

Ejercicios resueltos de Desarrollo de Interfaces

Los 28 ejercicios de Desarrollo de Interfaces 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

Generación de Interfaces Gráficas

[1. La validación de un formulario de registro](#)

[3. Captura y burbuja de un evento](#)

Interfaces Naturales de Usuario

[4. Un intérprete de órdenes de voz](#)

[6. ¿Qué gesto hace la mano?](#)

Creación de Componentes Visuales

[7. Un componente de valoración por estrellas](#)

[9. Un campo de texto con máscara](#)

Usabilidad y Accesibilidad

[10. Contraste de colores según WCAG](#)

[12. ¿Se entiende este texto?](#)

Documentación de Aplicaciones

[13. El CHANGELOG a partir de los commits](#)

[15. Un revisor automático de manuales](#)

Distribución de Aplicaciones

[16. Versiones semánticas y rangos de dependencias](#)

[18. Un actualizador incremental con despliegue escalonado](#)

Pruebas de Interfaces

[19. Casos de prueba por valores límite](#)

[21. Pruebas de interfaz guionizadas](#)

Ejercicios de Desarrollo de Interfaces

[22. Validar un NIF en un formulario](#)

[24. Paginación de una tabla](#)

Ejercicios largos

[26. Componente web accesible: valoración con estrellas](#)

[28. Página de un evento accesible: HTML semántico y ARIA](#)

[2. Repartir el ancho como un GridPane](#)

[5. Clasificar gestos táctiles](#)

[8. Propiedades observables y bindings](#)

[11. El orden de tabulación de un formulario](#)

[14. De Javadoc a una página de documentación](#)

[17. ¿Puedo distribuir mi aplicación con estas librerías?](#)

[20. Un mini ejecutor de pruebas unitarias](#)

[23. Informe agrupado con subtotales](#)

[25. Comparar versiones semánticas](#)

[27. Informe de ventas imprimible: agrupación, formatos y CSS de impresión](#)

Generación de Interfaces Gráficas

1. La validación de un formulario de registro

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

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

Código de partida (java)

```
import java.util.*;

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

Ejemplo: Formulario correcto

ENTRADA

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

SALIDA ESPERADA

Formulario válido: botón Registrar activado

Ejemplo: Varios errores

ENTRADA

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

SALIDA ESPERADA

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

2. Repartir el ancho como un GridPane

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

Un contenedor de rejilla (GridPane de JavaFX, GridBagLayout de Swing o grid de CSS) reparte su ancho entre columnas. La entrada es «ancho W», «hueco H» (separación entre columnas) y una línea por columna: «nombre fija N» (N píxeles), «nombre porcentaje P» (P % del espacio disponible) o «nombre crece» (se reparte a partes iguales lo que sobra). El espacio disponible es $W - H \times (\text{columnas} - 1)$. Muestra para cada columna «nombre: x=X ancho=A» (1 decimal, x empieza en 0 y entre columnas se suma el hueco). Si no hay columnas que crezcan y sobra espacio, añade

«Sobran S px a la derecha». Si lo fijo y los porcentajes ya superan el disponible, o no queda nada para las columnas que crecen, muestra solo «No cabe: faltan F px» (lo que excede, más 1 px por cada columna que crece, que necesita al menos uno).

Código de partida (java)

```
import java.util.*;

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

Ejemplo: Formulario de tres columnas

ENTRADA

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

SALIDA ESPERADA

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

Ejemplo: Dos que crecen

ENTRADA

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

SALIDA ESPERADA

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

Ejemplo: Sobra espacio

ENTRADA

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

SALIDA ESPERADA

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

3. Captura y burbuja de un evento

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

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

Código de partida (java)

```
import java.util.*;

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

Ejemplo: Sin consumir

ENTRADA

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

SALIDA ESPERADA

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

Ejemplo: Un filtro consume

ENTRADA

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

SALIDA ESPERADA

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

Interfaces Naturales de Usuario

4. Un intérprete de órdenes de voz

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

El reconocedor de voz (SpeechRecognizer en Android) devuelve texto; la aplicación debe extraer la intención y sus datos. Cada línea es una transcripción. Normalízala (la función normalizar quita tildes, signos y espacios de más) y reconoce, en este orden: una alarma («pon/poner/pone/programa [una] alarma a las H[:MM] [y media|y cuarto]») → «ALARMA HH:MM» con dos cifras, o «ERROR: hora no válida» si la hora pasa de 23 o los minutos de 59; una llamada («llama/llamar a NOMBRE») → «LLAMAR Nombre Apellido» con la inicial de cada palabra en mayúscula (sin tildes,

porque se han quitado al normalizar); el volumen («sube/baja [el] volumen [N]», 10 si no se dice) → «VOLUMEN +N» o «VOLUMEN -N»; abrir una aplicación («abre/abrir/inicia [la app|la aplicacion] NOMBRE») → «ABRIR nombre». Si no encaja nada, «NO ENTENDIDO: frase original».

Código de partida (java)

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

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

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

Ejemplo: Órdenes habituales

ENTRADA

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

SALIDA ESPERADA

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

Ejemplo: Variantes

ENTRADA

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

SALIDA ESPERADA

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

5. Clasificar gestos táctiles

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

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

Código de partida (java)

```
import java.util.*;

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

Ejemplo: Gestos básicos

ENTRADA

```
0 100 200
80 102 201

0 50 300
120 200 305
200 320 310

0 300 300
700 303 302
```

SALIDA ESPERADA

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

Ejemplo: Vertical y lento

ENTRADA

```
0 200 600
150 205 300

0 100 100
1000 260 100

0 10 10
400 12 13
```

SALIDA ESPERADA

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

6. ¿Qué gesto hace la mano?

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

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

Código de partida (java)

```
import java.util.*;

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

Ejemplo: Gestos básicos

ENTRADA

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

SALIDA ESPERADA

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

Ejemplo: Mano girada y pinza

ENTRADA

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

SALIDA ESPERADA

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

Creación de Componentes Visuales

7. Un componente de valoración por estrellas

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

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

Código de partida (java)

```
import java.util.*;

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

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

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

Ejemplo: Uso normal

ENTRADA

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

SALIDA ESPERADA

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

8. Propiedades observables y bindings

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

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

Código de partida (java)

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

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

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

Ejemplo: Enlace sencillo

ENTRADA

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

SALIDA ESPERADA

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

Ejemplo: Cadena y ciclo

ENTRADA

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

SALIDA ESPERADA

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

9. Un campo de texto con máscara

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

componentes

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

Código de partida (java)

```
import java.util.*;

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

Ejemplo: Fecha

ENTRADA

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

SALIDA ESPERADA

```
Pantalla: 15/03/2026
Completo: sí
```

Ejemplo: Matrícula con errores

ENTRADA

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

SALIDA ESPERADA

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

Ejemplo: Teléfono con borrado

ENTRADA

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

SALIDA ESPERADA

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

Usabilidad y Accesibilidad

10. Contraste de colores según WCAG

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

Cada línea tiene dos colores en hexadecimal (#rrggbb o #rgb): el del texto y el del fondo. Implementa la luminancia relativa de WCAG (cada canal se divide entre 255 y se linealiza: $v \leq 0.03928 \rightarrow v / 12.92$; si no, $((v + 0.055) / 1.055)^{2.4}$; después $L = 0.2126 R + 0.7152 G + 0.0722 B$) y el contraste $(L_{\text{clara}} + 0.05) / (L_{\text{oscura}} + 0.05)$. Muestra

«texto sobre fondo: R:1 · texto normal X · texto grande Y», con R con 2 decimales; para texto normal, AAA si $R \geq 7$, AA si $R \geq 4.5$; para texto grande (18 pt, o 14 pt en negrita), AAA si $R \geq 4.5$, AA si $R \geq 3$; si no llega, «no cumple».

Código de partida (java)

```
import java.util.*;

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

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

Ejemplo: Combinaciones habituales

ENTRADA

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

SALIDA ESPERADA

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

11. El orden de tabulación de un formulario

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

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

Código de partida (java)

```
import java.util.*;

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

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

Ejemplo: Formulario bien hecho

ENTRADA

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

SALIDA ESPERADA

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

Ejemplo: Con tabindex positivos y deshabilitados

ENTRADA

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

SALIDA ESPERADA

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

12. ¿Se entiende este texto?

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

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

Código de partida (java)

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

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

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

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

Ejemplo: Mensaje de error claro

ENTRADA

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

SALIDA ESPERADA

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

Ejemplo: Texto administrativo

ENTRADA

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

SALIDA ESPERADA

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

Ejemplo: Palabras con diptongos e hiatos

ENTRADA

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

SALIDA ESPERADA

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

Documentación de Aplicaciones

13. El CHANGELOG a partir de los commits

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

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

Código de partida (java)

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

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

Ejemplo: Versión menor

ENTRADA

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

SALIDA ESPERADA

```
## 1.5.0

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

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

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

Ejemplo: Cambio incompatible

ENTRADA

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

SALIDA ESPERADA

```
## 3.0.0

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

### Novedades
- modo oscuro

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

Ejemplo: Sin cambios publicables

ENTRADA

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

SALIDA ESPERADA

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

14. De Javadoc a una página de documentación

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

Las herramientas como javadoc o Dokka generan la documentación de una API a partir de los comentarios `/** ... */`. La entrada es el código de una clase Java. Para cada método público (una línea que empieza por «public», contiene «(» y no es la declaración de la clase), en orden: si justo antes tiene un comentario Javadoc, genera «### nombre», la firma entre comillas invertidas (sin «public» ni lo que va desde la llave), una línea en blanco y la descripción (las líneas de texto del comentario unidas con un espacio); después, si los hay, una línea en blanco y «**Parámetros**» con «- nombre : descripción» por cada @param; «**Devuelve:** ...» para @return; y «**Lanza**» con «- Excepción : descripción» por cada @throws; y una línea en blanco final. Un Javadoc seguido de un método no público se descarta. Termina con «Documentados: N · Sin documentar: método1, método2» (o «ninguno») con los públicos sin Javadoc.

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<String> lineas = new ArrayList<>();
        while (sc.hasNextLine()) lineas.add(sc.nextLine());
        // TODO: extrae los comentarios Javadoc de los métodos públicos y genera la documentación en
        // Markdown
    }
}
```

