 1
COMMIT (1 sentencia en el flush)
SQL: select * from alumno where id = 1
e4 = null
```

80. Arrays y tipos compuestos de PostgreSQL

Medio · BD Objeto-Relacionales y Orientadas a Objetos · apuntesdam.com/subject/acceso-datos/topic/bd-objeto-relacionales

PostgreSQL permite columnas de tipo array (text[]) y tipos compuestos (CREATE TYPE direccion AS (calle text, cp text, ciudad text)). Si se leen por JDBC con getString, llegan en su formato de texto y hay que interpretarlo. Cada línea de la entrada es «array {...}» o «compuesto (...);» implementa leerArray y leerCompuesto, que devuelven la lista de valores (null para los nulos). Array: elementos separados por comas; un elemento entre comillas dobles puede contener comas y espacios (los espacios de fuera de las comillas se ignoran) y dentro de él \\ escapa el carácter siguiente (\ es una comilla); sin comillas se quitan los espacios de alrededor y NULL (en cualquier combinación de mayúsculas) es nulo, pero "NULL"

entre comillas es el texto NULL; {} es un array vacío. Compuesto: campos separados por comas; un campo vacío sin comillas es nulo y "" es una cadena vacía; entre comillas, "" o \" es una comilla; sin comillas el texto se toma tal cual. El main ya muestra cada resultado como «tipo ["a", null, ...] · n valores · nulos: k».

Código de partida (java)

```
import java.util.*;

public class Main {
    static String json(List<String> valores) {
        List<String> partes = new ArrayList<>();
        for (String v : valores)
            partes.add(v == null ? "null" : "\"" + v.replace("\\", "\\\\").replace("\"", "\\\"") + "\"");
        return "[" + String.join(", ", partes) + "]";
    }

    static List<String> leerArray(String texto) {
        // TODO: elementos separados por comas, comillas con escapes y NULL
        return new ArrayList<>();
    }

    static List<String> leerCompuesto(String texto) {
        // TODO: campos separados por comas, vacío = nulo y comillas dobladas
        return new ArrayList<>();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String l = sc.nextLine();
            if (l.isBlank()) continue;
            String tipo = l.substring(0, l.indexOf(' ')), texto = l.substring(l.indexOf(' ') + 1).trim();
            List<String> v = tipo.equals("array") ? leerArray(texto) : leerCompuesto(texto);
            System.out.println(tipo + " " + json(v) + " · " + v.size() + (v.size() == 1 ? " valor" : " valores")
                               + " · nulos: " + v.stream().filter(Objects::isNull).count());
        }
    }
}
```

Ejemplo: Arrays y compuestos

ENTRADA

```
array {Lunes,Martes,"Fin de semana",NULL}
array {}
array {"NULL",null, 3 , "con \"comillas\" y \\barra"}
compuesto ("Calle Mayor, 3",28013,Madrid,)
compuesto (,"",abc)
compuesto ("dijo ""hola""",x)
```

SALIDA ESPERADA

```
array ["Lunes", "Martes", "Fin de semana", null] ·
4 valores · nulos: 1
array [] · 0 valores · nulos: 0
array ["NULL", null, "3", "con \"comillas\" y \\barra"] · 4 valores · nulos: 1
compuesto ["Calle Mayor, 3", "28013", "Madrid",
null] · 4 valores · nulos: 1
compuesto [null, "", "abc"] · 3 valores · nulos: 1
compuesto ["dijo \"hola\"", "x"] · 2 valores ·
nulos: 0
```

81. Profundidad de activación en una base de objetos

Difícil · BD Objeto-Relacionales y Orientadas a Objetos · apuntesdam.com/subject/acceso-datos/topic/bd-objeto-relacionales

Las bases de datos orientadas a objetos (db4o, ObjectDB) guardan grafos de objetos con referencias entre ellos. Al recuperar un objeto no cargan todo el grafo: lo «activan» solo hasta cierta profundidad, y los objetos más lejanos quedan como referencias sin rellenar. Las líneas «objeto ID Clase campo=valor ref=@ID ...» definen el almacén (los valores con @ son referencias a otros objetos, que pueden formar ciclos). Implementa activar para que «activar ID k» muestre el árbol: con profundidad ≥ 1 el objeto se activa y se muestra «Clase ID {campo=valor, ...}» y después cada referencia, en orden, con dos espacios más de sangría y el prefijo «campo → », activada con profundidad k-1; con profundidad 0 se muestra «Clase ID (sin activar)»; si el id no existe, «null (ID no existe)». El main cuenta los objetos distintos activados.

Código de partida (java)

```
import java.util.*;

public class Main {
    static class Objeto {
        String id, clase;
        Map<String, String> datos = new LinkedHashMap<>(); // campos simples
        Map<String, String> referencias = new LinkedHashMap<>(); // campo → id del objeto apuntado
    }

    static Map<String, Objeto> almacen = new LinkedHashMap<>();
    static Set<String> activados = new HashSet<>();

    static void activar(String id, int profundidad, String sangria, String prefijo) {
        // TODO: mostrar el objeto y activar sus referencias con profundidad - 1
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");
            if (p[0].equals("objeto")) { // objeto ID Clase campo=valor ref=@ID ...
                Objeto o = new Objeto();
                o.id = p[1]; o.clase = p[2];
                for (int i = 3; i < p.length; i++) {
                    String[] kv = p[i].split("=");
                    if (kv[1].startsWith("@")) o.referencias.put(kv[0], kv[1].substring(1));
                    else o.datos.put(kv[0], kv[1]);
                }
                almacen.put(o.id, o);
            } else if (p[0].equals("activar")) { // activar ID profundidad
                activados.clear();
                System.out.println("== activar " + p[1] + " con profundidad " + p[2]);
                activar(p[1], Integer.parseInt(p[2]), "", "");
                System.out.println("Objetos leídos del disco: " + activados.size());
            }
        }
    }
}
```

Ejemplo: Profundidad 1

ENTRADA

```
objeto C1 Curso nombre=1DAM tutor=@P1 delegado=@A1
objeto P1 Profesor nombre=Ana departamento=@D1
objeto D1 Departamento nombre=Informática jefe=@P1
objeto A1 Alumno nombre=Luis grupo=@C1 mentor=@A9
activar C1 1
```

SALIDA ESPERADA

```
== activar C1 con profundidad 1
Curso C1 {nombre=1DAM}
  tutor → Profesor P1 (sin activar)
  delegado → Alumno A1 (sin activar)
Objetos leídos del disco: 1
```

Ejemplo: Profundidad 3 con ciclo

ENTRADA

```
objeto C1 Curso nombre=1DAM tutor=@P1 delegado=@A1
objeto P1 Profesor nombre=Ana departamento=@D1
objeto D1 Departamento nombre=Informática jefe=@P1
objeto A1 Alumno nombre=Luis grupo=@C1 mentor=@A9
activar C1 3
```

SALIDA ESPERADA

```
== activar C1 con profundidad 3
Curso C1 {nombre=1DAM}
  tutor → Profesor P1 {nombre=Ana}
    departamento → Departamento D1
      {nombre=Informática}
        jefe → Profesor P1 (sin activar)
          delegado → Alumno A1 {nombre=Luis}
            grupo → Curso C1 {nombre=1DAM}
              tutor → Profesor P1 (sin activar)
                delegado → Alumno A1 (sin activar)
                  mentor → null (A9 no existe)
Objetos leídos del disco: 4
```

82. Herencia de tablas en PostgreSQL

Muy difícil · BD Objeto-Relacionales y Orientadas a Objetos · apuntesdam.com/subject/acceso-datos/topic/bd-objeto-relacionales

PostgreSQL permite que una tabla herede de otra (CREATE TABLE alumno (curso text) INHERITS (persona)): la hija tiene las columnas del padre más las suyas y, al consultar el padre, también salen las filas de las hijas. Pero las restricciones UNIQUE (y las claves primarias) no se heredan entre tablas: cada tabla solo comprueba sus propias filas. Simúlalo.

Órdenes: «tabla t columnas=a,b [hereda=padre] [unico=col]» (muestra «CREATE TABLE t (todas las columnas) [INHERITS (padre)] [UNIQUE (col)]»; las columnas heredadas van primero); «insertar t a=1,b=2» (columnas no indicadas a NULL; si una no existe, «ERROR: la columna x no existe en t»; si viola el único de esa tabla, comparando solo con las filas de esa misma tabla, «ERROR: valor duplicado col=v en t»; si no, «INSERT en t»); «consultar t» y «consultar solo t» (como SELECT * FROM t y SELECT * FROM ONLY t). La consulta muestra «SELECT * FROM [ONLY] t (col | col ...)», cada fila con dos espacios delante y sus valores para las columnas de t separados por « | » (NULL si no tiene), añadiendo « [hija]» a las filas que vienen de otra tabla, y al final « n filas» («1 fila»). Primero van las filas propias y después, sin ONLY, las de cada hija en orden de creación (y las de sus hijas, recursivamente).

Código de partida (java)

```
import java.util.*;

public class Main {
    static class Tabla {
        String nombre, padre, unico;
        List<String> columnas = new ArrayList<>();
        List<Map<String, String>> filas = new ArrayList<>();
        List<String> hijas = new ArrayList<>();
    }

    static Map<String, Tabla> tablas = new LinkedHashMap<>();

    // TODO: static void recoger(Tabla consultada, Tabla t, boolean solo, List<String> salida)

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (l.isEmpty()) continue;
            String[] p = l.split(" ");
            // TODO: tabla, insertar, consultar y consultar solo
        }
    }
}
```

Ejemplo: Personas, alumnos y profesores

ENTRADA

```
tabla persona columnas=dni,nombre unico=dni
tabla alumno columnas=curso hereda=persona
unico=dni
tabla profesor columnas=departamento
hereda=persona
tabla delegado columnas=desde hereda=alumno
insertar persona dni=111,nombre=Conserje
insertar alumno dni=222,nombre=Ana,curso=1DAM
insertar alumno dni=222,nombre=Otra,curso=2DAM
insertar alumno dni=111,nombre=Luis,curso=1DAM
insertar profesor
dni=333,nombre=Marta,departamento=Informática
insertar delegado
dni=444,nombre=Eva,curso=2DAM,desde=2025
insertar profesor dni=555,nombre=Pepe,curso=1DAM
insertar persona dni=111,nombre=Duplicado
consultar persona
consultar solo persona
consultar alumno
```

SALIDA ESPERADA

```
CREATE TABLE persona (dni, nombre) UNIQUE (dni)
CREATE TABLE alumno (dni, nombre, curso) INHERITS
(persona) UNIQUE (dni)
CREATE TABLE profesor (dni, nombre, departamento)
INHERITS (persona)
CREATE TABLE delegado (dni, nombre, curso, desde)
INHERITS (alumno)
INSERT en persona
INSERT en alumno
ERROR: valor duplicado dni=222 en alumno
INSERT en alumno
INSERT en profesor
INSERT en delegado
ERROR: la columna curso no existe en profesor
ERROR: valor duplicado dni=111 en persona
SELECT * FROM persona (dni | nombre)
  111 | Conserje
  222 | Ana    [alumno]
  111 | Luis   [alumno]
  444 | Eva    [delegado]
  333 | Marta  [profesor]
  5 filas
SELECT * FROM ONLY persona (dni | nombre)
  111 | Conserje
  1 fila
SELECT * FROM alumno (dni | nombre | curso)
  222 | Ana | 1DAM
  111 | Luis | 1DAM
  444 | Eva | 2DAM [delegado]
  3 filas
```

83. Propiedad ligada (bound property)

Fácil · Componentes de Acceso a Datos · apuntesdam.com/subject/acceso-datos/topic/componentes-acceso-datos

Completa el JavaBean Termostato para que su propiedad temperatura sea ligada: cada vez que cambie de valor debe avisar a los oyentes con `firePropertyChange`. El programa lee temperaturas por teclado y los oyentes imprimen cada cambio (si el valor no cambia, no se debe notificar).

Código de partida (java)

```
import java.beans.*;
import java.io.Serializable;
import java.util.Scanner;

public class Main {

    public static class Termostato implements Serializable {
        private final PropertyChangeSupport soporte = new PropertyChangeSupport(this);
        private int temperatura = 20;

        public Termostato() {}

        public int getTemperatura() { return temperatura; }

        public void setTemperatura(int nueva) {
            // TODO: asigna el nuevo valor y avisa a los oyentes del cambio
            temperatura = nueva;
        }

        public void addPropertyChangeListener(PropertyChangeListener l) {
            soporte.addPropertyChangeListener(l); }
        public void removePropertyChangeListener(PropertyChangeListener l) {
            soporte.removePropertyChangeListener(l); }
    }

    public static void main(String[] args) {
        Termostato t = new Termostato();
        t.addPropertyChangeListener(e ->
            System.out.println(e.getPropertyName() + ": " + e.getOldValue() + " -> " + e.getNewValue()));

        Scanner sc = new Scanner(System.in);
        while (sc.hasNextInt()) t.setTemperatura(sc.nextInt());
        System.out.println("Final: " + t.getTemperatura());
    }
}
```

Ejemplo: Tres cambios

ENTRADA

22 25 18

SALIDA ESPERADA

temperatura: 20 -> 22
temperatura: 22 -> 25
temperatura: 25 -> 18
Final: 18

Ejemplo: Valor repetido: no se notifica

ENTRADA

20 21 21

SALIDA ESPERADA

temperatura: 20 -> 21
Final: 21

84. Propiedad restringida (constrained)

Medio · Componentes de Acceso a Datos · apuntesdam.com/subject/acceso-datos/topic/componentes-acceso-datos

El componente CuentaBancaria tiene una propiedad saldo restringida: antes de cambiarla pregunta a los oyentes con `fireVetoableChange`, y cualquiera puede vetar el cambio lanzando `PropertyVetoException`. Completa `setSaldo`. El programa registra un oyente que veta saldos negativos y aplica los movimientos leídos por teclado.

Código de partida (java)

```
import java.beans.*;
import java.util.Scanner;

public class Main {

    public static class CuentaBancaria {
        private final VetoableChangeSupport vetos = new VetoableChangeSupport(this);
        private double saldo = 100;

        public double getSaldo() { return saldo; }

        public void setSaldo(double nuevo) throws PropertyVetoException {
            // TODO: pregunta a los oyentes y, si nadie veta, cambia el saldo
            saldo = nuevo;
        }

        public void addVetoableChangeListener(VetoableChangeListener l) {
            vetos.addVetoableChangeListener(l); }
    }

    public static void main(String[] args) {
        CuentaBancaria cuenta = new CuentaBancaria();
        cuenta.addVetoableChangeListener(e -> {
            if ((double) e.getNewValue() < 0) throw new PropertyVetoException("saldo negativo", e);
        });

        Scanner sc = new Scanner(System.in);
        while (sc.hasNextInt()) {
            int movimiento = sc.nextInt();
            try {
                cuenta.setSaldo(cuenta.getSaldo() + movimiento);
                System.out.println("OK " + movimiento + " -> " + cuenta.getSaldo());
            } catch (PropertyVetoException ex) {
                System.out.println("Vetado " + movimiento + ": " + ex.getMessage());
            }
        }
    }
}
```

Ejemplo: Ingreso y retirada válidos

ENTRADA

50 -120

SALIDA ESPERADA

OK 50 -> 150.0
OK -120 -> 30.0

Ejemplo: Retirada que deja saldo negativo

ENTRADA

-150 20

SALIDA ESPERADA

Vetado -150: saldo negativo
OK 20 -> 120.0

85. Persistencia de un JavaBean en XML

Medio · Componentes de Acceso a Datos · apuntesdam.com/subject/acceso-datos/topic/componentes-acceso-datos

Los JavaBeans pueden guardarse en XML con XMLEncoder y recuperarse con XMLDecoder, siempre que sigan las convenciones (constructor vacío, getters y setters). Completa guardar() y cargar() para el bean Configuración. El programa lee el tema y el tamaño de letra por teclado, guarda el bean en config.xml, lo vuelve a cargar y muestra sus valores.

Código de partida (java)

```
import java.beans.*;
import java.io.*;
import java.util.Scanner;

public class Main {

    public static class Configuracion {
        private String tema = "claro";
        private int tamanoLetra = 14;

        public Configuracion() {}
        public String getTema() { return tema; }
        public void setTema(String tema) { this.tema = tema; }
        public int getTamanoLetra() { return tamanoLetra; }
        public void setTamanoLetra(int t) { this.tamanoLetra = t; }
    }

    static void guardar(Configuracion c, File fichero) throws IOException {
        // TODO: guarda el bean en XML con XMLEncoder
    }

    static Configuracion cargar(File fichero) throws IOException {
        // TODO: recupera el bean con XMLDecoder
        return new Configuracion();
    }

    public static void main(String[] args) throws IOException {
        Scanner sc = new Scanner(System.in);
        Configuracion c = new Configuracion();
        c.setTema(sc.next());
        c.setTamanoLetra(sc.nextInt());

        File f = new File("config.xml");
        guardar(c, f);
        Configuracion leida = cargar(f);
        System.out.println(leida.getTema() + " " + leida.getTamanoLetra());
        System.out.println("XML generado: " + (f.length() > 0));
    }
}
```

Ejemplo: Tema oscuro, letra 16

ENTRADA

oscuro 16

SALIDA ESPERADA

oscuro 16
XML generado: true

86. Componente DAO con eventos

Difícil · Componentes de Acceso a Datos · apuntesdam.com/subject/acceso-datos/topic/componentes-acceso-datos

Implementa AlumnoDaoMemoria, un componente de acceso a datos que cumple la interfaz AlumnoDao y además avisa a sus oyentes (DaoListener) cada vez que se guarda o se borra un alumno. El programa ejecuta las órdenes leídas por teclado: 'G nombre nota' guarda, 'B nombre' borra y 'L' lista los alumnos ordenados por nombre.

Código de partida (java)

```
import java.util.*;

public class Main {

    public interface DaoListener { void datosCambiados(String accion, String nombre); }

    public interface AlumnoDao {
        void guardar(String nombre, double nota);
        boolean borrar(String nombre);
        Map<String, Double> listar();
        void addDaoListener(DaoListener l);
    }

    public static class AlumnoDaoMemoria implements AlumnoDao {
        private final Map<String, Double> alumnos = new TreeMap<>();
        private final List<DaoListener> oyentes = new ArrayList<>();

        public void guardar(String nombre, double nota) {
            // TODO: guarda y avisa a los oyentes
        }

        public boolean borrar(String nombre) {
            // TODO: borra y avisa a los oyentes si existía
            return false;
        }

        public Map<String, Double> listar() { return Collections.unmodifiableMap(alumnos); }

        public void addDaoListener(DaoListener l) { oyentes.add(l); }
    }

    public static void main(String[] args) {
        AlumnoDao dao = new AlumnoDaoMemoria();
        dao.addDaoListener((accion, nombre) -> System.out.println("[evento] " + accion + " " + nombre));

        Scanner sc = new Scanner(System.in);
        while (sc.hasNext()) {
            String orden = sc.next();
            switch (orden) {
                case "G" -> dao.guardar(sc.next(), Double.parseDouble(sc.next()));
                case "B" -> { String n = sc.next(); if (!dao.borrar(n)) System.out.println("No existe " + n); }
                case "L" -> dao.listar().forEach((n, nota) -> System.out.println(n + " " + nota));
                default -> System.out.println("Orden desconocida");
            }
        }
    }
}
```

Ejemplo: Guardar y listar

ENTRADA

G Luis 6 G Ana 8.5 L

SALIDA ESPERADA

[evento] guardado Luis
[evento] guardado Ana
Ana 8.5
Luis 6.0

Ejemplo: Borrar existente e inexistente

ENTRADA

G Ana 7 B Ana B Pepe L

SALIDA ESPERADA

[evento] guardado Ana
[evento] borrado Ana
No existe Pepe

87. Un contenedor de inyección de dependencias

Muy difícil · Componentes de Acceso a Datos · apuntesdam.com/subject/acceso-datos/topic/componentes-acceso-datos

Spring, Jakarta CDI o Micronaut son contenedores de componentes: el programa declara qué componentes hay y de qué dependen, y el contenedor los crea en el orden correcto y los inyecta. Simula uno. «componente N [depende=A,B] [ambito=prototype]» declara un componente (singleton por defecto: una sola instancia; prototype: una nueva cada vez). «obtener N» muestra «obtener N» y antes de crear nada valida el grafo recorriendo las dependencias en orden: si falta un componente, « Error: no existe el componente N» (el pedido) o « Error: falta el componente X (lo necesita Y)»; si hay un ciclo, « Error: dependencia circular A → B → C → A» (desde la primera aparición). Si es válido, crea primero las dependencias (en orden, recursivamente) y después el componente, numerando las instancias globalmente desde 1: « crear X#k» (con « (prototype)» si lo es); los singletons ya creados se reutilizan sin mostrar nada. Termina con «→ N#k» y « (ya existía)» si era un singleton ya creado. «cerrar» destruye los singletons en orden inverso a su creación (« destruir X#k»; los prototype no los gestiona el contenedor), los olvida y muestra «Contenedor cerrado».

Código de partida (java)

```
import java.util.*;

public class Main {
    static Map<String, List<String>> dependencias = new LinkedHashMap<>();
    static Map<String, String> ambitos = new HashMap<>();
    static Map<String, Integer> singletons = new HashMap<>(); // componente → número de instancia
    static List<String> creados = new ArrayList<>();           // singletons en orden de creación (para cerrar)
    static int contador = 0;

    // TODO: static String validar(String nombre, List<String> camino, String quien)
    //         static int resolver(String nombre)

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");
            switch (p[0]) {
                case "componente" -> { // componente N [depende=A,B] [ambito=prototype]
                    List<String> deps = new ArrayList<>();
                    String ambito = "singleton";
                    for (int i = 2; i < p.length; i++) {
                        if (p[i].startsWith("depende=")) deps.addAll(List.of(p[i].substring(8).split(",")));
                        if (p[i].startsWith("ambito=")) ambito = p[i].substring(7);
                    }
                    dependencias.put(p[1], deps);
                    ambitos.put(p[1], ambito);
                }
                case "obtener" -> {
                    System.out.println("obtener " + p[1]);
                    // TODO: validar y, si no hay error, resolver
                }
                case "cerrar" -> {
                    // TODO: destruir los singletons en orden inverso al de creación
                    System.out.println("Contenedor cerrado");
                }
            }
        }
    }
}
```


Ejemplo: Tienda

ENTRADA

```
componente DataSource
componente PedidoDao depende=DataSource
componente ClienteDao depende=DataSource
componente Correo ambito=prototype
componente Notificador depende=Correo
componente PedidoService
depende=PedidoDao,ClienteDao,Notificador
componente Informe depende=PedidoDao,Correo
ambito=prototype
obtener PedidoService
obtener PedidoService
obtener Informe
obtener Informe
obtener Factura
cerrar
obtener ClienteDao
```

SALIDA ESPERADA

```
obtener PedidoService
  crear DataSource#1
  crear PedidoDao#2
  crear ClienteDao#3
  crear Correo#4 (prototype)
  crear Notificador#5
  crear PedidoService#6
→ PedidoService#6
obtener PedidoService
→ PedidoService#6 (ya existía)
obtener Informe
  crear Correo#7 (prototype)
  crear Informe#8 (prototype)
→ Informe#8
obtener Informe
  crear Correo#9 (prototype)
  crear Informe#10 (prototype)
→ Informe#10
obtener Factura
Error: no existe el componente Factura
destruir PedidoService#6
destruir Notificador#5
destruir ClienteDao#3
destruir PedidoDao#2
destruir DataSource#1
Contenedor cerrado
obtener ClienteDao
  crear DataSource#11
  crear ClienteDao#12
→ ClienteDao#12
```

88. Filtrar un fichero de log

Fácil · Ejercicios de Acceso a Datos · apuntesdam.com/subject/acceso-datos/topic/ejercicios-acceso-datos

El programa guarda en app.log las líneas recibidas por teclado. Completa filtrar() para que lea app.log y escriba en errores.log solo las líneas que contienen "ERROR", devolviendo cuántas ha copiado. Después se muestra el número y el contenido de errores.log.

Código de partida (java)

```
import java.io.*;
import java.nio.file.*;
import java.util.*;

public class Main {

    static int filtrar(Path origen, Path destino) throws IOException {
        // TODO: copia a destino las líneas de origen que contienen "ERROR"
        return 0;
    }

    public static void main(String[] args) throws IOException {
        Path log = Path.of("app.log");
        Files.write(log, new String(System.in.readAllBytes()).lines().toList());

        Path errores = Path.of("errores.log");
        int n = filtrar(log, errores);
        System.out.println("Errores: " + n);
        if (Files.exists(errores)) Files.readAllLines(errores).forEach(System.out::println);
    }
}
```

Ejemplo: Dos errores

ENTRADA

```
10:00 INFO arranque
10:01 ERROR conexión perdida
10:02 WARN reintento
10:03 ERROR tiempo agotado
```

SALIDA ESPERADA

```
Errores: 2
10:01 ERROR conexión perdida
10:03 ERROR tiempo agotado
```

Ejemplo: Sin errores

ENTRADA

```
INFO todo bien
INFO fin
```

SALIDA ESPERADA

```
Errores: 0
```

89. Conversor de CSV a JSON

Medio · Ejercicios de Acceso a Datos · apuntesdam.com/subject/acceso-datos/topic/ejercicios-acceso-datos

Convierte un fichero CSV con cabecera (nombre;ciudad;edad) en un array JSON, sin librerías externas. El programa guarda la entrada en personas.csv; completa convertir() para que devuelva el JSON con el formato exacto del ejemplo: [{"nombre":"Ana","ciudad":"Logroño","edad":20}, ...] (la edad sin comillas, porque es un número).

Código de partida (java)

```
import java.io.*;
import java.nio.file.*;
import java.util.*;

public class Main {

    static String convertir(Path csv) throws IOException {
        List<String> lineas = Files.readAllLines(csv);
        String[] cabecera = lineas.get(0).split(";");
        List<String> objetos = new ArrayList<>();
        // TODO: recorre las líneas de datos y construye un objeto JSON por línea

        return "[" + String.join(",", objetos) + "]";
    }

    public static void main(String[] args) throws IOException {
        Path csv = Path.of("personas.csv");
        Files.write(csv, new String(System.in.readAllBytes()).lines().toList());
        System.out.println(convertir(csv));
    }
}
```

Ejemplo: Dos personas

ENTRADA

```
nombre;ciudad;edad
Ana;Logroño;20
Luis;Arnedo;22
```

SALIDA ESPERADA

```
[{"nombre":"Ana","ciudad":"Logroño","edad":20},
{"nombre":"Luis","ciudad":"Arnedo","edad":22}]
```

Ejemplo: Solo cabecera

ENTRADA

```
nombre;ciudad;edad
```

SALIDA ESPERADA

```
[]
```

90. Media de precios por categoría (DOM)

Difícil · Ejercicios de Acceso a Datos · apuntesdam.com/subject/acceso-datos/topic/ejercicios-acceso-datos

El programa guarda el XML recibido en catalogo.xml. Cada <producto> tiene un atributo categoria y un elemento <precio>. Completa medias() para devolver un mapa ordenado categoría → precio medio. Se imprime cada categoría con su media con dos decimales.

Código de partida (java)

```
import java.io.*;
import java.nio.file.*;
import java.util.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;

public class Main {

    static Map<String, Double> medias(File fichero) throws Exception {
        Map<String, Double> resultado = new TreeMap<>();
        // TODO: calcula el precio medio de cada categoría

        return resultado;
    }

    public static void main(String[] args) throws Exception {
        File f = new File("catalogo.xml");
        Files.write(f.toPath(), System.in.readAllBytes());
        medias(f).forEach((cat, media) -> System.out.printf(Locale.US, "%s %.2f%n", cat, media));
    }
}
```

Ejemplo: Dos categorías

ENTRADA

```
<catalogo><producto categoria="audio">
<precio>40</precio></producto><producto
categoria="video"><precio>100</precio></producto>
<producto categoria="audio"><precio>50</precio>
</producto></catalogo>
```

SALIDA ESPERADA

```
audio 45.00
video 100.00
```

Desarrollo de Interfaces

91. La validación de un formulario de registro

Medio · Generación de Interfaces Gráficas · apuntesdam.com/subject/desarrollo-interfaces/topic/generacion-interfaces

Antes de habilitar el botón «Registrar», una interfaz valida cada campo y muestra un mensaje junto a él. Cada línea de la entrada es «campo=valor» (nombre, email, edad, clave y repetir; puede faltar alguno, y entonces vale vacío). Comprueba, en este orden: nombre obligatorio («nombre: es obligatorio») y de al menos 2 caracteres («nombre: debe tener al menos 2 caracteres»); email con formato algo@algo.dominio, con al menos 2 letras en el dominio («email: formato no válido»); edad entera («edad: debe ser un número entero») entre 18 y 120 («edad: debe estar entre 18 y 120»); clave con al menos 8 caracteres, una mayúscula y un número («clave: necesita 8 caracteres, una mayúscula, un número», con solo lo que falte); y repetir igual a clave («repetir: las contraseñas no coinciden»). Muestra cada error y al final «N errores: botón Registrar desactivado» («1 error» en singular), o «Formulario válido: botón Registrar activado».

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, String> campos = new LinkedHashMap<>();
        while (sc.hasNextLine()) {
            String linea = sc.nextLine();
            int i = linea.indexOf('=');
            if (i > 0) campos.put(linea.substring(0, i).trim(), linea.substring(i + 1).trim());
        }
        // TODO: valida cada campo y muestra los errores como lo haría el formulario
    }
}
```

Ejemplo: Formulario correcto

ENTRADA

```
nombre=Lucía
email=lucia@correo.es
edad=19
clave=Segura123
repetir=Segura123
```

SALIDA ESPERADA

Formulario válido: botón Registrar activado

Ejemplo: Varios errores

ENTRADA

```
nombre=L
email=lucia@correo
edad=17
clave=abc
repetir=abd
```

SALIDA ESPERADA

```
nombre: debe tener al menos 2 caracteres
email: formato no válido
edad: debe estar entre 18 y 120
clave: necesita 8 caracteres, una mayúscula, un
número
repetir: las contraseñas no coinciden
5 errores: botón Registrar desactivado
```

92. Repartir el ancho como un GridPane

Difícil · Generación de Interfaces Gráficas · apuntesdam.com/subject/desarrollo-interfaces/topic/generacion-interfaces

Un contenedor de rejilla (GridPane de JavaFX, GridBagLayout de Swing o grid de CSS) reparte su ancho entre columnas. La entrada es «ancho W», «hueco H» (separación entre columnas) y una línea por columna: «nombre fija N» (N píxeles), «nombre porcentaje P» (P % del espacio disponible) o «nombre crece» (se reparte a partes iguales lo que sobra). El espacio disponible es $W - H \times (\text{columnas} - 1)$. Muestra para cada columna «nombre: x=X ancho=A» (1 decimal, x empieza en 0 y entre columnas se suma el hueco). Si no hay columnas que crezcan y sobra espacio, añade «Sobran S px a la derecha». Si lo fijo y los porcentajes ya superan el disponible, o no queda nada para las columnas que crecen, muestra solo «No cabe: faltan F px» (lo que excede, más 1 px por cada columna que crece, que necesita al menos uno).

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int ancho = Integer.parseInt(sc.nextLine().split(" ")[1]);
        int hueco = Integer.parseInt(sc.nextLine().split(" ")[1]);
        List<String[]> columnas = new ArrayList<>(); // {nombre, tipo, valor}
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (!l.isEmpty()) columnas.add(l.split(" "));
        }
        // TODO: calcula la posición y el ancho de cada columna como un GridPane
    }
}
```

Ejemplo: Formulario de tres columnas

ENTRADA

```
ancho 800
hueco 10
etiqueta fija 120
campo crece
ayuda porcentaje 25
```

SALIDA ESPERADA

```
etiqueta: x=0.0 ancho=120.0
campo: x=130.0 ancho=465.0
ayuda: x=605.0 ancho=195.0
```

Ejemplo: Dos que crecen

ENTRADA

```
ancho 1000
hueco 20
menu fija 200
contenido crece
lateral crece
```

SALIDA ESPERADA

```
menu: x=0.0 ancho=200.0
contenido: x=220.0 ancho=380.0
lateral: x=620.0 ancho=380.0
```

Ejemplo: Sobra espacio

ENTRADA

```
ancho 600
hueco 0
a fija 100
b porcentaje 50
```

SALIDA ESPERADA

```
a: x=0.0 ancho=100.0
b: x=100.0 ancho=300.0
Sobran 200.0 px a la derecha
```

93. Captura y burbuja de un evento

Muy difícil · Generación de Interfaces Gráficas · apuntesdam.com/subject/desarrollo-interfaces/topic/generacion-interfaces

En JavaFX (y en el DOM), un evento recorre la jerarquía de nodos en dos fases: en la de captura se ejecutan los filtros (addEventFilter) desde la raíz hasta el nodo objetivo, y en la de burbuja los manejadores (addEventHandler / setOnAction) desde el objetivo hasta la raíz. Si alguno consume el evento (event.consume()), los demás nodos ya no lo reciben, aunque se terminan de ejecutar los demás del mismo nodo. Las líneas son «nodo HIJO PADRE», «filtro NODO [consume]», «manejador NODO [consume]» (en el orden en que se registraron) y «clic NODO» (el objetivo). Muestra «Ruta: Raíz > ... > Objetivo», cada filtro o manejador ejecutado con «filtro de NODO» o «manejador de NODO» (y «consume el evento») si lo consume) y al final «Evento consumido» o «El evento llegó a la raíz sin consumirse».

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, String> padre = new HashMap<>();
        List<String[]> manejadores = new ArrayList<>(); // {tipo (filtro|manejador), nodo, consume?}
        String objetivo = null;
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");
            switch (p[0]) {
                case "nodo" -> padre.put(p[1], p[2]); // nodo HIJO PADRE
                case "filtro", "manejador" -> manejadores.add(p);
                case "clic" -> objetivo = p[1];
                default -> { }
            }
        }
        // TODO: simula la cadena de despacho del evento (captura y burbuja)
    }
}
```

Ejemplo: Sin consumir

ENTRADA

```
nodo VBox Ventana
nodo HBox VBox
nodo Boton HBox
nodo Etiqueta HBox
manejador Boton
filtro VBox
manejador Ventana
manejador HBox
clic Boton
```

SALIDA ESPERADA

```
Ruta: Ventana > VBox > HBox > Boton
filtro de VBox
manejador de Boton
manejador de HBox
manejador de Ventana
El evento llegó a la raíz sin consumirse
```

Ejemplo: Un filtro consume

ENTRADA

```
nodo VBox Ventana
nodo HBox VBox
nodo Boton HBox
nodo Etiqueta HBox
filtro Ventana
filtro HBox consume
filtro Boton
manejador Boton
clic Boton
```

SALIDA ESPERADA

```
Ruta: Ventana > VBox > HBox > Boton
filtro de Ventana
filtro de HBox (consume el evento)
Evento consumido
```

94. Un intérprete de órdenes de voz

Medio · Interfaces Naturales de Usuario · apuntesdam.com/subject/desarrollo-interfaces/topic/interfaces-naturales

El reconocedor de voz (SpeechRecognizer en Android) devuelve texto; la aplicación debe extraer la intención y sus datos. Cada línea es una transcripción. Normalízala (la función normalizar quita tildes, signos y espacios de más) y reconoce, en este orden: una alarma («pon/poner/pone/programa [una] alarma a las H[:MM] [y media|y cuarto]») → «ALARMA HH:MM» con dos cifras, o «ERROR: hora no válida» si la hora pasa de 23 o los minutos de 59; una llamada («llama/llamar a NOMBRE») → «LLAMAR Nombre Apellido» con la inicial de cada palabra en mayúscula (sin tildes, porque se han quitado al normalizar); el volumen («sube/baja [el] volumen [N]», 10 si no se dice) → «VOLUMEN +N» o «VOLUMEN -N»; abrir una aplicación («abre/abrir/inicia [la app|la aplicacion] NOMBRE») → «ABRIR nombre». Si no encaja nada, «NO ENTENDIDO: frase original».

Código de partida (java)

```
import java.text.Normalizer;
import java.util.*;
import java.util.regex.*;

public class Main {
    static String normalizar(String s) {
        String sinTildes = Normalizer.normalize(s.toLowerCase(), Normalizer.Form.NFD).replaceAll("\\p{M}", "");
        return sinTildes.replaceAll("[^a-z0-9: ]", " ").replaceAll("\\s+", " ").trim();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String frase = sc.nextLine();
            if (frase.isBlank()) continue;
            String t = normalizar(frase);
            // TODO: reconoce la intención y sus datos
        }
    }
}
```

Ejemplo: Órdenes habituales

ENTRADA

Pon una alarma a las 7:30
Llama a mamá
Sube el volumen
Abre Spotify, por favor
¿Qué tiempo hace mañana?

SALIDA ESPERADA

ALARMA 07:30
LLAMAR Mama
VOLUMEN +10
ABRIR spotify
NO ENTENDIDO: ¿Qué tiempo hace mañana?

Ejemplo: Variantes

ENTRADA

Oye, programa alarma a las 6 y media
baja volumen 25
Poner alarma a las 25
llamar a Juan Carlos

SALIDA ESPERADA

ALARMA 06:30
VOLUMEN -25
ERROR: hora no válida
LLAMAR Juan Carlos

95. Clasificar gestos táctiles

Difícil · Interfaces Naturales de Usuario · apuntesdam.com/subject/desarrollo-interfaces/topic/interfaces-naturales

Un detector de gestos (GestureDetector en Android, o los eventos de puntero en JavaFX y en la web) decide qué ha hecho el dedo a partir de sus posiciones. Cada gesto es un bloque de líneas «tiempo_ms x y» (de que el dedo toca a que se levanta), y los gestos se separan con una línea en blanco. Con el primer y el último punto calcula la distancia (Math.hypot), la duración y la velocidad (px/ms). Clasifica: si la distancia es menor que 10 px, «TOQUE» (menos de 300 ms), «PULSACIÓN LARGA» (500 ms o más) o «INDECISO» (entre medias); si es de al menos 50 px y la velocidad de al menos 0.3 px/ms, un deslizamiento en la dirección del eje dominante: «DESLIZAR DERECHA/IZQUIERDA» si $|dx| \geq |dy|$ o «DESLIZAR ABAJO/ARRIBA» (la y crece hacia abajo en la pantalla); en otro caso, «ARRASTRE». Muestra «Gesto N: TIPO (D px en T ms, V px/ms)» con D y T sin decimales y V con 2.

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<List<double[]>> gestos = new ArrayList<>();
        List<double[]> actual = new ArrayList<>();
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (l.isEmpty()) { if (!actual.isEmpty()) { gestos.add(actual); actual = new ArrayList<>(); }
                continue; }
            String[] p = l.split(" ");
            actual.add(new double[]{Double.parseDouble(p[0]), Double.parseDouble(p[1]),
                Double.parseDouble(p[2])});
        }
        if (!actual.isEmpty()) gestos.add(actual);
        // TODO: clasifica cada gesto (toque, pulsación larga, deslizamiento o arrastre)
    }
}
```

Ejemplo: Gestos básicos

ENTRADA

0 100 200
80 102 201

0 50 300
120 200 305
200 320 310

0 300 300
700 303 302

SALIDA ESPERADA

Gesto 1: TOQUE (2 px en 80 ms, 0.03 px/ms)
Gesto 2: DESLIZAR DERECHA (270 px en 200 ms, 1.35 px/ms)
Gesto 3: PULSACIÓN LARGA (4 px en 700 ms, 0.01 px/ms)

Ejemplo: Vertical y lento

ENTRADA

```
0 200 600
150 205 300

0 100 100
1000 260 100
```

```
0 10 10
400 12 13
```

SALIDA ESPERADA

```
Gesto 1: DESLIZAR ARRIBA (300 px en 150 ms, 2.00
px/ms)
Gesto 2: ARRASTRE (160 px en 1000 ms, 0.16 px/ms)
Gesto 3: INDECISO (4 px en 400 ms, 0.01 px/ms)
```

96. ¿Qué gesto hace la mano?

Muy difícil · Interfaces Naturales de Usuario · apuntesdam.com/subject/desarrollo-interfaces/topic/interfaces-naturales

Los detectores de manos (MediaPipe Hands, ML Kit) devuelven las coordenadas de 21 puntos de la mano. Aquí recibes 11 por mano, en coordenadas normalizadas (de 0 a 1, la y hacia abajo): «muñeca x y» y, para cada dedo (pulgar, índice, corazón, anular, menique), su punta («dedo_punta x y») y su articulación media («dedo_medio x y»). Las manos se separan con «---». Un dedo está extendido si su punta está más lejos de la muñeca que su articulación media multiplicada por 1.15 (1.1 en el pulgar): así no importa cómo esté girada la mano. Hay PINZA si la distancia entre las puntas del pulgar y del índice es menor que 0.25 veces la distancia de la muñeca a la articulación media del índice. Gestos, en este orden: PINZA, PUÑO (ningún dedo extendido), MANO ABIERTA (los cinco), PULGAR (solo el pulgar), VICTORIA (solo índice y corazón), SEÑALAR (solo el índice) o DESCONOCIDO. Muestra «Mano N: GESTO (extendidos: dedo1, dedo2...)», con los dedos en el orden del array DEDOS o «ninguno».

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<Map<String, double[]>> manos = new ArrayList<>();
        Map<String, double[]> actual = new LinkedHashMap<>();
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (l.equals("---")) { manos.add(actual); actual = new LinkedHashMap<>(); continue; }
            if (l.isEmpty()) continue;
            String[] p = l.split(" ");
            // punto x y (coordenadas normalizadas de 0 a 1)
            actual.put(p[0], new double[]{Double.parseDouble(p[1]), Double.parseDouble(p[2])});
        }
        if (!actual.isEmpty()) manos.add(actual);
        String[] DEDOS = {"pulgar", "indice", "corazon", "anular", "menique"};
        // TODO: decide qué dedos están extendidos y qué gesto hace cada mano
    }
}
```


Ejemplo: Gestos básicos

ENTRADA

```
muneca 0.500 0.850
pulgar_medio 0.344 0.760
pulgar_punta 0.370 0.775
indice_medio 0.435 0.609
indice_punta 0.448 0.657
corazon_medio 0.500 0.600
corazon_punta 0.500 0.650
anular_medio 0.565 0.609
anular_punta 0.552 0.657
menique_medio 0.625 0.633
menique_punta 0.600 0.677
---
muneca 0.500 0.850
pulgar_medio 0.344 0.760
pulgar_punta 0.240 0.700
indice_medio 0.435 0.609
indice_punta 0.391 0.444
corazon_medio 0.500 0.600
corazon_punta 0.500 0.430
anular_medio 0.565 0.609
anular_punta 0.609 0.444
menique_medio 0.625 0.633
menique_punta 0.710 0.486
---
muneca 0.500 0.850
pulgar_medio 0.344 0.760
pulgar_punta 0.370 0.775
indice_medio 0.435 0.609
indice_punta 0.391 0.444
corazon_medio 0.500 0.600
corazon_punta 0.500 0.430
anular_medio 0.565 0.609
anular_punta 0.552 0.657
menique_medio 0.625 0.633
menique_punta 0.600 0.677
---
muneca 0.500 0.850
pulgar_medio 0.344 0.760
pulgar_punta 0.370 0.775
indice_medio 0.435 0.609
indice_punta 0.391 0.444
corazon_medio 0.500 0.600
corazon_punta 0.500 0.650
anular_medio 0.565 0.609
anular_punta 0.552 0.657
menique_medio 0.625 0.633
menique_punta 0.600 0.677
```

SALIDA ESPERADA

```
Mano 1: PUÑO (extendidos: ninguno)
Mano 2: MANO ABIERTA (extendidos: pulgar, indice,
corazon, anular, menique)
Mano 3: VICTORIA (extendidos: indice, corazon)
Mano 4: SEÑALAR (extendidos: indice)
```

Ejemplo: Mano girada y pinza

ENTRADA

```
muneca 0.500 0.850
pulgar_medio 0.554 0.756
pulgar_punta 0.545 0.772
indice_medio 0.645 0.811
indice_punta 0.743 0.785
corazon_medio 0.650 0.850
corazon_punta 0.752 0.850
anular_medio 0.645 0.889
anular_punta 0.616 0.881
menique_medio 0.630 0.925
menique_punta 0.604 0.910
---
muneca 0.500 0.850
pulgar_medio 0.326 0.897
pulgar_punta 0.210 0.928
indice_medio 0.283 0.725
indice_punta 0.327 0.750
corazon_medio 0.323 0.673
corazon_punta 0.359 0.709
anular_medio 0.375 0.633
anular_punta 0.400 0.677
menique_medio 0.435 0.609
menique_punta 0.448 0.657
---
muneca 0.500 0.850
pulgar_medio 0.344 0.760
pulgar_punta 0.401 0.454
indice_medio 0.435 0.609
indice_punta 0.391 0.444
corazon_medio 0.500 0.600
corazon_punta 0.500 0.430
anular_medio 0.565 0.609
anular_punta 0.609 0.444
menique_medio 0.625 0.633
menique_punta 0.710 0.486
```

SALIDA ESPERADA

```
Mano 1: VICTORIA (extendidos: indice, corazon)
Mano 2: PULGAR (extendidos: pulgar)
Mano 3: PINZA (extendidos: pulgar, indice,
corazon, anular, menique)
```

97. Un componente de valoración por estrellas

Medio · Creación de Componentes Visuales · apuntesdam.com/subject/desarrollo-interfaces/topic/creacion-componentes

Completa la clase Valoracion, un componente reutilizable como los que se crean extendiendo un control (JavaFX) o una View (Android). Tiene un valor entre 0 y 5 en medios puntos. setValor recorta el valor al rango [0, 5] y lo redondea al medio punto más cercano ($\text{Math.round}(v * 2) / 2.0$); si cambia, avisa con «cambio: viejo → nuevo». clic(n) pone n estrellas, salvo que n sea igual al valor actual, que lo pone a 0 (pulsar otra vez la misma estrella borra la valoración). pintar() devuelve las 5 estrellas («★» llena, «½» media, «☆» vacía) seguidas del valor entre paréntesis. Los valores se muestran sin «.0» si son enteros (3, no 3.0).

Código de partida (java)

```
import java.util.*;

public class Main {
    // TODO: completa el componente de valoración
    static class Valoracion {
        private double valor = 0;

        void setValor(double v) { }
        void clic(int estrella) { }
        String pintar() { return ""; }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Valoracion v = new Valoracion();
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");
            if (p[0].equals("set")) v.setValor(Double.parseDouble(p[1]));
            else if (p[0].equals("clic")) v.clic(Integer.parseInt(p[1]));
            else continue;
            System.out.println(v.pintar());
        }
    }
}
```

Ejemplo: Uso normal

ENTRADA

```
clic 4
set 3.7
set 3.6
clic 4
clic 4
```

SALIDA ESPERADA

```
cambio: 0 → 4
★★★★☆ (4)
cambio: 4 → 3.5
★★★½☆ (3.5)
★★★½☆ (3.5)
cambio: 3.5 → 4
★★★★☆ (4)
cambio: 4 → 0
☆☆☆☆☆ (0)
```

98. Propiedades observables y bindings

Difícil · Creación de Componentes Visuales · apuntesdam.com/subject/desarrollo-interfaces/topic/creacion-componentes

Las propiedades de JavaFX (`IntegerProperty`, `StringProperty`...) avisan a sus oyentes cuando cambian y se pueden enlazar: una propiedad enlazada con `bind` sigue siempre el valor de otra. Implementa Propiedad (valor inicial 0) para que funcionen las órdenes: «set P v» cambia el valor (si P está enlazada, muestra «ERROR: P está enlazada a Q y no se puede cambiar»); «escuchar P» añade un oyente; «bind P Q» enlaza P a Q (P toma el valor de Q y lo sigue en adelante; si crearía un ciclo, «ERROR: enlace circular»); «unbind P» rompe el enlace; y «get P» muestra «P = valor». Cuando un valor cambia de verdad (si es igual, no pasa nada), cada oyente de esa propiedad muestra « P oyente N: viejo → nuevo» (N desde 1, con dos espacios delante) y el cambio se propaga a las propiedades enlazadas a ella, en el orden en que se enlazaron.

Código de partida (java)

```
import java.util.*;
import java.util.function.*;

public class Main {
    // TODO: implementa una propiedad observable con listeners y enlace (bind) como las de JavaFX
    static class Propiedad {
        final String nombre;
        Propiedad(String nombre) { this.nombre = nombre; }
        int get() { return 0; }
        void set(int v) { }
        void escuchar() { }
        void bind(Propiedad otra) { }
        void unbind() { }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, Propiedad> props = new LinkedHashMap<>();
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");
            if (p.length < 2) continue;
            Propiedad a = props.computeIfAbsent(p[1], Propiedad::new);
            switch (p[0]) {
                case "set" -> a.set(Integer.parseInt(p[2]));
                case "escuchar" -> a.escuchar();
                case "bind" -> a.bind(props.computeIfAbsent(p[2], Propiedad::new));
                case "unbind" -> a.unbind();
                case "get" -> System.out.println(a.nombre + " = " + a.get());
            }
        }
    }
}
```

Ejemplo: Enlace sencillo

ENTRADA

```
escuchar ancho
set ancho 100
set ancho 100
escuchar copia
bind copia ancho
set ancho 250
set copia 5
get copia
unbind copia
set copia 5
get copia
```

SALIDA ESPERADA

```
ancho oyente 1: 0 → 100
copia oyente 1: 0 → 100
ancho oyente 1: 100 → 250
copia oyente 1: 100 → 250
ERROR: copia está enlazada a ancho y no se puede
cambiar
copia = 250
copia oyente 1: 250 → 5
copia = 5
```

Ejemplo: Cadena y ciclo

ENTRADA

```
escuchar c
bind b a
bind c b
set a 7
bind a c
escuchar a
set a 8
get c
```

SALIDA ESPERADA

```
c oyente 1: 0 → 7
ERROR: enlace circular
a oyente 1: 7 → 8
c oyente 1: 7 → 8
c = 8
```

99. Un campo de texto con máscara

Muy difícil · Creación de Componentes Visuales · apuntesdam.com/subject/desarrollo-interfaces/topic/creacion-componentes

Los componentes de entrada con máscara (fechas, teléfonos, matrículas) solo aceptan caracteres válidos e insertan solos los separadores. La primera línea es «mascara M», donde en M # es un dígito, A una letra (se guarda en mayúscula), ? cualquier carácter y el resto son literales; la segunda, las teclas pulsadas separadas por espacios (un carácter, ESPACIO o BORRAR). Al escribir, primero se añaden los literales que haya antes del siguiente hueco; si la tecla no encaja con el hueco, se rechaza «(se esperaba un dígito)» o «(se esperaba una letra)»; si la máscara ya está llena, «(campo completo)». BORRAR quita el último carácter y, además, los literales que queden al final. Muestra «Pantalla: ...» con lo escrito y los huecos restantes como _ (los literales sin escribir se muestran tal cual), «Completo: sí/no» y una línea «Rechazada: tecla (motivo)» por cada tecla rechazada.

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String mascara = sc.nextLine().substring("mascara ".length()); // # dígito, A letra, ? cualquiera;
        // lo demás es literal
        List<String> teclas = new ArrayList<>(Arrays.asList(sc.nextLine().split(" ")));
        // TODO: simula un campo de texto con máscara pulsando las teclas en orden
    }
}
```

Ejemplo: Fecha

ENTRADA

mascara ##/##/####
1 5 0 3 2 0 2 6

SALIDA ESPERADA

Pantalla: 15/03/2026
Completo: sí

Ejemplo: Matrícula con errores

ENTRADA

mascara ##### AAA
1 2 x 3 4 b c 9 d e

SALIDA ESPERADA

Pantalla: 1234 BCD
Completo: sí
Rechazada: x (se esperaba un dígito)
Rechazada: 9 (se esperaba una letra)
Rechazada: e (campo completo)

Ejemplo: Teléfono con borrado

ENTRADA

mascara (###) ### ##
9 4 2 BORRAR BORRAR BORRAR 6 1 2 3 4 5 6 7 8 9

SALIDA ESPERADA

Pantalla: (612) 345 678
Completo: sí
Rechazada: 9 (campo completo)

100. Contraste de colores según WCAG

Medio · Usabilidad y Accesibilidad · apuntesdam.com/subject/desarrollo-interfaces/topic/usabilidad-accesibilidad

Cada línea tiene dos colores en hexadecimal (#rrggbb o #rgb): el del texto y el del fondo. Implementa la luminancia relativa de WCAG (cada canal se divide entre 255 y se linealiza: $v \leq 0.03928 \rightarrow v / 12.92$; si no, $((v + 0.055) / 1.055)^{2.4}$; después $L = 0.2126 R + 0.7152 G + 0.0722 B$) y el contraste $(L_{\text{clara}} + 0.05) / (L_{\text{oscura}} + 0.05)$. Muestra «texto sobre fondo: R:1 · texto normal X · texto grande Y», con R con 2 decimales; para texto normal, AAA si $R \geq 7$, AA si $R \geq 4.5$; para texto grande (18 pt, o 14 pt en negrita), AAA si $R \geq 4.5$, AA si $R \geq 3$; si no llega, «no cumple».

Código de partida (java)

```
import java.util.*;

public class Main {
    // TODO: luminancia relativa y contraste según WCAG
    static double contraste(String a, String b) { return 0; }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");
            if (p.length < 2) continue;
            double r = contraste(p[0], p[1]);
            // TODO: muestra el contraste y si cumple AA y AAA para texto normal y grande
        }
    }
}
```

Ejemplo: Combinaciones habituales

ENTRADA

```
#000000 #ffffff
#767676 #ffffff
#ffffff #4f46e5
#999 #fff
```

SALIDA ESPERADA

```
#000000 sobre #ffffff: 21.00:1 · texto normal AAA
· texto grande AAA
#767676 sobre #ffffff: 4.54:1 · texto normal AA ·
texto grande AAA
#ffffff sobre #4f46e5: 6.29:1 · texto normal AA ·
texto grande AAA
#999 sobre #fff: 2.85:1 · texto normal no cumple ·
texto grande no cumple
```

101. El orden de tabulación de un formulario

Difícil · Usabilidad y Accesibilidad · apuntesdam.com/subject/desarrollo-interfaces/topic/usabilidad-accesibilidad

Quien usa el teclado recorre la interfaz con Tab, así que el orden del foco debe ser lógico. Cada línea es un control «id x y tabindex habilitado» (habilitado vale si o no). Calcula el orden como lo hacen los navegadores y la mayoría de frameworks: primero los controles con tabindex positivo, de menor a mayor (a igualdad, por posición); después los de tabindex 0, en orden de lectura (por y y, a igual y, por x). Los deshabilitados y los de tabindex negativo no reciben el foco. Muestra «N. id» para cada control en orden, después «Aviso: K control(es) con tabindex positivo alteran el orden natural» si hay alguno, y por último, en el orden de la entrada, «Fuera del orden: id (deshabilitado)» o «Fuera del orden: id (tabindex -1)».

Código de partida (java)

```
import java.util.*;

public class Main {
    record Control(String id, int x, int y, int tabIndex, boolean habilitado) { }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<Control> controles = new ArrayList<>();
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" "); // id x y tabIndex habilitado(si|no)
            if (p.length < 5) continue;
            controles.add(new Control(p[0], Integer.parseInt(p[1]), Integer.parseInt(p[2]),
                Integer.parseInt(p[3]), p[4].equals("si")));
        }
        // TODO: calcula el orden en que recibe el foco cada control al pulsar Tab
    }
}
```

Ejemplo: Formulario bien hecho

ENTRADA

```
nombre 20 10 0 si
apellidos 220 10 0 si
email 20 60 0 si
enviar 20 120 0 si
cancelar 120 120 0 si
```

SALIDA ESPERADA

```
1. nombre
2. apellidos
3. email
4. enviar
5. cancelar
```

Ejemplo: Con tabindex positivos y deshabilitados

ENTRADA

```
buscar 500 5 3 si
nombre 20 50 0 si
menu 10 5 1 si
ayuda 400 5 -1 si
email 20 100 0 no
enviar 20 150 0 si
logo 5 5 2 si
```

SALIDA ESPERADA

```
1. menu
2. logo
3. buscar
4. nombre
5. enviar
Aviso: 3 control(es) con tabindex positivo alteran
el orden natural
Fuera del orden: ayuda (tabindex -1)
Fuera del orden: email (deshabilitado)
```

102. ¿Se entiende este texto?

Muy difícil · Usabilidad y Accesibilidad · apuntesdam.com/subject/desarrollo-interfaces/topic/usabilidad-accesibilidad

La claridad del lenguaje también es accesibilidad. El índice de Fernández Huerta mide la legibilidad de un texto en castellano: $L = 206.84 - 0.60 \cdot P - 1.02 \cdot F$, donde P son las sílabas por cada 100 palabras y F la media de palabras por frase: cuanto más alto, más fácil. Implementa silabas(): cuenta los núcleos vocálicos, de modo que una vocal empieza sílaba si va tras una consonante o si ella y la anterior son fuertes (a, e, o y cualquier vocal con tilde: hiato, como en «po-e-sí-a» o «le-í»); i y u sin tilde se unen a la vocal de al lado (diptongo, como en «ai-re» o «cie-lo»); la y final tras consonante cuenta como vocal; toda palabra tiene al menos 1 sílaba. Las frases se separan por . ! ? y las palabras son las secuencias de letras. Muestra «Palabras: N · Frases: N · Sílabas: N», «Sílabas por palabra: X · Palabras por frase: Y» (2 y 1 decimales) e «Índice Fernández Huerta: L (nivel)», con L con 1 decimal y el nivel: muy fácil (≥ 90), fácil (≥ 80), algo fácil (≥ 70), normal (≥ 60), algo difícil (≥ 50), difícil (≥ 30) o muy difícil.

Código de partida (java)

```
import java.text.Normalizer;
import java.util.*;

public class Main {
    static final String VOCALES = "aeiouáéíóúü";
    static final String FUERTES = "aeoáéíóú"; // las débiles i, u sin tilde forman diptongo

    // TODO: cuenta las sílabas de una palabra en castellano
    static int silabas(String palabra) { return 0; }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        StringBuilder todo = new StringBuilder();
        while (sc.hasNextLine()) todo.append(sc.nextLine()).append(' ');
        String texto = todo.toString().trim();
        // TODO: calcula el índice de legibilidad de Fernández Huerta
    }
}
```

Ejemplo: Mensaje de error claro

ENTRADA

```
No se ha podido guardar. Revisa tu conexión y
vuelve a intentarlo.
```

SALIDA ESPERADA

```
Palabras: 12 · Frases: 2 · Sílabas: 23
Sílabas por palabra: 1.92 · Palabras por frase:
6.0
Índice Fernández Huerta: 85.7 (fácil)
```

Ejemplo: Texto administrativo

ENTRADA

La presente aplicación procederá a la verificación pormenorizada de la documentación aportada por el interesado, conforme a la normativa vigente, con carácter previo a la tramitación definitiva del expediente correspondiente.

SALIDA ESPERADA

Palabras: 30 · Frases: 1 · Sílabas: 82
Sílabas por palabra: 2.73 · Palabras por frase: 30.0
Índice Fernández Huerta: 12.2 (muy difícil)

Ejemplo: Palabras con diptongos e hiatos

ENTRADA

El día que leí la poesía de mi abuela, oí el ruido del aire. Hoy sigue aquí.

SALIDA ESPERADA

Palabras: 17 · Frases: 2 · Sílabas: 29
Sílabas por palabra: 1.71 · Palabras por frase: 8.5
Índice Fernández Huerta: 95.8 (muy fácil)

103. El CHANGELOG a partir de los commits

Medio · Documentación de Aplicaciones · apuntesdam.com/subject/desarrollo-interfaces/topic/documentacion-aplicaciones

Con la convención Conventional Commits, cada mensaje de commit tiene la forma «tipo(ámbito opcional)!opcional: descripción», y de ellos se deducen la siguiente versión semántica y el CHANGELOG. La primera línea es «version X.Y.Z»; las siguientes, los commits. Un «!» tras el tipo indica un cambio incompatible (sube la versión mayor y pone a cero las otras), «feat» sube la menor (y pone a cero el parche) y «fix» el parche; gana el cambio más importante. Genera «## X.Y.Z» con la nueva versión y, separadas por líneas en blanco, las secciones que tengan entradas, en este orden: «### Cambios incompatibles», «### Novedades» y «### Correcciones», con una línea «- ámbito: descripción» (o «- descripción» sin ámbito) por commit. Termina con una línea en blanco y «(N commits sin entrada en el CHANGELOG)», contando los de otros tipos y los que no siguen la convención. Si no hay ningún feat, fix ni «!», muestra solo «Sin cambios publicables: se mantiene la versión X.Y.Z».

Código de partida (java)

```
import java.util.*;
import java.util.regex.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String version = sc.nextLine().split(" ")[1]; // versión actual, p. ej. 1.4.2
        List<String> commits = new ArrayList<>();
        while (sc.hasNextLine()) { String l = sc.nextLine().trim(); if (!l.isEmpty()) commits.add(l); }
        // TODO: calcula la nueva versión semántica y genera el CHANGELOG
    }
}
```

Ejemplo: Versión menor

ENTRADA

version 1.4.2
feat(informes): exportar a Excel
fix: el total no incluía el IVA
docs: actualizar el manual
fix(login): mensaje de error confuso
Arreglos varios

SALIDA ESPERADA

```
## 1.5.0

### Novedades
- **informes**: exportar a Excel

### Correcciones
- el total no incluía el IVA
- **login**: mensaje de error confuso

(2 commits sin entrada en el CHANGELOG)
```


Ejemplo: Cambio incompatible

ENTRADA

```
version 2.9.7
feat(api)!: los importes pasan a ser céntimos
feat: modo oscuro
chore: actualizar dependencias
```

SALIDA ESPERADA

```
## 3.0.0

### Cambios incompatibles
- **api**: los importes pasan a ser céntimos

### Novedades
- modo oscuro

(1 commits sin entrada en el CHANGELOG)
```

Ejemplo: Sin cambios publicables

ENTRADA

```
version 0.3.0
docs: README
test: pruebas del carrito
```

SALIDA ESPERADA

```
Sin cambios publicables: se mantiene la versión
0.3.0
```

104. De Javadoc a una página de documentación

Difícil · Documentación de Aplicaciones · apuntesdam.com/subject/desarrollo-interfaces/topic/documentacion-aplicaciones

Las herramientas como javadoc o Dokka generan la documentación de una API a partir de los comentarios `/** ... */`. La entrada es el código de una clase Java. Para cada método público (una línea que empieza por «public», contiene «(» y no es la declaración de la clase), en orden: si justo antes tiene un comentario Javadoc, genera «**###** nombre», la firma entre comillas invertidas (sin «public» ni lo que va desde la llave), una línea en blanco y la descripción (las líneas de texto del comentario unidas con un espacio); después, si los hay, una línea en blanco y «**Parámetros**» con «- nombre : descripción» por cada @param; «**Devuelve:** ...» para @return; y «**Lanza**» con «- Excepción : descripción» por cada @throws; y una línea en blanco final. Un Javadoc seguido de un método no público se descarta. Termina con «Documentados: N · Sin documentar: método1, método2» (o «ninguno») con los públicos sin Javadoc.

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<String> lineas = new ArrayList<>();
        while (sc.hasNextLine()) lineas.add(sc.nextLine());
        // TODO: extrae los comentarios Javadoc de los métodos públicos y genera la documentación en
        Markdown
    }
}
```

Ejemplo: Clase Carrito

ENTRADA

```
public class Carrito {
    private final List<Linea> lineas = new
    ArrayList<>();

    /**
     * Añade un producto al carrito. Si ya estaba,
     * suma la cantidad.
     * @param producto producto que se añade
     * @param cantidad unidades, mayor que cero
     * @throws IllegalArgumentException si la
     * cantidad no es positiva
     */
    public void anadir(Producto producto, int
    cantidad) {
        // ...
    }

    /** Importe total con IVA.
     * @return el total en euros, redondeado a dos
     * decimales */
    public double total() {
        return 0;
    }

    public void vaciar() {
        lineas.clear();
    }

    /**
     * Cuenta las unidades de todas las líneas.
     */
    private int unidades() {
        return 0;
    }

    public boolean estaVacio() { return
    lineas.isEmpty(); }
}
```

SALIDA ESPERADA

```
### anadir
`void anadir(Producto producto, int cantidad)`

Añade un producto al carrito. Si ya estaba, suma
la cantidad.

**Parámetros**
- `producto`: producto que se añade
- `cantidad`: unidades, mayor que cero

**Lanza**
- `IllegalArgumentException`: si la cantidad no es
positiva

### total
`double total()`

Importe total con IVA.

**Devuelve**: el total en euros, redondeado a dos
decimales

Documentados: 2 · Sin documentar: vaciar,
estaVacio
```

105. Un revisor automático de manuales

Muy difícil · Documentación de Aplicaciones · apuntesdam.com/subject/desarrollo-interfaces/topic/documentacion-aplicaciones

Igual que se pasa un linter al código, se puede revisar la documentación. La entrada es un manual en Markdown. Ignorando lo que hay dentro de bloques de código (entre líneas que empiezan por ``), detecta en este orden: (1) en una primera pasada por los títulos, que el documento empiece con un título de nivel 1 («Línea N: el documento debe empezar con un título de nivel 1») y que no se salten niveles («Línea N: salto de título de nivel A a B», por ejemplo de ## a #####), guardando el ancla de cada título (minúsculas, sin tildes, solo letras, números, espacios y guiones, y los espacios cambiados por guiones); (2) en una segunda pasada, las imágenes ![texto](ruta) sin texto alternativo («Línea N: la imagen RUTA no tiene texto alternativo»), los enlaces cuyo texto es «aquí», «aqui», «pulsa aquí» o «este enlace» («Línea N: el texto del enlace «X» no dice adónde lleva») y los enlaces internos #ancla que no existen («Línea N: el enlace #X apunta a una sección que no existe»); (3) si queda un bloque de código sin cerrar, «Hay un bloque de código sin cerrar». Termina con «N problemas» o «Manual correcto».

Código de partida (java)

```
import java.util.*;
import java.util.regex.*;
import java.text.Normalizer;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<String> lineas = new ArrayList<>();
        while (sc.hasNextLine()) lineas.add(sc.nextLine());
        // TODO: revisa el manual y muestra los problemas encontrados
    }
}
```

Ejemplo: Manual con problemas

ENTRADA

```
# Manual de usuario de Reservas

Consulta la [instalación](#instalacion) antes de
empezar.

## Instalación
Descarga el instalador [aquí]
(https://ejemplo.es/descargas).

#### Requisitos


## Crear una reserva
1. Pulsa **Nueva**.
2. Revisa la sección de [errores frecuentes]
(#errores-frecuentes).

...

# esto es un comentario, no un título

...

## Errores habituales
![Mensaje de error de conexión](img/error.png)
```

SALIDA ESPERADA

Línea 8: salto de título de nivel 2 a 4
Línea 6: el texto del enlace «aquí» no dice adónde lleva
Línea 9: la imagen img/requisitos.png no tiene texto alternativo
Línea 13: el enlace #errores-frecuentes apunta a una sección que no existe
4 problemas

Ejemplo: Manual correcto

ENTRADA

```
# Guía

## Uso
Ver [la instalación](#instalacion).

## Instalación
![Pantalla de inicio](inicio.png)
```

SALIDA ESPERADA

Manual correcto

106. Versiones semánticas y rangos de dependencias

Medio · Distribución de Aplicaciones · apuntesdam.com/subject/desarrollo-interfaces/topic/distribucion-aplicaciones

Los gestores de paquetes (Maven, npm, Gradle) eligen versiones comparándolas según el versionado semántico. La primera línea es una lista de versiones separadas por espacios, que pueden tener una etiqueta de prelanzamiento («1.2.0-beta.2»); el programa las ordena con `comparar()` y las muestra. Cada línea siguiente es un rango: «`^X.Y.Z`» acepta versiones desde X.Y.Z hasta antes de la siguiente mayor (si X es 0, hasta la siguiente menor; si X e Y son 0, solo ese parche), «`~X.Y.Z`» hasta antes de la siguiente menor, y una versión sola solo esa. Para cada rango, el programa muestra la versión estable (sin etiqueta) más alta que lo cumple, o «ninguna». Implementa `comparar`: primero por los tres números;

una versión sin etiqueta es mayor que la misma con etiqueta; entre etiquetas se comparan sus partes separadas por puntos (las numéricas como números y menores que las de texto, y las de texto alfabéticamente) y, si una se acaba antes, es menor.

Código de partida (java)

```
import java.util.*;

public class Main {
    // TODO: compara dos versiones semánticas (negativo si a < b, 0 si iguales, positivo si a > b)
    static int comparar(String a, String b) { return 0; }

    // TODO: ¿cumple la versión el rango? (^X.Y.Z, ~X.Y.Z o una versión exacta)
    static boolean cumple(String version, String rango) { return false; }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<String> versiones = new ArrayList<>(Arrays.asList(sc.nextLine().trim().split(" ")));
        versiones.sort(Main::comparar);
        System.out.println("Ordenadas: " + String.join(" < ", versiones));
        while (sc.hasNextLine()) {
            String rango = sc.nextLine().trim();
            if (rango.isEmpty()) continue;
            String mejor = null;
            for (String v : versiones) if (cumple(v, rango) && !v.contains("-")) mejor = v;    // la más alta estable
            System.out.println(rango + " → " + (mejor == null ? "ninguna" : mejor));
        }
    }
}
```

Ejemplo: Dependencia típica

ENTRADA

```
1.2.0 1.10.1 1.2.10 2.0.0 1.9.0-beta 2.0.0-rc.1
1.2.3
^1.2.0
~1.2.0
^2.0.0
1.9.0
```

SALIDA ESPERADA

```
Ordenadas: 1.2.0 < 1.2.3 < 1.2.10 < 1.9.0-beta <
1.10.1 < 2.0.0-rc.1 < 2.0.0
^1.2.0 → 1.10.1
~1.2.0 → 1.2.10
^2.0.0 → 2.0.0
1.9.0 → ninguna
```

Ejemplo: Versiones 0.x

ENTRADA

```
0.3.1 0.3.9 0.4.0 0.0.3 0.0.4 1.0.0-alpha
^0.3.1
^0.0.3
~0.3.0
```

SALIDA ESPERADA

```
Ordenadas: 0.0.3 < 0.0.4 < 0.3.1 < 0.3.9 < 0.4.0 <
1.0.0-alpha
^0.3.1 → 0.3.9
^0.0.3 → 0.0.3
~0.3.0 → 0.3.9
```

107. ¿Puedo distribuir mi aplicación con estas librerías?

Difícil · Distribución de Aplicaciones · apuntesdam.com/subject/desarrollo-interfaces/topic/distribucion-aplicaciones

Antes de distribuir hay que revisar las licencias de las dependencias. La primera línea es «proyecto LICENCIA» (MIT, Apache-2.0, GPL-3.0, GPL-2.0 o Propietaria) y cada línea siguiente una dependencia «nombre licencia enlace», con enlace estatico (incluida en el mismo JAR) o dinamico (un JAR aparte que se carga). Reglas: MIT, BSD-3-Clause, Apache-2.0 y EPL-2.0 son permisivas → «OK (permisiva: incluye su aviso de copyright)», salvo Apache-2.0 en un proyecto GPL-2.0 → «PROBLEMA (Apache-2.0 no es compatible con GPL-2.0)»; LGPL-3.0 dinámica → «OK (LGPL enlazada dinámicamente: el usuario debe poder sustituirla)» (cuenta como aviso), estática en proyecto GPL → «OK (compatible con GPL)», estática en otro → «PROBLEMA (LGPL enlazada estáticamente obliga a facilitar el código objeto para reenlazar)»; GPL-3.0 en proyecto GPL-3.0 → «OK (misma licencia)», GPL-2.0 en proyecto GPL-3.0 → «PROBLEMA (GPL-2.0 sola no es compatible con GPL-3.0)», cualquier GPL en otro proyecto → «PROBLEMA (copyleft fuerte: obligaría a publicar el

proyecto como LICENCIA)»; AGPL-3.0 → «PROBLEMA (AGPL: también obliga a compartir el código si se ofrece como servicio en red)»; cualquier otra → «REVISAR (licencia desconocida: hay que leerla)» (aviso). Muestra «nombre [licencia, enlace]: veredicto» y «Resumen: P problemas, A avisos → se puede distribuir» o «→ no distribuir sin resolverlos».

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] proyecto = sc.nextLine().trim().split(" ");    // proyecto LICENCIA (MIT, Apache-2.0, GPL-
        3.0, Propietaria)
        String licProyecto = proyecto[1];
        List<String[]> deps = new ArrayList<>();                // nombre licencia
        enlace(estatico|dinamico)
        while (sc.hasNextLine()) { String l = sc.nextLine().trim(); if (!l.isEmpty()) deps.add(l.split("
        ")); }
        // TODO: revisa si cada dependencia es compatible con la licencia del proyecto
    }
}
```

Ejemplo: App propietaria

ENTRADA

```
proyecto Propietaria
jackson Apache-2.0 estatico
openpdf LGPL-3.0 dinamico
mysql-connector GPL-2.0 dinamico
logback EPL-2.0 estatico
hibernate LGPL-3.0 estatico
```

SALIDA ESPERADA

```
jackson [Apache-2.0, estatico]: OK (permisiva:
incluye su aviso de copyright)
openpdf [LGPL-3.0, dinamico]: OK (LGPL enlazada
dinámicamente: el usuario debe poder sustituirla)
mysql-connector [GPL-2.0, dinamico]: PROBLEMA
(copyleft fuerte: obligaría a publicar el proyecto
como GPL-2.0)
logback [EPL-2.0, estatico]: OK (permisiva:
incluye su aviso de copyright)
hibernate [LGPL-3.0, estatico]: PROBLEMA (LGPL
enlazada estáticamente obliga a facilitar el
código objeto para reenlazar)
Resumen: 2 problemas, 1 avisos → no distribuir sin
resolverlos
```

Ejemplo: App GPL

ENTRADA

```
proyecto GPL-3.0
libreria-gpl GPL-3.0 estatico
vieja GPL-2.0 estatico
jsoup MIT estatico
rara WTFPL estatico
```

SALIDA ESPERADA

```
libreria-gpl [GPL-3.0, estatico]: OK (misma
licencia)
vieja [GPL-2.0, estatico]: PROBLEMA (GPL-2.0 sola
no es compatible con GPL-3.0)
jsoup [MIT, estatico]: OK (permisiva: incluye su
aviso de copyright)
rara [WTFPL, estatico]: REVISAR (licencia
desconocida: hay que leerla)
Resumen: 1 problemas, 1 avisos → no distribuir sin
resolverlos
```

108. Un actualizador incremental con despliegue escalonado

Muy difícil · Distribución de Aplicaciones · apuntesdam.com/subject/desarrollo-interfaces/topic/distribucion-aplicaciones

Las aplicaciones de escritorio modernas se actualizan solas descargando solo lo que ha cambiado y publicando la versión nueva poco a poco. La entrada tiene una configuración «clave valor» (version_instalada, version_nueva, minima, porcentaje y usuario) y dos manifiestos, tras «[local]» y «[remoto]», con líneas «ruta tamaño_bytes hash». Si la versión instalada ya es la nueva o superior: «Ya tienes la última versión (X)». Si es menor que la mínima, la actualización es obligatoria. Si no, el usuario la recibe solo si su grupo, $\text{Math.floorMod}(\text{usuario.hashCode}(), 100)$, es menor que el porcentaje publicado; si no, «Actualización N aún no disponible para este usuario (grupo G, se publica al P %)». Cuando

hay actualización: «Actualizar de A a N (obligatoria: la versión mínima es M)» o «... (grupo G)», después « NUEVO ruta», « CAMBIA ruta» (mismo nombre, distinto hash) en orden alfabético, « BORRAR ruta» para lo que ya no existe, y «Descarga: F ficheros, X KB de T KB (ahorro del S %)» (KB = 1024 bytes, 1 decimal; ahorro sin decimales).

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, String> conf = new HashMap<>();
        Map<String, String[]> local = new TreeMap<>(), remoto = new TreeMap<>(); // ruta → {tamaño, hash}
        Map<String, String[]> actual = null;
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (l.isEmpty()) continue;
            if (l.equals("[local]")) { actual = local; continue; }
            if (l.equals("[remoto]")) { actual = remoto; continue; }
            String[] p = l.split(" ");
            if (actual == null) conf.put(p[0], p[1]); // version_instalada, version_nueva,
                minima, porcentaje, usuario
            else actual.put(p[0], new String[]{p[1], p[2]});
        }
        // TODO: decide si este usuario recibe la actualización y qué ficheros descargar o borrar
    }
}
```

Ejemplo: Actualización parcial

ENTRADA

```
version_instalada 2.3.1
version_nueva 2.4.0
minima 2.0.0
porcentaje 100
usuario ana@hotel.es
[local]
app.jar 812000 a1f3
lib/javafx-controls.jar 2400000 9c0d
lib/openpdf.jar 1300000 77aa
informes/factura.jasper 45000 c3c3
ayuda/manual.html 120000 e5e5
[remoto]
app.jar 830000 b2b2
lib/javafx-controls.jar 2400000 9c0d
lib/openpdf.jar 1350000 88bb
informes/factura.jasper 45000 c3c3
informes/ocupacion.jasper 52000 d4d4
```

SALIDA ESPERADA

```
Actualizar de 2.3.1 a 2.4.0 (grupo 48)
CAMBIA app.jar
NUEVO informes/ocupacion.jasper
CAMBIA lib/openpdf.jar
BORRAR ayuda/manual.html
Descarga: 3 ficheros, 2179.7 KB de 4567.4 KB
(ahorro del 52 %)
```

Ejemplo: Aún no publicada para este usuario

ENTRADA

```
version_instalada 2.3.1
version_nueva 2.4.0
minima 2.0.0
porcentaje 10
usuario ana@hotel.es
[local]
app.jar 812000 a1f3
lib/javafx-controls.jar 2400000 9c0d
lib/openpdf.jar 1300000 77aa
informes/factura.jasper 45000 c3c3
ayuda/manual.html 120000 e5e5
[remoto]
app.jar 830000 b2b2
lib/javafx-controls.jar 2400000 9c0d
lib/openpdf.jar 1350000 88bb
informes/factura.jasper 45000 c3c3
informes/ocupacion.jasper 52000 d4d4
```

SALIDA ESPERADA

```
Actualización 2.4.0 aún no disponible para este
usuario (grupo 48, se publica al 10 %)
```

Ejemplo: Obligatoria y al día

ENTRADA

```
version_instalada 1.9.5
version_nueva 2.4.0
minima 2.0.0
porcentaje 0
usuario luis
[local]
app.jar 812000 a1f3
lib/javafx-controls.jar 2400000 9c0d
lib/openpdf.jar 1300000 77aa
informes/factura.jasper 45000 c3c3
ayuda/manual.html 120000 e5e5
[remoto]
app.jar 812000 a1f3
lib/javafx-controls.jar 2400000 9c0d
lib/openpdf.jar 1350000 88bb
informes/factura.jasper 45000 c3c3
informes/ocupacion.jasper 52000 d4d4
```

SALIDA ESPERADA

```
Actualizar de 1.9.5 a 2.4.0 (obligatoria: la
versión mínima es 2.0.0)
  NUEVO informes/ocupacion.jasper
  CAMBIA lib/openpdf.jar
  BORRAR ayuda/manual.html
Descarga: 2 ficheros, 1369.1 KB de 4549.8 KB
(ahorro del 70 %)
```

109. Casos de prueba por valores límite

Medio · Pruebas de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/pruebas-interfaces

Los errores se esconden en los bordes. Cada línea describe un campo de un formulario: «campo tipo min max», con tipo entero (el valor debe estar entre min y max) o texto (la longitud debe estar entre min y max). Genera sus casos de prueba: «== campo (tipo, min..max)» y, numerados « campo-N: valor → resultado», primero los valores límite min – 1, min, min + 1, max – 1, max y max + 1 (sin repetir y, en texto, sin longitudes negativas), cada uno «válido» o «inválido». En los enteros, el valor límite se muestra tal cual, y después se añaden un valor intermedio $((\text{min} + \text{max}) / 2 \rightarrow \text{válido})$, si no estaba ya y «abc» → «inválido (no es un número)». En los textos, una longitud 0 se muestra como "", hasta 12 como la cadena de «a» entre comillas seguida de «(N caracteres)» («(1 carácter)» en singular) y más larga como «texto de N caracteres»; después se añade «" " (solo espacios) → inválido (vacío tras recortar)». En ambos, el último caso es «(vacío) → inválido (obligatorio)».

