

Ejercicios resueltos de Programación

Los 62 ejercicios de Programación de la web, tema a tema: 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

Fundamentos de la Programación

[1. Del pseudocódigo a Java](#)

[3. Conversor de bases sin atajos](#)

[2. Traza del algoritmo de Euclides](#)

[4. Simula una máquina con su propio ensamblador](#)

Introducción a Java

[5. Desglose de segundos](#)

[7. La cuota de un préstamo](#)

[6. Las trampas de los tipos numéricos](#)

[8. Permisos de Unix con operadores de bits](#)

Estructuras Condicionales

[9. Calificación a partir de la nota](#)

[11. Encuentra el fallo: la nota en palabras](#)

[13. Validar una fecha y calcular el día siguiente](#)

[10. Encuentra el fallo: años bisiestos](#)

[12. Año bisiesto](#)

[14. IRPF por tramos \(simplificado\)](#)

Bucles

[15. Tabla de multiplicar](#)

[17. Suma de los dígitos](#)

[19. La conjetura de Collatz](#)

[16. Encuentra el fallo: la suma de 1 a n](#)

[18. Encuentra el fallo: la media que pierde decimales](#)

[20. El calendario de un mes](#)

Métodos

[21. Números primos con una función](#)

[23. Encuentra el fallo: el método que no cambia nada](#)

[25. Una calculadora con descenso recursivo](#)

[22. Factorial recursivo](#)

[24. Las torres de Hanói](#)

Arrays

[26. Máximo y mínimo de un array](#)

[28. Invertir un array](#)

[30. Suma de filas y columnas de una matriz](#)

[27. Encuentra el fallo: el máximo de temperaturas bajo cero](#)

[29. Estadísticas completas de notas](#)

[31. El juego de la vida de Conway](#)

Strings

[32. Contar vocales](#)

[34. Palíndromos](#)

[36. El cifrado de Vigenère](#)

[33. Encuentra el fallo: la contraseña que nunca es correcta](#)

[35. Encuentra el fallo: el palíndromo exigente](#)

[37. Justificar un texto](#)

Clases y Objetos

[38. Encuentra el fallo: el alumno sin nombre](#)

[40. Máquina expendedora](#)

[39. Una clase Fraccion inmutable](#)

Herencia

[41. Cuentas bancarias con herencia](#)

[43. Expresiones matemáticas con polimorfismo](#)

[42. Encuentra el fallo: el producto que no está en la lista](#)

Interfaces

[44. Clasificación de una liga con Comparator](#)

[45. Reglas de contraseñas componibles](#)

Ficheros y Streams

46. Validar y resumir un CSV de ventas

Colecciones

48. Encuentra el fallo: borrar mientras se recorre

50. Agrupar anagramas

Ejercicios largos

52. Biblioteca: préstamos, devoluciones y multas

54. Cajero automático con PIN, límite diario y billetes

56. Cifrado de Vigenère con el alfabeto español

58. Evaluador de expresiones con pilas y variables

60. Nóminas de una plantilla con distintos contratos

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

47. Comparar dos ficheros (diff)

49. Encuentra el fallo: el contador de palabras olvidadizo

51. Ruta más corta en el metro

53. Clasificación de una liga de fútbol

55. Buscaminas: números, casilla más peligrosa y zonas despejadas

57. Parking con tarifas por tipo de vehículo

59. Conversor de CSV a JSON con comillas y tipos

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

Fundamentos de la Programación

1. 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

2. 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

3. 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 100000000000 · octal 10000 · hexadecimal 1000

4. 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
```

Introducción a Java

5. 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)
```

6. 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)
```

7. 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

8. 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
```

Estructuras Condicionales

9. 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

10. 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

11. 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

12. 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

13. 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)

14. 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

Bucles

15. 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

16. 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

17. 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

18. 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

19. 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

20. 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

Métodos

21. 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

22. 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

$0! = 1$

23. 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

24. 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 ($2^1 - 1 = 1$)

25. 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
```

Arrays

26. 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)
```

27. 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

28. 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]

29. 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
```

30. 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
```