Ejemplo: Clase Carrito

ENTRADA

```
public class Carrito {
    private final List<Linea> lineas = new
    ArrayList<>();

    /**
     * Añade un producto al carrito. Si ya
    estaba,
     * suma la cantidad.
     * @param producto producto que se añade
     * @param cantidad unidades, mayor que cero
     * @throws IllegalArgumentException si la
    cantidad no es positiva
     */
    public void anadir(Producto producto, int
    cantidad) {
        // ...
    }

    /** Importe total con IVA.
     * @return el total en euros, redondeado a
    dos decimales */
    public double total() {
        return 0;
    }

    public void vaciar() {
        lineas.clear();
    }

    /**
     * Cuenta las unidades de todas las líneas.
     */
    private int unidades() {
        return 0;
    }

    public boolean estaVacio() { return
    lineas.isEmpty(); }
}
```

SALIDA ESPERADA

```
### anadir
`void anadir(Producto producto, int cantidad)`

Añade un producto al carrito. Si ya estaba, suma
la cantidad.

**Parámetros**
- `producto`: producto que se añade
- `cantidad`: unidades, mayor que cero

**Lanza**
- `IllegalArgumentException`: si la cantidad no
es positiva

### total
`double total()`

Importe total con IVA.

**Devuelve**: el total en euros, redondeado a dos
decimales

Documentados: 2 · Sin documentar: vaciar,
estaVacio
```

15. Un revisor automático de manuales

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

Igual que se pasa un linter al código, se puede revisar la documentación. La entrada es un manual en Markdown. Ignorando lo que hay dentro de bloques de código (entre líneas que empiezan por ````), detecta en este orden: (1) en una primera pasada por los títulos, que el documento empiece con un título de nivel 1 («Línea N: el documento debe empezar con un título de nivel 1») y que no se salten niveles («Línea N: salto de título de nivel A a B», por ejemplo de `##` a `####`), guardando el ancla de cada título (minúsculas, sin tildes, solo letras, números, espacios y guiones, y los espacios cambiados por guiones); (2) en una segunda pasada, las imágenes `![texto](ruta)` sin texto alternativo («Línea N: la imagen RUTA no tiene texto alternativo»), los enlaces cuyo texto es «aquí», «aquí», «pulsa aquí» o «este enlace» («Línea N: el texto del enlace «X» no dice adónde lleva») y los enlaces internos `#ancla` que no existen («Línea N: el enlace `#X` apunta a una sección que no existe»); (3) si queda un bloque de código sin cerrar, «Hay un bloque de código sin cerrar». Termina con «N problemas» o «Manual correcto».

Código de partida (java)

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

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<String> lineas = new ArrayList<>();
        while (sc.hasNextLine()) lineas.add(sc.nextLine());
        // TODO: revisa el manual y muestra los problemas encontrados
    }
}
```

Ejemplo: Manual con problemas

ENTRADA

```
# Manual de usuario de Reservas

Consulta la [instalación](#instalacion) antes de
empezar.

## Instalación
Descarga el instalador [aquí]
(https://ejemplo.es/descargas).

#### Requisitos


## Crear una reserva
1. Pulsa **Nueva**.
2. Revisa la sección de [errores frecuentes]
(#errores-frecuentes).

...

# esto es un comentario, no un título

...

## Errores habituales
![Mensaje de error de conexión](img/error.png)
```

SALIDA ESPERADA

Línea 8: salto de título de nivel 2 a 4
Línea 6: el texto del enlace «aquí» no dice
adónde lleva
Línea 9: la imagen img/requisitos.png no tiene
texto alternativo
Línea 13: el enlace #errores-frecuentes apunta a
una sección que no existe
4 problemas

Ejemplo: Manual correcto

ENTRADA

```
# Guía

## Uso
Ver [la instalación](#instalacion).

## Instalación
![Pantalla de inicio](inicio.png)
```

SALIDA ESPERADA

Manual correcto

Distribución de Aplicaciones

16. Versiones semánticas y rangos de dependencias

Medio · Distribución de Aplicaciones · apuntesdam.com/subject/desarrollo-interfaces/topic/distribucion-aplicaciones

Los gestores de paquetes (Maven, npm, Gradle) eligen versiones comparándolas según el versionado semántico. La primera línea es una lista de versiones separadas por espacios, que pueden tener una etiqueta de prelanzamiento («1.2.0-beta.2»); el programa las ordena con `comparar()` y las muestra. Cada línea siguiente es un rango: «`^X.Y.Z`»

acepta versiones desde X.Y.Z hasta antes de la siguiente mayor (si X es 0, hasta la siguiente menor; si X e Y son 0, solo ese parche), «~X.Y.Z» hasta antes de la siguiente menor, y una versión sola solo esa. Para cada rango, el programa muestra la versión estable (sin etiqueta) más alta que lo cumple, o «ninguna». Implementa comparar: primero por los tres números; una versión sin etiqueta es mayor que la misma con etiqueta; entre etiquetas se comparan sus partes separadas por puntos (las numéricas como números y menores que las de texto, y las de texto alfabéticamente) y, si una se acaba antes, es menor.

Código de partida (java)

```
import java.util.*;

public class Main {
    // TODO: compara dos versiones semánticas (negativo si a < b, 0 si iguales, positivo si a > b)
    static int comparar(String a, String b) { return 0; }

    // TODO: ¿cumple la versión el rango? (^X.Y.Z, ~X.Y.Z o una versión exacta)
    static boolean cumple(String version, String rango) { return false; }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<String> versiones = new ArrayList<>(Arrays.asList(sc.nextLine().trim().split(" ")));
        versiones.sort(Main::comparar);
        System.out.println("Ordenadas: " + String.join(" < ", versiones));
        while (sc.hasNextLine()) {
            String rango = sc.nextLine().trim();
            if (rango.isEmpty()) continue;
            String mejor = null;
            for (String v : versiones) if (cumple(v, rango) && !v.contains("-")) mejor = v;    // la más alta estable
            System.out.println(rango + " → " + (mejor == null ? "ninguna" : mejor));
        }
    }
}
```

Ejemplo: Dependencia típica

ENTRADA

```
1.2.0 1.10.1 1.2.10 2.0.0 1.9.0-beta 2.0.0-rc.1
1.2.3
^1.2.0
~1.2.0
^2.0.0
1.9.0
```

SALIDA ESPERADA

```
Ordenadas: 1.2.0 < 1.2.3 < 1.2.10 < 1.9.0-beta <
1.10.1 < 2.0.0-rc.1 < 2.0.0
^1.2.0 → 1.10.1
~1.2.0 → 1.2.10
^2.0.0 → 2.0.0
1.9.0 → ninguna
```

Ejemplo: Versiones 0.x

ENTRADA

```
0.3.1 0.3.9 0.4.0 0.0.3 0.0.4 1.0.0-alpha
^0.3.1
^0.0.3
~0.3.0
```

SALIDA ESPERADA

```
Ordenadas: 0.0.3 < 0.0.4 < 0.3.1 < 0.3.9 < 0.4.0
< 1.0.0-alpha
^0.3.1 → 0.3.9
^0.0.3 → 0.0.3
~0.3.0 → 0.3.9
```

17. ¿Puedo distribuir mi aplicación con estas librerías?

Difícil · Distribución de Aplicaciones · apuntesdam.com/subject/desarrollo-interfaces/topic/distribucion-aplicaciones

Antes de distribuir hay que revisar las licencias de las dependencias. La primera línea es «proyecto LICENCIA» (MIT, Apache-2.0, GPL-3.0, GPL-2.0 o Propietaria) y cada línea siguiente una dependencia «nombre licencia enlace», con enlace estatico (incluida en el mismo JAR) o dinamico (un JAR aparte que se carga). Reglas: MIT, BSD-3-Clause, Apache-2.0 y EPL-2.0 son permisivas → «OK (permisiva: incluye su aviso de copyright)», salvo Apache-2.0 en un proyecto GPL-2.0 → «PROBLEMA (Apache-2.0 no es compatible con GPL-2.0)»; LGPL-3.0 dinámica → «OK (LGPL enlazada dinámicamente: el usuario debe poder sustituirla)» (cuenta como aviso), estática en proyecto GPL → «OK

(compatible con GPL)», estática en otro → «PROBLEMA (LGPL enlazada estáticamente obliga a facilitar el código objeto para reenlazar)»; GPL-3.0 en proyecto GPL-3.0 → «OK (misma licencia)», GPL-2.0 en proyecto GPL-3.0 → «PROBLEMA (GPL-2.0 sola no es compatible con GPL-3.0)», cualquier GPL en otro proyecto → «PROBLEMA (copyleft fuerte: obligaría a publicar el proyecto como LICENCIA)»; AGPL-3.0 → «PROBLEMA (AGPL: también obliga a compartir el código si se ofrece como servicio en red)»; cualquier otra → «REVISAR (licencia desconocida: hay que leerla)» (aviso). Muestra «nombre [licencia, enlace]: veredicto» y «Resumen: P problemas, A avisos → se puede distribuir» o «→ no distribuir sin resolverlos».

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] proyecto = sc.nextLine().trim().split(" ");    // proyecto LICENCIA (MIT, Apache-2.0,
                                                                // GPL-3.0, Propietaria)
        String licProyecto = proyecto[1];
        List<String[]> deps = new ArrayList<>();                // nombre licencia
                                                                // enlace(estatico|dinamico)
        while (sc.hasNextLine()) { String l = sc.nextLine().trim(); if (!l.isEmpty()) deps.add(l.split("
")); }
        // TODO: revisa si cada dependencia es compatible con la licencia del proyecto
    }
}
```

Ejemplo: App propietaria

ENTRADA

```
proyecto Propietaria
jackson Apache-2.0 estatico
openpdf LGPL-3.0 dinamico
mysql-connector GPL-2.0 dinamico
logback EPL-2.0 estatico
hibernate LGPL-3.0 estatico
```