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");    // campo tipo min max
            if (p.length < 4) continue;
            String campo = p[0], tipo = p[1];
            int min = Integer.parseInt(p[2]), max = Integer.parseInt(p[3]);
            // TODO: genera los casos de prueba por valores límite y clases de equivalencia
        }
    }
}
```

Ejemplo: Edad y nombre

ENTRADA

```
edad entero 18 65
nombre texto 2 10
```

SALIDA ESPERADA

```
== edad (entero, 18..65)
edad-1: 17 → inválido
edad-2: 18 → válido
edad-3: 19 → válido
edad-4: 64 → válido
edad-5: 65 → válido
edad-6: 66 → inválido
edad-7: 41 → válido
edad-8: abc → inválido (no es un número)
edad-9: (vacío) → inválido (obligatorio)
== nombre (texto, 2..10)
nombre-1: "a" (1 carácter) → inválido
nombre-2: "aa" (2 caracteres) → válido
nombre-3: "aaa" (3 caracteres) → válido
nombre-4: "aaaaaaaa" (9 caracteres) → válido
nombre-5: "aaaaaaaaaa" (10 caracteres) → válido
nombre-6: "aaaaaaaaaaa" (11 caracteres) →
inválido
nombre-7: " " (solo espacios) → inválido
(vacío tras recortar)
nombre-8: (vacío) → inválido (obligatorio)
```

Ejemplo: Rango estrecho

ENTRADA

```
planta entero 0 2
cp texto 5 5
```

SALIDA ESPERADA

```
== planta (entero, 0..2)
planta-1: -1 → inválido
planta-2: 0 → válido
planta-3: 1 → válido
planta-4: 2 → válido
planta-5: 3 → inválido
planta-6: abc → inválido (no es un número)
planta-7: (vacío) → inválido (obligatorio)
== cp (texto, 5..5)
cp-1: "aaaa" (4 caracteres) → inválido
cp-2: "aaaaa" (5 caracteres) → válido
cp-3: "aaaaaa" (6 caracteres) → inválido
cp-4: " " (solo espacios) → inválido (vacío
tras recortar)
cp-5: (vacío) → inválido (obligatorio)
```

110. Un mini ejecutor de pruebas unitarias

Difícil · Pruebas de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/pruebas-interfaces

Implementa `precioFinal(importe, código, socio)` según esta especificación y un pequeño ejecutor de pruebas al estilo de JUnit. Especificación: un importe negativo lanza `IllegalArgumentException`; el código VERANO10 descuenta un 10 % y VIP25 un 25 % solo si el importe es de al menos 100 €; los socios tienen un 5 % adicional; el descuento total nunca pasa del 30 %; se suman 4,95 € de envío si el importe con descuento es menor de 50 € (salvo que el importe sea 0); y el resultado se redondea a céntimos. Cada línea de la entrada es un caso «nombre importe código(o -) socio(si|no) esperado», donde esperado es un número o la palabra `excepcion`. El ejecutor muestra «PASA nombre» (dos espacios), «PASA nombre (lanza `IllegalArgumentException`)», «FALLA nombre: esperado X, obtenido Y», «FALLA nombre: se esperaba una excepción y devolvió Y» o «ERROR nombre: Excepcion: mensaje» si salta una excepción no esperada, y al final «Pruebas: N · Pasan: P · Fallan: F · Errores: E». Compara con una tolerancia de 0,005.

Código de partida (java)

```
import java.util.*;

public class Main {
    // TODO: implementa precioFinal según la especificación del enunciado
    static double precioFinal(double importe, String codigo, boolean socio) {
        return importe;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // Cada línea: nombre_prueba importe codigo(o -) socio(si|no) esperado
        // TODO: ejecuta cada caso, compara con el esperado (tolerancia 0.005) y muestra el informe
    }
}
```

Ejemplo: Especificación correcta

ENTRADA

```
sin_descuento 40 - no 44.95
envio_gratis_justo 50 - no 50.00
verano 100 VERANO10 no 90.00
vip_minimo 100 VIP25 no 75.00
vip_insuficiente 99.99 VIP25 no 99.99
socio 40 - si 42.95
tope_30 200 VIP25 si 140.00
importe_negativo -5 - no excepcion
gratis 0 VERANO10 no 0.00
```

SALIDA ESPERADA

```
PASA sin_descuento
PASA envio_gratis_justo
PASA verano
PASA vip_minimo
PASA vip_insuficiente
PASA socio
PASA tope_30
PASA importe_negativo (lanza
IllegalArgumentException)
PASA gratis
Pruebas: 9 · Pasan: 9 · Fallan: 0 · Errores: 0
```

Ejemplo: Casos mal escritos

ENTRADA

```
esperado_mal 40 - no 40.00
espera_excepcion 10 - no excepcion
negativo_sin_esperarlo -1 - no 0
```

SALIDA ESPERADA

```
FALLA esperado_mal: esperado 40.00, obtenido 44.95
FALLA espera_excepcion: se esperaba una excepción
y devolvió 14.95
ERROR negativo_sin_esperarlo:
IllegalArgumentException: importe negativo
Pruebas: 3 · Pasan: 0 · Fallan: 2 · Errores: 1
```

111. Pruebas de interfaz guionizadas

Muy difícil · Pruebas de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/pruebas-interfaces

Las herramientas de pruebas de extremo a extremo (Playwright, Selenium, TestFX) siguen un guion: escribir, pulsar y esperar un resultado. Completa el modelo de PantallaAcceso y el ejecutor. Reglas de la pantalla: el botón Entrar se habilita cuando el usuario no está en blanco, la clave tiene al menos 4 caracteres y hay menos de 3 intentos fallidos; pulsar con el botón deshabilitado no hace nada; con datos correctos (usuario sin distinguir mayúsculas y sin espacios alrededor) se pasa a la pantalla principal con el mensaje «Bienvenida, Usuario» (tal como se escribió, sin espacios); si no, se borra la clave, se deshabilita el botón, suma un intento y el mensaje es «Usuario o contraseña incorrectos (N intentos más)» o, al tercer fallo, «Cuenta bloqueada: demasiados intentos». El guion tiene líneas «escribe campo texto», «pulsar entrar» y «espera qué valor» (qué: mensaje, boton —habilitado/deshabilitado—, clave o pantalla —acceso/principal—); las líneas vacías y las que empiezan por # se ignoran. Por cada «espera» muestra «✓ paso N: qué = «valor»» o «X paso N: qué esperaba «X» pero es «Y»»; para una orden desconocida, «? paso N: orden desconocida «orden»». Termina con «PRUEBA SUPERADA» o «PRUEBA FALLIDA (F comprobaciones)».

Código de partida (java)

```
import java.util.*;

public class Main {
    // Modelo de una pantalla de acceso: campos usuario y clave, botón entrar y un mensaje
    static class PantallaAcceso {
        String usuario = "", clave = "", mensaje = "";
        boolean entrarHabilitado = false, dentro = false;
        int intentos = 0;
        static final Map<String, String> CUENTAS = Map.of("ana", "Segura123", "luis", "Clave2026");

        // TODO: escribir en un campo, pulsar el botón y actualizar el estado según las reglas
        void escribir(String campo, String texto) { }
        void pulsarEntrar() { }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // TODO: ejecuta el guion de prueba paso a paso y comprueba cada «espera»
    }
}
```

Ejemplo: Acceso correcto

ENTRADA

```
# camino feliz
espera boton deshabilitado
escribe usuario Ana
escribe clave Segura123
espera boton habilitado
pulsar entrar
espera pantalla principal
espera mensaje Bienvenida, Ana
```

SALIDA ESPERADA

```
✓ paso 1: boton = «deshabilitado»
✓ paso 4: boton = «habilitado»
✓ paso 6: pantalla = «principal»
✓ paso 7: mensaje = «Bienvenida, Ana»
PRUEBA SUPERADA
```

Ejemplo: Tres intentos fallidos

ENTRADA

```
escribe usuario luis
escribe clave mala
pulsar entrar
espera mensaje Usuario o contraseña incorrectos (2
intentos más)
espera clave
escribe clave otra1
pulsar entrar
escribe clave otra2
pulsar entrar
espera mensaje Cuenta bloqueada: demasiados
intentos
escribe clave Clave2026
espera boton deshabilitado
pulsar entrar
espera pantalla acceso
```

SALIDA ESPERADA

```
✓ paso 4: mensaje = «Usuario o contraseña
incorrectos (2 intentos más)»
✓ paso 5: clave = «»
✓ paso 10: mensaje = «Cuenta bloqueada: demasiados
intentos»
✓ paso 12: boton = «deshabilitado»
✓ paso 14: pantalla = «acceso»
PRUEBA SUPERADA
```

112. Validar un NIF en un formulario

Fácil · Ejercicios de Desarrollo de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/ejercicios-desarrollo-interfaces

Antes de guardar un formulario hay que validar el NIF. Implementa `nifValido`: 8 dígitos y una letra (mayúscula o minúscula) cuya letra de control sea la correcta (`TRWAGMYFPDXBNJZSQVHLCKE[número % 23]`). Se ignoran espacios y guiones. El programa lee un NIF por línea y muestra «válido» o «no válido».

Código de partida (java)

```
import java.util.*;

public class Main {
    static final String LETRAS = "TRWAGMYFPDXBNJZSQVHLCKE";

    static boolean nifValido(String nif) {
        // TODO
        return false;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String l = sc.nextLine();
            if (!l.isBlank()) System.out.println(nifValido(l) ? "válido" : "no válido");
        }
    }
}
```

Ejemplo: NIF correctos e incorrectos

ENTRADA

12345678Z
12345678A
00000000T

SALIDA ESPERADA

válido
no válido
válido

Ejemplo: Formato

ENTRADA

1234567Z
12 345 678-z
ABCDEFGHYZ

SALIDA ESPERADA

no válido
válido
no válido

113. Informe agrupado con subtotales

Medio · Ejercicios de Desarrollo de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/ejercicios-desarrollo-interfaces

Una pantalla de informes recibe líneas «categoría;importe». Muestra las categorías en orden alfabético con su subtotal y al final el total, con dos decimales y coma decimal:

Audio: 104,95

Informática: 179,00

TOTAL: 283,95

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // TODO: lee las líneas categoría;importe y muestra el informe
    }
}
```

Ejemplo: Tres líneas

ENTRADA

Audio;59.95
Informática;179
Audio;45

SALIDA ESPERADA

Audio: 104,95
Informática: 179,00
TOTAL: 283,95

114. Paginación de una tabla

Medio · Ejercicios de Desarrollo de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/ejercicios-desarrollo-interfaces

Una tabla muestra los resultados en páginas. La primera línea de la entrada tiene el tamaño de página y la página pedida (empezando en 1); después, un elemento por línea. Muestra los elementos de esa página y al final «Página X de Y». Si la página no existe, muestra solo «Página no válida». Si no hay elementos, «Sin resultados».

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int tam = sc.nextInt(), pagina = sc.nextInt();
        sc.nextLine();
        List<String> elementos = new ArrayList<>();
        while (sc.hasNextLine()) {
            String l = sc.nextLine();
            if (!l.isBlank()) elementos.add(l);
        }
        // TODO
    }
}
```

Ejemplo: Segunda página

ENTRADA

```
3 2
Ana
Luis
Marta
Pablo
Rosa
```

SALIDA ESPERADA

```
Pablo
Rosa
Página 2 de 2
```

Ejemplo: Página fuera de rango

ENTRADA

```
2 4
a
b
c
```

SALIDA ESPERADA

```
Página no válida
```

115. Comparar versiones semánticas

Medio · Ejercicios de Desarrollo de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/ejercicios-desarrollo-interfaces

El actualizador de la aplicación compara la versión instalada con la publicada. Cada línea tiene dos versiones MAYOR.MENOR.PARCHE separadas por un espacio. Muestra «<», «>» o «=» según la primera sea menor, mayor o igual que la segunda. Ojo: 2.10.0 es mayor que 2.9.5 (se comparan números, no textos).

Código de partida (java)

```
import java.util.*;

public class Main {

    static int comparar(String a, String b) {
        // TODO: negativo si a < b, 0 si son iguales, positivo si a > b
        return a.compareTo(b);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNext()) {
            int c = comparar(sc.next(), sc.next());
            System.out.println(c < 0 ? "<" : c > 0 ? ">" : "=");
        }
    }
}
```

Ejemplo: Casos básicos

ENTRADA

```
1.0.0 1.0.1
2.1.0 2.0.9
3.4.5 3.4.5
```

SALIDA ESPERADA

```
<
>
=
```

Ejemplo: Números de dos cifras

ENTRADA

```
2.10.0 2.9.5
1.2.10 1.2.9
```

SALIDA ESPERADA

```
>
>
```

Programación Multimedia y Dispositivos Móviles

116. Densidades de pantalla y recursos de Android

Medio · Tecnologías y Plataformas Móviles · apuntesdam.com/subject/prog-multimedia/topic/tecnologias-plataformas-movil

Android agrupa las pantallas por densidad para escoger imágenes y por anchura mínima (smallest width) para escoger diseños. Cada línea es un dispositivo «nombre anchoPx altoPx pulgadas». Calcula su densidad real (diagonal en píxeles / pulgadas), el grupo de densidad más cercano (ldpi 120, mdpi 160, hdpi 240, xhdpi 320, xxhdpi 480, xxxhdpi 640; ante empate, el menor), el factor de escala (dpi del grupo / 160), su tamaño en dp (lado corto y lado largo divididos por el factor, redondeados), si es «móvil», «tableta» (lado corto ≥ 600 dp) o «tableta grande» (≥ 720 dp) con su carpeta de diseños (layout, layout-sw600dp o layout-sw720dp) y cuántos píxeles mide un icono de 48 dp. Formato: «nombre: D dpi $\rightarrow$ grupo ($\times F$), A $\times$ L dp, tipo $\rightarrow$ carpeta, icono 48 dp = N px», con el factor sin «.0» si es entero ($\times 1.5$, $\times 2$).

Código de partida (java)

```
import java.util.*;

public class Main {
    static final String[] NOMBRES = {"ldpi", "mdpi", "hdpi", "xhdpi", "xxhdpi", "xxxhdpi"};
    static final int[] DPI = {120, 160, 240, 320, 480, 640};

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");    // nombre anchoPx altoPx pulgadas
            if (p.length < 4) continue;
            // TODO: calcula densidad, tamaño en dp y carpeta de recursos
        }
    }
}
```

Ejemplo: Móviles y tableta

ENTRADA

```
Pixel7 1080 2400 6.3
GalaxyTabS9 1600 2560 11
MovilBarato 720 1600 6.5
```

SALIDA ESPERADA

```
Pixel7: 418 dpi → xxhdpi (×3), 360 × 800 dp, móvil
→ layout, icono 48 dp = 144 px
GalaxyTabS9: 274 dpi → hdpi (×1.5), 1067 × 1707
dp, tableta grande → layout-sw720dp, icono 48 dp =
72 px
MovilBarato: 270 dpi → hdpi (×1.5), 480 × 1067 dp,
móvil → layout, icono 48 dp = 72 px
```

117. El ciclo de vida de una Activity

Difícil · Tecnologías y Plataformas Móviles · apuntesdam.com/subject/prog-multimedia/topic/tecnologias-plataformas-movil

Entender cuándo llama Android a cada método del ciclo de vida evita perder datos. Cada línea es un evento; simula las llamadas y el estado de la pantalla con un campo de texto: «abrir» (onCreate → onStart → onResume), «escribir TEXTO» (el usuario escribe, sin llamadas), «dialogo» (un diálogo encima: onPause si estaba reanudada) y «cerrar_dialogo» (onResume si estaba pausada), «home» (onPause si estaba reanudada, onStop, onSaveInstanceState: se guarda el texto), «volver» (si estaba detenida: onRestart → onStart → onResume), «girar» (se destruye y se recrea: onPause → onStop → onSaveInstanceState → onDestroy → onCreate → onStart → onRestoreInstanceState → onResume, y el texto se recupera de lo guardado), «matar_proceso» (si estaba detenida, el sistema elimina el proceso: «(proceso eliminado sin más llamadas)», estado NINGUNO y el texto en memoria se pierde), «reabrir» (onCreate → onStart → onRestoreInstanceState → onResume, recuperando lo guardado) y «atras» (onPause si estaba reanudada, onStop, onDestroy: estado DESTRUIDA y no se guarda nada). Muestra «evento: llamadas separadas por → | estado ESTADO | texto «texto»» (para «escribir», la llamada es «(el usuario escribe «texto»)»; para un evento desconocido, «(evento desconocido)»).

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // Estados: NINGUNO, CREADA, INICIADA, REANUDADA, PAUSADA, DETENIDA, DESTRUIDA
        // TODO: simula el ciclo de vida de una Activity ante cada evento
    }
}
```

Ejemplo: Uso normal

ENTRADA

```
abrir
escribir Hola
dialogo
cerrar_dialogo
home
volver
atras
```

SALIDA ESPERADA

```
abrir: onCreate → onStart → onResume | estado
REANUDADA | texto «»
escribir: (el usuario escribe «Hola») | estado
REANUDADA | texto «Hola»
dialogo: onPause | estado PAUSADA | texto «Hola»
cerrar_dialogo: onResume | estado REANUDADA |
texto «Hola»
home: onPause → onStop → onSaveInstanceState |
estado DETENIDA | texto «Hola»
volver: onRestart → onStart → onResume | estado
REANUDADA | texto «Hola»
atras: onPause → onStop → onDestroy | estado
DESTRUIDA | texto «»
```

Ejemplo: Giro y proceso eliminado

ENTRADA

```
abrir
escribir Pedido 42
girar
home
matar_proceso
reabrir
```

SALIDA ESPERADA

```
abrir: onCreate → onStart → onResume | estado
REANUDADA | texto «»
escribir: (el usuario escribe «Pedido 42») |
estado REANUDADA | texto «Pedido 42»
girar: onPause → onStop → onSaveInstanceState →
onDestroy → onCreate → onStart →
onRestoreInstanceState → onResume | estado
REANUDADA | texto «Pedido 42»
home: onPause → onStop → onSaveInstanceState |
estado DETENIDA | texto «Pedido 42»
matar_proceso: (proceso eliminado sin más
llamadas) | estado NINGUNO | texto «»
reabrir: onCreate → onStart →
onRestoreInstanceState → onResume | estado
REANUDADA | texto «Pedido 42»
```

118. Elegir el minSdk de una app

Muy difícil · Tecnologías y Plataformas Móviles · apuntesdam.com/subject/prog-multimedia/topic/tecnologias-plataformas-movil

Subir el minSdk permite usar APIs más nuevas sin comprobaciones, pero deja fuera a quienes tienen móviles antiguos. La entrada tiene líneas «cuota NIVEL PORCENTAJE» (dispositivos activos con cada nivel de API; puede que no sumen 100, así que hay que normalizar), «api NOMBRE NIVEL» (APIs que usa la app y el nivel en que aparecieron), «minSdk N» y «objetivo P». Muestra «minSdk N: X % de los dispositivos» (los de nivel $\geq N$, 1 decimal); el minSdk más alto que aún cubre al menos el objetivo: «Recomendado para el P %: minSdk M (X %)» (P sin decimales) o «Ningún minSdk alcanza el P %»; y para cada API, «NOMBRE: disponible siempre (API L)» si $L \leq \text{minSdk}$, o «NOMBRE: comprobar Build.VERSION.SDK_INT $\geq L$ (no disponible en el X % de los dispositivos)» con el porcentaje de dispositivos entre minSdk y L - 1.

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        TreeMap<Integer, Double> cuota = new TreeMap<>();           // nivel de API → % de dispositivos
        List<String[]> apis = new ArrayList<>();                   // nombre nivelMinimo
        int minSdk = 0;
        double objetivo = 0;
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");
            switch (p[0]) {
                case "cuota" -> cuota.put(Integer.parseInt(p[1]), Double.parseDouble(p[2]));
                case "api" -> apis.add(new String[]{p[1], p[2]});
                case "minSdk" -> minSdk = Integer.parseInt(p[1]);
                case "objetivo" -> objetivo = Double.parseDouble(p[1]);
                default -> { }
            }
        }
        // TODO: cobertura del minSdk, minSdk recomendado y comprobaciones de versión necesarias
    }
}
```

Ejemplo: App de pagos

ENTRADA

```
cuota 24 2.0
cuota 26 4.0
cuota 28 6.5
cuota 29 9.0
cuota 30 13.5
cuota 31 15.0
cuota 33 20.0
cuota 34 18.0
cuota 35 12.0
api Notificaciones 26
api PhotoPicker 33
api PermisoNotificaciones 33
minSdk 26
objetivo 95
```

SALIDA ESPERADA

```
minSdk 26: 98.0 % de los dispositivos
Recomendado para el 95 %: minSdk 26 (98.0 %)
Notificaciones: disponible siempre (API 26)
PhotoPicker: comprobar Build.VERSION.SDK_INT >= 33
(no disponible en el 48.0 % de los dispositivos)
PermisoNotificaciones: comprobar
Build.VERSION.SDK_INT >= 33 (no disponible en el
48.0 % de los dispositivos)
```

119. Las diferencias de una lista, como DiffUtil

Medio · Desarrollo de Aplicaciones Móviles · apuntesdam.com/subject/prog-multimedia/topic/desarrollo-aplicaciones-movil

Para no redibujar una lista entera (RecyclerView o LazyColumn) cada vez que cambian los datos, Android calcula las diferencias entre la lista vieja y la nueva (DiffUtil, ListAdapter). La primera línea es la lista antigua y la segunda la nueva, como elementos «id:contenido» separados por espacios (cualquiera puede estar vacía). Muestra, primero, «notifyItemRemoved: id» por cada elemento que ya no está (en el orden de la lista vieja); después, recorriendo la lista nueva, «notifyItemInserted(posición): id» para los nuevos y «notifyItemChanged(posición): id viejo → nuevo» para los que cambian de contenido. Termina con «Sin cambios: A · Insertados: B · Borrados: C · Modificados: D» (añadiendo « · Hay elementos movidos» si los elementos que siguen en ambas listas cambian de orden relativo) y con «No hace falta redibujar» o «Se redibujan N de M filas (notifyDataSetChanged redibujaría M)», donde N son insertados más modificados.

Código de partida (java)

```
import java.util.*;

public class Main {
    record Elemento(String id, String contenido) { }

    static List<Elemento> leer(String linea) {
        List<Elemento> l = new ArrayList<>();
        if (linea.isBlank()) return l;
        for (String par : linea.trim().split(" ")) {
            String[] p = par.split(":");
            l.add(new Elemento(p[0], p[1]));
        }
        return l;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<Elemento> antes = leer(sc.nextLine()), despues = leer(sc.hasNextLine() ? sc.nextLine() : "");
        // TODO: calcula los cambios entre las dos listas como DiffUtil
    }
}
```

Ejemplo: Lista de tareas

ENTRADA

```
1:Pan 2:Leche 3:Huevos 4:Fruta
1:Pan 3:Huevos(12) 5:Café 4:Fruta
```

SALIDA ESPERADA

```
notifyItemRemoved: 2
notifyItemChanged(1): 3 Huevos → Huevos(12)
notifyItemInserted(2): 5
Sin cambios: 2 · Insertados: 1 · Borrados: 1 ·
Modificados: 1
Se redibujan 2 de 4 filas (notifyDataSetChanged
redibujaría 4)
```

Ejemplo: Reordenada

ENTRADA

```
a:x b:y c:z
c:z a:x b:y
```

SALIDA ESPERADA

```
Sin cambios: 3 · Insertados: 0 · Borrados: 0 ·
Modificados: 0 · Hay elementos movidos
Se redibujan 0 de 3 filas (notifyDataSetChanged
redibujaría 3)
```

Ejemplo: Sin cambios

ENTRADA

```
a:1 b:2
a:1 b:2
```

SALIDA ESPERADA

```
Sin cambios: 2 · Insertados: 0 · Borrados: 0 ·
Modificados: 0
No hace falta redibujar
```

120. La pila de navegación de una app

Difícil · Desarrollo de Aplicaciones Móviles · apuntesdam.com/subject/prog-multimedia/topic/desarrollo-aplicaciones-movil

El componente Navigation de Android mantiene una pila de pantallas (back stack). La primera línea es «inicio DESTINO»; después vienen órdenes «navigate DESTINO [popUpTo=X] [inclusive] [singleTop]» y «back». navigate apila el destino; con popUpTo=X, antes se desapilan las pantallas hasta dejar X en la cima (con inclusive, también X), lo que se usa por ejemplo tras iniciar sesión para que «atrás» no vuelva al acceso; con singleTop, si el destino ya está en la cima no se apila otra vez («orden: ya estás en DESTINO (singleTop)»). Si X no está en la pila, «ERROR: X no está en la pila» y no se hace nada. back desapila; si solo queda una pantalla, «back: se cierra la app desde DESTINO» y se termina. Tras cada orden que cambia la pila (y tras un back que no cierra), muestra la orden y «→ [base > ... > cima]».

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Deque<String> pila = new ArrayDeque<>(); // la cima es la pantalla visible
        pila.push(sc.nextLine().trim().split(" ")[1]); // inicio DESTINO
        // TODO: procesa navigate y back como el componente Navigation
    }
}
```

Ejemplo: Inicio de sesión

ENTRADA

```
inicio splash
navigate login popUpTo=splash inclusive
navigate inicio popUpTo=login inclusive
navigate detalle
navigate detalle singleTop
navigate carrito
back
back
back
```

SALIDA ESPERADA

```
navigate login popUpTo=splash inclusive → [login]
navigate inicio popUpTo=login inclusive → [inicio]
navigate detalle → [inicio > detalle]
navigate detalle singleTop: ya estás en detalle
(singleTop)
navigate carrito → [inicio > detalle > carrito]
back → [inicio > detalle]
back → [inicio]
back: se cierra la app desde inicio
```

121. Sincronización sin conexión con conflictos

Muy difícil · Desarrollo de Aplicaciones Móviles · apuntesdam.com/subject/prog-multimedia/topic/desarrollo-aplicaciones-movil

Una app «offline first» guarda los cambios hechos sin conexión y los envía al recuperarla. La entrada tiene tres secciones: [servidor] con el estado actual «id valor tiempo», [base] con la versión que tenía el móvil cuando se quedó sin conexión, y [cambios] con las operaciones pendientes «crear id valor tiempo», «editar id valor tiempo» o «borrar id tiempo» (los valores no tienen espacios). Para cada cambio, en orden: crear algo que ya existe → «CONFLICTO id: ya existe en el servidor (se conserva «valor»)»; editar o borrar algo que ya no existe → «IGNORADO id: ya no existe en el servidor»; si el elemento ha cambiado en el servidor desde la base (distinto tiempo) y el cambio del servidor es más reciente → «CONFLICTO id: gana el servidor «valor» (tiempo) frente al cambio local (tiempo)» (estrategia «gana el último»); en otro caso se aplica: «CREADO id = «valor»», «BORRADO id» o «EDITADO id = «valor»», añadiendo « (gana el cambio local, más reciente)» o « (sobre un cambio del servidor más antiguo)» si había cambio en el servidor. Termina con «Servidor tras sincronizar:», cada elemento « id = «valor» (tiempo)» por orden de id y «Aplicados: A · Conflictos: C».

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, String[]> servidor = new TreeMap<>(), local = new TreeMap<>(); // id → {valor, marca de tiempo}
        List<String[]> pendientes = new ArrayList<>(); // cambios hechos sin conexión
        String seccion = "";
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (l.isEmpty()) continue;
            if (l.startsWith("(")) { seccion = l; continue; }
            String[] p = l.split(" ");
            switch (seccion) {
                case "[servidor]" -> servidor.put(p[0], new String[]{p[1], p[2]});
                case "[base]" -> local.put(p[0], new String[]{p[1], p[2]});
                case "[cambios]" -> pendientes.add(p); // op id valor tiempo (op: crear, editar, borrar)
                default -> { }
            }
        }
        // TODO: aplica los cambios pendientes al servidor resolviendo conflictos y muestra el resultado
    }
}
```

Ejemplo: Tareas

ENTRADA

```
[servidor]
t1 Comprar_pan 100
t2 Llamar_al_banco 250
t3 Entregar_la_práctica 120
[base]
t1 Comprar_pan 100
t2 Llamar_al_banco 110
t3 Entregar_la_práctica 120
[cambios]
editar t1 Comprar_pan_integral 300
editar t2 Llamar_a_las_9 200
crear t4 Recoger_paquete 210
borrar t3 220
crear t1 Duplicada 230
```

SALIDA ESPERADA

```
EDITADO t1 = «Comprar_pan_integral»
CONFLICTO t2: gana el servidor «Llamar_al_banco» (250) frente al cambio local (200)
CREADO t4 = «Recoger_paquete»
BORRADO t3
CONFLICTO t1: ya existe en el servidor (se conserva «Comprar_pan_integral»)
Servidor tras sincronizar:
    t1 = «Comprar_pan_integral» (300)
    t2 = «Llamar_al_banco» (250)
    t4 = «Recoger_paquete» (210)
Aplicados: 3 · Conflictos: 2
```

122. Física con paso fijo, acumulador e interpolación

Medio · Motores de Juegos · apuntesdam.com/subject/prog-multimedia/topic/motores-juegos

Los motores dibujan a la frecuencia que pueden, pero calculan la física en pasos fijos (FixedUpdate en Unity, `_physics_process` en Godot). La entrada tiene «paso P» (ms de cada paso de física), «velocidad V» (px/s de un objeto que avanza en x desde 0) y una línea con la duración en ms de cada fotograma. Para cada fotograma: suma su duración a un acumulador (como máximo 250 ms por fotograma, para que un parón no obligue a simular cientos de pasos), ejecuta pasos de física mientras el acumulador llegue a P (guardando la posición anterior y avanzando $V \cdot P / 1000$) y dibuja interpolando entre la posición anterior y la actual con $\alpha = \text{acumulador} / P$. Muestra «Fi dt=D[(limitado a 250)]: N paso(s), sobran A ms, x=X, se dibuja en Y (alfa Z)» con un decimal en las posiciones y dos en alfa, y al final «Fotogramas: F · Pasos de física: N · Tiempo simulado: T ms».

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int paso = Integer.parseInt(sc.nextLine().trim().split(" ")[1]);           // «paso 20» (ms)
        double velocidad = Double.parseDouble(sc.nextLine().trim().split(" ")[1]); // «velocidad 120»
        (px/s)
        String[] deltas = sc.nextLine().trim().split(" ");                       // ms de cada
        fotograma
        final int MAXIMO = 250;                                                  // evita la «espiral
        de la muerte»
        double x = 0, xAnterior = 0;
        int acumulado = 0, totalPasos = 0;
        // TODO: bucle con paso fijo, acumulador e interpolación
    }
}
```

Ejemplo: 60 FPS con tirones

ENTRADA

```
paso 20
velocidad 120
16 16 17 16 50 16 300 8
```

SALIDA ESPERADA

```
F1 dt=16: 0 pasos, sobran 16 ms, x=0.0, se dibuja
en 0.0 (alfa 0.80)
F2 dt=16: 1 paso, sobran 12 ms, x=2.4, se dibuja
en 1.4 (alfa 0.60)
F3 dt=17: 1 paso, sobran 9 ms, x=4.8, se dibuja en
3.5 (alfa 0.45)
F4 dt=16: 1 paso, sobran 5 ms, x=7.2, se dibuja en
5.4 (alfa 0.25)
F5 dt=50: 2 pasos, sobran 15 ms, x=12.0, se dibuja
en 11.4 (alfa 0.75)
F6 dt=16: 1 paso, sobran 11 ms, x=14.4, se dibuja
en 13.3 (alfa 0.55)
F7 dt=300 (limitado a 250): 13 pasos, sobran 1 ms,
x=45.6, se dibuja en 43.3 (alfa 0.05)
F8 dt=8: 0 pasos, sobran 9 ms, x=45.6, se dibuja
en 44.3 (alfa 0.45)
Fotogramas: 8 · Pasos de física: 19 · Tiempo
simulado: 380 ms
```

123. Colisiones con rejilla espacial

Difícil · Motores de Juegos · apuntesdam.com/subject/prog-multimedia/topic/motores-juegos

Comprobar todos los objetos contra todos cuesta $n \cdot (n-1)/2$ comprobaciones: con 1000 balas y enemigos, medio millón por fotograma. Los motores primero descartan los pares lejanos (fase amplia) con estructuras como una rejilla espacial. La primera línea es «celda C» y las siguientes cajas «nombre x y ancho alto» (enteros, y hacia abajo). Calcula los choques por fuerza bruta (AABB, ya dado en chocan) y con la rejilla: cada caja se apunta en todas las celdas que ocupa, de $\text{floorDiv}(x, C)$ a $\text{floorDiv}(x+\text{ancho}-1, C)$ en x e igual en y; solo se comprueban los pares distintos que comparten al menos una celda. Muestra «Celdas ocupadas: K · La más llena: (cx,cy) con M» (la primera creada si hay empate), «Choques: a-b c-d...» (cada par en el orden de entrada, ordenados por la primera caja y luego por la segunda; «ninguno» si no hay), «Comprobaciones: fuerza bruta N · rejilla R (ahorro P %)» (P redondeado; 0 si N es 0) y «Mismo resultado: sí».

Código de partida (java)

```
import java.util.*;

public class Main {
    record Caja(String nombre, int x, int y, int ancho, int alto) { }

    static boolean chocan(Caja a, Caja b) {
        return a.x() < b.x() + b.ancho() && a.x() + a.ancho() > b.x()
            && a.y() < b.y() + b.alto() && a.y() + a.alto() > b.y();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int celda = Integer.parseInt(sc.nextLine().trim().split(" ")[1]); // «celda 64»
        List<Caja> cajas = new ArrayList<>();
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");
            if (p.length < 5) continue;
            cajas.add(new Caja(p[0], Integer.parseInt(p[1]), Integer.parseInt(p[2]), Integer.parseInt(p[3]),
                Integer.parseInt(p[4])));
        }
        // TODO: fuerza bruta frente a rejilla espacial
    }
}
```

Ejemplo: Nivel de plataformas

ENTRADA

```
celda 64
jugador 40 300 24 32
suelo 0 330 640 20
moneda1 60 300 12 12
moneda2 400 200 12 12
enemigo 380 196 30 30
bala 600 40 6 6
plataforma 360 226 120 16
```

SALIDA ESPERADA

```
Celdas ocupadas: 16 · La más llena: (6,3) con 3
Choques: jugador-suelo jugador-moneda1 moneda2-
enemigo
Comprobaciones: fuerza bruta 21 · rejilla 5
(ahorro 76 %)
Mismo resultado: sí
```

Ejemplo: Coordenadas negativas

ENTRADA

```
celda 50
a -30 -30 40 40
b 0 0 10 10
c 200 200 10 10
```

SALIDA ESPERADA

```
Celdas ocupadas: 5 · La más llena: (0,0) con 2
Choques: a-b
Comprobaciones: fuerza bruta 3 · rejilla 1 (ahorro
67 %)
Mismo resultado: sí
```

124. El efecto túnel: detección discreta frente a continua

Muy difícil · Motores de Juegos · apuntesdam.com/subject/prog-multimedia/topic/motores-juegos

Si una bala avanza 25 píxeles por fotograma y la pared mide 10, puede estar delante en un fotograma y detrás en el siguiente sin que nunca se detecte el choque (efecto túnel). Por eso los motores ofrecen detección continua (Continuous en el Rigidbody de Unity, CCD en Godot). La entrada tiene «pared x y ancho alto», «fps F» y balas «nombre x y vx vy» (velocidades en px/s, y hacia abajo). Simula 1 segundo. Discreta: en cada fotograma $k = 1 \dots F$ la bala está en $(x + vx \cdot k/F, y + vy \cdot k/F)$ y choca si está estrictamente dentro de la pared. Continua: calcula con el método de las franjas el instante $t \in [0, 1]$ en que el segmento recorrido entra en la pared (si la velocidad en un eje es 0, la bala debe estar estrictamente dentro de la franja de ese eje; si no, no hay impacto). Muestra «nombre: discreta → choca en el fotograma K ($t=T \cdot s$) | no detecta nada | continua → impacto en $t=T \cdot s$ en (X, Y) por la cara izquierda/derecha/superior/inferior | no hay impacto | empieza dentro de la pared», añadiendo « ¡ATRAVIESA LA PARED! » si la continua ve un impacto (sin empezar dentro) que la discreta no detecta; al final «Efecto túnel: N de M balas». Tiempos con tres decimales y posiciones con uno.

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] m = sc.nextLine().trim().split(" "); // «pared x y ancho alto»
        double px = Double.parseDouble(m[1]), py = Double.parseDouble(m[2]);
        double pw = Double.parseDouble(m[3]), ph = Double.parseDouble(m[4]);
        int fps = Integer.parseInt(sc.nextLine().trim().split(" ")[1]); // «fps 60»
        int tuneles = 0, balas = 0;
        while (sc.hasNextLine()) {
            String[] b = sc.nextLine().trim().split(" ");
            if (b.length < 5) continue;
            String nombre = b[0];
            double x0 = Double.parseDouble(b[1]), y0 = Double.parseDouble(b[2]);
            double vx = Double.parseDouble(b[3]), vy = Double.parseDouble(b[4]);
            balas++;
            // TODO: detección discreta frente a continua (barrido)
        }
        System.out.println("Efecto túnel: " + tuneles + " de " + balas + " balas");
    }
}
```

Ejemplo: Pared fina

ENTRADA

```
pared 300 0 10 200
fps 60
lenta 5 100 600 0
rapida 0 50 1500 0
vertical 305 -50 0 400
diagonal 100 250 600 -300
falla 0 300 900 0
```

SALIDA ESPERADA

```
lenta: discreta → choca en el fotograma 30
(t=0.500 s) | continua → impacto en t=0.492 s en
(300.0, 100.0) por la cara izquierda
rapida: discreta → no detecta nada | continua →
impacto en t=0.200 s en (300.0, 50.0) por la cara
izquierda ¡ATRAVIESA LA PARED!
vertical: discreta → choca en el fotograma 8
(t=0.133 s) | continua → impacto en t=0.125 s en
(305.0, 0.0) por la cara superior
diagonal: discreta → no detecta nada | continua →
impacto en t=0.333 s en (300.0, 150.0) por la cara
izquierda ¡ATRAVIESA LA PARED!
falla: discreta → no detecta nada | continua → no
hay impacto
Efecto túnel: 2 de 5 balas
```

125. ¿Quién gana al tres en raya?

Fácil · Ejercicios de Programación Multimedia y Dispositivos Móviles · apuntesdam.com/subject/programa-multimedia/topic/ejercicios-pmdm

La entrada son 3 líneas con el tablero (X, O o . para las casillas vacías). Muestra «Gana X», «Gana O», «Empate» (tablero lleno sin ganador) o «Sigue la partida».

Código de partida (java)

```
import java.util.*;

public class Main {

    static String resultado(char[][] t) {
        // TODO
        return "Sigue la partida";
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        char[][] t = new char[3][3];
        for (int i = 0; i < 3; i++) t[i] = sc.next().toCharArray();
        System.out.println(resultado(t));
    }
}
```

Ejemplo: Gana X en diagonal

ENTRADA

```
X.O
OX.
..X
```

SALIDA ESPERADA

Gana X

Ejemplo: Empate

ENTRADA

```
XOX
XOO
OXX
```

SALIDA ESPERADA

Empate

Ejemplo: Sigue

ENTRADA

```
X..
.O.
...
```

SALIDA ESPERADA

Sigue la partida

126. Una generación del juego de la vida

Medio · Ejercicios de Programación Multimedia y Dispositivos Móviles · apuntesdam.com/subject/prog-multimedia/topic/ejercicios-pmdm

La primera línea tiene filas y columnas; después, el tablero con # (célula viva) y . (muerta). Calcula la siguiente generación del juego de la vida de Conway y muéstrala: una célula viva con 2 o 3 vecinas vivas sobrevive; una muerta con exactamente 3 vecinas vivas nace; el resto muere o sigue muerta. Los bordes no se conectan.

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int f = sc.nextInt(), c = sc.nextInt();
        char[][] t = new char[f][];
        for (int i = 0; i < f; i++) t[i] = sc.next().toCharArray();
        // TODO: calcula y muestra la siguiente generación
        for (char[] fila : t) System.out.println(new String(fila));
    }
}
```

Ejemplo: Parpadeador

ENTRADA

```
5 5
.....
..#..
..#..
..#..
.....
```

SALIDA ESPERADA

```
.....
.....
.###.
.....
.....
```

Ejemplo: Bloque estable

ENTRADA

```
4 4
....
.##.
.##.
....
```

SALIDA ESPERADA

```
....
.##.
.##.
....
```

127. Puntuación con combos

Medio · Ejercicios de Programación Multimedia y Dispositivos Móviles · apuntesdam.com/subject/prog-multimedia/topic/ejercicios-pmdm

Por la entrada llega la secuencia de eventos de una partida, uno por línea: MONEDA (10 puntos), ENEMIGO (50 puntos), GOLPE (pierdes una vida y el combo vuelve a 0). Cada MONEDA o ENEMIGO seguidos aumentan el combo en 1 y los puntos del evento se multiplican por el multiplicador actual: 1 con combo 0-4, 2 con combo 5-9 y 3 con combo 10 o más (el combo se incrementa antes de calcular). Empiezas con 3 vidas; con 0 vidas la partida termina y se ignoran los eventos siguientes. Muestra «Puntos: N», «Vidas: V» y «Mejor combo: C».

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int puntos = 0, vidas = 3, combo = 0, mejor = 0;
        while (sc.hasNext()) {
            String evento = sc.next();
            // TODO
        }
        System.out.println("Puntos: " + puntos);
        System.out.println("Vidas: " + vidas);
        System.out.println("Mejor combo: " + mejor);
    }
}
```

Ejemplo: Combo hasta x2

ENTRADA

MONEDA
MONEDA
MONEDA
MONEDA
MONEDA
ENEMIGO

SALIDA ESPERADA

Puntos: 160
Vidas: 3
Mejor combo: 6

Ejemplo: Golpe reinicia el combo

ENTRADA

ENEMIGO
GOLPE
MONEDA

SALIDA ESPERADA

Puntos: 60
Vidas: 2
Mejor combo: 1

128. Camino más corto en un laberinto

Difícil · Ejercicios de Programación Multimedia y Dispositivos Móviles · apuntesdam.com/subject/prog-multimedia/topic/ejercicios-pmdm

La IA de un enemigo necesita saber cuántos pasos hay hasta el jugador. La primera línea tiene filas y columnas; después el mapa con # (muro), . (suelo), E (enemigo) y J (jugador). Moviéndose en 4 direcciones, muestra el número mínimo de pasos de E a J, o «Sin camino» si no se puede llegar.

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int f = sc.nextInt(), c = sc.nextInt();
        char[][] m = new char[f][];
        for (int i = 0; i < f; i++) m[i] = sc.next().toCharArray();
        // TODO: BFS desde E hasta J
        System.out.println("Sin camino");
    }
}
```

Ejemplo: Laberinto sencillo

ENTRADA

```
3 5
E..#.
.#...
...#J
```

SALIDA ESPERADA

6

Ejemplo: Camino directo

ENTRADA

```
1 4
E..J
```

SALIDA ESPERADA

3

Ejemplo: Sin camino

ENTRADA

```
3 3
E#.
##.
..J
```

SALIDA ESPERADA

Sin camino

Programación de Servicios y Procesos

129. Coordinador de procesos

Difícil · Programación Concurrente · apuntesdam.com/subject/prog-servicios/topic/concurrencia

El programa puede funcionar en dos modos. Si recibe los argumentos «trabajador desde hasta», es un proceso hijo: cuenta los primos de ese tramo y escribe el número (ya está hecho). Si no, es el coordinador: lee un límite y un número de procesos, reparte el intervalo [1, límite] en tantos tramos iguales como procesos (el último se queda con el resto) y lanza un proceso hijo por tramo con ProcessBuilder, ejecutando esta misma clase. Primero lanza todos y después lee la salida de cada uno en orden y espera a que termine. Muestra «Proceso i [desde, hasta]: n primos» y, al final, «Total: N primos hasta límite».

Código de partida (java)

```
import java.io.*;
import java.util.*;

public class Main {
    static boolean esPrimo(long n) {
        if (n < 2) return false;
        for (long d = 2; d * d <= n; d++) if (n % d == 0) return false;
        return true;
    }

    static long contar(long desde, long hasta) {
        long c = 0;
        for (long i = desde; i <= hasta; i++) if (esPrimo(i)) c++;
        return c;
    }

    public static void main(String[] args) throws Exception {
        // Modo trabajador: el proceso hijo cuenta su tramo y escribe el resultado
        if (args.length == 3 && args[0].equals("trabajador")) {
            System.out.println(contar(Long.parseLong(args[1]), Long.parseLong(args[2])));
            return;
        }
        Scanner sc = new Scanner(System.in);
        long limite = sc.nextLong();
        int procesos = sc.nextInt();
        // TODO: repartir [1, limite] en tramos, lanzar un proceso hijo por tramo,
        // leer su resultado y mostrar el resumen
    }
}
```

Ejemplo: 10 000 con 4 procesos

ENTRADA

10000 4

SALIDA ESPERADA

Proceso 1 [1, 2500]: 367 primos
Proceso 2 [2501, 5000]: 302 primos
Proceso 3 [5001, 7500]: 281 primos
Proceso 4 [7501, 10000]: 279 primos
Total: 1229 primos hasta 10000

Ejemplo: Reparto con resto

ENTRADA

100 3

SALIDA ESPERADA

Proceso 1 [1, 33]: 11 primos
Proceso 2 [34, 66]: 7 primos
Proceso 3 [67, 100]: 7 primos
Total: 25 primos hasta 100

130. Planificador de CPU: FCFS, SJF y Round Robin

Muy difícil · Programación Concurrente · apuntesdam.com/subject/prog-servicios/topic/concurrencia

El sistema operativo decide qué proceso usa la CPU en cada momento. La entrada es el quantum y, después, cada proceso con su identificador (una letra), su instante de llegada y su ráfaga de CPU. Implementa los tres algoritmos devolviendo la línea de tiempo: un carácter por unidad de tiempo con la letra del proceso que se ejecuta, o «-» si la CPU está libre. FCFS ejecuta los procesos enteros por orden de llegada (a igualdad, en el orden de la entrada). SJF, no expropiativo, cada vez que la CPU queda libre elige, de los que ya han llegado, el de ráfaga más corta (a igualdad, el primero de la entrada). Round Robin da turnos de como mucho quantum unidades con una cola FIFO: los procesos que llegan durante un turno entran en la cola antes que el proceso que acaba de ser expulsado. El método informe, ya hecho, calcula el tiempo de espera de cada proceso (fin - llegada - ráfaga) y la media.

Código de partida (java)

```
import java.util.*;

public class Main {
    record Proceso(String id, int llegada, int rafaga) { }

    // Devuelve la línea de tiempo: una letra (id) por unidad de tiempo, o '-' si la CPU está libre
    static String fcfs(List<Proceso> ps) {
        // TODO: el primero que llega se ejecuta entero
        return "";
    }

    static String sjf(List<Proceso> ps) {
        // TODO: no expropiativo; de los que han llegado, el de ráfaga más corta
        return "";
    }

    static String roundRobin(List<Proceso> ps, int quantum) {
        // TODO: turnos de como mucho quantum unidades
        return "";
    }

    // Calcula y muestra los tiempos de espera a partir de la línea de tiempo
    static void informe(String nombre, String linea, List<Proceso> ps) {
        StringBuilder sb = new StringBuilder(nombre + ": " + linea + " |");
        double suma = 0;
        for (Proceso p : ps) {
            int fin = linea.lastIndexOf(p.id().charAt(0)) + 1;
            int espera = fin - p.llegada() - p.rafaga();
            suma += espera;
            sb.append(" ").append(p.id()).append("=").append(espera);
        }
        sb.append(String.format(Locale.US, " | media %.2f", suma / ps.size()));
        System.out.println(sb);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int quantum = sc.nextInt();
        List<Proceso> ps = new ArrayList<>();
        while (sc.hasNext()) ps.add(new Proceso(sc.next(), sc.nextInt(), sc.nextInt()));
        informe("FCFS", fcfs(ps), ps);
        informe("SJF ", sjf(ps), ps);
        informe("RR ", roundRobin(ps, quantum), ps);
    }
}
```

Ejemplo: Ejemplo de clase

ENTRADA

```
2
A 0 5
B 1 3
C 2 1
D 3 2
```

SALIDA ESPERADA

```
FCFS: AAAAABBBCCDD | A=0 B=4 C=6 D=6 | media 4.00
SJF : AAAAACDDBBB | A=0 B=7 C=3 D=3 | media 3.25
RR : AABBCAADDDBA | A=6 B=6 C=2 D=4 | media 4.50
```

Ejemplo: CPU libre al principio

ENTRADA

```
3
A 2 4
B 3 2
C 10 1
```

SALIDA ESPERADA

```
FCFS: --AAAABB--C | A=0 B=3 C=0 | media 1.00
SJF : --AAAABB--C | A=0 B=3 C=0 | media 1.00
RR : --AAABBA--C | A=2 B=2 C=0 | media 1.33
```

131. Suma en paralelo con hilos

Difícil · Hilos y Multithreading en Java · apuntesdam.com/subject/prog-servicios/topic/hilos

La primera cifra de la entrada es el número de hilos y el resto, los números que hay que sumar. Reparte el array en tantos bloques consecutivos como hilos, lo más iguales posible: si no es divisible, los primeros bloques tienen un elemento más (con 10 números y 3 hilos: 4, 3 y 3). Cada hilo suma su bloque y guarda el resultado en su posición de parciales. El main debe esperar a que terminen todos antes de mostrar los resultados (esa parte ya está hecha). Un hilo puede quedarse sin elementos: entonces su parcial es 0.

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) throws InterruptedException {
        Scanner sc = new Scanner(System.in);
        int hilos = sc.nextInt();
        List<Long> lista = new ArrayList<>();
        while (sc.hasNextLong()) lista.add(sc.nextLong());
        long[] datos = lista.stream().mapToLong(Long::longValue).toArray();

        long[] parciales = new long[hilos]; // cada hilo escribe solo en su posición
        // TODO: crear los hilos, repartir el array en bloques, arrancarlos y esperar a que terminen

        long total = 0;
        for (int i = 0; i < hilos; i++) {
            System.out.println("Hilo " + i + ": " + parciales[i]);
            total += parciales[i];
        }
        System.out.println("Total: " + total);
    }
}
```

Ejemplo: 3 hilos, 10 números

ENTRADA

```
3
1 2 3 4 5 6 7 8 9 10
```

SALIDA ESPERADA

```
Hilo 0: 10
Hilo 1: 18
Hilo 2: 27
Total: 55
```

Ejemplo: Más hilos que números

ENTRADA

```
4
100 200 300
```

SALIDA ESPERADA

```
Hilo 0: 100
Hilo 1: 200
Hilo 2: 300
Hilo 3: 0
Total: 600
```

132. Productor-consumidor con un búfer acotado

Muy difícil · Hilos y Multithreading en Java · apuntesdam.com/subject/prog-servicios/topic/hilos

Varios productores dejan números en un búfer de capacidad limitada y un consumidor los va sacando. Implementa los métodos sincronizados de BufferAcotado con wait y notifyAll: poner espera mientras el búfer está lleno y sacar espera mientras está vacío; los dos avisan a los demás hilos cuando cambian el estado. Usa un array circular (inicio y cantidad) para que sacar sea inmediato. El main, ya hecho, lanza los productores, espera a que terminen, mete un FIN para el consumidor y muestra cuántos elementos de cada productor se consumieron y la suma: si no se pierde ni se repite ningún elemento, cada productor aporta todos los suyos.

Código de partida (java)

```
import java.util.*;

class BufferAcotado {
    private final int[] datos;
    private int inicio = 0, cantidad = 0;

    BufferAcotado(int capacidad) {
        datos = new int[capacidad];
    }

    // Espera mientras el búfer está lleno
    public synchronized void poner(int v) throws InterruptedException {
        // TODO
    }

    // Espera mientras el búfer está vacío
    public synchronized int sacar() throws InterruptedException {
        // TODO
        return -1;
    }
}

public class Main {
    static final int FIN = -1; // «píldora venenosa»: avisa al consumidor de que no hay
                               // más

    public static void main(String[] args) throws InterruptedException {
        Scanner sc = new Scanner(System.in);
        int capacidad = sc.nextInt(), productores = sc.nextInt(), porProductor = sc.nextInt();
        BufferAcotado buffer = new BufferAcotado(capacidad);
        int[] consumidosDe = new int[productores];
        long[] suma = new long[1];

        List<Thread> hilos = new ArrayList<>();
        for (int p = 0; p < productores; p++) {
            final int id = p;
            hilos.add(new Thread(() -> {
                try {
                    // Cada valor codifica su productor: id * 1000 + número
                    for (int k = 1; k <= porProductor; k++) buffer.poner(id * 1000 + k);
                } catch (InterruptedException e) { Thread.currentThread().interrupt(); }
            }));
        }
        Thread consumidor = new Thread(() -> {
            try {
                int v;
                while ((v = buffer.sacar()) != FIN) {
                    consumidosDe[v / 1000]++;
                    suma[0] += v % 1000;
                }
            } catch (InterruptedException e) { Thread.currentThread().interrupt(); }
        });
        consumidor.start();
        for (Thread h : hilos) h.start();
        for (Thread h : hilos) h.join();
        buffer.poner(FIN);
        consumidor.join();

        for (int p = 0; p < productores; p++) System.out.println("Productor " + p + ": " + consumidosDe[p] +
            " elementos consumidos");
        System.out.println("Suma de los números: " + suma[0]);
    }
}
```

Ejemplo: Capacidad 2, 3 productores

ENTRADA

2 3 50

SALIDA ESPERADA

Productor 0: 50 elementos consumidos
Productor 1: 50 elementos consumidos
Productor 2: 50 elementos consumidos
Suma de los números: 3825

Ejemplo: Capacidad 1

ENTRADA

1 4 25

SALIDA ESPERADA

Productor 0: 25 elementos consumidos
Productor 1: 25 elementos consumidos
Productor 2: 25 elementos consumidos
Productor 3: 25 elementos consumidos
Suma de los números: 1300

133. Servidor de un protocolo de texto

Difícil · Sockets en Java · apuntesdam.com/subject/prog-servicios/topic/sockets

El main ya arranca un servidor TCP en un puerto libre y un cliente que le envía cada línea de la entrada y muestra la respuesta. Implementa atender, que gestiona la conexión de un cliente siguiendo este protocolo (una orden por línea, una respuesta por línea): «SUMA n1 n2 ...» responde «OK total» (o «ERROR números no válidos»); «MAYUS texto» responde «OK TEXTO»; «INVERTIR texto» responde el texto al revés; «CONTAR» responde «OK n», con el número de órdenes atendidas antes en esta conexión (todas, incluidas las erróneas y otros CONTAR); «ADIOS» responde «OK hasta luego» y cierra. Las órdenes no distinguen mayúsculas y cualquier otra da «ERROR orden desconocida: X». Usa UTF-8 en los dos sentidos.

Código de partida (java)

```
import java.io.*;
import java.net.*;
import java.util.*;

public class Main {
    // Atiende a UN cliente: lee órdenes línea a línea y responde a cada una
    static void atender(Socket cliente) throws IOException {
        // TODO: leer cada línea, responder según el protocolo y terminar con ADIOS
    }

    public static void main(String[] args) throws Exception {
        try (ServerSocket servidor = new ServerSocket(0)) { // puerto libre cualquiera
            Thread hiloServidor = new Thread(() -> {
                try (Socket c = servidor.accept()) {
                    atender(c);
                } catch (IOException e) {
                    System.out.println("Error del servidor: " + e.getMessage());
                }
            });
            hiloServidor.start();

            // Cliente: envía cada línea de la entrada y muestra la respuesta
            Scanner sc = new Scanner(System.in);
            try (Socket s = new Socket("localhost", servidor.getLocalPort());
                PrintWriter out = new PrintWriter(new OutputStreamWriter(s.getOutputStream(), "UTF-8"),
                    true);
                BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream(), "UTF-8")))
            {
                while (sc.hasNextLine()) {
                    String orden = sc.nextLine();
                    out.println(orden);
                    String r;
                    try {
                        r = in.readLine();
                    } catch (IOException e) {
                        r = null; // el servidor cerró la conexión
                    }
                    System.out.println("> " + orden);
                    System.out.println("< " + (r == null ? "(conexión cerrada)" : r));
                    if (r == null || orden.equals("ADIOS")) break;
                }
            }
            hiloServidor.join();
        }
    }
}
```

Ejemplo: Órdenes básicas

ENTRADA

```
SUMA 2 3 4
MAYUS hola mundo
INVERTIR socket
CONTAR
ADIOS
```

SALIDA ESPERADA

```
> SUMA 2 3 4
< OK 9
> MAYUS hola mundo
< OK HOLA MUNDO
> INVERTIR socket
< OK tekcos
> CONTAR
< OK 3
> ADIOS
< OK hasta luego
```

Ejemplo: Errores

ENTRADA

```
suma 1 dos  
BORRAR todo  
CONTAR  
adios
```

SALIDA ESPERADA

```
> suma 1 dos  
< ERROR números no válidos  
> BORRAR todo  
< ERROR orden desconocida: BORRAR  
> CONTAR  
< OK 2  
> adios  
< OK hasta luego
```

134. Mensajes fiables sobre UDP

Muy difícil · Sockets en Java · apuntesdam.com/subject/prog-servicios/topic/sockets

UDP no garantiza que los paquetes lleguen. El servidor de este reto lo simula: pierde uno de cada cuatro paquetes que recibe y una de cada tres respuestas. Implementa `enviarFiable` con el protocolo de parada y espera (stop-and-wait) que usan TCP y muchos juegos: envía el datagrama «seq:texto», espera como mucho `ESPERA_MS` milisegundos un datagrama «ACK seq» y, si no llega, reenvía el mismo mensaje, hasta `MAX_INTENTOS` veces. Devuelve el número de intentos que han hecho falta, o `-1`. Ignora cualquier ACK de otro número de secuencia. El servidor descarta los mensajes duplicados (cuando se pierde un ACK, el cliente reenvía algo que ya había llegado) gracias al número de secuencia, así que al final debe haber recibido todos los mensajes, en orden y una sola vez.

Código de partida (java)

```
import java.io.*;
import java.net.*;
import java.nio.charset.StandardCharsets;
import java.util.*;

public class Main {
    static final int ESPERA_MS = 150, MAX_INTENTOS = 5;

    // ---- Servidor (ya hecho): pierde paquetes a propósito para simular una red mala
    static class Servidor extends Thread {
        final DatagramSocket ds;
        final List<String> recibidos = new ArrayList<>();
        int paquetes = 0, duplicados = 0, ultimoSeq = 0;

        Servidor() throws SocketException { ds = new DatagramSocket(0); }

        public void run() {
            byte[] buf = new byte[1024];
            try {
                while (true) {
                    DatagramPacket p = new DatagramPacket(buf, buf.length);
                    ds.receive(p);
                    paquetes++;
                    if (paquetes % 4 == 0) continue; // se pierde el paquete que llega
                    String msg = new String(p.getData(), 0, p.getLength(), StandardCharsets.UTF_8);
                    int seq = Integer.parseInt(msg.substring(0, msg.indexOf(':')));
                    String datos = msg.substring(msg.indexOf(':') + 1);
                    if (seq == ultimoSeq) duplicados++; // ya lo teníamos: era una
                        retransmisión
                    else if (seq == ultimoSeq + 1) { ultimoSeq = seq; if (!datos.equals("FIN"))
                        recibidos.add(datos); }
                    if (paquetes % 3 == 0) continue; // se pierde el ACK de respuesta
                    byte[] ack = ("ACK " + seq).getBytes(StandardCharsets.UTF_8);
                    ds.send(new DatagramPacket(ack, ack.length, p.getSocketAddress()));
                    if (datos.equals("FIN")) return;
                }
            } catch (IOException e) { }
        }
    }

    // Envía un mensaje y espera su ACK; si no llega a tiempo, lo reenvía.
    // Devuelve el número de intentos, o -1 si se agotan.
    static int enviarFiable(DatagramSocket ds, SocketAddress destino, int seq, String texto) throws
        IOException {
        // TODO
        byte[] datos = (seq + ":" + texto).getBytes(StandardCharsets.UTF_8);
        ds.send(new DatagramPacket(datos, datos.length, destino));
        return 1;
    }

    public static void main(String[] args) throws Exception {
        Servidor srv = new Servidor();
        srv.start();
        SocketAddress destino = new InetSocketAddress("localhost", srv.ds.getLocalPort());
        Scanner sc = new Scanner(System.in);
        List<String> mensajes = new ArrayList<>();
        while (sc.hasNextLine()) { String l = sc.nextLine().trim(); if (!l.isEmpty()) mensajes.add(l); }
        mensajes.add("FIN");
        try (DatagramSocket ds = new DatagramSocket()) {
            for (int i = 0; i < mensajes.size(); i++) {
                int intentos = enviarFiable(ds, destino, i + 1, mensajes.get(i));
                System.out.println("Mensaje " + (i + 1) + " «" + mensajes.get(i) + "»: "
                    + (intentos < 0 ? "sin respuesta" : intentos + (intentos == 1 ? " intento" : "
                        intentos")));
            }
        }
        srv.join(2000);
        srv.ds.close();
        System.out.println("El servidor recibió en orden: " + String.join(" | ", srv.recibidos));
        System.out.println("Retransmisiones duplicadas descartadas: " + srv.duplicados);
    }
}
```

Ejemplo: Cuatro mensajes

ENTRADA

hola
que
tal
estás

SALIDA ESPERADA

Mensaje 1 «hola»: 1 intento
Mensaje 2 «que»: 1 intento
Mensaje 3 «tal»: 3 intentos
Mensaje 4 «estás»: 2 intentos
Mensaje 5 «FIN»: 3 intentos
El servidor recibió en orden: hola | que | tal |
estás
Retransmisiones duplicadas descartadas: 3

Ejemplo: Uno solo

ENTRADA

ping

SALIDA ESPERADA

Mensaje 1 «ping»: 1 intento
Mensaje 2 «FIN»: 1 intento
El servidor recibió en orden: ping
Retransmisiones duplicadas descartadas: 0

135. Una API REST con HttpServer

Difícil · Desarrollo de Servicios en Red · apuntesdam.com/subject/prog-servicios/topic/servicios-red-java

Completa manejar para que el servidor ofrezca una API REST de tareas. GET /tareas devuelve 200 y la lista en JSON, ordenada por id: [{"id":1,"texto":"..."}]. POST /tareas con el texto como cuerpo crea una tarea con el siguiente id, responde 201 con la tarea en JSON y la cabecera Location; si el texto está vacío, 400 con {"error":"texto vacío"}. GET /tareas/{id} devuelve 200 con la tarea; DELETE /tareas/{id} la borra y responde 204 sin cuerpo. Si el id no es un número: 400 {"error":"id no válido"}; si no existe: 404 {"error":"no existe la tarea N"}. Cualquier otro método: 405 sin cuerpo. Usa los métodos responder y json, ya hechos. El main envía con HttpClient las peticiones de la entrada y muestra el código y el cuerpo.

Código de partida (java)

```
import com.sun.net.httpserver.*;
import java.io.*;
import java.net.*;
import java.net.http.*;
import java.nio.charset.StandardCharsets;
import java.util.*;
import java.util.concurrent.*;
import java.util.concurrent.atomic.AtomicInteger;

public class Main {
    static final Map<Integer, String> tareas = new ConcurrentSkipListMap<>(); // id → texto, ordenadas por id
    static final AtomicInteger siguienteId = new AtomicInteger(1);

    static void responder(HttpExchange ex, int estado, String cuerpo) throws IOException {
        byte[] b = cuerpo.getBytes(StandardCharsets.UTF_8);
        ex.getResponseHeaders().set("Content-Type", "application/json; charset=utf-8");
        ex.sendResponseHeaders(estado, b.length == 0 ? -1 : b.length);
        if (b.length > 0) ex.getResponseBody().write(b);
        ex.close();
    }

    static String json(int id, String texto) {
        return "{\"id\":\"" + id + "\",\"texto\":\"" + texto.replace("\\", "\\\\").replace("\"", "\\\"") + "\"}";
    }

    // Atiende /tareas y /tareas/{id}
    static void manejar(HttpExchange ex) throws IOException {
        // TODO: GET /tareas, POST /tareas, GET /tareas/{id}, DELETE /tareas/{id} y los errores
        responder(ex, 501, "");
    }

    public static void main(String[] args) throws Exception {
        HttpServer srv = HttpServer.create(new InetSocketAddress("localhost", 0), 0);
        srv.createContext("/tareas", Main::manejar);
        srv.start();
        String base = "http://localhost:" + srv.getAddress().getPort();

        HttpClient cliente = HttpClient.newHttpClient();
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] t = sc.nextLine().trim().split(" ", 3); // MÉTODO ruta [cuerpo]
            HttpRequest.BodyPublisher cuerpo = t.length > 2
                ? HttpRequest.BodyPublishers.ofString(t[2]) : HttpRequest.BodyPublishers.noBody();
            HttpRequest req = HttpRequest.newBuilder(URI.create(base + t[1])).method(t[0], cuerpo).build();
            HttpResponse<String> r = cliente.send(req, HttpResponse.BodyHandlers.ofString());
            System.out.println(t[0] + " " + t[1] + " → " + r.statusCode() + (r.body().isEmpty() ? "" : " " + r.body()));
        }
        srv.stop(0);
    }
}
```

Ejemplo: Crear, listar y borrar

ENTRADA

```
POST /tareas Comprar pan
POST /tareas Estudiar PSP
GET /tareas
DELETE /tareas/1
GET /tareas
```

SALIDA ESPERADA

```
POST /tareas → 201 {"id":1,"texto":"Comprar pan"}
POST /tareas → 201 {"id":2,"texto":"Estudiar PSP"}
GET /tareas → 200 [{"id":1,"texto":"Comprar pan"}, {"id":2,"texto":"Estudiar PSP"}]
DELETE /tareas/1 → 204
GET /tareas → 200 [{"id":2,"texto":"Estudiar PSP"}]
```

Ejemplo: Errores

ENTRADA

```
GET /tareas/7
GET /tareas/abc
POST /tareas
PUT /tareas
DELETE /tareas/1
```

SALIDA ESPERADA

```
GET /tareas/7 → 404 {"error":"no existe la tarea 7"}
GET /tareas/abc → 400 {"error":"id no válido"}
POST /tareas → 400 {"error":"texto vacío"}
PUT /tareas → 405
DELETE /tareas/1 → 404 {"error":"no existe la tarea 1"}
```

136. Disyuntor (circuit breaker) para un servicio inestable

Muy difícil · Desarrollo de Servicios en Red · apuntesdam.com/subject/prog-servicios/topic/servicios-red-java

Cuando un servicio remoto se cae, seguir llamándolo empeora las cosas: cada petición espera un tiempo de espera completo y sobrecarga al servicio cuando intenta recuperarse. El patrón disyuntor lo evita con tres estados. CERRADO: las llamadas pasan; tras umbralFallos fallos seguidos, pasa a ABIERTO y el contador de fallos vuelve a 0. ABIERTO: se rechazan las llamadas sin hacerlas durante espera instantes desde que se abrió. Pasado ese tiempo, la siguiente llamada se permite como prueba y el estado pasa a SEMIABIERTO. SEMIABIERTO: si la prueba sale bien, vuelve a CERRADO (y se reinician los fallos); si falla, vuelve a ABIERTO desde ese instante (aunque sea un solo fallo). Un éxito en CERRADO también reinicia el contador de fallos. Implementa permitir y registrar. La entrada es el umbral, la espera y una cadena con el resultado que daría el servicio en cada instante (E o F); el main simula el tiempo.

Código de partida (java)

```
import java.util.*;

class CircuitBreaker {
    enum Estado { CERRADO, ABIERTO, SEMIABIERTO }

    private final int umbralFallos;          // fallos seguidos para abrir
    private final int espera;               // instantes que permanece abierto
    private Estado estado = Estado.CERRADO;
    private int fallosSeguidos = 0;
    private int abiertoDesde = 0;

    CircuitBreaker(int umbralFallos, int espera) {
        this.umbralFallos = umbralFallos;
        this.espera = espera;
    }

    Estado estado() { return estado; }

    // ¿Se puede llamar al servicio en el instante t? (puede pasar de ABIERTO a SEMIABIERTO)
    boolean permitir(int t) {
        // TODO
        return true;
    }

    // Se informa del resultado de una llamada que se permitió
    void registrar(boolean exito, int t) {
        // TODO
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        CircuitBreaker cb = new CircuitBreaker(sc.nextInt(), sc.nextInt());
        String resultados = sc.next();          // una letra por instante: E = el servicio responde, F
        = falla
        int llamadas = 0, rechazadas = 0;
        for (int t = 0; t < resultados.length(); t++) {
            String linea;
            if (cb.permidir(t)) {
                llamadas++;
                boolean ok = resultados.charAt(t) == 'E';
                cb.registrar(ok, t);
                linea = (ok ? "llamada OK " : "llamada FALLA");
            } else {
                rechazadas++;
                linea = "rechazada ";
            }
            System.out.println("t=" + t + " " + linea + " → " + cb.estado());
        }
        System.out.println("Llamadas al servicio: " + llamadas + ", rechazadas sin llamar: " + rechazadas);
    }
}
```

Ejemplo: Caída y recuperación

ENTRADA

```
3 4
EEFFFFEEEEEEEE
```

SALIDA ESPERADA

```
t=0 llamada OK → CERRADO
t=1 llamada OK → CERRADO
t=2 llamada FALLA → CERRADO
t=3 llamada FALLA → CERRADO
t=4 llamada FALLA → ABIERTO
t=5 rechazada → ABIERTO
t=6 rechazada → ABIERTO
t=7 rechazada → ABIERTO
t=8 llamada OK → CERRADO
t=9 llamada OK → CERRADO
t=10 llamada FALLA → CERRADO
t=11 llamada OK → CERRADO
t=12 llamada OK → CERRADO
t=13 llamada OK → CERRADO
Llamadas al servicio: 11, rechazadas sin llamar: 3
```

Ejemplo: La prueba falla

ENTRADA

```
3 2
FFFEFEFEFE
```

SALIDA ESPERADA

```
t=0 llamada FALLA → CERRADO
t=1 llamada FALLA → CERRADO
t=2 llamada FALLA → ABIERTO
t=3 rechazada → ABIERTO
t=4 llamada FALLA → ABIERTO
t=5 rechazada → ABIERTO
t=6 llamada FALLA → ABIERTO
t=7 rechazada → ABIERTO
t=8 llamada OK → CERRADO
t=9 llamada OK → CERRADO
Llamadas al servicio: 7, rechazadas sin llamar: 3
```

137. Guardar contraseñas con PBKDF2

Difícil · Programación Segura en Java · apuntesdam.com/subject/prog-servicios/topic/programacion-segura

Nunca se guarda una contraseña, ni cifrada: se guarda un hash lento con sal. Implementa guardar, que genera una sal aleatoria de 16 bytes con `SecureRandom`, deriva el hash con el método `derive` (`PBKDF2-HMAC-SHA256`, 32 bytes) usando `ITERACIONES` y devuelve el texto «pbkdf2\$iteraciones\$sal\$hash», con la sal y el hash en Base64. Implementa también `verificar`, que separa las cuatro partes (devuelve `false` si el formato no es correcto), vuelve a derivar el hash de la contraseña introducida con la misma sal y las mismas iteraciones guardadas y lo compara en tiempo constante con `MessageDigest.isEqual`. El `main` comprueba el formato, que dos guardados de la misma contraseña son distintos y la verificación; la orden `VECTOR` calcula un hash con una sal fija para comprobar que `derive` se usa bien.

Código de partida (java)

```
import java.security.*;
import java.security.spec.*;
import java.util.*;
import javax.crypto.*;
import javax.crypto.spec.*;

public class Main {
    static final int ITERACIONES = 120_000;

    // Deriva 32 bytes de la contraseña con PBKDF2-HMAC-SHA256
    static byte[] derivar(String clave, byte[] sal, int iteraciones) throws GeneralSecurityException {
        PBEKeySpec spec = new PBEKeySpec(clave.toCharArray(), sal, iteraciones, 256);
        return SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256").generateSecret(spec).getEncoded();
    }

    // Devuelve "pbkdf2$iteraciones$salBase64$hashBase64" con una sal aleatoria de 16 bytes
    static String guardar(String clave) throws GeneralSecurityException {
        // TODO
        return clave;
    }

    // Comprueba una contraseña contra lo guardado, en tiempo constante
    static boolean verificar(String clave, String guardado) throws GeneralSecurityException {
        // TODO
        return guardado.equals(clave);
    }

    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] t = sc.nextLine().split(" ", 3);
            switch (t[0]) {
                case "VECTOR" -> { // VECTOR salHex clave: resultado reproducible con 1000
                    iteraciones
                    byte[] sal = HexFormat.of().parseHex(t[1]);
                    System.out.println("Hash: " + HexFormat.of().formatHex(derivar(t[2], sal, 1000)));
                }
                case "PROBAR" -> { // PROBAR claveGuardada claveIntroducida
                    String g1 = guardar(t[1]), g2 = guardar(t[1]);
                    String[] p = g1.split("\\$");
                    System.out.println("Formato: " + p.length + " partes, algoritmo " + p[0] + ", " +
                        (p.length > 1 ? p[1] : "?") + " iteraciones, sal de " +
                        + Base64.getDecoder().decode(p.length > 2 ? p[2] : "").length + " bytes, hash de "
                        + Base64.getDecoder().decode(p.length > 3 ? p[3] : "").length + " bytes");
                    System.out.println("Dos guardados de la misma clave son distintos: " + !g1.equals(g2));
                    System.out.println("La clave guardada no aparece en el texto: " + !g1.contains(t[1]));
                    System.out.println("Verificar «" + t[2] + "»: " + verificar(t[2], g1));
                }
            }
        }
    }
}
```

Ejemplo: Vector fijo

ENTRADA

VECTOR 000102030405060708090a0b0c0d0e0f patata

SALIDA ESPERADA

Hash:
e5816b525c317a12ddd9f411a9304dc0e131d02a179b24a028
34f989d4613578

Ejemplo: Clave correcta

ENTRADA

PROBAR Clave.Segura-2025 Clave.Segura-2025

SALIDA ESPERADA

Formato: 4 partes, algoritmo pbkdf2, 120000 iteraciones, sal de 16 bytes, hash de 32 bytes
Dos guardados de la misma clave son distintos: true
La clave guardada no aparece en el texto: true
Verificar «Clave.Segura-2025»: true

138. Tokens firmados con HMAC (como un JWT)

Muy difícil · Programación Segura en Java · apuntesdam.com/subject/prog-servicios/topic/programacion-segura

Tras iniciar sesión, muchas APIs entregan un token que el cliente envía en cada petición. Para que no se pueda falsificar, va firmado. Implementa crear: el token son tres partes separadas por puntos, cada una en Base64URL sin relleno: la cabecera {"alg":"HS256","typ":"JWT"}, los datos {"sub":"usuario","rol":"rol","exp":instante} (exactamente con ese formato, sin espacios) y la firma HMAC-SHA256 de «cabecera.datos» con la clave del servidor (usa el método hmac). Implementa verificar: si no hay tres partes o no se pueden decodificar, «INVÁLIDO: formato»; si la firma no coincide (compárala en tiempo constante), «INVÁLIDO: firma»; si ahora $\geq$ exp, «INVÁLIDO: caducado»; si todo está bien, «VÁLIDO usuario rol». Comprueba la firma antes de mirar los datos. El método campo, ya hecho, lee un valor del JSON de los datos.

Código de partida (java)

```
import java.nio.charset.StandardCharsets;
import java.security.*;
import java.util.*;
import javax.crypto.*;
import javax.crypto.spec.*;

public class Main {
    static final Base64.Encoder B64 = Base64.getUrlEncoder().withoutPadding();
    static final Base64.Decoder D64 = Base64.getUrlDecoder();

    static byte[] hmac(byte[] clave, String datos) throws GeneralSecurityException {
        Mac mac = Mac.getInstance("HmacSHA256");
        mac.init(new SecretKeySpec(clave, "HmacSHA256"));
        return mac.doFinal(datos.getBytes(StandardCharsets.UTF_8));
    }

    // Token: base64url(cabecera) + "." + base64url(datos) + "." + base64url(HMAC de las dos primeras partes)
    // La cabecera es {"alg":"HS256","typ":"JWT"} y los datos {"sub":"usuario","rol":"rol","exp":instante}
    static String crear(byte[] clave, String usuario, String rol, long exp) throws GeneralSecurityException {
        // TODO
        return "";
    }

    // Devuelve "VÁLIDO usuario rol", o "INVÁLIDO: motivo" (formato, firma o caducado)
    static String verificar(byte[] clave, String token, long ahora) throws GeneralSecurityException {
        // TODO
        return "VÁLIDO";
    }

    // Extrae un campo de un JSON plano y sencillo como el de los datos
    static String campo(String json, String nombre) {
        int i = json.indexOf("\"" + nombre + "\":");
        if (i < 0) return null;
        int ini = i + nombre.length() + 3;
        if (json.charAt(ini) == '"') return json.substring(ini + 1, json.indexOf('"', ini + 1));
        int fin = ini;
        while (fin < json.length() && (Character.isDigit(json.charAt(fin)) || json.charAt(fin) == '-'))
            fin++;
        return json.substring(ini, fin);
    }

    public static void main(String[] args) throws Exception {
        byte[] clave = "clave-secreta-del-servidor-de-32b".getBytes(StandardCharsets.UTF_8);
        byte[] otraClave = "otra-clave-que-no-es-la-buena-32".getBytes(StandardCharsets.UTF_8);
        Scanner sc = new Scanner(System.in);
        String usuario = sc.next(), rol = sc.next();
        long exp = sc.nextLong();
        String token = crear(clave, usuario, rol, exp);
        System.out.println("Token: " + token);
        while (sc.hasNext()) {
            String prueba = sc.next();
            long ahora = sc.nextLong();
            String t = switch (prueba) {
                case "normal" -> token;
                case "otra-clave" -> crear(otraClave, usuario, rol, exp);
                case "cambiar-rol" -> { // el atacante edita los datos y deja la firma original
                    String[] p = token.split("\\.");
                    String datos = new String(D64.decode(p.length > 1 ? p[1] : ""),
                        StandardCharsets.UTF_8).replace(rol, "admin");
                    yield p[0] + "." + B64.encodeToString(datos.getBytes(StandardCharsets.UTF_8)) + "." +
                        (p.length > 2 ? p[2] : "");
                }
                default -> "esto.no.es-un-token.valido";
            };
            System.out.println(prueba + " en t=" + ahora + ": " + verificar(clave, t, ahora));
        }
    }
}
```

Ejemplo: Token válido y caducado

ENTRADA

```
ana alumna 1000
normal 500
normal 1000
```

SALIDA ESPERADA

```
Token:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhbG88JmEiLCJyb2wiOiJhbHVtbnEiLCJleHAiOiJwMDB9.IcQ4Kt
kBG8JvEw1Zgu3Yd5TgKgDUZUhGdUUXpKs
normal en t=500: VÁLIDO ana alumna
normal en t=1000: INVÁLIDO: caducado
```

Ejemplo: Ataques

ENTRADA

```
luis profesor 2000
cambiar-rol 10
otra-clave 10
basura 10
```

SALIDA ESPERADA

```
Token:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJsdWlziWicm9sIjoicHJvZmVzb3IiLCJleHAiOiJwMDB9.IcQ4Kt
kPdyIVXmF3_Qw5jwIsK4X8GRGP0noHWUgIYKY
cambiar-rol en t=10: INVÁLIDO: firma
otra-clave en t=10: INVÁLIDO: firma
basura en t=10: INVÁLIDO: formato
```

Ejercicios largos

139. Biblioteca: préstamos, devoluciones y multas

Difícil · 90 minutos · apuntesdam.com/ejercicios/java/biblioteca-prestamos-y-multas

La biblioteca de un instituto lleva los préstamos en una libreta y quiere un pequeño programa de consola. Cada línea que recibe el programa es una orden: dar de alta un libro o un socio, prestar un libro, devolverlo o pedir un informe. El programa contesta a cada orden con una línea (o con varias, en el informe).

Los días se cuentan como números enteros desde el comienzo del curso (día 0, día 1...), así que no hace falta trabajar con fechas. Un libro puede tener varios ejemplares: se puede prestar mientras quede alguno libre.

Hay tres normas: cada préstamo dura 14 días; un socio no puede tener más de 3 libros a la vez; y si devuelve un libro tarde, paga 0,50 € por cada día de retraso y no puede llevarse libros durante tantos días como se retrasó, contados desde el día en que lo devuelve.

Requisitos

- **LIBRO isbn ejemplares título** da de alta un libro (el título puede tener espacios) y responde **Libro añadido: título (n ejemplares)**, con «ejemplar» en singular si es 1. Si el ISBN ya existe: **Error: ya existe un libro con el ISBN isbn**.
- **SOCIO número nombre** da de alta un socio y responde **Socio añadido: nombre (número)**. Si el número ya existe: **Error: ya existe un socio con el número número**.
- **PRESTAR día socio isbn** comprueba, en este orden: que exista el socio (**Error: no existe el socio número**), que exista el libro (**Error: no existe el libro isbn**), que el socio no esté bloqueado (**Error: nombre no puede llevarse libros hasta el día X**), que no tenga ya 3 préstamos (**Error: nombre ya tiene 3 libros prestados**), que no tenga ya ese libro (**Error: nombre ya tiene «título»**) y que quede algún ejemplar (**Error: no quedan ejemplares de «título»**). Si todo va bien: **Préstamo: nombre se lleva «título» hasta el día X**, donde X es el día del préstamo más 14.
- **DEVOLVER día socio isbn**: si ese socio no tiene prestado ese libro, **Error: el socio número no tiene prestado el libro isbn**. Si lo devuelve el día del vencimiento o antes, **Devolución: «título» a tiempo**. Si no, **Devolución: «título» con N días de retraso. Multa: X,XX €** («día» en singular si es 1), se suma la multa al socio y queda bloqueado hasta el día de la devolución más los días de retraso (si ya estaba bloqueado más tiempo, se queda el bloqueo más largo).

- **INFORME** muestra `== Informe ==`, **Préstamos activos: N**, una línea por cada socio con préstamos, ordenados por número de socio (`- nombre (número): «título 1», «título 2»`, con los libros en el orden en que se los llevó) y por último **Multas: X,XX €** con la suma de todas las multas.
- Cualquier otra orden responde **Error: orden desconocida ORDEN**. Las líneas vacías se ignoran. Los importes se escriben con coma decimal y dos decimales.

Formato de la entrada

- Una orden por línea, con las palabras separadas por un espacio. Los números de día y de socio son enteros; el ISBN es una palabra sin espacios.
- El programa termina cuando se acaba la entrada.

Ejemplo: Préstamos, un ejemplar agotado y una devolución a tiempo

ENTRADA

```
LIBRO 978-84-376-0494-7 2 Cien años de soledad
LIBRO 978-84-204-8291-3 1 El Hobbit
SOCIO 1 Ana Ruiz
SOCIO 2 Luis Gil
PRESTAR 1 1 978-84-376-0494-7
PRESTAR 1 2 978-84-204-8291-3
PRESTAR 3 1 978-84-204-8291-3
DEVOLVER 10 2 978-84-204-8291-3
PRESTAR 11 1 978-84-204-8291-3
INFORME
```

SALIDA ESPERADA

```
Libro añadido: Cien años de soledad (2 ejemplares)
Libro añadido: El Hobbit (1 ejemplar)
Socio añadido: Ana Ruiz (1)
Socio añadido: Luis Gil (2)
Préstamo: Ana Ruiz se lleva «Cien años de soledad»
hasta el día 15
Préstamo: Luis Gil se lleva «El Hobbit» hasta el
día 15
Error: no quedan ejemplares de «El Hobbit»
Devolución: «El Hobbit» a tiempo
Préstamo: Ana Ruiz se lleva «El Hobbit» hasta el
día 25
== Informe ==
Préstamos activos: 2
- Ana Ruiz (1): «Cien años de soledad», «El
Hobbit»
Multas: 0,00 €
```

Ejemplo: Retrasos, multas y bloqueo

ENTRADA