31. 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
```

Strings

32. 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

33. 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

34. 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

35. 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

36. 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

37. 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
```

Clases y Objetos

38. 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
```

39. 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
```

40. 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

```

Herencia

41. 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
```

42. 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
```

43. 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)

```

Interfaces

44. 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

45. 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»
```

Ficheros y Streams

46. 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
```

47. 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 @@
```

Colecciones

48. 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]
```

49. 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
```

50. 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

51. 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)
```

Ejercicios largos

52. Biblioteca: préstamos, devoluciones y multas

Difícil · Java · 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);
    }
}

```

53. Clasificación de una liga de fútbol

Difícil · Java · 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
1	Atlético Norte	3	3	0	0	6	2	+4
2	Real Sur	3	1	1	1	5	5	0
3	Deportivo Este	3	0	2	1	3	5	-2
4	Unión Oeste	3	0	1	2	1	3	-2

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
1	Ciudad Alta	3	1	1	1	2	3	-1
2	Puerto Bajo	2	1	0	1	3	2	+1
3	Valle Verde	2	1	0	1	3	3	0
4	Montaña Azul	1	0	1	0	1	1	0

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%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
    }
}
```

54. Cajero automático con PIN, límite diario y billetes

Medio · Java · 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
        }
    }
}
```

55. Buscaminas: números, casilla más peligrosa y zonas despejadas

Difícil · Java · 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
    }
}
```

56. Cifrado de Vigenère con el alfabeto español

Difícil · Java · 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 Lbmfho 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: Lbmfho 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 = "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) {
        // 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
        }
    }
}
```

57. Parking con tarifas por tipo de vehículo

Difícil · Java · 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

```

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
    }
}

```

58. Evaluador de expresiones con pilas y variables

Muy difícil · Java · 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 línea = sc.nextLine().trim();
            // TODO: VARIABLES, asignaciones y expresiones
        }
    }
}
```

59. Conversor de CSV a JSON con comillas y tipos

Difícil · Java · 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
    }
}
```

60. Nóminas de una plantilla con distintos contratos

Difícil · Java · 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
    }
}
```

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

Medio · Java · 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
    }
}
```

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

Muy difícil · Java · 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);
            }
        }
    }
}
```

Soluciones

1. 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.

2. 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».

3. 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.

4. 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.

5. 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.

6. 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`.

7. 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.

8. 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.

9. 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.

10. 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.

11. 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.

12. 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.

13. 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 bisiesto = (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, bisiesto ? 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) +
                (bisiesto ? " (año bisiesto)" : ""));
        }
    }
}
```

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.

14. 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.

15. 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».

16. 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`.

17. 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.

18. 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.

19. 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.

20. 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.

21. 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.

22. 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`.

23. 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);`.

24. 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.

25. 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`.

26. 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.

27. 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`).

28. 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.

29. 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.

30. 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.

31. 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.

32. 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.

33. 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).

34. 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.

35. 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()`.

36. 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 restándole 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.

37. 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.

38. 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.

39. 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».

40. 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.

41. 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.

42. 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.

43. 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.

44. 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.

45. 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».

46. 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.

47. 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.

48. 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()`.

49. 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)`.

50. 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.

51. 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.

52. Biblioteca: préstamos, devoluciones y multas

Difícil · Java · 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.

53. Clasificación de una liga de fútbol

Difícil · Java · 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.

54. Cajero automático con PIN, límite diario y billetes

Medio · Java · 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(Locale.ROOT, "%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.

55. Buscaminas: números, casilla más peligrosa y zonas despejadas

Difícil · Java · 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][];
        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.

56. Cifrado de Vigenère con el alfabeto español

Difícil · Java · 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.

57. Parking con tarifas por tipo de vehículo

Difícil · Java · 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.

58. Evaluador de expresiones con pilas y variables

Muy difícil · Java · 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 ("+-*/%().index".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.

59. Conversor de CSV a JSON con comillas y tipos

Difícil · Java · 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.

60. Nóminas de una plantilla con distintos contratos

Difícil · Java · 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.

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

Medio · Java · 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.

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

Muy difícil · Java · 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.