SALIDA ESPERADA

```
jackson [Apache-2.0, estatico]: OK (permisiva:
incluye su aviso de copyright)
openpdf [LGPL-3.0, dinamico]: OK (LGPL enlazada
dinámicamente: el usuario debe poder sustituirla)
mysql-connector [GPL-2.0, dinamico]: PROBLEMA
(copyleft fuerte: obligaría a publicar el
proyecto como GPL-2.0)
logback [EPL-2.0, estatico]: OK (permisiva:
incluye su aviso de copyright)
hibernate [LGPL-3.0, estatico]: PROBLEMA (LGPL
enlazada estáticamente obliga a facilitar el
código objeto para reenlazar)
Resumen: 2 problemas, 1 avisos → no distribuir
sin resolverlos
```

Ejemplo: App GPL

ENTRADA

```
proyecto GPL-3.0
libreria-gpl GPL-3.0 estatico
vieja GPL-2.0 estatico
jsoup MIT estatico
rara WTFPL estatico
```

SALIDA ESPERADA

```
libreria-gpl [GPL-3.0, estatico]: OK (misma
licencia)
vieja [GPL-2.0, estatico]: PROBLEMA (GPL-2.0 sola
no es compatible con GPL-3.0)
jsoup [MIT, estatico]: OK (permisiva: incluye su
aviso de copyright)
rara [WTFPL, estatico]: REVISAR (licencia
desconocida: hay que leerla)
Resumen: 1 problemas, 1 avisos → no distribuir
sin resolverlos
```

18. Un actualizador incremental con despliegue escalonado

Muy difícil · Distribución de Aplicaciones · apuntesdam.com/subject/desarrollo-interfaces/topic/distribucion-aplicaciones

Las aplicaciones de escritorio modernas se actualizan solas descargando solo lo que ha cambiado y publicando la versión nueva poco a poco. La entrada tiene una configuración «clave valor» (version_instalada, version_nueva, minima, porcentaje y usuario) y dos manifiestos, tras «[local]» y «[remoto]», con líneas «ruta tamaño_bytes hash». Si la versión instalada ya es la nueva o superior: «Ya tienes la última versión (X)». Si es menor que la mínima, la actualización es obligatoria. Si no, el usuario la recibe solo si su grupo, $\text{Math.floorMod}(\text{usuario.hashCode()}, 100)$, es menor que el porcentaje publicado; si no, «Actualización N aún no disponible para este usuario (grupo G, se publica al P %)». Cuando hay actualización: «Actualizar de A a N (obligatoria: la versión mínima es M)» o «... (grupo G)», después « NUEVO ruta», « CAMBIA ruta» (mismo nombre, distinto hash) en orden alfabético, « BORRAR ruta» para lo que ya no existe, y «Descarga: F ficheros, X KB de T KB (ahorro del S %)» (KB = 1024 bytes, 1 decimal; ahorro sin decimales).

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, String> conf = new HashMap<>();
        Map<String, String[]> local = new TreeMap<>(), remoto = new TreeMap<>();    // ruta → {tamaño, hash}
        Map<String, String[]> actual = null;
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (l.isEmpty()) continue;
            if (l.equals("[local]")) { actual = local; continue; }
            if (l.equals("[remoto]")) { actual = remoto; continue; }
            String[] p = l.split(" ");
            if (actual == null) conf.put(p[0], p[1]);                // version_instalada, version_nueva, minima, porcentaje, usuario
            else actual.put(p[0], new String[]{p[1], p[2]});
        }
        // TODO: decide si este usuario recibe la actualización y qué ficheros descargar o borrar
    }
}
```

Ejemplo: Actualización parcial

ENTRADA

```
version_instalada 2.3.1
version_nueva 2.4.0
minima 2.0.0
porcentaje 100
usuario ana@hotel.es
[local]
app.jar 812000 a1f3
lib/javafx-controls.jar 2400000 9c0d
lib/openpdf.jar 1300000 77aa
informes/factura.jasper 45000 c3c3
ayuda/manual.html 120000 e5e5
[remoto]
app.jar 830000 b2b2
lib/javafx-controls.jar 2400000 9c0d
lib/openpdf.jar 1350000 88bb
informes/factura.jasper 45000 c3c3
informes/ocupacion.jasper 52000 d4d4
```

SALIDA ESPERADA

```
Actualizar de 2.3.1 a 2.4.0 (grupo 48)
CAMBIA app.jar
NUEVO informes/ocupacion.jasper
CAMBIA lib/openpdf.jar
BORRAR ayuda/manual.html
Descarga: 3 ficheros, 2179.7 KB de 4567.4 KB
(ahorro del 52 %)
```

Ejemplo: Aún no publicada para este usuario

ENTRADA

```
version_instalada 2.3.1
version_nueva 2.4.0
minima 2.0.0
porcentaje 10
usuario ana@hotel.es
[local]
app.jar 812000 a1f3
lib/javafx-controls.jar 2400000 9c0d
lib/openpdf.jar 1300000 77aa
informes/factura.jasper 45000 c3c3
ayuda/manual.html 120000 e5e5
[remoto]
app.jar 830000 b2b2
lib/javafx-controls.jar 2400000 9c0d
lib/openpdf.jar 1350000 88bb
informes/factura.jasper 45000 c3c3
informes/ocupacion.jasper 52000 d4d4
```

SALIDA ESPERADA

Actualización 2.4.0 aún no disponible para este usuario (grupo 48, se publica al 10 %)

Ejemplo: Obligatoria y al día

ENTRADA

```
version_instalada 1.9.5
version_nueva 2.4.0
minima 2.0.0
porcentaje 0
usuario luis
[local]
app.jar 812000 a1f3
lib/javafx-controls.jar 2400000 9c0d
lib/openpdf.jar 1300000 77aa
informes/factura.jasper 45000 c3c3
ayuda/manual.html 120000 e5e5
[remoto]
app.jar 812000 a1f3
lib/javafx-controls.jar 2400000 9c0d
lib/openpdf.jar 1350000 88bb
informes/factura.jasper 45000 c3c3
informes/ocupacion.jasper 52000 d4d4
```

SALIDA ESPERADA

Actualizar de 1.9.5 a 2.4.0 (obligatoria: la versión mínima es 2.0.0)
NUEVO informes/ocupacion.jasper
CAMBIA lib/openpdf.jar
BORRAR ayuda/manual.html
Descarga: 2 ficheros, 1369.1 KB de 4549.8 KB
(ahorro del 70 %)

Pruebas de Interfaces

19. Casos de prueba por valores límite

Medio · Pruebas de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/pruebas-interfaces

Los errores se esconden en los bordes. Cada línea describe un campo de un formulario: «campo tipo min max», con tipo entero (el valor debe estar entre min y max) o texto (la longitud debe estar entre min y max). Genera sus casos de prueba: «== campo (tipo, min..max)» y, numerados « campo-N: valor → resultado», primero los valores límite min – 1, min, min + 1, max – 1, max y max + 1 (sin repetir y, en texto, sin longitudes negativas), cada uno «válido» o «inválido». En los enteros, el valor límite se muestra tal cual, y después se añaden un valor intermedio ($(\text{min} + \text{max}) / 2$ → válido, si no estaba ya) y «abc» → «inválido (no es un número)». En los textos, una longitud 0 se muestra como "", hasta 12 como la cadena de «a» entre comillas seguida de «(N caracteres)» («(1 carácter)» en singular) y más larga como «texto de N caracteres»; después se añade « " " (solo espacios) → inválido (vacío tras recortar)». En ambos, el último caso es «(vacío) → inválido (obligatorio)».

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");    // campo tipo min max
            if (p.length < 4) continue;
            String campo = p[0], tipo = p[1];
            int min = Integer.parseInt(p[2]), max = Integer.parseInt(p[3]);
            // TODO: genera los casos de prueba por valores límite y clases de equivalencia
        }
    }
}
```

Ejemplo: Edad y nombre

ENTRADA

edad entero 18 65
nombre texto 2 10

SALIDA ESPERADA

```
== edad (entero, 18..65)
edad-1: 17 → inválido
edad-2: 18 → válido
edad-3: 19 → válido
edad-4: 64 → válido
edad-5: 65 → válido
edad-6: 66 → inválido
edad-7: 41 → válido
edad-8: abc → inválido (no es un número)
edad-9: (vacío) → inválido (obligatorio)
== nombre (texto, 2..10)
nombre-1: "a" (1 carácter) → inválido
nombre-2: "aa" (2 caracteres) → válido
nombre-3: "aaa" (3 caracteres) → válido
nombre-4: "aaaaaaaa" (9 caracteres) → válido
nombre-5: "aaaaaaaaaa" (10 caracteres) → válido
nombre-6: "aaaaaaaaaaa" (11 caracteres) →
inválido
nombre-7: " " (solo espacios) → inválido
(vacío tras recortar)
nombre-8: (vacío) → inválido (obligatorio)
```

Ejemplo: Rango estrecho

ENTRADA

planta entero 0 2
cp texto 5 5

SALIDA ESPERADA

```
== planta (entero, 0..2)
planta-1: -1 → inválido
planta-2: 0 → válido
planta-3: 1 → válido
planta-4: 2 → válido
planta-5: 3 → inválido
planta-6: abc → inválido (no es un número)
planta-7: (vacío) → inválido (obligatorio)
== cp (texto, 5..5)
cp-1: "aaaa" (4 caracteres) → inválido
cp-2: "aaaaa" (5 caracteres) → válido
cp-3: "aaaaaa" (6 caracteres) → inválido
cp-4: " " (solo espacios) → inválido (vacío
tras recortar)
cp-5: (vacío) → inválido (obligatorio)
```

20. Un mini ejecutor de pruebas unitarias

Difícil · Pruebas de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/pruebas-interfaces

Implementa `precioFinal(importe, código, socio)` según esta especificación y un pequeño ejecutor de pruebas al estilo de JUnit. Especificación: un importe negativo lanza `IllegalArgumentException`; el código `VERANO10` descuenta un 10 % y `VIP25` un 25 % solo si el importe es de al menos 100 €; los socios tienen un 5 % adicional; el descuento total nunca pasa del 30 %; se suman 4,95 € de envío si el importe con descuento es menor de 50 € (salvo que el importe sea 0); y el resultado se redondea a céntimos. Cada línea de la entrada es un caso «nombre importe código(o -) socio(si/no) esperado», donde esperado es un número o la palabra excepción. El ejecutor muestra «PASA nombre» (dos espacios), «PASA nombre (lanza `IllegalArgumentException`)», «FALLA nombre: esperado X, obtenido Y», «FALLA nombre: se esperaba una excepción y devolvió Y» o «ERROR nombre: Excepcion: mensaje» si salta una excepción no esperada, y al final «Pruebas: N · Pasan: P · Fallan: F · Errores: E». Compara con una tolerancia de 0,005.