```
LIBRO A1 1 Dune
LIBRO B2 1 Fundación
SOCIO 7 Marta Sanz
PRESTAR 0 7 A1
DEVOLVER 20 7 A1
PRESTAR 22 7 B2
PRESTAR 26 7 B2
DEVOLVER 27 7 A1
DEVOLVER 41 7 B2
INFORME
```

SALIDA ESPERADA

```
Libro añadido: Dune (1 ejemplar)
Libro añadido: Fundación (1 ejemplar)
Socio añadido: Marta Sanz (7)
Préstamo: Marta Sanz se lleva «Dune» hasta el día
14
Devolución: «Dune» con 6 días de retraso. Multa:
3,00 €
Error: Marta Sanz no puede llevarse libros hasta
el día 26
Préstamo: Marta Sanz se lleva «Fundación» hasta el
día 40
Error: el socio 7 no tiene prestado el libro A1
Devolución: «Fundación» con 1 día de retraso.
Multa: 0,50 €
== Informe ==
Préstamos activos: 0
Multas: 3,50 €
```

```

import java.util.*;

public class Main {
    static class Libro {
        final String isbn;
        final String titulo;
        int disponibles;

        Libro(String isbn, String titulo, int ejemplares) {
            this.isbn = isbn;
            this.titulo = titulo;
            this.disponibles = ejemplares;
        }
    }

    static class Prestamo {
        final Libro libro;
        final int vence;

        Prestamo(Libro libro, int vence) {
            this.libro = libro;
            this.vence = vence;
        }
    }

    static class Socio {
        final int numero;
        final String nombre;
        final List<Prestamo> prestamos = new ArrayList<>();
        int bloqueadoHasta = 0;
        double multas = 0;

        Socio(int numero, String nombre) {
            this.numero = numero;
            this.nombre = nombre;
        }
    }

    static final Locale ES = Locale.forLanguageTag("es-ES");
    static final Map<String, Libro> libros = new HashMap<>();
    static final Map<Integer, Socio> socios = new TreeMap<>();

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String linea = sc.nextLine().trim();
            if (!linea.isEmpty()) procesar(linea);
        }
    }

    static void procesar(String linea) {
        String[] p = linea.split(" ");
        switch (p[0]) {
            case "LIBRO" -> altaLibro(p[1], Integer.parseInt(p[2]), resto(p, 3));
            case "SOCIO" -> altaSocio(Integer.parseInt(p[1]), resto(p, 2));
            case "PRESTAR" -> prestar(Integer.parseInt(p[1]), Integer.parseInt(p[2]), p[3]);
            case "DEVOLVER" -> devolver(Integer.parseInt(p[1]), Integer.parseInt(p[2]), p[3]);
            case "INFORME" -> informe();
            default -> System.out.println("Error: orden desconocida " + p[0]);
        }
    }

    static String resto(String[] p, int desde) {
        return String.join(" ", Arrays.copyOfRange(p, desde, p.length));
    }

    static void altaLibro(String isbn, int ejemplares, String titulo) {
        // TODO
    }

    static void altaSocio(int numero, String nombre) {
        // TODO
    }
}

```

```

    }

    static void prestar(int dia, int numero, String isbn) {
        // TODO: comprueba las condiciones en el orden del enunciado
    }

    static void devolver(int dia, int numero, String isbn) {
        // TODO: retraso, multa y bloqueo
    }

    static void informe() {
        // TODO
    }

    static String euros(double importe) {
        return String.format(ES, "%.2f €", importe);
    }
}

```

140. Clasificación de una liga de fútbol

Difícil · 75 minutos · apuntesdam.com/ejercicios/java/clasificacion-de-una-liga

Una liga de fútbol de barrio apunta los resultados de cada jornada en un fichero de texto y quiere un programa que saque la clasificación sin tener que echar cuentas a mano. El fichero lo rellenan varias personas, así que a veces hay líneas vacías o mal escritas: el programa debe avisar de ellas y seguir con las demás.

Cada línea válida es un partido: `equipo local;goles-goles;equipo visitante`, por ejemplo `Atlético Norte;2-1;Real Sur`. Una victoria da 3 puntos, un empate 1 y una derrota 0. Los nombres de los equipos tienen como mucho 18 caracteres y pueden llevar espacios.

La clasificación se ordena por puntos. Si dos equipos empatan a puntos, va delante el de mayor diferencia de goles (goles a favor menos goles en contra); si siguen empatados, el que más goles ha marcado; y si aún así empatan, por orden alfabético del nombre.

Requisitos

- Lee la entrada línea a línea. Las líneas vacías (o solo con espacios) se ignoran sin avisar.
- Una línea es válida si al separarla por `;` salen exactamente tres partes, los dos nombres no están vacíos, son distintos (un equipo no juega contra sí mismo) y el marcador son dos números enteros separados por un guion. Se ignoran los espacios al principio y al final de cada parte.
- Por cada línea no válida escribe en el momento `Aviso: la línea N no es un resultado válido y se ignora`, donde N es su número de línea en la entrada (contando también las vacías, empezando en 1).
- Para cada equipo cuenta: partidos jugados (PJ), ganados (G), empatados (E), perdidos (P), goles a favor (GF), goles en contra (GC), diferencia de goles (DG) y puntos (Pts).
- Escribe la clasificación con una cabecera y una fila por equipo, con este formato exacto de columnas: `String.format("%-4d%-20s%3d%3d%3d%3d%4d%4d%5s%5d", ...)` para las filas (la DG va como texto, con signo + si es positiva) y la cabecera que ya trae el código inicial.
- Después, una línea en blanco y dos líneas más: `Partidos: N · Goles: M · Media: X,XX goles por partido` (media con dos decimales y coma) y `Equipo menos goleado: nombre (N goles en contra)`, con «gol» en singular si es 1. Si varios equipos empatan con menos goles en contra, se escribe el que va más arriba en la clasificación.
- Si no hay ningún resultado válido, después de los avisos escribe solo `No hay resultados válidos.`

Formato de la entrada

- Una línea por partido: `local;goles-goles;visitante`. Puede haber líneas vacías y líneas mal escritas.

Ejemplo: Una liga de cuatro equipos

ENTRADA

Atlético Norte;2-1;Real Sur
Deportivo Este;0-0;Unión Oeste
Real Sur;3-3;Deportivo Este
Unión Oeste;1-2;Atlético Norte
Atlético Norte;2-0;Deportivo Este
Real Sur;1-0;Unión Oeste

SALIDA ESPERADA

Pos	Equipo	PJ	G	E	P	GF	GC	DG
Pts								
1	Atlético Norte	3	3	0	0	6	2	+4
9								
2	Real Sur	3	1	1	1	5	5	0
4								
3	Deportivo Este	3	0	2	1	3	5	-2
2								
4	Unión Oeste	3	0	1	2	1	3	-2
1								

Partidos: 6 · Goles: 15 · Media: 2,50 goles por partido
Equipo menos goleado: Atlético Norte (2 goles en contra)

Ejemplo: Con avisos y empates a puntos

ENTRADA

Ciudad Alta;1-0;Puerto Bajo
Puerto Bajo;3-1;Valle Verde
Valle Verde;2-0;Ciudad Alta

esto no es un resultado
Montaña Azul;2;Valle Verde
Puerto Bajo;0-0;Puerto Bajo
Montaña Azul;1-1;Ciudad Alta

SALIDA ESPERADA

Aviso: la línea 5 no es un resultado válido y se ignora
Aviso: la línea 6 no es un resultado válido y se ignora
Aviso: la línea 7 no es un resultado válido y se ignora

Pos	Equipo	PJ	G	E	P	GF	GC	DG
Pts								
1	Ciudad Alta	3	1	1	1	2	3	-1
4								
2	Puerto Bajo	2	1	0	1	3	2	+1
3								
3	Valle Verde	2	1	0	1	3	3	0
3								
4	Montaña Azul	1	0	1	0	1	1	0
1								

Partidos: 4 · Goles: 9 · Media: 2,25 goles por partido
Equipo menos goleado: Montaña Azul (1 gol en contra)

Código de partida (java)

```
import java.util.*;

public class Main {
    static class Equipo {
        final String nombre;
        int pj, g, e, p, gf, gc;

        Equipo(String nombre) {
            this.nombre = nombre;
        }

        int dg() {
            return 0; // TODO: diferencia de goles
        }

        int puntos() {
            return 0; // TODO: 3 por victoria y 1 por empate
        }
    }

    static final Locale ES = Locale.forLanguageTag("es-ES");
    static final Map<String, Equipo> equipos = new LinkedHashMap<>();

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int numLinea = 0;
        while (sc.hasNextLine()) {
            String linea = sc.nextLine();
            numLinea++;
            // TODO: ignora las líneas vacías, valida el resto y registra el resultado en los dos equipos
        }
        // TODO: ordena los equipos, imprime la tabla y las dos líneas finales
    }

    static boolean esValida(String[] partes) {
        return false; // TODO
    }

    static Equipo equipo(String nombre) {
        return equipos.computeIfAbsent(nombre, Equipo::new);
    }

    static void registrar(Equipo t, int aFavor, int enContra) {
        // TODO
    }

    static void imprimirTabla(List<Equipo> tabla) {
        System.out.println(String.format("%-4s%-20s%3s%3s%3s%4s%4s%5s%5s", "Pos", "Equipo", "PJ", "G",
            "E", "P", "GF", "GC", "DG", "Pts"));
        // TODO: una fila por equipo con el formato del enunciado
    }
}
```

141. Cajero automático con PIN, límite diario y billetes

Medio · 50 minutos · apuntesdam.com/ejercicios/java/cajero-automatico

Un banco quiere probar la lógica de sus cajeros antes de instalarla en las máquinas. En lugar de pulsar botones, el programa recibe por teclado lo que hace el cliente, una orden por línea, y responde con lo que aparecería en la pantalla del cajero.

La primera línea no la escribe el cliente: es lo que el cajero lee de la tarjeta, el saldo de la cuenta y el PIN correcto.

Mientras el cliente no acierte el PIN no puede hacer nada más, y si falla tres veces el cajero se queda la tarjeta.

El cajero solo tiene billetes de 50, 20 y 10 €, así que solo se puede sacar dinero en múltiplos de 10, y nunca más de 600 € al día entre todas las retiradas. Como cualquier programa que maneja dinero, trabaja en céntimos enteros: sumar 49,95 en coma flotante acaba dando 49,94999... y el saldo dejaría de cuadrar.

Requisitos

- La primera línea es `saldo pin`, por ejemplo `850.50 1234`. Después, cada línea es una orden (en mayúsculas o minúsculas) con su argumento separado por un espacio. Las líneas vacías se ignoran.
- Hasta acertar el PIN, cualquier orden que no sea `PIN` responde `Introduzca el PIN`. `PIN n` responde `PIN correcto` si acierta; si falla, `PIN incorrecto`. Quedan 2 intentos o `PIN incorrecto`. Queda 1 intento, y al tercer fallo `PIN incorrecto`. Tarjeta retenida y el programa termina sin leer nada más.
- Con el PIN correcto: `SALDO` escribe `Saldo: X`; `PIN` escribe `Ya ha introducido el PIN`; cualquier orden desconocida, `Operación desconocida`.
- `INGRESAR cantidad`: la cantidad es un número positivo con punto decimal y como mucho dos decimales. Escribe `Ingresados X. Saldo: Y`, o `Cantidad no válida`.
- `RETIRAR cantidad`, comprobando en este orden: que sea un múltiplo de 10 € mayor que 0 (si no, `Importe no válido: debe ser múltiplo de 10 €`), que haya saldo (`Saldo insuficiente`) y que con lo ya retirado hoy no se pasen los 600 € (`Supera el límite diario: hoy puede retirar hasta X`). Si todo va bien: `Retirados X: desglose. Saldo: Y`, con el desglose en el menor número de billetes, de mayor a menor y sin los que no se usan (por ejemplo `3x50 1x20 1x10`).
- `SALIR` escribe `Operaciones: N · Retirado hoy: X` (N cuenta los ingresos y retiradas hechos) y `Retire su tarjeta`, y termina: lo que venga después no se procesa. Si la entrada se acaba sin `SALIR` y el cliente se había identificado, el cajero escribe esas dos mismas líneas.
- Los importes se escriben con punto de millar, coma decimal y dos decimales, seguidos de un espacio y `€`:
`1.012.345,60 €`.

Formato de la entrada

- Línea 1: `saldo pin`. Resto: una orden por línea (`PIN`, `SALDO`, `INGRESAR`, `RETIRAR`, `SALIR`).

Ejemplo: Una sesión completa

ENTRADA

```
850.50 1234
SALDO
PIN 1111
PIN 1234
SALDO
RETIRAR 180
INGRESAR 49.95
RETIRAR 75
RETIRAR 500
RETIRAR 420
CONSULTAR
SALIR
SALDO
```

SALIDA ESPERADA

```
Introduzca el PIN
PIN incorrecto. Quedan 2 intentos
PIN correcto
Saldo: 850,50 €
Retirados 180,00 €: 3x50 1x20 1x10. Saldo: 670,50 €
Ingresados 49,95 €. Saldo: 720,45 €
Importe no válido: debe ser múltiplo de 10 €
Supera el límite diario: hoy puede retirar hasta 420,00 €
Retirados 420,00 €: 8x50 1x20. Saldo: 300,45 €
Operación desconocida
Operaciones: 3 · Retirado hoy: 600,00 €
Retire su tarjeta
```

Ejemplo: Tres fallos de PIN

ENTRADA

```
2000 9876
PIN 1
PIN 2
PIN 3
SALDO
```

SALIDA ESPERADA

```
PIN incorrecto. Quedan 2 intentos
PIN incorrecto. Queda 1 intento
PIN incorrecto. Tarjeta retenida
```


Código de partida (java)

```
import java.util.Locale;
import java.util.Scanner;

public class Main {
    static final long LIMITE_DIARIO = 60000; // 600 € en céntimos
    static final int[] BILLETES = {50, 20, 10};

    /** Céntimos → «1.234,50 €». */
    static String euros(long centimos) {
        String entero = String.format(Locale.ROOT, "%d", centimos / 100).replace(',', '.');
        return entero + "," + String.format("%02d", centimos % 100) + " €";
    }

    /** «49.95» → 4995 céntimos; -1 si no es una cantidad válida (hasta 2 decimales). */
    static long centimos(String texto) {
        // TODO
        return -1;
    }

    /** Desglose en billetes de 50, 20 y 10 con el menor número de billetes: «3x50 1x20 1x10». */
    static String desglose(long euros) {
        // TODO
        return "";
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] inicio = sc.nextLine().trim().split("\\s+");
        long saldo = centimos(inicio[0]);
        String pin = inicio[1];

        while (sc.hasNextLine()) {
            String linea = sc.nextLine().trim();
            // TODO: PIN (3 intentos), SALDO, INGRESAR, RETIRAR y SALIR
        }
    }
}
```

142. Buscaminas: números, casilla más peligrosa y zonas despejadas

Difícil · 70 minutos · apuntesdam.com/ejercicios/java/buscaminas-numeros-y-zonas

En el Buscaminas, cada casilla sin mina muestra cuántas minas hay en las ocho casillas que la rodean (también en diagonal). Con esos números el jugador deduce dónde están las minas.

Cuando se pulsa una casilla con un 0, el juego no la abre sola: abre de golpe todos los ceros conectados con ella (también en diagonal), como una mancha. Cada una de esas manchas es una «zona despejada», y contar cuántas hay da una idea de lo fácil que es un tablero.

Vas a escribir la parte del juego que prepara un tablero nuevo: recibe dónde están las minas y calcula todo lo demás. Los tableros vienen de un fichero de niveles que a veces está mal escrito, así que hay que validarlo.

Requisitos

- La primera línea tiene el número de filas y de columnas separados por un espacio (de 1 a 30 cada uno). Después vienen tantas líneas como filas, con exactamente tantos caracteres como columnas: * es una mina y . una casilla libre.
- Si la cabecera no es válida, escribe **Tablero no válido: línea 1**. Si una fila no tiene la longitud correcta o tiene otro carácter, **Tablero no válido: línea N** (N es su número de línea en la entrada, contando la cabecera). Si faltan filas, **Tablero no válido: faltan filas**. En todos los casos no se escribe nada más.
- Escribe **Tablero:** y el tablero con las minas como * y cada casilla libre sustituida por el número de minas vecinas (de 0 a 8).
- Minas: M de T casillas (P %)**, con el porcentaje con un decimal y coma decimal.

- **Casilla más peligrosa:** fila F, columna C (K minas alrededor) : la casilla libre con más minas vecinas (filas y columnas se numeran desde 1; si hay empate, la primera leyendo por filas; «mina» en singular si K es 1). Si ninguna casilla libre tiene minas alrededor (o no hay casillas libres), **Casilla más peligrosa: ninguna** .
- **Zonas despejadas:** Z : cuántos grupos de casillas con 0 hay, considerando que dos ceros están conectados si son vecinos, también en diagonal.

Formato de la entrada

- Línea 1: **filas columnas** . Después, una línea por fila con ***** (mina) y **.** (libre).

Ejemplo: Un tablero de 5 × 6

ENTRADA

```
5 6
*.....
..*....
.....
...*,
.....
```

SALIDA ESPERADA

```
Tablero:
*21100
12*100
012321
001**1
001221
Minas: 4 de 30 casillas (13,3 %)
Casilla más peligrosa: fila 3, columna 4 (3 minas
alrededor)
Zonas despejadas: 2
```

Ejemplo: Un tablero mal escrito

ENTRADA

```
3 4
..*.
.X..
....
```

SALIDA ESPERADA

```
Tablero no válido: línea 3
```

Código de partida (java)

```
import java.util.Locale;
import java.util.Scanner;

public class Main {
    static int filas, columnas;
    static char[][] tablero;
    static int[][] cuenta;

    /** Minas en las 8 casillas vecinas de (f, c). */
    static int minasAlrededor(int f, int c) {
        // TODO
        return 0;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // TODO: lee y valida la cabecera y las filas del tablero
        // TODO: escribe el tablero con los números, las minas, la casilla más peligrosa y las zonas
        despejadas
    }
}
```

143. Cifrado de Vigenère con el alfabeto español

Difícil · 60 minutos · apuntesdam.com/ejercicios/java/cifrado-vigenere-con-enie

El cifrado de Vigenère se usó durante siglos para mensajes militares y diplomáticos. Funciona como el de César (desplazar cada letra unas posiciones en el alfabeto), pero el desplazamiento cambia en cada letra según una palabra clave: con la clave LIMON, la primera letra se desplaza 11 posiciones (la L), la segunda 8 (la I), y así sucesivamente, volviendo a empezar la clave cuando se acaba.

La mayoría de las implementaciones usan el alfabeto inglés de 26 letras. Aquí se usa el español, con la Ñ entre la N y la O: 27 letras en total, de la A (posición 0) a la Z (posición 26). Al desplazar más allá de la Z se vuelve a empezar por la A.

Los espacios, los números, los signos y las vocales con tilde no se cifran: se copian tal cual y no gastan letra de la clave. Las minúsculas se cifran como su mayúscula y siguen siendo minúsculas.

Requisitos

- Cada línea es una orden seguida de su argumento (todo lo que hay tras el primer espacio). La orden puede ir en mayúsculas o minúsculas; las líneas vacías se ignoran. Una orden desconocida escribe `Orden desconocida: ORDEN` (en mayúsculas).
- `CLAVE palabra` guarda la clave si todas sus letras son del alfabeto (en mayúsculas o minúsculas, la Ñ incluida) y escribe `Clave guardada (N letras)` («letra» si es una). Si no, `Clave no válida: solo letras de la A a la Z y la Ñ` y se mantiene la clave anterior.
- `CIFRAR texto` escribe `Cifrado: ...`: cada letra del alfabeto se desplaza hacia delante tantas posiciones como la posición de la letra de la clave que le toca. La clave vuelve a empezar en cada orden y solo avanza con las letras que se cifran.
- `DESCIFRAR texto` escribe `Descifrado: ...` deshaciendo el desplazamiento.
- Si no hay clave guardada, `CIFRAR` y `DESCIFRAR` escriben `Falta la clave`; si hay clave pero no texto, `Falta el texto`.
- `ANALIZAR texto` escribe `Letras: N · Más frecuentes: A 5, B 3, C 2`: el total de letras del alfabeto del texto (sin distinguir mayúsculas) y las tres más repetidas con su número de apariciones, de más a menos (si hay empate, en el orden del alfabeto; si hay menos de tres letras distintas, las que haya). Si no hay ninguna letra, solo `Letras: 0`.

Formato de la entrada

- Una orden por línea: `CLAVE`, `CIFRAR`, `DESCIFRAR` o `ANALIZAR`, seguida de su argumento.

Posición de cada letra

A	B	C	D	E	F	G	H	I	J	K	L	M	N	Ñ	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Ejemplo: Cifrar, descifrar y analizar

ENTRADA

CIFRAR hola
 CLAVE Limón
 CLAVE Limon
 CIFRAR Ataque al amanecer, ¡ya!
 DESCIFRAR Lbmfo iw oylupqqc, ¡gm!
 ANALIZAR El pingüino Ñoño come ñoquis

SALIDA ESPERADA

Falta la clave
 Clave no válida: solo letras de la A a la Z y la Ñ
 Clave guardada (5 letras)
 Cifrado: Lbmfo iw oylupqqc, ¡gm!
 Descifrado: Ataque al amanecer, ¡ya!
 Letras: 23 · Más frecuentes: O 5, I 3, Ñ 3

Ejemplo: Órdenes en minúsculas

ENTRADA

clave sol
 cifrar
 descifrar Zzz zzz
 analizar ab ba
 analizar 1234 ¿?
 borrar todo

SALIDA ESPERADA

Clave guardada (3 letras)
 Falta el texto
 Descifrado: Hlo hlo
 Letras: 4 · Más frecuentes: A 2, B 2
 Letras: 0
 Orden desconocida: BORRAR

Código de partida (java)

```
import java.util.Scanner;

public class Main {
    static final String ALFABETO = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    static final int N = ALFABETO.length(); // 27 letras

    /** La clave en mayúsculas, o null si tiene algo que no sea una letra del alfabeto. */
    static String validarClave(String texto) {
        // TODO
        return null;
    }

    /** Cifra (sentido = 1) o descifra (sentido = -1). Solo cambian las letras del alfabeto, que conservan su caja. */
    static String vigenere(String texto, String clave, int sentido) {
        // TODO
        return texto;
    }

    /** «Letras: 23 · Más frecuentes: O 5, Ñ 3, E 2». */
    static String analizar(String texto) {
        // TODO
        return "Letras: 0";
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String clave = null;
        while (sc.hasNextLine()) {
            String linea = sc.nextLine().trim();
            // TODO: separa la orden del argumento y atiende CLAVE, CIFRAR, DESCIFRAR y ANALIZAR
        }
    }
}
```

144. Parking con tarifas por tipo de vehículo

Difícil · 75 minutos · apuntesdam.com/ejercicios/java/parking-con-tarifas

El aparcamiento de un centro comercial tiene 4 plazas para coches, 2 para motos y 1 para furgonetas (es una versión de prueba: el real tiene cientos). La barrera de entrada y la de salida envían al programa una línea por cada vehículo que pasa.

Cada tipo de vehículo paga distinto, pero todos con la misma regla: los 10 primeros minutos son gratis (para quien entra y se va) y, a partir de ahí, se cobra cada cuarto de hora empezado, con un máximo por día. Esa regla común es justo lo que debe ir en una clase padre, y lo que cambia (el precio y el máximo), en cada clase hija.

Las cámaras leen las matrículas en el formato actual español: cuatro cifras y tres consonantes, sin vocales, ni Ñ ni Q, para evitar palabras y confusiones.

Requisitos

- Cada línea es una orden: `ENTRA HH:MM tipo matrícula`, `SALE HH:MM matrícula` u `OCUPACION`. La orden, el tipo y la matrícula pueden venir en mayúsculas o minúsculas (la matrícula se guarda y se muestra en mayúsculas). Una línea con un número de palabras distinto del de su orden escribe `Orden incompleta`; una orden desconocida, `Orden desconocida`. Las líneas vacías se ignoran.
- `ENTRA` comprueba, en este orden: la hora (`Hora no válida: X`, de 00:00 a 23:59), el tipo (`Tipo no válido: X`; son `coche`, `moto` y `furgoneta`), la matrícula (`Matrícula no válida: X`: 4 cifras y 3 letras de `BCDFGHJKLMNPQRSTVWXYZ`), que no esté ya dentro (`X ya está dentro`) y que queden plazas de su tipo (`No quedan plazas de tipo`). Si todo va bien: `Entra X (tipo) a las HH:MM`.
- `SALE` comprueba la hora, que el vehículo esté dentro (`X no está en el parking`) y que la salida no sea anterior a la entrada (`La hora de salida es anterior a la entrada`). Si todo va bien, el vehículo deja libre su plaza y se escribe `Sale X (tipo) tras D: I`, con la duración como `2 h 35 min`, `3 h` o `8 min` y el importe en euros (`6,60 €`).
- Tarifas: hasta 10 minutos, 0 €. Después, por cada 15 minutos empezados: coche 0,60 €, moto 0,30 € y furgoneta 0,90 €, con un máximo de 18 €, 9 € y 25 € respectivamente. Todas las estancias son del mismo día.
- `OCUPACION` escribe `Ocupación: coche 2/4 · moto 0/2 · furgoneta 1/1`.
- Al terminar la entrada: `Recaudación: X (N salidas)` («salida» si es 1) y `Siguen dentro: ...` con las matrículas que no han salido en orden alfabético, separadas por coma y espacio, o `ninguno`.

Formato de la entrada

- Una orden por línea: `ENTRA HH:MM tipo matrícula`, `SALE HH:MM matrícula` u `OCUPACION`.

Plazas y tarifas

Tipo	Plazas	Cada 15 min empezados	Máximo diario
coche	4	0,60 €	18,00 €
moto	2	0,30 €	9,00 €
furgoneta	1	0,90 €	25,00 €

Ejemplo: Un día de parking

ENTRADA

ENTRA 08:15 coche 1234BCD
 ENTRA 08:20 moto 5678FGH
 ENTRA 08:25 furgoneta 9012JKL
 ENTRA 08:30 furgoneta 3456MNP
 OCUPACION
 SALE 08:28 5678FGH
 SALE 10:50 1234BCD
 ENTRA 11:00 coche 1234bcd
 SALE 19:45 9012JKL
 SALE 12:00 0000BBB

SALIDA ESPERADA

Entra 1234BCD (coche) a las 08:15
 Entra 5678FGH (moto) a las 08:20
 Entra 9012JKL (furgoneta) a las 08:25
 No quedan plazas de furgoneta
 Ocupación: coche 1/4 · moto 1/2 · furgoneta 1/1
 Sale 5678FGH (moto) tras 8 min: 0,00 €
 Sale 1234BCD (coche) tras 2 h 35 min: 6,60 €
 Entra 1234BCD (coche) a las 11:00
 Sale 9012JKL (furgoneta) tras 11 h 20 min: 25,00 €
 0000BBB no está en el parking
 Recaudación: 31,60 € (3 salidas)
 Siguen dentro: 1234BCD

Ejemplo: Datos no válidos

ENTRADA

ENTRA 25:00 coche 1234BCD
 ENTRA 09:00 bici 1234BCD
 ENTRA 09:00 coche 1234ABC
 ENTRA 09:00 coche 1234BC
 ENTRA 09:05 moto 1111BBB
 ENTRA 09:06 moto 1111BBB
 SALE 09:00 1111BBB
 ENTRA
 ESTADO

SALIDA ESPERADA

Hora no válida: 25:00
 Tipo no válido: bici
 Matrícula no válida: 1234ABC
 Matrícula no válida: 1234BC
 Entra 1111BBB (moto) a las 09:05
 1111BBB ya está dentro
 La hora de salida es anterior a la entrada
 Orden incompleta
 Orden desconocida
 Recaudación: 0,00 € (0 salidas)
 Siguen dentro: 1111BBB

Código de partida (java)

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.Scanner;

abstract class Vehiculo {
    private final String matricula;
    private final int entrada;    // minutos desde las 00:00

    Vehiculo(String matricula, int entrada) {
        this.matricula = matricula;
        this.entrada = entrada;
    }

    String getMatricula() { return matricula; }
    int getEntrada() { return entrada; }

    abstract String tipo();
    /** Céntimos por cada 15 minutos empezados. */
    abstract long precioFraccion();
    /** Lo máximo que se cobra en un día, en céntimos. */
    abstract long maximoDiario();

    /** Importe en céntimos de una estancia: 10 minutos gratis y, después, cada cuarto de hora empezado, con un máximo. */
    long importe(int minutos) {
        // TODO
        return 0;
    }
}

// TODO: las clases Coche, Moto y Furgoneta

public class Main {
    static final List<String> TIPOS = List.of("coche", "moto", "furgoneta");
    static final Map<String, Integer> PLAZAS = Map.of("coche", 4, "moto", 2, "furgoneta", 1);

    static final Map<String, Vehiculo> dentro = new HashMap<>();

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String linea = sc.nextLine().trim();
            // TODO: ENTRA, SALE y OCUPACION
        }
        // TODO: recaudación y vehículos que siguen dentro
    }
}
```

145. Evaluador de expresiones con pilas y variables

Muy difícil · 100 minutos · apuntesdam.com/ejercicios/java/evaluador-de-expresiones-con-variables

Las calculadoras, las hojas de cálculo y los compiladores tienen que entender expresiones como `3 + 4 * (2 - 1)`, donde la multiplicación va antes que la suma y los paréntesis cambian el orden. Hacerlo leyendo de izquierda a derecha no funciona: hay que reorganizar la expresión.

La solución clásica, del informático Edsger Dijkstra, es el algoritmo de la estación de maniobras (shunting-yard): pasa la expresión a notación postfija, en la que los operadores van detrás de sus operandos (`3 4 2 1 - * +`) y ya no hacen falta paréntesis ni prioridades. Evaluar la postfija es muy sencillo con una pila: cada número se apila y cada operador desapila dos, opera y apila el resultado.

Tu calculadora trabajará con números enteros (`long`), recordará variables y explicará cada error en lugar de fallar: es la parte de un intérprete que se ocupa de las expresiones.

Requisitos

- Cada línea no vacía es una expresión, una asignación `nombre = expresión` o la orden `VARIABLES`. Los nombres de variable empiezan por letra o `_` y siguen con letras, cifras o `_`; distinguen mayúsculas.
- Las expresiones usan enteros sin signo, variables, paréntesis, `+`, `-`, `*`, `/` (división entera, como en Java) y `%` (resto, como en Java), con los espacios que se quiera. `*`, `/` y `%` tienen más prioridad que `+` y `-`, y operadores de la misma prioridad se aplican de izquierda a derecha. Un `-` al principio, tras `(` o tras otro operador es un menos unario, que tiene la mayor prioridad.
- Para una expresión correcta escribe `línea → postfija → resultado`: la línea tal como se ha leído (sin espacios al principio ni al final), la postfija con sus elementos separados por un espacio (el menos unario se escribe `~`) y el valor. En una asignación el resultado es `nombre = valor`, y la variable queda guardada.
- Si hay un error al separar la expresión o al pasarla a postfija, escribe `línea → Error: mensaje`; si el error aparece al evaluarla, `línea → postfija → Error: mensaje` (sin la postfija si está vacía). Los mensajes son: `carácter no válido` `'c'`, `paréntesis sin abrir`, `paréntesis sin cerrar`, `expresión incompleta` (falta un operando o un operador), `división entre cero`, `variable desconocida nombre`, `número demasiado grande` (no cabe en un long), `número no válido texto` (empieza por cifra pero tiene letras) y `desbordamiento` (el resultado de una operación no cabe en un long). Una asignación con error no cambia la variable.
- `VARIABLES` escribe `Variables: a = 1, b = 2` con todas las variables por orden alfabético, o `Variables: ninguna`.

Formato de la entrada

- Una expresión, asignación u orden `VARIABLES` por línea.

Ejemplo: Prioridades, menos unario y variables

ENTRADA

```
3 + 4 * (2 - 1)
(3 + 4) * 2 - 10 / 3
x = 2 * -5
y = x * x % 7
-(x + 3) * y
VARIABLES
```

SALIDA ESPERADA

```
3 + 4 * (2 - 1) → 3 4 2 1 - * + → 7
(3 + 4) * 2 - 10 / 3 → 3 4 + 2 * 10 3 / - → 11
x = 2 * -5 → 2 5 ~ * → x = -10
y = x * x % 7 → x x * 7 % → y = 2
-(x + 3) * y → x 3 + ~ y * → 14
Variables: x = -10, y = 2
```

Ejemplo: Errores

ENTRADA

```
8 / (4 - 2 * 2)
(1 + 2
1 + 2)
3 +
5 $ 2
z + 1
4 5
```

SALIDA ESPERADA

```
8 / (4 - 2 * 2) → 8 4 2 2 * - / → Error: división
entre cero
(1 + 2 → Error: paréntesis sin cerrar
1 + 2) → Error: paréntesis sin abrir
3 + → 3 + → Error: expresión incompleta
5 $ 2 → Error: carácter no válido '$'
z + 1 → z 1 + → Error: variable desconocida z
4 5 → 4 5 → Error: expresión incompleta
```


Código de partida (java)

```
import java.util.ArrayDeque;
import java.util.ArrayList;
import java.util.Deque;
import java.util.List;
import java.util.Map;
import java.util.Scanner;
import java.util.TreeMap;

public class Main {
    /** Error de una expresión, con el mensaje que se muestra. */
    static class ErrorExpresion extends Exception {
        ErrorExpresion(String mensaje) { super(mensaje); }
    }

    static final Map<String, Long> variables = new TreeMap<>();

    /** «x * (2 - 15)» → [x, *, (, 2, -, 15, )]. */
    static List<String> separar(String texto) throws ErrorExpresion {
        // TODO
        return new ArrayList<>();
    }

    /** Notación infija → postfija (algoritmo de la estación de maniobras). */
    static List<String> aPostfija(List<String> tokens) throws ErrorExpresion {
        // TODO
        return new ArrayList<>();
    }

    static long evaluar(List<String> postfija) throws ErrorExpresion {
        // TODO
        return 0;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String linea = sc.nextLine().trim();
            // TODO: VARIABLES, asignaciones y expresiones
        }
    }
}
```

146. Conversor de CSV a JSON con comillas y tipos

Difícil · 75 minutos · apuntesdam.com/ejercicios/java/conversor-csv-a-json

Una academia exporta sus alumnos de una hoja de cálculo en CSV y quiere cargarlos en una aplicación web que solo entiende JSON. Parece trivial (separar por comas), pero los datos reales lo complican: hay apellidos como «López, Juan» que llevan una coma dentro y por eso van entre comillas, y textos que contienen las propias comillas, que en CSV se escriben dobles ("").

Además, en JSON los tipos importan: `8.5` es un número, `true` un booleano, un campo vacío es `null` y el resto son textos entre comillas. Hay una trampa clásica: un código postal como `08001` parece un número, pero si se convierte se pierde el cero. En JSON un número no puede empezar por cero, así que esos valores se quedan como texto.

Algunas líneas vienen mal (les faltan campos o tienen comillas sin cerrar). No se convierten, pero el programa dice cuáles eran para poder corregirlas.

Requisitos

- La primera línea no vacía es la cabecera, con los nombres de los campos. Cada línea siguiente es un registro. Las líneas en blanco se ignoran (pero cuentan para numerar las líneas).
- Los campos se separan por comas. Un campo puede ir entre comillas dobles, y entonces puede contener comas; dentro de las comillas, `""` representa una comilla. A cada campo (y a cada nombre de la cabecera) se le quitan los espacios del principio y del final.

- Un registro con un número de campos distinto del de la cabecera, o con comillas sin cerrar, no se convierte: se anota su número de línea.
- Tipos: un campo vacío es `null`; un campo que cumple el formato de número JSON (signo menos opcional, `0` o una cifra del 1 al 9 seguida de más cifras, y opcionalmente un punto y decimales) se escribe tal cual; `true` y `false` son booleanos; todo lo demás es texto entre comillas, con `"` escrito `\"`, `\` escrito `\\` y el tabulador escrito `\t`.
- Salida: un array JSON con un objeto por registro, en el orden del fichero, con dos espacios de sangría para cada objeto y cuatro para cada propiedad (`"campo": valor`), comas entre elementos y nada al final de la última línea. Sin registros, `[]`.
- Después del JSON, `Registros: N · Líneas ignoradas: M (11, 12...)`, sin el paréntesis si M es 0. Si la entrada no tiene ni la cabecera, solo `El fichero está vacío`.

Formato de la entrada

- Un fichero CSV: cabecera y una línea por registro, campos separados por comas y entre comillas cuando hace falta.

Ejemplo: Alumnos de una academia

ENTRADA

```
id,nombre,ciudad,cp,nota,beca
1,Ana Gil,Madrid,28001,8.5,true
2,"López, Juan",Barcelona,08001,6,false
3,"Eva \"la rápida\" Ruiz",Sevilla,41001,,true
4,Luis,Valencia,46001,7.25
5,Marta Díaz,Bilbao,48001,-0.5,false
```

SALIDA ESPERADA

```
[
  {
    "id": 1,
    "nombre": "Ana Gil",
    "ciudad": "Madrid",
    "cp": "28001",
    "nota": 8.5,
    "beca": true
  },
  {
    "id": 2,
    "nombre": "López, Juan",
    "ciudad": "Barcelona",
    "cp": "08001",
    "nota": 6,
    "beca": false
  },
  {
    "id": 3,
    "nombre": "Eva \"la rápida\" Ruiz",
    "ciudad": "Sevilla",
    "cp": "41001",
    "nota": null,
    "beca": true
  },
  {
    "id": 5,
    "nombre": "Marta Díaz",
    "ciudad": "Bilbao",
    "cp": "48001",
    "nota": -0.5,
    "beca": false
  }
]
Registros: 4 · Líneas ignoradas: 1 (5)
```

Ejemplo: Espacios, barras y comillas sin cerrar

ENTRADA

```
codigo , descripcion , precio
A1 , "Cable HDMI 2m" , 7.95
A2,"Ruta C:\temp",0
A3,"Sin cerrar,1

A4,,
```

SALIDA ESPERADA

```
[
  {
    "codigo": "A1",
    "descripcion": "Cable HDMI 2m",
    "precio": 7.95
  },
  {
    "codigo": "A2",
    "descripcion": "Ruta C:\\temp",
    "precio": 0
  },
  {
    "codigo": "A4",
    "descripcion": null,
    "precio": null
  }
]
Registros: 3 · Líneas ignoradas: 1 (4)
```

Código de partida (java)

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Main {
    /** Campos de una línea CSV separados por comas, con comillas dobles opcionales ("" es una comilla).
        null si hay comillas sin cerrar. */
    static List<String> campos(String linea) {
        // TODO
        return new ArrayList<>();
    }

    /** Texto JSON entre comillas, con las comillas, barras y tabuladores escapados. */
    static String cadena(String s) {
        // TODO
        return "\"" + s + "\"";
    }

    /** El valor JSON de un campo: null, número, booleano o texto. */
    static String valor(String campo) {
        // TODO
        return cadena(campo);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // TODO: lee la cabecera y cada línea, y escribe el JSON y el resumen
    }
}
```

147. Nóminas de una plantilla con distintos contratos

Difícil · 80 minutos · apuntesdam.com/ejercicios/java/nominas-de-una-plantilla

Una pequeña empresa calcula las nóminas con una hoja de cálculo que nadie se atreve a tocar. Quiere sustituirla por un programa que reciba la plantilla del mes y saque la tabla de nóminas.

Hay cuatro tipos de contrato y cada uno calcula el bruto a su manera, pero las deducciones son iguales para todos: una retención de IRPF según el tramo del bruto y una cotización a la Seguridad Social. Excepto los becarios, a los que en este ejercicio no se les retiene IRPF. Es el caso de libro para una interfaz con métodos por defecto que una clase concreta puede sobrescribir.

Las cifras están simplificadas (una tabla de retenciones de la empresa en lugar del cálculo real del IRPF, y un único porcentaje de cotización): lo que importa es la estructura del programa y que cada céntimo cuadre.

Requisitos

- Cada línea es `TIPO datos`, con los datos separados por `;`: `ASALARIADO nombre;salario anual;pagas` (12 o 14 pagas), `HORAS nombre;horas;precio por hora` (horas enteras, como mucho 300), `COMERCIAL nombre;base;ventas;comisión %` (comisión hasta el 20 %) y `BECARIO nombre;ayuda`. Las cantidades usan punto decimal y hasta dos decimales. El tipo puede ir en minúsculas.
- Una línea con un tipo desconocido escribe `Línea N ignorada: tipo desconocido TIPO`; con más o menos datos de los que le tocan (o sin nombre), `Línea N ignorada: faltan o sobran datos`; con un número mal escrito o fuera de rango, `Línea N ignorada: dato no válido`. N es el número de línea en la entrada (las vacías también cuentan). Estos avisos salen antes que la tabla.
- Bruto mensual: asalariado, salario anual ÷ pagas; por horas, las primeras 160 horas al precio normal y las siguientes al 125 %; comercial, base + comisión sobre las ventas, con la comisión limitada a 2.000 €; becario, la ayuda.
- Retención de IRPF, un único tipo sobre todo el bruto: hasta 1.000 €, el 2 %; hasta 2.000 €, el 12 %; hasta 3.000 €, el 18 %; por encima, el 24 %. Los becarios no tienen retención. Seguridad Social: el 6,5 % del bruto para todos. Neto = bruto – IRPF – Seguridad Social.
- Todos los cálculos se hacen en céntimos y cada porcentaje se redondea al céntimo más cercano (las mitades suben).
- Tabla: la cabecera `Empleado`, `Tipo`, `Bruto`, `IRPF`, `SS`, `Neto` y una fila por empleado en el orden de la entrada, con el nombre en 20 caracteres y el tipo (`asalariado`, `por horas`, `comercial`, `becario`) en 12, alineados a la izquierda, y los cuatro importes en 11 caracteres a la derecha, con punto de millar y coma decimal y sin símbolo. Después, una fila `TOTAL` con las sumas (el tipo en blanco).
- Para terminar, `Coste para la empresa: X €` (el bruto total más un 30 % de cotización de la empresa) y `Mayor neto: nombre (X €)` (el primero si hay empate). Si no hay ningún empleado válido, después de los avisos solo `No hay nóminas que calcular`.

Formato de la entrada

- Una línea por empleado: `TIPO nombre;dato;dato...`.

Retención de IRPF (tabla de la empresa)

Bruto mensual	Tipo
hasta 1.000,00 €	2 %
de 1.000,01 € a 2.000,00 €	12 %
de 2.000,01 € a 3.000,00 €	18 %
más de 3.000,00 €	24 %

Ejemplo: Una plantilla con los cuatro contratos

ENTRADA

ASALARIADO Ana Gil;30000;14
 ASALARIADO Luis Mora;48000;12
 HORAS Eva Ruiz;172;12.50
 COMERCIAL Pablo Sanz;1100;42000;3.5
 BECARIO Irene Vidal;600

SALIDA ESPERADA

Empleado	SS	Tipo	Bruto
IRPF		Neto	
Ana Gil		asalariado	2.142,86
385,71	139,29	1.617,86	
Luis Mora		asalariado	4.000,00
960,00	260,00	2.780,00	
Eva Ruiz		por horas	2.187,50
393,75	142,19	1.651,56	
Pablo Sanz		comercial	2.570,00
462,60	167,05	1.940,35	
Irene Vidal		becario	600,00
0,00	39,00	561,00	
TOTAL			11.500,36
2.202,06	747,53	8.550,77	
Coste para la empresa: 14.950,47 €			
Mayor neto: Luis Mora (2.780,00 €)			

Ejemplo: Líneas que no se pueden calcular

ENTRADA

ASALARIADO Mar Gómez;25000;13
 JEFE Pedro;5000
 HORAS Ana;abc;10

 COMERCIAL Leo;1000;5000
 becario Sol;450.5

SALIDA ESPERADA

Línea 1 ignorada: dato no válido
 Línea 2 ignorada: tipo desconocido JEFE
 Línea 3 ignorada: dato no válido
 Línea 5 ignorada: faltan o sobran datos

Empleado	SS	Tipo	Bruto
IRPF		Neto	
Sol		becario	450,50
0,00	29,28	421,22	
TOTAL			450,50
0,00	29,28	421,22	
Coste para la empresa: 585,65 €			
Mayor neto: Sol (421,22 €)			

Código de partida (java)

```
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;
import java.util.Scanner;

interface Retribuible {
    /** Bruto del mes, en céntimos. */
    long bruto();

    // TODO: métodos por defecto retencion(), seguridadSocial() y neto()
}

abstract class Empleado implements Retribuible {
    private final String nombre;

    Empleado(String nombre) { this.nombre = nombre; }

    String getNombre() { return nombre; }

    abstract String tipo();
}

// TODO: Asalariado, PorHoras, Comercial y Becario

public class Main {
    /** cantidad x centésimas / 10000, redondeado al céntimo (las mitades suben). */
    static long porcentaje(long cantidad, long centesimas) {
        return (cantidad * centesimas + 5000) / 10000;
    }

    static String euros(long c) {
        return String.format(Locale.ROOT, "%,d", c / 100).replace(',', '.') + "," + String.format("%02d", c % 100);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<Empleado> plantilla = new ArrayList<>();
        // TODO: lee y valida cada línea, y escribe la tabla de nóminas y el resumen
    }
}
```

148. Contador de palabras en paralelo con un pool de hilos

Difícil · 70 minutos · apuntesdam.com/ejercicios/java/contador-de-palabras-con-hilos

Contar las palabras de un texto enorme (las actas de un ayuntamiento, los registros de un servidor) se puede repartir: cada hilo cuenta un trozo y al final se suman los recuentos. Es el esquema «divide y vencerás» que usan desde los procesadores de varios núcleos hasta MapReduce en los centros de datos.

El error típico es que todos los hilos escriban en el mismo `HashMap`: no está preparado para accesos simultáneos y se pierden recuentos (o se corrompe). La forma limpia es que cada hilo cuente en su propio mapa y devuelva el resultado; el hilo principal es el único que los combina.

La prueba de que el programa es correcto es que el resultado no dependa del número de hilos ni del orden en que terminen: con 1, 3 u 8 hilos el informe de palabras tiene que ser exactamente el mismo.

Requisitos

- La primera línea es el número de hilos, de 1 a 8 (si no, `Número de hilos no válido: debe estar entre 1 y 8` y nada más). El resto es el texto; las líneas en blanco no cuentan. Si no queda ninguna línea, `No hay texto que analizar`.
- Se usan tantos hilos como se pidan, pero nunca más que líneas. Las líneas se reparten en bloques consecutivos lo más iguales posible: si no salen exactos, los primeros bloques tienen una línea más. Cada bloque lo cuenta un hilo de un `ExecutorService`.
- Una palabra es una secuencia de letras (de la a a la z, las vocales con tilde, la ü y la ñ), sin distinguir mayúsculas; cualquier otro carácter separa palabras.

- Escribe `Hilos: H · Líneas: L` y una línea por bloque, en orden, con dos espacios delante: `Bloque [i-j]: N palabras` (o `[i]` si el bloque tiene una sola línea; las líneas se numeran desde 1 sin contar las vacías).
- Después, `Palabras: N · Distintas: D` con el total de palabras y de palabras distintas, y `Más frecuentes:` seguido de las 5 palabras más repetidas (o las que haya) sin contar las palabras vacías de la tabla, cada una como el número en 4 caracteres alineado a la derecha, un espacio y la palabra. A igual número, por orden alfabético según `compareTo`.

Formato de la entrada

- Línea 1: número de hilos (1 a 8). Resto: el texto.

Palabras vacías (no salen en «Más frecuentes»)

Palabras

de, la, que, el, en, y, a, los, se, del, las, un, por, con, no, una, su, para, es, al, lo, como

Ejemplo: Un texto con tres hilos

ENTRADA

```
3
La programación concurrente permite que varios
hilos trabajen a la vez.
Cada hilo cuenta las palabras de su bloque de
líneas.
Si los hilos comparten un mapa sin protección, se
pierden datos.
Por eso cada hilo usa un mapa propio y devuelve el
resultado.
El hilo principal espera con get() y suma los
mapas parciales.
Con un hilo o con ocho hilos, el resultado debe
ser el mismo.
La concurrencia bien hecha no cambia el resultado:
solo el tiempo.
```

SALIDA ESPERADA

```
Hilos: 3 · Líneas: 7
Bloque [1-3]: 32 palabras
Bloque [4-5]: 23 palabras
Bloque [6-7]: 24 palabras
Palabras: 79 · Distintas: 56
Más frecuentes:
4 hilo
3 hilos
3 resultado
2 cada
2 mapa
```

Ejemplo: Número de hilos no válido

ENTRADA

```
12
Un texto cualquiera
```

SALIDA ESPERADA

```
Número de hilos no válido: debe estar entre 1 y 8
```

Código de partida (java)

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Scanner;
import java.util.Set;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class Main {
    static final Set<String> VACIAS = Set.of("de", "la", "que", "el", "en", "y", "a", "los", "se", "del",
        "las", "un",
        "por", "con", "no", "una", "su", "para", "es", "al", "lo", "como");

    /** Lo que hace cada hilo: contar las palabras de su bloque en un mapa propio, sin compartir nada. */
    static Map<String, Integer> contar(List<String> lineas) {
        // TODO
        return new HashMap<>();
    }

    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        // TODO: lee el número de hilos y el texto, reparte los bloques en un pool de hilos y une los
        resultados
    }
}
```

149. Validador de DNI, NIE e IBAN diseñado con clases de equivalencia

Medio · 50 minutos · apuntesdam.com/ejercicios/java/validador-dni-nie-iban

Casi todas las aplicaciones de gestión piden un DNI, un NIE o un IBAN, y casi todas los validan mal: aceptan letras que no corresponden, rechazan minúsculas o se atascan con los espacios del IBAN. Los tres llevan un control matemático que permite detectar la mayoría de los errores al teclear.

La letra del DNI es el resto de dividir el número entre 23, buscado en la tabla TRWAGMYFPDXBNJZSQVHLCKE. El NIE de los extranjeros funciona igual cambiando la X, la Y y la Z iniciales por 0, 1 y 2. El IBAN usa otro método (ISO 13616): se pasan los cuatro primeros caracteres al final, cada letra se sustituye por un número (A = 10, B = 11... Z = 35) y el número resultante, de unas 26 cifras, tiene que dar resto 1 al dividirlo entre 97.

Este ejercicio es también de Entornos de Desarrollo: las pruebas no se han elegido al azar, sino con la técnica de clases de equivalencia (un caso representativo de cada grupo de entradas que el programa trata igual) y valores límite (los bordes de cada grupo). Tienes la tabla abajo: tu programa tiene que pasar todos esos casos.

Requisitos

- Cada línea es `TIPO valor`, con `TIPO` `DNI`, `NIE` o `IBAN` en mayúsculas o minúsculas. Las líneas vacías se ignoran. Para cada línea escribe `TIPO valor → válido` o `TIPO valor → no válido: motivo`, con el tipo en mayúsculas y el valor tal como venía (sin espacios al principio ni al final).
- DNI: 8 cifras y una letra (mayúscula o minúscula). Si no tiene ese formato, `formato incorrecto (8 cifras y una letra)`; si la letra no es la que toca, `la letra debería ser L`.
- NIE: X, Y o Z, 7 cifras y una letra. Si no tiene ese formato, `formato incorrecto (X, Y o Z, 7 cifras y una letra)`; si la letra no cuadra, el mismo mensaje que el DNI.
- IBAN (se ignoran los espacios y las minúsculas): si no empieza por ES, `solo se admiten IBAN españoles`; si no tiene 24 caracteres, `un IBAN español tiene 24 caracteres`; si tras ES hay algo que no sea una cifra, `después de ES solo puede haber cifras`; si el control módulo 97 no da 1, `los dígitos de control no cuadran`. Se comprueba en ese orden.
- Un tipo desconocido escribe `TIPO valor → no válido: tipo de documento desconocido`.
- Al final, `Válidos: V de T`.

Formato de la entrada

- Una línea por documento: `DNI valor`, `NIE valor` o `IBAN valor`.

Clases de equivalencia y valores límite del DNI

Clase	Caso de prueba	Resultado esperado
Válida: 8 cifras y su letra	12345678Z	válido
Válida: letra en minúscula	12345678z	válido
Límite inferior del número	00000000T	válido
Límite superior del número	99999999R	válido
No válida: letra que no toca	12345678A	la letra debería ser Z
No válida: 7 cifras	1234567Z	formato incorrecto
No válida: 9 cifras	123456789Z	formato incorrecto
No válida: sin letra	12345678	formato incorrecto

Clases de equivalencia del IBAN

Clase	Caso de prueba	Resultado esperado
Válida, con espacios	ES91 2100 0418 4502 0005 1332	válido
Válida, en minúsculas y sin espacios	es9121000418450200051332	válido
Otro país	DE89370400440532013000	solo se admiten IBAN españoles
Longitud incorrecta	ES912100041845020005133	24 caracteres
Letras tras ES	ES91 2100 0418 4502 0005 13AB	solo cifras
Control incorrecto (una cifra cambiada)	ES91 2100 0418 4502 0005 1333	no cuadran

Ejemplo: DNI y NIE

ENTRADA

DNI 12345678Z
DNI 12345678z
DNI 12345678A
DNI 1234567Z
NIE X1234567L
NIE W1234567L
PASAPORTE AB123456

SALIDA ESPERADA

DNI 12345678Z → válido
DNI 12345678z → válido
DNI 12345678A → no válido: la letra debería ser Z
DNI 1234567Z → no válido: formato incorrecto (8 cifras y una letra)
NIE X1234567L → válido
NIE W1234567L → no válido: formato incorrecto (X, Y o Z, 7 cifras y una letra)
PASAPORTE AB123456 → no válido: tipo de documento desconocido
Válidos: 3 de 7

Ejemplo: IBAN

ENTRADA

IBAN ES91 2100 0418 4502 0005 1332
iban es9121000418450200051332
IBAN DE89370400440532013000
IBAN ES912100041845020005133
IBAN ES91 2100 0418 4502 0005 13AB
IBAN ES91 2100 0418 4502 0005 1333

SALIDA ESPERADA

IBAN ES91 2100 0418 4502 0005 1332 → válido
IBAN es9121000418450200051332 → válido
IBAN DE89370400440532013000 → no válido: solo se admiten IBAN españoles
IBAN ES912100041845020005133 → no válido: un IBAN español tiene 24 caracteres
IBAN ES91 2100 0418 4502 0005 13AB → no válido: después de ES solo puede haber cifras
IBAN ES91 2100 0418 4502 0005 1333 → no válido: los dígitos de control no cuadran
Válidos: 2 de 6

```
import java.util.Scanner;

public class Main {
    static final String LETRAS = "TRWAGMYFPDXBNJZSQVHLCKE";

    /** null si el DNI es correcto, o el motivo por el que no lo es. */
    static String validarDni(String dni) {
        // TODO
        return null;
    }

    /** El NIE se valida como un DNI cambiando X, Y y Z por 0, 1 y 2. */
    static String validarNie(String nie) {
        // TODO
        return null;
    }

    /** IBAN español: 24 caracteres y control módulo 97 (ISO 13616). */
    static String validarIban(String iban) {
        // TODO
        return null;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // TODO: valida cada línea y escribe el resultado y el resumen
    }
}
```

150. Fichero de productos con registros de tamaño fijo y acceso directo

Muy difícil · 90 minutos · apuntesdam.com/ejercicios/java/fichero-de-registros-con-acceso-directo

Antes de las bases de datos, los programas de gestión guardaban sus datos en ficheros binarios con registros de tamaño fijo. Todavía hoy es la forma de entender cómo trabaja un gestor de bases de datos por dentro: si cada registro ocupa siempre lo mismo, la posición del registro número N se calcula con una multiplicación y se puede ir directamente a él, sin leer los anteriores.

En este fichero cada producto ocupa 77 bytes: el código (`int` , 4 bytes), el nombre (30 caracteres `char` de 2 bytes, 60 en total, rellenando con el carácter `\0`), el precio (`double` , 8), el stock (`int` , 4) y una marca de borrado (`boolean` , 1). El producto con código N vive en el byte $(N - 1) \times 77$: es acceso directo, y por eso puede haber huecos entre productos.

Borrar un registro desplazando todos los siguientes sería lentísimo en un fichero grande, así que las bajas son lógicas: se pone la marca de borrado a `true` y el hueco queda libre para un alta futura con ese código.

Requisitos

- El programa crea un fichero temporal vacío y ejecuta una orden por línea (la orden puede ir en minúsculas; las líneas vacías se ignoran; una orden desconocida escribe `Orden desconocida`).
- ALTA código;nombre;precio;stock** : el código es un entero de 1 a 9999, el nombre no puede estar vacío, el precio es un número no negativo con hasta dos decimales y el stock un entero no negativo (si no, `Datos no válidos`). Si ya hay un producto activo con ese código, `Ya existe el producto N` . Si no, se escribe el registro en su posición y se escribe `Alta del producto N en el byte B` , añadiendo `(nombre recortado a 30 caracteres)` si el nombre era más largo.
- PRECIO código precio** cambia solo el precio, sin tocar el resto del registro: `Precio del producto N: antes → después` (o `Datos no válidos` / `No existe el producto N`).
- BAJA código** pone la marca de borrado: `Baja lógica del producto N` , o `No existe el producto N` si no está activo. **VER código** escribe `N · nombre · precio · S uds` o `No existe el producto N` .
- LISTAR** escribe `Productos activos: A` y, con dos espacios delante, cada producto activo por orden de código con el formato de `VER` .
- INFORME** escribe `Fichero: T bytes, H huecos de registro, A activos, L libres o borrados · valor del stock X` , donde H es el tamaño del fichero entre 77 y el valor es la suma de `precio × stock` de los activos. Los precios se escriben con dos decimales, coma decimal y `€` .

- Un registro está activo si el fichero llega hasta él, el código guardado en él es el suyo (los huecos que se crean al escribir más allá del final están llenos de ceros) y no tiene la marca de borrado.

Formato de la entrada

- Una orden por línea: ALTA , PRECIO , BAJA , VER , LISTAR o INFORME .

Estructura de un registro (77 bytes)

Campo	Tipo	Bytes	Desplazamiento
código	int	4	0
nombre	30 × char	60	4
precio	double	8	64
stock	int	4	72
borrado	boolean	1	76

Ejemplo: Altas, cambios y bajas

ENTRADA

```
ALTA 1;Ratón inalámbrico;9.95;20
ALTA 3;Teclado mecánico;49.90;5
ALTA 1;Repetido;1;1
LISTAR
PRECIO 3 44.50
BAJA 1
VER 1
VER 3
VER 2
INFORME
```

SALIDA ESPERADA

```
Alta del producto 1 en el byte 0
Alta del producto 3 en el byte 154
Ya existe el producto 1
Productos activos: 2
  1 · Ratón inalámbrico · 9,95 € · 20 uds
  3 · Teclado mecánico · 49,90 € · 5 uds
Precio del producto 3: 49,90 € → 44,50 €
Baja lógica del producto 1
No existe el producto 1
3 · Teclado mecánico · 44,50 € · 5 uds
No existe el producto 2
Fichero: 231 bytes, 3 huecos de registro, 1
activos, 2 libres o borrados · valor del stock
222,50 €
```

Ejemplo: Datos que no valen

ENTRADA

```
ALTA 0;Cero;1;1
ALTA 10;;1;1
ALTA 10;Monitor;-5;1
ALTA 10;Monitor;99.999;1
ALTA abc
PRECIO 7 10
BAJA 7
listar
BORRAR 1
```

SALIDA ESPERADA

```
Datos no válidos
Datos no válidos
Datos no válidos
Datos no válidos
Datos no válidos
No existe el producto 7
No existe el producto 7
Productos activos: 0
Orden desconocida
```

Código de partida (java)

```
import java.io.File;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.util.Locale;
import java.util.Scanner;

public class Main {
    static final int NOMBRE = 30; // caracteres del nombre
    static final int TAM = 4 + NOMBRE * 2 + 8 + 4 + 1; // código + nombre + precio + stock + borrado = 77 bytes

    static RandomAccessFile f;

    /** Acceso directo: el registro del código N empieza en el byte (N - 1) x 77. */
    static long posicion(int codigo) {
        return (long) (codigo - 1) * TAM;
    }

    static String euros(double x) {
        return String.format(Locale.ROOT, "%.2f", x).replace('.', ',') + " €";
    }

    static void orden(String linea) throws IOException {
        // TODO: ALTA, PRECIO, BAJA, VER, LISTAR e INFORME con seek, read... y write...
    }

    public static void main(String[] args) throws IOException {
        File fichero = File.createTempFile("productos", ".dat");
        fichero.deleteOnExit();
        try (RandomAccessFile raf = new RandomAccessFile(fichero, "rw")) {
            f = raf;
            Scanner sc = new Scanner(System.in);
            while (sc.hasNextLine()) {
                String linea = sc.nextLine().trim();
                if (!linea.isEmpty()) orden(linea);
            }
        }
    }
}
```

151. Servidor de órdenes con sockets e hilos

Muy difícil · 100 minutos · apuntesdam.com/ejercicios/java/servidor-de-ordenes-con-sockets

Muchos servicios de Internet (SMTP, FTP, Redis, POP3) usan protocolos de texto: el cliente envía una orden por línea y el servidor responde con otra línea que empieza por un código (OK, ERR). Son fáciles de depurar, porque se pueden probar a mano con telnet o netcat.

Vas a escribir un servidor de este tipo en Java. Escucha en un puerto, acepta conexiones y atiende a cada cliente en un hilo distinto, para que un cliente lento no bloquee a los demás. Tiene un pequeño almacén clave-valor compartido por todas las conexiones, al estilo de Redis, así que tiene que ser seguro para varios hilos.

Para poder probarlo sin dos programas, el mismo `main` arranca el servidor en un hilo y hace de cliente: lee las órdenes de la entrada, las envía por el socket y escribe lo que contesta el servidor. Una línea `---` cierra la conexión y abre otra, como si llegara un cliente nuevo.

Requisitos

- El servidor escucha en `localhost` en un puerto libre (puerto 0) y atiende cada conexión en un hilo nuevo. Lee líneas en UTF-8 y responde a cada una con una línea. Cuando recibe `ADIOS` responde y cierra la conexión.
- Protocolo (las órdenes se reconocen en mayúsculas o minúsculas): `SUMA a b` → `OK suma` con enteros de hasta 9 cifras y signo opcional, o `ERR números no válidos`; `MAYUS texto` → `OK TEXTO`; `INVIERTE texto` → `OK otxet` (sin texto, `ERR falta el texto`); `GUARDA clave valor` (el valor puede tener espacios) → `OK guardado` u `OK sustituido` (antes: `valor`), y `ERR uso: GUARDA clave valor` si falta algo; `LEE clave` → `OK valor` o `ERR no existe clave`; `CUENTA` → `OK número de claves`; `ADIOS` → `OK adiós`. Una línea vacía es `ERR línea vacía` y cualquier otra orden `ERR orden desconocida`.

- El cliente conecta al empezar y escribe `== Cliente N conectado`. Por cada línea de la entrada (incluidas las vacías) la envía, espera la respuesta y escribe `> línea` (solo `>` si la línea está vacía) y `< respuesta`. Una línea `---` cierra la conexión actual y abre una nueva (con su mensaje de conexión); después de `ADIOS`, la siguiente línea también abre una conexión nueva.
- Al terminar la entrada, el cliente cierra su conexión, se cierra el servidor y se escribe `Servidor cerrado · claves guardadas: N`.
- El almacén es común a todas las conexiones: lo que guarda un cliente lo puede leer el siguiente.

Formato de la entrada

- Una orden del protocolo por línea; `---` abre una conexión nueva.

Ejemplo: Dos clientes que comparten el almacén

ENTRADA

```
SUMA 20 22
MAYUS hola, mundo
INVIERTE ñandú
GUARDA curso 2º DAM
GUARDA curso 1º DAW
CUENTA
ADIOS
LEE curso
GUARDA tutor Ana Gil
---
LEE tutor
LEE profe
CUENTA
```

SALIDA ESPERADA

```
== Cliente 1 conectado
> SUMA 20 22
< OK 42
> MAYUS hola, mundo
< OK HOLA, MUNDO
> INVIERTE ñandú
< OK údnañ
> GUARDA curso 2º DAM
< OK guardado
> GUARDA curso 1º DAW
< OK sustituido (antes: 2º DAM)
> CUENTA
< OK 1
> ADIOS
< OK adiós
== Cliente 2 conectado
> LEE curso
< OK 1º DAW
> GUARDA tutor Ana Gil
< OK guardado
== Cliente 3 conectado
> LEE tutor
< OK Ana Gil
> LEE profe
< ERR no existe profe
> CUENTA
< OK 2
Servidor cerrado · claves guardadas: 2
```

Ejemplo: Errores del protocolo

ENTRADA

```
SUMA 1
SUMA uno dos
MAYUS

GUARDA solo
BORRA todo
suma -5 3
```

SALIDA ESPERADA

```
== Cliente 1 conectado
> SUMA 1
< ERR números no válidos
> SUMA uno dos
< ERR números no válidos
> MAYUS
< ERR falta el texto
>
< ERR línea vacía
> GUARDA solo
< ERR uso: GUARDA clave valor
> BORRA todo
< ERR orden desconocida
> suma -5 3
< OK -2
Servidor cerrado · claves guardadas: 0
```


Código de partida (java)

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.charset.StandardCharsets;
import java.util.Map;
import java.util.Scanner;
import java.util.concurrent.ConcurrentHashMap;

public class Main {
    static final Map<String, String> almacen = new ConcurrentHashMap<>();

    /** La respuesta del protocolo a una línea. */
    static String responder(String linea) {
        // TODO: SUMA, MAYUS, INVIERTE, GUARDA, LEE, CUENTA y ADIOS
        return "ERR sin hacer";
    }

    /** Atiende a un cliente: una respuesta por línea hasta ADIOS o hasta que cierre la conexión. */
    static void atender(Socket socket) {
        // TODO
    }

    public static void main(String[] args) throws Exception {
        // TODO: arranca el servidor en un hilo, conecta el cliente y envía las líneas de la entrada
    }
}
```

152. Almacén de contraseñas seguro con PBKDF2 y sal

Difícil · 70 minutos · apuntesdam.com/ejercicios/java/almacen-de-contrasenas-con-pbkdf2

Guardar contraseñas en texto plano, o con un hash rápido como MD5 o SHA-256 sin más, es uno de los fallos de seguridad más graves y más frecuentes. Si alguien roba la base de datos, con un hash rápido puede probar miles de millones de contraseñas por segundo.

La forma correcta es usar una función de derivación lenta a propósito, como PBKDF2, bcrypt o Argon2, con una sal aleatoria distinta para cada usuario. La sal hace que dos usuarios con la misma contraseña tengan hashes distintos (y que no sirvan las tablas precalculadas), y las miles de iteraciones hacen que cada intento cueste tiempo.

El inicio de sesión también tiene sus reglas: comparar los hashes en tiempo constante, bloquear la cuenta tras varios fallos seguidos y responder lo mismo si el usuario no existe que si la contraseña es incorrecta, para que nadie pueda averiguar qué usuarios hay. Como la sal es aleatoria, el programa nunca escribe hashes: solo el resultado de cada operación.

Requisitos

- Cada línea es una orden con sus argumentos separados por espacios (la orden en mayúsculas o minúsculas; las líneas vacías se ignoran). Antes de responder se escribe `> ORDEN usuario:` (la orden en mayúsculas y solo el primer argumento, nunca la contraseña) y después la respuesta en la misma línea. Una orden desconocida responde `orden desconocida`.
- `REGISTRA usuario contraseña`: `el usuario ya existe`; o, si la contraseña no cumple la política, `contraseña rechazada, motivo` con el primer motivo: `debe tener al menos 10 caracteres`, `debe tener mayúsculas`, `minúsculas` y `números`, `no puede contener el nombre de usuario` (sin distinguir mayúsculas); si no, guarda una sal aleatoria de 16 bytes y el hash PBKDF2WithHmacSHA256 (10.000 iteraciones, 256 bits) y responde `registrado`.
- `ENTRA usuario contraseña`: si el usuario no existe o la contraseña no es correcta, `usuario o contraseña incorrectos`; al tercer fallo seguido, `usuario o contraseña incorrectos`; `cuenta bloqueada tras 3 fallos`; con la cuenta bloqueada, `cuenta bloqueada` aunque la contraseña sea buena; si es correcta, `bienvenido` y el contador de fallos vuelve a 0.

- **CAMBIA** usuario actual nueva : no se puede cambiar la contraseña si el usuario no existe, está bloqueado o la actual no es correcta; después, la política (contraseña rechazada, motivo) y la nueva contraseña debe ser distinta ; si todo va bien, nueva sal y nuevo hash, y contraseña cambiada .
- **DESBLOQUEA** usuario → desbloqueada (y fallos a 0) o no existe . **COMPARA** u1 u2 → ¿mismo hash que u2? sí/no o no existe alguno de los usuarios . **INFO** usuario → PBKDF2WithHmacSHA256, 10000 iteraciones, sal de 16 bytes, hash de 32 bytes (con , bloqueada al final si lo está) o no existe .
- Una orden con un número de argumentos que no es el suyo responde uso: ... con su formato (uso: REGISTRA usuario contraseña , uso: ENTRA usuario contraseña , uso: CAMBIA usuario actual nueva).
- Las contraseñas se comparan con MessageDigest.isEqual , nunca con Arrays.equals ni comparando textos.

Formato de la entrada

- Una orden por línea: REGISTRA , ENTRA , CAMBIA , DESBLOQUEA , COMPARA o INFO .

Ejemplo: Registro, inicio de sesión y bloqueo

ENTRADA

```
REGISTRA ana Corta1
REGISTRA ana sinmayusculas123
REGISTRA ana MiAnaClave2026
REGISTRA ana Gatos.Tigres.2026
REGISTRA eva Gatos.Tigres.2026
REGISTRA ana OtraClave99999
COMPARA ana eva
ENTRA ana gatos.tigres.2026
ENTRA nadie Gatos.Tigres.2026
ENTRA ana Gatos.Tigres.2026
ENTRA eva mal
ENTRA eva mal
ENTRA eva mal
ENTRA eva Gatos.Tigres.2026
INFO eva
```

SALIDA ESPERADA

```
> REGISTRA ana: contraseña rechazada, debe tener
al menos 10 caracteres
> REGISTRA ana: contraseña rechazada, debe tener
mayúsculas, minúsculas y números
> REGISTRA ana: contraseña rechazada, no puede
contener el nombre de usuario
> REGISTRA ana: registrado
> REGISTRA eva: registrado
> REGISTRA ana: el usuario ya existe
> COMPARA ana: ¿mismo hash que eva? no
> ENTRA ana: usuario o contraseña incorrectos
> ENTRA nadie: usuario o contraseña incorrectos
> ENTRA ana: bienvenido
> ENTRA eva: usuario o contraseña incorrectos
> ENTRA eva: usuario o contraseña incorrectos
> ENTRA eva: usuario o contraseña incorrectos;
cuenta bloqueada tras 3 fallos
> ENTRA eva: cuenta bloqueada
> INFO eva: PBKDF2WithHmacSHA256, 10000
iteraciones, sal de 16 bytes, hash de 32 bytes,
bloqueada
```

Ejemplo: Cambio de contraseña y desbloqueo

ENTRADA

```
REGISTRA luis Montañas.2026
CAMBIA luis Equivocada1 Nueva.Clave.27
CAMBIA luis Montañas.2026 Montañas.2026
CAMBIA luis Montañas.2026 corta
CAMBIA luis Montañas.2026 Nueva.Clave.27
ENTRA luis Montañas.2026
ENTRA luis Nueva.Clave.27
DESBLOQUEA luis
INFO luis
```

SALIDA ESPERADA

```
> REGISTRA luis: registrado
> CAMBIA luis: no se puede cambiar la contraseña
> CAMBIA luis: la nueva contraseña debe ser
distinta
> CAMBIA luis: contraseña rechazada, debe tener al
menos 10 caracteres
> CAMBIA luis: contraseña cambiada
> ENTRA luis: usuario o contraseña incorrectos
> ENTRA luis: bienvenido
> DESBLOQUEA luis: desbloqueada
> INFO luis: PBKDF2WithHmacSHA256, 10000
iteraciones, sal de 16 bytes, hash de 32 bytes
```

Código de partida (java)

```
import java.security.MessageDigest;
import java.security.SecureRandom;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;

public class Main {
    static final String ALGORITMO = "PBKDF2WithHmacSHA256";
    static final int ITERACIONES = 10_000;    // en producción, cientos de miles: aquí menos para que las
        pruebas vayan rápido
    static final int MAX_FALLOS = 3;

    static class Cuenta {
        byte[] sal;
        byte[] hash;
        int fallos;
        boolean bloqueada;
    }

    static final Map<String, Cuenta> cuentas = new HashMap<>();
    static final SecureRandom azar = new SecureRandom();

    static byte[] derivar(String contrasena, byte[] sal) throws Exception {
        PBEKeySpec spec = new PBEKeySpec(contrasena.toCharArray(), sal, ITERACIONES, 256);
        return SecretKeyFactory.getInstance(ALGORITMO).generateSecret(spec).getEncoded();
    }

    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        // TODO: REGISTRA, ENTRA, CAMBIA, DESBLOQUEA, COMPARA e INFO
    }
}
```

153. Depura el programa de notas: cinco errores escondidos

Medio · 45 minutos · apuntesdam.com/ejercicios/java/depura-el-calculo-de-notas

Un compañero ha escrito el programa que calcula las notas finales de Programación y lo ha dejado para que lo revises antes de publicar las notas. Compila, no lanza excepciones y con su ejemplo de prueba (tres alumnos con notas «normales») el resultado parecía razonable. Pero varios alumnos se han quejado de que su nota no cuadra.

La regla de evaluación es: cada una de las dos prácticas vale un 20 % y el examen un 60 %; si el examen tiene menos de un 4, la nota final no puede pasar de 4 (el examen no se compensa), y la nota se redondea a un decimal. Aprueba quien tiene un 5 o más.

El código del editor tiene exactamente cinco errores, cada uno de una línea. No está permitido reescribir el programa: hay que encontrar cada error y corregirlo. Es lo que se hace en un proyecto real, donde reescribir lo que no se entiende introduce errores nuevos.

Requisitos

- La entrada tiene una línea por alumno: `nombre;práctica1;práctica2;examen`, con notas de 0 a 10 con punto decimal.
- Nota final = $0,2 \times \text{práctica1} + 0,2 \times \text{práctica2} + 0,6 \times \text{examen}$; si el examen es menor que 4, la nota final es como mucho 4; se redondea a un decimal (las mitades suben).
- Se escribe una línea por alumno, de mayor a menor nota (a igualdad, por nombre): el nombre en 20 caracteres a la izquierda y la nota con un decimal y coma en 5 caracteres a la derecha.
- Al final, `Media: X` (la media de las notas finales con un decimal) y `Aprobados: A de T`.
- Corrige los cinco errores del programa sin cambiar su estructura. Las pruebas ocultas están pensadas para detectar cada uno.

Formato de la entrada

- Una línea por alumno: `nombre;práctica1;práctica2;examen`.

Ejemplo: Una clase de cinco alumnos

ENTRADA

```
Ana Gil;8;9;7.5
Luis Mora;6;5;4
Eva Ruiz;10;10;3.5
Pablo Sanz;5;5;5
Marta Peña;7.5;6.5;6.2
```

SALIDA ESPERADA

```
Ana Gil          7,9
Marta Peña       6,5
Pablo Sanz       5,0
Luis Mora        4,6
Eva Ruiz        4,0
Media: 5,6
Aprobados: 3 de 5
```

Ejemplo: El examen de 4 sí compensa

ENTRADA

```
Irene Vidal;9;9;4
Jorge Lara;3;3;3.9
```

SALIDA ESPERADA

```
Irene Vidal      6,0
Jorge Lara       3,5
Media: 4,8
Aprobados: 1 de 2
```

Código de partida (java)

```
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;
import java.util.Scanner;

public class Main {
    record Alumno(String nombre, double nota) {}

    /** Nota final: prácticas al 20 % cada una y examen al 60 %; con menos de un 4 en el examen, como mucho un 4. */
    static double notaFinal(double practica1, double practica2, double examen) {
        double nota = 0.2 * practica1 + 0.2 * practica2 + 0.6 * examen;
        if (examen <= 4) nota = Math.min(nota, 4);
        return Math.round(nota * 10) / 10;
    }

    static String uno(double x) {
        return String.format(Locale.ROOT, "%.1f", x).replace('.', ',');
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<String> lineas = new ArrayList<>();
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (!l.isEmpty()) lineas.add(l);
        }

        List<Alumno> alumnos = new ArrayList<>();
        for (int i = 1; i < lineas.size(); i++) {
            String[] p = lineas.get(i).split(";");
            alumnos.add(new Alumno(p[0], notaFinal(Double.parseDouble(p[1]), Double.parseDouble(p[2]),
                Double.parseDouble(p[3]))));
        }
        alumnos.sort((a, b) -> a.nota() != b.nota() ? Double.compare(b.nota(), a.nota()) :
            a.nombre().compareTo(b.nombre()));

        double suma = 0;
        int aprobados = 0;
        for (Alumno a : alumnos) {
            System.out.printf("%-20s%5s%n", a.nombre(), uno(a.nota()));
            suma += a.nota();
            if (a.nota() > 5) aprobados++;
        }
        System.out.println("Media: " + uno(suma / alumnos.size()));
        System.out.println("Aprobados: " + aprobados + " de " + alumnos.size());
    }
}
```

Soluciones

1. Gráfico de trabajo pendiente (burndown) de un sprint

Difícil · Desarrollo de Software · apuntesdam.com/subject/entornos/topic/desarrollo-software

Solución (java)

```
import java.util.*;

public class Main {
    record Historia(String id, int puntos, int diaTerminada) { } // diaTerminada = 0 si no se terminó

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int dias = sc.nextInt();
        List<Historia> hs = new ArrayList<>();
        while (sc.hasNext()) hs.add(new Historia(sc.next(), sc.nextInt(), sc.nextInt()));

        int total = 0;
        for (Historia h : hs) total += h.puntos();
        System.out.println("Sprint de " + dias + " días, " + total + " puntos comprometidos");

        for (int d = 0; d <= dias; d++) {
            int quedan = total;
            for (Historia h : hs) {
                if (h.diaTerminada() >= 1 && h.diaTerminada() <= d) quedan -= h.puntos(); // solo
                historias terminadas
            }
            double ideal = total - (double) total * d / dias;
            String barra = "#".repeat(quedan) + " ".repeat(total - quedan);
            String linea = String.format(Locale.US, "Día %2d | %s | %2d (ideal %.1f)", d, barra, quedan,
                ideal);
            if (quedan > Math.ceil(ideal)) linea += " (retraso)";
            System.out.println(linea);
        }

        int velocidad = 0;
        List<String> pendientes = new ArrayList<>();
        for (Historia h : hs) {
            if (h.diaTerminada() > 0) velocidad += h.puntos();
            else pendientes.add(h.id());
        }
        System.out.println("Velocidad: " + velocidad + " puntos");
        System.out.println("Vuelven al product backlog: " + (pendientes.isEmpty() ? "ninguna" :
            String.join(", ", pendientes)));
    }
}
```

Para cada día se parte del total comprometido y se restan los puntos de las historias terminadas ese día o antes; las no terminadas (día 0) nunca restan. La barra son tantos # como puntos quedan, rellena con espacios hasta el total para que las columnas queden alineadas.

El ideal es una recta que baja de T a 0: $T - T \cdot d / D$, con la división en `double`. Comparar con el ideal redondeado hacia arriba (`Math.ceil`) evita marcar retraso por una fracción de punto, ya que las historias no se terminan a medias.

2. Planificación con dependencias y camino crítico

Muy difícil · Desarrollo de Software · apuntesdam.com/subject/entornos/topic/desarrollo-software

Solución (java)

```
import java.util.*;

public class Main {
    record Tarea(String id, int duracion, List<String> previas) { }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, Tarea> tareas = new LinkedHashMap<>();
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (l.isEmpty()) continue;
            // Formato: ID duración [previa1,previa2,...]
            String[] p = l.split("\\s+");
            List<String> previas = p.length > 2 ? List.of(p[2].split(",")) : List.of();
            tareas.put(p[0], new Tarea(p[0], Integer.parseInt(p[1]), previas));
        }

        // Quién depende de cada tarea y cuántas previas le faltan
        Map<String, List<String>> dependientes = new LinkedHashMap<>();
        Map<String, Integer> pendientes = new HashMap<>();
        for (Tarea t : tareas.values()) dependientes.put(t.id(), new ArrayList<>());
        for (Tarea t : tareas.values()) {
            for (String p : t.previas()) {
                if (!tareas.containsKey(p)) {
                    System.out.println("Error: " + t.id() + " depende de " + p + ", que no existe");
                    return;
                }
                dependientes.get(p).add(t.id());
            }
            pendientes.put(t.id(), t.previas().size());
        }

        // Algoritmo de Kahn
        Deque<String> cola = new ArrayDeque<>();
        for (Tarea t : tareas.values()) if (t.previas().isEmpty()) cola.add(t.id());
        List<String> orden = new ArrayList<>();
        while (!cola.isEmpty()) {
            String t = cola.poll();
            orden.add(t);
            for (String d : dependientes.get(t)) {
                pendientes.merge(d, -1, Integer::sum);
                if (pendientes.get(d) == 0) cola.add(d);
            }
        }
        if (orden.size() < tareas.size()) {
            System.out.println("Error: hay dependencias circulares");
            return;
        }

        // Fechas tempranas (hacia delante)
        Map<String, Integer> inicio = new HashMap<>(), fin = new HashMap<>();
        int duracion = 0;
        for (String id : orden) {
            int ini = 0;
            for (String p : tareas.get(id).previas()) ini = Math.max(ini, fin.get(p));
            inicio.put(id, ini);
            fin.put(id, ini + tareas.get(id).duracion());
            duracion = Math.max(duracion, fin.get(id));
        }
        // Fin más tardío (hacia atrás)
        Map<String, Integer> finTardio = new HashMap<>();
        for (int i = orden.size() - 1; i >= 0; i--) {
            String id = orden.get(i);
            int ft = duracion;
            for (String d : dependientes.get(id)) ft = Math.min(ft, finTardio.get(d) -
                tareas.get(d).duracion());
            finTardio.put(id, ft);
        }
    }
}
```