Código de partida (java)

```
import java.util.*;

public class Main {
    // TODO: implementa precioFinal según la especificación del enunciado
    static double precioFinal(double importe, String codigo, boolean socio) {
        return importe;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // Cada línea: nombre_prueba importe codigo(o -) socio(si/no) esperado
        // TODO: ejecuta cada caso, compara con el esperado (tolerancia 0.005) y muestra el informe
    }
}
```

Ejemplo: Especificación correcta

ENTRADA

```
sin_descuento 40 - no 44.95
envio_gratis_justo 50 - no 50.00
verano 100 VERANO10 no 90.00
vip_minimo 100 VIP25 no 75.00
vip_insuficiente 99.99 VIP25 no 99.99
socio 40 - si 42.95
tope_30 200 VIP25 si 140.00
importe_negativo -5 - no excepcion
gratis 0 VERANO10 no 0.00
```

SALIDA ESPERADA

```
PASA sin_descuento
PASA envio_gratis_justo
PASA verano
PASA vip_minimo
PASA vip_insuficiente
PASA socio
PASA tope_30
PASA importe_negativo (lanza
IllegalArgumentException)
PASA gratis
Pruebas: 9 · Pasan: 9 · Fallan: 0 · Errores: 0
```

Ejemplo: Casos mal escritos

ENTRADA

```
esperado_mal 40 - no 40.00
espera_excepcion 10 - no excepcion
negativo_sin_esperarlo -1 - no 0
```

SALIDA ESPERADA

```
FALLA esperado_mal: esperado 40.00, obtenido
44.95
FALLA espera_excepcion: se esperaba una excepción
y devolvió 14.95
ERROR negativo_sin_esperarlo:
IllegalArgumentException: importe negativo
Pruebas: 3 · Pasan: 0 · Fallan: 2 · Errores: 1
```

21. Pruebas de interfaz guionizadas

Muy difícil · Pruebas de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/pruebas-interfaces

Las herramientas de pruebas de extremo a extremo (Playwright, Selenium, TestFX) siguen un guion: escribir, pulsar y esperar un resultado. Completa el modelo de PantallaAcceso y el ejecutor. Reglas de la pantalla: el botón Entrar se habilita cuando el usuario no está en blanco, la clave tiene al menos 4 caracteres y hay menos de 3 intentos fallidos; pulsar con el botón deshabilitado no hace nada; con datos correctos (usuario sin distinguir mayúsculas y sin espacios alrededor) se pasa a la pantalla principal con el mensaje «Bienvenida, Usuario» (tal como se escribió, sin espacios); si no, se borra la clave, se deshabilita el botón, suma un intento y el mensaje es «Usuario o contraseña incorrectos (N intentos más)» o, al tercer fallo, «Cuenta bloqueada: demasiados intentos». El guion tiene líneas «escribe campo texto», «pulsar entrar» y «espera qué valor» (qué: mensaje, boton —habilitado/deshabilitado—, clave o pantalla —acceso/principal—); las líneas vacías y las que empiezan por # se ignoran. Por cada «espera» muestra «✓ paso N: qué = «valor»» o «X paso N: qué esperaba «X» pero es «Y»»; para una orden desconocida, «? paso N: orden desconocida «orden»». Termina con «PRUEBA SUPERADA» o «PRUEBA FALLIDA (F comprobaciones)».

Código de partida (java)

```
import java.util.*;

public class Main {
    // Modelo de una pantalla de acceso: campos usuario y clave, botón entrar y un mensaje
    static class PantallaAcceso {
        String usuario = "", clave = "", mensaje = "";
        boolean entrarHabilitado = false, dentro = false;
        int intentos = 0;
        static final Map<String, String> CUENTAS = Map.of("ana", "Segura123", "luis", "Clave2026");

        // TODO: escribir en un campo, pulsar el botón y actualizar el estado según las reglas
        void escribir(String campo, String texto) { }
        void pulsarEntrar() { }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // TODO: ejecuta el guion de prueba paso a paso y comprueba cada «espera»
    }
}
```

Ejemplo: Acceso correcto

ENTRADA

```
# camino feliz
espera boton deshabilitado
escribe usuario Ana
escribe clave Segura123
espera boton habilitado
pulsar entrar
espera pantalla principal
espera mensaje Bienvenida, Ana
```

SALIDA ESPERADA

```
✓ paso 1: boton = «deshabilitado»
✓ paso 4: boton = «habilitado»
✓ paso 6: pantalla = «principal»
✓ paso 7: mensaje = «Bienvenida, Ana»
PRUEBA SUPERADA
```

Ejemplo: Tres intentos fallidos

ENTRADA

```
escribe usuario luis
escribe clave mala
pulsa entrar
espera mensaje Usuario o contraseña incorrectos
(2 intentos más)
espera clave
escribe clave otra1
pulsa entrar
escribe clave otra2
pulsa entrar
espera mensaje Cuenta bloqueada: demasiados
intentos
escribe clave Clave2026
espera boton deshabilitado
pulsa entrar
espera pantalla acceso
```

SALIDA ESPERADA

```
✓ paso 4: mensaje = «Usuario o contraseña
incorrectos (2 intentos más)»
✓ paso 5: clave = «»
✓ paso 10: mensaje = «Cuenta bloqueada:
demasiados intentos»
✓ paso 12: boton = «deshabilitado»
✓ paso 14: pantalla = «acceso»
PRUEBA SUPERADA
```

Ejercicios de Desarrollo de Interfaces

22. Validar un NIF en un formulario

Fácil · Ejercicios de Desarrollo de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/ejercicios-desarrollo-interfaces

Antes de guardar un formulario hay que validar el NIF. Implementa `nifValido`: 8 dígitos y una letra (mayúscula o minúscula) cuya letra de control sea la correcta (TRWAGMYFPDXBNJZSQVHLCKE[número % 23]). Se ignoran espacios y guiones. El programa lee un NIF por línea y muestra «válido» o «no válido».

Código de partida (java)

```
import java.util.*;

public class Main {
    static final String LETRAS = "TRWAGMYFPDXBNJZSQVHLCKE";

    static boolean nifValido(String nif) {
        // TODO
        return false;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String l = sc.nextLine();
            if (!l.isBlank()) System.out.println(nifValido(l) ? "válido" : "no válido");
        }
    }
}
```

Ejemplo: NIF correctos e incorrectos

ENTRADA

```
12345678Z
12345678A
00000000T
```

SALIDA ESPERADA

```
válido
no válido
válido
```

Ejemplo: Formato

ENTRADA

```
1234567Z
12 345 678-z
ABCDEFGHZ
```

SALIDA ESPERADA

```
no válido
válido
no válido
```

23. Informe agrupado con subtotales

Medio · Ejercicios de Desarrollo de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/ejercicios-desarrollo-interfaces

Una pantalla de informes recibe líneas «categoría;importe». Muestra las categorías en orden alfabético con su subtotal y al final el total, con dos decimales y coma decimal:

Audio: 104,95

Informática: 179,00

TOTAL: 283,95

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // TODO: lee las líneas categoría;importe y muestra el informe
    }
}
```

Ejemplo: Tres líneas

ENTRADA

```
Audio;59.95
Informática;179
Audio;45
```

SALIDA ESPERADA

```
Audio: 104,95
Informática: 179,00
TOTAL: 283,95
```

24. Paginación de una tabla

Medio · Ejercicios de Desarrollo de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/ejercicios-desarrollo-interfaces

Una tabla muestra los resultados en páginas. La primera línea de la entrada tiene el tamaño de página y la página pedida (empezando en 1); después, un elemento por línea. Muestra los elementos de esa página y al final «Página X de Y». Si la página no existe, muestra solo «Página no válida». Si no hay elementos, «Sin resultados».

Código de partida (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int tam = sc.nextInt(), pagina = sc.nextInt();
        sc.nextLine();
        List<String> elementos = new ArrayList<>();
        while (sc.hasNextLine()) {
            String l = sc.nextLine();
            if (!l.isBlank()) elementos.add(l);
        }
        // TODO
    }
}
```

Ejemplo: Segunda página

ENTRADA

3 2
Ana
Luis
Marta
Pablo
Rosa

SALIDA ESPERADA

Pablo
Rosa
Página 2 de 2

Ejemplo: Página fuera de rango

ENTRADA

2 4
a
b
c

SALIDA ESPERADA

Página no válida

25. Comparar versiones semánticas

Medio · Ejercicios de Desarrollo de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/ejercicios-desarrollo-interfaces

El actualizador de la aplicación compara la versión instalada con la publicada. Cada línea tiene dos versiones MAYOR.MENOR.PARCHE separadas por un espacio. Muestra «<», «>» o «=» según la primera sea menor, mayor o igual que la segunda. Ojo: 2.10.0 es mayor que 2.9.5 (se comparan números, no textos).

Código de partida (java)

```
import java.util.*;

public class Main {

    static int comparar(String a, String b) {
        // TODO: negativo si a < b, 0 si son iguales, positivo si a > b
        return a.compareTo(b);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNext()) {
            int c = comparar(sc.next(), sc.next());
            System.out.println(c < 0 ? "<" : c > 0 ? ">" : "=");
        }
    }
}
```

Ejemplo: Casos básicos

ENTRADA

1.0.0 1.0.1
2.1.0 2.0.9
3.4.5 3.4.5

SALIDA ESPERADA

<
>
=

Ejemplo: Números de dos cifras

ENTRADA

2.10.0 2.9.5
1.2.10 1.2.9

SALIDA ESPERADA

>
>

Ejercicios largos

26. Componente web accesible: valoración con estrellas

Muy difícil · JavaScript · 100 minutos · apuntesdam.com/ejercicios/javascript/componente-web-de-valoracion

La plataforma de cursos de un centro quiere que los alumnos valoren cada curso con estrellas, y el mismo control aparece en varias páginas y en dos aplicaciones distintas. En lugar de copiar el HTML y el JavaScript en cada sitio, se crea un componente: una etiqueta propia, `<valoracion-estrellas>`, que cualquier página puede usar como si fuera un elemento más de HTML.

Los Web Components son el estándar del navegador para esto: `customElements.define` registra la etiqueta, una clase que extiende `HTMLElement` define su comportamiento y el Shadow DOM encapsula su HTML y su CSS para que la página no los rompa ni ellos rompan la página.

Un control de estrellas es, en realidad, un grupo de opciones excluyentes: para un lector de pantalla tiene que comportarse como un grupo de botones de radio, con su nombre, el estado de cada opción y el manejo con las flechas del teclado.

Requisitos

- Cada apartado reescribe el script completo del componente: empieza con la solución del anterior. La página tiene tres componentes: `#v1` (máximo 5, valor 3, etiqueta «Valora el curso»), `#v2` (sin atributos) y `#v3` (máximo 10, valor 7).
- El componente usa un Shadow DOM abierto (`mode: "open"`) con un contenedor `role="radiogroup"` y un `button` por estrella, cuyo texto es `★`.
- Cada apartado se corrige por separado con pruebas que se ejecutan dentro de la página (algunas a varios anchos de pantalla). Debajo del editor tienes el resultado en directo.

Página de partida (html)

```
<p>Valora el curso:</p>
<valoracion-estrellas id="v1" max="5" valor="3" etiqueta="Valora el curso"></valoracion-estrellas>
<p>Sin valorar todavía:</p>
<valoracion-estrellas id="v2"></valoracion-estrellas>
<p>Sobre 10:</p>
<valoracion-estrellas id="v3" max="10" valor="7"></valoracion-estrellas>
```

Apartados

- Define la clase `ValoracionEstrellas` y regístrala como `valoracion-estrellas`. Al conectarse a la página dibuja en su Shadow DOM un `div` con `role="radiogroup"` y el `aria-label` del atributo `etiqueta` (o «Valoración» si no lo tiene), y dentro tantos `button` como diga el atributo `max` (5 si no lo tiene), cada uno con el texto `★` y el `aria-label` «1 estrella», «2 estrellas»...
- Añade el valor: el atributo `valor` (0 si no lo tiene, nunca más que `max`) marca como llenas (clase `llena`) las estrellas hasta él. Cada botón tiene `role="radio"` y `aria-checked` (`"true"` solo en la estrella del valor). Al pulsar una estrella, el atributo `valor` cambia, el componente se redibuja y lanza el evento `cambio` con `detail: { valor }`, que debe llegar a la página (`bubbles` y `composed`).
- Hazlo usable con el teclado como un grupo de radios: solo la estrella elegida (o la primera, si el valor es 0) tiene `tabindex="0"` y las demás `-1`, para que el tabulador entre y salga del grupo de una vez; con las flechas derecha y arriba sube una estrella y con izquierda y abajo baja una (entre 1 y `max`), igual que al pulsar (atributo, redibujo y evento), y el foco pasa a la estrella elegida.
- Sincroniza el componente con el exterior: si otro código cambia los atributos `valor`, `max` o `etiqueta`, el componente se redibuja (`observedAttributes` y `attributeChangedCallback`). Y añade la propiedad `valor`: leerla devuelve el número y asignarla (`el.valor = 4`) cambia el atributo.

27. Informe de ventas imprimible: agrupación, formatos y CSS de impresión

Difícil · JavaScript · 75 minutos · apuntesdam.com/ejercicios/javascript/informe-de-ventas-imprimible

El departamento comercial necesita cada trimestre un informe de ventas que se pueda ver en pantalla y también imprimir o guardar en PDF desde el navegador. Herramientas como JasperReports generan informes desde Java; en una aplicación web, lo habitual es generarlos con HTML y CSS, y el diálogo de impresión del navegador se encarga del PDF.

Los datos llegan como JSON (aquí, dentro de la propia página en un `<script type="application/json">`). El informe agrupa las ventas por comercial, con un subtotal por grupo y un total general: es la estructura clásica de un informe con grupos y sumas.

Lo que se ve bien en pantalla no tiene por qué imprimirse bien: los botones sobran en el papel, las tablas largas se parten entre páginas y los fondos gastan tinta. Para eso existe la hoja de estilos de impresión.

Requisitos

- Los datos se leen del elemento `#datos` con `JSON.parse`. La tabla se añade dentro de `#informe`.
- Cada apartado se corrige por separado con pruebas que se ejecutan dentro de la página (algunas a varios anchos de pantalla). Debajo del editor tienes el resultado en directo.

Página de partida (html)

```
<h1>Ventas del tercer trimestre</h1>
<button class="no-imprimir" type="button" onclick="print()">Imprimir</button>
<div id="informe"></div>
<script id="datos" type="application/json">[{"comercial": "Luis Mora", "cliente": "Ferretería López",
"fecha": "2026-07-03", "importe": 1250.5}, {"comercial": "Ana Gil", "cliente": "Papelería Sol",
"fecha": "2026-07-10", "importe": 320.0}, {"comercial": "Ana Gil", "cliente": "Talleres Ruiz", "fecha":
"2026-07-02", "importe": 2875.9}, {"comercial": "Eva Ruiz", "cliente": "Hotel Mar", "fecha": "2026-08-
14", "importe": 4120.0}, {"comercial": "Luis Mora", "cliente": "Bar Central", "fecha": "2026-08-01",
"importe": 89.95}, {"comercial": "Ana Gil", "cliente": "Clínica Norte", "fecha": "2026-09-22",
"importe": 1500.0}, {"comercial": "Eva Ruiz", "cliente": "Colegio Sur", "fecha": "2026-09-05",
"importe": 760.4}, {"comercial": "Luis Mora", "cliente": "Ferretería López", "fecha": "2026-09-18",
"importe": 640.0}]</script>
```

Hoja de estilos de partida (css)

```
table { border-collapse: collapse; width: 100%; font-size: 0.9rem; }
th, td { border-bottom: 1px solid #ccc; padding: 4px 8px; text-align: left; }
.importe { text-align: right; }
.subtotal td { font-weight: bold; background: #f2f2f2; }
.total td { font-weight: bold; border-top: 2px solid #000; }
```

Apartados

- Genera la tabla: un `thead` con las columnas Comercial, Cliente, Fecha e Importe; en el `tbody`, las ventas agrupadas por comercial (los comerciales por orden alfabético y sus ventas por fecha), y detrás de cada grupo una fila con la clase `subtotal`: «Total Nombre», dos celdas vacías y la suma. Al final, un `tfoot` con una fila `total`: «Total general», dos celdas vacías y la suma de todo. En este apartado, las fechas tal como vienen (`2026-07-03`) y los importes con dos decimales y punto (`1250.50`).
- Formato para personas: importes con `Intl.NumberFormat` en euros a la española (`1.250,50 €`) y fechas con `Intl.DateTimeFormat` como `03/07/2026`. Añade un `caption` con el periodo y el número de ventas: `Del 02/07/2026 al 22/09/2026 · 8 ventas`, pon `scope="col"` en los encabezados y la clase `importe` en las celdas de importe.
- La hoja de impresión: dentro de `@media print`, oculta `.no-imprimir`, pon el texto a 10pt y negro, haz que el `thead` se repita en cada página (`display: table-header-group`), evita que una fila se parta entre dos páginas (`break-inside: avoid` en `tr`) y quita el fondo de los subtotales. Fuera, una regla `@page` con tamaño A4 y márgenes de 2cm. Conserva los estilos de pantalla.

28. Página de un evento accesible: HTML semántico y ARIA

Medio · HTML y CSS · 60 minutos · apuntesdam.com/ejercicios/css/pagina-de-un-evento-accesible

El instituto organiza unas jornadas de software libre y alguien hizo la página a toda prisa con `div` y `span`. Se ve bien, pero una persona que navega con teclado o con un lector de pantalla no puede usarla: el menú no se puede enfocar, no hay encabezados para saltar de sección en sección, las imágenes no tienen texto alternativo y la tabla del programa es una cuadrícula sin sentido.

Las pautas WCAG 2.2 (nivel AA, el que exige la ley a las webs públicas) piden que todo funcione con teclado, que la estructura se pueda leer sin verla y que los controles anuncien su nombre y su estado. Casi todo se consigue con el elemento HTML adecuado; ARIA solo hace falta para lo que el HTML no tiene, como el estado abierto o cerrado de un menú.

Requisitos

- Cada apartado se corrige por separado con pruebas que se ejecutan dentro de la página (algunas a varios anchos de pantalla). Debajo del editor tienes el resultado en directo.
- Conserva las clases y los textos visibles de la página: las pruebas buscan los elementos por ellos.