```

        System.out.println("Orden: " + String.join(" ", orden));
        List<String> critico = new ArrayList<>();
        for (String id : orden) {
            int holgura = finTardio.get(id) - fin.get(id);
            System.out.printf("%-3s inicio %2d fin %2d holgura %d\n", id, inicio.get(id), fin.get(id), holgura);
            if (holgura == 0) critico.add(id);
        }
        System.out.println("Duración del proyecto: " + duracion + " días");
        System.out.println("Camino crítico: " + String.join(" → ", critico));
    }
}

```

El algoritmo de Kahn lleva la cuenta de cuántas previas le faltan a cada tarea. Empiezan las que no tienen ninguna; cada vez que una sale de la cola, sus dependientes tienen una previa menos y, al llegar a 0, entran en la cola. Si al final no han salido todas, las que quedan se esperan unas a otras: hay un ciclo.

Con el orden topológico, las fechas se calculan en dos pasadas. Hacia delante, cada tarea empieza cuando termina la última de sus previas. Hacia atrás, cada tarea debe terminar antes de que tenga que empezar la más urgente de sus dependientes (su fin tardío menos su duración); las que no tienen dependientes pueden acabar al final del proyecto.

La holgura es cuánto puede retrasarse una tarea sin retrasar el proyecto. Las de holgura 0 forman el camino crítico: cualquier retraso en ellas se nota en la fecha final.

3. Depura el resumen de notas

Difícil · Instalación y uso de IDEs · apuntesdam.com/subject/entornos/topic/ides

Solución (java)

```

import java.util.*;

public class Main {
    static String resumen(int[] notas) {
        int suma = 0, aprobados = 0, max = notas[0], min = notas[0]; // 3: empezar con una nota real, no con 0
        for (int i = 0; i < notas.length; i++) { // 1: el primer índice es 0
            suma += notas[i];
            if (notas[i] >= 5) aprobados++; // 2: un 5 también aprueba
            if (notas[i] > max) max = notas[i];
            if (notas[i] < min) min = notas[i];
        }
        double media = (double) suma / notas.length; // 4: división con decimales
        return String.format(Locale.US, "media %.2f, aprobados %d de %d, máxima %d, mínima %d",
            media, aprobados, notas.length, max, min);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (l.isEmpty()) continue;
            int[] notas = Arrays.stream(l.split("\\s+")).mapToInt(Integer::parseInt).toArray();
            System.out.println(Arrays.toString(notas) + " → " + resumen(notas));
        }
    }
}

```

Los cuatro errores se ven en cuanto se observa el bucle con el depurador: (1) empieza en `i = 0`, así que la primera nota nunca se cuenta; (2) `>= 5` deja fuera el 5, que es un aprobado; (3) `min` empieza en 0 y ninguna nota es menor que 0, así que la mínima siempre sale 0; y (4) `suma / notas.length` es una división entera y pierde los decimales.

La corrección de `min` (y de `max`, por coherencia) es empezar con una nota real, `notas[0]`. Con un punto de ruptura dentro del bucle se ve enseguida que `min` no cambia nunca, y en la primera parada, que `i` ya vale 1.

4. Depura una búsqueda binaria

Muy difícil · Instalación y uso de IDEs · apuntesdam.com/subject/entornos/topic/ides

Solución (java)

```
import java.util.*;

public class Main {
    // Primera posición i tal que a[i] >= x (a.length si no hay ninguna). El array está ordenado.
    static int primeraMayorOIgual(int[] a, int x) {
        int lo = 0, hi = a.length;           // hi = a.length: la respuesta puede ser «ninguna»
        while (lo < hi) {
            int medio = lo + (hi - lo) / 2;    // sin desbordamiento, a diferencia de (lo + hi) / 2
            if (a[medio] < x) lo = medio + 1;  // medio no es la respuesta: se descarta
            else hi = medio;                  // medio puede ser la respuesta: se conserva
        }
        return lo;
    }

    static int contar(int[] a, int x) {
        return primeraMayorOIgual(a, x + 1) - primeraMayorOIgual(a, x);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int[] a = Arrays.stream(sc.nextLine().trim().split("\\s+")).mapToInt(Integer::parseInt).toArray();
        while (sc.hasNextInt()) {
            int x = sc.nextInt();
            System.out.println("x=" + x + ": primera posición >= x → " + primeraMayorOIgual(a, x) + ",
                               apariciones → " + contar(a, x));
        }
    }
}
```

La clave de una búsqueda binaria correcta es el invariante: la respuesta está siempre en el intervalo $[lo, hi]$. Como puede ser `a.length` (todos menores), `hi` tiene que empezar en `a.length` y no en `a.length - 1`; ese era el motivo de que nunca devolviera `a.length`.

En cada vuelta, si `a[medio] < x` la respuesta está estrictamente a la derecha (`lo = medio + 1`; con `lo = medio` el intervalo podía no encogerse y se quedaba en bucle); si no, `medio` es candidata y se conserva con `hi = medio` (con `medio - 1` se perdía). El bucle termina cuando `lo == hi`.

`(lo + hi) / 2` se desborda si la suma pasa de `Integer.MAX_VALUE`; `lo + (hi - lo) / 2` calcula lo mismo sin sumar dos índices grandes.

5. Valores límite: casos que detectan errores

Difícil · Diseño y realización de pruebas · apuntesdam.com/subject/entornos/topic/disenio-pruebas

Solución (java)

```
import java.util.*;
import java.util.function.DoubleUnaryOperator;

public class Main {
    // Tarifa de envío según el peso en kg. -1 significa «no admitido».
    // peso <= 0      → -1
    // 0 < peso <= 1  → 4.5
    // 1 < peso <= 5  → 7
    // 5 < peso <= 20 → 12
    // peso > 20     → -1
    static double tarifa(double peso) {
        if (peso <= 0) return -1;
        if (peso <= 1) return 4.5;
        if (peso <= 5) return 7;
        if (peso <= 20) return 12;
        return -1;
    }

    // Cada límite se prueba justo en él y justo por encima
    static final double[][] CASOS = {
        {0, -1},          // límite inferior: 0 no se admite (detecta M4)
        {0.01, 4.5},
        {1, 4.5},         // 1 kg aún es el primer tramo (detecta M1)
        {1.01, 7},        // primer valor del segundo tramo (detecta M6)
        {5, 7},           // 5 kg aún es el segundo tramo (detecta M2)
        {5.01, 12},
        {20, 12},         // 20 kg aún se admite (detecta M3)
        {20.01, -1},     // por encima de 20 no se admite (detecta M5)
    };

    // ---- No modifique lo que sigue: versiones con un error cada una («mutantes»)
    static final Map<String, DoubleUnaryOperator> MUTANTES = new LinkedHashMap<>();
    static {
        MUTANTES.put("M1 (< 1 en vez de <= 1)", p -> p <= 0 ? -1 : p < 1 ? 4.5 : p <= 5 ? 7 : p <= 20 ? 12 : -1);
        MUTANTES.put("M2 (< 5 en vez de <= 5)", p -> p <= 0 ? -1 : p <= 1 ? 4.5 : p < 5 ? 7 : p <= 20 ? 12 : -1);
        MUTANTES.put("M3 (< 20 en vez de <= 20)", p -> p <= 0 ? -1 : p <= 1 ? 4.5 : p <= 5 ? 7 : p < 20 ? 12 : -1);
        MUTANTES.put("M4 (< 0 en vez de <= 0)", p -> p < 0 ? -1 : p <= 1 ? 4.5 : p <= 5 ? 7 : p <= 20 ? 12 : -1);
        MUTANTES.put("M5 (sin límite superior)", p -> p <= 0 ? -1 : p <= 1 ? 4.5 : p <= 5 ? 7 : 12);
        MUTANTES.put("M6 (7.5 en vez de 7)", p -> p <= 0 ? -1 : p <= 1 ? 4.5 : p <= 5 ? 7.5 : p <= 20 ? 12 : -1);
    }

    public static void main(String[] args) {
        String fallo = null;
        for (double[] c : CASOS)
            if (tarifa(c[0]) != c[1] && fallo == null) fallo = "tarifa(" + c[0] + ") da " + tarifa(c[0]) + ", no " + c[1];
        System.out.println("Casos correctos con el original: " + (fallo == null ? "sí" : "no (" + fallo + ")"));
        List<String> vivos = new ArrayList<>();
        for (var m : MUTANTES.entrySet()) {
            boolean detectado = false;
            for (double[] c : CASOS) if (m.getValue().applyAsDouble(c[0]) != c[1]) detectado = true;
            if (!detectado) vivos.add(m.getKey());
        }
        System.out.println("Mutantes detectados: " + (MUTANTES.size() - vivos.size()) + " de " + MUTANTES.size());
        System.out.println("Sobreviven: " + (vivos.isEmpty() ? "ninguno" : String.join(", ", vivos)));
    }
}
```

Un `<` en lugar de `<=` solo cambia el resultado exactamente en el límite, así que un caso en mitad del tramo (como el 3) no detecta nada. Por eso se prueba cada frontera dos veces: justo en el límite (0, 1, 5, 20) y justo por encima (0,01, 1,01, 5,01, 20,01).

Cada caso se explica por el error que detecta: el 0 caza `< 0`, el 1 caza `< 1`, el 20,01 caza el límite superior olvidado y cualquier peso del segundo tramo caza el precio equivocado. Es la idea del análisis de valores límite: pocos casos, pero elegidos donde los errores son probables.

6. Matar mutantes con varias condiciones

Muy difícil · Diseño y realización de pruebas · apuntesdam.com/subject/entornos/topic/disenio-pruebas

Solución (java)

```
import java.util.*;

public class Main {
    interface Precio { double calcular(int edad, boolean socio, boolean finDeSemana); }

    // Precio de la entrada a un museo:
    //  edad < 0 → -1 (error)
    //  base 10 €; los fines de semana, 12 €
    //  menores de 12: 50 % de descuento; desde 65: 40 % de descuento
    //  los socios tienen además 2 € menos sobre el precio ya descontado (nunca por debajo de 0)
    static final Precio ORIGINAL = (edad, socio, finde) -> {
        if (edad < 0) return -1;
        double p = finde ? 12 : 10;
        if (edad < 12) p *= 0.5;
        else if (edad >= 65) p *= 0.6;
        if (socio) p = Math.max(0, p - 2);
        return p;
    };

    // {edad, socio (0/1), fin de semana (0/1), precio esperado}
    static final double[][] CASOS = {
        {30, 0, 0, 10}, // adulto entre semana
        {30, 0, 1, 12}, // adulto en fin de semana
        {-1, 0, 0, -1}, // edad no válida
        {0, 0, 0, 5}, // 0 años es válido (detecta M3)
        {11, 0, 0, 5}, // último año de niño
        {12, 0, 0, 10}, // 12 ya no es niño (detecta M1)
        {64, 0, 0, 10}, // último año sin descuento de mayor
        {65, 0, 0, 6}, // 65 ya tiene el 40 % (detecta M2 y M8)
        {10, 0, 1, 6}, // niño en fin de semana: 50 % de 12 (detecta M6)
        {10, 1, 0, 3}, // niño socio: 5 - 2 (detecta M4 y M7)
        {70, 1, 1, 5.2}, // mayor socio en fin de semana: 7,2 - 2
        {30, 1, 0, 8}, // adulto socio
    };

    // ---- No modifique lo que sigue
    static final Map<String, Precio> MUTANTES = new LinkedHashMap<>();
    static {
        MUTANTES.put("M1 (<= 12)", (e, s, f) -> e < 0 ? -1 : aplicar(f ? 12 : 10, e <= 12, e >= 65, s));
        MUTANTES.put("M2 (> 65)", (e, s, f) -> e < 0 ? -1 : aplicar(f ? 12 : 10, e < 12, e > 65, s));
        MUTANTES.put("M3 (<= 0 es error)", (e, s, f) -> e <= 0 ? -1 : aplicar(f ? 12 : 10, e < 12, e >= 65, s));
        MUTANTES.put("M4 (socio antes del %)", (e, s, f) -> {
            if (e < 0) return -1;
            double p = (f ? 12 : 10) - (s ? 2 : 0);
            return e < 12 ? p * 0.5 : e >= 65 ? p * 0.6 : p;
        });
        MUTANTES.put("M5 (sin tope en 0)", (e, s, f) -> {
            if (e < 0) return -1;
            double p = f ? 12 : 10;
            if (e < 12) p *= 0.5; else if (e >= 65) p *= 0.6;
            return s ? p - 2 : p;
        });
        MUTANTES.put("M6 (ignora el fin de semana con descuento)", (e, s, f) -> e < 0 ? -1 : aplicar((f && e >= 12 && e < 65) ? 12 : 10, e < 12, e >= 65, s));
        MUTANTES.put("M7 (el socio solo descuenta a adultos)", (e, s, f) -> e < 0 ? -1 : aplicar(f ? 12 : 10, e < 12, e >= 65, s && e >= 12 && e < 65));
        MUTANTES.put("M8 (65 o más: 50 %)", (e, s, f) -> {
            if (e < 0) return -1;
            double p = f ? 12 : 10;
            if (e < 12 || e >= 65) p *= 0.5;
            if (s) p = Math.max(0, p - 2);
            return p;
        });
    }

    static double aplicar(double p, boolean nino, boolean mayor, boolean socio) {
        if (nino) p *= 0.5; else if (mayor) p *= 0.6;
    }
}
```

```

        if (socio) p = Math.max(0, p - 2);
        return p;
    }

    static boolean igual(double a, double b) { return Math.abs(a - b) < 1e-9; }

    public static void main(String[] args) {
        String fallo = null;
        for (double[] c : CASOS) {
            double r = ORIGINAL.calcular((int) c[0], c[1] == 1, c[2] == 1);
            if (!igual(r, c[3]) && fallo == null) fallo = Arrays.toString(c) + " da " + r;
        }
        System.out.println("Casos correctos con el original: " + (fallo == null ? "sí" : "no (" + fallo +
            ")"));
        List<String> vivos = new ArrayList<>();
        for (var m : MUTANTES.entrySet()) {
            boolean detectado = false;
            for (double[] c : CASOS)
                if (!igual(m.getValue().calcular((int) c[0], c[1] == 1, c[2] == 1), c[3])) detectado = true;
            if (!detectado) vivos.add(m.getKey());
        }
        System.out.println("Mutantes detectados: " + (MUTANTES.size() - vivos.size()) + " de " +
            MUTANTES.size());
        System.out.println("Sobreviven: " + (vivos.isEmpty() ? "ninguno" : String.join(", ", vivos)));
    }
}

```

Cada límite de edad se prueba a los dos lados (11 y 12, 64 y 65, -1 y 0), y cada regla se combina con otra que la haga visible: el fin de semana con un niño (si se ignora, sale 5 en vez de 6) y el descuento de socio con un niño (si se aplica antes del 50 %, sale 4 en vez de 3; si solo se aplica a adultos, sale 5).

El mutante equivalente es M5, el que quita el tope de 0 €. El precio más bajo posible antes del descuento de socio es 5 € (un niño entre semana), y $5 - 2 = 3$ nunca es negativo, así que `Math.max(0, ...)` no cambia nada con ninguna entrada. Ningún caso puede detectarlo: no es un fallo de las pruebas, es código que sobra.

7. Un analizador de código en miniatura

Difícil · Optimización y Documentación · apuntesdam.com/subject/entornos/topic/optimizacion-documentacion

Solución (java)

```
import java.util.*;
import java.util.regex.*;

public class Main {
    static final int MAX_LINEAS_METODO = 8, MAX_PARAMETROS = 3;
    static final Pattern METODO = Pattern.compile(
        "^\\s*(public|private|protected|static|\\s)+[\\w<\\[\\]]+\\s+(\\w+)\\s*\\(((\\^)*\\))\\s*\\{\\}";
    static final Pattern NUMERO = Pattern.compile("\\b\\d+(\\.\\d+)?\\b");

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<String> lineas = new ArrayList<>();
        while (sc.hasNextLine()) lineas.add(sc.nextLine());
        List<String> avisos = analizar(lineas);
        avisos.forEach(System.out::println);
        System.out.println(avisos.size() + (avisos.size() == 1 ? " aviso" : " avisos"));
    }

    // La línea sin cadenas ni comentarios //
    static String codigo(String linea) {
        String s = linea.replaceAll("\\\"(\\\\\\\\.|[^\\"\\\\\\\\])*\\\\\"", "\\\"\"");
        int c = s.indexOf("//");
        return c >= 0 ? s.substring(0, c) : s;
    }

    static List<String> analizar(List<String> lineas) {
        List<String> avisos = new ArrayList<>();
        String metodo = null; // método en el que estamos
        int inicio = 0, llaves = 0;
        for (int i = 0; i < lineas.size(); i++) {
            int n = i + 1;
            String linea = codigo(lineas.get(i));
            Matcher m = METODO.matcher(linea);
            if (m.find()) {
                String nombre = m.group(2), params = m.group(3).trim();
                if (!nombre.matches("[a-z][a-zA-Z0-9]*")) avisos.add("Línea " + n + ": el método «" + nombre
                    + "» no sigue lowerCamelCase");
                int p = params.isEmpty() ? 0 : params.split(",").length;
                if (p > MAX_PARAMETROS) avisos.add("Línea " + n + ": «" + nombre + "» tiene " + p + "
                    parámetros (máximo " + MAX_PARAMETROS + ")");
                if (metodo == null) {
                    metodo = nombre;
                    inicio = n;
                    llaves = 0;
                }
            }
            if (!linea.contains("final")) {
                Matcher num = NUMERO.matcher(linea);
                while (num.find()) {
                    if (!num.group().equals("0") && !num.group().equals("1")) avisos.add("Línea " + n + ":
                        número mágico " + num.group());
                }
            }
            if (metodo != null) {
                for (char c : linea.toCharArray()) {
                    if (c == '{') llaves++;
                    if (c == '}') llaves--;
                }
                if (llaves == 0) { // se cierra el método
                    int largo = n - inicio + 1;
                    if (largo > MAX_LINEAS_METODO) avisos.add("Línea " + inicio + ": «" + metodo + "» mide "
                        + largo + " líneas (máximo " + MAX_LINEAS_METODO + ")");
                    metodo = null;
                }
            }
        }
        return avisos;
    }
}
```

```
}
```

Antes de aplicar las reglas, cada línea se limpia: las cadenas se vacían con una expresión regular y se corta lo que va tras `//`. Así ni el 99 de «Pedido de 99 unidades» cuenta como número mágico ni una llave dentro de un texto descuadra el recuento.

Las reglas 1 y 2 se aplican al reconocer una cabecera de método; la 3 a cada línea que no declare una constante, buscando números con `\b\d+(\.\d+)?\b` (los `\b` evitan coger la cifra de un nombre como `x1`).

Para la regla 4 se recuerda el método actual y se cuentan las llaves desde su cabecera: cuando el contador vuelve a 0, el método ha terminado y se sabe cuántas líneas mide. Es la misma técnica que usan los analizadores reales, aunque ellos trabajan sobre el árbol sintáctico en lugar del texto.

8. Un Git en miniatura: base de fusión

Muy difícil · Optimización y Documentación · apuntesdam.com/subject/entornos/topic/optimizacion-documentacion

Solución (java)

```
import java.util.*;

public class Main {
    static final Map<String, List<String>> padres = new LinkedHashMap<>(); // commit → sus padres
    static final Map<String, String> ramas = new TreeMap<>(); // rama → commit al que apunta

    // Todos los antepasados de un commit, incluido él mismo
    static Set<String> antepasados(String c) {
        Set<String> vistos = new HashSet<>();
        Deque<String> pendientes = new ArrayDeque<>();
        if (c != null) pendientes.push(c);
        while (!pendientes.isEmpty()) {
            String x = pendientes.pop();
            if (!vistos.add(x)) continue; // ya visitado (en un grafo con fusiones se llega por
            // varios caminos)
            for (String p : padres.getOrDefault(x, List.of())) pendientes.push(p);
        }
        return vistos;
    }

    // Antepasado común «más reciente» (el que no es antepasado de otro antepasado común); null si no hay
    static String baseDeFusion(String a, String b) {
        Set<String> comunes = antepasados(a);
        comunes.retainAll(antepasados(b));
        for (String c : padres.keySet()) { // en orden de creación
            if (!comunes.contains(c)) continue;
            boolean esAntepasadoDeOtro = false;
            for (String otro : comunes) {
                if (!otro.equals(c) && antepasados(otro).contains(c)) esAntepasadoDeOtro = true;
            }
            if (!esAntepasadoDeOtro) return c;
        }
        return null;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] t = sc.nextLine().trim().split("\\s+");
            switch (t[0]) {
                case "COMMIT" -> padres.put(t[1], t.length > 2 ? List.of(t[2].split(",")) : List.of());
                case "RAMA" -> ramas.put(t[1], t[2]);
                case "BASE" -> System.out.println("merge-base " + t[1] + " " + t[2] + " → " +
                    baseDeFusion(ramas.get(t[1]), ramas.get(t[2])));
                case "FUSIONAR" -> { // FUSIONAR destino origen: ¿qué haría git merge origen estando
                    // en destino?
                    String d = ramas.get(t[1]), o = ramas.get(t[2]);
                    String base = baseDeFusion(d, o);
                    String r;
                    if (o.equals(base)) r = "ya actualizado (nada que fusionar)";
                    else if (d.equals(base)) r = "fast-forward: " + t[1] + " pasa de " + d + " a " + o;
                    else r = "fusión de tres vías (base " + base + ") : se crea un commit de fusión";
                    System.out.println("git merge " + t[2] + " en " + t[1] + " → " + r);
                }
            }
        }
    }
}
```

antepasados es un recorrido del grafo de commits siguiendo los padres, con una pila y un conjunto de visitados. El conjunto es necesario porque, tras una fusión, un commit tiene dos padres y a un mismo antepasado se puede llegar por dos caminos.

Los antepasados comunes son la intersección de los dos conjuntos (`retainAll`). De ellos, la base de fusión es el más reciente, es decir, el que no es antepasado de ningún otro común. Con eso, el `main` ya distingue los tres casos de `git merge`: si la base es el origen no hay nada nuevo, si es el destino basta con mover la rama, y si no, hay trabajo en los dos

lados.

9. Dobles de prueba escritos a mano

Difícil · Pruebas con JUnit y Mockito · apuntesdam.com/subject/entornos/topic/pruebas-junit

Solución (java)

```
import java.util.*;

interface Inventario {
    int stock(String producto);
    void descontar(String producto, int unidades);
}

interface Correo {
    void enviar(String destinatario, String asunto);
}

// Clase que se quiere probar: no sabe si habla con una base de datos real o con un doble
class ServicioPedidos {
    private final Inventario inventario;
    private final Correo correo;

    ServicioPedidos(Inventario inventario, Correo correo) {
        this.inventario = inventario;
        this.correo = correo;
    }

    boolean pedir(String cliente, String producto, int unidades) {
        if (unidades <= 0) return false;
        if (inventario.stock(producto) < unidades) {
            correo.enviar("almacen@tienda.es", "Sin stock: " + producto);
            return false;
        }
        inventario.descontar(producto, unidades);
        correo.enviar(cliente, "Pedido confirmado: " + unidades + " x " + producto);
        return true;
    }
}

// Stub (devuelve lo que fija la prueba) y a la vez espía (anota las llamadas)
class InventarioFalso implements Inventario {
    private final int stockFijo;
    private final List<String> descuentos = new ArrayList<>();

    InventarioFalso(int stockFijo) { this.stockFijo = stockFijo; }
    public int stock(String producto) { return stockFijo; }
    public void descontar(String producto, int unidades) { descuentos.add(producto + " -" + unidades); }
    List<String> descuentos() { return descuentos; }
}

// Espía: no envía nada, solo recuerda lo que se le pidió
class CorreoEspia implements Correo {
    private final List<String> enviados = new ArrayList<>();

    public void enviar(String destinatario, String asunto) { enviados.add(destinatario + " | " + asunto); }
    List<String> enviados() { return enviados; }
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] t = sc.nextLine().trim().split(" "); // stock cliente producto unidades
            InventarioFalso inv = new InventarioFalso(Integer.parseInt(t[0]));
            CorreoEspia correo = new CorreoEspia();
            boolean ok = new ServicioPedidos(inv, correo).pedir(t[1], t[2], Integer.parseInt(t[3]));
            System.out.println("pedir(" + t[1] + ", " + t[2] + ", " + t[3] + ") con stock " + t[0] + " → " + ok);
            System.out.println(" descuentos: " + inv.descuentos());
            System.out.println(" correos: " + correo.enviados());
        }
    }
}
```

Los dobles implementan las mismas interfaces que los objetos reales, así que `ServicioPedidos` los usa sin enterarse. `InventarioFalso` devuelve siempre el stock que fija la prueba (eso es un stub) y anota cada llamada a `descontar` (eso es un espía); `CorreoEspia` solo anota los envíos.

Con las listas que guardan, la prueba puede comprobar no solo lo que devuelve el método, sino también con quién habló y cómo: que no se descontó nada si no había stock, o que se avisó al almacén. Es lo que Mockito hace por ti con `when(...).thenReturn(...)` y `verify(...)`.

10. Tu propio JUnit con reflexión

Muy difícil · Pruebas con JUnit y Mockito · apuntesdam.com/subject/entornos/topic/pruebas-junit

Solución (java)

```
import java.lang.annotation.*;
import java.lang.reflect.*;
import java.util.*;

// Anotaciones propias, como @Test y @BeforeEach de JUnit
@Retention(RetentionPolicy.RUNTIME) @Target(ElementType.METHOD) @interface Prueba { }
@Retention(RetentionPolicy.RUNTIME) @Target(ElementType.METHOD) @interface Antes { }

class FalloAsercion extends RuntimeException {
    FalloAsercion(String m) { super(m); }
}

class Afirmar {
    static void iguales(Object esperado, Object real) {
        if (!Objects.equals(esperado, real)) throw new FalloAsercion("se esperaba <" + esperado + "> pero fue <" + real + ">");
    }
    static void cierto(boolean c, String mensaje) {
        if (!c) throw new FalloAsercion(mensaje);
    }
}

// ---- Las pruebas de ejemplo (no las modifiques)
class PilaTest {
    private Deque<Integer> pila;

    @Antes void preparar() { pila = new ArrayDeque<>(); pila.push(1); pila.push(2); }

    @Prueba void cimaEsElUltimo() { Afirmar.iguales(2, pila.peek()); }
    @Prueba void sacarReduce() { pila.pop(); Afirmar.iguales(1, pila.size()); }
    @Prueba void cadaPruebaEmpiezaLimpia() { Afirmar.iguales(2, pila.size()); }
    @Prueba void fallaAProposito() { Afirmar.iguales(3, pila.size()); }
    @Prueba void lanzaExcepcion() { new ArrayDeque<Integer>().pop(); }
    void noEsUnaPrueba() { throw new IllegalStateException("no debería ejecutarse"); }
}

public class Main {
    // Ejecuta todos los métodos @Prueba de la clase, en orden alfabético, con una instancia nueva cada vez
    static void ejecutar(Class<?> clase) throws Exception {
        List<Method> pruebas = new ArrayList<>(), antes = new ArrayList<>();
        for (Method m : clase.getDeclaredMethods()) {
            if (m.isAnnotationPresent(Prueba.class)) pruebas.add(m);
            if (m.isAnnotationPresent(Antes.class)) antes.add(m);
        }
        pruebas.sort(Comparator.comparing(Method::getName));

        int correctas = 0, fallidas = 0, errores = 0;
        for (Method prueba : pruebas) {
            Constructor<?> constructor = clase.getDeclaredConstructor();
            constructor.setAccessible(true);
            Object instancia = constructor.newInstance(); // una instancia nueva por prueba
            try {
                for (Method a : antes) {
                    a.setAccessible(true);
                    a.invoke(instancia);
                }
                prueba.setAccessible(true);
                prueba.invoke(instancia);
                System.out.println("✓ " + prueba.getName());
                correctas++;
            } catch (InvocationTargetException e) {
                Throwable causa = e.getCause(); // la excepción que lanzó la prueba
                if (causa instanceof FalloAsercion) {
                    System.out.println("X " + prueba.getName() + ": " + causa.getMessage());
                    fallidas++;
                } else {
                    System.out.println("!! " + prueba.getName() + ": " + causa.getClass().getSimpleName());
                    errores++;
                }
            }
        }
    }
}
```

```

        }
    }
    System.out.println("Pruebas: " + pruebas.size() + ", correctas: " + correctas + ", fallidas: " +
        fallidas + ", con error: " + errores);
}

public static void main(String[] args) throws Exception {
    ejecutar(PilaTest.class);
}
}

```

`getDeclaredMethods()` devuelve los métodos de la clase y `isAnnotationPresent` dice si llevan una anotación. Esto solo funciona porque las anotaciones tienen `@Retention(RUNTIME)`: si no, el compilador no las guardaría en la clase y la reflexión no las vería.

Cada prueba recibe una instancia recién creada, y sobre ella se ejecutan los métodos `@Antes` y la prueba con `invoke`. Por eso `cadaPruebaEmpiezaLimpia` pasa aunque `sacarReduce` haya quitado un elemento: no comparten la pila.

Cuando un método llamado por reflexión lanza una excepción, `invoke` la envuelve en una `InvocationTargetException`; la original está en `getCause()`. Su tipo distingue un fallo (una aserción que no se cumple) de un error (cualquier otra excepción), igual que JUnit.

11. Del diagrama de estados al código

Difícil · Diagramas UML · apuntesdam.com/subject/entornos/topic/diagramas-uml

Solución (java)

```
import java.util.*;

public class Main {
    // Diagrama de estados de un pedido (estado inicial: CARRITO)
    // CARRITO --pagar--> PAGADO
    // CARRITO --cancelar--> CANCELADO
    // PAGADO --enviar--> ENVIADO
    // PAGADO --cancelar--> CANCELADO [acción: devolver el dinero]
    // ENVIADO --entregar--> ENTREGADO
    // ENVIADO --perder--> PAGADO [acción: reenviar; como mucho 2 veces, después CANCELADO con devolución]
    // ENTREGADO y CANCELADO son estados finales
    enum Estado { CARRITO, PAGADO, ENVIADO, ENTREGADO, CANCELADO }

    static Estado estado = Estado.CARRITO;
    static int perdidas = 0;

    // Aplica un evento y devuelve el texto que describe lo ocurrido
    static String aplicar(String evento) {
        Estado siguiente = null;
        String accion = null;
        switch (estado) {
            case CARRITO -> {
                if (evento.equals("pagar")) siguiente = Estado.PAGADO;
                else if (evento.equals("cancelar")) siguiente = Estado.CANCELADO;
            }
            case PAGADO -> {
                if (evento.equals("enviar")) siguiente = Estado.ENVIADO;
                else if (evento.equals("cancelar")) { siguiente = Estado.CANCELADO; accion = "devolver el dinero"; }
            }
            case ENVIADO -> {
                if (evento.equals("entregar")) siguiente = Estado.ENTREGADO;
                else if (evento.equals("perder")) {
                    perdidas++;
                    if (perdidas < 3) { siguiente = Estado.PAGADO; accion = "reenviar"; }
                    else { siguiente = Estado.CANCELADO; accion = "devolver el dinero"; }
                }
            }
            default -> { } // ENTREGADO y CANCELADO: estados finales, sin transiciones
        }
        if (siguiente == null) return "transición no permitida (sigue en " + estado + ")";
        estado = siguiente;
        return estado + (accion != null ? " [" + accion + "]" : "");
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNext()) {
            String ev = sc.next();
            Estado antes = estado;
            String r = aplicar(ev);
            System.out.println(antes + " --" + ev + "--> " + r);
        }
        System.out.println("Estado final: " + estado);
    }
}
```

Cada flecha del diagrama se convierte en una rama: el `switch` elige el estado actual y, dentro, cada `if` es un evento que sale de él. Si ninguna rama asigna `siguiente`, el evento no tiene transición desde ese estado y no cambia nada; los estados finales simplemente no tienen ramas.

La condición de la pérdida (como mucho dos reenvíos) es lo que en UML se llama una guarda: la misma transición lleva a un estado u otro según el contador `perdidas`, que forma parte del estado del objeto aunque no aparezca como un estado del diagrama.

12. Ingeniería inversa: de las clases al diagrama

Muy difícil · Diagramas UML · apuntesdam.com/subject/entornos/topic/diagramas-uml

Solución (java)

```
import java.lang.reflect.*;
import java.util.*;

// ---- Modelo de ejemplo (no lo modifiques)
abstract class Persona {
    protected String nombre;
    private String dni;
    public String getNombre() { return nombre; }
}
interface Evaluable { double nota(); }
class Alumno extends Persona implements Evaluable {
    private int curso;
    private List<Matricula> matriculas;
    public double nota() { return 0; }
    public void matricular(Asignatura a) { }
}
class Profesor extends Persona {
    private double salario;
    private List<Asignatura> imparte;
}
class Asignatura {
    private String codigo;
    int horas;
    private Profesor titular;
}
class Matricula {
    private Asignatura asignatura;
    private double nota;
}

public class Main {
    static final List<Class<?>> MODELO = List.of(Persona.class, Evaluable.class, Alumno.class,
        Profesor.class, Asignatura.class, Matricula.class);

    static String visibilidad(int mod) {
        return Modifier.isPublic(mod) ? "+" : Modifier.isPrivate(mod) ? "-" : Modifier.isProtected(mod) ?
            "#" : "~";
    }

    // Genera el diagrama de clases en PlantUML a partir de las clases compiladas
    static String plantUml(List<Class<?>> clases) {
        StringBuilder sb = new StringBuilder("@startuml\n");
        List<String> relaciones = new ArrayList<>();
        for (Class<?> c : clases) {
            String nombre = c.getSimpleName();
            String tipo = c.isInterface() ? "interface" : Modifier.isAbstract(c.getModifiers()) ? "abstract
                class" : "class";
            sb.append(tipo).append(' ').append(nombre).append(" {\n");

            Field[] campos = c.getDeclaredFields();
            Arrays.sort(campos, Comparator.comparing(Field::getName));
            for (Field f : campos) {
                if (f.isSynthetic()) continue;
                Class<?> t = f.getType();
                if (clases.contains(t)) { // referencia a otra clase del
                    modelo
                    relaciones.add(nombre + " --> \"1\" " + t.getSimpleName() + " : " + f.getName());
                } else if (List.class.isAssignableFrom(t) && f.getGenericType() instanceof ParameterizedType
                    pt
                    && pt.getActualTypeArguments()[0] instanceof Class<?> e && clases.contains(e)) {
                    relaciones.add(nombre + " --> \"*\n\" " + e.getSimpleName() + " : " + f.getName());
                } else {
                    sb.append(" ").append(visibilidad(f.getModifiers())).append(f.getName())
                        .append(" : ").append(t.getSimpleName()).append('\n');
                }
            }

            Method[] metodos = c.getDeclaredMethods();
```

```

        Arrays.sort(metodos, Comparator.comparing(Method::getName));
        for (Method m : metodos) {
            if (m.isSynthetic()) continue;
            List<String> params = new ArrayList<>();
            for (Class<> p : m.getParameterTypes()) params.add(p.getSimpleName());
            sb.append(" ").append(visibilidad(m.getModifiers())).append(m.getName())
                .append('(').append(String.join(", ", params)).append(") : ")
                .append(m.getReturnType().getSimpleName()).append('\n');
        }
        sb.append("}\n");

        if (c.getSuperclass() != null && clases.contains(c.getSuperclass())) {
            relaciones.add(c.getSuperclass().getSimpleName() + " <|-- " + nombre);
        }
        for (Class<?> i : c.getInterfaces()) {
            if (clases.contains(i)) relaciones.add(i.getSimpleName() + (c.isInterface() ? " <|.. " : "
                <|.. ") + nombre);
        }
    }
    relaciones.forEach(r -> sb.append(r).append('\n'));
    return sb.append("@enduml").toString();
}

public static void main(String[] args) {
    System.out.println(plantUml(MODELO));
}
}

```

La reflexión da toda la información del diagrama: `getDeclaredFields` y `getDeclaredMethods` (solo los de la propia clase, no los heredados), los modificadores de cada uno para la visibilidad, `getSuperclass` para la herencia y `getInterfaces` para la realización. Como el orden en que la JVM devuelve campos y métodos no está garantizado, se ordenan por nombre.

Lo interesante es decidir qué es un atributo y qué una asociación: si el tipo del campo es otra clase del modelo, es una relación con multiplicidad 1; si es una `List`, hay que mirar su tipo genérico con `getGenericType()` (el tipo «borrado» `List` no dice de qué es la lista) y, si es del modelo, es una asociación con multiplicidad *.

Las relaciones se guardan en una lista mientras se recorren las clases y se escriben al final, que es donde PlantUML las espera tras las definiciones.

13. Depura el programa de notas

Fácil · Ejercicios de Entornos de Desarrollo · apuntesdam.com/subject/entornos/topic/ejercicios-entornos

Solución (java)

```
import java.util.*;

public class Main {

    // Devuelve la media de las notas (con decimales)
    static double media(int[] notas) {
        int suma = 0;
        for (int i = 0; i < notas.length; i++) {    // error 1: empezaba en 1 y se saltaba la primera nota
            suma += notas[i];
        }
        return (double) suma / notas.length;        // error 2: la división entera perdía los decimales
    }

    // Devuelve la nota más alta
    static int maxima(int[] notas) {
        int max = notas[0];
        for (int n : notas) {
            if (n > max) max = n;                    // error 3: con < se quedaba con la mínima
        }
        return max;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] notas = new int[n];
        for (int i = 0; i < n; i++) notas[i] = sc.nextInt();
        System.out.printf(Locale.US, "Media: %.2f\n", media(notas));
        System.out.println("Máxima: " + maxima(notas));
    }
}
```

Los tres errores son clásicos: el bucle de la media empieza en 1 y se salta la primera nota; `suma / notas.length` es una división entre enteros y trunca el resultado; y en `maxima` la comparación está al revés, así que guarda la nota más baja.

Con el depurador se ven en seguida: al parar en el bucle, `i` empieza valiendo 1; al evaluar `suma / notas.length` en la vista de expresiones sale un número sin decimales; y al seguir `max` vuelta a vuelta, se ve que baja en lugar de subir.

14. Valores límite: clasificar edades

Fácil · Ejercicios de Entornos de Desarrollo · apuntesdam.com/subject/entornos/topic/ejercicios-entornos

Solución (java)

```
import java.util.*;

public class Main {

    static String clasificar(int edad) {
        if (edad < 0 || edad > 120) return "Error";
        if (edad <= 11) return "Niño";
        if (edad <= 17) return "Adolescente";
        if (edad <= 64) return "Adulto";
        return "Jubilado";
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextInt()) {
            System.out.println(clasificar(sc.nextInt()));
        }
    }
}
```


Primero se descartan los valores fuera de rango y después se comprueban los tramos de menor a mayor. Como cada `return` termina el método, cada condición solo necesita el límite superior de su tramo: si se llega a `edad <= 17`, ya se sabe que la edad es al menos 12.

Fíjate en que los límites se escriben tal como dice la especificación (`<= 11`, `<= 17`, `<= 64`): confundir un `<` con un `<=` es justo el error que detectan las pruebas de valores límite.

15. Caja negra: el problema del triángulo

Medio · Ejercicios de Entornos de Desarrollo · apuntesdam.com/subject/entornos/topic/ejercicios-entornos

Solución (java)

```
import java.util.*;

public class Main {

    static String tipo(int a, int b, int c) {
        if (a <= 0 || b <= 0 || c <= 0) return "No es triángulo";
        if (a >= b + c || b >= a + c || c >= a + b) return "No es triángulo";    // desigualdad triangular
        if (a == b && b == c) return "Equilátero";
        if (a == b || b == c || a == c) return "Isósceles";
        return "Escaleno";
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextInt()) {
            System.out.println(tipo(sc.nextInt(), sc.nextInt(), sc.nextInt()));
        }
    }
}
```

Primero se comprueba que los lados formen un triángulo: todos positivos y ninguno mayor o igual que la suma de los otros dos (con 1, 2 y 3 los lados quedarían planos). Hay que comprobar los tres lados, no solo el primero: por eso el caso «10 2 3» aparece en las pruebas con el lado largo en otra posición.

Después, el orden de las comprobaciones importa: equilátero antes que isósceles, porque un triángulo con los tres lados iguales también tiene dos iguales.

16. Refactoriza: extraer método

Medio · Ejercicios de Entornos de Desarrollo · apuntesdam.com/subject/entornos/topic/ejercicios-entornos

Solución (java)

```
import java.util.*;

public class Main {

    static final double IVA = 0.21;

    static double descuentoCliente(String tipo) {
        return switch (tipo) {
            case "VIP" -> 0.15;
            case "SOCIO" -> 0.05;
            default -> 0;
        };
    }

    static double descuentoVolumen(int unidades) {
        return unidades >= 10 ? 0.10 : 0;
    }

    static double aplicarDescuento(double importe, double descuento) {
        return importe * (1 - descuento);
    }

    static double precioFinal(double base, int unidades, String tipo) {
        double importe = base * unidades;
        importe = aplicarDescuento(importe, descuentoCliente(tipo));
        importe = aplicarDescuento(importe, descuentoVolumen(unidades));
        return importe * (1 + IVA);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in).useLocale(Locale.US);
        while (sc.hasNext()) {
            double base = sc.nextDouble();
            int unidades = sc.nextInt();
            String tipo = sc.next();
            System.out.printf(Locale.US, "%.2f%n", precioFinal(base, unidades, tipo));
        }
    }
}
```

Cada método extraído tiene un nombre que explica lo que hace y una sola responsabilidad, y los números mágicos han pasado a estar en un único sitio. `precioFinal` se lee ahora como la descripción del cálculo: importe, descuento del cliente, descuento por volumen e IVA.

Una refactorización no debe cambiar el comportamiento: los descuentos se aplican en el mismo orden y sobre el importe ya descontado, igual que en el código original. Por eso los resultados de las pruebas son idénticos.

17. Del diagrama de clases al código

Medio · Ejercicios de Entornos de Desarrollo · apuntesdam.com/subject/entornos/topic/ejercicios-entornos

Solución (java)

```
import java.util.*;

abstract class Figura {
    protected String nombre;

    public Figura(String nombre) { this.nombre = nombre; }

    public abstract double area();

    @Override
    public String toString() {
        return String.format(Locale.US, "%s: %.2f", nombre, area());
    }
}

class Circulo extends Figura {
    private double radio;

    public Circulo(double radio) {
        super("Círculo");
        this.radio = radio;
    }

    @Override
    public double area() {
        return Math.PI * radio * radio;
    }
}

class Rectangulo extends Figura {
    private double base;
    private double altura;

    public Rectangulo(double base, double altura) {
        super("Rectángulo");
        this.base = base;
        this.altura = altura;
    }

    @Override
    public double area() {
        return base * altura;
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in).useLocale(Locale.US);
        List<Figura> figuras = new ArrayList<>();
        while (sc.hasNext()) {
            String tipo = sc.next();
            if (tipo.equals("C")) figuras.add(new Circulo(sc.nextDouble()));
            else figuras.add(new Rectangulo(sc.nextDouble(), sc.nextDouble()));
        }
        double total = 0;
        for (Figura f : figuras) {
            System.out.println(f);
            total += f.area();
        }
        System.out.printf(Locale.US, "Total: %.2f\n", total);
    }
}
```

Cada elemento del diagrama tiene su traducción directa: la flecha de herencia es `extends`, los atributos con «-» son campos `private` y el método abstracto de la clase padre obliga a cada subclase a dar su versión de `area()`.

El constructor de cada subclase llama primero a `super(...)` para inicializar la parte heredada (el nombre) y después guarda sus propios atributos. Gracias al polimorfismo, el `toString` de `Figura` llama a la `area()` de cada subclase sin saber cuál es.

18. Del pseudocódigo a Java

Fácil · Fundamentos de la Programación · apuntesdam.com/subject/programacion/topic/introduccion-programacion

Solución (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        if (n == 0) {
            System.out.println("cero");
        } else {
            String signo = n > 0 ? "positivo" : "negativo";
            String paridad = n % 2 == 0 ? "par" : "impar";
            System.out.println(signo + " " + paridad);
        }
    }
}
```

Cada «Si ... entonces ... si no» del pseudocódigo se convierte en un `if/else`, y los dos que solo eligen un valor se escriben con el operador ternario, que es más corto.

Ojo con los negativos: en Java `-15 % 2` vale `-1`, no `1`. Por eso la condición de par es `n % 2 == 0`; comprobar `n % 2 == 1` daría «par» para los impares negativos.

19. Traza del algoritmo de Euclides

Medio · Fundamentos de la Programación · apuntesdam.com/subject/programacion/topic/introduccion-programacion

Solución (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int a = sc.nextInt(), b = sc.nextInt();
        int pasos = 0;
        while (b != 0) {
            pasos++;
            int cociente = a / b, resto = a % b;
            System.out.println("Paso " + pasos + ": " + a + " = " + cociente + " · " + b + " + " + resto);
            a = b;
            b = resto;
        }
        System.out.println("mcd = " + a + " en " + pasos + (pasos == 1 ? " paso" : " pasos"));
    }
}
```

El bucle sigue el algoritmo al pie de la letra: mientras `b` no sea 0, se calcula el cociente y el resto, se muestra la línea de la traza y los valores se desplazan (`a` pasa a ser `b` y `b` pasa a ser el resto).

Cuando `b` llega a 0, el máximo común divisor es lo que queda en `a`. El contador de pasos sirve para la traza y para elegir entre «paso» y «pasos».

20. Conversor de bases sin atajos

Difícil · Fundamentos de la Programación · apuntesdam.com/subject/programacion/topic/introduccion-programacion

Solución (java)

```
import java.util.Scanner;

public class Main {
    static String aBase(int n, int base) {
        if (n == 0) return "0";
        String cifras = "0123456789ABCDEF";
        StringBuilder sb = new StringBuilder();
        while (n > 0) {
            sb.append(cifras.charAt(n % base)); // el resto es la cifra
            n /= base;
        }
        return sb.reverse().toString(); // los restos se leen de abajo arriba
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextInt()) {
            int n = sc.nextInt();
            System.out.println(n + " → binario " + aBase(n, 2) + " · octal " + aBase(n, 8) + " · hexadecimal " + aBase(n, 16));
        }
    }
}
```

Es el método de las divisiones sucesivas: cada resto de dividir entre la base es una cifra, empezando por la de menor valor. Por eso se van añadiendo a un `StringBuilder` y al final se le da la vuelta con `reverse()`.

La cadena `"0123456789ABCDEF"` traduce un resto a su cifra: `charAt(11)` es `B`. El 0 es un caso especial, porque el bucle no se ejecutaría ninguna vez.

21. Simula una máquina con su propio ensamblador

Muy difícil · Fundamentos de la Programación · apuntesdam.com/subject/programacion/topic/introduccion-programacion

Solución (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<String[]> programa = new ArrayList<>();
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (!l.isEmpty() && !l.startsWith(";")) programa.add(l.split("\\s+"));
        }
        int[] memoria = new int[16];
        int acumulador = 0, pc = 0, ejecutadas = 0;
        while (true) {
            if (pc < 0 || pc >= programa.size()) {
                System.out.println("ERROR: el programa termina sin HALT");
                return;
            }
            if (ejecutadas == 1000) {
                System.out.println("ERROR: demasiadas instrucciones (¿bucle infinito?)");
                return;
            }
            // Captación: la instrucción a la que apunta el PC; el PC pasa a la siguiente
            String[] ins = programa.get(pc);
            int linea = pc;
            pc++;
            ejecutadas++;
            int arg = ins.length > 1 && !ins[1].startsWith(";") ? Integer.parseInt(ins[1]) : 0;
            // Ejecución
            switch (ins[0]) {
                case "LOADI" -> acumulador = arg;
                case "LOAD" -> acumulador = memoria[arg];
                case "STORE" -> memoria[arg] = acumulador;
                case "ADD" -> acumulador += memoria[arg];
                case "SUB" -> acumulador -= memoria[arg];
                case "JMP" -> pc = arg;
                case "JZ" -> { if (acumulador == 0) pc = arg; }
                case "JNEG" -> { if (acumulador < 0) pc = arg; }
                case "PRINT" -> System.out.println(acumulador);
                case "HALT" -> {
                    System.out.println("HALT tras " + ejecutadas + " instrucciones · acumulador = " +
                        acumulador);
                    return;
                }
                default -> {
                    System.out.println("ERROR: instrucción desconocida " + ins[0] + " en la línea " +
                        linea);
                    return;
                }
            }
        }
    }
}
```

El bucle `while (true)` es el ciclo de la CPU: comprobar que el PC está dentro del programa, captar la instrucción, avanzar el PC y ejecutar. Los saltos funcionan simplemente sobrescribiendo el PC que ya había avanzado.

Un `switch` con flechas ejecuta cada instrucción en una línea. El límite de 1000 instrucciones protege de los bucles infinitos, el mismo problema que tienen los depuradores y los sistemas operativos con los programas que no terminan.

22. Desglose de segundos

Fácil · Introducción a Java · apuntesdam.com/subject/programacion/topic/introduccion-java

Solución (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        long total = sc.nextLong();
        long dias = total / 86400;
        long horas = total % 86400 / 3600;
        long minutos = total % 3600 / 60;
        long segundos = total % 60;
        System.out.println(total + " s = " + dias + " d " + horas + " h " + minutos + " min " + segundos + " s");
        System.out.printf("Formato reloj: %02d:%02d:%02d (%d %s)%n", horas, minutos, segundos, dias, dias == 1 ? "día" : "días");
    }
}
```

Un día tiene 86 400 segundos y una hora 3 600. La división entera da cuántas unidades completas caben y el resto (%) lo que sobra: `total % 86400 / 3600` son las horas que quedan después de quitar los días enteros.

En `printf`, `%02d` rellena con un cero a la izquierda hasta dos cifras, que es justo el formato de un reloj.

23. Las trampas de los tipos numéricos

Medio · Introducción a Java · apuntesdam.com/subject/programacion/topic/introduccion-java

Solución (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int a = sc.nextInt(), b = sc.nextInt();
        System.out.println("int: " + a + " * " + b + " = " + (a * b));
        System.out.println("long: " + a + " * " + b + " = " + ((long) a * b));
        System.out.println("división entera: " + a + " / " + b + " = " + (a / b));
        double real = (double) a / b;
        System.out.println("división real: " + a + " / " + b + " = " + real);
        System.out.println("casting: (int) " + real + " = " + (int) real);
        int k = Math.abs(a) % 26;
        char letra = (char) ('A' + k);
        System.out.println("char: 'A' + " + k + " = " + letra + " (código " + (int) letra + ")");
    }
}
```

`a * b` con dos `int` se calcula en 32 bits y, si no cabe, da la vuelta (desbordamiento) sin avisar. Convirtiendo uno de los dos a `long` antes de multiplicar, la operación se hace en 64 bits.

Con la división pasa lo mismo: entre enteros es entera y trunca hacia cero ($-45 / 7 = -6$). Hay que convertir antes de dividir: `(double) a / b`. Y un `char` es un número: `'A' + k` da un `int` que hay que volver a convertir a `char`.

24. La cuota de un préstamo

Difícil · Introducción a Java · apuntesdam.com/subject/programacion/topic/introduccion-java

Solución (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in).useLocale(Locale.US);
        double capital = sc.nextDouble();
        double interesAnual = sc.nextDouble();
        int anios = sc.nextInt();
        int n = anios * 12;
        double i = interesAnual / 100 / 12;
        double cuota = i == 0 ? capital / n : capital * i / (1 - Math.pow(1 + i, -n));
        System.out.printf(Locale.US, "Cuota mensual: %.2f EUR (%d cuotas)%n", cuota, n);
        System.out.printf(Locale.US, "Total pagado: %.2f EUR · intereses: %.2f EUR%n", cuota * n, cuota * n - capital);
        double pendiente = capital;
        for (int k = 1; k <= Math.min(3, n); k++) {
            double intereses = pendiente * i;
            double amortiza = cuota - intereses;
            pendiente -= amortiza;
            System.out.printf(Locale.US, "Cuota %d: intereses %.2f · amortiza %.2f · pendiente %.2f%n", k, intereses, amortiza, pendiente);
        }
    }
}
```

La fórmula del sistema francés se escribe directamente con `Math.pow(1 + i, -n)`. El caso del interés 0 se trata aparte, porque la fórmula dividiría 0 entre 0.

En cada cuota, los intereses se calculan sobre lo que queda por pagar; como el pendiente baja, cada mes se pagan menos intereses y se amortiza más, aunque la cuota sea la misma. `Locale.US` en `printf` garantiza el punto decimal.

25. Permisos de Unix con operadores de bits

Muy difícil · Introducción a Java · apuntesdam.com/subject/programacion/topic/introduccion-java

Solución (java)

```
import java.util.Scanner;

public class Main {
    static String simbolico(int permisos) {
        String letras = "rwx";
        StringBuilder sb = new StringBuilder();
        for (int bit = 8; bit >= 0; bit--) {
            sb.append((permisos & (1 << bit)) != 0 ? letras.charAt((8 - bit) % 3) : '-');
        }
        return sb.toString();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int permisos = Integer.parseInt(sc.next(), 8);
        System.out.println("Inicial: " + String.format("%03o", permisos) + " " + simbolico(permisos));
        while (sc.hasNext()) {
            String orden = sc.next();
            String quien = sc.next();
            int bits = 0;
            for (char c : orden.substring(1).toCharArray()) bits |= c == 'r' ? 4 : c == 'w' ? 2 : 1;
            for (char g : (quien.equals("a") ? "ugo" : quien).toCharArray()) {
                int desp = g == 'u' ? 6 : g == 'g' ? 3 : 0; // posición del grupo
                int mascara = bits << desp;
                switch (orden.charAt(0)) {
                    case '+' -> permisos |= mascara;
                    case '-' -> permisos &= ~mascara;
                    default -> permisos = (permisos & ~(7 << desp)) | mascara; // = : se borra el grupo y se ponen los bits
                }
            }
            System.out.println(orden + " " + quien + " → " + String.format("%03o", permisos) + " " + simbolico(permisos));
        }
    }
}
```

Cada grupo de tres bits se mueve a su sitio con un desplazamiento: los del usuario empiezan en el bit 6, los del grupo en el 3 y los de los demás en el 0. `bits << desp` construye la máscara de la orden para ese grupo.

| enciende los bits de la máscara, `& ~mascara` los apaga y `=` primero borra el grupo entero (`& ~(7 << desp)`) y luego enciende los pedidos. `simbolico` recorre los 9 bits de mayor a menor y elige r, w o x según la posición.

26. Calificación a partir de la nota

Fácil · Estructuras Condicionales · apuntesdam.com/subject/programacion/topic/condicionales

Solución (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int nota = sc.nextInt();
        if (nota < 0 || nota > 10) {
            System.out.println("Nota no valida");
        } else if (nota <= 4) {
            System.out.println("Suspendido");
        } else if (nota == 5) {
            System.out.println("Aprobado");
        } else if (nota == 6) {
            System.out.println("Bien");
        } else if (nota <= 8) {
            System.out.println("Notable");
        } else {
            System.out.println("Sobresaliente");
        }
    }
}
```

Primero se descarta lo que no es válido y después se encadenan los tramos con `else if`, de menor a mayor. Como cada condición solo se evalúa si las anteriores han fallado, basta con comprobar el límite superior de cada tramo.

27. Encuentra el fallo: años bisiestos

Fácil · Estructuras Condicionales · apuntesdam.com/subject/programacion/topic/condicionales

Solución (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextInt()) {
            int a = sc.nextInt();
            boolean bisiesto = (a % 4 == 0 && a % 100 != 0) || a % 400 == 0;
            System.out.println(a + (bisiesto ? ": bisiesto" : ": no bisiesto"));
        }
    }
}
```

La condición original unía las tres partes con `||`, así que bastaba con que el año no fuera divisible entre 100 (casi todos) para darlo por bisiesto. La regla correcta es (divisible entre 4 y no entre 100) o divisible entre 400.

28. Encuentra el fallo: la nota en palabras

Fácil · Estructuras Condicionales · apuntesdam.com/subject/programacion/topic/condicionales

Solución (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        int nota = new Scanner(System.in).nextInt();
        switch (nota) {
            case 5:
                System.out.println("Suficiente");
                break;
            case 6:
                System.out.println("Bien");
                break;
            case 7:
            case 8:
                System.out.println("Notable");
                break;
            case 9:
            case 10:
                System.out.println("Sobresaliente");
                break;
            default:
                System.out.println("Insuficiente");
        }
    }
}
```

Faltaba el `break` del `case 6`: en un `switch` clásico, la ejecución sigue en el caso siguiente hasta encontrar un `break` (fall-through), así que un 6 escribía «Bien» y también «Notable».

Los `case 7` y `case 8` sin `break` entre ellos sí son intencionados: es la forma de agrupar varios valores. Con el `switch` de flechas (`case 7, 8 ->`) este error no puede ocurrir.

29. Año bisiesto

Medio · Estructuras Condicionales · apuntesdam.com/subject/programacion/topic/condicionales

Solución (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int anio = sc.nextInt();

        boolean bisiesto = (anio % 4 == 0 && anio % 100 != 0) || anio % 400 == 0;

        if (bisiesto) {
            System.out.println(anio + " es bisiesto");
        } else {
            System.out.println(anio + " no es bisiesto");
        }
    }
}
```

La regla se traduce tal cual: divisible entre 4 y no entre 100, o divisible entre 400. Los paréntesis dejan clara la agrupación; `&&` tiene más prioridad que `||`, pero escribirlos evita dudas al leerlo.

30. Validar una fecha y calcular el día siguiente

Difícil · Estructuras Condicionales · apuntesdam.com/subject/programacion/topic/condicionales

Solución (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextInt()) {
            int d = sc.nextInt(), m = sc.nextInt(), a = sc.nextInt();
            boolean bisiestro = (a % 4 == 0 && a % 100 != 0) || a % 400 == 0;
            String fecha = String.format("%02d/%02d/%04d", d, m, a);
            if (m < 1 || m > 12) {
                System.out.println(fecha + ": no válida (el mes debe estar entre 1 y 12)");
                continue;
            }
            int[] dias = {31, bisiestro ? 29 : 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
            if (d < 1 || d > dias[m - 1]) {
                System.out.println(fecha + ": no válida (ese mes tiene " + dias[m - 1] + " días)");
                continue;
            }
            int sd = d + 1, sm = m, sa = a;
            if (sd > dias[m - 1]) { // fin de mes
                sd = 1;
                sm++;
            }
            if (sm > 12) { // fin de año
                sm = 1;
                sa++;
            }
            System.out.println(fecha + " → siguiente: " + String.format("%02d/%02d/%04d", sd, sm, sa) +
                (bisiestro ? " (año bisiestro)" : ""));
        }
    }
}
```

Un array con los días de cada mes (con febrero según el año) evita una cadena de condiciones: `dias[m - 1]` da los días del mes. La validación va antes que el cálculo, y `continue` pasa a la siguiente fecha.

El día siguiente es sumar 1 y desbordar como un cuentakilómetros: si se pasa de los días del mes, día 1 del mes siguiente; si se pasa de diciembre, enero del año siguiente.

31. IRPF por tramos (simplificado)

Muy difícil · Estructuras Condicionales · apuntesdam.com/subject/programacion/topic/condicionales

Solución (java)

```
import java.util.*;

public class Main {
    static double escala(double base) {
        double[] limites = {12450, 20200, 35200, 60000, 300000, Double.MAX_VALUE};
        double[] tipos = {0.19, 0.24, 0.30, 0.37, 0.45, 0.47};
        double impuesto = 0, anterior = 0;
        for (int i = 0; i < limites.length && base > anterior; i++) {
            impuesto += (Math.min(base, limites[i]) - anterior) * tipos[i]; // la parte de la base que cae
            // en este tramo
            anterior = limites[i];
        }
        return impuesto;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in).useLocale(Locale.US);
        double bruto = sc.nextDouble();
        int hijos = sc.nextInt();
        int edad = sc.nextInt();
        double ss = bruto * 0.0635;
        double base = Math.max(0, bruto - ss - 2000);
        double minimo = 5550 + (edad > 65 ? 1150 : 0);
        double[] porHijo = {2400, 2700, 4000};
        for (int h = 0; h < hijos; h++) minimo += h < 3 ? porHijo[h] : 4500;
        double cuota = minimo < base ? escala(base) - escala(minimo) : 0;
        double neto = bruto - ss - cuota;
        System.out.printf(Locale.US, "Base liquidable: %.2f\n", base);
        System.out.printf(Locale.US, "Mínimo personal y familiar: %.2f\n", minimo);
        System.out.printf(Locale.US, "Cuota de IRPF: %.2f\n", cuota);
        System.out.printf(Locale.US, "Tipo efectivo: %.2f %%\n", cuota / bruto * 100);
        System.out.printf(Locale.US, "Neto anual: %.2f · al mes (14 pagas): %.2f\n", neto, neto / 14);
    }
}
```

`escala` recorre los tramos y suma, para cada uno, la parte de la base que cae dentro multiplicada por su tipo. Así nadie paga el 30 % por todo su sueldo, solo por la parte que supera los 20 200 €: eso es un impuesto progresivo.

El mínimo personal y familiar no se resta de la base: se calcula la escala de la base y la de ese mínimo, y la diferencia es la cuota. En `printf`, `%%` escribe el símbolo del porcentaje.

32. Tabla de multiplicar

Fácil · Bucles · apuntesdam.com/subject/programacion/topic/bucles

Solución (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        for (int i = 1; i <= 10; i++) {
            System.out.println(n + " x " + i + " = " + (n * i));
        }
    }
}
```

Un `for` de 1 a 10 (con `<=`, para incluir el 10) escribe una línea en cada vuelta. Los paréntesis de `(n * i)` son necesarios: sin ellos, el `+` uniría primero el texto con `n` y después con `i`, y saldría «7 x 1 = 71».

33. Encuentra el fallo: la suma de 1 a n

Fácil · Bucles · apuntesdam.com/subject/programacion/topic/bucles

Solución (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        int n = new Scanner(System.in).nextInt();
        int suma = 0;
        for (int i = 1; i <= n; i++) {
            suma += i;
        }
        System.out.println("Suma: " + suma);
    }
}
```

Es un error «por uno» (off-by-one): con `i < n` el bucle se detiene antes de sumar el propio `n`. Para incluir el último valor la condición tiene que ser `i <= n`.

34. Suma de los dígitos

Medio · Bucles · apuntesdam.com/subject/programacion/topic/bucles

Solución (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int suma = 0;

        while (n > 0) {
            suma += n % 10;    // la última cifra
            n /= 10;          // se quita la última cifra
        }

        System.out.println("Suma: " + suma);
    }
}
```

`n % 10` es la última cifra de un número y `n / 10` (división entera) es el número sin esa cifra. Repitiendo las dos operaciones hasta que `n` vale 0 se recorren todas las cifras de derecha a izquierda, ceros incluidos.

35. Encuentra el fallo: la media que pierde decimales

Medio · Bucles · apuntesdam.com/subject/programacion/topic/bucles

Solución (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int suma = 0, contador = 0;
        int n = sc.nextInt();
        while (n != 0) {
            suma += n;
            contador++;
            n = sc.nextInt();
        }
        double media = (double) suma / contador;
        System.out.printf(java.util.Locale.US, "Media: %.2f\n", media);
    }
}
```

`suma / contador` es una división entre dos `int`, así que es entera ($23 / 3 = 7$) y los decimales se pierden antes de guardarse en el `double`. Convirtiendo uno de los operandos (`(double) suma`), la división se hace con decimales.

Ojo: `(double) (suma / contador)` no sirve, porque convierte el resultado cuando ya se ha truncado.

36. La conjetura de Collatz

Difícil · Bucles · apuntesdam.com/subject/programacion/topic/bucles

Solución (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int a = sc.nextInt(), b = sc.nextInt();

        StringBuilder secuencia = new StringBuilder();
        long x = a;
        secuencia.append(x);
        while (x != 1) {
            x = x % 2 == 0 ? x / 2 : 3 * x + 1;
            secuencia.append(" → ").append(x);
        }
        System.out.println(secuencia);

        int mejor = a, maxPasos = -1;
        long maximo = 0;
        for (int n = a; n <= b; n++) {
            long v = n;
            int pasos = 0;
            maximo = Math.max(maximo, v);
            while (v != 1) {
                v = v % 2 == 0 ? v / 2 : 3 * v + 1;
                pasos++;
                maximo = Math.max(maximo, v);
            }
            if (pasos > maxPasos) { // con > (y no >=) gana el menor en caso de empate
                maxPasos = pasos;
                mejor = n;
            }
        }
        System.out.println("Entre " + a + " y " + b + ": " + mejor + " necesita " + maxPasos + " pasos; el valor más alto alcanzado es " + maximo);
    }
}
```

La regla se aplica con un ternario dentro de un `while` que termina al llegar a 1. Los valores se guardan en `long` porque, aunque se empiece por un número pequeño, la secuencia sube mucho (el 27 llega a 9232, y con números mayores se desbordaría un `int`).

Para el intervalo se repite el mismo bucle con cada número, contando los pasos y guardando el mayor valor visto.

Comparar con `>` en lugar de `>=` hace que, en un empate, se quede el primero, que es el menor.

37. El calendario de un mes

Muy difícil · Bucles · apuntesdam.com/subject/programacion/topic/bucles

Solución (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int anio = sc.nextInt(), mes = sc.nextInt();
        String[] nombres = {"Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio", "Julio", "Agosto",
                            "Septiembre", "Octubre", "Noviembre", "Diciembre"};
        String[] semana = {"lunes", "martes", "miércoles", "jueves", "viernes", "sábado", "domingo"};

        boolean bisiesto = (anio % 4 == 0 && anio % 100 != 0) || anio % 400 == 0;
        int[] diasMes = {31, bisiesto ? 29 : 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
        int dias = diasMes[mes - 1];

        // Zeller: enero y febrero cuentan como los meses 13 y 14 del año anterior
        int m = mes, y = anio;
        if (m < 3) {
            m += 12;
            y--;
        }
        int k = y % 100, j = y / 100;
        int h = (1 + 13 * (m + 1) / 5 + k + k / 4 + j / 4 + 5 * j) % 7;
        int primero = (h + 5) % 7; // 0 = lunes ... 6 = domingo

        System.out.println(nombres[mes - 1] + " " + anio);
        System.out.println("Lu Ma Mi Ju Vi Sá Do");
        StringBuilder linea = new StringBuilder("   ".repeat(primero));
        int finde = 0;
        for (int d = 1; d <= dias; d++) {
            int col = (primero + d - 1) % 7;
            if (col >= 5) finde++;
            linea.append(String.format("%2d", d));
            if (col == 6 || d == dias) {
                System.out.println(linea);
                linea.setLength(0);
            } else {
                linea.append(' ');
            }
        }
        System.out.println(dias + " días · " + finde + " de fin de semana · empieza en " + semana[primero]);
    }
}
```

Zeller numera los días empezando en sábado (0) y el calendario empieza en lunes, así que `(h + 5) % 7` traduce un orden al otro: el lunes (2) pasa a 0 y el domingo (1) a 6.

Cada día ocupa la columna `(primero + d - 1) % 7`. Se escribe con `%2d` y un espacio detrás, salvo al final de la semana (columna 6) o del mes, donde se imprime la línea; así nunca quedan espacios al final. Los huecos iniciales son tres espacios por cada columna vacía.

38. Números primos con una función

Medio · Métodos · apuntesdam.com/subject/programacion/topic/metodos

Solución (java)

```
import java.util.Scanner;

public class Main {

    static boolean esPrimo(int n) {
        if (n < 2) return false;
        for (int d = 2; d * d <= n; d++) {
            if (n % d == 0) return false;    // tiene un divisor: no es primo
        }
        return true;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int limite = sc.nextInt();
        StringBuilder sb = new StringBuilder();
        for (int i = 2; i <= limite; i++) {
            if (esPrimo(i)) sb.append(i).append(" ");
        }
        System.out.println(sb.toString().trim());
    }
}
```

Un número es primo si no tiene divisores entre 2 y él mismo. Basta con probar hasta la raíz cuadrada ($d * d \leq n$): si n tuviera un divisor mayor, también tendría otro menor que ya se habría encontrado.

En cuanto aparece un divisor, `return false` termina la función sin seguir probando.

39. Factorial recursivo

Medio · Métodos · apuntesdam.com/subject/programacion/topic/metodos

Solución (java)

```
import java.util.Scanner;

public class Main {

    static long factorial(int n) {
        if (n <= 1) return 1;           // caso base: 0! = 1! = 1
        return n * factorial(n - 1);    // caso recursivo
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        System.out.println(n + "! = " + factorial(n));
    }
}
```

Toda función recursiva necesita un caso base que no se llame a sí misma (aquí 0! y 1! valen 1) y un caso recursivo que se acerque a él: $n! = n \cdot (n - 1)!$.

El resultado es `long` porque el factorial crece muy deprisa: 13! ya no cabe en un `int`.

40. Encuentra el fallo: el método que no cambia nada

Medio · Métodos · apuntesdam.com/subject/programacion/topic/metodos

Solución (java)

```
public class Main {
    static double aplicarDescuento(double precio, int porcentaje) {
        return precio - precio * porcentaje / 100;
    }

    public static void main(String[] args) {
        double precio = 50;
        precio = aplicarDescuento(precio, 20);
        System.out.println("Precio final: " + precio);
    }
}
```

En Java los parámetros se pasan por valor: el método recibe una copia de `precio`, y cambiar la copia no afecta a la variable del `main`.

La solución es que el método devuelva el nuevo precio (`double` en lugar de `void`, con `return`) y que quien lo llama guarde el resultado: `precio = aplicarDescuento(precio, 20);`.

41. Las torres de Hanói

Difícil · Métodos · apuntesdam.com/subject/programacion/topic/metodos

Solución (java)

```
import java.util.Scanner;

public class Main {
    static int movimientos = 0;

    static void hanoi(int n, char origen, char destino, char auxiliar, boolean mostrar) {
        if (n == 0) return;
        hanoi(n - 1, origen, auxiliar, destino, mostrar); // los n-1 de encima, al auxiliar
        movimientos++;
        if (mostrar) System.out.println("Disco " + n + ": " + origen + " → " + destino);
        hanoi(n - 1, auxiliar, destino, origen, mostrar); // y otra vez encima del disco n
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        hanoi(n, 'A', 'C', 'B', n <= 4);
        System.out.println(n + (n == 1 ? " disco: " : " discos: ") + movimientos + (movimientos == 1 ? " movimiento" : " movimientos") + " (2^" + n + " - 1 = " + ((1L << n) - 1) + ")");
    }
}
```

Las tres líneas de la solución recursiva se escriben tal cual: los $n-1$ discos de encima van al poste auxiliar (usando el destino como ayuda), el disco n va al destino, y los $n-1$ vuelven encima de él (usando el origen como ayuda).

Lo único que cambia entre las llamadas es el papel de cada poste. Con 0 discos no hay nada que hacer: ese es el caso base.

42. Una calculadora con descenso recursivo

Muy difícil · Métodos · apuntesdam.com/subject/programacion/topic/metodos

Solución (java)

```
import java.util.Scanner;

public class Main {
    static String texto;
    static int pos;

    static class ErrorExpresion extends RuntimeException {
        ErrorExpresion(String mensaje) { super(mensaje); }
    }

    static boolean hay(char c) {
        return pos < texto.length() && texto.charAt(pos) == c;
    }

    static long expresion() {
        long valor = termino();
        while (hay('+') || hay('-')) {
            char op = texto.charAt(pos++);
            long otro = termino();
            valor = op == '+' ? valor + otro : valor - otro;
        }
        return valor;
    }

    static long termino() {
        long valor = factor();
        while (hay('*') || hay('/') || hay('%')) {
            char op = texto.charAt(pos++);
            long otro = factor();
            if (op != '*' && otro == 0) throw new ErrorExpresion("división entre cero");
            valor = op == '*' ? valor * otro : op == '/' ? valor / otro : valor % otro;
        }
        return valor;
    }

    static long factor() {
        if (pos >= texto.length()) throw new ErrorExpresion("la expresión termina antes de tiempo");
        if (hay('-')) {
            pos++;
            return -factor();
        }
        if (hay('(')) {
            pos++;
            long valor = expresion(); // la recursividad: una expresión completa dentro del paréntesis
            if (!hay(')')) throw new ErrorExpresion("falta «)»");
            pos++;
            return valor;
        }
        int inicio = pos;
        while (pos < texto.length() && Character.isDigit(texto.charAt(pos))) pos++;
        if (inicio == pos) throw new ErrorExpresion("se esperaba un número en la posición " + pos);
        return Long.parseLong(texto.substring(inicio, pos));
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String linea = sc.nextLine();
            if (linea.isBlank()) continue;
            texto = linea.replace(" ", "");
            pos = 0;
            try {
                long valor = expresion();
                if (pos < texto.length()) throw new ErrorExpresion("sobra «" + texto.charAt(pos) + "» en la posición " + pos);
                System.out.println(linea.trim() + " = " + valor);
            } catch (ErrorExpresion e) {
                System.out.println(linea.trim() + " → error: " + e.getMessage());
            }
        }
    }
}
```

```

    }
  }
}

```

Cada nivel de prioridad es un método: **expresion** suma y resta términos, **termino** multiplica y divide factores y **factor** lee lo más prioritario (un número, un signo menos o un paréntesis). Como **expresion** llama a **termino** antes de mirar el **+**, la multiplicación se resuelve primero sin necesidad de reglas extra.

Los bucles **while** de **expresion** y **termino** acumulan de izquierda a derecha (así $100 / 7 \% 3$ es $(100 / 7) \% 3$). El paréntesis vuelve a llamar a **expresion**: esa recursividad es la que permite anidar paréntesis a cualquier profundidad.

El método auxiliar **hay(c)** comprueba el carácter actual sin salirse del texto, y cada error se lanza justo donde se detecta, con la posición en la que se ha quedado **pos**.

43. Máximo y mínimo de un array

Fácil · Arrays · apuntesdam.com/subject/programacion/topic/arrays

Solución (java)

```

public class Main {
    public static void main(String[] args) {
        int[] numeros = {34, 12, 78, 5, 90, 23, 67, 45};

        int max = numeros[0];
        int posMax = 0;
        int min = numeros[0];
        int posMin = 0;
        for (int i = 1; i < numeros.length; i++) {
            if (numeros[i] > max) {
                max = numeros[i];
                posMax = i;
            }
            if (numeros[i] < min) {
                min = numeros[i];
                posMin = i;
            }
        }

        System.out.println("Maximo: " + max + " (posicion " + posMax + ")");
        System.out.println("Minimo: " + min + " (posicion " + posMin + ")");
    }
}

```

Se parte del primer elemento como máximo y mínimo provisionales y se recorre el resto: cada vez que aparece uno mayor (o menor) se actualizan el valor y su posición. Un solo bucle basta para las dos cosas.

Empezar con **numeros[0]** en vez de con 0 es importante: si todos los números fueran negativos, un máximo inicial de 0 nunca se actualizaría.

44. Encuentra el fallo: el máximo de temperaturas bajo cero

Fácil · Arrays · apuntesdam.com/subject/programacion/topic/arrays

Solución (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] t = new int[n];
        for (int i = 0; i < n; i++) t[i] = sc.nextInt();
        int max = t[0];
        for (int x : t) {
            if (x > max) max = x;
        }
        System.out.println("Máxima: " + max);
    }
}
```

Con `max = 0`, si todas las temperaturas son negativas ninguna es mayor que 0 y el máximo nunca se actualiza. El valor inicial tiene que ser uno de los datos (`t[0]`) o el menor `int` posible (`Integer.MIN_VALUE`).

45. Invertir un array

Medio · Arrays · apuntesdam.com/subject/programacion/topic/arrays

Solución (java)

```
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        int[] array = {1, 2, 3, 4, 5, 6, 7, 8};

        System.out.println("Original: " + Arrays.toString(array));

        for (int i = 0, j = array.length - 1; i < j; i++, j--) {
            int tmp = array[i];
            array[i] = array[j];
            array[j] = tmp;
        }

        System.out.println("Invertido: " + Arrays.toString(array));
    }
}
```

Dos índices avanzan desde los extremos hacia el centro e intercambian sus elementos con una variable temporal. El bucle para cuando se cruzan (`i < j`); si siguiera hasta el final, volvería a intercambiar todo y el array quedaría como al principio.

46. Estadísticas completas de notas

Medio · Arrays · apuntesdam.com/subject/programacion/topic/arrays

Solución (java)

```
public class Main {
    public static void main(String[] args) {
        double[] notas = {8.5, 4.0, 6.5, 9.0, 3.5, 7.0, 5.0, 8.0, 2.5, 6.0};

        double suma = 0;
        int aprobados = 0;
        int suspensos = 0;
        double maxima = notas[0];
        double minima = notas[0];

        for (double nota : notas) {
            suma += nota;
            if (nota >= 5) aprobados++;
            else suspensos++;
            maxima = Math.max(maxima, nota);
            minima = Math.min(minima, nota);
        }

        double media = suma / notas.length;

        System.out.printf("Media: %.2f\n", media);
        System.out.println("Aprobados: " + aprobados);
        System.out.println("Suspensos: " + suspensos);
        System.out.printf("Nota maxima: %.1f\n", maxima);
        System.out.printf("Nota minima: %.1f\n", minima);
    }
}
```

Un `for-each` recorre las notas una vez y actualiza todos los acumuladores a la vez: la suma, los contadores de aprobados y suspensos, y el máximo y el mínimo con `Math.max` y `Math.min`.

La media se calcula después del bucle, cuando la suma ya está completa.

47. Suma de filas y columnas de una matriz

Difícil · Arrays · apuntesdam.com/subject/programacion/topic/arrays

Solución (java)

```
public class Main {
    public static void main(String[] args) {
        int[][] matriz = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };

        for (int i = 0; i < matriz.length; i++) {
            int sumaFila = 0;
            for (int j = 0; j < matriz[i].length; j++) {
                sumaFila += matriz[i][j];
            }
            System.out.println("Fila " + i + ": " + sumaFila);
        }

        for (int j = 0; j < matriz[0].length; j++) {
            int sumaCol = 0;
            for (int i = 0; i < matriz.length; i++) {
                sumaCol += matriz[i][j];
            }
            System.out.println("Col " + j + ": " + sumaCol);
        }
    }
}
```

En una fila, el índice que cambia es el de la columna (`matriz[i][j]` con `j` variando); en una columna es al revés, se fija `j` y se recorren las filas con `i`.

`matriz.length` es el número de filas y `matriz[0].length` el de columnas.

48. El juego de la vida de Conway

Muy difícil · Arrays · apuntesdam.com/subject/programacion/topic/arrays

Solución (java)

```
import java.util.*;

public class Main {
    static boolean[][] siguiente(boolean[][] t) {
        int filas = t.length, cols = t[0].length;
        boolean[][] nueva = new boolean[filas][cols];
        for (int i = 0; i < filas; i++) {
            for (int j = 0; j < cols; j++) {
                int vecinas = 0;
                for (int di = -1; di <= 1; di++) {
                    for (int dj = -1; dj <= 1; dj++) {
                        int a = i + di, b = j + dj;
                        if ((di != 0 || dj != 0) && a >= 0 && a < filas && b >= 0 && b < cols && t[a][b])
                            vecinas++;
                    }
                }
                nueva[i][j] = t[i][j] ? vecinas == 2 || vecinas == 3 : vecinas == 3;
            }
        }
        return nueva;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int generaciones = sc.nextInt();
        List<String> filas = new ArrayList<>();
        while (sc.hasNext()) filas.add(sc.next());
        boolean[][] t = new boolean[filas.size()][filas.get(0).length()];
        for (int i = 0; i < t.length; i++)
            for (int j = 0; j < t[0].length; j++) t[i][j] = filas.get(i).charAt(j) == '#';

        for (int g = 1; g <= generaciones; g++) {
            boolean[][] nueva = siguiente(t);
            if (Arrays.deepEquals(nueva, t)) {
                System.out.println("Estable desde la generación " + (g - 1));
                break;
            }
            t = nueva;
        }

        int vivas = 0;
        for (boolean[] fila : t) {
            StringBuilder sb = new StringBuilder();
            for (boolean c : fila) {
                sb.append(c ? '#' : '.');
                if (c) vivas++;
            }
            System.out.println(sb);
        }
        System.out.println("Células vivas: " + vivas);
    }
}
```

La clave es que todas las células cambian a la vez: por eso `siguiente` escribe en una matriz nueva y solo lee de la anterior. Si se modificara la misma matriz, las células ya actualizadas cambiarían el recuento de sus vecinas.

Las 8 vecinas se recorren con dos bucles de -1 a 1 , saltando la propia célula y las posiciones fuera de la cuadrícula.

`Arrays.deepEquals` compara las dos generaciones casilla a casilla para detectar cuándo se estabiliza.

49. Contar vocales

Fácil · Strings · apuntesdam.com/subject/programacion/topic/strings

Solución (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String frase = sc.nextLine();
        int vocales = 0;

        for (char c : frase.toCharArray()) {
            if ("aeiouAEIOU".indexOf(c) >= 0) vocales++;
        }

        System.out.println("Vocales: " + vocales);
    }
}
```

Se recorre la frase carácter a carácter y se cuenta el que aparece en la cadena "aeiouAEIOU": `indexOf` devuelve `-1` cuando no lo encuentra. Es más corto que cinco comparaciones con `||` y ya incluye las mayúsculas.

50. Encuentra el fallo: la contraseña que nunca es correcta

Fácil · Strings · apuntesdam.com/subject/programacion/topic/strings

Solución (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        String clave = new Scanner(System.in).nextLine();
        if (clave.equals("java2025")) {
            System.out.println("Acceso concedido");
        } else {
            System.out.println("Acceso denegado");
        }
    }
}
```

Con objetos, `==` compara si son el mismo objeto en memoria, no si tienen el mismo contenido. La cadena leída del teclado es un objeto nuevo, distinto del literal "java2025", aunque tenga las mismas letras.

Para comparar el contenido de dos `String` hay que usar `equals` (o `equalsIgnoreCase` si no importan las mayúsculas).

51. Palíndromos

Medio · Strings · apuntesdam.com/subject/programacion/topic/strings

Solución (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String frase = sc.nextLine();

        String limpia = frase.replace(" ", "").toLowerCase();
        String inversa = new StringBuilder(limpia).reverse().toString();
        boolean palindromo = limpia.equals(inversa);

        System.out.println(palindromo ? "Es palindromo" : "No es palindromo");
    }
}
```


Primero se normaliza la frase: sin espacios y en minúsculas. Después `StringBuilder.reverse()` le da la vuelta y `equals` compara el contenido de las dos cadenas.