Página de partida (html)

```
<div class="cabecera">
  
  <div class="menu">
    <span onclick="location='#programa'">Programa</span>
    <span onclick="location='#entradas'">Entradas</span>
  </div>
</div>
<div class="contenido">
  <div class="titulo">Jornadas de Software Libre 2026</div>
  
  
  <div class="subtitulo">Programa</div>
  <table>
    <tr><td>Hora</td><td>Charla</td><td>Sala</td></tr>
    <tr><td>10:00</td><td>Linux en el aula</td><td>A</td></tr>
    <tr><td>11:30</td><td>Accesibilidad web</td><td>B</td></tr>
  </table>
  <a href="programa.pdf">Pincha aquí</a>
</div>
```

Apartados

- Rehaz el HTML de la página: un enlace «Saltar al contenido» (clase `salto`) como primer elemento, que lleve al `main` (con id `principal`); la cabecera en un `header`, el menú en un `nav` con `aria-label="Principal"` y sus opciones como enlaces (`#programa` y `#entradas`); el contenido en `main`; el título como `h1` y «Programa» como `h2` con id `programa`; el logo con el `alt` «Jornadas de Software Libre», el cartel con un `alt` que describa su información (fechas y lugar) y el separador, que es decorativo, con `alt=""`; la tabla con un `caption` y su primera fila con `th scope="col"`; y el enlace al PDF con un texto que diga qué es y que es un PDF (sin «aquí»).
- En el móvil, el menú se despliega con un botón. El HTML ya es correcto (un `button` con `aria-expanded` y `aria-controls`, y el `nav` con `hidden`); escribe el JavaScript: al pulsar el botón, el menú se muestra u oculta (atributo `hidden`) y `aria-expanded` pasa a `"true"` o `"false"`; con el menú abierto, la tecla Escape lo cierra y devuelve el foco al botón.

Soluciones

1. La validación de un formulario de registro

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

Solución (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, String> campos = new LinkedHashMap<>();
        while (sc.hasNextLine()) {
            String linea = sc.nextLine();
            int i = linea.indexOf('=');
            if (i > 0) campos.put(linea.substring(0, i).trim(), linea.substring(i + 1).trim());
        }
        String nombre = campos.getDefault("nombre", ""), email = campos.getDefault("email", ""),
            edad = campos.getDefault("edad", ""), clave = campos.getDefault("clave", ""),
            repetir = campos.getDefault("repetir", "");
        List<String> errores = new ArrayList<>();

        if (nombre.isEmpty()) errores.add("nombre: es obligatorio");
        else if (nombre.length() < 2) errores.add("nombre: debe tener al menos 2 caracteres");

        if (!email.matches("[^@\\s]+@[^@\\s]+\\.([A-Za-z]{2,})")) errores.add("email: formato no válido");

        try {
            int años = Integer.parseInt(edad);
            if (años < 18 || años > 120) errores.add("edad: debe estar entre 18 y 120");
        } catch (NumberFormatException e) {
            errores.add("edad: debe ser un número entero");
        }

        List<String> falta = new ArrayList<>();
        if (clave.length() < 8) falta.add("8 caracteres");
        if (clave.chars().noneMatch(Character::isUpperCase)) falta.add("una mayúscula");
        if (clave.chars().noneMatch(Character::isDigit)) falta.add("un número");
        if (!falta.isEmpty()) errores.add("clave: necesita " + String.join(", ", falta));

        if (!repetir.equals(clave)) errores.add("repetir: las contraseñas no coinciden");

        errores.forEach(System.out::println);
        if (errores.isEmpty()) System.out.println("Formulario válido: botón Registrar activado");
        else System.out.println(errores.size() + (errores.size() == 1 ? " error" : " errores") + ": botón Registrar desactivado");
    }
}
```

Cada campo se valida por separado y sus errores se acumulan en una lista, en el orden del formulario.

`getOrDefault(campo, "")` hace que un campo que no llega se trate igual que uno vacío.

El email se comprueba con una expresión regular: algo sin @ ni espacios, una @, algo más y un punto seguido de al menos dos letras. La edad se intenta convertir con `parseInt` y, si no es un número, la excepción se transforma en el mensaje de error.

Para la clave se reúne en otra lista solo lo que falta, y `chars().noneMatch(...)` comprueba si no hay ningún carácter que cumpla la condición. Con la lista de errores, el botón se activa si está vacía: así funcionan los formularios reales.

2. Repartir el ancho como un GridPane

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

Solución (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int ancho = Integer.parseInt(sc.nextLine().split(" ")[1]);
        int hueco = Integer.parseInt(sc.nextLine().split(" ")[1]);
        List<String[]> columnas = new ArrayList<>(); // {nombre, tipo, valor}
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (!l.isEmpty()) columnas.add(l.split(" "));
        }

        double disponible = ancho - hueco * (columnas.size() - 1);
        double usado = 0;
        int crecen = 0;
        for (String[] c : columnas) {
            switch (c[1]) {
                case "fija" -> usado += Double.parseDouble(c[2]);
                case "porcentaje" -> usado += disponible * Double.parseDouble(c[2]) / 100;
                default -> crecen++;
            }
        }
        double resto = disponible - usado;
        if (resto < crecen) { // cada columna que crece necesita al menos 1 px
            System.out.printf(Locale.US, "No cabe: faltan %.1f px%n", crecen - resto);
            return;
        }

        double x = 0;
        for (String[] c : columnas) {
            double a = switch (c[1]) {
                case "fija" -> Double.parseDouble(c[2]);
                case "porcentaje" -> disponible * Double.parseDouble(c[2]) / 100;
                default -> resto / crecen;
            };
            System.out.printf(Locale.US, "%s: x=%.1f ancho=%.1f%n", c[0], x, a);
            x += a + hueco;
        }
        if (crecen == 0 && resto > 0) System.out.printf(Locale.US, "Sobran %.1f px a la derecha%n",
            resto);
    }
}
```

Es el mismo algoritmo que siguen los contenedores de rejilla: primero se descuentan los huecos entre columnas, después se reserva lo fijo y lo porcentual, y lo que sobra se reparte a partes iguales entre las columnas que crecen.

Hacen falta dos recorridos: en el primero se suma lo reservado y se cuentan las columnas que crecen (no se puede saber cuánto les toca hasta conocer el total); en el segundo se calcula el ancho de cada una y se va acumulando la x, sumando el hueco tras cada columna.

La condición `resto < crecen` cubre los dos casos de error a la vez: si no crece ninguna, salta cuando lo reservado supera el disponible; si crecen, cuando no queda al menos 1 px para cada una.

3. Captura y burbuja de un evento

Muy difícil · Generación de Interfaces Gráficas · [apuntesdam.com/subject/desarrollo-interfaces/topic/generacion-](https://apuntesdam.com/subject/desarrollo-interfaces/topic/generacion-interfaces)

interfaces

Solución (java)

```
import java.util.*;

public class Main {
    // Ejecuta, en orden de registro, los filtros o manejadores de un nodo; devuelve si alguno consumió
    // el evento
    static boolean ejecutar(List<String[]> manejadores, String tipo, String nodo) {
        boolean consumido = false;
        for (String[] m : manejadores) {
            if (!m[0].equals(tipo) || !m[1].equals(nodo)) continue;
            boolean consume = m.length > 2 && m[2].equals("consume");
            System.out.println(tipo + " de " + nodo + (consume ? " (consume el evento)" : ""));
            consumido |= consume; // los demás del mismo nodo se ejecutan igualmente
        }
        return consumido;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, String> padre = new HashMap<>();
        List<String[]> manejadores = new ArrayList<>(); // {tipo (filtro|manejador), nodo, consume?}
        String objetivo = null;
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");
            switch (p[0]) {
                case "nodo" -> padre.put(p[1], p[2]); // nodo HIJO PADRE
                case "filtro", "manejador" -> manejadores.add(p);
                case "clic" -> objetivo = p[1];
                default -> { }
            }
        }

        // Ruta de despacho: del objetivo a la raíz, siguiendo los padres, y al revés
        List<String> ruta = new ArrayList<>();
        for (String n = objetivo; n != null; n = padre.get(n)) ruta.add(0, n);
        System.out.println("Ruta: " + String.join(" > ", ruta));

        boolean consumido = false;
        for (int i = 0; i < ruta.size() && !consumido; i++) { // captura: raíz → objetivo
            consumido = ejecutar(manejadores, "filtro", ruta.get(i));
        }
        for (int i = ruta.size() - 1; i >= 0 && !consumido; i--) { // burbuja: objetivo → raíz
            consumido = ejecutar(manejadores, "manejador", ruta.get(i));
        }
        System.out.println(consumido ? "Evento consumido" : "El evento llegó a la raíz sin consumirse");
    }
}
```

Primero se construye la ruta subiendo desde el objetivo por el mapa de padres hasta la raíz (que no tiene padre) e insertando cada nodo al principio, para que quede de la raíz al objetivo.

Las dos fases son dos bucles sobre esa ruta, uno en cada sentido: la captura ejecuta los filtros de la raíz al objetivo y la burbuja los manejadores del objetivo a la raíz. Ambos se detienen en cuanto un nodo consume el evento.

El método `ejecutar` recorre todos los manejadores registrados de un nodo antes de devolver si alguno lo consumió (`consumido |= consume`), que es justo la regla de JavaFX: el consumo afecta a los nodos siguientes, no a los del mismo nodo.