52. Encuentra el fallo: el palíndromo exigente

Medio · Strings · apuntesdam.com/subject/programacion/topic/strings

Solución (java)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        String frase = new Scanner(System.in).nextLine();
        String limpia = frase.replace(" ", "").toLowerCase();
        String inversa = new StringBuilder(limpia).reverse().toString();
        System.out.println(limpia.equals(inversa) ? "Es palíndromo" : "No es palíndromo");
    }
}
```

La comparación era correcta, pero se hacía con la frase tal cual: la «A» mayúscula del principio no coincide con la «a» final y los espacios caen en otros sitios al darle la vuelta. Basta con limpiar la frase antes: `replace(" ", "")` y `toLowerCase()`.

53. El cifrado de Vigenère

Difícil · Strings · apuntesdam.com/subject/programacion/topic/strings

Solución (java)

```
import java.util.Scanner;

public class Main {
    static String vigenere(String texto, String clave, boolean cifrar) {
        StringBuilder sb = new StringBuilder();
        int k = 0; // letra de la clave que toca
        for (char c : texto.toCharArray()) {
            boolean mayus = c >= 'A' && c <= 'Z', minus = c >= 'a' && c <= 'z';
            if (!mayus && !minus) {
                sb.append(c); // no es una letra: se copia y no gasta clave
                continue;
            }
            int desp = Character.toUpperCase(clave.charAt(k % clave.length())) - 'A';
            if (!cifrar) desp = 26 - desp;
            char base = mayus ? 'A' : 'a';
            sb.append((char) (base + (c - base + desp) % 26));
            k++;
        }
        return sb.toString();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String modo = sc.nextLine().trim(), clave = sc.nextLine().trim(), texto = sc.nextLine();
        System.out.println(vigenere(texto, clave, modo.equals("cifrar")));
    }
}
```

Cada letra se convierte en un número del 0 al 25 restandole su base ('A' o 'a'), se le suma el desplazamiento y `% 26` hace que, al pasar de la Z, vuelva a la A. Después se suma la base otra vez para recuperar la letra.

El contador `k` solo avanza con las letras, así los espacios y los signos no consumen la clave. Para descifrar se desplaza `26 - desp`, que equivale a ir hacia atrás sin obtener números negativos.

54. Justificar un texto

Muy difícil · Strings · apuntesdam.com/subject/programacion/topic/strings

Solución (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int ancho = Integer.parseInt(sc.nextLine().trim());
        StringBuilder todo = new StringBuilder();
        while (sc.hasNextLine()) todo.append(sc.nextLine()).append(' ');
        String[] palabras = todo.toString().trim().split("\\s+");

        for (String p : palabras) {
            if (p.length() > ancho) {
                System.out.println("Error: la palabra «" + p + "» no cabe en " + ancho + " caracteres");
                return;
            }
        }

        // 1. Repartir: en cada línea, todas las palabras que quepan con un espacio entre ellas
        List<List<String>> lineas = new ArrayList<>();
        List<String> actual = new ArrayList<>();
        int largo = 0;
        for (String p : palabras) {
            if (!actual.isEmpty() && largo + 1 + p.length() > ancho) {
                lineas.add(actual);
                actual = new ArrayList<>();
                largo = 0;
            }
            largo += (actual.isEmpty() ? 0 : 1) + p.length();
            actual.add(p);
        }
        lineas.add(actual);

        // 2. Justificar
        for (int i = 0; i < lineas.size(); i++) {
            List<String> l = lineas.get(i);
            String texto;
            if (i == lineas.size() - 1 || l.size() == 1) {
                texto = String.join(" ", l);
            } else {
                int letras = 0;
                for (String p : l) letras += p.length();
                int huecos = l.size() - 1, espacios = ancho - letras;
                StringBuilder sb = new StringBuilder(l.get(0));
                for (int h = 0; h < huecos; h++) {
                    int n = espacios / huecos + (h < espacios % huecos ? 1 : 0); // los primeros, uno más
                    sb.append(" ".repeat(n)).append(l.get(h + 1));
                }
                texto = sb.toString();
            }
            System.out.println("|" + texto + " ".repeat(ancho - texto.length()) + "|");
        }
        System.out.println(lineas.size() + (lineas.size() == 1 ? " línea" : " líneas") + " de " + ancho + " caracteres");
    }
}
```

El problema se divide en dos fases. Primero se reparten las palabras: una palabra entra en la línea actual si la longitud acumulada, más un espacio, más la palabra no pasa del ancho; si no, la línea se cierra y empieza otra.

Después, para justificar, se calculan los espacios que faltan (`ancho - letras`) y se reparten entre los huecos: todos reciben `espacios / huecos` y los `espacios % huecos` primeros reciben uno más. La última línea y las de una sola palabra se dejan con un espacio y se rellenan por la derecha.

55. Encuentra el fallo: el alumno sin nombre

Fácil · Clases y Objetos · apuntesdam.com/subject/programacion/topic/clases

Solución (java)

```
public class Main {
    static class Alumno {
        private String nombre;
        private double nota;

        Alumno(String nombre, double nota) {
            this.nombre = nombre;
            this.nota = nota;
        }

        @Override
        public String toString() {
            return nombre + ": " + nota;
        }
    }

    public static void main(String[] args) {
        System.out.println(new Alumno("Lucía", 8.5));
        System.out.println(new Alumno("Mario", 6.0));
    }
}
```

Dentro del constructor, `nombre` se refiere al parámetro, que tapa al atributo del mismo nombre. `nombre = nombre` asigna el parámetro a sí mismo y el atributo se queda con su valor por defecto (`null` y `0.0`).

`this.nombre` indica que es el atributo del objeto: `this.nombre = nombre` copia el parámetro en él.

56. Una clase Fraccion immutable

Difícil · Clases y Objetos · apuntesdam.com/subject/programacion/topic/clases

Solución (java)

```
import java.util.Scanner;

class Fraccion {
    private final long num;
    private final long den;

    Fraccion(long num, long den) {
        if (den == 0) throw new IllegalArgumentException("denominador 0");
        if (den < 0) { // el signo, en el numerador
            num = -num;
            den = -den;
        }
        long m = mcd(num, den);
        this.num = num / m;
        this.den = den / m;
    }

    private static long mcd(long a, long b) {
        return b == 0 ? Math.abs(a) : mcd(b, a % b);
    }

    Fraccion sumar(Fraccion o) {
        return new Fraccion(num * o.den + o.num * den, den * o.den);
    }

    Fraccion restar(Fraccion o) {
        return new Fraccion(num * o.den - o.num * den, den * o.den);
    }

    Fraccion multiplicar(Fraccion o) {
        return new Fraccion(num * o.num, den * o.den);
    }

    Fraccion dividir(Fraccion o) {
        if (o.num == 0) throw new ArithmeticException("división entre cero");
        return new Fraccion(num * o.den, den * o.num);
    }

    @Override
    public String toString() {
        return den == 1 ? String.valueOf(num) : num + "/" + den;
    }
}

public class Main {
    static Fraccion leer(String s) {
        String[] p = s.split("/");
        return new Fraccion(Long.parseLong(p[0]), p.length > 1 ? Long.parseLong(p[1]) : 1);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = Integer.parseInt(sc.nextLine().trim());
        for (int i = 0; i < n; i++) {
            String linea = sc.nextLine().trim();
            String[] t = linea.split(" ");
            try {
                Fraccion a = leer(t[0]), b = leer(t[2]);
                Fraccion r = switch (t[1]) {
                    case "+" -> a.sumar(b);
                    case "-" -> a.restar(b);
                    case "*" -> a.multiplicar(b);
                    default -> a.dividir(b);
                };
                System.out.println(linea + " = " + r);
            } catch (ArithmeticException | IllegalArgumentException e) {
                System.out.println(linea + " → Error: " + e.getMessage());
            }
        }
    }
}
```

```
}  
    }  
}
```

Todo el trabajo de normalizar se hace en el constructor: rechaza el denominador 0, pasa el signo al numerador y divide los dos entre su máximo común divisor. Así cualquier fracción que exista ya está simplificada, y las operaciones no tienen que preocuparse de ello.

Como los campos son `final`, las operaciones no pueden modificar el objeto: calculan el resultado con las fórmulas de siempre ($a/b + c/d = (ad + cb)/bd$) y devuelven `new Fraccion(...)`. Eso es una clase inmutable, como `String` o `BigDecimal`.

Con un numerador 0, `mcd(0, den)` vale `den`, así que 0/5 queda como 0/1 y se muestra «0».

57. Máquina expendedora

Muy difícil · Clases y Objetos · apuntesdam.com/subject/programacion/topic/clases

Solución (java)

```
import java.util.*;

class Producto {
    final String codigo, nombre;
    final int precio;           // en céntimos
    int stock;

    Producto(String codigo, String nombre, int precio, int stock) {
        this.codigo = codigo;
        this.nombre = nombre;
        this.precio = precio;
        this.stock = stock;
    }
}

class Maquina {
    static final int[] VALORES = {200, 100, 50, 20, 10, 5};
    private final Map<String, Producto> productos = new LinkedHashMap<>();
    private final Map<Integer, Integer> monedas = new TreeMap<>(Comparator.reverseOrder());
    private final List<Integer> insertadas = new ArrayList<>();

    Maquina() {
        for (int v : VALORES) monedas.put(v, 0);
    }

    static String euros(int centimos) {
        return centimos / 100 + "," + String.format("%02d", centimos % 100) + " €";
    }

    void cargarMonedas(int valor, int cantidad) {
        monedas.merge(valor, cantidad, Integer::sum);
    }

    void añadirProducto(Producto p) {
        productos.put(p.codigo, p);
    }

    private int credito() {
        int total = 0;
        for (int m : insertadas) total += m;
        return total;
    }

    // «1x2,00 €, 1x0,50 €» a partir de un mapa valor → cantidad
    private static String lista(Map<Integer, Integer> cantidades) {
        List<String> partes = new ArrayList<>();
        for (int v : VALORES) {
            int n = cantidades.getOrDefault(v, 0);
            if (n > 0) partes.add(n + "x" + euros(v));
        }
        return partes.isEmpty() ? "nada" : String.join(", ", partes);
    }

    String insertar(int valor) {
        if (Arrays.stream(VALORES).noneMatch(v -> v == valor)) return "Moneda no válida: " + valor;
        insertadas.add(valor);
        return "Crédito: " + euros(credito());
    }

    String comprar(String codigo) {
        Producto p = productos.get(codigo);
        if (p == null) return "No existe el producto " + codigo;
        if (p.stock == 0) return p.nombre + " agotado";
        int credito = credito();
        if (credito < p.precio) return "Saldo insuficiente: faltan " + euros(p.precio - credito);

        // Monedas disponibles: la caja más las que se acaban de insertar
        Map<Integer, Integer> disponibles = new TreeMap<>(monedas);
    }
}
```

```

        for (int m : insertadas) disponibles.merge(m, 1, Integer::sum);
        Map<Integer, Integer> cambio = new TreeMap<>();
        int resto = credito - p.precio;
        for (int v : VALORES) {
            int n = Math.min(resto / v, disponibles.get(v));
            if (n > 0) {
                cambio.put(v, n);
                disponibles.put(v, disponibles.get(v) - n);
                resto -= n * v;
            }
        }
        if (resto != 0) return "No hay cambio: inserte el importe exacto";

        // Se vende: la caja pasa a ser lo disponible menos el cambio entregado
        monedas.putAll(disponibles);
        insertadas.clear();
        p.stock--;
        return "Sale " + p.nombre + ". Cambio: " + lista(cambio);
    }

    String devolver() {
        Map<Integer, Integer> cantidades = new TreeMap<>();
        for (int m : insertadas) cantidades.merge(m, 1, Integer::sum);
        insertadas.clear();
        return "Devuelto: " + lista(cantidades);
    }

    String estado() {
        StringBuilder sb = new StringBuilder();
        for (Producto p : productos.values()) {
            sb.append(p.codigo).append(' ').append(p.nombre).append(' ').append(euros(p.precio))
              .append(" (").append(p.stock).append(")\n");
        }
        int caja = 0;
        for (Map.Entry<Integer, Integer> e : monedas.entrySet()) caja += e.getKey() * e.getValue();
        return sb.append("Caja: ").append(euros(caja)).append('\n').toString();
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Maquina m = new Maquina();
        while (sc.hasNextLine()) {
            String[] t = sc.nextLine().trim().split(" ");
            switch (t[0]) {
                case "MONEDAS" -> m.cargarMonedas(Integer.parseInt(t[1]), Integer.parseInt(t[2]));
                case "PRODUCTO" -> m.añadirProducto(new Producto(t[1], t[2], Integer.parseInt(t[3]),
                    Integer.parseInt(t[4])));
                case "INSERTAR" -> System.out.println(m.insertar(Integer.parseInt(t[1])));
                case "COMPRAR" -> System.out.println(m.comprar(t[1]));
                case "DEVOLVER" -> System.out.println(m.devolver());
                case "ESTADO" -> System.out.print(m.estado());
                default -> { }
            }
        }
    }
}

```

Trabajar en céntimos con `int` evita los errores de redondeo de `double` (0,1 + 0,2 no da exactamente 0,3). El crédito no se guarda aparte: se calcula sumando las monedas insertadas, así nunca puede desajustarse.

Para el cambio se hace una copia de las monedas disponibles (caja + insertadas) y se reparte de mayor a menor, tomando de cada valor tantas como quepan y haya. Si sobra algo, no se puede dar exacto y la máquina no toca nada; si cuadra, la copia pasa a ser la nueva caja.

Los métodos auxiliares `credito` y `lista` evitan repetir código: `lista` sirve tanto para el cambio como para la devolución.

58. Cuentas bancarias con herencia

Difícil · Herencia · apuntesdam.com/subject/programacion/topic/herencia

Solución (java)

```
import java.util.*;

abstract class Cuenta {
    private final String iban;
    protected double saldo;

    Cuenta(String iban, double saldoInicial) {
        this.iban = iban;
        this.saldo = saldoInicial;
    }

    String getIban() { return iban; }
    double getSaldo() { return saldo; }

    void ingresar(double importe) {
        if (importe <= 0) throw new IllegalArgumentException("importe no válido");
        saldo += importe;
    }

    // Cada tipo de cuenta decide si se puede retirar
    abstract void retirar(double importe);

    // Se llama al final de cada mes
    abstract void cierreMes();

    abstract String tipo();

    @Override
    public String toString() {
        return String.format(Locale.US, "%s %s: %.2f", tipo(), iban, saldo);
    }
}

class CuentaCorriente extends Cuenta {
    static final double DESCUBIERTO = -300;

    CuentaCorriente(String iban, double saldo) { super(iban, saldo); }

    void retirar(double importe) {
        if (importe <= 0) throw new IllegalArgumentException("importe no válido");
        if (saldo - importe < DESCUBIERTO) throw new IllegalStateException("supera el descubierto");
        saldo -= importe;
    }

    void cierreMes() {
        if (saldo < 0) saldo -= 2;
    }

    String tipo() { return "Corriente"; }
}

class CuentaAhorro extends Cuenta {
    private int retiradas = 0;

    CuentaAhorro(String iban, double saldo) { super(iban, saldo); }

    void retirar(double importe) {
        if (importe <= 0) throw new IllegalArgumentException("importe no válido");
        if (retiradas == 3) throw new IllegalStateException("máximo de retiradas del mes");
        if (importe > saldo) throw new IllegalStateException("saldo insuficiente");
        saldo -= importe;
        retiradas++;
    }

    void cierreMes() {
        saldo *= 1.005;
        retiradas = 0;
    }
}
```



```

    String tipo() { return "Ahorro"; }
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, Cuenta> cuentas = new LinkedHashMap<>();
        while (sc.hasNextLine()) {
            String[] t = sc.nextLine().trim().split(" ");
            try {
                switch (t[0]) {
                    case "CORRIENTE" -> cuentas.put(t[1], new CuentaCorriente(t[1],
                        Double.parseDouble(t[2])));
                    case "AHORRO" -> cuentas.put(t[1], new CuentaAhorro(t[1], Double.parseDouble(t[2])));
                    case "INGRESAR" -> cuentas.get(t[1]).ingresar(Double.parseDouble(t[2]));
                    case "RETIRAR" -> cuentas.get(t[1]).retirar(Double.parseDouble(t[2]));
                    case "MES" -> {
                        for (Cuenta c : cuentas.values()) c.cierreMes();
                        System.out.println("-- cierre de mes --");
                    }
                    case "VER" -> cuentas.values().forEach(System.out::println);
                }
            } catch (IllegalArgumentException | IllegalStateException e) {
                System.out.println(String.join(" ", t) + " → " + e.getMessage());
            }
        }
    }
}

```

Cada subclase implementa a su manera los métodos abstractos `retirar` y `cierreMes`, y el `main` las trata a todas como `Cuenta`: cuando llama a `c.cierreMes()`, Java ejecuta la versión de la clase real del objeto. Eso es el polimorfismo.

Las comprobaciones van antes de tocar el saldo, y cada una lanza su excepción: si una falla, la operación no se hace a medias. `saldo` es `protected` en la clase base precisamente para que las subclases puedan modificarlo.

El contador de retiradas es un atributo propio de `CuentaAhorro`, porque solo tiene sentido en ese tipo de cuenta; el cierre de mes lo reinicia.

59. Encuentra el fallo: el producto que no está en la lista

Difícil · Herencia · apuntesdam.com/subject/programacion/topic/herencia

Solución (java)

```
import java.util.*;

public class Main {
    static class Producto {
        final String codigo;
        final String nombre;
        Producto(String codigo, String nombre) { this.codigo = codigo; this.nombre = nombre; }

        @Override
        public boolean equals(Object o) {
            if (this == o) return true;
            if (!(o instanceof Producto otro)) return false;
            return codigo.equals(otro.codigo);
        }

        @Override
        public int hashCode() {
            return codigo.hashCode();
        }
    }

    public static void main(String[] args) {
        List<Producto> lista = new ArrayList<>(List.of(new Producto("A1", "Teclado"), new Producto("B2", "Ratón")));
        System.out.println("¿Está B2? " + lista.contains(new Producto("B2", "Ratón inalámbrico")));
        System.out.println("¿Está C3? " + lista.contains(new Producto("C3", "Monitor")));
    }
}
```

`equals(Producto otro)` no sobrescribe el `equals` de `Object`: lo sobrecarga, porque el parámetro es de otro tipo. `contains` llama a `equals(Object)`, que sigue siendo el original y compara referencias, así que nunca encuentra un objeto distinto.

Con el parámetro `Object` y la anotación `@Override` (que hace que el compilador avise si no se está sobrescribiendo nada) el problema desaparece. Y siempre que se redefine `equals` hay que redefinir `hashCode` con los mismos campos, o los `HashSet` y `HashMap` fallarán de la misma forma.

60. Expresiones matemáticas con polimorfismo

Muy difícil · Herencia · apuntesdam.com/subject/programacion/topic/herencia

Solución (java)

```
import java.util.*;

abstract class Expr {
    abstract double evaluar(Map<String, Double> vars);
    abstract Expr derivar(String v);
    Expr simplificar() { return this; }

    static String num(double d) {
        return d == Math.floor(d) && !Double.isInfinite(d) ? String.valueOf((long) d) : String.valueOf(d);
    }

    // ¿Es el número n?
    static boolean es(Expr e, double n) {
        return e instanceof Num x && x.valor == n;
    }
}

class Num extends Expr {
    final double valor;
    Num(double valor) { this.valor = valor; }
    double evaluar(Map<String, Double> vars) { return valor; }
    Expr derivar(String v) { return new Num(0); }
    public String toString() { return num(valor); }
}

class Var extends Expr {
    final String nombre;
    Var(String nombre) { this.nombre = nombre; }
    double evaluar(Map<String, Double> vars) { return vars.get(nombre); }
    Expr derivar(String v) { return new Num(nombre.equals(v) ? 1 : 0); }
    public String toString() { return nombre; }
}

// Clase base de las operaciones con dos operandos
abstract class Binaria extends Expr {
    final Expr izq, der;
    Binaria(Expr izq, Expr der) { this.izq = izq; this.der = der; }
    abstract String simbolo();
    public String toString() { return "(" + izq + " " + simbolo() + " " + der + ")"; }
}

class Suma extends Binaria {
    Suma(Expr a, Expr b) { super(a, b); }
    String simbolo() { return "+"; }
    double evaluar(Map<String, Double> vars) { return izq.evaluar(vars) + der.evaluar(vars); }
    Expr derivar(String v) { return new Suma(izq.derivar(v), der.derivar(v)); }

    Expr simplificar() {
        Expr a = izq.simplificar(), b = der.simplificar();
        if (a instanceof Num x && b instanceof Num y) return new Num(x.valor + y.valor);
        if (es(a, 0)) return b;
        if (es(b, 0)) return a;
        return new Suma(a, b);
    }
}

class Resta extends Binaria {
    Resta(Expr a, Expr b) { super(a, b); }
    String simbolo() { return "-"; }
    double evaluar(Map<String, Double> vars) { return izq.evaluar(vars) - der.evaluar(vars); }
    Expr derivar(String v) { return new Resta(izq.derivar(v), der.derivar(v)); }

    Expr simplificar() {
        Expr a = izq.simplificar(), b = der.simplificar();
        if (a instanceof Num x && b instanceof Num y) return new Num(x.valor - y.valor);
        if (es(b, 0)) return a;
        if (a.toString().equals(b.toString())) return new Num(0);
        return new Resta(a, b);
    }
}
```

```

    }
}

class Producto extends Binaria {
    Producto(Expr a, Expr b) { super(a, b); }
    String simbolo() { return "*"; }
    double evaluar(Map<String, Double> vars) { return izq.evaluar(vars) * der.evaluar(vars); }

    // (a·b)' = a'·b + a·b'
    Expr derivar(String v) {
        return new Suma(new Producto(izq.derivar(v), der), new Producto(izq, der.derivar(v)));
    }

    Expr simplificar() {
        Expr a = izq.simplificar(), b = der.simplificar();
        if (a instanceof Num x && b instanceof Num y) return new Num(x.valor * y.valor);
        if (es(a, 0) || es(b, 0)) return new Num(0);
        if (es(a, 1)) return b;
        if (es(b, 1)) return a;
        return new Producto(a, b);
    }
}

public class Main {
    // Lee una expresión en notación prefija: "+ * 2 x 3" es (2 * x) + 3
    static Expr leer(Iterator<String> it) {
        String t = it.next();
        return switch (t) {
            case "+" -> new Suma(leer(it), leer(it));
            case "-" -> new Resta(leer(it), leer(it));
            case "*" -> new Producto(leer(it), leer(it));
            default -> Character.isLetter(t.charAt(0)) ? new Var(t) : new Num(Double.parseDouble(t));
        };
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Expr e = leer(Arrays.asList(sc.nextLine().trim().split(" ")).iterator());
        Map<String, Double> vars = new TreeMap<>();
        while (sc.hasNext()) vars.put(sc.next(), Double.parseDouble(sc.next()));
        System.out.println("Expresión: " + e);
        System.out.println("Simplificada: " + e.simplificar());
        System.out.println("Valor: " + Expr.num(e.evaluar(vars)));
        Expr d = e.derivar("x");
        System.out.println("Derivada respecto de x: " + d);
        System.out.println("Derivada simplificada: " + d.simplificar());
    }
}

```

Cada clase solo sabe hacer su parte y delega en sus operandos: una `Suma` se evalúa sumando lo que devuelven `izq.evaluar` y `der.evaluar`, sin saber si son números, variables u otras operaciones. La recursión termina en las hojas del árbol (`Num` y `Var`).

`derivar` construye un árbol nuevo con las reglas de derivación; por eso sale lleno de `0 *` y `* 1`. `simplificar` lo limpia de abajo arriba: primero simplifica los operandos y luego aplica las reglas a lo que queda. El orden importa: si se comprobaran las reglas antes de simplificar los hijos, `(1 * y)` no se reconocería como `y`.

El método auxiliar `es(e, n)` usa `instanceof` con patrón (`e instanceof Num x`), que comprueba el tipo y declara la variable ya convertida en una sola expresión.

61. Clasificación de una liga con Comparator

Difícil · Interfaces · apuntesdam.com/subject/programacion/topic/interfaces

Solución (java)

```
import java.util.*;

class Equipo {
    final String nombre;
    int jugados, ganados, empatados, perdidos, golesFavor, golesContra;

    Equipo(String nombre) { this.nombre = nombre; }

    int puntos() { return ganados * 3 + empatados; }
    int diferencia() { return golesFavor - golesContra; }

    void registrar(int favor, int contra) {
        jugados++;
        golesFavor += favor;
        golesContra += contra;
        if (favor > contra) ganados++;
        else if (favor == contra) empatados++;
        else perdidos++;
    }
}

public class Main {
    static final Comparator<Equipo> CLASIFICACION = Comparator
        .comparingInt(Equipo::puntos).reversed()
        .thenComparing(Comparator.comparingInt(Equipo::diferencia).reversed())
        .thenComparing(Comparator.comparingInt((Equipo e) -> e.golesFavor).reversed())
        .thenComparing(e -> e.nombre);

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, Equipo> equipos = new HashMap<>();
        while (sc.hasNextLine()) {
            String linea = sc.nextLine().trim();
            if (linea.isEmpty()) continue;
            // Formato: Local 2-1 Visitante
            String[] t = linea.split(" ");
            String[] goles = t[1].split("-");
            int gl = Integer.parseInt(goles[0]), gv = Integer.parseInt(goles[1]);
            equipos.computeIfAbsent(t[0], Equipo::new).registrar(gl, gv);
            equipos.computeIfAbsent(t[2], Equipo::new).registrar(gv, gl);
        }
        List<Equipo> tabla = new ArrayList<>(equipos.values());
        tabla.sort(CLASIFICACION);
        System.out.println("Pos Equipo      PJ  G  E  P  GF  GC  DG Pts");
        int pos = 1;
        for (Equipo e : tabla)
            System.out.printf("%3d %-10s %3d %2d %2d %2d %3d %3d %+3d %3d%n", pos++, e.nombre, e.jugados,
                e.ganados, e.empatados, e.perdidos, e.golesFavor, e.golesContra, e.diferencia(),
                e.puntos());
    }
}
```

registrar recibe los goles desde el punto de vista del equipo, así sirve igual para el local y para el visitante: el **main** lo llama con **(gl, gv)** para uno y **(gv, gl)** para el otro.

El **Comparator** se encadena con **thenComparing**, que solo se consulta cuando los criterios anteriores empatan. Cada criterio de mayor a menor lleva su propio **.reversed()**: si se pusiera uno solo al final, invertiría toda la cadena, y el nombre también quedaría de la Z a la A.

62. Reglas de contraseñas componibles

Muy difícil · Interfaces · apuntesdam.com/subject/programacion/topic/interfaces

Solución (java)

```
import java.util.*;
import java.util.regex.Pattern;

@FunctionalInterface
interface Regla {
    // Lista de errores; vacía si el texto cumple la regla
    List<String> comprobar(String s);

    default Regla y(Regla otra) {
        return s -> {
            List<String> errores = new ArrayList<>(comprobar(s));
            errores.addAll(otra.comprobar(s));
            return errores;
        };
    }

    default Regla o(Regla otra) {
        return s -> {
            List<String> e1 = comprobar(s);
            if (e1.isEmpty()) return e1;
            List<String> e2 = otra.comprobar(s);
            if (e2.isEmpty()) return e2;
            return List.of(String.join("; ", e1) + " o " + String.join("; ", e2));
        };
    }

    static Regla minimo(int n) {
        return s -> s.length() >= n ? List.of() : List.of("mínimo " + n + " caracteres");
    }

    static Regla contiene(String clase, String nombre) {
        Pattern patron = Pattern.compile("[ " + clase + " ]");
        return s -> patron.matcher(s).find() ? List.of() : List.of("falta " + nombre);
    }

    static Regla noContiene(String palabra) {
        return s -> s.toLowerCase().contains(palabra.toLowerCase())
            ? List.of("no puede contener «" + palabra + "»")
            : List.of();
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String politica = sc.nextLine().trim();
        Regla regla = switch (politica) {
            case "BASICA" -> Regla.minimo(6).y(Regla.contiene("0-9", "un número"));
            default -> Regla.minimo(10)
                .y(Regla.contiene("0-9", "un número"))
                .y(Regla.contiene("A-Z", "una mayúscula").o(Regla.contiene("^a-zA-Z0-9", "un símbolo")))
                .y(Regla.noContiene("password"))
                .y(Regla.noContiene("1234"));
        };
        while (sc.hasNextLine()) {
            String clave = sc.nextLine();
            List<String> errores = regla.comprobar(clave);
            System.out.println(errores.isEmpty() ? "✓ " + clave : "X " + clave + ": " + String.join("; ", errores));
        }
    }
}
```

Como `Regla` tiene un único método abstracto, cualquier lambda `s -> lista` es una `Regla`. Los métodos `static` son fábricas que devuelven reglas sencillas, y los `default` (`y`, `o`) combinan la regla actual con otra devolviendo una lambda nueva: así se construyen políticas complejas encadenando piezas.

Dentro de la lambda de un método `default`, `comprobar(s)` se refiere a la regla sobre la que se llamó a `y` u `o`, porque una lambda no tiene su propio `this`. `o` no evalúa la segunda regla si la primera ya se cumple.

En `contiene`, el patrón se compila una sola vez, fuera de la lambda, y `"[" + clase + "]"` forma la clase de caracteres; con `^a-zA-Z0-9` queda `[^a-zA-Z0-9]`, «cualquier cosa que no sea letra ni número».

63. Validar y resumir un CSV de ventas

Difícil · Ficheros y Streams · apuntesdam.com/subject/programacion/topic/ficheros

Solución (java)

```
import java.io.*;
import java.util.*;

public class Main {
    public static void main(String[] args) throws IOException {
        // La entrada estándar hace de fichero: se lee igual que con un FileReader
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        Map<String, Double> ventas = new HashMap<>();
        int correctas = 0, erroneas = 0;
        String cabecera = br.readLine();
        String linea;
        int numLinea = 1;
        while ((linea = br.readLine()) != null) {
            numLinea++;
            if (linea.isBlank()) continue;
            String error = null;
            String[] c = linea.split(";", -1);
            String producto = "";
            int unidades = 0;
            double precio = 0;
            if (c.length != 4) {
                error = "se esperaban 4 campos y hay " + c.length;
            } else if ((producto = c[1].trim()).isEmpty()) {
                error = "producto vacío";
            } else {
                try {
                    unidades = Integer.parseInt(c[2].trim());
                } catch (NumberFormatException e) {
                    error = "unidades no válidas («" + c[2].trim() + "»)";
                }
                if (error == null) {
                    try {
                        precio = Double.parseDouble(c[3].trim().replace(',', '.'));
                    } catch (NumberFormatException e) {
                        error = "precio no válido («" + c[3].trim() + "»)";
                    }
                }
                if (error == null && (unidades <= 0 || precio < 0)) error = "valores negativos o cero";
            }
            if (error != null) {
                System.out.println("Línea " + numLinea + ": " + error);
                erroneas++;
            } else {
                ventas.merge(producto, unidades * precio, Double::sum);
                correctas++;
            }
        }

        System.out.println("--- " + correctas + " líneas correctas, " + erroneas + " con errores ---");
        List<Map.Entry<String, Double>> orden = new ArrayList<>(ventas.entrySet());
        orden.sort(Map.Entry.<String, Double>.comparingByValue().reversed().thenComparing(Map.Entry.comparingByKey()));
        double total = 0;
        for (Map.Entry<String, Double> e : orden) {
            System.out.printf(Locale.US, "%-10s %9.2f%n", e.getKey(), e.getValue());
            total += e.getValue();
        }
        System.out.printf(Locale.US, "%-10s %9.2f%n", "TOTAL", total);
    }
}
```

Cada línea se valida por pasos y se guarda solo el primer error: las comprobaciones posteriores se saltan en cuanto `error` deja de ser `null`. `split(";", -1)` es importante: sin el `-1`, Java descarta los campos vacíos del final y una línea como «2025-03-02;Zumo;1;» parecería tener 3 campos.

Los números con coma decimal se convierten cambiando la coma por un punto antes de `Double.parseDouble`, y los fallos de conversión se capturan con `NumberFormatException` para convertirlos en un mensaje.

`merge(producto, importe, Double::sum)` acumula por producto. Para ordenar, las entradas del mapa se pasan a una lista y se ordenan por valor descendente y, en caso de empate, por clave.

64. Comparar dos ficheros (diff)

Muy difícil · Ficheros y Streams · apuntesdam.com/subject/programacion/topic/ficheros

Solución (java)

```
import java.io.*;
import java.util.*;

public class Main {
    // Devuelve las líneas del diff: " línea" (igual), "-línea" (solo en a), "+línea" (solo en b)
    static List<String> diff(List<String> a, List<String> b) {
        int n = a.size(), m = b.size();
        // lcs[i][j] = longitud de la LCS de a desde i y b desde j
        int[][] lcs = new int[n + 1][m + 1];
        for (int i = n - 1; i >= 0; i--) {
            for (int j = m - 1; j >= 0; j--) {
                lcs[i][j] = a.get(i).equals(b.get(j))
                    ? lcs[i + 1][j + 1] + 1
                    : Math.max(lcs[i + 1][j], lcs[i][j + 1]);
            }
        }
        List<String> res = new ArrayList<>();
        int i = 0, j = 0;
        while (i < n && j < m) {
            if (a.get(i).equals(b.get(j))) {
                res.add(" " + a.get(i++));
                j++;
            } else if (lcs[i + 1][j] >= lcs[i][j + 1]) {
                res.add("-" + a.get(i++));
            } else {
                res.add "+" + b.get(j++);
            }
        }
        while (i < n) res.add("-" + a.get(i++));
        while (j < m) res.add "+" + b.get(j++);
        return res;
    }

    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        List<String> a = new ArrayList<>(), b = new ArrayList<>();
        List<String> actual = a;
        String linea;
        while ((linea = br.readLine()) != null) {
            if (linea.equals("====")) { actual = b; continue; }
            actual.add(linea);
        }
        List<String> d = diff(a, b);
        int quitadas = 0, añadidas = 0;
        for (String l : d) {
            System.out.println(l);
            if (l.startsWith("-")) quitadas++;
            if (l.startsWith("+")) añadidas++;
        }
        System.out.println("@@ " + quitadas + (quitadas == 1 ? " línea quitada, " : " líneas quitadas, ")
            + añadidas + (añadidas == 1 ? " añadida @" : " añadidas @@"));
    }
}
```


Es programación dinámica: `lcs[i][j]` guarda cuántas líneas comunes se pueden conservar como máximo entre el final de la versión antigua (desde `i`) y el de la nueva (desde `j`). Se rellena desde el final porque cada casilla depende de la de su derecha y la de abajo; la fila y la columna extra valen 0.

Con la tabla hecha, el recorrido decide cada paso sin probar alternativas: si las líneas coinciden se conservan, y si no, la tabla dice si se pierde menos quitando la antigua o añadiendo la nueva. Lo que quede en una de las dos versiones cuando la otra se acaba son líneas quitadas o añadidas.

65. Encuentra el fallo: borrar mientras se recorre

Medio · Colecciones · apuntesdam.com/subject/programacion/topic/colecciones

Solución (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        List<String> tareas = new ArrayList<>(List.of("x comprar pan", "estudiar Java", "x llamar a Ana", "x pagar luz", "hacer la práctica"));
        tareas.removeIf(t -> t.startsWith("x "));
        System.out.println("Pendientes: " + tareas);
    }
}
```

Un `for-each` recorre la lista con un iterador, y si la lista cambia por otro camino (`tareas.remove`) el iterador lo detecta en el siguiente paso y lanza `ConcurrentModificationException`.

`removeIf` hace el recorrido y el borrado de forma segura en una sola línea. La alternativa clásica es usar el `Iterator` explícitamente y borrar con `it.remove()`.

66. Encuentra el fallo: el contador de palabras olvidadizo

Medio · Colecciones · apuntesdam.com/subject/programacion/topic/colecciones

Solución (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        String frase = new Scanner(System.in).nextLine().toLowerCase();
        Map<String, Integer> contador = new TreeMap<>();
        for (String p : frase.split(" ")) {
            contador.merge(p, 1, Integer::sum);
        }
        contador.forEach((p, n) -> System.out.println(p + " " + n));
    }
}
```

`put(p, 1)` sustituye el valor anterior en lugar de incrementarlo, así que todas las palabras se quedan en 1. `merge(p, 1, Integer::sum)` pone 1 si la palabra no estaba y, si estaba, suma 1 a lo que había.

Es equivalente a `contador.put(p, contador.getOrDefault(p, 0) + 1)`.

67. Agrupar anagramas

Difícil · Colecciones · apuntesdam.com/subject/programacion/topic/colecciones

Solución (java)

```
import java.util.*;

public class Main {
    // Devuelve los grupos de anagramas (2 o más palabras), cada uno ordenado alfabéticamente
    // y los grupos de más a menos palabras (a igualdad, por su primera palabra)
    static List<List<String>> agrupar(List<String> palabras) {
        Map<String, TreeSet<String>> grupos = new HashMap<>();
        for (String p : palabras) {
            String palabra = p.toLowerCase();
            char[] letras = palabra.toCharArray();
            Arrays.sort(letras);
            // Los anagramas tienen las mismas letras ordenadas: esa es la clave del grupo
            grupos.computeIfAbsent(new String(letras), k -> new TreeSet<>()).add(palabra);
        }
        List<List<String>> res = new ArrayList<>();
        for (TreeSet<String> g : grupos.values()) {
            if (g.size() >= 2) res.add(new ArrayList<>(g));
        }
        res.sort(Comparator.comparingInt((List<String> g) -> -g.size()).thenComparing(g -> g.get(0)));
        return res;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<String> palabras = new ArrayList<>();
        while (sc.hasNext()) palabras.add(sc.next());
        List<List<String>> grupos = agrupar(palabras);
        if (grupos.isEmpty()) System.out.println("No hay anagramas");
        for (List<String> g : grupos) System.out.println(g.size() + ": " + String.join(", ", g));
    }
}
```

El truco es encontrar una «firma» común a todos los anagramas: sus letras ordenadas. «amor», «mora» y «roma» dan las tres «amor», así que un mapa de firma → palabras los agrupa en una sola pasada.

Cada grupo es un `TreeSet`, que a la vez elimina las repetidas y mantiene el orden alfabético. Al final se descartan los grupos de una palabra y se ordenan por tamaño (de mayor a menor, con el signo menos) y por la primera palabra.

68. Ruta más corta en el metro

Muy difícil · Colecciones · apuntesdam.com/subject/programacion/topic/colecciones

Solución (java)

```
import java.util.*;

public class Main {
    // Estación → estaciones vecinas (en orden alfabético)
    static Map<String, Set<String>> red = new TreeMap<>();

    static void conectar(String a, String b) {
        red.computeIfAbsent(a, k -> new TreeSet<>()).add(b);
        red.computeIfAbsent(b, k -> new TreeSet<>()).add(a);
    }

    // Camino con menos paradas de origen a destino (incluidos los dos), o null si no hay
    static List<String> ruta(String origen, String destino) {
        Map<String, String> desde = new HashMap<>(); // estación → desde cuál se descubrió
        Deque<String> cola = new ArrayDeque<>();
        desde.put(origen, null);
        cola.add(origen);
        while (!cola.isEmpty()) {
            String actual = cola.poll();
            if (actual.equals(destino)) {
                LinkedList<String> camino = new LinkedList<>();
                for (String e = destino; e != null; e = desde.get(e)) camino.addFirst(e);
                return camino;
            }
            for (String vecina : red.get(actual)) {
                if (!desde.containsKey(vecina)) { // aún no descubierta
                    desde.put(vecina, actual);
                    cola.add(vecina);
                }
            }
        }
        return null;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String linea = sc.nextLine().trim();
            if (linea.isEmpty()) continue;
            String[] t = linea.split(" ");
            if (t[0].equals("LINEA")) {
                for (int i = 2; i < t.length - 1; i++) conectar(t[i], t[i + 1]);
            } else if (t[0].equals("RUTA")) {
                if (!red.containsKey(t[1]) || !red.containsKey(t[2])) {
                    System.out.println("Estación desconocida");
                    continue;
                }
                List<String> r = ruta(t[1], t[2]);
                if (r == null) System.out.println(t[1] + " → " + t[2] + ": sin conexión");
                else System.out.println(String.join(" → ", r) + " (" + (r.size() - 1) + " paradas)");
            }
        }
    }
}
```

La búsqueda en anchura explora por capas: primero todas las estaciones a una parada, después las que están a dos, etc. Por eso, la primera vez que se saca el destino de la cola, se ha llegado por el camino con menos paradas.

El mapa `desde` cumple dos funciones: saber si una estación ya se descubrió (para no meterla dos veces en la cola) y reconstruir el camino al final, siguiendo los «desde» hacia atrás desde el destino y añadiendo cada estación al principio de la lista.

69. Media de notas desde un fichero CSV

Fácil · Manejo de Ficheros · apuntesdam.com/subject/acceso-datos/topic/gestion-ficheros

Solución (java)

```
import java.io.*;
import java.util.*;

public class Main {

    static double leerMedia(File fichero) throws IOException {
        double suma = 0;
        int contador = 0;
        try (BufferedReader br = new BufferedReader(new FileReader(fichero))) {
            String linea;
            while ((linea = br.readLine()) != null) {
                String[] partes = linea.split(";");
                if (partes.length < 2) continue; // líneas vacías o mal formadas
                suma += Double.parseDouble(partes[1].trim());
                contador++;
            }
        }
        return contador == 0 ? 0 : suma / contador;
    }

    public static void main(String[] args) throws IOException {
        // Guarda en notas.csv las líneas recibidas por teclado
        File fichero = new File("notas.csv");
        Scanner sc = new Scanner(System.in);
        try (PrintWriter pw = new PrintWriter(new FileWriter(fichero))) {
            while (sc.hasNextLine()) pw.println(sc.nextLine());
        }

        System.out.printf(Locale.US, "Media: %.2f\n", leerMedia(fichero));
    }
}
```

`BufferedReader.readLine()` devuelve una línea cada vez y `null` al llegar al final, por eso el bucle típico es `while ((linea = br.readLine()) != null)`. Cada línea se parte por el `;` y la segunda parte es la nota.

El `try (...)` con recursos cierra el fichero automáticamente al salir, incluso si salta una excepción: es la forma recomendada de abrir cualquier fichero en Java.

70. Serializar una lista de productos

Medio · Manejo de Ficheros · apuntesdam.com/subject/acceso-datos/topic/gestion-ficheros

Solución (java)

```
import java.io.*;
import java.util.*;

public class Main {

    static class Producto implements Serializable {
        private static final long serialVersionUID = 1L;
        String nombre;
        double precio;
        Producto(String nombre, double precio) { this.nombre = nombre; this.precio = precio; }
    }

    static void guardar(List<Producto> productos, File fichero) throws IOException {
        try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(fichero))) {
            out.writeObject(new ArrayList<>(productos));
        }
    }

    @SuppressWarnings("unchecked")
    static List<Producto> cargar(File fichero) throws IOException, ClassNotFoundException {
        try (ObjectInputStream in = new ObjectInputStream(new FileInputStream(fichero))) {
            return (List<Producto>) in.readObject();
        }
    }

    public static void main(String[] args) throws Exception {
        List<Producto> productos = new ArrayList<>();
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().split(";");
            if (p.length == 2) productos.add(new Producto(p[0], Double.parseDouble(p[1])));
        }

        File fichero = new File("productos.obj");
        guardar(productos, fichero);
        List<Producto> leidos = cargar(fichero);

        double total = 0;
        for (Producto p : leidos) total += p.precio;
        System.out.printf(Locale.US, "%d productos, total %.2f%n", leidos.size(), total);
    }
}
```

`writeObject` serializa la lista entera de una vez: el `ArrayList` y todos los productos que contiene, que deben implementar `Serializable`. Al leer, `readObject` devuelve un `Object` y hay que convertirlo al tipo que se guardó; de ahí el `@SuppressWarnings("unchecked")`.

Se guarda una copia como `ArrayList` porque la lista recibida podría ser de un tipo no serializable (por ejemplo, la de `List.of`). Los dos flujos se abren con `try` con recursos para que el fichero se cierre siempre.

71. Leer un XML con DOM

Medio · Manejo de Ficheros · apuntesdam.com/subject/acceso-datos/topic/gestion-ficheros

Solución (java)

```
import java.io.*;
import java.nio.file.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;

public class Main {

    static void procesar(File fichero) throws Exception {
        int total = 0, disponibles = 0;
        Document doc = DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(fichero);
        NodeList libros = doc.getElementsByTagName("libro");
        total = libros.getLength();
        for (int i = 0; i < libros.getLength(); i++) {
            Element libro = (Element) libros.item(i);
            if (libro.getAttribute("disponible").equals("si")) disponibles++;
        }

        System.out.println("Libros: " + total + ", disponibles: " + disponibles);
    }

    public static void main(String[] args) throws Exception {
        File fichero = new File("biblioteca.xml");
        Files.write(fichero.toPath(), System.in.readAllBytes());
        procesar(fichero);
    }
}
```

DOM carga el XML entero en memoria como un árbol. Se obtiene con la cadena `DocumentBuilderFactory` → `DocumentBuilder` → `parse`, y `getElementsByTagName("libro")` devuelve todos los elementos con ese nombre, estén donde estén.

Cada nodo de la lista se convierte a `Element` para poder leer sus atributos con `getAttribute`, que devuelve una cadena vacía si el atributo no existe (nunca `null`).

72. Acceso aleatorio a registros

Difícil · Manejo de Ficheros · apuntesdam.com/subject/acceso-datos/topic/gestion-ficheros

Solución (java)

```
import java.io.*;
import java.util.*;

public class Main {
    static final int TAM = 4 + 20 + 8; // id + nombre(10 chars) + salario

    static void escribir(RandomAccessFile raf, int id, String nombre, double salario) throws IOException {
        raf.writeInt(id);
        StringBuilder sb = new StringBuilder(nombre);
        sb.setLength(10); // rellena o recorta a 10 caracteres
        raf.writeChars(sb.toString());
        raf.writeDouble(salario);
    }

    static String leerRegistro(RandomAccessFile raf, int n) throws IOException {
        raf.seek((long) (n - 1) * TAM); // salta directamente al registro n
        int id = raf.readInt();
        StringBuilder nombre = new StringBuilder();
        for (int i = 0; i < 10; i++) nombre.append(raf.readChar());
        double salario = raf.readDouble();
        return id + " " + nombre.toString().trim() + " " + salario;
    }

    public static void main(String[] args) throws IOException {
        try (RandomAccessFile raf = new RandomAccessFile("empleados.dat", "rw")) {
            raf.setLength(0);
            escribir(raf, 1, "Ana", 1850.5);
            escribir(raf, 2, "Luis", 2100);
            escribir(raf, 3, "Marta", 2475.25);
            escribir(raf, 4, "Jorge", 1990);

            int n = new Scanner(System.in).nextInt();
            System.out.println(leerRegistro(raf, n));
        }
    }
}
```

Como todos los registros miden lo mismo (32 bytes), el registro n empieza en el byte $(n - 1) * TAM$. `seek` coloca ahí el puntero del fichero sin leer nada de lo anterior: eso es el acceso aleatorio.

Los campos se leen en el mismo orden y con los mismos tipos con que se escribieron: `readInt`, diez `readChar` y `readDouble`. El nombre se rellenó con caracteres nulos hasta 10, y `trim()` los quita porque son menores que el espacio.

73. Un CSV de verdad: comillas, separadores y saltos de línea

Difícil · Manejo de Ficheros · apuntesdam.com/subject/acceso-datos/topic/gestion-ficheros

Solución (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in).useDelimiter("\\\\A");
        String texto = sc.hasNext() ? sc.next().replace("\\r", "") : "";
        List<List<String>> registros = new ArrayList<>();

        List<String> registro = new ArrayList<>();
        StringBuilder campo = new StringBuilder();
        boolean enComillas = false, alInicio = true, entrecomillado = false;
        for (int i = 0; i < texto.length(); i++) {
            char c = texto.charAt(i);
            if (enComillas) {
                if (c != '"') campo.append(c);
                else if (i + 1 < texto.length() && texto.charAt(i + 1) == '"') { campo.append('"'); i++; }
                // "" es una comilla
                else enComillas = false;
                // " cierra
            } else if (c == '"' && alInicio) {
                enComillas = true;
                entrecomillado = true;
                alInicio = false;
            } else if (c == ';' || c == '\\n') {
                registro.add(campo.toString());
                campo.setLength(0);
                alInicio = true;
                if (c == '\\n') {
                    boolean vacio = registro.size() == 1 && registro.get(0).isEmpty() && !entrecomillado;
                    if (!vacio) registros.add(registro); // las líneas vacías se ignoran
                    registro = new ArrayList<>();
                }
                entrecomillado = false;
            } else {
                campo.append(c);
                alInicio = false;
            }
        }
        if (enComillas) {
            System.out.println("Error: comilla sin cerrar en el registro " + registros.size());
            return;
        }
        if (!alInicio || !registro.isEmpty()) { // último registro sin salto de línea
            registro.add(campo.toString());
            registros.add(registro);
        }
        if (registros.isEmpty()) {
            System.out.println("Registros válidos: 0 de 0 · Campos: 0");
            return;
        }

        List<String> cabecera = registros.get(0);
        int validos = 0;
        for (int r = 1; r < registros.size(); r++) {
            List<String> reg = registros.get(r);
            if (reg.size() != cabecera.size()) {
                System.out.println("Error en el registro " + r + ": tiene " + reg.size() + " campos y se esperaban " + cabecera.size());
                continue;
            }
            validos++;
            System.out.println("Registro " + r + ":");
            for (int k = 0; k < reg.size(); k++) {
                String v = reg.get(k);
                System.out.println("  " + cabecera.get(k) + " = " + (v.isEmpty() ? "(vacío)" : v.replace("\\n", "\\n")));
            }
        }
    }
}
```



```
        System.out.println("Registros válidos: " + validos + " de " + (registros.size() - 1) + " . Campos: "
            + cabecera.size());
    }
}
```

Un CSV real no se puede partir con `split`, porque el separador y el salto de línea pueden estar dentro de un campo. La solución es una pequeña máquina de estados que lee carácter a carácter y recuerda si está dentro de comillas: dentro, todo se copia salvo la comilla (que cierra) o la comilla doble `""` (que es una comilla); fuera, `;` cierra el campo y el salto de línea cierra el registro.

La variable `alInicio` distingue la comilla que abre un campo (la primera) de una comilla en mitad de un texto, que es un carácter normal. `entrecomillado` sirve para no confundir una línea vacía con un registro que tiene un único campo `""`.

Si al terminar el texto seguimos dentro de comillas, el fichero está mal formado; si no, el último registro puede no tener salto de línea final y hay que añadirlo a mano.

74. Leer un formato binario propio con versión y CRC

Muy difícil · Manejo de Ficheros · apuntesdam.com/subject/acceso-datos/topic/gestion-ficheros

Solución (java)

```
import java.nio.*;
import java.nio.charset.StandardCharsets;
import java.util.*;
import java.util.zip.CRC32;

public class Main {
    static byte[] desdeHex(String h) {
        byte[] b = new byte[h.length() / 2];
        for (int i = 0; i < b.length; i++) b[i] = (byte) Integer.parseInt(h.substring(2 * i, 2 * i + 2),
            16);
        return b;
    }

    public static void main(String[] args) {
        byte[] datos = desdeHex(new Scanner(System.in).nextLine().trim());
        ByteBuffer buf = ByteBuffer.wrap(datos); // big-endian, como DataInputStream

        if (datos.length < 4 || !new String(datos, 0, 4, StandardCharsets.US_ASCII).equals("ALUM")) {
            System.out.println("Formato desconocido");
            return;
        }
        buf.position(4);
        List<String> lineas = new ArrayList<>();
        int version, n;
        double suma = 0;
        try {
            version = buf.getShort();
            if (version != 1 && version != 2) {
                System.out.println("Versión " + version + " no soportada");
                return;
            }
            n = buf.getInt();
            for (int i = 0; i < n; i++) {
                int id = buf.getInt();
                byte[] nombre = new byte[buf.getShort() & 0xFFFF];
                buf.get(nombre);
                String curso = version == 2 ? " · curso " + buf.get() : "";
                float nota = buf.getFloat();
                suma += nota;
                lineas.add(String.format(Locale.US, "%d · %s%s · %.2f", id, new String(nombre,
                    StandardCharsets.UTF_8), curso, nota));
            }
            int finDatos = buf.position();
            long esperado = buf.getInt() & 0xFFFFFFFFL; // el int del fichero, sin signo
            CRC32 crc = new CRC32();
            crc.update(datos, 0, finDatos);
            if (crc.getValue() != esperado) {
                System.out.printf("CRC incorrecto: esperado %08X, calculado %08X%n", esperado,
                    crc.getValue());
                return;
            }
        } catch (BufferUnderflowException e) { // se intentó leer más allá del final
            System.out.println("Fichero truncado");
            return;
        }
        lineas.forEach(System.out::println);
        System.out.printf(Locale.US, "Registros: %d · media %.2f · versión %d · CRC OK%n", n, n == 0 ? 0 :
            suma / n, version);
    }
}
```

`ByteBuffer` lee tipos binarios en big-endian, igual que `DataInputStream`: `getShort` lee 2 bytes, `getInt` 4 y `getFloat` 4. El nombre tiene longitud variable, así que primero se lee su longitud y luego exactamente esos bytes, que se decodifican como UTF-8 (por eso «lñaki» ocupa más bytes que letras).

Para detectar un fichero truncado no hace falta comprobar el tamaño antes de cada lectura: si se intenta leer más allá del final, `ByteBuffer` lanza `BufferUnderflowException`, y basta con capturarla una vez alrededor de toda la lectura.

El CRC se calcula sobre los bytes desde el principio hasta donde empieza el CRC guardado. Como el `int` de Java tiene signo y el CRC32 no, se convierte con `& 0xFFFFFFFFL` antes de comparar.

75. Sentencias preparadas frente a concatenación

Medio · Conectores JDBC · apuntesdam.com/subject/acceso-datos/topic/jdbc

Solución (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String plantilla = sc.nextLine();
        List<String[]> params = new ArrayList<>();          // {tipo, valor}
        while (sc.hasNextLine()) {
            String l = sc.nextLine();
            if (l.isBlank()) continue;
            int dos = l.indexOf(':');
            params.add(new String[] { l.substring(0, dos), l.substring(dos + 1) });
        }

        // Posiciones de los ? que no están dentro de un literal '...'
        List<Integer> marcadores = new ArrayList<>();
        boolean enLiteral = false;
        for (int i = 0; i < plantilla.length(); i++) {
            char c = plantilla.charAt(i);
            if (c == '\\') enLiteral = !enLiteral;
            else if (c == '?' && !enLiteral) marcadores.add(i);
        }
        System.out.println("Parámetros: " + marcadores.size());
        if (marcadores.size() != params.size()) {
            System.out.println("Error: la sentencia tiene " + marcadores.size() + " parámetros y se han dado " + params.size());
            return;
        }
        for (int k = 0; k < params.size(); k++) {
            String tipo = params.get(k)[0], valor = params.get(k)[1];
            if (tipo.equals("i") && !valor.matches("-?\\d+")) {
                System.out.println("Error: el parámetro " + (k + 1) + " no es un entero: " + valor);
                return;
            }
            if (tipo.equals("d") && !valor.matches("-?\\d+(\\.\\d+)?")) {
                System.out.println("Error: el parámetro " + (k + 1) + " no es un decimal: " + valor);
                return;
            }
        }

        StringBuilder concatenada = new StringBuilder(), preparada = new StringBuilder();
        boolean inyeccion = false;
        int desde = 0;
        for (int k = 0; k < marcadores.size(); k++) {
            String trozo = plantilla.substring(desde, marcadores.get(k));
            String tipo = params.get(k)[0], valor = params.get(k)[1];
            String literal = switch (tipo) {
                case "b" -> valor.equalsIgnoreCase("true") ? "TRUE" : "FALSE";
                case "f" -> "DATE '" + valor + "'";
                case "n" -> "NULL";
                case "s" -> null;
                default -> valor; // enteros y decimales, tal cual
            };
            if (tipo.equals("s")) {
                if (valor.contains("'")) inyeccion = true;
                concatenada.append(trozo).append('\\').append(valor).append('\\');
                preparada.append(trozo).append('\\').append(valor.replace("'", "''")).append('\\');
            } else {
                concatenada.append(trozo).append(literal);
                preparada.append(trozo).append(literal);
            }
            desde = marcadores.get(k) + 1;
        }
        concatenada.append(plantilla.substring(desde));
        preparada.append(plantilla.substring(desde));
        System.out.println("Concatenando: " + concatenada);
        System.out.println("Preparada: " + preparada);
        System.out.println(inyeccion);
    }
}
```

```
        ? "¡Inyección! Con la concatenación el texto del usuario cambia la estructura de la  
          consulta"  
        : "Las dos versiones son equivalentes con estos datos");  
    }  
}
```

Para localizar los marcadores se recorre la plantilla recordando si se está dentro de un literal entre comillas simples: cada `'` cambia el estado, y solo cuentan los `?` de fuera. Así el `?` de `'¿Qué?'` no se toma por un parámetro.

Las dos versiones se construyen a la vez, copiando los trozos de plantilla entre marcadores y poniendo en su lugar cada valor. La única diferencia está en los textos: al concatenar se pegan tal cual, y con la sentencia preparada la comilla del dato se duplica, de modo que nunca puede cerrar el literal.

Con `' OR '1'='1` se ve el problema: al concatenar, la comilla del usuario cierra el literal y el resto se convierte en una condición que siempre es cierta. Por eso los datos del usuario se pasan siempre con `PreparedStatement` y `setString`.

76. Un pool de conexiones por dentro

Difícil · Conectores JDBC · apuntesdam.com/subject/acceso-datos/topic/jdbc

Solución (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] conf = sc.nextLine().trim().split(" "); // «max M espera W»
        int max = Integer.parseInt(conf[1]), espera = Integer.parseInt(conf[3]);
        List<String[]> eventos = new ArrayList<>(); // {tiempo, accion, cliente}
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (!l.isEmpty()) eventos.add(l.split(" "));
        }

        Deque<Integer> libres = new ArrayDeque<>(); // la primera es la que lleva más tiempo libre
        Map<String, Integer> enUso = new HashMap<>(); // cliente → conexión
        Deque<Object[]> cola = new ArrayDeque<>(); // {cliente, instante en que empezó a esperar}
        int creadas = 0, reutilizadas = 0, esperas = 0, timeouts = 0, servidas = 0;
        long esperaTotal = 0;

        for (int e = 0; e <= eventos.size(); e++) {
            long t = e < eventos.size() ? Long.parseLong(eventos.get(e)[0]) : Long.MAX_VALUE;
            // Antes de cada evento (y al final), se van quienes han superado la espera
            while (!cola.isEmpty() && (long) cola.peekFirst()[1] + espera < t) {
                Object[] w = cola.pollFirst();
                System.out.println(((long) w[1] + espera) + ": " + w[0] + " agota la espera (timeout)");
                timeouts++;
            }
            if (e == eventos.size()) break;
            String accion = eventos.get(e)[1], cliente = eventos.get(e)[2];

            if (accion.equals("pedir")) {
                if (!libres.isEmpty()) {
                    int k = libres.pollFirst();
                    enUso.put(cliente, k);
                    reutilizadas++;
                    System.out.println(t + ": " + cliente + " recibe la conexión #" + k + " (reutilizada)");
                } else if (creadas < max) {
                    int k = ++creadas;
                    enUso.put(cliente, k);
                    System.out.println(t + ": " + cliente + " recibe la conexión #" + k + " (nueva)");
                } else {
                    cola.addLast(new Object[] { cliente, t });
                    esperas++;
                    System.out.println(t + ": " + cliente + " espera (pool lleno)");
                }
            } else {
                Integer k = enUso.remove(cliente);
                if (k == null) {
                    System.out.println(t + ": Error: " + cliente + " no tiene conexión");
                } else if (!cola.isEmpty()) {
                    Object[] w = cola.pollFirst();
                    long esperado = t - (long) w[1];
                    enUso.put((String) w[0], k);
                    servidas++;
                    esperaTotal += esperado;
                    System.out.println(t + ": " + cliente + " devuelve #" + k + " → pasa a " + w[0] + " (esperó " + esperado + " ms)");
                } else {
                    libres.addLast(k);
                    System.out.println(t + ": " + cliente + " devuelve #" + k + " (queda libre)");
                }
            }
        }

        System.out.println("Conexiones creadas: " + creadas + " · Reutilizadas: " + reutilizadas + " · Esperas: " + esperas);
    }
}
```

```

        + " · Timeouts: " + timeouts + " · Espera media: " + (servidas == 0 ? 0 : esperaTotal /
servidas) + " ms");
System.out.println("Al final: " + creadas + " abiertas, " + libres.size() + " libres, " +
enUso.size() + " en uso");
    }
}

```

El pool se modela con tres estructuras: una cola de conexiones libres (`ArrayDeque` : se devuelven por el final y se entregan por el principio, así sale la que lleva más tiempo libre), un mapa de quién tiene cada conexión y una cola FIFO de clientes esperando con el instante en que empezaron.

Los tiempos límite no son eventos de la entrada, así que se comprueban antes de procesar cada evento: mientras el primero de la cola lleve esperando más de W , se va. El bucle da una vuelta extra al final con un instante «infinito» para agotar los que queden, sin repetir el código.

Al devolver una conexión, si alguien espera se le pasa directamente, sin que llegue a quedar libre: así funciona un pool real, y por eso no cuenta como reutilizada.

77. Transacciones con puntos de guardado y bloqueos

Muy difícil · Conectores JDBC · apuntesdam.com/subject/acceso-datos/topic/jdbc

Solución (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, Integer> confirmado = new LinkedHashMap<>(); // lo que ve cualquiera
        for (String par : sc.nextLine().trim().split(" ")) {
            if (!par.contains("=")) continue; // «cuentas A1=100 A2=50 ...»
            String[] kv = par.split("=");
            confirmado.put(kv[0], Integer.parseInt(kv[1]));
        }
        // Por sesión: cambios pendientes (null si no hay transacción) y puntos de guardado
        Map<String, Map<String, Integer>> pendientes = new HashMap<>();
        Map<String, LinkedHashMap<String, Map<String, Integer>>> puntos = new HashMap<>();

        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");
            if (p.length < 2) continue;
            String s = p[0], otra = s.equals("A") ? "B" : "A";
            Map<String, Integer> pend = pendientes.get(s);
            String orden = p[1].equals("ROLLBACK") && p.length > 3 ? "ROLLBACK TO" : p[1];
            switch (orden) {
                case "BEGIN" -> {
                    if (pend != null) System.out.println(s + ": ya hay una transacción abierta");
                    else {
                        pendientes.put(s, new LinkedHashMap<>());
                        puntos.put(s, new LinkedHashMap<>());
                        System.out.println(s + ": BEGIN");
                    }
                }
                case "SELECT" -> {
                    String c = p[2];
                    int v = pend != null && pend.containsKey(c) ? pend.get(c) : confirmado.get(c);
                    System.out.println(s + " ve " + c + " = " + v);
                }
                case "UPDATE" -> {
                    String c = p[2];
                    Map<String, Integer> deLaOtra = pendientes.get(otra);
                    if (deLaOtra != null && deLaOtra.containsKey(c)) {
                        System.out.println(s + ": UPDATE " + c + " bloqueado por " + otra + " (se cancela la sentencia)");
                        break;
                    }
                    int actual = pend != null && pend.containsKey(c) ? pend.get(c) : confirmado.get(c);
                    int nuevo = actual + Integer.parseInt(p[3]);
                    if (nuevo < 0) System.out.println(s + ": UPDATE " + c + " rechazado: el saldo quedaría en " + nuevo);
                    else if (pend == null) { // autocommit
                        confirmado.put(c, nuevo);
                        System.out.println(s + ": " + c + " = " + nuevo + " (confirmado)");
                    } else {
                        pend.put(c, nuevo);
                        System.out.println(s + ": " + c + " = " + nuevo);
                    }
                }
                case "SAVEPOINT" -> {
                    if (pend == null) { System.out.println(s + ": no hay transacción"); break; }
                    puntos.get(s).remove(p[2]);
                    puntos.get(s).put(p[2], new LinkedHashMap<>(pend)); // foto de los cambios pendientes
                    System.out.println(s + ": SAVEPOINT " + p[2]);
                }
                case "ROLLBACK TO" -> {
                    if (pend == null) { System.out.println(s + ": no hay transacción"); break; }
                    LinkedHashMap<String, Map<String, Integer>> pts = puntos.get(s);
                    String nombre = p[3];
                    if (!pts.containsKey(nombre)) {
                        System.out.println(s + ": no existe el punto de guardado " + nombre);
                        break;
                    }
                }
            }
        }
    }
}
```



```

    }
    pendientes.put(s, new LinkedHashMap<>(pts.get(nombre)));
    // se borran los puntos posteriores
    boolean despues = false;
    for (Iterator<String> it = pts.keySet().iterator(); it.hasNext(); ) {
        String k = it.next();
        if (despues) it.remove();
        if (k.equals(nombre)) despues = true;
    }
    System.out.println(s + ": vuelta a " + nombre);
}
case "COMMIT" -> {
    if (pend == null) { System.out.println(s + ": no hay transacción"); break; }
    confirmado.putAll(pend);
    System.out.println(s + ": COMMIT (" + pend.size() + " cambios)");
    pendientes.put(s, null);
}
case "ROLLBACK" -> {
    if (pend == null) { System.out.println(s + ": no hay transacción"); break; }
    System.out.println(s + ": ROLLBACK (" + pend.size() + " cambios descartados)");
    pendientes.put(s, null);
}
default -> { }
}
}

for (String s : List.of("A", "B")) {
    if (pendientes.get(s) != null) System.out.println(s + " tenía una transacción abierta: se deshace al cerrar la conexión");
}
StringBuilder fin = new StringBuilder("Confirmado:");
confirmado.forEach((c, v) -> fin.append(' ').append(c).append('=').append(v));
System.out.println(fin);
}
}

```

La idea clave del aislamiento READ COMMITTED es que cada sesión trabaja sobre su propio mapa de cambios pendientes y nadie más los ve hasta el COMMIT. Por eso leer es «el valor pendiente propio si existe y, si no, el confirmado», y confirmar es copiar los pendientes sobre el mapa común con `putAll`.

Un punto de guardado es una foto (una copia) de los cambios pendientes en ese momento. Volver a él es restaurar la copia y borrar los puntos creados después; usar un `LinkedHashMap` mantiene su orden de creación, y se borran con un `Iterator` para poder eliminar mientras se recorre.

El bloqueo se simula comprobando si la otra sesión tiene esa cuenta entre sus pendientes. Sin transacción (`pend == null`), cada UPDATE se confirma en el acto: es el modo autocommit de JDBC.

78. El problema N+1 y las estrategias de carga

Medio · Mapeo Objeto-Relacional (Hibernate/JPA) · apuntesdam.com/subject/acceso-datos/topic/hibernate

Solución (java)

```
import java.util.*;

public class Main {
    record Autor(int id, String nombre) { }
    record Libro(int id, int autorId, String titulo) { }

    static int consultas = 0;

    static void sql(String s) {
        System.out.println("SQL: " + s);
        consultas++;
    }

    static void mostrar(Autor a, List<Libro> libros) {
        List<String> titulos = new ArrayList<>();
        for (Libro l : libros) if (l.autorId() == a.id()) titulos.add(l.titulo());
        int n = titulos.size();
        System.out.println(a.nombre() + ": " + n + (n == 1 ? " libro" : " libros") + (n == 0 ? "" : " (" +
            String.join(", ", titulos) + ")"));
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<Autor> autores = new ArrayList<>();
        List<Libro> libros = new ArrayList<>();
        String estrategia = "LAZY";
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (l.startsWith("autor ")) {
                String[] p = l.split(" ", 3);
                autores.add(new Autor(Integer.parseInt(p[1]), p[2]));
            } else if (l.startsWith("libro ")) {
                String[] p = l.split(" ", 4);
                libros.add(new Libro(Integer.parseInt(p[1]), Integer.parseInt(p[2]), p[3]));
            } else if (l.startsWith("estrategia ")) {
                estrategia = l.substring(11).trim();
            }
        }
        autores.sort(Comparator.comparingInt(Autor::id));
        libros.sort(Comparator.comparingInt(Libro::id));

        if (estrategia.equals("JOIN FETCH")) {
            sql("select * from autor a left join libro l on l.autor_id = a.id order by a.id");
            for (Autor a : autores) mostrar(a, libros);
        } else {
            sql("select * from autor order by id");
            Set<Integer> cargados = new HashSet<>(); // autores con sus libros ya cargados
            if (estrategia.equals("EAGER")) {
                for (Autor a : autores) {
                    sql("select * from libro where autor_id = " + a.id());
                    cargados.add(a.id());
                }
            }
            for (int i = 0; i < autores.size(); i++) {
                Autor a = autores.get(i);
                if (!cargados.contains(a.id())) { // a.getLibros() en una colección sin cargar
                    if (estrategia.equals("SUBSELECT")) {
                        sql("select * from libro where autor_id in (select id from autor)");
                        for (Autor x : autores) cargados.add(x.id());
                    } else if (estrategia.startsWith("BATCH")) {
                        int k = Integer.parseInt(estrategia.substring(6).trim());
                        List<String> ids = new ArrayList<>();
                        for (int j = i; j < autores.size() && ids.size() < k; j++) {
                            if (cargados.add(autores.get(j).id()))
                                ids.add(String.valueOf(autores.get(j).id()));
                        }
                        sql("select * from libro where autor_id in (" + String.join(", ", ids) + ")");
                    } else {

```

```

        sql("select * from libro where autor_id = " + a.id());
        cargados.add(a.id());
    }
}
mostrar(a, libros);
}
}
System.out.println("Consultas: " + consultas + " (estrategia " + estrategia + ")");
}
}

```

Todas las estrategias comparten la misma estructura: se recorren los autores y, antes de usar los libros de uno, se comprueba si su colección ya está cargada (un `Set` de ids). Lo único que cambia es qué consulta se lanza cuando no lo está y cuántos autores deja cargados: uno (LAZY), un lote de k (BATCH) o todos (SUBSELECT).

EAGER carga todo justo después de la primera consulta, pero con una consulta por autor: sigue siendo N+1, solo que antes. JOIN FETCH es la única que lo trae todo en una sola consulta.

El método `sql` imprime y cuenta a la vez, así el total de consultas no puede desajustarse. `cargados.add(id)` devuelve `false` si el autor ya estaba, lo que permite construir el lote saltando los ya cargados en la misma línea.

79. Los estados de una entidad JPA

Difícil · Mapeo Objeto-Relacional (Hibernate/JPA) · apuntesdam.com/subject/acceso-datos/topic/hibernate

Solución (java)

```
import java.util.*;

public class Main {
    static class Entidad {
        Long id;
        String tabla;
        Map<String, String> campos = new LinkedHashMap<>();
        String estado = "NUEVA"; // NUEVA, GESTIONADA, SEPARADA, ELIMINADA
        Map<String, String> instantanea; // copia al pasar a gestionada o tras el último flush
    }

    static Map<String, String> leerCampos(String texto) {
        Map<String, String> m = new LinkedHashMap<>();
        for (String par : texto.split(",")) {
            int i = par.indexOf('=');
            m.put(par.substring(0, i), par.substring(i + 1));
        }
        return m;
    }

    static Entidad gestionada(List<Entidad> contexto, String tabla, long id) {
        for (Entidad e : contexto) if (e.tabla.equals(tabla) && e.id == id && e.estado.equals("GESTIONADA"))
            return e;
        return null;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, Entidad> variables = new HashMap<>();
        Map<Long, Map<String, String>> baseDatos = new TreeMap<>();
        List<Entidad> contexto = new ArrayList<>(); // contexto de persistencia (en orden de entrada)
        long[] siguienteId = { 1 };
        while (sc.hasNextLine()) {
            String linea = sc.nextLine().trim();
            if (linea.isEmpty()) continue;
            String[] p = linea.split(" ", 3);
            String v = p.length > 1 ? p[1] : "";
            Entidad e = variables.get(v);
            switch (p[0]) {
                case "new" -> {
                    String[] t = p[2].split(" ", 2);
                    Entidad n = new Entidad();
                    n.tabla = t[0].toLowerCase();
                    n.campos = leerCampos(t[1]);
                    variables.put(v, n);
                    System.out.println(v + ": NUEVA (transient)");
                }
                case "persist" -> {
                    switch (e.estado) {
                        case "GESTIONADA" -> System.out.println(v + " ya está gestionada: no se hace nada");
                        case "SEPARADA" -> System.out.println("PersistentObjectException: " + v + " está separada (usa merge)");
                        default -> {
                            e.id = siguienteId[0]++;
                            List<String> valores = new ArrayList<>();
                            for (String x : e.campos.values()) valores.add("'" + x + "'");
                            System.out.println("SQL: insert into " + e.tabla + " (" + String.join(", ", e.campos.keySet()) + ") values (" + String.join(", ", valores) + ") → id " + e.id);
                            baseDatos.put(e.id, new LinkedHashMap<>(e.campos));
                            e.estado = "GESTIONADA";
                            e.instantanea = new LinkedHashMap<>(e.campos);
                            if (!contexto.contains(e)) contexto.add(e);
                        }
                    }
                }
                case "set" -> {
                    e.campos.putAll(leerCampos(p[2]));
                }
            }
        }
    }
}
```

```

        if (!e.estado.equals("GESTIONADA")) {
            System.out.println(v + " (" + e.estado.toLowerCase() + "): el cambio solo está en memoria");
        }
    }
}
case "find" -> {
    String[] t = p[2].split(" ");
    String tabla = t[0].toLowerCase();
    long id = Long.parseLong(t[1]);
    Entidad g = gestionada(contexto, tabla, id);
    if (g != null) { // caché de primer nivel
        String otra = new TreeMap<>(variables).entrySet().stream()
            .filter(x -> x.getValue() == g && !x.getKey().equals(v))
            .map(Map.Entry::getKey).findFirst().orElse(v);
        variables.put(v, g);
        System.out.println(v + " es la misma instancia que " + otra + " (caché de primer nivel, sin SQL)");
        break;
    }
    System.out.println("SQL: select * from " + tabla + " where id = " + id);
    Map<String, String> fila = baseDatos.get(id);
    if (fila == null) {
        variables.remove(v);
        System.out.println(v + " = null");
        break;
    }
    Entidad n = new Entidad();
    n.id = id;
    n.tabla = tabla;
    n.campos = new LinkedHashMap<>(fila);
    n.estado = "GESTIONADA";
    n.instantanea = new LinkedHashMap<>(fila);
    contexto.add(n);
    variables.put(v, n);
    System.out.println(v + ": GESTIONADA " + n.campos);
}
case "detach" -> {
    contexto.remove(e);
    e.estado = "SEPARADA";
    System.out.println(v + ": SEPARADA");
}
case "merge" -> {
    String w = p[2].trim();
    Entidad origen = variables.get(w);
    Entidad g = gestionada(contexto, origen.tabla, origen.id);
    if (g == null) { // se carga la fila en el contexto
        System.out.println("SQL: select * from " + origen.tabla + " where id = " + origen.id);
        g = new Entidad();
        g.id = origen.id;
        g.tabla = origen.tabla;
        g.campos = new LinkedHashMap<>(baseDatos.get(origen.id));
        g.estado = "GESTIONADA";
        g.instantanea = new LinkedHashMap<>(g.campos);
        contexto.add(g);
    }
    g.campos.putAll(origen.campos); // se copia el estado; origen no cambia
    variables.put(v, g);
    System.out.println(v + ": copia gestionada de " + w + " (" + w + " sigue separada)");
}
case "remove" -> {
    if (!e.estado.equals("GESTIONADA")) {
        System.out.println("IllegalArgumentException: " + v + " no está gestionada");
    } else {
        e.estado = "ELIMINADA";
        System.out.println(v + ": ELIMINADA (se borrará en el flush)");
    }
}
case "flush", "commit" -> {
    int n = 0;
    for (Entidad x : new ArrayList<>(contexto)) {
        if (x.estado.equals("ELIMINADA")) {
            System.out.println("SQL: delete from " + x.tabla + " where id = " + x.id);
            baseDatos.remove(x.id);
        }
    }
}

```

```

        contexto.remove(x);
        n++;
    } else {
        List<String> cambios = new ArrayList<>();
        x.campos.forEach((k, val) -> {
            if (!val.equals(x.instantanea.get(k))) cambios.add(k + " = '" + val + "'");
        });
        if (cambios.isEmpty()) continue; // dirty checking: sin cambios, sin SQL
        System.out.println("SQL: update " + x.tabla + " set " + String.join(", ",
            cambios) + " where id = " + x.id);
        baseDatos.put(x.id, new LinkedHashMap<>(x.campos));
        x.instantanea = new LinkedHashMap<>(x.campos);
        n++;
    }
}
String sentencias = n + (n == 1 ? " sentencia" : " sentencias");
System.out.println(p[0].equals("flush") ? "flush: " + sentencias : "COMMIT (" +
    sentencias + " en el flush)");
}
case "close" -> {
    for (Entidad x : contexto) x.estado = "SEPARADA";
    System.out.println("Contexto cerrado: " + contexto.size() + " entidades pasan a
        SEPARADA");
    contexto.clear();
}
case "estado" -> {
    if (e == null) System.out.println(v + " = null");
    else System.out.println(v + ": " + e.estado + (e.id != null ? " id=" + e.id : "") + " "
        + e.campos);
}
default -> { }
}
}
}
}

```

El contexto de persistencia es una lista de las entidades gestionadas, y cada una guarda una instantánea de sus campos tal como están en la base de datos. En el flush se compara cada entidad con su instantánea (dirty checking) y solo se lanza un UPDATE con los campos que han cambiado; si nada cambió, no hay SQL. Por eso cambiar una nota a 8 y otra vez a 6 no genera ninguna sentencia.

`find` primero busca en el contexto: si la fila ya está gestionada devuelve la misma instancia (la caché de primer nivel), y solo si no está lanza el `SELECT`. Para elegir el nombre de la otra variable se recorren las variables ordenadas con un `TreeMap` y se comparan por identidad (`==`), porque lo que importa es si es el mismo objeto.

`merge` nunca convierte el objeto separado en gestionado: busca (o carga) la instancia gestionada con ese id y le copia los campos. Por eso, después, `remove` del objeto separado falla y el de la copia funciona.

80. Arrays y tipos compuestos de PostgreSQL

Medio · BD Objeto-Relacionales y Orientadas a Objetos · apuntesdam.com/subject/acceso-datos/topic/bd-objeto-

```

import java.util.*;

public class Main {
    static String json(List<String> valores) {
        List<String> partes = new ArrayList<>();
        for (String v : valores)
            partes.add(v == null ? "null" : "\"" + v.replace("\\", "\\\\").replace("'", "\\'") + "\"");
        return "[" + String.join(", ", partes) + "]";
    }

    static List<String> leerArray(String texto) {
        List<String> res = new ArrayList<>();
        String s = texto.substring(1, texto.length() - 1); // sin las llaves
        if (s.isBlank()) return res;
        int i = 0;
        while (true) {
            while (i < s.length() && s.charAt(i) == ' ') i++;
            if (i < s.length() && s.charAt(i) == '"') { // elemento entre comillas
                StringBuilder sb = new StringBuilder();
                i++;
                while (i < s.length() && s.charAt(i) != '"') {
                    if (s.charAt(i) == '\\') i++; // \ escapa el siguiente carácter
                    sb.append(s.charAt(i++));
                }
                res.add(sb.toString());
                while (i < s.length() && s.charAt(i) != ',') i++;
            } else { // sin comillas
                int fin = s.indexOf(',', i);
                if (fin < 0) fin = s.length();
                String v = s.substring(i, fin).trim();
                res.add(v.equalsIgnoreCase("NULL") ? null : v);
                i = fin;
            }
            if (i >= s.length()) break;
            i++; // la coma
        }
        return res;
    }

    static List<String> leerCompuesto(String texto) {
        List<String> res = new ArrayList<>();
        String s = texto.substring(1, texto.length() - 1); // sin los paréntesis
        int i = 0;
        while (true) {
            if (i < s.length() && s.charAt(i) == '"') {
                StringBuilder sb = new StringBuilder();
                i++;
                while (i < s.length()) {
                    char c = s.charAt(i);
                    if (c == '"' && i + 1 < s.length() && s.charAt(i + 1) == '"') { sb.append('"'); i += 2; }
                    else if (c == '"') { i++; break; }
                    else if (c == '\\' && i + 1 < s.length()) { sb.append(s.charAt(i + 1)); i += 2; }
                    else { sb.append(c); i++; }
                }
                res.add(sb.toString()); // "" es una cadena vacía, no nulo
                while (i < s.length() && s.charAt(i) != ',') i++;
            } else {
                int fin = s.indexOf(',', i);
                if (fin < 0) fin = s.length();
                String v = s.substring(i, fin);
                res.add(v.isEmpty() ? null : v); // vacío sin comillas = nulo
                i = fin;
            }
            if (i >= s.length()) break;
            i++;
        }
        return res;
    }
}

```

```

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    while (sc.hasNextLine()) {
        String l = sc.nextLine();
        if (l.isBlank()) continue;
        String tipo = l.substring(0, l.indexOf(' ')), texto = l.substring(l.indexOf(' ') + 1).trim();
        List<String> v = tipo.equals("array") ? leerArray(texto) : leerCompuesto(texto);
        System.out.println(tipo + " " + json(v) + " " + v.size() + (v.size() == 1 ? " valor" : " valores")
            + " " + nulos: " + v.stream().filter(Objects::isNull).count());
    }
}

```

Los dos analizadores siguen el mismo esquema: leer un valor, saltar hasta la coma y repetir hasta el final. Cada valor puede ir entre comillas (y entonces se lee carácter a carácter atendiendo a los escapes) o sin ellas (y entonces llega hasta la siguiente coma).

La diferencia está en cómo se representa el nulo: en un array es la palabra NULL sin comillas; en un tipo compuesto es un campo vacío. Por eso "NULL" entre comillas es un texto y "" es una cadena vacía: las comillas indican siempre que hay un valor.

Fíjate en la coma final de un compuesto: tras saltarla, la siguiente vuelta encuentra un campo vacío y lo añade como nulo, que es justo lo que significa.

81. Profundidad de activación en una base de objetos

Difícil · BD Objeto-Relacionales y Orientadas a Objetos · apuntesdam.com/subject/acceso-datos/topic/bd-objeto-


```

import java.util.*;

public class Main {
    static class Objeto {
        String id, clase;
        Map<String, String> datos = new LinkedHashMap<>(); // campos simples
        Map<String, String> referencias = new LinkedHashMap<>(); // campo → id del objeto apuntado
    }

    static Map<String, Objeto> almacen = new LinkedHashMap<>();
    static Set<String> activados = new HashSet<>();

    static void activar(String id, int profundidad, String sangria, String prefijo) {
        Objeto o = almacen.get(id);
        if (o == null) {
            System.out.println(sangria + prefijo + "null (" + id + " no existe)");
            return;
        }
        if (profundidad <= 0) { // se queda como referencia sin rellenar
            System.out.println(sangria + prefijo + o.clase + " " + o.id + " (sin activar)");
            return;
        }
        activados.add(id);
        System.out.println(sangria + prefijo + o.clase + " " + o.id + " " + o.datos);
        for (Map.Entry<String, String> r : o.referencias.entrySet()) {
            activar(r.getValue(), profundidad - 1, sangria + " ", r.getKey() + " → ");
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");
            if (p[0].equals("objeto")) { // objeto ID Clase campo=valor ref=@ID ...
                Objeto o = new Objeto();
                o.id = p[1]; o.clase = p[2];
                for (int i = 3; i < p.length; i++) {
                    String[] kv = p[i].split("=");
                    if (kv[1].startsWith("@")) o.referencias.put(kv[0], kv[1].substring(1));
                    else o.datos.put(kv[0], kv[1]);
                }
                almacen.put(o.id, o);
            } else if (p[0].equals("activar")) { // activar ID profundidad
                activados.clear();
                System.out.println("== activar " + p[1] + " con profundidad " + p[2]);
                activar(p[1], Integer.parseInt(p[2]), "", "");
                System.out.println("Objetos leídos del disco: " + activados.size());
            }
        }
    }
}

```

Es un recorrido recursivo del grafo con un límite: cada llamada activa el objeto y llama a **activar** con cada referencia y una profundidad menos. Cuando la profundidad llega a 0 se para, y por eso los ciclos ($P1 \rightarrow D1 \rightarrow P1 \dots$) no producen una recursión infinita: el límite de profundidad hace de caso base.

La sangría y el prefijo se pasan como parámetros y crecen en cada nivel, así cada línea sabe dónde dibujarse sin variables globales. El conjunto **activados** cuenta los objetos distintos: si un objeto aparece dos veces en el árbol, solo se lee una vez del disco.

o.datos se imprime directamente porque el **toString** de un mapa ya tiene el formato **{nombre=Ana}**.

82. Herencia de tablas en PostgreSQL

Muy difícil · BD Objeto-Relacionales y Orientadas a Objetos · apuntesdam.com/subject/acceso-datos/topic/bd-objeto-

```

import java.util.*;

public class Main {
    static class Tabla {
        String nombre, padre, unico;
        List<String> columnas = new ArrayList<>();
        List<Map<String, String>> filas = new ArrayList<>();
        List<String> hijas = new ArrayList<>();
    }

    static Map<String, Tabla> tablas = new LinkedHashMap<>();

    // Añade a salida las filas de t (y, si no es ONLY, las de sus hijas) con las columnas de la tabla
    // consultada
    static void recoger(Tabla consultada, Tabla t, boolean solo, List<String> salida) {
        for (Map<String, String> fila : t.filas) {
            List<String> valores = new ArrayList<>();
            for (String c : consultada.columnas) valores.add(fila.getOrDefault(c, "NULL"));
            salida.add(" " + String.join(" | ", valores) + (t == consultada ? "" : " [" + t.nombre +
                "]"));
        }
        if (solo) return;
        for (String h : t.hijas) recoger(consultada, tablas.get(h), false, salida);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (l.isEmpty()) continue;
            String[] p = l.split(" ");
            switch (p[0]) {
                case "tabla" -> {
                    Tabla t = new Tabla();
                    t.nombre = p[1];
                    List<String> propias = new ArrayList<>();
                    for (int i = 2; i < p.length; i++) {
                        String[] kv = p[i].split("=", 2);
                        switch (kv[0]) {
                            case "columnas" -> propias.addAll(List.of(kv[1].split(",")));
                            case "hereda" -> t.padre = kv[1];
                            case "unico" -> t.unico = kv[1];
                        }
                    }
                    if (t.padre != null) { // las heredadas van primero
                        t.columnas.addAll(tablas.get(t.padre).columnas);
                        tablas.get(t.padre).hijas.add(t.nombre);
                    }
                    t.columnas.addAll(propias);
                    tablas.put(t.nombre, t);
                    System.out.println("CREATE TABLE " + t.nombre + " (" + String.join(", ", t.columnas) +
                        ")
                        + (t.padre != null ? " INHERITS (" + t.padre + ")" : "")
                        + (t.unico != null ? " UNIQUE (" + t.unico + ")" : ""));
                }
                case "insertar" -> {
                    Tabla t = tablas.get(p[1]);
                    Map<String, String> fila = new LinkedHashMap<>();
                    String error = null;
                    for (String par : l.split(" ", 3)[2].split(",")) {
                        String[] kv = par.split("=", 2);
                        if (!t.columnas.contains(kv[0]) && error == null) error = "ERROR: la columna " +
                            kv[0] + " no existe en " + t.nombre;
                        fila.put(kv[0], kv[1]);
                    }
                    if (error == null && t.unico != null && fila.get(t.unico) != null) {
                        String v = fila.get(t.unico);
                        // UNIQUE solo mira las filas de esta tabla, no las de sus hijas ni las del padre
                        for (Map<String, String> f : t.filas) {

```

```

        if (v.equals(f.get(t.unico))) { error = "ERROR: valor duplicado " + t.unico +
            "=" + v + " en " + t.nombre; break; }
    }
}
if (error != null) System.out.println(error);
else {
    t.filas.add(fila);
    System.out.println("INSERT en " + t.nombre);
}
}
case "consultar" -> {
    boolean solo = p[1].equals("solo");
    Tabla t = tablas.get(solo ? p[2] : p[1]);
    System.out.println("SELECT * FROM " + (solo ? "ONLY " : "") + t.nombre + " (" +
        String.join(" | ", t.columnas) + ")");
    List<String> salida = new ArrayList<>();
    recoger(t, t, solo, salida);
    salida.forEach(System.out::println);
    System.out.println(" " + salida.size() + (salida.size() == 1 ? " fila" : " filas"));
}
default -> { }
}
}
}
}
}

```

Cada tabla guarda sus columnas completas (las del padre primero) y la lista de sus hijas. Así, consultar una tabla es mostrar sus filas y, si no es ONLY, llamar recursivamente a `recoger` con cada hija, que a su vez hará lo mismo con las suyas: un recorrido en profundidad del árbol de herencia.

Al recoger se usan siempre las columnas de la tabla consultada, no las de la tabla de la fila: por eso una fila de `delegado` vista desde `persona` solo muestra dni y nombre, y una columna que la fila no tiene sale como NULL con `getOrDefault`.

La comprobación de UNIQUE recorre solo `t.filas`, que es justo el comportamiento real (y sorprendente) de PostgreSQL: el mismo dni puede estar en `persona` y en `alumno`, y la consulta al padre los mostrará duplicados.

83. Propiedad ligada (bound property)

Fácil · Componentes de Acceso a Datos · apuntesdam.com/subject/acceso-datos/topic/componentes-acceso-datos

Solución (java)

```
import java.beans.*;
import java.io.Serializable;
import java.util.Scanner;

public class Main {

    public static class Termostato implements Serializable {
        private final PropertyChangeSupport soporte = new PropertyChangeSupport(this);
        private int temperatura = 20;

        public Termostato() {}

        public int getTemperatura() { return temperatura; }

        public void setTemperatura(int nueva) {
            int anterior = temperatura;
            temperatura = nueva;
            soporte.firePropertyChange("temperatura", anterior, nueva);
        }

        public void addPropertyChangeListener(PropertyChangeListener l) {
            soporte.addPropertyChangeListener(l); }
        public void removePropertyChangeListener(PropertyChangeListener l) {
            soporte.removePropertyChangeListener(l); }
    }

    public static void main(String[] args) {
        Termostato t = new Termostato();
        t.addPropertyChangeListener(e ->
            System.out.println(e.getPropertyName() + ": " + e.getOldValue() + " -> " + e.getNewValue()));

        Scanner sc = new Scanner(System.in);
        while (sc.hasNextInt()) t.setTemperatura(sc.nextInt());
        System.out.println("Final: " + t.getTemperatura());
    }
}
```

Antes de cambiar el valor se guarda el anterior, porque el evento necesita los dos. `firePropertyChange` se encarga del resto: avisa a todos los oyentes registrados y, si el valor viejo y el nuevo son iguales, no avisa a nadie. Por eso no hace falta comprobarlo a mano.

84. Propiedad restringida (constrained)

Medio · Componentes de Acceso a Datos · apuntesdam.com/subject/acceso-datos/topic/componentes-acceso-datos

Solución (java)

```
import java.beans.*;
import java.util.Scanner;

public class Main {

    public static class CuentaBancaria {
        private final VetoableChangeSupport vetos = new VetoableChangeSupport(this);
        private double saldo = 100;

        public double getSaldo() { return saldo; }

        public void setSaldo(double nuevo) throws PropertyVetoException {
            vetos.fireVetoableChange("saldo", saldo, nuevo);    // si alguien veta, lanza la excepción y no
            seguimos
            saldo = nuevo;
        }

        public void addVetoableChangeListener(VetoableChangeListener l) {
            vetos.addVetoableChangeListener(l); }
    }

    public static void main(String[] args) {
        CuentaBancaria cuenta = new CuentaBancaria();
        cuenta.addVetoableChangeListener(e -> {
            if ((double) e.getNewValue() < 0) throw new PropertyVetoException("saldo negativo", e);
        });

        Scanner sc = new Scanner(System.in);
        while (sc.hasNextInt()) {
            int movimiento = sc.nextInt();
            try {
                cuenta.setSaldo(cuenta.getSaldo() + movimiento);
                System.out.println("OK " + movimiento + " -> " + cuenta.getSaldo());
            } catch (PropertyVetoException ex) {
                System.out.println("Vetado " + movimiento + ": " + ex.getMessage());
            }
        }
    }
}
```

El orden es lo importante: primero se pregunta a los oyentes con `fireVetoableChange` y solo después se cambia el saldo. Si un oyente veta, la excepción sale del método antes de llegar a la asignación, así que el valor nunca llega a cambiar.

Es la diferencia con una propiedad ligada: allí se avisa después del cambio; aquí se pide permiso antes.

85. Persistencia de un JavaBean en XML

Medio · Componentes de Acceso a Datos · apuntesdam.com/subject/acceso-datos/topic/componentes-acceso-datos

Solución (java)

```
import java.beans.*;
import java.io.*;
import java.util.Scanner;

public class Main {

    public static class Configuracion {
        private String tema = "claro";
        private int tamanoLetra = 14;

        public Configuracion() {}
        public String getTema() { return tema; }
        public void setTema(String tema) { this.tema = tema; }
        public int getTamanoLetra() { return tamanoLetra; }
        public void setTamanoLetra(int t) { this.tamanoLetra = t; }
    }

    static void guardar(Configuracion c, File fichero) throws IOException {
        try (XMLEncoder enc = new XMLEncoder(new BufferedOutputStream(new FileOutputStream(fichero)))) {
            enc.writeObject(c);
        }
    }

    static Configuracion cargar(File fichero) throws IOException {
        try (XMLDecoder dec = new XMLDecoder(new BufferedInputStream(new FileInputStream(fichero)))) {
            return (Configuracion) dec.readObject();
        }
    }

    public static void main(String[] args) throws IOException {
        Scanner sc = new Scanner(System.in);
        Configuracion c = new Configuracion();
        c.setTema(sc.next());
        c.setTamanoLetra(sc.nextInt());

        File f = new File("config.xml");
        guardar(c, f);
        Configuracion leida = cargar(f);
        System.out.println(leida.getTema() + " " + leida.getTamanoLetra());
        System.out.println("XML generado: " + (f.length() > 0));
    }
}
```

XMLEncoder recorre los getters del bean y escribe en XML las propiedades que tienen un valor distinto del que daría el constructor vacío. **XMLDecoder** hace lo contrario: crea el objeto con ese constructor y llama a los setters. Por eso el bean tiene que seguir las convenciones de JavaBeans.

Los dos implementan **AutoCloseable**, así que se usan con **try** con recursos. Es importante cerrar el encoder: hasta que se cierra no se termina de escribir el documento.

86. Componente DAO con eventos

Difícil · Componentes de Acceso a Datos · apuntesdam.com/subject/acceso-datos/topic/componentes-acceso-datos

Solución (java)

```
import java.util.*;

public class Main {

    public interface DaoListener { void datosCambiados(String accion, String nombre); }

    public interface AlumnoDao {
        void guardar(String nombre, double nota);
        boolean borrar(String nombre);
        Map<String, Double> listar();
        void addDaoListener(DaoListener l);
    }

    public static class AlumnoDaoMemoria implements AlumnoDao {
        private final Map<String, Double> alumnos = new TreeMap<>();
        private final List<DaoListener> oyentes = new ArrayList<>();

        private void avisar(String accion, String nombre) {
            for (DaoListener l : oyentes) l.datosCambiados(accion, nombre);
        }

        public void guardar(String nombre, double nota) {
            alumnos.put(nombre, nota);
            avisar("guardado", nombre);
        }

        public boolean borrar(String nombre) {
            if (alumnos.remove(nombre) == null) return false;    // no existía: no hay nada que avisar
            avisar("borrado", nombre);
            return true;
        }

        public Map<String, Double> listar() { return Collections.unmodifiableMap(alumnos); }

        public void addDaoListener(DaoListener l) { oyentes.add(l); }
    }

    public static void main(String[] args) {
        AlumnoDao dao = new AlumnoDaoMemoria();
        dao.addDaoListener((accion, nombre) -> System.out.println("[evento] " + accion + " " + nombre));

        Scanner sc = new Scanner(System.in);
        while (sc.hasNext()) {
            String orden = sc.next();
            switch (orden) {
                case "G" -> dao.guardar(sc.next(), Double.parseDouble(sc.next()));
                case "B" -> { String n = sc.next(); if (!dao.borrar(n)) System.out.println("No existe " + n); }
                case "L" -> dao.listar().forEach((n, nota) -> System.out.println(n + " " + nota));
                default -> System.out.println("Orden desconocida");
            }
        }
    }
}
```

Un método privado `avisar` recorre los oyentes, y tanto `guardar` como `borrar` lo llaman después de modificar los datos. `Map.remove` devuelve el valor que había (o `null` si la clave no existía), lo que sirve para saber si de verdad se borró algo sin buscarlo antes.

El `main` solo conoce la interfaz `AlumnoDao`: podría cambiarse la implementación en memoria por una con JDBC sin tocar el resto del programa. Esa es la ventaja del patrón DAO.

87. Un contenedor de inyección de dependencias

Muy difícil · Componentes de Acceso a Datos · apuntesdam.com/subject/acceso-datos/topic/componentes-acceso-datos

Solución (java)

```
import java.util.*;

public class Main {
    static Map<String, List<String>> dependencias = new LinkedHashMap<>();
    static Map<String, String> ambitos = new HashMap<>();
    static Map<String, Integer> singletons = new HashMap<>(); // componente → número de instancia
    static List<String> creados = new ArrayList<>();           // singletons en orden de creación (para
                                                                cerrar)
    static int contador = 0;

    // Devuelve el primer error del grafo de dependencias de nombre, o null si es válido
    static String validar(String nombre, List<String> camino, String quien) {
        if (!dependencias.containsKey(nombre)) {
            return quien == null ? "no existe el componente " + nombre
                : "falta el componente " + nombre + " (lo necesita " + quien + ")";
        }
        int i = camino.indexOf(nombre);
        if (i >= 0) { // ya estamos dentro de él: ciclo
            List<String> ciclo = new ArrayList<>(camino.subList(i, camino.size()));
            ciclo.add(nombre);
            return "dependencia circular " + String.join(" → ", ciclo);
        }
        camino.add(nombre);
        for (String d : dependencias.get(nombre)) {
            String error = validar(d, camino, nombre);
            if (error != null) return error;
        }
        camino.remove(camino.size() - 1);
        return null;
    }

    // Crea (o reutiliza) el componente y sus dependencias; devuelve su número de instancia
    static int resolver(String nombre) {
        boolean prototype = ambitos.get(nombre).equals("prototype");
        if (!prototype && singletons.containsKey(nombre)) return singletons.get(nombre);
        for (String d : dependencias.get(nombre)) resolver(d);
        int n = ++contador;
        System.out.println(" crear " + nombre + "#" + n + (prototype ? " (prototype)" : ""));
        if (!prototype) {
            singletons.put(nombre, n);
            creados.add(nombre);
        }
        return n;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");
            switch (p[0]) {
                case "componente" -> { // componente N [depende=A,B] [ambito=prototype]
                    List<String> deps = new ArrayList<>();
                    String ambito = "singleton";
                    for (int i = 2; i < p.length; i++) {
                        if (p[i].startsWith("depende=")) deps.addAll(List.of(p[i].substring(8).split(",")));
                        if (p[i].startsWith("ambito=")) ambito = p[i].substring(7);
                    }
                    dependencias.put(p[1], deps);
                    ambitos.put(p[1], ambito);
                }
                case "obtener" -> {
                    System.out.println("obtener " + p[1]);
                    String error = validar(p[1], new ArrayList<>(), null);
                    if (error != null) {
                        System.out.println(" Error: " + error);
                        break;
                    }
                }
            }
        }
    }
}
```



```

        boolean existia = !ambitos.get(p[1]).equals("prototype") &&
            singletons.containsKey(p[1]);
        int n = resolver(p[1]);
        System.out.println("→ " + p[1] + "#" + n + (existia ? " (ya existía)" : ""));
    }
    case "cerrar" -> {
        for (int i = creados.size() - 1; i >= 0; i--) {
            String c = creados.get(i);
            System.out.println(" destruir " + c + "#" + singletons.get(c));
        }
        creados.clear();
        singletons.clear();
        System.out.println("Contenedor cerrado");
    }
}
}
}
}

```

La validación es un recorrido en profundidad del grafo que lleva el camino actual: si una dependencia ya está en el camino, hay un ciclo, y el trozo de camino desde su primera aparición es justo lo que hay que mostrar. Al volver de un componente se quita del camino, así dos ramas que comparten una dependencia (DataSource) no se confunden con un ciclo.

`resolver` es el mismo recorrido, pero creando: primero las dependencias y después el componente, que es el orden en que un contenedor puede inyectarlas. Los singletons se guardan en un mapa y se devuelven sin crear nada si ya existen; los prototype se crean siempre.

Destruir en orden inverso a la creación garantiza que ningún componente se destruya antes que los que dependen de él.

88. Filtrar un fichero de log

Fácil · Ejercicios de Acceso a Datos · apuntesdam.com/subject/acceso-datos/topic/ejercicios-acceso-datos

Solución (java)

```
import java.io.*;
import java.nio.file.*;
import java.util.*;

public class Main {

    static int filtrar(Path origen, Path destino) throws IOException {
        List<String> errores = new ArrayList<>();
        for (String linea : Files.readAllLines(origen)) {
            if (linea.contains("ERROR")) errores.add(linea);
        }
        Files.write(destino, errores);
        return errores.size();
    }

    public static void main(String[] args) throws IOException {
        Path log = Path.of("app.log");
        Files.write(log, new String(System.in.readAllBytes()).lines().toList());

        Path errores = Path.of("errores.log");
        int n = filtrar(log, errores);
        System.out.println("Errores: " + n);
        if (Files.exists(errores)) Files.readAllLines(errores).forEach(System.out::println);
    }
}
```

`Files.readAllLines` lee el fichero entero como una lista de líneas y `Files.write` escribe una lista de líneas de golpe: con ficheros pequeños es la forma más corta de trabajar con NIO. Se filtran las líneas que contienen «ERROR» y el tamaño de la lista es el número de copias.

Con ficheros de log enormes convendría no cargarlo todo en memoria y usar `Files.lines` (un stream) o un `BufferedReader`.

89. Conversor de CSV a JSON

Medio · Ejercicios de Acceso a Datos · apuntesdam.com/subject/acceso-datos/topic/ejercicios-acceso-datos

Solución (java)

```
import java.io.*;
import java.nio.file.*;
import java.util.*;

public class Main {

    static String convertir(Path csv) throws IOException {
        List<String> lineas = Files.readAllLines(csv);
        String[] cabecera = lineas.get(0).split(";");
        List<String> objetos = new ArrayList<>();
        for (int i = 1; i < lineas.size(); i++) {
            if (lineas.get(i).isBlank()) continue;
            String[] valores = lineas.get(i).split(";");
            List<String> pares = new ArrayList<>();
            for (int k = 0; k < cabecera.length; k++) {
                String v = valores[k];
                // los números van sin comillas; el texto, entre comillas
                String json = v.matches("-?\\d+(\\.\\d+)?") ? v : "\"" + v.replace("\"", "\\\"") + "\"";
                pares.add("\"" + cabecera[k] + "\": " + json);
            }
            objetos.add("{ " + String.join(", ", pares) + " }");
        }

        return "[" + String.join(", ", objetos) + "]";
    }

    public static void main(String[] args) throws IOException {
        Path csv = Path.of("personas.csv");
        Files.write(csv, new String(System.in.readAllBytes()).lines().toList());
        System.out.println(convertir(csv));
    }
}
```

Cada línea de datos se convierte en un objeto JSON emparejando cada valor con el nombre de su columna de la cabecera, y los objetos se unen con comas dentro de los corchetes. Construirlo con listas y `String.join` evita tener que controlar si sobra una coma al final.

Los valores que son números se escriben sin comillas y el resto entre comillas, escapando las comillas que pudiera haber dentro del texto para que el JSON siga siendo válido.

90. Media de precios por categoría (DOM)

Difícil · Ejercicios de Acceso a Datos · apuntesdam.com/subject/acceso-datos/topic/ejercicios-acceso-datos

Solución (java)

```
import java.io.*;
import java.nio.file.*;
import java.util.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;

public class Main {

    static Map<String, Double> medias(File fichero) throws Exception {
        Map<String, Double> resultado = new TreeMap<>();
        Document doc = DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(fichero);
        NodeList productos = doc.getElementsByTagName("producto");
        Map<String, Double> sumas = new HashMap<>();
        Map<String, Integer> cuentas = new HashMap<>();
        for (int i = 0; i < productos.getLength(); i++) {
            Element p = (Element) productos.item(i);
            String categoria = p.getAttribute("categoria");
            double precio =
                Double.parseDouble(p.getElementsByTagName("precio").item(0).getTextContent().trim());
            sumas.merge(categoria, precio, Double::sum);
            cuentas.merge(categoria, 1, Integer::sum);
        }
        for (String c : sumas.keySet()) resultado.put(c, sumas.get(c) / cuentas.get(c));

        return resultado;
    }

    public static void main(String[] args) throws Exception {
        File f = new File("catalogo.xml");
        Files.write(f.toPath(), System.in.readAllBytes());
        medias(f).forEach((cat, media) -> System.out.printf(Locale.US, "%s %.2f%n", cat, media));
    }
}
```

Para cada `<producto>` se lee el atributo con `getAttribute` y el precio buscando el elemento hijo con `getElementsByTagName("precio")` y tomando su texto con `getTextContent()`.

La media necesita la suma y el número de elementos de cada categoría, así que se acumulan en dos mapas con `merge` y al final se divide. El resultado es un `TreeMap`, por eso las categorías salen en orden alfabético.

91. La validación de un formulario de registro

Medio · Generación de Interfaces Gráficas · apuntesdam.com/subject/desarrollo-interfaces/topic/generacion-interfaces

Solución (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, String> campos = new LinkedHashMap<>();
        while (sc.hasNextLine()) {
            String linea = sc.nextLine();
            int i = linea.indexOf('=');
            if (i > 0) campos.put(linea.substring(0, i).trim(), linea.substring(i + 1).trim());
        }
        String nombre = campos.getDefault("nombre", ""), email = campos.getDefault("email", ""),
            edad = campos.getDefault("edad", ""), clave = campos.getDefault("clave", ""),
            repetir = campos.getDefault("repetir", "");
        List<String> errores = new ArrayList<>();

        if (nombre.isEmpty()) errores.add("nombre: es obligatorio");
        else if (nombre.length() < 2) errores.add("nombre: debe tener al menos 2 caracteres");

        if (!email.matches("[^@\\s]+@[^@\\s]+\\.\\.[A-Za-z]{2,}")) errores.add("email: formato no válido");

        try {
            int años = Integer.parseInt(edad);
            if (años < 18 || años > 120) errores.add("edad: debe estar entre 18 y 120");
        } catch (NumberFormatException e) {
            errores.add("edad: debe ser un número entero");
        }

        List<String> falta = new ArrayList<>();
        if (clave.length() < 8) falta.add("8 caracteres");
        if (clave.chars().noneMatch(Character::isUpperCase)) falta.add("una mayúscula");
        if (clave.chars().noneMatch(Character::isDigit)) falta.add("un número");
        if (!falta.isEmpty()) errores.add("clave: necesita " + String.join(", ", falta));

        if (!repetir.equals(clave)) errores.add("repetir: las contraseñas no coinciden");

        errores.forEach(System.out::println);
        if (errores.isEmpty()) System.out.println("Formulario válido: botón Registrar activado");
        else System.out.println(errores.size() + (errores.size() == 1 ? " error" : " errores") + ": botón Registrar desactivado");
    }
}
```

Cada campo se valida por separado y sus errores se acumulan en una lista, en el orden del formulario.

`getDefault(campo, "")` hace que un campo que no llega se trate igual que uno vacío.

El email se comprueba con una expresión regular: algo sin @ ni espacios, una @, algo más y un punto seguido de al menos dos letras. La edad se intenta convertir con `parseInt` y, si no es un número, la excepción se transforma en el mensaje de error.

Para la clave se reúne en otra lista solo lo que falta, y `chars().noneMatch(...)` comprueba si no hay ningún carácter que cumpla la condición. Con la lista de errores, el botón se activa si está vacía: así funcionan los formularios reales.

92. Repartir el ancho como un GridPane

Difícil · Generación de Interfaces Gráficas · apuntesdam.com/subject/desarrollo-interfaces/topic/generacion-interfaces

Solución (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int ancho = Integer.parseInt(sc.nextLine().split(" ")[1]);
        int hueco = Integer.parseInt(sc.nextLine().split(" ")[1]);
        List<String[]> columnas = new ArrayList<>(); // {nombre, tipo, valor}
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (!l.isEmpty()) columnas.add(l.split(" "));
        }

        double disponible = ancho - hueco * (columnas.size() - 1);
        double usado = 0;
        int crecen = 0;
        for (String[] c : columnas) {
            switch (c[1]) {
                case "fija" -> usado += Double.parseDouble(c[2]);
                case "porcentaje" -> usado += disponible * Double.parseDouble(c[2]) / 100;
                default -> crecen++;
            }
        }
        double resto = disponible - usado;
        if (resto < crecen) { // cada columna que crece necesita al menos 1 px
            System.out.printf(Locale.US, "No cabe: faltan %.1f px%n", crecen - resto);
            return;
        }

        double x = 0;
        for (String[] c : columnas) {
            double a = switch (c[1]) {
                case "fija" -> Double.parseDouble(c[2]);
                case "porcentaje" -> disponible * Double.parseDouble(c[2]) / 100;
                default -> resto / crecen;
            };
            System.out.printf(Locale.US, "%s: x=%.1f ancho=%.1f%n", c[0], x, a);
            x += a + hueco;
        }
        if (crecen == 0 && resto > 0) System.out.printf(Locale.US, "Sobran %.1f px a la derecha%n", resto);
    }
}
```

Es el mismo algoritmo que siguen los contenedores de rejilla: primero se descuentan los huecos entre columnas, después se reserva lo fijo y lo porcentual, y lo que sobra se reparte a partes iguales entre las columnas que crecen.

Hacen falta dos recorridos: en el primero se suma lo reservado y se cuentan las columnas que crecen (no se puede saber cuánto les toca hasta conocer el total); en el segundo se calcula el ancho de cada una y se va acumulando la x, sumando el hueco tras cada columna.

La condición `resto < crecen` cubre los dos casos de error a la vez: si no crece ninguna, salta cuando lo reservado supera el disponible; si crecen, cuando no queda al menos 1 px para cada una.

93. Captura y burbuja de un evento

Muy difícil · Generación de Interfaces Gráficas · apuntesdam.com/subject/desarrollo-interfaces/topic/generacion-interfaces

Solución (java)

```
import java.util.*;

public class Main {
    // Ejecuta, en orden de registro, los filtros o manejadores de un nodo; devuelve si alguno consumió el evento
    static boolean ejecutar(List<String[]> manejadores, String tipo, String nodo) {
        boolean consumido = false;
        for (String[] m : manejadores) {
            if (!m[0].equals(tipo) || !m[1].equals(nodo)) continue;
            boolean consume = m.length > 2 && m[2].equals("consume");
            System.out.println(tipo + " de " + nodo + (consume ? " (consume el evento)" : ""));
            consumido |= consume; // los demás del mismo nodo se ejecutan igualmente
        }
        return consumido;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, String> padre = new HashMap<>();
        List<String[]> manejadores = new ArrayList<>(); // {tipo (filtro|manejador), nodo, consume?}
        String objetivo = null;
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");
            switch (p[0]) {
                case "nodo" -> padre.put(p[1], p[2]); // nodo HIJO PADRE
                case "filtro", "manejador" -> manejadores.add(p);
                case "clic" -> objetivo = p[1];
                default -> { }
            }
        }

        // Ruta de despacho: del objetivo a la raíz, siguiendo los padres, y al revés
        List<String> ruta = new ArrayList<>();
        for (String n = objetivo; n != null; n = padre.get(n)) ruta.add(0, n);
        System.out.println("Ruta: " + String.join(" > ", ruta));

        boolean consumido = false;
        for (int i = 0; i < ruta.size() && !consumido; i++) { // captura: raíz → objetivo
            consumido = ejecutar(manejadores, "filtro", ruta.get(i));
        }
        for (int i = ruta.size() - 1; i >= 0 && !consumido; i--) { // burbuja: objetivo → raíz
            consumido = ejecutar(manejadores, "manejador", ruta.get(i));
        }
        System.out.println(consumido ? "Evento consumido" : "El evento llegó a la raíz sin consumirse");
    }
}
```

Primero se construye la ruta subiendo desde el objetivo por el mapa de padres hasta la raíz (que no tiene padre) e insertando cada nodo al principio, para que quede de la raíz al objetivo.

Las dos fases son dos bucles sobre esa ruta, uno en cada sentido: la captura ejecuta los filtros de la raíz al objetivo y la burbuja los manejadores del objetivo a la raíz. Ambos se detienen en cuanto un nodo consume el evento.

El método `ejecutar` recorre todos los manejadores registrados de un nodo antes de devolver si alguno lo consumió (`consumido |= consume`), que es justo la regla de JavaFX: el consumo afecta a los nodos siguientes, no a los del mismo nodo.

94. Un intérprete de órdenes de voz

Medio · Interfaces Naturales de Usuario · apuntesdam.com/subject/desarrollo-interfaces/topic/interfaces-naturales

Solución (java)

```
import java.text.Normalizer;
import java.util.*;
import java.util.regex.*;

public class Main {
    static final Pattern ALARMA = Pattern.compile("\\b(pon|poner|pone|programa) (una )?alarma a las (\\d{1,2}):((\\d{2}))?( y (media|cuarto))?"");
    static final Pattern LLAMADA = Pattern.compile("\\b(llama|llamar) a (.+)");
    static final Pattern VOLUMEN = Pattern.compile("\\b(sube|baja) (el )?volumen( (\\d+))?"");
    static final Pattern ABRIR = Pattern.compile("\\b(abre|abrir|inicia) (la app |la aplicacion )?(\\S+)");

    static String normalizar(String s) {
        String sinTildes = Normalizer.normalize(s.toLowerCase(), Normalizer.Form.NFD).replaceAll("\\p{M}", "");
        return sinTildes.replaceAll("[^a-z0-9: ]", " ").replaceAll("\\s+", " ").trim();
    }

    static String capitalizar(String nombre) {
        StringBuilder sb = new StringBuilder();
        for (String p : nombre.split(" ")) {
            if (sb.length() > 0) sb.append(' ');
            sb.append(Character.toUpperCase(p.charAt(0))).append(p.substring(1));
        }
        return sb.toString();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String frase = sc.nextLine();
            if (frase.isBlank()) continue;
            String t = normalizar(frase);
            Matcher m;
            if ((m = ALARMA.matcher(t)).find()) {
                int h = Integer.parseInt(m.group(3));
                int min = m.group(5) != null ? Integer.parseInt(m.group(5)) : "media".equals(m.group(7)) ? 30 : "cuarto".equals(m.group(7)) ? 15 : 0;
                System.out.println(h > 23 || min > 59 ? "ERROR: hora no válida" : String.format("ALARMA %02d:%02d", h, min));
            } else if ((m = LLAMADA.matcher(t)).find()) {
                System.out.println("LLAMAR " + capitalizar(m.group(2)));
            } else if ((m = VOLUMEN.matcher(t)).find()) {
                String n = m.group(4) != null ? m.group(4) : "10";
                System.out.println("VOLUMEN " + (m.group(1).equals("sube") ? "+" : "-") + n);
            } else if ((m = ABRIR.matcher(t)).find()) {
                System.out.println("ABRIR " + m.group(3));
            } else {
                System.out.println("NO ENTENDIDO: " + frase);
            }
        }
    }
}
```

Cada intención es una expresión regular con grupos para los datos que interesan: la hora, los minutos o el «y media» de la alarma, el nombre de la llamada o la cantidad de volumen. Se usa `find()` en lugar de `matches()` para que la orden se reconozca aunque haya palabras antes («Oye, programa...»).

Las partes opcionales se escriben con `( ... )?`, y si no aparecen su grupo vale `null`: así se decide si hay minutos, si se dijo «y media» o si se usa el volumen por defecto. Gracias a `normalizar`, las expresiones no tienen que preocuparse de mayúsculas, tildes ni signos.

Las expresiones se prueban en el orden del enunciado y se guardan como constantes `static final` para compilarlas una sola vez, no en cada frase.

95. Clasificar gestos táctiles

Difícil · Interfaces Naturales de Usuario · apuntesdam.com/subject/desarrollo-interfaces/topic/interfaces-naturales

Solución (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<List<double[]>> gestos = new ArrayList<>();
        List<double[]> actual = new ArrayList<>();
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (l.isEmpty()) { if (!actual.isEmpty()) { gestos.add(actual); actual = new ArrayList<>(); }
                continue; }
            String[] p = l.split(" "); // tiempo_ms x y
            actual.add(new double[]{Double.parseDouble(p[0]), Double.parseDouble(p[1]),
                Double.parseDouble(p[2])});
        }
        if (!actual.isEmpty()) gestos.add(actual);

        for (int i = 0; i < gestos.size(); i++) {
            List<double[]> g = gestos.get(i);
            double[] ini = g.get(0), fin = g.get(g.size() - 1);
            double dx = fin[1] - ini[1], dy = fin[2] - ini[2];
            double distancia = Math.hypot(dx, dy);
            double duracion = fin[0] - ini[0];
            double velocidad = distancia / duracion;
            String tipo;
            if (distancia < 10) {
                tipo = duracion < 300 ? "TOQUE" : duracion >= 500 ? "PULSACIÓN LARGA" : "INDECISO";
            } else if (distancia >= 50 && velocidad >= 0.3) {
                if (Math.abs(dx) >= Math.abs(dy)) tipo = dx > 0 ? "DESLIZAR DERECHA" : "DESLIZAR IZQUIERDA";
                else tipo = dy > 0 ? "DESLIZAR ABAJO" : "DESLIZAR ARRIBA"; // la y crece hacia abajo
            } else {
                tipo = "ARRASTRE";
            }
            System.out.printf(Locale.US, "Gesto %d: %s (%.0f px en %.0f ms, %.2f px/ms)%n", i + 1, tipo,
                distancia, duracion, velocidad);
        }
    }
}
```

Solo hacen falta el primer y el último punto: la diferencia de x e y da el desplazamiento, `Math.hypot(dx, dy)` la distancia en línea recta y la diferencia de tiempos la duración.

La clasificación es un árbol de decisiones: si el dedo apenas se movió, decide el tiempo (toque, indeciso o pulsación larga); si se movió mucho y deprisa es un deslizamiento, y el eje con mayor desplazamiento da la dirección; en otro caso, un arrastre. Como en pantalla la y crece hacia abajo, una dy negativa es un deslizamiento hacia arriba.

96. ¿Qué gesto hace la mano?

Muy difícil · Interfaces Naturales de Usuario · apuntesdam.com/subject/desarrollo-interfaces/topic/interfaces-naturales

Solución (java)

```
import java.util.*;

public class Main {
    static double dist(double[] a, double[] b) {
        return Math.hypot(a[0] - b[0], a[1] - b[1]);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<Map<String, double[]>> manos = new ArrayList<>();
        Map<String, double[]> actual = new LinkedHashMap<>();
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (l.equals("---")) { manos.add(actual); actual = new LinkedHashMap<>(); continue; }
            if (l.isEmpty()) continue;
            String[] p = l.split(" "); // punto x y (coordenadas normalizadas de 0 a 1)
            actual.put(p[0], new double[]{Double.parseDouble(p[1]), Double.parseDouble(p[2])});
        }
        if (!actual.isEmpty()) manos.add(actual);
        String[] DEDOS = {"pulgar", "indice", "corazon", "anular", "menique"};

        for (int i = 0; i < manos.size(); i++) {
            Map<String, double[]> mano = manos.get(i);
            double[] muneca = mano.get("muneca");
            List<String> extendidos = new ArrayList<>();
            for (String dedo : DEDOS) {
                double punta = dist(muneca, mano.get(dedo + "_punta"));
                double medio = dist(muneca, mano.get(dedo + "_medio"));
                if (punta > medio * (dedo.equals("pulgar") ? 1.1 : 1.15)) extendidos.add(dedo);
            }
            boolean pinza = dist(mano.get("pulgar_punta"), mano.get("indice_punta"))
                < 0.25 * dist(muneca, mano.get("indice_medio"));
            String gesto = pinza ? "PINZA"
                : extendidos.isEmpty() ? "PUÑO"
                : extendidos.size() == 5 ? "MANO ABIERTA"
                : extendidos.equals(List.of("pulgar")) ? "PULGAR"
                : extendidos.equals(List.of("indice", "corazon")) ? "VICTORIA"
                : extendidos.equals(List.of("indice")) ? "SEÑALAR"
                : "DESCONOCIDO";
            System.out.println("Mano " + (i + 1) + ": " + gesto + " (extendidos: "
                + (extendidos.isEmpty() ? "ninguno" : String.join(", ", extendidos)) + ")");
        }
    }
}
```

Comparar distancias a la muñeca en lugar de coordenadas hace que el resultado no dependa de cómo esté girada la mano ni de lo cerca que esté de la cámara: un dedo estirado tiene la punta bastante más lejos de la muñeca que su articulación media, y uno doblado no.

La pinza también usa una distancia relativa (al tamaño de la mano, medido de la muñeca al índice), porque en coordenadas normalizadas una mano lejana es más pequeña.

Como los dedos extendidos se añaden en el orden de `DEDOS`, cada gesto se reconoce comparando la lista con otra fija (`List.of("indice", "corazon")`); la cadena de ternarios respeta el orden de prioridad del enunciado.

97. Un componente de valoración por estrellas

Medio · Creación de Componentes Visuales · apuntesdam.com/subject/desarrollo-interfaces/topic/creacion-componentes

Solución (java)

```
import java.util.*;

public class Main {
    static class Valoracion {
        private double valor = 0;

        private static String texto(double v) {
            return v == Math.floor(v) ? String.valueOf((long) v) : String.valueOf(v);
        }

        void setValor(double v) {
            double nuevo = Math.round(Math.max(0, Math.min(5, v)) * 2) / 2.0;
            if (nuevo != valor) {
                System.out.println("cambio: " + texto(valor) + " → " + texto(nuevo));
                valor = nuevo;
            }
        }

        void clic(int estrella) {
            setValor(estrella == valor ? 0 : estrella); // pulsar la misma estrella borra la valoración
        }

        String pintar() {
            StringBuilder sb = new StringBuilder();
            for (int i = 1; i <= 5; i++) {
                sb.append(valor >= i ? "★" : valor >= i - 0.5 ? "½" : "☆");
            }
            return sb + " (" + texto(valor) + ")";
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Valoracion v = new Valoracion();
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");
            if (p[0].equals("set")) v.setValor(Double.parseDouble(p[1]));
            else if (p[0].equals("clic")) v.clic(Integer.parseInt(p[1]));
            else continue;
            System.out.println(v.pintar());
        }
    }
}
```

Todo cambio de valor pasa por `setValor`, que es el único sitio que valida (recorta a [0, 5] y redondea al medio punto) y avisa del cambio. `clic` no toca el atributo directamente: decide el nuevo valor y llama a `setValor`, así la regla nunca se salta. Es el mismo principio que siguen los componentes reales con sus propiedades.

Para pintar, la estrella *i* está llena si el valor llega a *i*, a medias si llega a *i* - 0,5 y vacía en otro caso. El método `texto` quita el «.0» de los valores enteros.

98. Propiedades observables y bindings

Difícil · Creación de Componentes Visuales · apuntesdam.com/subject/desarrollo-interfaces/topic/creacion-componentes

Solución (java)

```
import java.util.*;
import java.util.function.*;

public class Main {
    static class Propiedad {
        final String nombre;
        private int valor = 0;
        private final List<BiConsumer<Integer, Integer>> oyentes = new ArrayList<>();
        private final List<Propiedad> dependientes = new ArrayList<>(); // las enlazadas a esta
        private Propiedad fuente = null; // a la que está enlazada esta

        Propiedad(String nombre) { this.nombre = nombre; }

        int get() { return valor; }

        void set(int v) {
            if (fuente != null) {
                System.out.println("ERROR: " + nombre + " está enlazada a " + fuente.nombre + " y no se puede cambiar");
                return;
            }
            cambiar(v);
        }

        private void cambiar(int v) {
            if (v == valor) return;
            int viejo = valor;
            valor = v;
            for (BiConsumer<Integer, Integer> o : oyentes) o.accept(viejo, v);
            for (Propiedad d : dependientes) d.cambiar(v); // se propaga a las enlazadas
        }

        void escuchar() {
            int n = oyentes.size() + 1;
            oyentes.add((viejo, nuevo) -> System.out.println(" " + nombre + " oyente " + n + ": " + viejo + " → " + nuevo));
        }

        void bind(Propiedad otra) {
            for (Propiedad p = otra; p != null; p = p.fuente) {
                if (p == this) {
                    System.out.println("ERROR: enlace circular");
                    return;
                }
            }
            unbind();
            fuente = otra;
            otra.dependientes.add(this);
            cambiar(otra.valor);
        }

        void unbind() {
            if (fuente != null) {
                fuente.dependientes.remove(this);
                fuente = null;
            }
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, Propiedad> props = new LinkedHashMap<>();
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");
            if (p.length < 2) continue;
            Propiedad a = props.computeIfAbsent(p[1], Propiedad::new);
            switch (p[0]) {
                case "set" -> a.set(Integer.parseInt(p[2]));
            }
        }
    }
}
```

```

        case "escuchar" -> a.escuchar();
        case "bind" -> a.bind(props.computeIfAbsent(p[2], Propiedad::new));
        case "unbind" -> a.unbind();
        case "get" -> System.out.println(a.nombre + " = " + a.get());
    }
}
}

```

Es el patrón Observador: cada propiedad guarda una lista de oyentes (`BiConsumer` que reciben el valor viejo y el nuevo) y los llama cuando su valor cambia de verdad. Los enlaces son el mismo mecanismo: una propiedad enlazada es una dependiente a la que se le propaga cada cambio.

`set` (público) comprueba que la propiedad no esté enlazada; `cambiar` (privado) es el que realmente cambia el valor, avisa y propaga, y lo usan tanto `set` como la propagación. Si el valor no cambia, `cambiar` termina enseguida, lo que también evita avisos inútiles.

Para detectar un ciclo, `bind` sube por la cadena de fuentes de la otra propiedad: si en algún momento aparece la propia propiedad, enlazarlas crearía un bucle infinito de propagaciones.

99. Un campo de texto con máscara

Muy difícil · Creación de Componentes Visuales · apuntesdam.com/subject/desarrollo-interfaces/topic/creacion-

```

import java.util.*;

public class Main {
    static boolean esHueco(char m) {
        return m == '#' || m == 'A' || m == '?';
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String mascara = sc.nextLine().substring("mascara ".length()); // # dígito, A letra, ? cualquiera;
        // lo demás es literal
        List<String> teclas = new ArrayList<>(Arrays.asList(sc.nextLine().split(" ")));

        StringBuilder texto = new StringBuilder();
        List<String> rechazadas = new ArrayList<>();
        for (String tecla : teclas) {
            if (tecla.equals("BORRAR")) {
                if (texto.length() > 0) texto.setLength(texto.length() - 1);
                // también se quitan los literales que queden al final
                while (texto.length() > 0 && !esHueco(mascara.charAt(texto.length() - 1)))
                    texto.setLength(texto.length() - 1);
                continue;
            }
            char c = tecla.equals("ESPACIO") ? ' ' : tecla.charAt(0);
            // Siguiente hueco, saltando los literales
            int pos = texto.length();
            while (pos < mascara.length() && !esHueco(mascara.charAt(pos))) pos++;
            if (pos == mascara.length()) {
                rechazadas.add(tecla + " (campo completo)");
                continue;
            }
            char hueco = mascara.charAt(pos);
            if (hueco == '#' && !Character.isDigit(c)) {
                rechazadas.add(tecla + " (se esperaba un dígito)");
                continue;
            }
            if (hueco == 'A' && !Character.isLetter(c)) {
                rechazadas.add(tecla + " (se esperaba una letra)");
                continue;
            }
            texto.append(mascara, texto.length(), pos); // los literales de antes del hueco
            texto.append(hueco == 'A' ? Character.toUpperCase(c) : c);
        }

        StringBuilder pantalla = new StringBuilder(texto);
        boolean completo = true;
        for (int i = texto.length(); i < mascara.length(); i++) {
            char m = mascara.charAt(i);
            pantalla.append(esHueco(m) ? '_' : m);
            if (esHueco(m)) completo = false;
        }
        System.out.println("Pantalla: " + pantalla);
        System.out.println("Completo: " + (completo ? "sí" : "no"));
        rechazadas.forEach(r -> System.out.println("Rechazada: " + r));
    }
}

```

La posición en la máscara es simplemente la longitud de lo escrito, porque cada carácter escrito (incluidos los literales) ocupa una posición. Con cada tecla se busca el siguiente hueco saltando los literales y se valida la tecla contra él; solo si se acepta se escriben los literales intermedios y la tecla.

BORRAR quita el último carácter y luego sigue quitando mientras lo último sea un literal: así, al borrar el primer dígito de «(9», el paréntesis desaparece con él.

La pantalla es lo escrito más el resto de la máscara, cambiando los huecos por `_`. El campo está completo si en esa parte pendiente no queda ningún hueco.

100. Contraste de colores según WCAG

Medio · Usabilidad y Accesibilidad · apuntesdam.com/subject/desarrollo-interfaces/topic/usabilidad-accesibilidad

Solución (java)

```
import java.util.*;

public class Main {
    static double canal(int c) {
        double v = c / 255.0;
        return v <= 0.03928 ? v / 12.92 : Math.pow((v + 0.055) / 1.055, 2.4);
    }

    static double luminancia(String color) {
        String h = color.substring(1);
        if (h.length() == 3) { // #rgb → #rrggbb
            h = "" + h.charAt(0) + h.charAt(0) + h.charAt(1) + h.charAt(1) + h.charAt(2) + h.charAt(2);
        }
        int r = Integer.parseInt(h.substring(0, 2), 16);
        int g = Integer.parseInt(h.substring(2, 4), 16);
        int b = Integer.parseInt(h.substring(4, 6), 16);
        return 0.2126 * canal(r) + 0.7152 * canal(g) + 0.0722 * canal(b);
    }

    static double contraste(String a, String b) {
        double la = luminancia(a), lb = luminancia(b);
        return (Math.max(la, lb) + 0.05) / (Math.min(la, lb) + 0.05);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");
            if (p.length < 2) continue;
            double r = contraste(p[0], p[1]);
            String normal = r >= 7 ? "AAA" : r >= 4.5 ? "AA" : "no cumple";
            String grande = r >= 4.5 ? "AAA" : r >= 3 ? "AA" : "no cumple";
            System.out.printf(Locale.US, "%s sobre %s: %.2f:1 · texto normal %s · texto grande %s%n", p[0],
                p[1], r, normal, grande);
        }
    }
}
```

Cada color se descompone en sus tres canales con `Integer.parseInt(..., 16)` (el formato corto `#rgb` se amplía antes duplicando cada cifra). Cada canal se linealiza con la fórmula de WCAG, porque el ojo no percibe el brillo de forma lineal, y se combinan con pesos distintos: el verde pesa mucho más que el azul.

El contraste divide la luminancia del color más claro entre la del más oscuro (con 0,05 sumado a ambas), así que no importa cuál sea el texto y cuál el fondo. Va de 1:1 (mismo color) a 21:1 (negro sobre blanco).

101. El orden de tabulación de un formulario

Difícil · Usabilidad y Accesibilidad · apuntesdam.com/subject/desarrollo-interfaces/topic/usabilidad-accesibilidad

Solución (java)

```
import java.util.*;

public class Main {
    record Control(String id, int x, int y, int tabIndex, boolean habilitado) { }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<Control> controles = new ArrayList<>();
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" "); // id x y tabIndex habilitado(si|no)
            if (p.length < 5) continue;
            controles.add(new Control(p[0], Integer.parseInt(p[1]), Integer.parseInt(p[2]),
                Integer.parseInt(p[3]), p[4].equals("si")));
        }

        Comparator<Control> lectura = Comparator.comparingInt(Control::y).thenComparingInt(Control::x);
        List<Control> positivos = new ArrayList<>(), naturales = new ArrayList<>();
        for (Control c : controles) {
            if (!c.habilitado() || c.tabIndex() < 0) continue;
            (c.tabIndex() > 0 ? positivos : naturales).add(c);
        }
        positivos.sort(Comparator.comparingInt(Control::tabIndex).thenComparing(lectura));
        naturales.sort(lectura);

        List<Control> orden = new ArrayList<>(positivos);
        orden.addAll(naturales);
        for (int i = 0; i < orden.size(); i++) System.out.println((i + 1) + ". " + orden.get(i).id());
        if (!positivos.isEmpty()) {
            System.out.println("Aviso: " + positivos.size() + " control(es) con tabIndex positivo alteran el
                orden natural");
        }
        for (Control c : controles) {
            if (!c.habilitado()) System.out.println("Fuera del orden: " + c.id() + " (deshabilitado)");
            else if (c.tabIndex() < 0) System.out.println("Fuera del orden: " + c.id() + " (tabIndex " +
                c.tabIndex() + ")");
        }
    }
}
```

Se separan los controles que reciben el foco en dos listas, los de tabIndex positivo y los de tabIndex 0, y cada una se ordena con su `Comparator`; el orden final es la primera lista seguida de la segunda.

El orden de lectura (por y y después por x) se define una vez como `Comparator` y se reutiliza: ordena los de tabIndex 0 y desempata los positivos. Un `record` es ideal para un dato así: el constructor y los accesos `c.y()` vienen hechos.

El aviso de los tabIndex positivos tiene sentido: obligan a quien usa el teclado a saltar por la pantalla en un orden distinto del visual, y es una de las faltas de accesibilidad más comunes.

102. ¿Se entiende este texto?

Muy difícil · Usabilidad y Accesibilidad · apuntesdam.com/subject/desarrollo-interfaces/topic/usabilidad-accesibilidad

Solución (java)

```
import java.text.Normalizer;
import java.util.*;
import java.util.regex.*;

public class Main {
    static final String VOALES = "aeiouáéíóúü";
    static final String FUERTES = "aeoáéíóú"; // las débiles i, u sin tilde forman diptongo

    static boolean vocal(char c) { return VOALES.indexOf(c) >= 0; }
    static boolean fuerte(char c) { return FUERTES.indexOf(c) >= 0; }

    static int silabas(String palabra) {
        String p = palabra.toLowerCase();
        int n = 0;
        for (int i = 0; i < p.length(); i++) {
            char c = p.charAt(i);
            if (!vocal(c)) continue;
            if (i == 0 || !vocal(p.charAt(i - 1))) n++; // vocal tras consonante: sílaba nueva
            else if (fuerte(c) && fuerte(p.charAt(i - 1))) n++; // hiato: dos fuertes
            // si no, diptongo: se une a la vocal anterior
        }
        int ultima = p.length() - 1;
        if (ultima > 0 && p.charAt(ultima) == 'y' && !vocal(p.charAt(ultima - 1))) n++;
        return Math.max(n, 1);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        StringBuilder todo = new StringBuilder();
        while (sc.hasNextLine()) todo.append(sc.nextLine()).append(' ');
        String texto = todo.toString().trim();

        int palabras = 0, silabas = 0;
        Matcher m = Pattern.compile("\\p{L}+").matcher(texto);
        while (m.find()) {
            palabras++;
            silabas += silabas(m.group());
        }
        int frases = 0;
        for (String f : texto.split("[.!?]")) if (f.matches(".*\\p{L}.*")) frases++;

        double porPalabra = (double) silabas / palabras;
        double porFrase = (double) palabras / frases;
        double indice = 206.84 - 0.60 * (porPalabra * 100) - 1.02 * porFrase;
        String nivel = indice >= 90 ? "muy fácil" : indice >= 80 ? "fácil" : indice >= 70 ? "algo fácil"
            : indice >= 60 ? "normal" : indice >= 50 ? "algo difícil" : indice >= 30 ? "difícil" : "muy difícil";

        System.out.println("Palabras: " + palabras + " · Frases: " + frases + " · Sílabas: " + silabas);
        System.out.printf(Locale.US, "Sílabas por palabra: %.2f · Palabras por frase: %.1f%n", porPalabra, porFrase);
        System.out.printf(Locale.US, "Índice Fernández Huerta: %.1f (%s)%n", indice, nivel);
    }
}
```

Contar sílabas es contar núcleos vocálicos. Al recorrer la palabra, una vocal abre sílaba nueva si la anterior es una consonante, o si las dos son fuertes (hiato, «po-e-ta»); en cualquier otro caso hay diptongo y se une a la anterior («cie-lo»). Por eso hay dos funciones auxiliares, `vocal` y `fuerte`, que hacen el código casi igual que la regla.

Las palabras se cuentan con la expresión `\p{L}+` («una o más letras», incluidas las que llevan tilde o la ñ) y las frases partiendo por `. ! ?` y quedándose con los trozos que tienen alguna letra, para no contar el hueco que queda tras el último punto.

Las sílabas por cada 100 palabras son las sílabas por palabra multiplicadas por 100. Las divisiones se hacen con `(double)` para no perder los decimales.

103. El CHANGELOG a partir de los commits

Medio · Documentación de Aplicaciones · apuntesdam.com/subject/desarrollo-interfaces/topic/documentacion-aplicaciones

Solución (java)

```
import java.util.*;
import java.util.regex.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String version = sc.nextLine().split(" ")[1];          // versión actual, p. ej. 1.4.2
        List<String> commits = new ArrayList<>();
        while (sc.hasNextLine()) { String l = sc.nextLine().trim(); if (!l.isEmpty()) commits.add(l); }

        Pattern formato = Pattern.compile("^((\\w+)(?:\\(([^\\)]+\\))?)?!?: (.+)$");
        List<String> incompatibles = new ArrayList<>(), novedades = new ArrayList<>(), correcciones = new
            ArrayList<>();
        int sinEntrada = 0;
        for (String c : commits) {
            Matcher m = formato.matcher(c);
            if (!m.matches()) { sinEntrada++; continue; }
            String entrada = "- " + (m.group(2) != null ? "***" + m.group(2) + "***: " : "") + m.group(4);
            if (m.group(3) != null) incompatibles.add(entrada);
            else if (m.group(1).equals("feat")) novedades.add(entrada);
            else if (m.group(1).equals("fix")) correcciones.add(entrada);
            else sinEntrada++;
        }

        String[] v = version.split("\\.");
        int mayor = Integer.parseInt(v[0]), menor = Integer.parseInt(v[1]), parche = Integer.parseInt(v[2]);
        if (!incompatibles.isEmpty()) { mayor++; menor = 0; parche = 0; }
        else if (!novedades.isEmpty()) { menor++; parche = 0; }
        else if (!correcciones.isEmpty()) parche++;
        else {
            System.out.println("Sin cambios publicables: se mantiene la versión " + version);
            return;
        }

        System.out.println("## " + mayor + "." + menor + "." + parche);
        String[] titulos = {"Cambios incompatibles", "Novedades", "Correcciones"};
        List<List<String>> secciones = List.of(incompatibles, novedades, correcciones);
        for (int i = 0; i < 3; i++) {
            if (secciones.get(i).isEmpty()) continue;
            System.out.println();
            System.out.println("### " + titulos[i]);
            secciones.get(i).forEach(System.out::println);
        }
        System.out.println();
        System.out.println("(" + sinEntrada + " commits sin entrada en el CHANGELOG)");
    }
}
```

Una sola expresión regular separa las partes del mensaje: el tipo, el ámbito opcional entre paréntesis, el «!» opcional y la descripción. Los grupos `(?: ... )` no capturan; sirven para hacer opcional el paréntesis entero y quedarse solo con lo de dentro.

Cada commit va a una de tres listas, y la versión se decide por la lista más importante que tenga algo: un cambio incompatible sube la mayor y pone a cero las otras dos, una novedad sube la menor y pone a cero el parche, y una corrección sube el parche. Es exactamente lo que hacen herramientas como semantic-release.

Las tres secciones se imprimen con un bucle sobre dos arrays paralelos (títulos y listas) en lugar de repetir el mismo código tres veces.

104. De Javadoc a una página de documentación

Difícil · Documentación de Aplicaciones · apuntesdam.com/subject/desarrollo-interfaces/topic/documentacion-aplicaciones

Solución (java)

```
import java.util.*;

public class Main {
    // Texto de una línea de Javadoc sin /**, */ ni el * del principio
    static String limpiar(String l) {
        String s = l.trim();
        if (s.startsWith("/**")) s = s.substring(3);
        if (s.endsWith("*/")) s = s.substring(0, s.length() - 2);
        s = s.trim();
        if (s.startsWith("/*")) s = s.substring(1);
        return s.trim();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<String> lineas = new ArrayList<>();
        while (sc.hasNextLine()) lineas.add(sc.nextLine());

        List<String> javadoc = null;    // el último comentario /** */ pendiente
        int documentados = 0;
        List<String> sinDocumentar = new ArrayList<>();
        for (int i = 0; i < lineas.size(); i++) {
            String l = lineas.get(i).trim();
            if (l.isEmpty()) continue;
            if (l.startsWith("/**")) {
                javadoc = new ArrayList<>();
                while (true) {
                    String c = lineas.get(i);
                    String texto = limpiar(c);
                    if (!texto.isEmpty()) javadoc.add(texto);
                    if (c.contains("*/")) break;
                    i++;
                }
                continue;
            }
            boolean metodoPublico = l.startsWith("public") && l.contains("(") && !l.matches("public
            .*\\b(class|interface|enum|record)\\b.*");
            if (metodoPublico) {
                String firma = l.substring("public".length()).trim();
                if (firma.contains("{")) firma = firma.substring(0, firma.indexOf('{')).trim();
                String[] antes = firma.substring(0, firma.indexOf('(')).trim().split("\\s+");
                String nombre = antes[antes.length - 1];
                if (javadoc == null) {
                    sinDocumentar.add(nombre);
                } else {
                    documentados++;
                    imprimir(nombre, firma, javadoc);
                }
            }
            javadoc = null;    // un Javadoc solo vale para la línea que tiene justo debajo
        }
        System.out.println("Documentados: " + documentados + " · Sin documentar: "
            + (sinDocumentar.isEmpty() ? "ninguno" : String.join(", ", sinDocumentar)));
    }

    static void imprimir(String nombre, String firma, List<String> javadoc) {
        List<String> descripcion = new ArrayList<>(), params = new ArrayList<>(), lanza = new ArrayList<>();
        String devuelve = null;
        for (String t : javadoc) {
            String[] p = t.split("\\s+", 3);
            switch (p[0]) {
                case "@param" -> params.add("- `" + p[1] + "`: " + (p.length > 2 ? p[2] : ""));
                case "@throws", "@exception" -> lanza.add("- `" + p[1] + "`: " + (p.length > 2 ? p[2] : ""));
                case "@return" -> devuelve = t.substring("@return".length()).trim();
                default -> descripcion.add(t);
            }
        }
    }
}
```

```

        System.out.println("### " + nombre);
        System.out.println("`" + firma + "`");
        System.out.println();
        System.out.println(String.join(" ", descripcion));
        if (!params.isEmpty()) {
            System.out.println();
            System.out.println("**Parámetros**");
            params.forEach(System.out::println);
        }
        if (devuelve != null) {
            System.out.println();
            System.out.println("**Devuelve**:" + devuelve);
        }
        if (!lanza.isEmpty()) {
            System.out.println();
            System.out.println("**Lanza**");
            lanza.forEach(System.out::println);
        }
        System.out.println();
    }
}

```

El programa recorre las líneas recordando el último Javadoc leído. Cuando empieza un comentario `/**`, lee hasta la línea que contiene `*/`, limpiando de cada una los delimitadores y el asterisco inicial. La primera línea con código que viene después decide qué pasa con él: si es un método público se usa, y en cualquier caso se olvida (`javadoc = null`), para que no se aplique a un método posterior.

Para la firma se quita «public» y todo lo que va desde la llave; el nombre es la última palabra antes del paréntesis. La expresión con `\b(class|interface|enum|record)\b` descarta la declaración de la clase.

El método `imprimir` reparte las líneas del comentario según su etiqueta (`@param` , `@return` , `@throws`) y el resto forma la descripción. `split("\\s+", 3)` parte solo en tres trozos, así la descripción de un parámetro conserva sus espacios.

105. Un revisor automático de manuales

Muy difícil · Documentación de Aplicaciones · apuntesdam.com/subject/desarrollo-interfaces/topic/documentacion-

```

import java.util.*;
import java.util.regex.*;
import java.text.Normalizer;

public class Main {
    static String ancla(String titulo) {
        String s = Normalizer.normalize(titulo.toLowerCase(), Normalizer.Form.NFD).replaceAll("\\p{M}", "");
        return s.replaceAll("[^a-z0-9 -]", "").trim().replace(' ', '-');
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<String> lineas = new ArrayList<>();
        while (sc.hasNextLine()) lineas.add(sc.nextLine());

        // Qué líneas están dentro de un bloque de código
        boolean[] codigo = new boolean[lineas.size()];
        boolean dentro = false;
        for (int i = 0; i < lineas.size(); i++) {
            if (lineas.get(i).startsWith("`")) {
                dentro = !dentro;
                codigo[i] = true;
            } else {
                codigo[i] = dentro;
            }
        }

        List<String> problemas = new ArrayList<>();
        Pattern titulo = Pattern.compile("^#{1,6}\\s+(.*)$");
        Set<String> anclas = new HashSet<>();

        // 1. Títulos
        int anterior = 0;
        boolean primera = true;
        for (int i = 0; i < lineas.size(); i++) {
            if (codigo[i] || lineas.get(i).isBlank()) continue;
            Matcher m = titulo.matcher(lineas.get(i));
            if (primera && (!m.matches() || m.group(1).length() != 1)) {
                problemas.add("Línea " + (i + 1) + ": el documento debe empezar con un título de nivel 1");
            }
            primera = false;
            if (!m.matches()) continue;
            int nivel = m.group(1).length();
            if (anterior > 0 && nivel > anterior + 1) {
                problemas.add("Línea " + (i + 1) + ": salto de título de nivel " + anterior + " a " + nivel);
            }
            anterior = nivel;
            anclas.add(ancla(m.group(2)));
        }

        // 2. Imágenes y enlaces
        Pattern enlace = Pattern.compile("(!?)\\[[^\\]]*\\]\\[[^\\]]*\\]");
        Set<String> vagos = Set.of("aquí", "aquí", "pulsa aquí", "este enlace");
        for (int i = 0; i < lineas.size(); i++) {
            if (codigo[i]) continue;
            Matcher m = enlace.matcher(lineas.get(i));
            while (m.find()) {
                String texto = m.group(2).trim(), destino = m.group(3).trim();
                if (!m.group(1).isEmpty()) {
                    if (texto.isEmpty()) problemas.add("Línea " + (i + 1) + ": la imagen " + destino + " no tiene texto alternativo");
                    continue;
                }
                if (vagos.contains(texto.toLowerCase())) {
                    problemas.add("Línea " + (i + 1) + ": el texto del enlace «" + texto + "» no dice adónde lleva");
                }
                if (destino.startsWith("#") && !anclas.contains(destino.substring(1))) {

```

```

        problemas.add("Línea " + (i + 1) + ": el enlace " + destino + " apunta a una sección que
            no existe");
    }
}

// 3. Bloques sin cerrar
if (dentro) problemas.add("Hay un bloque de código sin cerrar");

problemas.forEach(System.out::println);
System.out.println(problemas.isEmpty() ? "Manual correcto"
    : problemas.size() + (problemas.size() == 1 ? " problema" : " problemas"));
}
}

```

Antes de nada se marca qué líneas están dentro de un bloque de código, cambiando de estado cada vez que aparece una línea con ``; así las dos pasadas pueden saltarse esas líneas sin repetir la lógica. Si al final el estado sigue «dentro», el bloque no se cerró.

Hacen falta dos pasadas porque un enlace puede apuntar a una sección que está más abajo: la primera recoge todos los títulos (y comprueba los niveles) y guarda sus anclas en un `Set`; la segunda ya puede comprobar cada enlace interno contra ese conjunto.

Una sola expresión regular reconoce imágenes y enlaces: el grupo `(!?)` dice si la marca empieza por «!», y los otros dos capturan el texto y el destino. Con `while (m.find())` se tratan todos los de una línea, en orden.

106. Versiones semánticas y rangos de dependencias

Medio · Distribución de Aplicaciones · apuntesdam.com/subject/desarrollo-interfaces/topic/distribucion-aplicaciones

Solución (java)

```
import java.util.*;

public class Main {
    // Compara dos partes de una etiqueta: las numéricas como números y menores que las de texto
    static int compararParte(String a, String b) {
        boolean na = a.matches("\\d+"), nb = b.matches("\\d+");
        if (na && nb) return Long.compare(Long.parseLong(a), Long.parseLong(b));
        if (na != nb) return na ? -1 : 1;
        return a.compareTo(b);
    }

    static int comparar(String a, String b) {
        String[] pa = a.split("-", 2), pb = b.split("-", 2);
        String[] na = pa[0].split("\\."), nb = pb[0].split("\\.");
        for (int i = 0; i < 3; i++) {
            int c = Integer.compare(Integer.parseInt(na[i]), Integer.parseInt(nb[i]));
            if (c != 0) return c;
        }
        boolean ea = pa.length > 1, eb = pb.length > 1;
        if (!ea || !eb) return Boolean.compare(!ea, !eb); // sin etiqueta es mayor
        String[] ta = pa[1].split("\\."), tb = pb[1].split("\\.");
        for (int i = 0; i < Math.min(ta.length, tb.length); i++) {
            int c = compararParte(ta[i], tb[i]);
            if (c != 0) return c;
        }
        return Integer.compare(ta.length, tb.length); // la que se acaba antes es menor
    }

    static boolean cumple(String version, String rango) {
        if (!rango.startsWith("^") && !rango.startsWith("~")) return comparar(version, rango) == 0;
        String base = rango.substring(1);
        String[] n = base.split("\\.");
        int x = Integer.parseInt(n[0]), y = Integer.parseInt(n[1]), z = Integer.parseInt(n[2]);
        String limite;
        if (rango.startsWith("~") || x == 0 && y > 0) limite = x + "." + (y + 1) + ".0";
        else if (x > 0) limite = (x + 1) + ".0.0";
        else limite = "0.0." + (z + 1);
        return comparar(version, base) >= 0 && comparar(version, limite) < 0;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<String> versiones = new ArrayList<>(Arrays.asList(sc.nextLine().trim().split(" ")));
        versiones.sort(Main::comparar);
        System.out.println("Ordenadas: " + String.join(" < ", versiones));
        while (sc.hasNextLine()) {
            String rango = sc.nextLine().trim();
            if (rango.isEmpty()) continue;
            String mejor = null;
            for (String v : versiones) if (cumple(v, rango) && !v.contains("-")) mejor = v; // la más alta estable
            System.out.println(rango + " → " + (mejor == null ? "ninguna" : mejor));
        }
    }
}
```

Comparar versiones como texto falla («1.10» < «1.9»), así que se separan y se comparan los tres números como enteros. Si coinciden, decide la etiqueta: sin etiqueta gana (una 2.0.0 es posterior a su 2.0.0-rc.1) y, entre dos etiquetas, se comparan parte a parte con `compararParte`, que distingue números de texto (por eso beta.2 < beta.11).

Cada rango se convierte en un intervalo [base, límite): el límite es la siguiente versión mayor para ^ (o la menor o el parche cuando las cifras de la izquierda son 0) y la siguiente menor para ~. Una vez calculado el límite como texto, basta con reutilizar `comparar`.

`Boolean.compare(!ea, !eb)` resume el caso de que alguna no tenga etiqueta: la que no la tiene vale `true`, y `true` es mayor que `false`.

107. ¿Puedo distribuir mi aplicación con estas librerías?

Difícil · Distribución de Aplicaciones · apuntesdam.com/subject/desarrollo-interfaces/topic/distribucion-aplicaciones

Solución (java)

```
import java.util.*;

public class Main {
    static final Set<String> PERMISIVAS = Set.of("MIT", "BSD-3-Clause", "Apache-2.0", "EPL-2.0");

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] proyecto = sc.nextLine().trim().split(" ");    // proyecto LICENCIA (MIT, Apache-2.0, GPL-3.0, Propietaria)
        String licProyecto = proyecto[1];
        List<String[]> deps = new ArrayList<>();                // nombre licencia
        enlace(estatico|dinamico)
        while (sc.hasNextLine()) { String l = sc.nextLine().trim(); if (!l.isEmpty()) deps.add(l.split(" ")); }

        int problemas = 0, avisos = 0;
        boolean proyectoGpl = licProyecto.startsWith("GPL");
        for (String[] d : deps) {
            String lic = d[1];
            boolean estatico = d[2].equals("estatico");
            String veredicto;
            if (PERMISIVAS.contains(lic)) {
                veredicto = lic.equals("Apache-2.0") && licProyecto.equals("GPL-2.0")
                    ? "PROBLEMA (Apache-2.0 no es compatible con GPL-2.0)"
                    : "OK (permisiva: incluye su aviso de copyright)";
            } else if (lic.equals("LGPL-3.0")) {
                if (!estatico) {
                    veredicto = "OK (LGPL enlazada dinámicamente: el usuario debe poder sustituirla)";
                    avisos++;
                } else if (proyectoGpl) {
                    veredicto = "OK (compatible con GPL)";
                } else {
                    veredicto = "PROBLEMA (LGPL enlazada estáticamente obliga a facilitar el código objeto para reenlazar)";
                }
            } else if (lic.startsWith("GPL")) {
                if (lic.equals(licProyecto)) veredicto = "OK (misma licencia)";
                else if (lic.equals("GPL-2.0") && licProyecto.equals("GPL-3.0")) veredicto = "PROBLEMA (GPL-2.0 sola no es compatible con GPL-3.0)";
                else veredicto = "PROBLEMA (copyright fuerte: obligaría a publicar el proyecto como " + lic + ")";
            } else if (lic.equals("AGPL-3.0")) {
                veredicto = "PROBLEMA (AGPL: también obliga a compartir el código si se ofrece como servicio en red)";
            } else {
                veredicto = "REVISAR (licencia desconocida: hay que leerla)";
                avisos++;
            }
            if (veredicto.startsWith("PROBLEMA")) problemas++;
            System.out.println(d[0] + " [" + lic + ", " + d[2] + "]: " + veredicto);
        }
        System.out.println("Resumen: " + problemas + " problemas, " + avisos + " avisos → "
            + (problemas == 0 ? "se puede distribuir" : "no distribuir sin resolverlos"));
    }
}
```

Las reglas se agrupan por familias de licencias (permisivas, LGPL, GPL, AGPL y desconocidas) en una cadena de `if/else if`, y dentro de cada familia se miran el tipo de enlace y la licencia del proyecto. Así cada regla del enunciado está en un solo sitio.

Los problemas no se cuentan en cada rama: basta con mirar al final si el veredicto empieza por «PROBLEMA». Los avisos sí se suman en las dos ramas que los producen. Las permisivas van en un `Set` para comprobarlas con `contains`.

108. Un actualizador incremental con despliegue escalonado

Muy difícil · Distribución de Aplicaciones · apuntesdam.com/subject/desarrollo-interfaces/topic/distribucion-aplicaciones

Solución (java)

```
import java.util.*;

public class Main {
    static int comparar(String a, String b) {
        String[] x = a.split("\\."), y = b.split("\\.");
        for (int i = 0; i < 3; i++) {
            int c = Integer.compare(Integer.parseInt(x[i]), Integer.parseInt(y[i]));
            if (c != 0) return c;
        }
        return 0;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, String> conf = new HashMap<>();
        Map<String, String[]> local = new TreeMap<>(), remoto = new TreeMap<>(); // ruta → {tamaño, hash}
        Map<String, String[]> actual = null;
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (l.isEmpty()) continue;
            if (l.equals("[local]")) { actual = local; continue; }
            if (l.equals("[remoto]")) { actual = remoto; continue; }
            String[] p = l.split(" ");
            if (actual == null) conf.put(p[0], p[1]); // version_instalada, version_nueva,
                minima, porcentaje, usuario
            else actual.put(p[0], new String[]{p[1], p[2]});
        }

        String instalada = conf.get("version_instalada"), nueva = conf.get("version_nueva"), minima =
            conf.get("minima");
        int porcentaje = Integer.parseInt(conf.get("porcentaje"));
        if (comparar(instalada, nueva) >= 0) {
            System.out.println("Ya tienes la última versión (" + instalada + ")");
            return;
        }
        boolean obligatoria = comparar(instalada, minima) < 0;
        int grupo = Math.floorMod(conf.get("usuario").hashCode(), 100);
        if (!obligatoria && grupo >= porcentaje) {
            System.out.println("Actualización " + nueva + " aún no disponible para este usuario (grupo " +
                grupo + ", se publica al " + porcentaje + "%)");
            return;
        }

        System.out.println("Actualizar de " + instalada + " a " + nueva
            + (obligatoria ? " (obligatoria: la versión mínima es " + minima + ")" : " (grupo " + grupo
            + ")"));
        long descarga = 0, total = 0;
        int ficheros = 0;
        for (Map.Entry<String, String[]> e : remoto.entrySet()) {
            long tam = Long.parseLong(e.getValue()[0]);
            total += tam;
            String[] antes = local.get(e.getKey());
            if (antes != null && antes[1].equals(e.getValue()[1])) continue; // mismo hash: no se descarga
            System.out.println(" " + (antes == null ? "NUEVO " : "CAMBIA ") + e.getKey());
            descarga += tam;
            ficheros++;
        }
        for (String ruta : local.keySet()) {
            if (!remoto.containsKey(ruta)) System.out.println(" BORRAR " + ruta);
        }
        System.out.printf(Locale.US, "Descarga: %d ficheros, %.1f KB de %.1f KB (ahorro del %.0f %%)\n",
            ficheros, descarga / 1024.0, total / 1024.0, 100.0 * (total - descarga) / total);
    }
}
```


Las versiones se comparan número a número, como en cualquier actualizador. El despliegue escalonado usa `Math.floorMod(usuario.hashCode(), 100)` para asignar a cada usuario un grupo fijo del 0 al 99: si el porcentaje publicado es 10, solo lo reciben los grupos del 0 al 9, y siempre los mismos usuarios. Se usa `floorMod` y no `%` porque `hashCode` puede ser negativo.

Para saber qué descargar se comparan los dos manifiestos por ruta: lo que no está en el local es nuevo, lo que está con otro hash ha cambiado y lo que solo está en el local hay que borrarlo. Como los dos mapas son `TreeMap`, se recorren ya en orden alfabético.

Los tamaños se suman en `long` y se convierten a KB al final dividiendo entre `1024.0` (con decimales). En `printf`, `%%` escribe el símbolo del porcentaje.

109. Casos de prueba por valores límite

Medio · Pruebas de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/pruebas-interfaces

Solución (java)

```
import java.util.*;

public class Main {
    static String texto(int n) {
        if (n == 0) return "\\\"";
        if (n > 12) return "texto de " + n + " caracteres";
        return "\\\" + \"a\".repeat(n) + \"\\\" (" + n + (n == 1 ? " carácter" : " caracteres"));
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");    // campo tipo min max
            if (p.length < 4) continue;
            String campo = p[0], tipo = p[1];
            int min = Integer.parseInt(p[2]), max = Integer.parseInt(p[3]);

            System.out.println("== " + campo + " (" + tipo + ", " + min + ".." + max + ")");
            List<String> casos = new ArrayList<>();
            // Valores límite, sin repetir y en orden
            Set<Integer> limites = new LinkedHashSet<>(List.of(min - 1, min, min + 1, max - 1, max, max + 1));
            for (int v : limites) {
                if (tipo.equals("texto") && v < 0) continue;
                String resultado = v >= min && v <= max ? "válido" : "inválido";
                casos.add((tipo.equals("entero") ? String.valueOf(v) : texto(v)) + " → " + resultado);
            }
            if (tipo.equals("entero")) {
                int medio = (min + max) / 2;
                if (!limites.contains(medio)) casos.add(medio + " → válido");
                casos.add("abc → inválido (no es un número)");
            } else {
                casos.add("\\\"  \" (solo espacios) → inválido (vacío tras recortar)");
            }
            casos.add("(vacío) → inválido (obligatorio)");

            for (int i = 0; i < casos.size(); i++) System.out.println("  " + campo + "-" + (i + 1) + ": " + casos.get(i));
        }
    }
}
```

Los seis valores límite se meten en un `LinkedHashSet`, que elimina los repetidos (en un rango 0..2, `min + 1` y `max - 1` son el mismo) y conserva el orden de inserción. Cada uno es válido si cae dentro del rango.

Los casos se acumulan en una lista y se numeran al final, así no hay que llevar la cuenta mientras se generan. La representación de un texto de longitud `n` está en su propio método, `texto`, con sus tres formatos.

Además de los límites, se añaden casos de clases de equivalencia inválidas que cualquier formulario debe rechazar: algo que no es un número, un texto que solo tiene espacios y el campo vacío.

110. Un mini ejecutor de pruebas unitarias

Difícil · Pruebas de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/pruebas-interfaces

Solución (java)

```
import java.util.*;

public class Main {
    static double precioFinal(double importe, String codigo, boolean socio) {
        if (importe < 0) throw new IllegalArgumentException("importe negativo");
        int descuento = 0;
        if (codigo.equals("VERANO10")) descuento = 10;
        else if (codigo.equals("VIP25") && importe >= 100) descuento = 25;
        if (socio) descuento += 5;
        descuento = Math.min(descuento, 30);
        double precio = importe * (100 - descuento) / 100;
        if (precio < 50 && importe != 0) precio += 4.95;
        return Math.round(precio * 100) / 100.0;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int pruebas = 0, pasan = 0, fallan = 0, errores = 0;
        // Cada línea: nombre_prueba importe codigo(o -) socio(si|no) esperado
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");
            if (p.length < 5) continue;
            pruebas++;
            String nombre = p[0], esperado = p[4];
            boolean esperaExcepcion = esperado.equals("excepcion");
            try {
                double obtenido = precioFinal(Double.parseDouble(p[1]), p[2], p[3].equals("si"));
                String texto = String.format(Locale.US, "%.2f", obtenido);
                if (esperaExcepcion) {
                    System.out.println("FALLA " + nombre + ": se esperaba una excepción y devolvió " +
                        texto);
                    fallan++;
                } else if (Math.abs(obtenido - Double.parseDouble(esperado)) < 0.005) {
                    System.out.println("PASA " + nombre);
                    pasan++;
                } else {
                    System.out.println("FALLA " + nombre + ": esperado " + esperado + ", obtenido " +
                        texto);
                    fallan++;
                }
            } catch (Exception e) {
                if (esperaExcepcion) {
                    System.out.println("PASA " + nombre + " (lanza " + e.getClass().getSimpleName() + ")");
                    pasan++;
                } else {
                    System.out.println("ERROR " + nombre + ": " + e.getClass().getSimpleName() + ": " +
                        e.getMessage());
                    errores++;
                }
            }
        }
        System.out.println("Pruebas: " + pruebas + " · Pasan: " + pasan + " · Fallan: " + fallan + " ·
            Errores: " + errores);
    }
}
```

`precioFinal` sigue la especificación paso a paso: valida, calcula el porcentaje de descuento (con el tope del 30 %), lo aplica, suma el envío y redondea a céntimos con `Math.round(x * 100) / 100.0`.

El ejecutor distingue tres resultados como JUnit: PASA si el valor coincide (o si se esperaba una excepción y se lanzó), FALLA si el resultado es otro, y ERROR si salta una excepción que nadie esperaba. Esa diferencia es útil: un fallo suele indicar un error de lógica y un error, un caso que el código no contempla.

Los números con decimales se comparan con una tolerancia (`Math.abs(a - b) < 0.005`) y nunca con `==`, porque los `double` arrastran pequeños errores de redondeo.

111. Pruebas de interfaz guionizadas

Muy difícil · Pruebas de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/pruebas-interfaces

Solución (java)

```
import java.util.*;

public class Main {
    // Modelo de una pantalla de acceso: campos usuario y clave, botón entrar y un mensaje
    static class PantallaAcceso {
        String usuario = "", clave = "", mensaje = "";
        boolean entrarHabilitado = false, dentro = false;
        int intentos = 0;
        static final Map<String, String> CUENTAS = Map.of("ana", "Segura123", "luis", "Clave2026");

        private void actualizarBoton() {
            entrarHabilitado = !usuario.isBlank() && clave.length() >= 4 && intentos < 3;
        }

        void escribir(String campo, String texto) {
            if (campo.equals("usuario")) usuario = texto;
            else if (campo.equals("clave")) clave = texto;
            actualizarBoton();
        }

        void pulsarEntrar() {
            if (!entrarHabilitado) return;
            String u = usuario.trim();
            if (clave.equals(CUENTAS.get(u.toLowerCase())) {
                dentro = true;
                mensaje = "Bienvenida, " + u;
                return;
            }
            clave = "";
            intentos++;
            mensaje = intentos >= 3 ? "Cuenta bloqueada: demasiados intentos"
                : "Usuario o contraseña incorrectos (" + (3 - intentos) + " intentos más)";
            actualizarBoton();
        }

        String consultar(String que) {
            return switch (que) {
                case "mensaje" -> mensaje;
                case "boton" -> entrarHabilitado ? "habilitado" : "deshabilitado";
                case "clave" -> clave;
                case "pantalla" -> dentro ? "principal" : "acceso";
                default -> "?";
            };
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        PantallaAcceso pantalla = new PantallaAcceso();
        int paso = 0, fallos = 0;
        while (sc.hasNextLine()) {
            String linea = sc.nextLine();
            if (linea.isBlank() || linea.trim().startsWith("#")) continue;
            paso++;
            String[] p = linea.split(" ", 3); // orden, objeto y el resto (que puede tener espacios)
            String valor = p.length > 2 ? p[2] : "";
            switch (p[0]) {
                case "escribe" -> pantalla.escribir(p[1], valor);
                case "pulsa" -> pantalla.pulsarEntrar();
                case "espera" -> {
                    String real = pantalla.consultar(p[1]);
                    if (real.equals(valor)) {
                        System.out.println("✓ paso " + paso + ": " + p[1] + " = «" + real + "»");
                    } else {
                        System.out.println("X paso " + paso + ": " + p[1] + " esperaba «" + valor + "» pero es «" + real + "»");
                        fallos++;
                    }
                }
            }
        }
    }
}
```

```

    }
    default -> System.out.println("? paso " + paso + ": orden desconocida «" + p[0] + "»");
  }
}
System.out.println(fallos == 0 ? "PRUEBA SUPERADA" : "PRUEBA FALLIDA (" + fallos + "
comprobaciones)");
}
}

```

Separar el modelo de la pantalla (su estado y sus reglas) de quien lo maneja es lo que permite probar una interfaz sin dibujarla: el guion solo escribe, pulsa y consulta, igual que haría un usuario. El método `actualizarBoton` concentra la regla de cuándo se habilita el botón y se llama después de cada cambio.

Cada línea se parte con `split(" ", 3)`: el tercer trozo es todo lo que queda, con sus espacios, que es lo que necesitan los mensajes. Por eso la línea no se recorta: en «espera clave» el valor esperado es la cadena vacía.

El método `consultar` traduce el estado interno (booleanos) a los valores del guion, de modo que comparar lo esperado con lo real es una simple comparación de cadenas.

112. Validar un NIF en un formulario

Fácil · Ejercicios de Desarrollo de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/ejercicios-desarrollo-interfaces

Solución (java)

```

import java.util.*;

public class Main {
    static final String LETRAS = "TRWAGMYFPDXBNJZSQVHLCKE";

    static boolean nifValido(String nif) {
        String s = nif.trim().replace(" ", "").replace("-", "").toUpperCase();
        if (!s.matches("\\d{8}[A-Z]")) return false;
        int numero = Integer.parseInt(s.substring(0, 8));
        return LETRAS.charAt(numero % 23) == s.charAt(8);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String l = sc.nextLine();
            if (!l.isBlank()) System.out.println(nifValido(l) ? "válido" : "no válido");
        }
    }
}

```

Primero se limpia el NIF (sin espacios ni guiones y en mayúsculas) y después se comprueba el formato con la expresión `\\d{8}[A-Z]`: exactamente ocho dígitos y una letra. Solo si el formato es correcto se calcula la letra de control, el resto de dividir el número entre 23 usado como posición en la cadena `LETRAS`.

113. Informe agrupado con subtotales

Medio · Ejercicios de Desarrollo de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/ejercicios-desarrollo-interfaces

interfaces

Solución (java)

```
import java.util.*;

public class Main {
    static String importe(double v) {
        return String.format(Locale.US, "%.2f", v).replace('.', ',');
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, Double> subtotales = new TreeMap<>(); // ordenado por categoría
        double total = 0;
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (l.isEmpty()) continue;
            String[] p = l.split(";");
            double v = Double.parseDouble(p[1]);
            subtotales.merge(p[0], v, Double::sum);
            total += v;
        }
        subtotales.forEach((cat, v) -> System.out.println(cat + ": " + importe(v)));
        System.out.println("TOTAL: " + importe(total));
    }
}
```

Un `TreeMap` agrupa y ordena a la vez: `merge(categoría, importe, Double::sum)` suma el importe al subtotal de su categoría (o lo crea), y al recorrerlo las categorías salen en orden alfabético.

Para la coma decimal se formatea con `Locale.US` (punto) y se cambia el punto por una coma, lo que da el mismo resultado en cualquier ordenador, sea cual sea su idioma.

114. Paginación de una tabla

Medio · Ejercicios de Desarrollo de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/ejercicios-desarrollo-interfaces

Solución (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int tam = sc.nextInt(), pagina = sc.nextInt();
        sc.nextLine();
        List<String> elementos = new ArrayList<>();
        while (sc.hasNextLine()) {
            String l = sc.nextLine();
            if (!l.isBlank()) elementos.add(l);
        }

        if (elementos.isEmpty()) {
            System.out.println("Sin resultados");
            return;
        }
        int paginas = (elementos.size() + tam - 1) / tam; // división redondeando hacia arriba
        if (pagina < 1 || pagina > paginas) {
            System.out.println("Página no válida");
            return;
        }
        int desde = (pagina - 1) * tam, hasta = Math.min(desde + tam, elementos.size());
        elementos.subList(desde, hasta).forEach(System.out::println);
        System.out.println("Página " + pagina + " de " + paginas);
    }
}
```

El número de páginas es la división redondeando hacia arriba: `(n + tam - 1) / tam` lo consigue con enteros (5 elementos en páginas de 3 son 2 páginas).