4. Un intérprete de órdenes de voz

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

Solución (java)

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

public class Main {
    static final Pattern ALARMA = Pattern.compile("\\b(pon|poner|pone|programa) (una )?alarma a las (\\d{1,2})(:(\\d{2}))?( y (media|cuarto))?");
    static final Pattern LLAMADA = Pattern.compile("\\b(llama|llamar) a (.+)");
    static final Pattern VOLUMEN = Pattern.compile("\\b(sube|baja) (el )?volumen( (\\d+))?");
    static final Pattern ABRIR = Pattern.compile("\\b(abre|abrir|inicia) (la app |la aplicacion )? (\\S+)");

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

    static String capitalizar(String nombre) {
        StringBuilder sb = new StringBuilder();
        for (String p : nombre.split(" ")) {
            if (sb.length() > 0) sb.append(' ');
            sb.append(Character.toUpperCase(p.charAt(0))).append(p.substring(1));
        }
        return sb.toString();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String frase = sc.nextLine();
            if (frase.isBlank()) continue;
            String t = normalizar(frase);
            Matcher m;
            if ((m = ALARMA.matcher(t)).find()) {
                int h = Integer.parseInt(m.group(3));
                int min = m.group(5) != null ? Integer.parseInt(m.group(5)) : "media".equals(m.group(7)) ? 30 : "cuarto".equals(m.group(7)) ? 15 : 0;
                System.out.println(h > 23 || min > 59 ? "ERROR: hora no válida" : String.format("ALARMA %02d:%02d", h, min));
            } else if ((m = LLAMADA.matcher(t)).find()) {
                System.out.println("LLAMAR " + capitalizar(m.group(2)));
            } else if ((m = VOLUMEN.matcher(t)).find()) {
                String n = m.group(4) != null ? m.group(4) : "10";
                System.out.println("VOLUMEN " + (m.group(1).equals("sube") ? "+" : "-") + n);
            } else if ((m = ABRIR.matcher(t)).find()) {
                System.out.println("ABRIR " + m.group(3));
            } else {
                System.out.println("NO ENTENDIDO: " + frase);
            }
        }
    }
}
```

Cada intención es una expresión regular con grupos para los datos que interesan: la hora, los minutos o el «y media» de la alarma, el nombre de la llamada o la cantidad de volumen. Se usa `find()` en lugar de `matches()` para que la orden se reconozca aunque haya palabras antes («Oye, programa...»).

Las partes opcionales se escriben con `( ... )?`, y si no aparecen su grupo vale `null`: así se decide si hay minutos, si se dijo «y media» o si se usa el volumen por defecto. Gracias a `normalizar`, las expresiones no tienen que preocuparse de mayúsculas, tildes ni signos.

Las expresiones se prueban en el orden del enunciado y se guardan como constantes `static final` para compilarlas una sola vez, no en cada frase.

5. Clasificar gestos táctiles

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

Solución (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<List<double[]>> gestos = new ArrayList<>();
        List<double[]> actual = new ArrayList<>();
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (l.isEmpty()) { if (!actual.isEmpty()) { gestos.add(actual); actual = new ArrayList<>(); }
                continue; }
            String[] p = l.split(" "); // tiempo_ms x y
            actual.add(new double[]{Double.parseDouble(p[0]), Double.parseDouble(p[1]),
                Double.parseDouble(p[2])});
        }
        if (!actual.isEmpty()) gestos.add(actual);

        for (int i = 0; i < gestos.size(); i++) {
            List<double[]> g = gestos.get(i);
            double[] ini = g.get(0), fin = g.get(g.size() - 1);
            double dx = fin[1] - ini[1], dy = fin[2] - ini[2];
            double distancia = Math.hypot(dx, dy);
            double duracion = fin[0] - ini[0];
            double velocidad = distancia / duracion;
            String tipo;
            if (distancia < 10) {
                tipo = duracion < 300 ? "TOQUE" : duracion >= 500 ? "PULSACIÓN LARGA" : "INDECISO";
            } else if (distancia >= 50 && velocidad >= 0.3) {
                if (Math.abs(dx) >= Math.abs(dy)) tipo = dx > 0 ? "DESGLIZAR DERECHA" : "DESGLIZAR
                IZQUIERDA";
                else tipo = dy > 0 ? "DESGLIZAR ABAJO" : "DESGLIZAR ARRIBA"; // la y crece hacia abajo
            } else {
                tipo = "ARRASTRE";
            }
            System.out.printf(Locale.US, "Gesto %d: %s (%.0f px en %.0f ms, %.2f px/ms)%n", i + 1, tipo,
                distancia, duracion, velocidad);
        }
    }
}
```

Solo hacen falta el primer y el último punto: la diferencia de x e y da el desplazamiento, `Math.hypot(dx, dy)` la distancia en línea recta y la diferencia de tiempos la duración.

La clasificación es un árbol de decisiones: si el dedo apenas se movió, decide el tiempo (toque, indeciso o pulsación larga); si se movió mucho y deprisa es un deslizamiento, y el eje con mayor desplazamiento da la dirección; en otro caso, un arrastre. Como en pantalla la y crece hacia abajo, una dy negativa es un deslizamiento hacia arriba.

6. ¿Qué gesto hace la mano?

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

Solución (java)

```
import java.util.*;

public class Main {
    static double dist(double[] a, double[] b) {
        return Math.hypot(a[0] - b[0], a[1] - b[1]);
    }

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

        for (int i = 0; i < manos.size(); i++) {
            Map<String, double[]> mano = manos.get(i);
            double[] muneca = mano.get("muneca");
            List<String> extendidos = new ArrayList<>();
            for (String dedo : DEDOS) {
                double punta = dist(muneca, mano.get(dedo + "_punta"));
                double medio = dist(muneca, mano.get(dedo + "_medio"));
                if (punta > medio * (dedo.equals("pulgar") ? 1.1 : 1.15)) extendidos.add(dedo);
            }
            boolean pinza = dist(mano.get("pulgar_punta"), mano.get("indice_punta"))
                < 0.25 * dist(muneca, mano.get("indice_medio"));
            String gesto = pinza ? "PINZA"
                : extendidos.isEmpty() ? "PUÑO"
                : extendidos.size() == 5 ? "MANO ABIERTA"
                : extendidos.equals(List.of("pulgar")) ? "PULGAR"
                : extendidos.equals(List.of("indice", "corazon")) ? "VICTORIA"
                : extendidos.equals(List.of("indice")) ? "SEÑALAR"
                : "DESCONOCIDO";
            System.out.println("Mano " + (i + 1) + ": " + gesto + " (extendidos: "
                + (extendidos.isEmpty() ? "ninguno" : String.join(", ", extendidos)) + ")");
        }
    }
}
```

Comparar distancias a la muñeca en lugar de coordenadas hace que el resultado no dependa de cómo esté girada la mano ni de lo cerca que esté de la cámara: un dedo estirado tiene la punta bastante más lejos de la muñeca que su articulación media, y uno doblado no.

La pinza también usa una distancia relativa (al tamaño de la mano, medido de la muñeca al índice), porque en coordenadas normalizadas una mano lejana es más pequeña.

Como los dedos extendidos se añaden en el orden de `DEDOS`, cada gesto se reconoce comparando la lista con otra fija (`List.of("indice", "corazon")`); la cadena de ternarios respeta el orden de prioridad del enunciado.

7. Un componente de valoración por estrellas

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

Solución (java)

```
import java.util.*;

public class Main {
    static class Valoracion {
        private double valor = 0;

        private static String texto(double v) {
            return v == Math.floor(v) ? String.valueOf((long) v) : String.valueOf(v);
        }

        void setValor(double v) {
            double nuevo = Math.round(Math.max(0, Math.min(5, v)) * 2) / 2.0;
            if (nuevo != valor) {
                System.out.println("cambio: " + texto(valor) + " → " + texto(nuevo));
                valor = nuevo;
            }
        }

        void clic(int estrella) {
            setValor(estrella == valor ? 0 : estrella); // pulsar la misma estrella borra la valoración
        }

        String pintar() {
            StringBuilder sb = new StringBuilder();
            for (int i = 1; i <= 5; i++) {
                sb.append(valor >= i ? "★" : valor >= i - 0.5 ? "½" : "☆");
            }
            return sb + " (" + texto(valor) + ")";
        }
    }

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

Todo cambio de valor pasa por `setValor`, que es el único sitio que valida (recorta a [0, 5] y redondea al medio punto) y avisa del cambio. `clic` no toca el atributo directamente: decide el nuevo valor y llama a `setValor`, así la regla nunca se salta. Es el mismo principio que siguen los componentes reales con sus propiedades.

Para pintar, la estrella *i* está llena si el valor llega a *i*, a medias si llega a *i* - 0,5 y vacía en otro caso. El método `texto` quita el «.0» de los valores enteros.

8. Propiedades observables y bindings

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

componentes

Solución (java)

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

public class Main {
    static class Propiedad {
        final String nombre;
        private int valor = 0;
        private final List<BiConsumer<Integer, Integer>> oyentes = new ArrayList<>();
        private final List<Propiedad> dependientes = new ArrayList<>(); // las enlazadas a esta
        private Propiedad fuente = null; // a la que está enlazada esta

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

        int get() { return valor; }

        void set(int v) {
            if (fuente != null) {
                System.out.println("ERROR: " + nombre + " está enlazada a " + fuente.nombre + " y no se puede cambiar");
                return;
            }
            cambiar(v);
        }

        private void cambiar(int v) {
            if (v == valor) return;
            int viejo = valor;
            valor = v;
            for (BiConsumer<Integer, Integer> o : oyentes) o.accept(viejo, v);
            for (Propiedad d : dependientes) d.cambiar(v); // se propaga a las enlazadas
        }

        void escuchar() {
            int n = oyentes.size() + 1;
            oyentes.add((viejo, nuevo) -> System.out.println(" " + nombre + " oyente " + n + ": " + viejo + " -> " + nuevo));
        }

        void bind(Propiedad otra) {
            for (Propiedad p = otra; p != null; p = p.fuente) {
                if (p == this) {
                    System.out.println("ERROR: enlace circular");
                    return;
                }
            }
            unbind();
            fuente = otra;
            otra.dependientes.add(this);
            cambiar(otra.valor);
        }

        void unbind() {
            if (fuente != null) {
                fuente.dependientes.remove(this);
                fuente = null;
            }
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, Propiedad> props = new LinkedHashMap<>();
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");
            if (p.length < 2) continue;
            Propiedad a = props.computeIfAbsent(p[1], Propiedad::new);
            switch (p[0]) {
```

```

        case "set" -> a.set(Integer.parseInt(p[2]));
        case "escuchar" -> a.escuchar();
        case "bind" -> a.bind(props.computeIfAbsent(p[2], Propiedad::new));
        case "unbind" -> a.unbind();
        case "get" -> System.out.println(a.nombre + " = " + a.get());
    }
}
}

```

Es el patrón Observador: cada propiedad guarda una lista de oyentes (`BiConsumer` que reciben el valor viejo y el nuevo) y los llama cuando su valor cambia de verdad. Los enlaces son el mismo mecanismo: una propiedad enlazada es una dependiente a la que se le propaga cada cambio.

`set` (público) comprueba que la propiedad no esté enlazada; `cambiar` (privado) es el que realmente cambia el valor, avisa y propaga, y lo usan tanto `set` como la propagación. Si el valor no cambia, `cambiar` termina enseguida, lo que también evita avisos inútiles.