La página `p` empieza en el elemento `(p - 1) * tam` y termina `tam` elementos después, salvo en la última, que puede tener menos: por eso el final se limita con `Math.min`. `subList` devuelve justo ese trozo de la lista.

115. Comparar versiones semánticas

Medio · Ejercicios de Desarrollo de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/ejercicios-desarrollo-interfaces

Solución (java)

```
import java.util.*;

public class Main {

    static int comparar(String a, String b) {
        String[] x = a.split("\\."), y = b.split("\\.");
        for (int i = 0; i < 3; i++) {
            int c = Integer.compare(Integer.parseInt(x[i]), Integer.parseInt(y[i]));
            if (c != 0) return c;    // la primera parte distinta decide
        }
        return 0;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNext()) {
            int c = comparar(sc.next(), sc.next());
            System.out.println(c < 0 ? "<" : c > 0 ? ">" : "=");
        }
    }
}
```

`compareTo` compara texto carácter a carácter, y como «1» va antes que «9», dice que 2.10.0 es menor que 2.9.5. Hay que separar las tres partes por el punto (en `split` el punto se escapa, `\\.`, porque en una expresión regular significa «cualquier carácter») y compararlas como números.

La primera parte que es distinta decide el resultado; si las tres coinciden, las versiones son iguales.

116. Densidades de pantalla y recursos de Android

Medio · Tecnologías y Plataformas Móviles · apuntesdam.com/subject/prog-multimedia/topic/tecnologias-plataformas-movil

Solución (java)

```
import java.util.*;

public class Main {
    static final String[] NOMBRES = {"ldpi", "mdpi", "hdpi", "xhdpi", "xxhdpi", "xxxhdpi"};
    static final int[] DPI = {120, 160, 240, 320, 480, 640};

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" "); // nombre anchoPx altoPx pulgadas
            if (p.length < 4) continue;
            int ancho = Integer.parseInt(p[1]), alto = Integer.parseInt(p[2]);
            double dpi = Math.hypot(ancho, alto) / Double.parseDouble(p[3]);

            int g = 0; // grupo más cercano (en empate, el menor)
            for (int i = 1; i < DPI.length; i++) {
                if (Math.abs(dpi - DPI[i]) < Math.abs(dpi - DPI[g])) g = i;
            }
            double factor = DPI[g] / 160.0;
            String textoFactor = factor == Math.floor(factor) ? String.valueOf((long) factor) :
                String.valueOf(factor);

            long corto = Math.round(Math.min(ancho, alto) / factor), largo = Math.round(Math.max(ancho,
                alto) / factor);
            String tipo = corto >= 720 ? "tableta grande" : corto >= 600 ? "tableta" : "móvil";
            String carpeta = corto >= 720 ? "layout-sw720dp" : corto >= 600 ? "layout-sw600dp" : "layout";

            System.out.println(p[0] + ": " + Math.round(dpi) + " dpi → " + NOMBRES[g] + " (x" + textoFactor
                + "), "
                + corto + " x " + largo + " dp, " + tipo + " → " + carpeta + ", icono 48 dp = " +
                Math.round(48 * factor) + " px");
        }
    }
}
```

La densidad real es la diagonal en píxeles (Pitágoras, con `Math.hypot`) dividida entre la diagonal en pulgadas. Android no usa ese valor exacto: redondea al grupo más cercano, y el factor de ese grupo respecto a mdpi (160) es el que convierte dp en píxeles. Por eso un icono de 48 dp mide $48 \times \text{factor}$ píxeles.

Al dividir los píxeles entre el factor se obtiene el tamaño en dp, que es parecido en todos los móviles (unos 360 dp de ancho) aunque sus píxeles sean muy distintos. La anchura mínima en dp decide la carpeta de diseños: así una tableta puede usar una interfaz de dos columnas.

117. El ciclo de vida de una Activity

Difícil · Tecnologías y Plataformas Móviles · apuntesdam.com/subject/prog-multimedia/topic/tecnologias-plataformas-movil

Solución (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // Estados: NINGUNO, CREADA, INICIADA, REANUDADA, PAUSADA, DETENIDA, DESTRUIDA
        String estado = "NINGUNO", texto = "", guardado = null; // guardado = el Bundle de
        onSaveInstanceState
        while (sc.hasNextLine()) {
            String linea = sc.nextLine().trim();
            if (linea.isEmpty()) continue;
            String[] p = linea.split(" ", 2);
            List<String> llamadas = new ArrayList<>();
            switch (p[0]) {
                case "abrir" -> {
                    llamadas.addAll(List.of("onCreate", "onStart", "onResume"));
                    estado = "REANUDADA";
                    texto = "";
                }
                case "escribir" -> {
                    texto = p.length > 1 ? p[1] : "";
                    llamadas.add("(el usuario escribe «" + texto + "»)");
                }
                case "dialogo" -> {
                    if (estado.equals("REANUDADA")) { llamadas.add("onPause"); estado = "PAUSADA"; }
                }
                case "cerrar_dialogo" -> {
                    if (estado.equals("PAUSADA")) { llamadas.add("onResume"); estado = "REANUDADA"; }
                }
                case "home" -> {
                    if (estado.equals("REANUDADA")) llamadas.add("onPause");
                    llamadas.addAll(List.of("onStop", "onSaveInstanceState"));
                    guardado = texto;
                    estado = "DETENIDA";
                }
                case "volver" -> {
                    if (estado.equals("DETENIDA")) { llamadas.addAll(List.of("onRestart", "onStart",
                        "onResume")); estado = "REANUDADA"; }
                }
                case "girar" -> {
                    llamadas.addAll(List.of("onPause", "onStop", "onSaveInstanceState", "onDestroy",
                        "onCreate", "onStart", "onRestoreInstanceState", "onResume"));
                    guardado = texto; // se guarda antes de destruirla...
                    texto = guardado; // ...y se recupera en la nueva instancia
                    estado = "REANUDADA";
                }
                case "matar_proceso" -> {
                    if (estado.equals("DETENIDA")) {
                        llamadas.add("(proceso eliminado sin más llamadas)");
                        estado = "NINGUNO";
                        texto = ""; // lo que había en memoria se pierde; el Bundle no
                    }
                }
                case "reabrir" -> {
                    llamadas.addAll(List.of("onCreate", "onStart"));
                    if (guardado != null) llamadas.add("onRestoreInstanceState");
                    llamadas.add("onResume");
                    texto = guardado != null ? guardado : "";
                    estado = "REANUDADA";
                }
                case "atras" -> {
                    if (estado.equals("REANUDADA")) llamadas.add("onPause");
                    llamadas.addAll(List.of("onStop", "onDestroy"));
                    estado = "DESTRUIDA";
                    texto = "";
                    guardado = null; // el usuario la cerró: no se guarda nada
                }
                default -> llamadas.add("(evento desconocido)");
            }
        }
    }
}
```



```

    }
    if (llamadas.isEmpty()) llamadas.add("(sin llamadas)");
    System.out.println(p[0] + ": " + String.join(" → ", llamadas) + " | estado " + estado + " |
        texto «" + texto + "»");
    }
}
}

```

Se guardan tres cosas: el estado de la Activity, el texto que hay en memoria y el que se guardó en `onSaveInstanceState` (el «Bundle»). La diferencia entre los dos textos es la lección del ejercicio: lo que está solo en memoria se pierde si el sistema mata el proceso, y lo guardado en el Bundle se recupera al recrear la pantalla.

Cada evento añade a una lista las llamadas que haría Android, algunas solo si el estado lo permite (por ejemplo, `onPause` solo si la pantalla estaba reanudada). Un giro destruye y recrea la Activity: sin `onSaveInstanceState` el usuario perdería lo que estaba escribiendo. En cambio, al pulsar «atrás» el usuario cierra la pantalla a propósito y no se guarda nada.

118. Elegir el minSdk de una app

Muy difícil · Tecnologías y Plataformas Móviles · apuntesdam.com/subject/prog-multimedia/topic/tecnologias-plataformas-

Solución (java)

```
import java.util.*;

public class Main {
    static String numero(double v) {
        return v == Math.floor(v) ? String.valueOf((long) v) : String.valueOf(v);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        TreeMap<Integer, Double> cuota = new TreeMap<>(); // nivel de API → % de dispositivos
        List<String[]> apis = new ArrayList<>(); // nombre nivelMinimo
        int minSdk = 0;
        double objetivo = 0;
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");
            switch (p[0]) {
                case "cuota" -> cuota.put(Integer.parseInt(p[1]), Double.parseDouble(p[2]));
                case "api" -> apis.add(new String[]{p[1], p[2]});
                case "minSdk" -> minSdk = Integer.parseInt(p[1]);
                case "objetivo" -> objetivo = Double.parseDouble(p[1]);
                default -> { }
            }
        }

        double total = 0;
        for (double c : cuota.values()) total += c;
        final double suma = total;
        // Porcentaje (normalizado) de dispositivos con nivel en [desde, hasta)
        java.util.function.BiFunction<Integer, Integer, Double> entre = (desde, hasta) -> {
            double s = 0;
            for (double c : cuota.subMap(desde, true, hasta, false).values()) s += c;
            return s / suma * 100;
        };

        System.out.printf(Locale.US, "minSdk %d: %.1f %% de los dispositivos%n", minSdk, entre.apply(minSdk, Integer.MAX_VALUE));

        Integer recomendado = null;
        for (int nivel : cuota.descendingKeySet()) { // del más alto al más bajo
            if (entre.apply(nivel, Integer.MAX_VALUE) >= objetivo) { recomendado = nivel; break; }
        }
        if (recomendado == null) System.out.println("Ningún minSdk alcanza el " + numero(objetivo) + " %");
        else System.out.printf(Locale.US, "Recomendado para el %s %%: minSdk %d (%.1f %%)%n",
            numero(objetivo), recomendado, entre.apply(recomendado, Integer.MAX_VALUE));

        for (String[] api : apis) {
            int nivel = Integer.parseInt(api[1]);
            if (nivel <= minSdk) System.out.println(api[0] + ": disponible siempre (API " + nivel + ")");
            else System.out.printf(Locale.US, "%s: comprobar Build.VERSION.SDK_INT >= %d (no disponible en el %.1f %% de los dispositivos)%n",
                api[0], nivel, entre.apply(minSdk, nivel));
        }
    }
}
```

Todo se reduce a una operación: qué porcentaje de dispositivos tiene un nivel de API dentro de un intervalo. Con un `TreeMap`, `subMap(desde, true, hasta, false)` devuelve justo ese tramo de niveles, y dividir entre la suma total normaliza las cuotas aunque no sumen 100.

La cobertura de un `minSdk` es el tramo `[minSdk, ∞)`; el recomendado se busca recorriendo los niveles de mayor a menor (`descendingKeySet`) y quedándose con el primero que aún llega al objetivo. Para cada API más nueva que el `minSdk`, los dispositivos que no la tienen son los del tramo `[minSdk, nivel de la API)`: para ellos hace falta la comprobación de versión.

119. Las diferencias de una lista, como DiffUtil

Medio · Desarrollo de Aplicaciones Móviles · apuntesdam.com/subject/prog-multimedia/topic/desarrollo-aplicaciones-movil

Solución (java)

```
import java.util.*;

public class Main {
    record Elemento(String id, String contenido) { }

    static List<Elemento> leer(String linea) {
        List<Elemento> l = new ArrayList<>();
        if (linea.isBlank()) return l;
        for (String par : linea.trim().split(" ")) {
            String[] p = par.split(":");
            l.add(new Elemento(p[0], p[1]));
        }
        return l;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<Elemento> antes = leer(sc.nextLine()), despues = leer(sc.hasNextLine() ? sc.nextLine() : "");

        Map<String, String> viejos = new LinkedHashMap<>(), nuevos = new LinkedHashMap<>();
        for (Elemento e : antes) viejos.put(e.id(), e.contenido());
        for (Elemento e : despues) nuevos.put(e.id(), e.contenido());

        int borrados = 0, insertados = 0, modificados = 0, iguales = 0;
        for (Elemento e : antes) {
            if (!nuevos.containsKey(e.id())) {
                System.out.println("notifyItemRemoved: " + e.id());
                borrados++;
            }
        }
        for (int i = 0; i < despues.size(); i++) {
            Elemento e = despues.get(i);
            String viejo = viejos.get(e.id());
            if (viejo == null) {
                System.out.println("notifyItemInserted(" + i + "): " + e.id());
                insertados++;
            } else if (!viejo.equals(e.contenido())) {
                System.out.println("notifyItemChanged(" + i + "): " + e.id() + " " + viejo + " → " +
                    e.contenido());
                modificados++;
            } else {
                iguales++;
            }
        }

        // ¿Los que siguen en las dos listas mantienen su orden relativo?
        List<String> comunesAntes = new ArrayList<>(), comunesDespues = new ArrayList<>();
        for (Elemento e : antes) if (nuevos.containsKey(e.id())) comunesAntes.add(e.id());
        for (Elemento e : despues) if (viejos.containsKey(e.id())) comunesDespues.add(e.id());
        boolean movidos = !comunesAntes.equals(comunesDespues);

        System.out.println("Sin cambios: " + iguales + " · Insertados: " + insertados + " · Borrados: " +
            borrados
            + " · Modificados: " + modificados + (movidos ? " · Hay elementos movidos" : ""));
        if (insertados + modificados + borrados == 0 && !movidos) System.out.println("No hace falta
            redibujar");
        else System.out.println("Se redibujan " + (insertados + modificados) + " de " + despues.size()
            + " filas (notifyDataSetChanged redibujaría " + despues.size() + ")");
    }
}
```

La identidad de cada elemento es su id, no su posición: dos mapas id → contenido permiten saber en una consulta si un elemento es nuevo (no estaba antes), ha desaparecido (no está después) o ha cambiado (está en los dos con distinto contenido). Es la misma distinción que hace DiffUtil con `areItemsTheSame` y `areContentsTheSame`.

Para detectar movimientos se comparan las listas de ids que están en las dos versiones, cada una en su orden: si no coinciden, algún elemento ha cambiado de sitio. Solo las filas insertadas o modificadas necesitan dibujarse otra vez, que es lo que se ahorra frente a `notifyDataSetChanged`.

120. La pila de navegación de una app

Difícil · Desarrollo de Aplicaciones Móviles · apuntesdam.com/subject/prog-multimedia/topic/desarrollo-aplicaciones-movil

Solución (java)

```
import java.util.*;

public class Main {
    static String mostrar(Deque<String> pila) {
        List<String> l = new ArrayList<>();
        pila.descendingIterator().forEachRemaining(l::add);    // de la base a la cima
        return "[" + String.join(" > ", l) + "]";
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Deque<String> pila = new ArrayDeque<>();    // la cima es la pantalla visible
        pila.push(sc.nextLine().trim().split(" ")[1]);    // inicio DESTINO
        while (sc.hasNextLine()) {
            String orden = sc.nextLine().trim();
            if (orden.isEmpty()) continue;
            String[] p = orden.split(" ");
            if (p[0].equals("back")) {
                if (pila.size() == 1) {
                    System.out.println("back: se cierra la app desde " + pila.peek());
                    return;
                }
                pila.pop();
                System.out.println("back → " + mostrar(pila));
                continue;
            }
            // navigate DESTINO [popUpTo=X] [inclusive] [singleTop]
            String destino = p[1], hasta = null;
            boolean inclusive = false, singleTop = false;
            for (int i = 2; i < p.length; i++) {
                if (p[i].startsWith("popUpTo=")) hasta = p[i].substring(8);
                else if (p[i].equals("inclusive")) inclusive = true;
                else if (p[i].equals("singleTop")) singleTop = true;
            }
            if (hasta != null) {
                if (!pila.contains(hasta)) {
                    System.out.println("ERROR: " + hasta + " no está en la pila");
                    continue;
                }
                while (!pila.peek().equals(hasta)) pila.pop();
                if (inclusive) pila.pop();
            }
            if (singleTop && destino.equals(pila.peek())) {
                System.out.println(orden + ": ya estás en " + destino + " (singleTop)");
                continue;
            }
            pila.push(destino);
            System.out.println(orden + " → " + mostrar(pila));
        }
    }
}
```

La pila de navegación es un `Deque` usado como pila: `push` apila la nueva pantalla, `pop` la quita y `peek` dice cuál se ve. Para mostrarla de la base a la cima se recorre al revés con `descendingIterator`.

`popUpTo` desapila hasta que `x` queda arriba (y, con `inclusive`, también `x`) antes de apilar el destino. Así, tras iniciar sesión, la pantalla de acceso desaparece de la pila y «atrás» no vuelve a ella. `singleTop` evita apilar dos veces la misma pantalla si ya es la visible.

121. Sincronización sin conexión con conflictos

Muy difícil · Desarrollo de Aplicaciones Móviles · apuntesdam.com/subject/prog-multimedia/topic/desarrollo-aplicaciones-

Solución (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, String[]> servidor = new TreeMap<>(), local = new TreeMap<>(); // id → {valor, marca de tiempo}
        List<String[]> pendientes = new ArrayList<>(); // cambios hechos sin conexión
        String seccion = "";
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (l.isEmpty()) continue;
            if (l.startsWith("[") ) { seccion = l; continue; }
            String[] p = l.split(" ");
            switch (seccion) {
                case "[servidor]" -> servidor.put(p[0], new String[]{p[1], p[2]});
                case "[base]" -> local.put(p[0], new String[]{p[1], p[2]});
                case "[cambios]" -> pendientes.add(p); // op id valor tiempo (op: crear, editar, borrar)
                default -> { }
            }
        }

        int aplicados = 0, conflictos = 0;
        for (String[] c : pendientes) {
            String op = c[0], id = c[1];
            String valor = op.equals("borrar") ? null : c[2];
            long tiempo = Long.parseLong(c[c.length - 1]);
            String[] actual = servidor.get(id);

            if (op.equals("crear")) {
                if (actual != null) {
                    System.out.println("CONFLICTO " + id + ": ya existe en el servidor (se conserva «" + actual[0] + "»");
                    conflictos++;
                } else {
                    servidor.put(id, new String[]{valor, String.valueOf(tiempo)});
                    System.out.println("CREADO " + id + " = «" + valor + "»");
                    aplicados++;
                }
                continue;
            }
            if (actual == null) {
                System.out.println("IGNORADO " + id + ": ya no existe en el servidor");
                continue;
            }
            String[] base = local.get(id);
            boolean cambioEnServidor = base == null || !base[1].equals(actual[1]);
            long tiempoServidor = Long.parseLong(actual[1]);
            if (cambioEnServidor && tiempoServidor > tiempo) { // gana el último
                System.out.println("CONFLICTO " + id + ": gana el servidor «" + actual[0] + "» (" + tiempoServidor + ") frente al cambio local (" + tiempo + ")");
                conflictos++;
                continue;
            }
            if (op.equals("borrar")) {
                servidor.remove(id);
                System.out.println("BORRADO " + id + (cambioEnServidor ? " (sobre un cambio del servidor más antiguo)" : ""));
            } else {
                servidor.put(id, new String[]{valor, String.valueOf(tiempo)});
                System.out.println("EDITADO " + id + " = «" + valor + "»" + (cambioEnServidor ? " (gana el cambio local, más reciente)" : ""));
            }
            aplicados++;
        }

        System.out.println("Servidor tras sincronizar:");
    }
}
```

```

        servidor.forEach((id, v) -> System.out.println(" " + id + " = «" + v[0] + "» (" + v[1] + ")");
        System.out.println("Aplicados: " + aplicados + " · Conflictos: " + conflictos);
    }
}

```

La clave está en la versión base: comparando la marca de tiempo del servidor con la que tenía el móvil al quedarse sin conexión se sabe si alguien más ha cambiado ese elemento mientras tanto. Si no ha cambiado, el cambio local se aplica sin más; si ha cambiado, hay que resolver el conflicto.

La estrategia «gana el último» compara las marcas de tiempo: si el cambio del servidor es más reciente, se conserva y se descarta el local. Los casos que no tienen sentido (crear algo que ya existe, editar algo que ya se borró) se detectan antes, mirando si el elemento está en el servidor.

122. Física con paso fijo, acumulador e interpolación

Medio · Motores de Juegos · apuntesdam.com/subject/prog-multimedia/topic/motores-juegos

Solución (java)

```

import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int paso = Integer.parseInt(sc.nextLine().trim().split(" ")[1]);           // «paso 20» (ms)
        double velocidad = Double.parseDouble(sc.nextLine().trim().split(" ")[1]); // «velocidad 120»
                                           (px/s)
        String[] deltas = sc.nextLine().trim().split(" ");                       // ms de cada
                                           fotograma
        final int MAXIMO = 250;                                                  // evita la «espiral
                                           de la muerte»
        double x = 0, xAnterior = 0;
        int acumulado = 0, totalPasos = 0;

        for (int f = 0; f < deltas.length; f++) {
            int dt = Integer.parseInt(deltas[f]);
            acumulado += Math.min(dt, MAXIMO);
            int pasos = 0;
            while (acumulado >= paso) {                                           // tantos pasos fijos como quepan
                xAnterior = x;
                x += velocidad * paso / 1000.0;
                acumulado -= paso;
                pasos++;
            }
            totalPasos += pasos;
            double alfa = (double) acumulado / paso; // cuánto llevamos del siguiente paso
            double dibujo = xAnterior + (x - xAnterior) * alfa;
            System.out.printf(Locale.US, "F%d dt=%d%s: %d %s, sobran %d ms, x=%.1f, se dibuja en %.1f (alfa
                %.2f)%n",
                f + 1, dt, dt > MAXIMO ? " (limitado a " + MAXIMO + ")" : "", pasos, pasos == 1 ? "paso"
                : "pasos",
                acumulado, x, dibujo, alfa);
        }
        System.out.println("Fotogramas: " + deltas.length + " · Pasos de física: " + totalPasos + " · Tiempo
            simulado: " + totalPasos * paso + " ms");
    }
}

```

El acumulador guarda el tiempo real que la física todavía no ha simulado. Cada fotograma le suma su duración y se ejecutan pasos fijos mientras quepa uno entero; lo que sobra se queda para el siguiente fotograma. Así la física avanza siempre igual, vaya el juego a 30 o a 144 FPS.

Como la posición de la física solo cambia en saltos de un paso, se dibuja interpolando entre la posición anterior y la actual con `alfa = acumulado / paso`: el movimiento se ve suave aunque la física vaya a otro ritmo. El tope de 250 ms evita que, tras un parón, el juego tenga que simular tantos pasos que vuelva a atascarse.

123. Colisiones con rejilla espacial

Difícil · Motores de Juegos · apuntesdam.com/subject/prog-multimedia/topic/motores-juegos

Solución (java)

```
import java.util.*;

public class Main {
    record Caja(String nombre, int x, int y, int ancho, int alto) { }

    static boolean chocan(Caja a, Caja b) {
        return a.x() < b.x() + b.ancho() && a.x() + a.ancho() > b.x()
            && a.y() < b.y() + b.alto() && a.y() + a.alto() > b.y();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int celda = Integer.parseInt(sc.nextLine().trim().split(" ")[1]); // «celda 64»
        List<Caja> cajas = new ArrayList<>();
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");
            if (p.length < 5) continue;
            cajas.add(new Caja(p[0], Integer.parseInt(p[1]), Integer.parseInt(p[2]), Integer.parseInt(p[3]),
                Integer.parseInt(p[4])));
        }
        int n = cajas.size();

        // Fuerza bruta: todos los pares
        Set<String> brutos = new LinkedHashSet<>();
        for (int i = 0; i < n; i++)
            for (int j = i + 1; j < n; j++)
                if (chocan(cajas.get(i), cajas.get(j))) brutos.add(i + "-" + j);

        // Rejilla: cada caja se apunta en todas las celdas que toca
        Map<String, List<Integer>> rejilla = new LinkedHashMap<>();
        for (int i = 0; i < n; i++) {
            Caja c = cajas.get(i);
            for (int cx = Math.floorDiv(c.x(), celda); cx <= Math.floorDiv(c.x() + c.ancho() - 1, celda);
                cx++)
                for (int cy = Math.floorDiv(c.y(), celda); cy <= Math.floorDiv(c.y() + c.alto() - 1, celda);
                    cy++)
                    rejilla.computeIfAbsent(cx + "," + cy, k -> new ArrayList<>()).add(i);
        }
        Set<List<Integer>> candidatos = new HashSet<>(); // pares distintos que comparten alguna celda
        String masLlena = null;
        for (Map.Entry<String, List<Integer>> e : rejilla.entrySet()) {
            List<Integer> l = e.getValue();
            if (masLlena == null || l.size() > rejilla.get(masLlena).size()) masLlena = e.getKey();
            for (int a = 0; a < l.size(); a++)
                for (int b = a + 1; b < l.size(); b++) candidatos.add(List.of(l.get(a), l.get(b)));
        }
        List<int[]> choques = new ArrayList<>();
        for (List<Integer> par : candidatos)
            if (chocan(cajas.get(par.get(0)), cajas.get(par.get(1)))) choques.add(new int[]{par.get(0),
                par.get(1)});
        choques.sort(Comparator.<int[]>comparingInt(c -> c[0]).thenComparingInt(c -> c[1]));

        List<String> textos = new ArrayList<>(), claves = new ArrayList<>();
        for (int[] c : choques) {
            textos.add(cajas.get(c[0]).nombre() + "-" + cajas.get(c[1]).nombre());
            claves.add(c[0] + "-" + c[1]);
        }
        long total = (long) n * (n - 1) / 2;
        System.out.println("Celdas ocupadas: " + rejilla.size() + " · La más llena: ("
            + (masLlena == null ? "-" : masLlena) + ") con " + (masLlena == null ? 0 :
                rejilla.get(masLlena).size()));
        System.out.println("Choques: " + (textos.isEmpty() ? "ninguno" : String.join(" ", textos)));
        System.out.println("Comprobaciones: fuerza bruta " + total + " · rejilla " + candidatos.size()
            + " (ahorro " + (total == 0 ? 0 : Math.round(100.0 * (total - candidatos.size()) / total)) +
                "%)");
        System.out.println("Mismo resultado: " + (new HashSet<>(claves).equals(brutos) ? "sí" : "no"));
    }
}
```

La fase amplia reparte las cajas en celdas: cada caja se apunta en todas las celdas que toca, calculadas con `Math.floorDiv` (que, a diferencia de `/`, redondea hacia abajo también con coordenadas negativas). Después solo se comprueban los pares que comparten celda; dos cajas en celdas distintas no pueden tocarse.

Un par puede compartir varias celdas (dos cajas grandes), así que los candidatos se guardan en un `Set` para no comprobarlos dos veces; su tamaño es el número de comprobaciones. La fuerza bruta se calcula también para demostrar que la rejilla encuentra exactamente los mismos choques con muchas menos comprobaciones.

124. El efecto túnel: detección discreta frente a continua

Muy difícil · Motores de Juegos · apuntesdam.com/subject/prog-multimedia/topic/motores-juegos

Solución (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] m = sc.nextLine().trim().split(" "); // «pared x y ancho alto»
        double px = Double.parseDouble(m[1]), py = Double.parseDouble(m[2]);
        double pw = Double.parseDouble(m[3]), ph = Double.parseDouble(m[4]);
        int fps = Integer.parseInt(sc.nextLine().trim().split(" ")[1]); // «fps 60»
        int tuneles = 0, balas = 0;
        while (sc.hasNextLine()) {
            String[] b = sc.nextLine().trim().split(" ");
            if (b.length < 5) continue;
            String nombre = b[0];
            double x0 = Double.parseDouble(b[1]), y0 = Double.parseDouble(b[2]);
            double vx = Double.parseDouble(b[3]), vy = Double.parseDouble(b[4]);
            balas++;

            // Discreta: solo se mira la posición al final de cada fotograma
            String discreta = "no detecta nada";
            for (int k = 1; k <= fps; k++) {
                double x = x0 + vx * k / fps, y = y0 + vy * k / fps;
                if (x > px && x < px + pw && y > py && y < py + ph) {
                    discreta = String.format(Locale.US, "choca en el fotograma %d (t=%.3f s)", k, (double) k / fps);
                    break;
                }
            }

            // Continua: método de las franjas sobre el segmento recorrido en [0, 1]
            String continua;
            boolean impacto = false;
            if (x0 > px && x0 < px + pw && y0 > py && y0 < py + ph) {
                continua = "empieza dentro de la pared";
            } else {
                double[] fx = franja(x0, vx, px, px + pw), fy = franja(y0, vy, py, py + ph);
                double entrada = fx == null || fy == null ? 0 : Math.max(fx[0], fy[0]);
                double salida = fx == null || fy == null ? 0 : Math.min(fx[1], fy[1]);
                if (fx == null || fy == null || entrada > salida || entrada < 0 || entrada > 1) {
                    continua = "no hay impacto";
                } else {
                    impacto = true;
                    String cara = fx[0] >= fy[0] ? (vx > 0 ? "izquierda" : "derecha") : (vy > 0 ? "superior" : "inferior");
                    continua = String.format(Locale.US, "impacto en t=%.3f s en (%.1f, %.1f) por la cara %s",
                        entrada, x0 + vx * entrada, y0 + vy * entrada, cara);
                }
            }
            boolean tunel = impacto && discreta.equals("no detecta nada");
            if (tunel) tuneles++;
            System.out.println(nombre + ": discreta → " + discreta + " | continua → " + continua + (tunel ? " ¡ATRAVIESA LA PARED!" : ""));
        }
        System.out.println("Efecto túnel: " + tuneles + " de " + balas + " balas");
    }

    // Intervalo de tiempo [entrada, salida] en que la coordenada está dentro de (min, max); null si nunca
    static double[] franja(double p0, double v, double min, double max) {
        if (v == 0) {
            return p0 > min && p0 < max ? new double[]{Double.NEGATIVE_INFINITY, Double.POSITIVE_INFINITY} : null;
        }
        double t1 = (min - p0) / v, t2 = (max - p0) / v;
        return new double[]{Math.min(t1, t2), Math.max(t1, t2)};
    }
}
```

La detección discreta solo mira dónde está la bala al final de cada fotograma; si la pared es más fina que lo que avanza la bala en un fotograma, puede saltársela entera. La continua mira el segmento completo que recorre la bala.

El método de las franjas calcula, en cada eje, el intervalo de tiempo en que la bala está entre los dos bordes de la pared: $(\text{borde} - p_0) / v$. La bala está dentro cuando está a la vez en las dos franjas, así que el impacto es el último instante de entrada (`max` de las entradas) siempre que sea anterior al primer instante de salida y esté dentro del segundo simulado. El eje que entra más tarde es la cara por la que choca.

Si la velocidad en un eje es 0, la franja de ese eje es «siempre» o «nunca», según la bala ya esté o no entre sus bordes; el método `franja` devuelve un intervalo infinito o `null`.

125. ¿Quién gana al tres en raya?

Fácil · Ejercicios de Programación Multimedia y Dispositivos Móviles · apuntesdam.com/subject/programa-multimedia/topic/ejercicios-pmdm

Solución (java)

```
import java.util.*;

public class Main {

    static boolean linea(char[][] t, char j, int f0, int c0, int df, int dc) {
        for (int k = 0; k < 3; k++) if (t[f0 + k * df][c0 + k * dc] != j) return false;
        return true;
    }

    static boolean gana(char[][] t, char j) {
        for (int i = 0; i < 3; i++) {
            if (linea(t, j, i, 0, 0, 1) || linea(t, j, 0, i, 1, 0)) return true; // fila i o columna i
        }
        return linea(t, j, 0, 0, 1, 1) || linea(t, j, 0, 2, 1, -1); // diagonales
    }

    static String resultado(char[][] t) {
        if (gana(t, 'X')) return "Gana X";
        if (gana(t, 'O')) return "Gana O";
        for (char[] fila : t) for (char c : fila) if (c == '.') return "Sigue la partida";
        return "Empate";
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        char[][] t = new char[3][3];
        for (int i = 0; i < 3; i++) t[i] = sc.next().toCharArray();
        System.out.println(resultado(t));
    }
}
```

Las ocho líneas ganadoras (tres filas, tres columnas y dos diagonales) son todas «tres casillas a partir de una, avanzando en una dirección». El método `linea` lo comprueba con un punto de partida y un desplazamiento (`df`, `dc`), y así no hay que escribir ocho condiciones distintas.

Si nadie gana, la partida sigue mientras quede alguna casilla vacía; si no queda ninguna, es empate.

126. Una generación del juego de la vida

Medio · Ejercicios de Programación Multimedia y Dispositivos Móviles · apuntesdam.com/subject/programa-multimedia/topic/ejercicios-pmdm

Solución (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int f = sc.nextInt(), c = sc.nextInt();
        char[][] t = new char[f][];
        for (int i = 0; i < f; i++) t[i] = sc.next().toCharArray();

        char[][] nueva = new char[f][c];
        for (int i = 0; i < f; i++) {
            for (int j = 0; j < c; j++) {
                int vecinas = 0;
                for (int di = -1; di <= 1; di++)
                    for (int dj = -1; dj <= 1; dj++) {
                        int a = i + di, b = j + dj;
                        if ((di != 0 || dj != 0) && a >= 0 && a < f && b >= 0 && b < c && t[a][b] == '#')
                            vecinas++;
                    }
                boolean viva = t[i][j] == '#';
                nueva[i][j] = (viva && (vecinas == 2 || vecinas == 3)) || (!viva && vecinas == 3) ? '#' :
                    '.';
            }
        }
        for (char[] fila : nueva) System.out.println(new String(fila));
    }
}
```

La siguiente generación se escribe en una matriz nueva: todas las células cambian a la vez, y si se modificara el tablero mientras se recorre, las células ya cambiadas alterarían el recuento de sus vecinas.

Las ocho vecinas se cuentan con dos bucles de -1 a 1 , saltando la propia célula y las posiciones fuera del tablero (los bordes no se conectan). Las reglas se escriben en una sola expresión: viva con 2 o 3 vecinas, o muerta con exactamente 3.

127. Puntuación con combos

Medio · Ejercicios de Programación Multimedia y Dispositivos Móviles · apuntesdam.com/subject/prog-

Solución (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int puntos = 0, vidas = 3, combo = 0, mejor = 0;
        while (sc.hasNext()) {
            String evento = sc.next();
            if (vidas == 0) break; // partida terminada: se ignora el resto
            switch (evento) {
                case "MONEDA", "ENEMIGO" -> {
                    combo++;
                    mejor = Math.max(mejor, combo);
                    int multiplicador = combo >= 10 ? 3 : combo >= 5 ? 2 : 1;
                    puntos += (evento.equals("MONEDA") ? 10 : 50) * multiplicador;
                }
                case "GOLPE" -> {
                    vidas--;
                    combo = 0;
                }
                default -> { }
            }
        }
        System.out.println("Puntos: " + puntos);
        System.out.println("Vidas: " + vidas);
        System.out.println("Mejor combo: " + mejor);
    }
}
```

Cada evento que puntúa incrementa primero el combo y después calcula el multiplicador con él, como pide el enunciado: por eso la quinta moneda seguida ya vale el doble. El mejor combo se actualiza con `Math.max` en el mismo momento.

La comprobación de las vidas va al principio del bucle: en cuanto llegan a 0, `break` termina la partida y los eventos posteriores no cuentan.

128. Camino más corto en un laberinto

Difícil · Ejercicios de Programación Multimedia y Dispositivos Móviles · apuntesdam.com/subject/prog-

Solución (java)

```

import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int f = sc.nextInt(), c = sc.nextInt();
        char[][] m = new char[f][];
        for (int i = 0; i < f; i++) m[i] = sc.next().toCharArray();

        int[][] dist = new int[f][c];
        for (int[] fila : dist) Arrays.fill(fila, -1); // -1 = sin visitar
        Deque<int[]> cola = new ArrayDeque<>();
        for (int i = 0; i < f; i++)
            for (int j = 0; j < c; j++)
                if (m[i][j] == 'E') { dist[i][j] = 0; cola.add(new int[]{i, j}); }

        int[][] dirs = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};
        while (!cola.isEmpty()) {
            int[] p = cola.poll();
            if (m[p[0]][p[1]] == 'J') {
                System.out.println(dist[p[0]][p[1]]);
                return;
            }
            for (int[] d : dirs) {
                int a = p[0] + d[0], b = p[1] + d[1];
                if (a >= 0 && a < f && b >= 0 && b < c && m[a][b] != '#' && dist[a][b] == -1) {
                    dist[a][b] = dist[p[0]][p[1]] + 1;
                    cola.add(new int[]{a, b});
                }
            }
        }
        System.out.println("Sin camino");
    }
}

```

La búsqueda en anchura (BFS) explora el mapa por capas de distancia: primero las casillas a 1 paso de E, luego las que están a 2, etc. Por eso, la primera vez que saca a J de la cola, la distancia guardada es la mínima.

La matriz `dist` sirve a la vez para marcar las casillas visitadas (-1 si no lo están) y para guardar a cuántos pasos están. Si la cola se vacía sin encontrar a J, no hay camino.

129. Coordinador de procesos

Difícil · Programación Concurrente · apuntesdam.com/subject/prog-servicios/topic/concurrencia

Solución (java)

```
import java.io.*;
import java.nio.file.Path;
import java.util.*;

public class Main {
    static boolean esPrimo(long n) {
        if (n < 2) return false;
        for (long d = 2; d * d <= n; d++) if (n % d == 0) return false;
        return true;
    }

    static long contar(long desde, long hasta) {
        long c = 0;
        for (long i = desde; i <= hasta; i++) if (esPrimo(i)) c++;
        return c;
    }

    public static void main(String[] args) throws Exception {
        // Modo trabajador: el proceso hijo cuenta su tramo y escribe el resultado
        if (args.length == 3 && args[0].equals("trabajador")) {
            System.out.println(contar(Long.parseLong(args[1]), Long.parseLong(args[2])));
            return;
        }
        Scanner sc = new Scanner(System.in);
        long limite = sc.nextLong();
        int procesos = sc.nextInt();

        // El mismo java que nos ejecuta y el mismo classpath, para lanzar esta clase en otro proceso
        String java = Path.of(System.getProperty("java.home"), "bin", "java").toString();
        String classpath = System.getProperty("java.class.path");
        long tamano = limite / procesos;
        List<Process> hijos = new ArrayList<>();
        List<long[]> tramos = new ArrayList<>();
        for (int i = 0; i < procesos; i++) {
            long desde = i * tamano + 1;
            long hasta = i == procesos - 1 ? limite : (i + 1) * tamano; // el último se queda con el resto
            tramos.add(new long[]{desde, hasta});
            ProcessBuilder pb = new ProcessBuilder(java, "-cp", classpath, "Main", "trabajador",
                String.valueOf(desde), String.valueOf(hasta));
            pb.redirectErrorStream(true);
            hijos.add(pb.start());
        }

        long total = 0;
        for (int i = 0; i < procesos; i++) {
            Process hijo = hijos.get(i);
            long n;
            try (BufferedReader br = new BufferedReader(new InputStreamReader(hijo.getInputStream()))) {
                n = Long.parseLong(br.readLine().trim());
            }
            hijo.waitFor();
            total += n;
            System.out.println("Proceso " + (i + 1) + " [" + tramos.get(i)[0] + ", " + tramos.get(i)[1] +
                "]: " + n + " primos");
        }
        System.out.println("Total: " + total + " primos hasta " + limite);
    }
}
```

ProcessBuilder lanza un programa como proceso independiente. Para que el hijo sea esta misma clase se usa el mismo ejecutable de Java (`java.home`) y el mismo classpath, y se le pasan los argumentos «trabajador desde hasta», que hacen que el **main** entre en el modo trabajador.

Lanzar todos los procesos antes de leer ninguno es lo que permite que trabajen a la vez: si se lanzara uno, se esperara su resultado y después se lanzara el siguiente, irían uno tras otro. La salida estándar de cada hijo se lee con **getInputStream()** y **waitFor()** espera a que termine.

El reparto usa la división entera: cada tramo mide `limite / procesos` y el último llega hasta el límite, quedándose con lo que sobra.

130. Planificador de CPU: FCFS, SJF y Round Robin

Muy difícil · Programación Concurrente · apuntesdam.com/subject/prog-servicios/topic/concurrencia

Solución (java)

```
import java.util.*;

public class Main {
    record Proceso(String id, int llegada, int rafaga) { }

    // Procesos por orden de llegada (a igualdad, el de la entrada: sort es estable)
    static List<Proceso> porLlegada(List<Proceso> ps) {
        List<Proceso> l = new ArrayList<>(ps);
        l.sort(Comparator.comparingInt(Proceso::llegada));
        return l;
    }

    // Devuelve la línea de tiempo: una letra (id) por unidad de tiempo, o '-' si la CPU está libre
    static String fcfs(List<Proceso> ps) {
        StringBuilder sb = new StringBuilder();
        for (Proceso p : porLlegada(ps)) {
            while (sb.length() < p.llegada()) sb.append('-'); // CPU libre hasta que llega
            sb.append(p.id().repeat(p.rafaga()));
        }
        return sb.toString();
    }

    static String sjf(List<Proceso> ps) {
        StringBuilder sb = new StringBuilder();
        List<Proceso> pendientes = new ArrayList<>(ps);
        while (!pendientes.isEmpty()) {
            int t = sb.length();
            Proceso elegido = null;
            for (Proceso p : pendientes) {
                if (p.llegada() <= t && (elegido == null || p.rafaga() < elegido.rafaga())) elegido = p;
            }
            if (elegido == null) { sb.append('-'); continue; } // nadie ha llegado todavía
            sb.append(elegido.id().repeat(elegido.rafaga())); // no expropiativo: hasta el final
            pendientes.remove(elegido);
        }
        return sb.toString();
    }

    static String roundRobin(List<Proceso> ps, int quantum) {
        StringBuilder sb = new StringBuilder();
        List<Proceso> llegadas = porLlegada(ps);
        Map<Proceso, Integer> resto = new HashMap<>();
        for (Proceso p : ps) resto.put(p, p.rafaga());
        Deque<Proceso> cola = new ArrayDeque<>();
        int siguiente = 0;
        while (siguiente < llegadas.size() || !cola.isEmpty()) {
            while (siguiente < llegadas.size() && llegadas.get(siguiente).llegada() <= sb.length())
                cola.add(llegadas.get(siguiente++));
            if (cola.isEmpty()) { sb.append('-'); continue; }
            Proceso p = cola.poll();
            int turno = Math.min(quantum, resto.get(p));
            sb.append(p.id().repeat(turno));
            resto.put(p, resto.get(p) - turno);
            // los que han llegado durante el turno entran antes que el expulsado
            while (siguiente < llegadas.size() && llegadas.get(siguiente).llegada() <= sb.length())
                cola.add(llegadas.get(siguiente++));
            if (resto.get(p) > 0) cola.add(p);
        }
        return sb.toString();
    }

    // Calcula y muestra los tiempos de espera a partir de la línea de tiempo
    static void informe(String nombre, String linea, List<Proceso> ps) {
        StringBuilder sb = new StringBuilder(nombre + ": " + linea + " |");
        double suma = 0;
        for (Proceso p : ps) {
            int fin = linea.lastIndexOf(p.id().charAt(0)) + 1;
            int espera = fin - p.llegada() - p.rafaga();
            suma += espera;
        }
        System.out.println(nombre + " suma de esperas: " + suma);
    }
}
```

```

        suma += espera;
        sb.append(" ").append(p.id()).append(" ").append(espera);
    }
    sb.append(String.format(Locale.US, " | media %.2f", suma / ps.size()));
    System.out.println(sb);
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int quantum = sc.nextInt();
    List<Proceso> ps = new ArrayList<>();
    while (sc.hasNext()) ps.add(new Proceso(sc.next(), sc.nextInt(), sc.nextInt()));
    informe("FCFS", fcfs(ps), ps);
    informe("SJF ", sjf(ps), ps);
    informe("RR  ", roundRobin(ps, quantum), ps);
}
}

```

En los tres algoritmos la longitud de la línea de tiempo es el reloj: `sb.length()` dice en qué instante estamos, y una CPU libre se representa añadiendo un «-» y avanzando una unidad.

FCFS ordena por llegada (el `sort` de Java es estable, así que respeta el orden de la entrada en los empates) y ejecuta cada proceso entero. SJF, cada vez que la CPU queda libre, busca entre los que ya han llegado el de ráfaga más corta; como compara con `<`, en un empate se queda el primero.

Round Robin usa una cola: el proceso de la cabeza se ejecuta como mucho un quantum y, si le queda ráfaga, vuelve al final. El orden de esas dos operaciones decide el resultado: primero entran en la cola los procesos que han llegado durante el turno y después el expulsado.

131. Suma en paralelo con hilos

Difícil · Hilos y Multithreading en Java · apuntesdam.com/subject/prog-servicios/topic/hilos

Solución (java)

```

import java.util.*;

public class Main {
    public static void main(String[] args) throws InterruptedException {
        Scanner sc = new Scanner(System.in);
        int hilos = sc.nextInt();
        List<Long> lista = new ArrayList<>();
        while (sc.hasNextLong()) lista.add(sc.nextLong());
        long[] datos = lista.stream().mapToLong(Long::longValue).toArray();

        long[] parciales = new long[hilos]; // cada hilo escribe solo en su posición
        Thread[] trabajadores = new Thread[hilos];
        int base = datos.length / hilos, extra = datos.length % hilos;
        for (int i = 0; i < hilos; i++) {
            final int id = i;
            final int desde = i * base + Math.min(i, extra); // los primeros bloques tienen uno más
            final int hasta = desde + base + (i < extra ? 1 : 0);
            trabajadores[i] = new Thread(() -> {
                long suma = 0;
                for (int k = desde; k < hasta; k++) suma += datos[k];
                parciales[id] = suma;
            });
            trabajadores[i].start();
        }
        for (Thread t : trabajadores) t.join(); // esperar a que terminen todos

        long total = 0;
        for (int i = 0; i < hilos; i++) {
            System.out.println("Hilo " + i + ": " + parciales[i]);
            total += parciales[i];
        }
        System.out.println("Total: " + total);
    }
}

```

Cada bloque mide `n / hilos` elementos, y los `n % hilos` primeros se llevan uno más. El inicio del bloque `i` es `i * base + min(i, extra)`: los bloques anteriores ocupan `base` cada uno más uno extra por cada uno de ellos que lo tenía.

Los hilos no comparten nada que puedan pisarse: leen el array (que nadie modifica) y cada uno escribe en su propia posición de `parciales`, así que no hace falta sincronización. Lo que sí es imprescindible es `join()`: sin él, el `main` podría leer los parciales antes de que los hilos los calculen.

Las variables que usa la lambda tienen que ser `final` (o efectivamente finales); por eso se copian `i`, `desde` y `hasta` en variables nuevas en cada vuelta.

132. Productor-consumidor con un búfer acotado

Muy difícil · Hilos y Multithreading en Java · apuntesdam.com/subject/prog-servicios/topic/hilos

Solución (java)

```
import java.util.*;

class BufferAcotado {
    private final int[] datos;
    private int inicio = 0, cantidad = 0;

    BufferAcotado(int capacidad) {
        datos = new int[capacidad];
    }

    // Espera mientras el búfer está lleno
    public synchronized void poner(int v) throws InterruptedException {
        while (cantidad == datos.length) wait(); // lleno: a esperar
        datos[(inicio + cantidad) % datos.length] = v; // el hueco tras el último
        cantidad++;
        notifyAll(); // puede haber alguien esperando para sacar
    }

    // Espera mientras el búfer está vacío
    public synchronized int sacar() throws InterruptedException {
        while (cantidad == 0) wait(); // vacío: a esperar
        int v = datos[inicio];
        inicio = (inicio + 1) % datos.length;
        cantidad--;
        notifyAll(); // puede haber alguien esperando para poner
        return v;
    }
}

public class Main {
    static final int FIN = -1; // «píldora venenosa»: avisa al consumidor de que no hay más

    public static void main(String[] args) throws InterruptedException {
        Scanner sc = new Scanner(System.in);
        int capacidad = sc.nextInt(), productores = sc.nextInt(), porProductor = sc.nextInt();
        BufferAcotado buffer = new BufferAcotado(capacidad);
        int[] consumidosDe = new int[productores];
        long[] suma = new long[1];

        List<Thread> hilos = new ArrayList<>();
        for (int p = 0; p < productores; p++) {
            final int id = p;
            hilos.add(new Thread(() -> {
                try {
                    // Cada valor codifica su productor: id * 1000 + número
                    for (int k = 1; k <= porProductor; k++) buffer.poner(id * 1000 + k);
                } catch (InterruptedException e) { Thread.currentThread().interrupt(); }
            }));
        }
        Thread consumidor = new Thread(() -> {
            try {
                int v;
                while ((v = buffer.sacar()) != FIN) {
                    consumidosDe[v / 1000]++;
                    suma[0] += v % 1000;
                }
            } catch (InterruptedException e) { Thread.currentThread().interrupt(); }
        });
        consumidor.start();
        for (Thread h : hilos) h.start();
        for (Thread h : hilos) h.join();
        buffer.poner(FIN);
        consumidor.join();

        for (int p = 0; p < productores; p++) System.out.println("Productor " + p + ": " + consumidosDe[p] +
            " elementos consumidos");
        System.out.println("Suma de los números: " + suma[0]);
    }
}
```

```
}  
}
```

`synchronized` garantiza que solo un hilo a la vez modifica el búfer. Dentro, `wait()` suelta el cerrojo y duerme al hilo hasta que otro llame a `notifyAll()`; por eso se avisa cada vez que el estado cambia, para despertar a quien esté esperando el cambio contrario.

La condición se comprueba con `while` y no con `if`: al despertar, otro hilo puede haber llenado (o vaciado) el búfer antes, y hay que volver a mirar. Con `notifyAll` se despierta a todos, productores y consumidor, y cada uno comprueba si puede seguir.

El array circular evita mover elementos: el primero está en `inicio`, el hueco libre en `(inicio + cantidad) % capacidad`, y los dos índices dan la vuelta al llegar al final con el módulo.

133. Servidor de un protocolo de texto

Difícil · Sockets en Java · apuntesdam.com/subject/prog-servicios/topic/sockets

Solución (java)

```
import java.io.*;
import java.net.*;
import java.util.*;

public class Main {
    // Atiende a UN cliente: lee órdenes línea a línea y responde a cada una
    static void atender(Socket cliente) throws IOException {
        BufferedReader in = new BufferedReader(new InputStreamReader(cliente.getInputStream(), "UTF-8"));
        PrintWriter out = new PrintWriter(new OutputStreamWriter(cliente.getOutputStream(), "UTF-8"), true);
        int atendidas = 0;
        String linea;
        while ((linea = in.readLine()) != null) {
            String[] p = linea.trim().split(" ", 2);
            String orden = p[0].toUpperCase(), resto = p.length > 1 ? p[1] : "";
            String respuesta;
            switch (orden) {
                case "SUMA" -> {
                    try {
                        long total = 0;
                        for (String n : resto.trim().split("\\s+")) total += Long.parseLong(n);
                        respuesta = "OK " + total;
                    } catch (NumberFormatException e) {
                        respuesta = "ERROR números no válidos";
                    }
                }
                case "MAYUS" -> respuesta = "OK " + resto.toUpperCase();
                case "INVERTIR" -> respuesta = "OK " + new StringBuilder(resto).reverse();
                case "CONTAR" -> respuesta = "OK " + atendidas;
                case "ADIOS" -> respuesta = "OK hasta luego";
                default -> respuesta = "ERROR orden desconocida: " + p[0];
            }
            atendidas++;
            out.println(respuesta);                // autoflush: se envía al momento
            if (orden.equals("ADIOS")) break;      // al salir, el main cierra el socket
        }
    }

    public static void main(String[] args) throws Exception {
        try (ServerSocket servidor = new ServerSocket(0)) {           // puerto libre cualquiera
            Thread hiloServidor = new Thread(() -> {
                try (Socket c = servidor.accept()) {
                    atender(c);
                } catch (IOException e) {
                    System.out.println("Error del servidor: " + e.getMessage());
                }
            });
            hiloServidor.start();

            // Cliente: envía cada línea de la entrada y muestra la respuesta
            Scanner sc = new Scanner(System.in);
            try (Socket s = new Socket("localhost", servidor.getLocalPort());
                PrintWriter out = new PrintWriter(new OutputStreamWriter(s.getOutputStream(), "UTF-8"), true);
                BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputStream(), "UTF-8")))
            {
                while (sc.hasNextLine()) {
                    String orden = sc.nextLine();
                    out.println(orden);
                    String r;
                    try {
                        r = in.readLine();
                    } catch (IOException e) {
                        r = null;                                // el servidor cerró la conexión
                    }
                    System.out.println("> " + orden);
                    System.out.println("< " + (r == null ? "(conexión cerrada)" : r));
                    if (r == null || orden.equals("ADIOS")) break;
                }
            }
        }
    }
}
```

```
        }  
        hiloServidor.join();  
    }  
}
```

Un protocolo de texto línea a línea se implementa con un `BufferedReader` para leer órdenes y un `PrintWriter` con `autoFlush` para responder: sin el `true` del constructor, la respuesta se quedaría en el búfer y el cliente esperaría para siempre. Los dos se crean con UTF-8 para que las tildes y la ñ viajen bien.

Cada línea se parte en la orden y el resto con `split(" ", 2)`, y la orden se pasa a mayúsculas para que «suma» y «SUMA» sean lo mismo. El contador se incrementa después de responder, así CONTAR informa de las órdenes anteriores a ella. El bucle termina con ADIOS o cuando `readLine` devuelve `null` porque el cliente cerró la conexión.

134. Mensajes fiables sobre UDP

Muy difícil · Sockets en Java · apuntesdam.com/subject/prog-servicios/topic/sockets

Solución (java)

```
import java.io.*;
import java.net.*;
import java.nio.charset.StandardCharsets;
import java.util.*;

public class Main {
    static final int ESPERA_MS = 150, MAX_INTENTOS = 5;

    // ---- Servidor (ya hecho): pierde paquetes a propósito para simular una red mala
    static class Servidor extends Thread {
        final DatagramSocket ds;
        final List<String> recibidos = new ArrayList<>();
        int paquetes = 0, duplicados = 0, ultimoSeq = 0;

        Servidor() throws SocketException { ds = new DatagramSocket(0); }

        public void run() {
            byte[] buf = new byte[1024];
            try {
                while (true) {
                    DatagramPacket p = new DatagramPacket(buf, buf.length);
                    ds.receive(p);
                    paquetes++;
                    if (paquetes % 4 == 0) continue; // se pierde el paquete que llega
                    String msg = new String(p.getData(), 0, p.getLength(), StandardCharsets.UTF_8);
                    int seq = Integer.parseInt(msg.substring(0, msg.indexOf(':')));
                    String datos = msg.substring(msg.indexOf(':') + 1);
                    if (seq == ultimoSeq) duplicados++; // ya lo teníamos: era una retransmisión
                    else if (seq == ultimoSeq + 1) { ultimoSeq = seq; if (!datos.equals("FIN"))
                        recibidos.add(datos); }
                    if (paquetes % 3 == 0) continue; // se pierde el ACK de respuesta
                    byte[] ack = ("ACK " + seq).getBytes(StandardCharsets.UTF_8);
                    ds.send(new DatagramPacket(ack, ack.length, p.getSocketAddress()));
                    if (datos.equals("FIN")) return;
                }
            } catch (IOException e) { }
        }
    }

    // Envía un mensaje y espera su ACK; si no llega a tiempo, lo reenvía.
    // Devuelve el número de intentos, o -1 si se agotan.
    static int enviarFiable(DatagramSocket ds, SocketAddress destino, int seq, String texto) throws
        IOException {
        byte[] datos = (seq + ":" + texto).getBytes(StandardCharsets.UTF_8);
        byte[] buf = new byte[1024];
        String esperado = "ACK " + seq;
        for (int intento = 1; intento <= MAX_INTENTOS; intento++) {
            ds.send(new DatagramPacket(datos, datos.length, destino));
            long limite = System.currentTimeMillis() + ESPERA_MS;
            while (true) {
                long queda = limite - System.currentTimeMillis();
                if (queda <= 0) break; // se acabó el tiempo: reenviar
                ds.setSoTimeout((int) queda);
                DatagramPacket p = new DatagramPacket(buf, buf.length);
                try {
                    ds.receive(p);
                } catch (SocketTimeoutException e) {
                    break;
                }
                String r = new String(p.getData(), 0, p.getLength(), StandardCharsets.UTF_8);
                if (r.equals(esperado)) return intento;
                // un ACK de otro número de secuencia (llegó tarde): se ignora y se sigue esperando
            }
        }
        return -1;
    }
}
```



```

public static void main(String[] args) throws Exception {
    Servidor srv = new Servidor();
    srv.start();
    SocketAddress destino = new InetSocketAddress("localhost", srv.ds.getLocalPort());
    Scanner sc = new Scanner(System.in);
    List<String> mensajes = new ArrayList<>();
    while (sc.hasNextLine()) { String l = sc.nextLine().trim(); if (!l.isEmpty()) mensajes.add(l); }
    mensajes.add("FIN");
    try (DatagramSocket ds = new DatagramSocket()) {
        for (int i = 0; i < mensajes.size(); i++) {
            int intentos = enviarFiable(ds, destino, i + 1, mensajes.get(i));
            System.out.println("Mensaje " + (i + 1) + " «" + mensajes.get(i) + "»: "
                + (intentos < 0 ? "sin respuesta" : intentos + (intentos == 1 ? " intento" : " intentos")));
        }
    }
    srv.join(2000);
    srv.ds.close();
    System.out.println("El servidor recibió en orden: " + String.join(" | ", srv.recibidos));
    System.out.println("Retransmisiones duplicadas descartadas: " + srv.duplicados);
}
}

```

Parada y espera: se envía el mensaje y se espera su confirmación antes de mandar el siguiente. `setSoTimeout` hace que `receive` deje de esperar al cabo de un tiempo lanzando `SocketTimeoutException`; ese es el momento de reenviar el mismo mensaje, con el mismo número de secuencia.

Si llega un ACK que no es el esperado, no se puede volver a esperar ESPERA_MS completos: el tiempo de espera se mide desde el envío. Por eso se calcula un instante límite y, en cada vuelta, se espera solo lo que queda hasta él.

El número de secuencia es lo que hace que el protocolo funcione cuando se pierde un ACK: el cliente reenvía un mensaje que ya había llegado, y el servidor lo reconoce como duplicado y no lo guarda dos veces.

135. Una API REST con HttpServer

Difícil · Desarrollo de Servicios en Red · apuntesdam.com/subject/prog-servicios/topic/servicios-red-java

Solución (java)

```
import com.sun.net.httpserver.*;
import java.io.*;
import java.net.*;
import java.net.http.*;
import java.nio.charset.StandardCharsets;
import java.util.*;
import java.util.concurrent.*;
import java.util.concurrent.atomic.AtomicInteger;

public class Main {
    static final Map<Integer, String> tareas = new ConcurrentSkipListMap<>(); // id → texto, ordenadas por
    id
    static final AtomicInteger siguienteId = new AtomicInteger(1);

    static void responder(HttpExchange ex, int estado, String cuerpo) throws IOException {
        byte[] b = cuerpo.getBytes(StandardCharsets.UTF_8);
        ex.getResponseHeaders().set("Content-Type", "application/json; charset=utf-8");
        ex.sendResponseHeaders(estado, b.length == 0 ? -1 : b.length);
        if (b.length > 0) ex.getResponseBody().write(b);
        ex.close();
    }

    static String json(int id, String texto) {
        return "{\"id\":\"" + id + "\",\"texto\":\"" + texto.replace("\\", "\\\\")}.replace(\"\\\"\", \"\\\\\\\") +
        \"\"}\"";
    }

    static String error(String mensaje) {
        return "{\"error\":\"" + mensaje + "\"}\"";
    }

    // Atiende /tareas y /tareas/{id}
    static void manejar(HttpExchange ex) throws IOException {
        String metodo = ex.getRequestMethod();
        String[] partes = ex.getRequestURI().getPath().split("/"); // "", "tareas" y quizá el id

        if (partes.length == 2) { // la colección: /tareas
            switch (metodo) {
                case "GET" -> {
                    List<String> lista = new ArrayList<>();
                    tareas.forEach((id, texto) -> lista.add(json(id, texto)));
                    responder(ex, 200, "[" + String.join(",", lista) + "]");
                }
                case "POST" -> {
                    String texto = new String(ex.getRequestBody().readAllBytes(),
                        StandardCharsets.UTF_8).trim();
                    if (texto.isEmpty()) {
                        responder(ex, 400, error("texto vacío"));
                        return;
                    }
                    int id = siguienteId.getAndIncrement();
                    tareas.put(id, texto);
                    ex.getResponseHeaders().set("Location", "/tareas/" + id);
                    responder(ex, 201, json(id, texto));
                }
                default -> responder(ex, 405, "");
            }
        }
        return;
    }

    int id; // un elemento: /tareas/{id}
    try {
        id = Integer.parseInt(partes[2]);
    } catch (NumberFormatException e) {
        responder(ex, 400, error("id no válido"));
        return;
    }
    if (!metodo.equals("GET") && !metodo.equals("DELETE")) {
```

```

        responder(ex, 405, "");
        return;
    }
    String texto = tareas.get(id);
    if (texto == null) {
        responder(ex, 404, error("no existe la tarea " + id));
    } else if (metodo.equals("GET")) {
        responder(ex, 200, json(id, texto));
    } else {
        tareas.remove(id);
        responder(ex, 204, "");
    }
}

public static void main(String[] args) throws Exception {
    HttpServer srv = HttpServer.create(new InetSocketAddress("localhost", 0), 0);
    srv.createContext("/tareas", Main::manejar);
    srv.start();
    String base = "http://localhost:" + srv.getAddress().getPort();

    HttpClient cliente = HttpClient.newHttpClient();
    Scanner sc = new Scanner(System.in);
    while (sc.hasNextLine()) {
        String[] t = sc.nextLine().trim().split(" ", 3);          // MÉTODO ruta [cuerpo]
        HttpRequest.BodyPublisher cuerpo = t.length > 2
            ? HttpRequest.BodyPublishers.ofString(t[2]) : HttpRequest.BodyPublishers.noBody();
        HttpRequest req = HttpRequest.newBuilder(URI.create(base + t[1])).method(t[0], cuerpo).build();
        HttpResponse<String> r = cliente.send(req, HttpResponse.BodyHandlers.ofString());
        System.out.println(t[0] + " " + t[1] + " → " + r.statusCode() + (r.body().isEmpty() ? "" : " " +
            r.body()));
    }
    srv.stop(0);
}
}

```

Un mismo contexto atiende la colección (`/tareas`) y cada elemento (`/tareas/{id}`); partiendo la ruta por «/» se distinguen por el número de partes. Dentro de cada caso, el método HTTP decide la operación, que es la idea central de REST: la URL nombra el recurso y el método dice qué hacer con él.

Cada error tiene su código: 400 si la petición está mal (texto vacío o id que no es un número), 404 si el recurso no existe y 405 si el método no tiene sentido para esa URL. Crear devuelve 201 con la cabecera Location que dice dónde está el nuevo recurso, y borrar devuelve 204, que significa «hecho, sin contenido».

El mapa es un `ConcurrentSkipListMap` porque `HttpServer` puede atender peticiones en varios hilos, y además mantiene las tareas ordenadas por id; el contador es un `AtomicInteger` por la misma razón.

136. Disyuntor (circuit breaker) para un servicio inestable

Muy difícil · Desarrollo de Servicios en Red · apuntesdam.com/subject/prog-servicios/topic/servicios-red-java

Solución (java)

```
import java.util.*;

class CircuitBreaker {
    enum Estado { CERRADO, ABIERTO, SEMIABIERTO }

    private final int umbralFallos;        // fallos seguidos para abrir
    private final int espera;              // instantes que permanece abierto
    private Estado estado = Estado.CERRADO;
    private int fallosSeguidos = 0;
    private int abiertoDesde = 0;

    CircuitBreaker(int umbralFallos, int espera) {
        this.umbralFallos = umbralFallos;
        this.espera = espera;
    }

    Estado estado() { return estado; }

    // ¿Se puede llamar al servicio en el instante t? (puede pasar de ABIERTO a SEMIABIERTO)
    boolean permitir(int t) {
        if (estado != Estado.ABIERTO) return true;
        if (t - abiertoDesde >= espera) {    // ya ha esperado bastante: se deja pasar una llamada de prueba
            estado = Estado.SEMIABIERTO;
            return true;
        }
        return false;
    }

    // Se informa del resultado de una llamada que se permitió
    void registrar(boolean exito, int t) {
        if (estado == Estado.SEMIABIERTO) {
            if (exito) {
                estado = Estado.CERRADO;
                fallosSeguidos = 0;
            } else {
                abrir(t);                    // un solo fallo basta para volver a abrir
            }
            return;
        }
        if (exito) fallosSeguidos = 0;
        else if (++fallosSeguidos >= umbralFallos) abrir(t);
    }

    private void abrir(int t) {
        estado = Estado.ABIERTO;
        abiertoDesde = t;
        fallosSeguidos = 0;
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        CircuitBreaker cb = new CircuitBreaker(sc.nextInt(), sc.nextInt());
        String resultados = sc.next();        // una letra por instante: E = el servicio responde, F = falla
        int llamadas = 0, rechazadas = 0;
        for (int t = 0; t < resultados.length(); t++) {
            String linea;
            if (cb.permidir(t)) {
                llamadas++;
                boolean ok = resultados.charAt(t) == 'E';
                cb.registrar(ok, t);
                linea = (ok ? "llamada OK " : "llamada FALLA");
            } else {
                rechazadas++;
                linea = "rechazada ";
            }
        }
    }
}
```

```

    }
    System.out.println("t=" + t + " " + linea + " → " + cb.estado());
}
System.out.println("Llamadas al servicio: " + llamadas + ", rechazadas sin llamar: " + rechazadas);
}
}

```

Es una máquina de tres estados. **permitir** solo tiene trabajo en ABIERTO: rechaza las llamadas hasta que pasa el tiempo de espera y entonces deja pasar una de prueba cambiando a SEMIABIERTO. **registrar** decide qué hacer con el resultado según el estado en que se hizo la llamada.

En CERRADO se cuentan los fallos seguidos (un éxito pone el contador a 0) y se abre al llegar al umbral. En SEMIABIERTO no hay segunda oportunidad: un éxito cierra el circuito y un fallo lo abre de nuevo. El método privado **abrir** reúne lo que hay que hacer en los dos sitios donde se abre.

137. Guardar contraseñas con PBKDF2

Difícil · Programación Segura en Java · apuntesdam.com/subject/prog-servicios/topic/programacion-segura

Solución (java)

```
import java.security.*;
import java.security.spec.*;
import java.util.*;
import javax.crypto.*;
import javax.crypto.spec.*;

public class Main {
    static final int ITERACIONES = 120_000;

    // Deriva 32 bytes de la contraseña con PBKDF2-HMAC-SHA256
    static byte[] derivar(String clave, byte[] sal, int iteraciones) throws GeneralSecurityException {
        PBEKeySpec spec = new PBEKeySpec(clave.toCharArray(), sal, iteraciones, 256);
        return SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256").generateSecret(spec).getEncoded();
    }

    // Devuelve "pbkdf2$iteraciones$salBase64$hashBase64" con una sal aleatoria de 16 bytes
    static String guardar(String clave) throws GeneralSecurityException {
        byte[] sal = new byte[16];
        new SecureRandom().nextBytes(sal);
        byte[] hash = derivar(clave, sal, ITERACIONES);
        Base64.Encoder b64 = Base64.getEncoder();
        return "pbkdf2$" + ITERACIONES + "$" + b64.encodeToString(sal) + "$" + b64.encodeToString(hash);
    }

    // Comprueba una contraseña contra lo guardado, en tiempo constante
    static boolean verificar(String clave, String guardado) throws GeneralSecurityException {
        String[] p = guardado.split("\\$");
        if (p.length != 4 || !p[0].equals("pbkdf2")) return false;
        try {
            int iteraciones = Integer.parseInt(p[1]);
            byte[] sal = Base64.getDecoder().decode(p[2]);
            byte[] esperado = Base64.getDecoder().decode(p[3]);
            byte[] calculado = derivar(clave, sal, iteraciones); // misma sal y mismas iteraciones
            return MessageDigest.isEqual(esperado, calculado);
        } catch (IllegalArgumentException e) { // número o Base64 mal formados
            return false;
        }
    }

    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] t = sc.nextLine().split(" ", 3);
            switch (t[0]) {
                case "VECTOR" -> { // VECTOR salHex clave: resultado reproducible con 1000
                    iteraciones
                    byte[] sal = HexFormat.of().parseHex(t[1]);
                    System.out.println("Hash: " + HexFormat.of().formatHex(derivar(t[2], sal, 1000)));
                }
                case "PROBAR" -> { // PROBAR claveGuardada claveIntroducida
                    String g1 = guardar(t[1]), g2 = guardar(t[1]);
                    String[] p = g1.split("\\$");
                    System.out.println("Formato: " + p.length + " partes, algoritmo " + p[0] + ", " +
                        (p.length > 1 ? p[1] : "?") + " iteraciones, sal de "
                        + Base64.getDecoder().decode(p.length > 2 ? p[2] : "").length + " bytes, hash de "
                        + Base64.getDecoder().decode(p.length > 3 ? p[3] : "").length + " bytes");
                    System.out.println("Dos guardados de la misma clave son distintos: " + !g1.equals(g2));
                    System.out.println("La clave guardada no aparece en el texto: " + !g1.contains(t[1]));
                    System.out.println("Verificar «" + t[2] + "»: " + verificar(t[2], g1));
                }
            }
        }
    }
}
```

La sal aleatoria hace que dos usuarios con la misma contraseña tengan hashes distintos y que las tablas precalculadas no sirvan; por eso se genera con `SecureRandom` (un generador criptográfico, no `Random`) y se guarda junto al hash, porque hará falta para verificar. Las muchas iteraciones hacen que cada intento cueste tiempo, lo que frena los ataques de fuerza bruta.

Verificar no es «descifrar»: se toma la contraseña introducida, se deriva otra vez con la misma sal y las mismas iteraciones que se guardaron, y se comparan los dos hashes. Guardar las iteraciones en el texto permite aumentarlas en el futuro sin invalidar las contraseñas antiguas.

`MessageDigest.isEqual` compara en tiempo constante: tarda lo mismo aunque los primeros bytes no coincidan, así un atacante no puede ir adivinando el hash byte a byte midiendo cuánto tarda la respuesta.

138. Tokens firmados con HMAC (como un JWT)

Muy difícil · Programación Segura en Java · apuntesdam.com/subject/prog-servicios/topic/programacion-segura

Solución (java)

```
import java.nio.charset.StandardCharsets;
import java.security.*;
import java.util.*;
import javax.crypto.*;
import javax.crypto.spec.*;

public class Main {
    static final Base64.Encoder B64 = Base64.getUrlEncoder().withoutPadding();
    static final Base64.Decoder D64 = Base64.getUrlDecoder();

    static byte[] hmac(byte[] clave, String datos) throws GeneralSecurityException {
        Mac mac = Mac.getInstance("HmacSHA256");
        mac.init(new SecretKeySpec(clave, "HmacSHA256"));
        return mac.doFinal(datos.getBytes(StandardCharsets.UTF_8));
    }

    static String b64(String texto) {
        return B64.encodeToString(texto.getBytes(StandardCharsets.UTF_8));
    }

    // Token: base64url(cabecera) + "." + base64url(datos) + "." + base64url(HMAC de las dos primeras partes)
    // La cabecera es {"alg":"HS256","typ":"JWT"} y los datos {"sub":"usuario","rol":"rol","exp":instante}
    static String crear(byte[] clave, String usuario, String rol, long exp) throws GeneralSecurityException {
        {
            String cabecera = b64("{\"alg\":\"HS256\",\"typ\":\"JWT\"}");
            String datos = b64("{\"sub\":\"" + usuario + "\",\"rol\":\"" + rol + "\",\"exp\":\"" + exp + "\"}");
            String firma = B64.encodeToString(hmac(clave, cabecera + "." + datos));
            return cabecera + "." + datos + "." + firma;
        }
    }

    // Devuelve "VÁLIDO usuario rol", o "INVÁLIDO: motivo" (formato, firma o caducado)
    static String verificar(byte[] clave, String token, long ahora) throws GeneralSecurityException {
        String[] p = token.split("\\.");
        if (p.length != 3) return "INVÁLIDO: formato";
        byte[] firma;
        String datos;
        try {
            D64.decode(p[0]);
            datos = new String(D64.decode(p[1]), StandardCharsets.UTF_8);
            firma = D64.decode(p[2]);
        } catch (IllegalArgumentException e) {
            return "INVÁLIDO: formato";
        }
        // Primero la firma: hasta que no se comprueba, nada de lo que dicen los datos es fiable
        if (!MessageDigest.isEqual(firma, hmac(clave, p[0] + "." + p[1]))) return "INVÁLIDO: firma";
        if (ahora >= Long.parseLong(campo(datos, "exp"))) return "INVÁLIDO: caducado";
        return "VÁLIDO " + campo(datos, "sub") + " " + campo(datos, "rol");
    }

    // Extrae un campo de un JSON plano y sencillo como el de los datos
    static String campo(String json, String nombre) {
        int i = json.indexOf("\"" + nombre + "\":");
        if (i < 0) return null;
        int ini = i + nombre.length() + 3;
        if (json.charAt(ini) == '"') return json.substring(ini + 1, json.indexOf('"', ini + 1));
        int fin = ini;
        while (fin < json.length() && (Character.isDigit(json.charAt(fin)) || json.charAt(fin) == '-'))
            fin++;
        return json.substring(ini, fin);
    }

    public static void main(String[] args) throws Exception {
        byte[] clave = "clave-secreta-del-servidor-de-32b".getBytes(StandardCharsets.UTF_8);
        byte[] otraClave = "otra-clave-que-no-es-la-buena-32".getBytes(StandardCharsets.UTF_8);
        Scanner sc = new Scanner(System.in);
        String usuario = sc.next(), rol = sc.next();
        long exp = sc.nextLong();
    }
}
```



```

String token = crear(clave, usuario, rol, exp);
System.out.println("Token: " + token);
while (sc.hasNext()) {
    String prueba = sc.next();
    long ahora = sc.nextLong();
    String t = switch (prueba) {
        case "normal" -> token;
        case "otra-clave" -> crear(otraClave, usuario, rol, exp);
        case "cambiar-rol" -> {
            // el atacante edita los datos y deja la firma
            original
            String[] p = token.split("\\.");
            String datos = new String(D64.decode(p.length > 1 ? p[1] : ""),
                StandardCharsets.UTF_8).replace(rol, "admin");
            yield p[0] + "." + B64.encodeToString(datos.getBytes(StandardCharsets.UTF_8)) + "." +
                (p.length > 2 ? p[2] : "");
        }
        default -> "esto.no.es-un-token.valido";
    };
    System.out.println(prueba + " en t=" + ahora + ": " + verificar(clave, t, ahora));
}
}
}

```

El token no está cifrado: cualquiera puede decodificar las dos primeras partes y leer los datos. Lo que lo protege es la firma, un HMAC calculado con una clave que solo conoce el servidor. Si alguien cambia un solo carácter de los datos (por ejemplo, el rol), la firma ya no coincide, y no puede calcular la buena sin la clave.

Por eso el orden de las comprobaciones importa: primero el formato, después la firma y solo entonces se confía en los datos para mirar la caducidad. La firma se compara con `MessageDigest.isEqual`, en tiempo constante, por la misma razón que las contraseñas.

Base64URL sin relleno cambia `+` y `/` por `-` y `_` y quita los `=` finales, para que el token pueda viajar en una URL o en una cabecera sin escapar nada.

139. Biblioteca: préstamos, devoluciones y multas

Difícil · 90 minutos · apuntesdam.com/ejercicios/java/biblioteca-prestamos-y-multas

Solución (java)

```
import java.util.*;

public class Main {
    static class Libro {
        final String isbn;
        final String titulo;
        int disponibles;

        Libro(String isbn, String titulo, int ejemplares) {
            this.isbn = isbn;
            this.titulo = titulo;
            this.disponibles = ejemplares;
        }
    }

    static class Prestamo {
        final Libro libro;
        final int vence;

        Prestamo(Libro libro, int vence) {
            this.libro = libro;
            this.vence = vence;
        }
    }

    static class Socio {
        final int numero;
        final String nombre;
        final List<Prestamo> prestamos = new ArrayList<>();
        int bloqueadoHasta = 0;
        double multas = 0;

        Socio(int numero, String nombre) {
            this.numero = numero;
            this.nombre = nombre;
        }
    }

    static final int DIAS_PRESTAMO = 14;
    static final int MAX_PRESTAMOS = 3;
    static final double MULTA_DIA = 0.50;
    static final Locale ES = Locale.forLanguageTag("es-ES");

    static final Map<String, Libro> libros = new HashMap<>();
    // TreeMap: los socios quedan ordenados por número, como pide el informe
    static final Map<Integer, Socio> socios = new TreeMap<>();

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String linea = sc.nextLine().trim();
            if (!linea.isEmpty()) procesar(linea);
        }
    }

    static void procesar(String linea) {
        String[] p = linea.split(" ");
        switch (p[0]) {
            case "LIBRO" -> altaLibro(p[1], Integer.parseInt(p[2]), resto(p, 3));
            case "SOCIO" -> altaSocio(Integer.parseInt(p[1]), resto(p, 2));
            case "PRESTAR" -> prestar(Integer.parseInt(p[1]), Integer.parseInt(p[2]), p[3]);
            case "DEVOLVER" -> devolver(Integer.parseInt(p[1]), Integer.parseInt(p[2]), p[3]);
            case "INFORME" -> informe();
            default -> System.out.println("Error: orden desconocida " + p[0]);
        }
    }

    /** Las palabras de la línea desde la posición indicada (títulos y nombres pueden tener espacios). */
}
```

```

static String resto(String[] p, int desde) {
    return String.join(" ", Arrays.copyOfRange(p, desde, p.length));
}

static void altaLibro(String isbn, int ejemplares, String titulo) {
    if (libros.containsKey(isbn)) {
        System.out.println("Error: ya existe un libro con el ISBN " + isbn);
        return;
    }
    libros.put(isbn, new Libro(isbn, titulo, ejemplares));
    System.out.println("Libro añadido: " + titulo + " (" + ejemplares + (ejemplares == 1 ? " ejemplar)"
        : " ejemplares)");
}

static void altaSocio(int numero, String nombre) {
    if (socios.containsKey(numero)) {
        System.out.println("Error: ya existe un socio con el número " + numero);
        return;
    }
    socios.put(numero, new Socio(numero, nombre));
    System.out.println("Socio añadido: " + nombre + " (" + numero + ")");
}

static void prestar(int dia, int numero, String isbn) {
    Socio s = socios.get(numero);
    Libro l = libros.get(isbn);
    if (s == null) {
        System.out.println("Error: no existe el socio " + numero);
    } else if (l == null) {
        System.out.println("Error: no existe el libro " + isbn);
    } else if (dia < s.bloqueadoHasta) {
        System.out.println("Error: " + s.nombre + " no puede llevarse libros hasta el día " +
            s.bloqueadoHasta);
    } else if (s.prestamos.size() == MAX_PRESTAMOS) {
        System.out.println("Error: " + s.nombre + " ya tiene " + MAX_PRESTAMOS + " libros prestados");
    } else if (s.prestamos.stream().anyMatch(pr -> pr.libro == l)) {
        System.out.println("Error: " + s.nombre + " ya tiene «" + l.titulo + "»");
    } else if (l.disponibles == 0) {
        System.out.println("Error: no quedan ejemplares de «" + l.titulo + "»");
    } else {
        l.disponibles--;
        s.prestamos.add(new Prestamo(l, dia + DIAS_PRESTAMO));
        System.out.println("Préstamo: " + s.nombre + " se lleva «" + l.titulo + "» hasta el día " + (dia
            + DIAS_PRESTAMO));
    }
}

static void devolver(int dia, int numero, String isbn) {
    Socio s = socios.get(numero);
    Prestamo pr = null;
    if (s != null) {
        for (Prestamo x : s.prestamos) {
            if (x.libro.isbn.equals(isbn)) pr = x;
        }
    }
    if (pr == null) {
        System.out.println("Error: el socio " + numero + " no tiene prestado el libro " + isbn);
        return;
    }
    s.prestamos.remove(pr);
    pr.libro.disponibles++;
    int retraso = dia - pr.vence;
    if (retraso <= 0) {
        System.out.println("Devolución: «" + pr.libro.titulo + "» a tiempo");
        return;
    }
    double multa = retraso * MULTA_DIA;
    s.multas += multa;
    // Un nuevo retraso puede alargar el bloqueo, nunca acortarlo
    s.bloqueadoHasta = Math.max(s.bloqueadoHasta, dia + retraso);
    System.out.println("Devolución: «" + pr.libro.titulo + "» con " + retraso + (retraso == 1 ? " día" :
        " días")
        + " de retraso. Multa: " + euros(multa));
}

```

```

static void informe() {
    System.out.println("== Informe ==");
    int activos = 0;
    for (Socio s : socios.values()) activos += s.prestamos.size();
    System.out.println("Préstamos activos: " + activos);
    double total = 0;
    for (Socio s : socios.values()) {
        total += s.multas;
        if (s.prestamos.isEmpty()) continue;
        List<String> titulos = new ArrayList<>();
        for (Prestamo pr : s.prestamos) titulos.add("«" + pr.libro.titulo + "»");
        System.out.println("- " + s.nombre + " (" + s.numero + "): " + String.join(", ", titulos));
    }
    System.out.println("Multas: " + euros(total));
}

static String euros(double importe) {
    return String.format(ES, "%.2f €", importe);
}
}

```

Cada tipo de dato es una clase pequeña: `Libro` guarda cuántos ejemplares quedan libres, `Prestamo` une un libro con su día de vencimiento y `Socio` tiene su lista de préstamos, su bloqueo y sus multas. Así, cada orden se reduce a buscar objetos y cambiar sus atributos.

Los libros se guardan en un `HashMap` por ISBN y los socios en un `TreeMap` por número: los dos permiten buscar por clave al instante, y el `TreeMap` además los mantiene ordenados, que es justo lo que necesita el informe.

En `prestar`, la cadena de `if ... else if` hace las comprobaciones en el orden exacto del enunciado y solo modifica datos cuando todas pasan. Es un patrón muy útil: primero validar todo, después cambiar.

En la devolución, `Math.max` evita que una multa pequeña acorte un bloqueo que ya era más largo. Los importes se escriben con `String.format` y el formato español (`es-ES`), que pone la coma decimal.

140. Clasificación de una liga de fútbol

Difícil · 75 minutos · apuntesdam.com/ejercicios/java/clasificacion-de-una-liga

Solución (java)

```
import java.util.*;

public class Main {
    static class Equipo {
        final String nombre;
        int pj, g, e, p, gf, gc;

        Equipo(String nombre) {
            this.nombre = nombre;
        }

        int dg() {
            return gf - gc;
        }

        int puntos() {
            return 3 * g + e;
        }
    }

    static final Locale ES = Locale.forLanguageTag("es-ES");
    // LinkedHashMap: busca por nombre al instante y recuerda el orden de aparición
    static final Map<String, Equipo> equipos = new LinkedHashMap<>();

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int numLinea = 0, partidos = 0, goles = 0;
        while (sc.hasNextLine()) {
            String linea = sc.nextLine();
            numLinea++;
            if (linea.isBlank()) continue;
            String[] partes = linea.split(" ", -1);
            if (!esValida(partes)) {
                System.out.println("Aviso: la línea " + numLinea + " no es un resultado válido y se ignora");
                continue;
            }
            String[] marcador = partes[1].trim().split("-");
            int golesLocal = Integer.parseInt(marcador[0]);
            int golesVisitante = Integer.parseInt(marcador[1]);
            registrar(equipo(partes[0].trim()), golesLocal, golesVisitante);
            registrar(equipo(partes[2].trim()), golesVisitante, golesLocal);
            partidos++;
            goles += golesLocal + golesVisitante;
        }
        if (partidos == 0) {
            System.out.println("No hay resultados válidos.");
            return;
        }
        List<Equipo> tabla = new ArrayList<>(equipos.values());
        tabla.sort(Comparator.comparingInt(Equipo::puntos).reversed()
            .thenComparing(Comparator.comparingInt(Equipo::dg).reversed())
            .thenComparing(Comparator.comparingInt((Equipo x) -> x.gf).reversed())
            .thenComparing((Equipo x) -> x.nombre));
        imprimirTabla(tabla);
        System.out.println();
        System.out.println(String.format(ES, "Partidos: %d · Goles: %d · Media: %.2f goles por partido",
            partidos, goles, (double) goles / partidos));
        // El primero de la tabla con menos goles en contra: así un empate lo gana el mejor clasificado
        Equipo menos = tabla.get(0);
        for (Equipo t : tabla) {
            if (t.gc < menos.gc) menos = t;
        }
        System.out.println("Equipo menos goleado: " + menos.nombre + " (" + menos.gc + (menos.gc == 1 ? " gol" : " goles") + " en contra)");
    }

    static boolean esValida(String[] partes) {
```

```

        return partes.length == 3
            && !partes[0].isBlank() && !partes[2].isBlank()
            && partes[1].trim().matches("[0-9]+-[0-9]+")
            && !partes[0].trim().equals(partes[2].trim());
    }

    static Equipo equipo(String nombre) {
        return equipos.computeIfAbsent(nombre, Equipo::new);
    }

    static void registrar(Equipo t, int aFavor, int enContra) {
        t.pj++;
        t.gf += aFavor;
        t.gc += enContra;
        if (aFavor > enContra) t.g++;
        else if (aFavor == enContra) t.e++;
        else t.p++;
    }

    static void imprimirTabla(List<Equipo> tabla) {
        System.out.println(String.format("%-4s%-20s%3s%3s%3s%3s%4s%4s%5s%5s", "Pos", "Equipo", "PJ", "G",
            "E", "P", "GF", "GC", "DG", "Pts"));
        for (int i = 0; i < tabla.size(); i++) {
            Equipo t = tabla.get(i);
            String dg = t.dg() > 0 ? "+" + t.dg() : String.valueOf(t.dg());
            System.out.println(String.format("%-4d%-20s%3d%3d%3d%3d%4d%4d%5s%5d", i + 1, t.nombre, t.pj,
                t.g, t.e, t.p, t.gf, t.gc, dg, t.puntos()));
        }
    }
}

```

La clase `Equipo` guarda solo los datos que cambian con cada partido; los puntos y la diferencia de goles se calculan con métodos, así nunca pueden quedar desincronizados.

Un `LinkedHashMap` permite buscar un equipo por su nombre al instante y recuerda el orden de aparición.

`computeIfAbsent` crea el equipo la primera vez que aparece y lo devuelve las siguientes.

La validación usa `split(";", -1)` para no perder partes vacías y una expresión regular sencilla para el marcador, así que

`Integer.parseInt` nunca recibe algo que no sea un número.

La ordenación encadena cuatro comparadores: puntos, diferencia de goles y goles a favor de mayor a menor (`reversed`), y el nombre en orden alfabético. Al recorrer la tabla ya ordenada para buscar al menos goleado, el primero que se encuentra con el mínimo es el mejor clasificado, que es lo que pide el enunciado para los empates.

141. Cajero automático con PIN, límite diario y billetes

Medio · 50 minutos · apuntesdam.com/ejercicios/java/cajero-automatiko

Solución (java)

```
import java.util.Locale;
import java.util.Scanner;

public class Main {
    static final long LIMITE_DIARIO = 60000; // 600 € en céntimos
    static final int[] BILLETES = {50, 20, 10};

    /** Céntimos → «1.234,50 €». */
    static String euros(long centimos) {
        String entero = String.format(Locale.ROOT, "%d", centimos / 100).replace(',', '.');
        return entero + "," + String.format("%02d", centimos % 100) + " €";
    }

    /** «49.95» → 4995 céntimos; -1 si no es una cantidad válida (hasta 2 decimales). */
    static long centimos(String texto) {
        if (!texto.matches("\\d{1,7}(\\.\\d{1,2})?")) return -1;
        String[] partes = texto.split("\\.");
        long resultado = Long.parseLong(partes[0]) * 100;
        if (partes.length == 2) resultado += Long.parseLong((partes[1] + "0").substring(0, 2));
        return resultado;
    }

    /** Desglose en billetes de 50, 20 y 10 con el menor número de billetes: «3x50 1x20 1x10». */
    static String desglose(long euros) {
        StringBuilder sb = new StringBuilder();
        for (int billete : BILLETES) {
            long n = euros / billete;
            euros %= billete;
            if (n > 0) sb.append(sb.length() > 0 ? " " : "").append(n).append("x").append(billete);
        }
        return sb.toString();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] inicio = sc.nextLine().trim().split("\\s+");
        long saldo = centimos(inicio[0]);
        String pin = inicio[1];

        boolean identificado = false;
        int fallos = 0, operaciones = 0;
        long retiradoHoy = 0;

        while (sc.hasNextLine()) {
            String linea = sc.nextLine().trim();
            if (linea.isEmpty()) continue;
            String[] partes = linea.split("\\s+");
            String orden = partes[0].toUpperCase();
            String argumento = partes.length > 1 ? partes[1] : "";

            // Sin el PIN correcto solo se acepta la orden PIN
            if (!identificado) {
                if (!orden.equals("PIN")) {
                    System.out.println("Introduzca el PIN");
                } else if (argumento.equals(pin)) {
                    identificado = true;
                    System.out.println("PIN correcto");
                } else if (++fallos == 3) {
                    System.out.println("PIN incorrecto. Tarjeta retenida");
                    return;
                } else {
                    int quedan = 3 - fallos;
                    System.out.println("PIN incorrecto. " + (quedan == 1 ? "Queda 1 intento" : "Quedan " +
                        quedan + " intentos"));
                }
            }
            continue;
        }
    }
}
```

```

switch (orden) {
    case "PIN" -> System.out.println("Ya ha introducido el PIN");
    case "SALDO" -> System.out.println("Saldo: " + euros(saldo));
    case "INGRESAR" -> {
        long cantidad = centimos(argumento);
        if (cantidad <= 0) {
            System.out.println("Cantidad no válida");
        } else {
            saldo += cantidad;
            operaciones++;
            System.out.println("Ingresados " + euros(cantidad) + ". Saldo: " + euros(saldo));
        }
    }
    case "RETIRAR" -> {
        long cantidad = centimos(argumento);
        if (cantidad <= 0 || cantidad % 1000 != 0) {
            System.out.println("Importe no válido: debe ser múltiplo de 10 €");
        } else if (cantidad > saldo) {
            System.out.println("Saldo insuficiente");
        } else if (retiradoHoy + cantidad > LIMITE_DIARIO) {
            System.out.println("Supera el límite diario: hoy puede retirar hasta " +
                euros(LIMITE_DIARIO - retiradoHoy));
        } else {
            saldo -= cantidad;
            retiradoHoy += cantidad;
            operaciones++;
            System.out.println("Retirados " + euros(cantidad) + ": " + desglose(cantidad / 100)
                + ". Saldo: " + euros(saldo));
        }
    }
    case "SALIR" -> {
        System.out.println("Operaciones: " + operaciones + " · Retirado hoy: " +
            euros(retiradoHoy));
        System.out.println("Retire su tarjeta");
        return;
    }
    default -> System.out.println("Operación desconocida");
}
}
// Se acabó la entrada sin SALIR: el cajero cierra la sesión igualmente
if (identificado) {
    System.out.println("Operaciones: " + operaciones + " · Retirado hoy: " + euros(retiradoHoy));
    System.out.println("Retire su tarjeta");
}
}
}

```

El programa es un bucle que lee una orden por vuelta y un pequeño estado (identificado, fallos, saldo, retirado hoy, operaciones) que las órdenes van cambiando. Tratar el PIN antes del `switch` separa las dos fases de la sesión: identificarse y operar.

El dinero va siempre en céntimos dentro de un `long`: la entrada se convierte una sola vez con `centimos` y la salida pasa siempre por `euros`. Así no hay redondeos y el formato español (punto de millar, coma decimal) está en un único sitio.

Las comprobaciones de la retirada siguen el orden del enunciado con `if / else if`: es la forma natural de expresar «el primer problema que encuentres es el que se muestra».

El desglose es un algoritmo voraz: con 50, 20 y 10 coger siempre el billete mayor da el mínimo de billetes. Con otros billetes no siempre es así (con 25, 20 y 5, para 40 € daría 25+5+5+5 en lugar de 20+20): por eso conviene saber por qué funciona y no solo que funciona.

142. Buscaminas: números, casilla más peligrosa y zonas despejadas

Difícil · 70 minutos · apuntesdam.com/ejercicios/java/buscaminas-numeros-y-zonas

Solución (java)

```
import java.util.ArrayDeque;
import java.util.Deque;
import java.util.Locale;
import java.util.Scanner;

public class Main {
    static int filas, columnas;
    static char[][] tablero;
    static int[][] cuenta;

    /** Minas en las 8 casillas vecinas de (f, c). */
    static int minasAlrededor(int f, int c) {
        int n = 0;
        for (int df = -1; df <= 1; df++) {
            for (int dc = -1; dc <= 1; dc++) {
                int ff = f + df, cc = c + dc;
                if ((df != 0 || dc != 0) && ff >= 0 && ff < filas && cc >= 0 && cc < columnas && tablero[ff][cc] == '*') n++;
            }
        }
        return n;
    }

    /** Marca como visitada toda la zona de ceros conectada (también en diagonal) con (f, c). */
    static void despejar(int f, int c, boolean[][] visto) {
        Deque<int[]> pendientes = new ArrayDeque<>();
        pendientes.push(new int[]{f, c});
        visto[f][c] = true;
        while (!pendientes.isEmpty()) {
            int[] actual = pendientes.pop();
            for (int df = -1; df <= 1; df++) {
                for (int dc = -1; dc <= 1; dc++) {
                    int ff = actual[0] + df, cc = actual[1] + dc;
                    if (ff >= 0 && ff < filas && cc >= 0 && cc < columnas && !visto[ff][cc] && cuenta[ff][cc] == 0) {
                        visto[ff][cc] = true;
                        pendientes.push(new int[]{ff, cc});
                    }
                }
            }
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String cabecera = sc.hasNextLine() ? sc.nextLine().trim() : "";
        if (!cabecera.matches("\\d{1,2} \\d{1,2}")) {
            System.out.println("Tablero no válido: línea 1");
            return;
        }
        filas = Integer.parseInt(cabecera.split(" ")[0]);
        columnas = Integer.parseInt(cabecera.split(" ")[1]);
        if (filas < 1 || filas > 30 || columnas < 1 || columnas > 30) {
            System.out.println("Tablero no válido: línea 1");
            return;
        }
        tablero = new char[filas][columnas];
        for (int f = 0; f < filas; f++) {
            if (!sc.hasNextLine()) {
                System.out.println("Tablero no válido: faltan filas");
                return;
            }
            String linea = sc.nextLine().trim();
            if (linea.length() != columnas || !linea.matches("[*.]+")) {
                System.out.println("Tablero no válido: línea " + (f + 2));
                return;
            }
            tablero[f] = linea.toCharArray();
        }
    }
}
```

```

    }

    // Números de cada casilla (-1 en las minas)
    cuenta = new int[filas][columnas];
    int minas = 0;
    System.out.println("Tablero:");
    for (int f = 0; f < filas; f++) {
        StringBuilder sb = new StringBuilder();
        for (int c = 0; c < columnas; c++) {
            if (tablero[f][c] == '*') {
                cuenta[f][c] = -1;
                minas++;
                sb.append('*');
            } else {
                cuenta[f][c] = minasAlrededor(f, c);
                sb.append(cuenta[f][c]);
            }
        }
        System.out.println(sb);
    }

    int total = filas * columnas;
    String porcentaje = String.format(Locale.ROOT, "%.1f", minas * 100.0 / total).replace('.', ',');
    System.out.println("Minas: " + minas + " de " + total + " casillas (" + porcentaje + " %)");

    // La primera casilla (por filas) con más minas alrededor
    int mejorF = -1, mejorC = -1, max = 0;
    for (int f = 0; f < filas; f++) {
        for (int c = 0; c < columnas; c++) {
            if (cuenta[f][c] > max) {
                max = cuenta[f][c];
                mejorF = f;
                mejorC = c;
            }
        }
    }
    if (max == 0) {
        System.out.println("Casilla más peligrosa: ninguna");
    } else {
        System.out.println("Casilla más peligrosa: fila " + (mejorF + 1) + ", columna " + (mejorC + 1) +
            " (" + max + (max == 1 ? " mina" : " minas") + " alrededor");
    }

    // Cada recorrido que empieza en un cero sin visitar es una zona nueva
    boolean[][] visto = new boolean[filas][columnas];
    int zonas = 0;
    for (int f = 0; f < filas; f++) {
        for (int c = 0; c < columnas; c++) {
            if (cuenta[f][c] == 0 && !visto[f][c]) {
                zonas++;
                despejar(f, c, visto);
            }
        }
    }
    System.out.println("Zonas despejadas: " + zonas);
}
}

```

La validación va primero y termina el programa al primer error, así el resto del código puede suponer que el tablero es correcto.

El cálculo de los números es el patrón clásico de los vecinos en una matriz: dos bucles de -1 a 1 y una comprobación de límites. Guardarlos en `cuenta` evita repetir el trabajo y deja preparadas las dos preguntas siguientes.

Contar zonas es un problema de componentes conexas, el mismo que el cubo de pintura de un programa de dibujo (flood fill). Cada recorrido que empieza en un cero no visitado marca su mancha entera, así que el número de recorridos es el número de zonas.

El recorrido usa una pila explícita en lugar de recursión: en un tablero grande lleno de ceros, una función recursiva podría desbordar la pila de llamadas de Java.

143. Cifrado de Vigenère con el alfabeto español

Difícil · 60 minutos · apuntesdam.com/ejercicios/java/cifrado-vigenere-con-enie

Solución (java)

```
import java.util.Scanner;

public class Main {
    static final String ALFABETO = "ABCDEFGHIJKLMNÑOPQRSTUVWXYZ";
    static final int N = ALFABETO.length(); // 27 letras

    /** La clave en mayúsculas, o null si tiene algo que no sea una letra del alfabeto. */
    static String validarClave(String texto) {
        String clave = texto.toUpperCase();
        if (clave.isEmpty()) return null;
        for (char c : clave.toCharArray()) {
            if (ALFABETO.indexOf(c) < 0) return null;
        }
        return clave;
    }

    /** Cifra (sentido = 1) o descifra (sentido = -1). Solo cambian las letras del alfabeto, que conservan su caja. */
    static String vigenere(String texto, String clave, int sentido) {
        StringBuilder sb = new StringBuilder();
        int k = 0; // posición en la clave: solo avanza con las letras
        for (char c : texto.toCharArray()) {
            int pos = ALFABETO.indexOf(Character.toUpperCase(c));
            if (pos < 0) {
                sb.append(c);
                continue;
            }
            int desplazamiento = ALFABETO.indexOf(clave.charAt(k % clave.length()));
            char nueva = ALFABETO.charAt(((pos + sentido * desplazamiento) % N + N) % N);
            sb.append(Character.isLowerCase(c) ? Character.toLowerCase(nueva) : nueva);
            k++;
        }
        return sb.toString();
    }

    /** «Letras: 23 · Más frecuentes: O 5, Ñ 3, E 2». */
    static String analizar(String texto) {
        int[] veces = new int[N];
        int letras = 0;
        for (char c : texto.toCharArray()) {
            int pos = ALFABETO.indexOf(Character.toUpperCase(c));
            if (pos >= 0) {
                veces[pos]++;
                letras++;
            }
        }
        if (letras == 0) return "Letras: 0";
        StringBuilder sb = new StringBuilder("Letras: " + letras + " · Más frecuentes: ");
        boolean[] usada = new boolean[N];
        for (int vuelta = 0; vuelta < 3; vuelta++) {
            int mejor = -1;
            for (int i = 0; i < N; i++) {
                // Con «>» gana la primera en el alfabeto si hay empate
                if (!usada[i] && veces[i] > 0 && (mejor < 0 || veces[i] > veces[mejor])) mejor = i;
            }
            if (mejor < 0) break;
            usada[mejor] = true;
            if (vuelta > 0) sb.append(", ");
            sb.append(ALFABETO.charAt(mejor)).append(' ').append(veces[mejor]);
        }
        return sb.toString();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String clave = null;
        while (sc.hasNextLine()) {
            String linea = sc.nextLine().trim();
        }
    }
}
```

```

if (linea.isEmpty()) continue;
int espacio = linea.indexOf(' ');
String orden = (espacio < 0 ? linea : linea.substring(0, espacio)).toUpperCase();
String argumento = espacio < 0 ? "" : linea.substring(espacio + 1).trim();

switch (orden) {
    case "CLAVE" -> {
        String nueva = validarClave(argumento);
        if (nueva == null) {
            System.out.println("Clave no válida: solo letras de la A a la Z y la Ñ");
        } else {
            clave = nueva;
            System.out.println("Clave guardada (" + clave.length() + (clave.length() == 1 ? "
                letra" : " letras"));
        }
    }
    case "CIFRAR", "DESCIFRAR" -> {
        if (clave == null) System.out.println("Falta la clave");
        else if (argumento.isEmpty()) System.out.println("Falta el texto");
        else if (orden.equals("CIFRAR")) System.out.println("Cifrado: " + vigenere(argumento,
            clave, 1));
        else System.out.println("Descifrado: " + vigenere(argumento, clave, -1));
    }
    case "ANALIZAR" -> System.out.println(analizar(argumento));
    default -> System.out.println("Orden desconocida: " + orden);
}
}
}
}

```

Toda la lógica del cifrado está en un único método, `vigenere`, que cifra o descifra según el signo del parámetro `sentido`. Descifrar es desplazar hacia atrás, así que no hace falta escribir dos algoritmos.

El alfabeto como cadena convierte el problema en aritmética: `indexOf` da la posición, la suma con módulo hace la vuelta de la Z a la A, y `charAt` devuelve la letra. Con la Ñ dentro, el módulo es 27 en lugar de 26; nada más cambia.

La expresión $((x \% N) + N) \% N$ es la forma segura de calcular un módulo que nunca sea negativo en Java, C o JavaScript, donde el operador `%` conserva el signo del dividendo.

El análisis de frecuencias es la herramienta con la que se rompen estos cifrados: en español la E y la A son las letras más frecuentes, y en un texto largo cifrado con una clave corta esas frecuencias se pueden recuperar letra a letra de la clave.

144. Parking con tarifas por tipo de vehículo

Difícil · 75 minutos · apuntesdam.com/ejercicios/java/parking-con-tarifas

Solución (java)

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.Scanner;

abstract class Vehiculo {
    private final String matricula;
    private final int entrada;    // minutos desde las 00:00

    Vehiculo(String matricula, int entrada) {
        this.matricula = matricula;
        this.entrada = entrada;
    }

    String getMatricula() { return matricula; }
    int getEntrada() { return entrada; }

    abstract String tipo();
    /** Céntimos por cada 15 minutos empezados. */
    abstract long precioFraccion();
    /** Lo máximo que se cobra en un día, en céntimos. */
    abstract long maximoDiario();

    /** Importe en céntimos de una estancia: 10 minutos gratis y, después, cada cuarto de hora empezado, con un máximo. */
    long importe(int minutos) {
        if (minutos <= 10) return 0;
        long fracciones = (minutos + 14) / 15;
        return Math.min(fracciones * precioFraccion(), maximoDiario());
    }
}

class Coche extends Vehiculo {
    Coche(String matricula, int entrada) { super(matricula, entrada); }
    String tipo() { return "coche"; }
    long precioFraccion() { return 60; }
    long maximoDiario() { return 1800; }
}

class Moto extends Vehiculo {
    Moto(String matricula, int entrada) { super(matricula, entrada); }
    String tipo() { return "moto"; }
    long precioFraccion() { return 30; }
    long maximoDiario() { return 900; }
}

class Furgoneta extends Vehiculo {
    Furgoneta(String matricula, int entrada) { super(matricula, entrada); }
    String tipo() { return "furgoneta"; }
    long precioFraccion() { return 90; }
    long maximoDiario() { return 2500; }
}

public class Main {
    static final List<String> TIPOS = List.of("coche", "moto", "furgoneta");
    static final Map<String, Integer> PLAZAS = Map.of("coche", 4, "moto", 2, "furgoneta", 1);

    static final Map<String, Vehiculo> dentro = new HashMap<>();
    static long recaudado = 0;
    static int salidas = 0;

    /** «08:15» → 495 minutos; -1 si no es una hora válida. */
    static int minutos(String hora) {
        if (!hora.matches("[01]\\d|2[0-3]:[0-5]\\d")) return -1;
        return Integer.parseInt(hora.substring(0, 2)) * 60 + Integer.parseInt(hora.substring(3));
    }
}
```

```

static String euros(long centimos) {
    return String.format(Locale.ROOT, "%d,%02d €", centimos / 100, centimos % 100);
}

static String duracion(int minutos) {
    int h = minutos / 60, m = minutos % 60;
    if (h == 0) return m + " min";
    return m == 0 ? h + " h" : h + " h " + m + " min";
}

static int ocupadas(String tipo) {
    int n = 0;
    for (Vehiculo v : dentro.values()) if (v.tipo().equals(tipo)) n++;
    return n;
}

static void entra(String hora, String tipo, String matricula) {
    int min = minutos(hora);
    if (min < 0) { System.out.println("Hora no válida: " + hora); return; }
    if (!TIPOS.contains(tipo)) { System.out.println("Tipo no válido: " + tipo); return; }
    // Matrículas españolas: 4 cifras y 3 consonantes (sin vocales, Ñ ni Q)
    if (!matricula.matches("\\d{4}[BCDFGHJKLMNPSTVWXYZ]{3}")) { System.out.println("Matrícula no válida: " + matricula); return; }
    if (dentro.containsKey(matricula)) { System.out.println(matricula + " ya está dentro"); return; }
    if (ocupadas(tipo) == PLAZAS.get(tipo)) { System.out.println("No quedan plazas de " + tipo); return; }

    Vehiculo v = switch (tipo) {
        case "coche" -> new Coche(matricula, min);
        case "moto" -> new Moto(matricula, min);
        default -> new Furgoneta(matricula, min);
    };
    dentro.put(matricula, v);
    System.out.println("Entra " + matricula + " (" + tipo + ") a las " + hora);
}

static void sale(String hora, String matricula) {
    int min = minutos(hora);
    if (min < 0) { System.out.println("Hora no válida: " + hora); return; }
    Vehiculo v = dentro.get(matricula);
    if (v == null) { System.out.println(matricula + " no está en el parking"); return; }
    if (min < v.getEntrada()) { System.out.println("La hora de salida es anterior a la entrada"); return; }

    int estancia = min - v.getEntrada();
    long importe = v.importe(estancia); // cada tipo de vehículo aplica su tarifa
    dentro.remove(matricula);
    recaudado += importe;
    salidas++;
    System.out.println("Sale " + matricula + " (" + v.tipo() + ") tras " + duracion(estancia) + ": " + euros(importe));
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    while (sc.hasNextLine()) {
        String linea = sc.nextLine().trim();
        if (linea.isEmpty()) continue;
        String[] p = linea.split("\\s+");
        switch (p[0].toUpperCase()) {
            case "ENTRA" -> {
                if (p.length != 4) System.out.println("Orden incompleta");
                else entra(p[1], p[2].toLowerCase(), p[3].toUpperCase());
            }
            case "SALE" -> {
                if (p.length != 3) System.out.println("Orden incompleta");
                else sale(p[1], p[2].toUpperCase());
            }
            case "OCUPACION" -> {
                List<String> partes = new ArrayList<>();
                for (String tipo : TIPOS) partes.add(tipo + " " + ocupadas(tipo) + "/" + PLAZAS.get(tipo));
                System.out.println("Ocupación: " + String.join(" · ", partes));
            }
        }
    }
}

```

```

        }
        default -> System.out.println("Orden desconocida");
    }
}

System.out.println("Recaudación: " + euros(recaudado) + " (" + salidas + (salidas == 1 ? " salida"
: " salidas"));
List<String> quedan = new ArrayList<>(dentro.keySet());
quedan.sort(null);
System.out.println("Siguen dentro: " + (quedan.isEmpty() ? "ninguno" : String.join(", ", quedan)));
}
}

```

La clase abstracta recoge la regla de cobro, que es la misma para todos, y deja abiertos solo los dos datos que cambian. Cada subclase es pequeña y declarativa: añadir un tipo nuevo (por ejemplo, autocaravana) sería escribir una clase de cinco líneas y añadirla al `switch` de creación.

El único lugar donde aparece el tipo como texto es al crear el objeto. A partir de ahí, el programa trabaja con `Vehiculo` y el polimorfismo elige el precio correcto: es la idea central de la herencia.

La fórmula $(\text{minutos} + 14) / 15$ es el redondeo hacia arriba de una división entera: 16 minutos dan 2 cuartos de hora, 15 dan 1. Combinada con `Math.min` aplica el máximo diario sin un `if` más.

Las comprobaciones se hacen en métodos con `return` temprano: cada error sale en cuanto se detecta, y el camino normal queda al final sin anidar condiciones.

145. Evaluador de expresiones con pilas y variables

Muy difícil · 100 minutos · apuntesdam.com/ejercicios/java/evaluador-de-expresiones-con-variables

Solución (java)

```
import java.util.ArrayDeque;
import java.util.ArrayList;
import java.util.Deque;
import java.util.List;
import java.util.Map;
import java.util.Scanner;
import java.util.TreeMap;

public class Main {
    /** Error de una expresión, con el mensaje que se muestra. */
    static class ErrorExpresion extends Exception {
        ErrorExpresion(String mensaje) { super(mensaje); }
    }

    // TreeMap: las variables salen siempre por orden alfabético
    static final Map<String, Long> variables = new TreeMap<>();

    static boolean esOperador(String t) {
        return t.equals("+") || t.equals("-") || t.equals("*") || t.equals("/") || t.equals("%") ||
            t.equals("~");
    }

    static int prioridad(String op) {
        return switch (op) {
            case "~" -> 3;
            case "*", "/", "%" -> 2;
            case "+", "-" -> 1;
            default -> 0;
        };
    }

    /** «x * (2 - 15)» → [x, *, (, 2, -, 15, )]. */
    static List<String> separar(String texto) throws ErrorExpresion {
        List<String> tokens = new ArrayList<>();
        int i = 0;
        while (i < texto.length()) {
            char c = texto.charAt(i);
            if (Character.isWhitespace(c)) {
                i++;
            } else if (Character.isDigit(c) || Character.isLetter(c) || c == '_' ) {
                int j = i;
                while (j < texto.length() && (Character.isLetterOrDigit(texto.charAt(j)) || texto.charAt(j) == '_')) j++;
                tokens.add(texto.substring(i, j));
                i = j;
            } else if ("+-*/%().".indexOf(c) >= 0) {
                tokens.add(String.valueOf(c));
                i++;
            } else {
                throw new ErrorExpresion("carácter no válido '" + c + "'");
            }
        }
        return tokens;
    }

    /** Algoritmo de la estación de maniobras (shunting-yard): notación infija → postfija. */
    static List<String> aPostfija(List<String> tokens) throws ErrorExpresion {
        List<String> salida = new ArrayList<>();
        Deque<String> pila = new ArrayDeque<>();
        String anterior = null;
        for (String t : tokens) {
            if (t.equals("(")) {
                pila.push(t);
            } else if (t.equals(")")) {
                while (!pila.isEmpty() && !pila.peek().equals("(")) salida.add(pila.pop());
                if (pila.isEmpty()) throw new ErrorExpresion("paréntesis sin abrir");
                pila.pop();
            } else if (esOperador(t)) {

```



```

        // Un menos al principio, tras «(» o tras otro operador cambia el signo: es el operador
        unario ~
        boolean unario = t.equals("-") && (anterior == null || anterior.equals("(") ||
            esOperador(anterior));
        String op = unario ? "~" : t;
        // Los binarios son asociativos por la izquierda (sacan los de igual prioridad); ~ por la
        derecha
        while (!pila.isEmpty() && !pila.peek().equals("(")
            && (unario ? prioridad(pila.peek()) > prioridad(op) : prioridad(pila.peek()) >=
                prioridad(op))) {
            salida.add(pila.pop());
        }
        pila.push(op);
        t = op;
    } else {
        salida.add(t);    // número o variable
    }
    anterior = t;
}
while (!pila.isEmpty()) {
    String op = pila.pop();
    if (op.equals("(")) throw new ErrorExpresion("paréntesis sin cerrar");
    salida.add(op);
}
return salida;
}

static long evaluar(List<String> postfija) throws ErrorExpresion {
    Deque<Long> pila = new ArrayDeque<>();
    try {
        for (String t : postfija) {
            if (t.equals("~")) {
                if (pila.isEmpty()) throw new ErrorExpresion("expresión incompleta");
                pila.push(Math.negateExact(pila.pop()));
            } else if (esOperador(t)) {
                if (pila.size() < 2) throw new ErrorExpresion("expresión incompleta");
                long b = pila.pop(), a = pila.pop();
                if ((t.equals("/") || t.equals("%")) && b == 0) throw new ErrorExpresion("división entre
                    cero");
                pila.push(switch (t) {
                    case "+" -> Math.addExact(a, b);
                    case "-" -> Math.subtractExact(a, b);
                    case "*" -> Math.multiplyExact(a, b);
                    case "/" -> a / b;
                    default -> a % b;
                });
            } else if (Character.isDigit(t.charAt(0))) {
                if (!t.matches("\\d+")) throw new ErrorExpresion("número no válido " + t);
                pila.push(Long.parseLong(t));
            } else {
                Long valor = variables.get(t);
                if (valor == null) throw new ErrorExpresion("variable desconocida " + t);
                pila.push(valor);
            }
        }
    } catch (NumberFormatException e) {
        throw new ErrorExpresion("número demasiado grande");
    } catch (ArithmeticException e) {
        throw new ErrorExpresion("desbordamiento");
    }
    if (pila.size() != 1) throw new ErrorExpresion("expresión incompleta");
    return pila.pop();
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    while (sc.hasNextLine()) {
        String linea = sc.nextLine().trim();
        if (linea.isEmpty()) continue;
        if (linea.equalsIgnoreCase("VARIABLES")) {
            List<String> lista = new ArrayList<>();
            variables.forEach((nombre, valor) -> lista.add(nombre + " = " + valor));
            System.out.println("Variables: " + (lista.isEmpty() ? "ninguna" : String.join(", ",
                lista)));
        }
    }
}

```

```

        continue;
    }
    // «nombre = expresión» es una asignación
    String nombre = null, expresion = linea;
    if (linea.matches("[A-Za-z_]\\w*\\s*=.*")) {
        int igual = linea.indexOf('=');
        nombre = linea.substring(0, igual).trim();
        expresion = linea.substring(igual + 1);
    }
    try {
        List<String> postfija = aPostfija(separar(expresion));
        String textoPostfija = String.join(" ", postfija);
        try {
            long valor = evaluar(postfija);
            if (nombre != null) variables.put(nombre, valor);
            System.out.println(linea + " → " + textoPostfija + " → " + (nombre != null ? nombre + " = " : "") + valor);
        } catch (ErrorExpresion e) {
            System.out.println(linea + " → " + (textoPostfija.isEmpty() ? "" : textoPostfija + " → ") + "Error: " + e.getMessage());
        }
    } catch (ErrorExpresion e) {
        System.out.println(linea + " → Error: " + e.getMessage());
    }
}
}
}

```

El programa sigue las fases de un intérprete real: análisis léxico (`separar`), análisis sintáctico (`aPostfija`) y evaluación (`evaluar`). Cada fase puede fallar con un mensaje propio, y por eso el `main` sabe si mostrar la postfija o no según en qué fase se produjo el error.

El algoritmo de la estación de maniobras usa la pila para «aparcar» operadores hasta saber si pueden salir: un operador solo sale cuando llega otro de menor o igual prioridad o un paréntesis de cierre. La diferencia entre `>=` y `>` es toda la diferencia entre asociatividad por la izquierda (`20 - 5 - 3 = 12`) y por la derecha (`--4 = 4`).

Evaluar la postfija con otra pila es lineal y no necesita prioridades: el orden ya está resuelto. Hay que desapilar el segundo operando antes que el primero, o `10 - 3` daría -7.

La excepción propia convierte todos los fallos (tokens raros, paréntesis, operandos que faltan, divisiones entre cero, desbordamientos) en un único tipo de error con mensaje. El `TreeMap` mantiene las variables ordenadas sin tener que ordenarlas al mostrarlas.

146. Conversor de CSV a JSON con comillas y tipos

Difícil · 75 minutos · apuntesdam.com/ejercicios/java/conversor-csv-a-json

Solución (java)

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Main {
    /** Campos de una línea CSV separados por comas, con comillas dobles opcionales ("" es una comilla).
     * null si hay comillas sin cerrar. */
    static List<String> campos(String linea) {
        List<String> lista = new ArrayList<>();
        StringBuilder actual = new StringBuilder();
        boolean entreComillas = false;
        for (int i = 0; i < linea.length(); i++) {
            char c = linea.charAt(i);
            if (entreComillas) {
                if (c != '"') {
                    actual.append(c);
                } else if (i + 1 < linea.length() && linea.charAt(i + 1) == '"') {
                    actual.append('');
                    i++;
                } else {
                    entreComillas = false;
                }
            } else if (c == '"') {
                entreComillas = true;
            } else if (c == ',') {
                lista.add(actual.toString().trim());
                actual.setLength(0);
            } else {
                actual.append(c);
            }
        }
        if (entreComillas) return null;
        lista.add(actual.toString().trim());
        return lista;
    }

    /** Texto JSON entre comillas, con las comillas, barras y tabuladores escapados. */
    static String cadena(String s) {
        StringBuilder sb = new StringBuilder("\"");
        for (char c : s.toCharArray()) {
            switch (c) {
                case '"' -> sb.append("\\");
                case '\\' -> sb.append("\\");
                case '\t' -> sb.append("\\t");
                default -> sb.append(c);
            }
        }
        return sb.append('"').toString();
    }

    /** El valor JSON de un campo: null, número, booleano o texto. */
    static String valor(String campo) {
        if (campo.isEmpty()) return "null";
        // Un número JSON no admite ceros a la izquierda: «08001» es un código postal, no el 8001
        if (campo.matches("-?(0|[1-9]\\d*)(\\.\\d+)?")) return campo;
        if (campo.equals("true") || campo.equals("false")) return campo;
        return cadena(campo);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<String> cabecera = null;
        List<String> objetos = new ArrayList<>();
        List<Integer> ignoradas = new ArrayList<>();
        int numero = 0;
        while (sc.hasNextLine()) {
            String linea = sc.nextLine();
            numero++;
        }
    }
}
```

```

        if (linea.isBlank()) continue;
        List<String> c = campos(linea);
        if (cabecera == null) {
            cabecera = c;
            continue;
        }
        if (c == null || c.size() != cabecera.size()) {
            ignoradas.add(numero);
            continue;
        }
        StringBuilder obj = new StringBuilder(" {\n");
        for (int i = 0; i < c.size(); i++) {
            obj.append("    ").append(cadena(cabecera.get(i))).append(": ").append(valor(c.get(i)));
            obj.append(i < c.size() - 1 ? ",\n" : "\n");
        }
        objetos.add(obj.append("  ").toString());
    }

    if (cabecera == null) {
        System.out.println("El fichero está vacío");
        return;
    }
    System.out.println(objetos.isEmpty() ? "[]" : "[\n" + String.join(",\n", objetos) + "\n]");
    String lista = ignoradas.isEmpty() ? "" : " (" + String.join(", ",
        ignoradas.stream().map(String::valueOf).toList()) + ")";
    System.out.println("Registros: " + objetos.size() + " · Líneas ignoradas: " + ignoradas.size() +
        lista);
}
}

```

El analizador de líneas es una máquina de estados con dos estados (dentro o fuera de comillas). Es la técnica general para leer formatos con delimitadores y escapes, y explica por qué en producción se usan librerías (OpenCSV, Apache Commons CSV) en lugar de `split`.

Separar el análisis (`campos`), la decisión de tipo (`valor`) y el escape (`cadena`) en métodos pequeños permite probar cada uno por separado y reutilizar el escape para las claves.

La regla del número sigue la gramática de JSON: por eso `08001` y `007` siguen siendo texto. Es un error real y frecuente al importar códigos postales, teléfonos o DNI desde hojas de cálculo, que se comen los ceros de la izquierda.

La salida se monta en memoria y se escribe al final: así el JSON es válido aunque haya líneas ignoradas por el medio, y el resumen va después sin mezclarse con él.

147. Nóminas de una plantilla con distintos contratos

Difícil · 80 minutos · apuntesdam.com/ejercicios/java/nominas-de-una-plantilla

Solución (java)

```
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;
import java.util.Scanner;

interface Retribuible {
    /** Bruto del mes, en céntimos. */
    long bruto();

    /** Retención de IRPF de la tabla simplificada de la empresa: un tipo según el tramo del bruto mensual.
     */
    default long retencion() {
        long b = bruto();
        int centesimas = b <= 100000 ? 200 : b <= 200000 ? 1200 : b <= 300000 ? 1800 : 2400;
        return Main.porcentaje(b, centesimas);
    }

    /** Cotización del trabajador a la Seguridad Social: un 6,5 % del bruto en este ejercicio. */
    default long seguridadSocial() {
        return Main.porcentaje(bruto(), 650);
    }

    default long neto() {
        return bruto() - retencion() - seguridadSocial();
    }
}

abstract class Empleado implements Retribuible {
    private final String nombre;

    Empleado(String nombre) { this.nombre = nombre; }

    String getNombre() { return nombre; }

    abstract String tipo();
}

class Asalariado extends Empleado {
    private final long anual;
    private final int pagas;

    Asalariado(String nombre, long anual, int pagas) {
        super(nombre);
        this.anual = anual;
        this.pagas = pagas;
    }

    String tipo() { return "asalariado"; }

    // Redondeo al céntimo más cercano con enteros: (2a + p) / 2p
    public long bruto() { return (anual * 2 + pagas) / (2L * pagas); }
}

class PorHoras extends Empleado {
    private final int horas;
    private final long precioHora;

    PorHoras(String nombre, int horas, long precioHora) {
        super(nombre);
        this.horas = horas;
        this.precioHora = precioHora;
    }

    String tipo() { return "por horas"; }

    public long bruto() {
        int normales = Math.min(horas, 160), extra = Math.max(0, horas - 160);
        return normales * precioHora + Main.porcentaje(extra * precioHora, 12500); // las extra, al 125 %
    }
}
```

```

    }
}

class Comercial extends Empleado {
    private final long base, ventas, comision;    // comisión en centésimas de %

    Comercial(String nombre, long base, long ventas, long comision) {
        super(nombre);
        this.base = base;
        this.ventas = ventas;
        this.comision = comision;
    }

    String tipo() { return "comercial"; }

    public long bruto() { return base + Math.min(Main.porcentaje(ventas, comision), 200000); }
}

class Becario extends Empleado {
    private final long ayuda;

    Becario(String nombre, long ayuda) {
        super(nombre);
        this.ayuda = ayuda;
    }

    String tipo() { return "becario"; }

    public long bruto() { return ayuda; }

    // La beca no lleva retención en este ejercicio: se sobrescribe el método por defecto
    @Override
    public long retencion() { return 0; }
}

public class Main {
    /** cantidad × centésimas / 10000, redondeado al céntimo (las mitades suben). */
    static long porcentaje(long cantidad, long centesimas) {
        return (cantidad * centesimas + 5000) / 10000;
    }

    /** «1234.5» → 123450; -1 si no es un número positivo con hasta 2 decimales. */
    static long centimos(String texto) {
        if (!texto.matches("\\d{1,9}(\\.\\d{1,2})?")) return -1;
        String[] p = texto.split("\\.");
        return Long.parseLong(p[0]) * 100 + (p.length == 2 ? Long.parseLong((p[1] + "0").substring(0, 2)) :
            0);
    }

    static String euros(long c) {
        return String.format(Locale.ROOT, "%d", c / 100).replace(',', '.') + "," + String.format("%02d", c
            % 100);
    }

    /** El empleado de una línea, o una excepción con el motivo por el que se ignora. */
    static Empleado leer(String linea) {
        int espacio = linea.indexOf(' ');
        if (espacio < 0) throw new IllegalArgumentException("faltan o sobran datos");
        String tipo = linea.substring(0, espacio).toUpperCase();
        String[] d = linea.substring(espacio + 1).split(";", -1);
        int esperados = switch (tipo) {
            case "ASALARIADO", "HORAS" -> 3;
            case "COMERCIAL" -> 4;
            case "BECARIO" -> 2;
            default -> throw new IllegalArgumentException("tipo desconocido " + tipo);
        };
        if (d.length != esperados || d[0].isBlank()) throw new IllegalArgumentException("faltan o sobran
            datos");
        String nombre = d[0].trim();
        long[] n = new long[d.length];
        for (int i = 1; i < d.length; i++) {
            n[i] = centimos(d[i].trim());
            if (n[i] < 0) throw new IllegalArgumentException("dato no válido");
        }
    }
}

```

```

switch (tipo) {
    case "ASALARIADO" -> {
        if (n[2] != 1200 && n[2] != 1400) throw new IllegalArgumentException("dato no válido");
        return new Asalariado(nombre, n[1], (int) (n[2] / 100));
    }
    case "HORAS" -> {
        if (n[1] % 100 != 0 || n[1] > 30000) throw new IllegalArgumentException("dato no válido");
        return new PorHoras(nombre, (int) (n[1] / 100), n[2]);
    }
    case "COMERCIAL" -> {
        if (n[3] > 2000) throw new IllegalArgumentException("dato no válido");
        return new Comercial(nombre, n[1], n[2], n[3]);
    }
    default -> {
        return new Becario(nombre, n[1]);
    }
}

}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    List<Empleado> plantilla = new ArrayList<>();
    int numero = 0;
    while (sc.hasNextLine()) {
        String linea = sc.nextLine().trim();
        numero++;
        if (linea.isEmpty()) continue;
        try {
            plantilla.add(leer(linea));
        } catch (IllegalArgumentException e) {
            System.out.println("Línea " + numero + " ignorada: " + e.getMessage());
        }
    }
    if (plantilla.isEmpty()) {
        System.out.println("No hay nóminas que calcular.");
        return;
    }

    String formato = "%-20s%-12s%11s%11s%11s%11s\n";
    System.out.printf(formato, "Empleado", "Tipo", "Bruto", "IRPF", "SS", "Neto");
    long bruto = 0, irpf = 0, ss = 0, neto = 0;
    Empleado mejor = plantilla.get(0);
    for (Empleado e : plantilla) {
        // El mismo código sirve para cualquier empleado: cada clase sabe calcular su bruto
        System.out.printf(formato, e.getNombre(), e.tipo(), euros(e.bruto()), euros(e.retencion()),
            euros(e.seguridadSocial()), euros(e.neto()));
        bruto += e.bruto();
        irpf += e.retencion();
        ss += e.seguridadSocial();
        neto += e.neto();
        if (e.neto() > mejor.neto()) mejor = e;
    }
    System.out.printf(formato, "TOTAL", "", euros(bruto), euros(irpf), euros(ss), euros(neto));
    System.out.println("Coste para la empresa: " + euros(bruto + porcentaje(bruto, 3000)) + " €");
    System.out.println("Mayor neto: " + mejor.getNombre() + " (" + euros(mejor.neto()) + " €");
}
}

```

La interfaz **Retribuible** define el contrato (todo lo que se puede pagar tiene un bruto) y sus métodos por defecto implementan las reglas comunes. Las clases concretas solo dicen cómo se calcula su bruto: el código de las deducciones existe una sola vez.

Becario muestra para qué sirve sobrescribir un método por defecto: cambia una regla para un caso concreto, y **neto()**, que no se ha tocado, la usa automáticamente porque llama a **retencion()** a través del objeto.

La clase abstracta **Empleado** aporta lo que la interfaz no puede tener: un atributo (el nombre) con su constructor. Interfaz para el comportamiento, clase abstracta para el estado común: es una combinación muy habitual en Java.

Todo el dinero va en céntimos y cada porcentaje se redondea en un solo sitio, **porcentaje**. Así la tabla siempre cuadra: el neto de cada fila es exactamente su bruto menos sus deducciones, y los totales son la suma de las filas.

148. Contador de palabras en paralelo con un pool de hilos

Difícil · 70 minutos · apuntesdam.com/ejercicios/java/contador-de-palabras-con-hilos

Solución (java)

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Scanner;
import java.util.Set;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class Main {
    static final Set<String> VACIAS = Set.of("de", "la", "que", "el", "en", "y", "a", "los", "se", "del",
        "las", "un",
        "por", "con", "no", "una", "su", "para", "es", "al", "lo", "como");

    /** Lo que hace cada hilo: contar las palabras de su bloque en un mapa propio, sin compartir nada. */
    static Map<String, Integer> contar(List<String> lineas) {
        Map<String, Integer> cuenta = new HashMap<>();
        for (String linea : lineas) {
            for (String palabra : linea.toLowerCase().split("[^a-záéíóúñ]+")) {
                if (!palabra.isEmpty()) cuenta.merge(palabra, 1, Integer::sum);
            }
        }
        return cuenta;
    }

    static int suma(Map<String, Integer> cuenta) {
        int total = 0;
        for (int n : cuenta.values()) total += n;
        return total;
    }

    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        String primera = sc.hasNextLine() ? sc.nextLine().trim() : "";
        if (!primera.matches("[1-8]")) {
            System.out.println("Número de hilos no válido: debe estar entre 1 y 8");
            return;
        }
        List<String> lineas = new ArrayList<>();
        while (sc.hasNextLine()) {
            String linea = sc.nextLine();
            if (!linea.isBlank()) lineas.add(linea);
        }
        if (lineas.isEmpty()) {
            System.out.println("No hay texto que analizar");
            return;
        }

        // Nunca más hilos que líneas; los bloques, lo más iguales posible
        int hilos = Math.min(Integer.parseInt(primera), lineas.size());
        ExecutorService pool = Executors.newFixedThreadPool(hilos);
        List<Future<Map<String, Integer>>> resultados = new ArrayList<>();
        List<String> bloques = new ArrayList<>();
        int inicio = 0;
        for (int i = 0; i < hilos; i++) {
            int tamaño = lineas.size() / hilos + (i < lineas.size() % hilos ? 1 : 0);
            List<String> bloque = lineas.subList(inicio, inicio + tamaño);
            resultados.add(pool.submit(() -> contar(bloque)));
            bloques.add(tamaño == 1 ? "[" + (inicio + 1) + "]" : "[" + (inicio + 1) + "-" + (inicio +
                tamaño) + "]");
            inicio += tamaño;
        }

        // Se unen los resultados en el hilo principal, en el orden de los bloques
        Map<String, Integer> total = new HashMap<>();
        List<String> parciales = new ArrayList<>();
        for (int i = 0; i < hilos; i++) {
```



```

        Map<String, Integer> parcial = resultados.get(i).get(); // get() espera a que termine ese hilo
        parciales.add(" Bloque " + bloques.get(i) + ": " + suma(parcial) + " palabras");
        parcial.forEach((palabra, n) -> total.merge(palabra, n, Integer::sum));
    }
    pool.shutdown();

    System.out.println("Hilos: " + hilos + " · Líneas: " + lineas.size());
    parciales.forEach(System.out::println);
    System.out.println("Palabras: " + suma(total) + " · Distintas: " + total.size());

    List<Map.Entry<String, Integer>> lista = new ArrayList<>();
    for (Map.Entry<String, Integer> e : total.entrySet()) if (!VACIAS.contains(e.getKey()))
        lista.add(e);
    lista.sort((a, b) -> !a.getValue().equals(b.getValue()) ? b.getValue() - a.getValue() :
        a.getKey().compareTo(b.getKey()));
    System.out.println("Más frecuentes:");
    for (int i = 0; i < Math.min(5, lista.size()); i++) {
        System.out.printf("%4d %s%n", lista.get(i).getValue(), lista.get(i).getKey());
    }
}
}

```

Cada hilo trabaja sobre su propio bloque y su propio mapa, y devuelve el resultado en lugar de escribirlo en una variable compartida. Sin estado compartido no hay condiciones de carrera, así que no hacen falta `synchronized`, cerrojos ni `ConcurrentHashMap`.

`Callable` es la versión de `Runnable` que devuelve un valor, y `Future.get()` es el punto de encuentro: bloquea hasta que la tarea termina. Recorrer los `Future` en orden hace que los parciales se impriman siempre igual, aunque los hilos acaben en cualquier orden.

La suma de recuentos es asociativa y conmutativa: da igual cómo se reparta el texto y en qué orden se combinen los trozos, el total es el mismo. Esa propiedad es la que permite paralelizar este problema (y la que hay que comprobar antes de paralelizar cualquier otro).

El pool limita los hilos a los pedidos (y a las líneas que hay) y los reutiliza. Crear un `Thread` a mano por cada bloque funcionaría con 8 bloques, pero no con 10.000.

149. Validador de DNI, NIE e IBAN diseñado con clases de equivalencia

Medio · 50 minutos · apuntesdam.com/ejercicios/java/validador-dni-nie-iban

Solución (java)

```
import java.util.Scanner;

public class Main {
    static final String LETRAS = "TRWAGMYFPDXBNJZSQVHLCKE";

    /** null si el DNI es correcto, o el motivo por el que no lo es. */
    static String validarDni(String dni) {
        dni = dni.toUpperCase();
        if (!dni.matches("\\d{8}[A-Z]")) return "formato incorrecto (8 cifras y una letra)";
        char esperada = LETRAS.charAt(Integer.parseInt(dni.substring(0, 8)) % 23);
        return dni.charAt(8) == esperada ? null : "la letra debería ser " + esperada;
    }

    /** El NIE se valida como un DNI cambiando X, Y y Z por 0, 1 y 2. */
    static String validarNie(String nie) {
        nie = nie.toUpperCase();
        if (!nie.matches("[XYZ]\\d{7}[A-Z]")) return "formato incorrecto (X, Y o Z, 7 cifras y una letra)";
        return validarDni("XYZ".indexOf(nie.charAt(0)) + nie.substring(1));
    }

    /** IBAN español: 24 caracteres y control módulo 97 (ISO 13616). */
    static String validarIban(String iban) {
        iban = iban.replace(" ", "").toUpperCase();
        if (!iban.startsWith("ES")) return "solo se admiten IBAN españoles";
        if (iban.length() != 24) return "un IBAN español tiene 24 caracteres";
        if (!iban.substring(2).matches("\\d{22}")) return "después de ES solo puede haber cifras";
        // Se pasan los 4 primeros caracteres al final, las letras a números (A = 10 ... Z = 35)
        // y se calcula el resto entre 97 cifra a cifra, sin construir un número enorme
        String reordenado = iban.substring(4) + iban.substring(0, 4);
        int resto = 0;
        for (char c : reordenado.toCharArray()) {
            String valor = Character.isLetter(c) ? String.valueOf(c - 'A' + 10) : String.valueOf(c);
            for (char cifra : valor.toCharArray()) resto = (resto * 10 + (cifra - '0')) % 97;
        }
        return resto == 1 ? null : "los dígitos de control no cuadran";
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int total = 0, validos = 0;
        while (sc.hasNextLine()) {
            String linea = sc.nextLine().trim();
            if (linea.isEmpty()) continue;
            int espacio = linea.indexOf(' ');
            String tipo = (espacio < 0 ? linea : linea.substring(0, espacio)).toUpperCase();
            String valor = espacio < 0 ? "" : linea.substring(espacio + 1).trim();
            String error = switch (tipo) {
                case "DNI" -> validarDni(valor);
                case "NIE" -> validarNie(valor);
                case "IBAN" -> validarIban(valor);
                default -> "tipo de documento desconocido";
            };
            total++;
            if (error == null) validos++;
            System.out.println(tipo + (valor.isEmpty() ? "" : " " + valor) + " → " + (error == null ?
                "válido" : "no válido: " + error));
        }
        System.out.println("Válidos: " + validos + " de " + total);
    }
}
```

Cada validador devuelve el motivo del error en lugar de un simple `boolean`. Eso hace el programa más útil para el usuario y las pruebas más precisas: no basta con saber que algo es inválido, hay que saber que lo es por la razón correcta.

El NIE se reduce a un DNI y se valida con el mismo método: reutilizar en lugar de copiar significa que un error corregido en el DNI queda corregido también en el NIE.

El cálculo del módulo 97 cifra a cifra se basa en que $(a \times 10 + b) \bmod 97 = ((a \bmod 97) \times 10 + b) \bmod 97$: nunca hay un número mayor que 970. Es lo que permite comprobar IBAN en cualquier lenguaje sin `BigInteger`.

Las pruebas salen de la tabla de clases de equivalencia y valores límite: un caso por cada grupo que el programa trata igual y los extremos (00000000 y 99999999). Con ocho casos bien elegidos se cubre lo que no cubrirían cien números al azar.

150. Fichero de productos con registros de tamaño fijo y acceso directo

Muy difícil · 90 minutos · apuntesdam.com/ejercicios/java/fichero-de-registros-con-acceso-directo

Solución (java)

```
import java.io.File;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.util.Locale;
import java.util.Scanner;

public class Main {
    static final int NOMBRE = 30; // caracteres del nombre
    static final int TAM = 4 + NOMBRE * 2 + 8 + 4 + 1; // código + nombre + precio + stock + borrado = 77 bytes

    static RandomAccessFile f;

    /** Acceso directo: el registro del código N empieza en el byte (N - 1) x 77. */
    static long posicion(int codigo) {
        return (long) (codigo - 1) * TAM;
    }

    /** Un registro está activo si existe, guarda ese código y no tiene la marca de borrado. */
    static boolean activo(int codigo) throws IOException {
        if (posicion(codigo) + TAM > f.length()) return false;
        f.seek(posicion(codigo));
        int guardado = f.readInt();
        f.seek(posicion(codigo) + TAM - 1);
        return guardado == codigo && !f.readBoolean();
    }

    static void escribir(int codigo, String nombre, double precio, int stock) throws IOException {
        f.seek(posicion(codigo));
        f.writeInt(codigo);
        StringBuilder sb = new StringBuilder(nombre);
        sb.setLength(NOMBRE); // recorta o rellena con '\0' hasta 30 caracteres
        f.writeChars(sb.toString());
        f.writeDouble(precio);
        f.writeInt(stock);
        f.writeBoolean(false);
    }

    /** Lee el registro del código (que debe estar activo) como texto. */
    static String leer(int codigo) throws IOException {
        f.seek(posicion(codigo) + 4);
        StringBuilder nombre = new StringBuilder();
        for (int i = 0; i < NOMBRE; i++) {
            char c = f.readChar();
            if (c != '\0') nombre.append(c);
        }
        double precio = f.readDouble();
        int stock = f.readInt();
        return codigo + " · " + nombre + " · " + euros(precio) + " · " + stock + " uds";
    }

    static String euros(double x) {
        return String.format(Locale.ROOT, "%.2f", x).replace('.', ',') + " €";
    }

    static void orden(String linea) throws IOException {
        String[] p = linea.split("\\s+", 2);
        String args = p.length > 1 ? p[1] : "";
        switch (p[0].toUpperCase()) {
            case "ALTA" -> {
                String[] c = args.split(";", -1);
                if (c.length != 4 || !c[0].matches("\\d{1,4}") || Integer.parseInt(c[0]) == 0 || c[1].isBlank() || !c[2].matches("\\d+(\\.\\d{1,2})?") || !c[3].matches("\\d+")) {
                    System.out.println("Datos no válidos");
                    return;
                }
                int codigo = Integer.parseInt(c[0]);
            }
        }
    }
```

```

        if (activo(codigo)) {
            System.out.println("Ya existe el producto " + codigo);
            return;
        }
        String nombre = c[1].trim();
        escribir(codigo, nombre, Double.parseDouble(c[2]), Integer.parseInt(c[3]));
        System.out.println("Alta del producto " + codigo + " en el byte " + posicion(codigo)
            + (nombre.length() > NOMBRE ? " (nombre recortado a " + NOMBRE + " caracteres)" :
            ""));
    }
    case "PRECIO" -> {
        String[] c = args.split("\\s+");
        if (c.length != 2 || !c[0].matches("\\d{1,4}") || !c[1].matches("\\d+(\\.\\d{1,2})?")) {
            System.out.println("Datos no válidos");
        } else if (!activo(Integer.parseInt(c[0]))) {
            System.out.println("No existe el producto " + c[0]);
        } else {
            // Solo se reescriben los 8 bytes del precio: no hace falta leer ni escribir el resto
            // del registro
            long donde = posicion(Integer.parseInt(c[0])) + 4 + NOMBRE * 2;
            f.seek(donde);
            double antes = f.readDouble();
            f.seek(donde);
            f.writeDouble(Double.parseDouble(c[1]));
            System.out.println("Precio del producto " + c[0] + ": " + euros(antes) + " → " +
                euros(Double.parseDouble(c[1])));
        }
    }
}
case "BAJA" -> {
    if (!args.matches("\\d{1,4}") || !activo(Integer.parseInt(args))) {
        System.out.println("No existe el producto " + args);
    } else {
        f.seek(posicion(Integer.parseInt(args)) + TAM - 1);
        f.writeBoolean(true); // borrado lógico: el registro sigue ahí,
        // marcado
        System.out.println("Baja lógica del producto " + args);
    }
}
case "VER" -> System.out.println(args.matches("\\d{1,4}") && activo(Integer.parseInt(args)) ?
    leer(Integer.parseInt(args)) : "No existe el producto " + args);
case "LISTAR" -> {
    int total = (int) (f.length() / TAM);
    int activos = 0;
    StringBuilder sb = new StringBuilder();
    for (int codigo = 1; codigo <= total; codigo++) {
        if (activo(codigo)) {
            activos++;
            sb.append(" ").append(leer(codigo)).append('\n');
        }
    }
    System.out.print("Productos activos: " + activos + "\n" + sb);
}
case "INFORME" -> {
    int total = (int) (f.length() / TAM);
    int activos = 0;
    double valor = 0;
    for (int codigo = 1; codigo <= total; codigo++) {
        if (!activo(codigo)) continue;
        activos++;
        f.seek(posicion(codigo) + 4 + NOMBRE * 2);
        double precio = f.readDouble();
        valor += precio * f.readInt();
    }
    System.out.println("Fichero: " + f.length() + " bytes, " + total + " huecos de registro, " +
        activos
        + " activos, " + (total - activos) + " libres o borrados · valor del stock " +
        euros(valor));
}
default -> System.out.println("Orden desconocida");
}
}

public static void main(String[] args) throws IOException {
    File fichero = File.createTempFile("productos", ".dat");

```

```

        fichero.deleteOnExit();
    try (RandomAccessFile raf = new RandomAccessFile(fichero, "rw")) {
        f = raf;
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String linea = sc.nextLine().trim();
            if (!linea.isEmpty()) orden(linea);
        }
    }
}

```

El tamaño fijo es lo que hace posible el acceso directo: con una multiplicación se sabe dónde está cualquier registro, sin leer los anteriores. Es el mismo principio que usan los gestores de bases de datos al organizar los datos en páginas de tamaño fijo.

`RandomAccessFile` tiene un puntero que avanza con cada lectura o escritura; `seek` lo coloca donde se quiera. Por eso cambiar el precio son dos `seek`: uno para leer el valor anterior y otro para volver a la misma posición antes de escribir.

El borrado lógico cambia un solo byte en lugar de reorganizar el fichero. El precio a pagar es que el fichero no encoge: con el tiempo se llena de huecos, y por eso los sistemas reales tienen operaciones de compactación o reutilizan los huecos (aquí, un `alta` con el mismo código).

Los huecos que se crean al escribir más allá del final contienen ceros. Guardar el propio código dentro del registro permite distinguir un registro real de un hueco sin una tabla aparte.

151. Servidor de órdenes con sockets e hilos

Muy difícil · 100 minutos · apuntesdam.com/ejercicios/java/servidor-de-ordenes-con-sockets

Solución (java)

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.charset.StandardCharsets;
import java.util.Map;
import java.util.Scanner;
import java.util.concurrent.ConcurrentHashMap;

public class Main {
    // Compartido por todas las conexiones: cada cliente se atiende en su propio hilo
    static final Map<String, String> almacen = new ConcurrentHashMap<>();

    /** La respuesta del protocolo a una línea. */
    static String responder(String linea) {
        String[] p = linea.trim().split("\\s+", 2);
        String orden = p[0].toUpperCase();
        String resto = p.length > 1 ? p[1] : "";
        switch (orden) {
            case "" -> { return "ERR línea vacía"; }
            case "SUMA" -> {
                String[] n = resto.split("\\s+");
                if (n.length != 2 || !n[0].matches("-?\\d{1,9}") || !n[1].matches("-?\\d{1,9}")) return "ERR números no válidos";
                return "OK " + (Long.parseLong(n[0]) + Long.parseLong(n[1]));
            }
            case "MAYUS" -> { return resto.isEmpty() ? "ERR falta el texto" : "OK " + resto.toUpperCase(); }
            case "INVIERTE" -> { return resto.isEmpty() ? "ERR falta el texto" : "OK " + new
                StringBuilder(resto).reverse(); }
            case "GUARDA" -> {
                String[] kv = resto.split("\\s+", 2);
                if (kv.length != 2) return "ERR uso: GUARDA clave valor";
                String anterior = almacen.put(kv[0], kv[1]);
                return anterior == null ? "OK guardado" : "OK sustituido (antes: " + anterior + ")";
            }
            case "LEE" -> {
                String valor = almacen.get(resto);
                return valor == null ? "ERR no existe " + resto : "OK " + valor;
            }
            case "CUENTA" -> { return "OK " + almacen.size(); }
            case "ADIOS" -> { return "OK adiós"; }
            default -> { return "ERR orden desconocida"; }
        }
    }

    /** Atiende a un cliente: una respuesta por línea hasta ADIOS o hasta que cierre la conexión. */
    static void atender(Socket socket) {
        try (socket;
            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream(),
                StandardCharsets.UTF_8));
            PrintWriter out = new PrintWriter(new OutputStreamWriter(socket.getOutputStream(),
                StandardCharsets.UTF_8), true)) {
            String linea;
            while ((linea = in.readLine()) != null) {
                out.println(responder(linea));
                if (linea.trim().equalsIgnoreCase("ADIOS")) break;
            }
        } catch (IOException e) {
            System.out.println("Error con un cliente: " + e.getMessage());
        }
    }

    public static void main(String[] args) throws Exception {
        // Puerto 0: el sistema elige uno libre
        ServerSocket servidor = new ServerSocket(0);
    }
```

```

Thread aceptador = new Thread(() -> {
    try {
        while (true) {
            Socket cliente = servidor.accept();
            new Thread(() -> atender(cliente)).start();
        }
    } catch (IOException e) {
        // El servidor se ha cerrado: se termina el bucle
    }
});
aceptador.start();

// El cliente: cada «---» de la entrada abre una conexión nueva
Scanner sc = new Scanner(System.in);
int numero = 0;
Socket conexion = null;
BufferedReader in = null;
PrintWriter out = null;
while (sc.hasNextLine()) {
    String linea = sc.nextLine();
    if (conexion == null || linea.equals("---")) {
        if (conexion != null) conexion.close();
        conexion = new Socket("localhost", servidor.getLocalPort());
        in = new BufferedReader(new InputStreamReader(conexion.getInputStream(),
            StandardCharsets.UTF_8));
        out = new PrintWriter(new OutputStreamWriter(conexion.getOutputStream(),
            StandardCharsets.UTF_8), true);
        System.out.println("== Cliente " + (++numero) + " conectado");
        if (linea.equals("---")) continue;
    }
    out.println(linea);
    String respuesta = in.readLine();
    System.out.println(linea.isEmpty() ? ">" : "> " + linea);
    System.out.println("< " + (respuesta == null ? "(conexión cerrada)" : respuesta));
    if (respuesta != null && linea.trim().equalsIgnoreCase("ADIOS")) {
        conexion.close();
        conexion = null;
    }
}
if (conexion != null) conexion.close();
servidor.close();
aceptador.join();
System.out.println("Servidor cerrado · claves guardadas: " + almacen.size());
}

```

Separar el protocolo (`responder`) de la red (`atender`) es lo que hace manejable un servidor: la lógica se prueba como cualquier método y la parte de sockets queda reducida a leer una línea, responder y repetir.

Un hilo por cliente es el modelo más sencillo de servidor concurrente: mientras un hilo espera a que su cliente escriba, los demás siguen atendiendo. Para miles de conexiones se usa un pool de hilos o E/S no bloqueante, pero el protocolo sería el mismo.

El almacén es el único estado compartido, y por eso es un `ConcurrentHashMap`. Su `put` es atómico: con un `HashMap` y un `get` seguido de un `put`, dos clientes podrían creer a la vez que la clave era nueva.

Usar el puerto 0 hace que el sistema elija uno libre: el programa nunca falla porque el puerto esté ocupado. El cliente lo averigua con `getLocalPort()`, igual que en un test de integración real.

152. Almacén de contraseñas seguro con PBKDF2 y sal

Difícil · 70 minutos · apuntesdam.com/ejercicios/java/almacen-de-contrasenas-con-pbkdf2

Solución (java)

```
import java.security.MessageDigest;
import java.security.SecureRandom;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;

public class Main {
    static final String ALGORITMO = "PBKDF2WithHmacSHA256";
    static final int ITERACIONES = 10_000;    // en producción, cientos de miles: aquí menos para que las
    pruebas vayan rápido
    static final int MAX_FALLOS = 3;

    static class Cuenta {
        byte[] sal;
        byte[] hash;
        int fallos;
        boolean bloqueada;
    }

    static final Map<String, Cuenta> cuentas = new HashMap<>();
    static final SecureRandom azar = new SecureRandom();

    static byte[] derivar(String contrasena, byte[] sal) throws Exception {
        PBEKeySpec spec = new PBEKeySpec(contrasena.toCharArray(), sal, ITERACIONES, 256);
        return SecretKeyFactory.getInstance(ALGORITMO).generateSecret(spec).getEncoded();
    }

    /** null si la contraseña cumple la política, o el primer motivo por el que no. */
    static String politica(String usuario, String c) {
        if (c.length() < 10) return "debe tener al menos 10 caracteres";
        if (!c.matches(".*[A-Z].*") || !c.matches(".*[a-z].*") || !c.matches(".*\\d.*")) return "debe tener
            mayúsculas, minúsculas y números";
        if (c.toLowerCase().contains(usuario.toLowerCase())) return "no puede contener el nombre de
            usuario";
        return null;
    }

    static Cuenta nueva(String contrasena) throws Exception {
        Cuenta cuenta = new Cuenta();
        cuenta.sal = new byte[16];
        azar.nextBytes(cuenta.sal);    // una sal distinta para cada contraseña
        cuenta.hash = derivar(contrasena, cuenta.sal);
        return cuenta;
    }

    /** Comprueba la contraseña en tiempo constante (MessageDigest.isEqual no corta en el primer byte
        distinto). */
    static boolean correcta(Cuenta cuenta, String contrasena) throws Exception {
        return MessageDigest.isEqual(cuenta.hash, derivar(contrasena, cuenta.sal));
    }

    public static void main(String[] args) throws Exception {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String linea = sc.nextLine().trim();
            if (linea.isEmpty()) continue;
            String[] p = linea.split("\\s+");
            String orden = p[0].toUpperCase();
            System.out.print("> " + orden + (p.length > 1 ? " " + p[1] : "") + ": ");
            switch (orden) {
                case "REGISTRA" -> {
                    if (p.length != 3) { System.out.println("uso: REGISTRA usuario contraseña"); break; }
                    if (cuentas.containsKey(p[1])) { System.out.println("el usuario ya existe"); break; }
                    String motivo = politica(p[1], p[2]);
                    if (motivo != null) { System.out.println("contraseña rechazada, " + motivo); break; }
                    cuentas.put(p[1], nueva(p[2]));
                }
            }
        }
    }
}
```

```

        System.out.println("registrado");
    }
    case "ENTRA" -> {
        if (p.length != 3) { System.out.println("uso: ENTRA usuario contraseña"); break; }
        Cuenta cuenta = cuentas.get(p[1]);
        // El mismo mensaje si el usuario no existe: así no se puede averiguar qué usuarios hay
        if (cuenta == null) { System.out.println("usuario o contraseña incorrectos"); break; }
        if (cuenta.bloqueada) { System.out.println("cuenta bloqueada"); break; }
        if (correcta(cuenta, p[2])) {
            cuenta.fallos = 0;
            System.out.println("bienvenido");
        } else if (++cuenta.fallos >= MAX_FALLOS) {
            cuenta.bloqueada = true;
            System.out.println("usuario o contraseña incorrectos; cuenta bloqueada tras " +
                MAX_FALLOS + " fallos");
        } else {
            System.out.println("usuario o contraseña incorrectos");
        }
    }
    case "CAMBIA" -> {
        if (p.length != 4) { System.out.println("uso: CAMBIA usuario actual nueva"); break; }
        Cuenta cuenta = cuentas.get(p[1]);
        if (cuenta == null || cuenta.bloqueada || !correcta(cuenta, p[2])) {
            System.out.println("no se puede cambiar la contraseña"); break; }
        String motivo = politica(p[1], p[3]);
        if (motivo != null) { System.out.println("contraseña rechazada, " + motivo); break; }
        if (p[3].equals(p[2])) { System.out.println("la nueva contraseña debe ser distinta");
            break; }
        cuentas.put(p[1], nueva(p[3]));
        System.out.println("contraseña cambiada");
    }
    case "DESBLOQUEA" -> {
        Cuenta cuenta = p.length == 2 ? cuentas.get(p[1]) : null;
        if (cuenta == null) { System.out.println("no existe"); break; }
        cuenta.bloqueada = false;
        cuenta.fallos = 0;
        System.out.println("desbloqueada");
    }
    case "COMPARA" -> {
        Cuenta a = p.length == 3 ? cuentas.get(p[1]) : null, b = p.length == 3 ?
            cuentas.get(p[2]) : null;
        if (a == null || b == null) { System.out.println("no existe alguno de los usuarios");
            break; }
        System.out.println("¿mismo hash que " + p[2] + "? " + (MessageDigest.isEqual(a.hash,
            b.hash) ? "sí" : "no"));
    }
    case "INFO" -> {
        Cuenta cuenta = p.length == 2 ? cuentas.get(p[1]) : null;
        if (cuenta == null) { System.out.println("no existe"); break; }
        System.out.println(ALGORITMO + ", " + ITERACIONES + " iteraciones, sal de " +
            cuenta.sal.length
                + " bytes, hash de " + cuenta.hash.length + " bytes" + (cuenta.bloqueada ? ",
                bloqueada" : ""));
    }
    default -> System.out.println("orden desconocida");
}
}
}
}
}

```

La contraseña nunca se guarda: se guardan la sal y el resultado de PBKDF2. Para comprobar un intento se repite la derivación con la misma sal; si coincide, la contraseña era la correcta. Ni siquiera el administrador puede recuperar una contraseña, solo cambiarla.

La sal aleatoria explica el resultado de `COMPARA`: aunque dos usuarios elijan la misma contraseña, sus hashes son distintos. Sin sal, bastaría ver dos hashes iguales en la base de datos robada para saber que comparten contraseña.

Las iteraciones son un parámetro de coste: cada una multiplica el trabajo de quien intente adivinar contraseñas por fuerza bruta. Aquí son 10.000 para que las pruebas vayan rápidas; las guías actuales recomiendan cientos de miles para PBKDF2, o usar Argon2.

El bloqueo tras tres fallos y los mensajes que no distinguen entre usuario y contraseña cierran las otras dos vías de ataque: probar contraseñas contra una cuenta y averiguar qué cuentas existen.

153. Depura el programa de notas: cinco errores escondidos

Medio · 45 minutos · apuntesdam.com/ejercicios/java/depura-el-calculo-de-notas

Solución (java)

```
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;
import java.util.Scanner;

public class Main {
    record Alumno(String nombre, double nota) {}

    /** Nota final: prácticas al 20 % cada una y examen al 60 %; con menos de un 4 en el examen, como mucho un 4. */
    static double notaFinal(double practica1, double practica2, double examen) {
        double nota = 0.2 * practica1 + 0.2 * practica2 + 0.6 * examen;
        if (examen < 4) nota = Math.min(nota, 4);
        return Math.round(nota * 10) / 10.0;
    }

    static String uno(double x) {
        return String.format(Locale.ROOT, "%.1f", x).replace('.', ',');
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<String> lineas = new ArrayList<>();
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (!l.isEmpty()) lineas.add(l);
        }

        List<Alumno> alumnos = new ArrayList<>();
        for (int i = 0; i < lineas.size(); i++) {
            String[] p = lineas.get(i).split(";");
            alumnos.add(new Alumno(p[0], notaFinal(Double.parseDouble(p[1]), Double.parseDouble(p[2]),
                Double.parseDouble(p[3]))));
        }
        alumnos.sort((a, b) -> a.nota() != b.nota() ? Double.compare(b.nota(), a.nota()) :
            a.nombre().compareTo(b.nombre()));

        double suma = 0;
        int aprobados = 0;
        for (Alumno a : alumnos) {
            System.out.printf("%-20s%5s%n", a.nombre(), uno(a.nota()));
            suma += a.nota();
            if (a.nota() >= 5) aprobados++;
        }
        System.out.println("Media: " + uno(suma / alumnos.size()));
        System.out.println("Aprobados: " + aprobados + " de " + alumnos.size());
    }
}
```

Los cinco errores son clásicos: un índice que empieza en 1 y se salta al primer alumno; un copiar y pegar que suma dos veces la primera práctica; una comparación `<=` donde iba `<` (el examen de 4 sí compensa); una división entera que se come los decimales; y un `>` donde iba `>=` que deja fuera a quien saca un 5 justo.

Ninguno lo detecta el compilador ni produce una excepción: el programa da resultados plausibles. Solo se ven con casos de prueba que pasan por los bordes (un 4 en el examen, un 5 de nota final, el primer alumno de la lista) o calculando a mano un resultado.

El depurador ahorra tiempo cuando ya sabes dónde mirar: un punto de ruptura en `notaFinal` y la vista de variables muestran el valor de `nota` antes y después de cada línea, y ahí aparecen la práctica sumada dos veces y la división entera.

Cada error corregido debería convertirse en una prueba automática (una prueba de regresión), para que nadie lo vuelva a introducir. Es lo que hacen las pruebas ocultas de este ejercicio.