Para detectar un ciclo, `bind` sube por la cadena de fuentes de la otra propiedad: si en algún momento aparece la propia propiedad, enlazarlas crearía un bucle infinito de propagaciones.

9. Un campo de texto con máscara

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

```

import java.util.*;

public class Main {
    static boolean esHueco(char m) {
        return m == '#' || m == 'A' || m == '?';
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String mascara = sc.nextLine().substring("mascara ".length()); // # dígito, A letra, ?
        cualquiera; lo demás es literal
        List<String> teclas = new ArrayList<>(Arrays.asList(sc.nextLine().split(" ")));

        StringBuilder texto = new StringBuilder();
        List<String> rechazadas = new ArrayList<>();
        for (String tecla : teclas) {
            if (tecla.equals("BORRAR")) {
                if (texto.length() > 0) texto.setLength(texto.length() - 1);
                // también se quitan los literales que queden al final
                while (texto.length() > 0 && !esHueco(mascara.charAt(texto.length() - 1)))
                    texto.setLength(texto.length() - 1);
                continue;
            }
            char c = tecla.equals("ESPACIO") ? ' ' : tecla.charAt(0);
            // Siguiente hueco, saltando los literales
            int pos = texto.length();
            while (pos < mascara.length() && !esHueco(mascara.charAt(pos))) pos++;
            if (pos == mascara.length()) {
                rechazadas.add(tecla + " (campo completo)");
                continue;
            }
            char hueco = mascara.charAt(pos);
            if (hueco == '#' && !Character.isDigit(c)) {
                rechazadas.add(tecla + " (se esperaba un dígito)");
                continue;
            }
            if (hueco == 'A' && !Character.isLetter(c)) {
                rechazadas.add(tecla + " (se esperaba una letra)");
                continue;
            }
            texto.append(mascara, texto.length(), pos); // los literales de antes del hueco
            texto.append(hueco == 'A' ? Character.toUpperCase(c) : c);
        }

        StringBuilder pantalla = new StringBuilder(texto);
        boolean completo = true;
        for (int i = texto.length(); i < mascara.length(); i++) {
            char m = mascara.charAt(i);
            pantalla.append(esHueco(m) ? '_' : m);
            if (esHueco(m)) completo = false;
        }
        System.out.println("Pantalla: " + pantalla);
        System.out.println("Completo: " + (completo ? "sí" : "no"));
        rechazadas.forEach(r -> System.out.println("Rechazada: " + r));
    }
}

```

La posición en la máscara es simplemente la longitud de lo escrito, porque cada carácter escrito (incluidos los literales) ocupa una posición. Con cada tecla se busca el siguiente hueco saltando los literales y se valida la tecla contra él; solo si se acepta se escriben los literales intermedios y la tecla.

BORRAR quita el último carácter y luego sigue quitando mientras lo último sea un literal: así, al borrar el primer dígito de «(9», el paréntesis desaparece con él.

La pantalla es lo escrito más el resto de la máscara, cambiando los huecos por `_`. El campo está completo si en esa parte pendiente no queda ningún hueco.

10. Contraste de colores según WCAG

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

Solución (java)

```
import java.util.*;

public class Main {
    static double canal(int c) {
        double v = c / 255.0;
        return v <= 0.03928 ? v / 12.92 : Math.pow((v + 0.055) / 1.055, 2.4);
    }

    static double luminancia(String color) {
        String h = color.substring(1);
        if (h.length() == 3) { // #rgb → #rrggbb
            h = "" + h.charAt(0) + h.charAt(0) + h.charAt(1) + h.charAt(1) + h.charAt(2) + h.charAt(2);
        }
        int r = Integer.parseInt(h.substring(0, 2), 16);
        int g = Integer.parseInt(h.substring(2, 4), 16);
        int b = Integer.parseInt(h.substring(4, 6), 16);
        return 0.2126 * canal(r) + 0.7152 * canal(g) + 0.0722 * canal(b);
    }

    static double contraste(String a, String b) {
        double la = luminancia(a), lb = luminancia(b);
        return (Math.max(la, lb) + 0.05) / (Math.min(la, lb) + 0.05);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");
            if (p.length < 2) continue;
            double r = contraste(p[0], p[1]);
            String normal = r >= 7 ? "AAA" : r >= 4.5 ? "AA" : "no cumple";
            String grande = r >= 4.5 ? "AAA" : r >= 3 ? "AA" : "no cumple";
            System.out.printf(Locale.US, "%s sobre %s: %.2f:1 · texto normal %s · texto grande %s\n",
                p[0], p[1], r, normal, grande);
        }
    }
}
```

Cada color se descompone en sus tres canales con `Integer.parseInt(..., 16)` (el formato corto `#rgb` se amplía antes duplicando cada cifra). Cada canal se linealiza con la fórmula de WCAG, porque el ojo no percibe el brillo de forma lineal, y se combinan con pesos distintos: el verde pesa mucho más que el azul.

El contraste divide la luminancia del color más claro entre la del más oscuro (con 0,05 sumado a ambas), así que no importa cuál sea el texto y cuál el fondo. Va de 1:1 (mismo color) a 21:1 (negro sobre blanco).

11. El orden de tabulación de un formulario

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

Solución (java)

```
import java.util.*;

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

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<Control> controles = new ArrayList<>();
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" "); // id x y tabIndex habilitado(si|no)
            if (p.length < 5) continue;
            controles.add(new Control(p[0], Integer.parseInt(p[1]), Integer.parseInt(p[2]),
                Integer.parseInt(p[3]), p[4].equals("si")));
        }

        Comparator<Control> lectura = Comparator.comparingInt(Control::y).thenComparingInt(Control::x);
        List<Control> positivos = new ArrayList<>(), naturales = new ArrayList<>();
        for (Control c : controles) {
            if (!c.habilitado() || c.tabIndex() < 0) continue;
            (c.tabIndex() > 0 ? positivos : naturales).add(c);
        }
        positivos.sort(Comparator.comparingInt(Control::tabIndex).thenComparing(lectura));
        naturales.sort(lectura);

        List<Control> orden = new ArrayList<>(positivos);
        orden.addAll(naturales);
        for (int i = 0; i < orden.size(); i++) System.out.println((i + 1) + ". " + orden.get(i).id());
        if (!positivos.isEmpty()) {
            System.out.println("Aviso: " + positivos.size() + " control(es) con tabIndex positivo alteran
                el orden natural");
        }
        for (Control c : controles) {
            if (!c.habilitado()) System.out.println("Fuera del orden: " + c.id() + " (deshabilitado)");
            else if (c.tabIndex() < 0) System.out.println("Fuera del orden: " + c.id() + " (tabIndex " +
                c.tabIndex() + ")");
        }
    }
}
```

Se separan los controles que reciben el foco en dos listas, los de tabIndex positivo y los de tabIndex 0, y cada una se ordena con su `Comparator`; el orden final es la primera lista seguida de la segunda.

El orden de lectura (por y y después por x) se define una vez como `Comparator` y se reutiliza: ordena los de tabIndex 0 y desempata los positivos. Un `record` es ideal para un dato así: el constructor y los accesos `c.y()` vienen hechos.

El aviso de los tabIndex positivos tiene sentido: obligan a quien usa el teclado a saltar por la pantalla en un orden distinto del visual, y es una de las faltas de accesibilidad más comunes.

12. ¿Se entiende este texto?

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

Solución (java)

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

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

    static boolean vocal(char c) { return VOCALES.indexOf(c) >= 0; }
    static boolean fuerte(char c) { return FUERTES.indexOf(c) >= 0; }

    static int silabas(String palabra) {
        String p = palabra.toLowerCase();
        int n = 0;
        for (int i = 0; i < p.length(); i++) {
            char c = p.charAt(i);
            if (!vocal(c)) continue;
            if (i == 0 || !vocal(p.charAt(i - 1))) n++; // vocal tras consonante: sílaba nueva
            else if (fuerte(c) && fuerte(p.charAt(i - 1))) n++; // hiato: dos fuertes
            // si no, diptongo: se une a la vocal anterior
        }
        int ultima = p.length() - 1;
        if (ultima > 0 && p.charAt(ultima) == 'y' && !vocal(p.charAt(ultima - 1))) n++;
        return Math.max(n, 1);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        StringBuilder todo = new StringBuilder();
        while (sc.hasNextLine()) todo.append(sc.nextLine()).append(' ');
        String texto = todo.toString().trim();

        int palabras = 0, silabas = 0;
        Matcher m = Pattern.compile("\\p{L}+").matcher(texto);
        while (m.find()) {
            palabras++;
            silabas += silabas(m.group());
        }
        int frases = 0;
        for (String f : texto.split("[.!? ]")) if (f.matches(".*\\p{L}.*")) frases++;

        double porPalabra = (double) silabas / palabras;
        double porFrase = (double) palabras / frases;
        double indice = 206.84 - 0.60 * (porPalabra * 100) - 1.02 * porFrase;
        String nivel = indice >= 90 ? "muy fácil" : indice >= 80 ? "fácil" : indice >= 70 ? "algo fácil" :
            indice >= 60 ? "normal" : indice >= 50 ? "algo difícil" : indice >= 30 ? "difícil" :
            "muy difícil";

        System.out.println("Palabras: " + palabras + " · Frases: " + frases + " · Sílabas: " + silabas);
        System.out.printf(Locale.US, "Sílabas por palabra: %.2f · Palabras por frase: %.1f%n",
            porPalabra, porFrase);
        System.out.printf(Locale.US, "Índice Fernández Huerta: %.1f (%s)%n", indice, nivel);
    }
}
```

Contar sílabas es contar núcleos vocálicos. Al recorrer la palabra, una vocal abre sílaba nueva si la anterior es una consonante, o si las dos son fuertes (hiato, «po-e-ta»); en cualquier otro caso hay diptongo y se une a la anterior («cie-lo»). Por eso hay dos funciones auxiliares, `vocal` y `fuerte`, que hacen el código casi igual que la regla.

Las palabras se cuentan con la expresión `\p{L}+` («una o más letras», incluidas las que llevan tilde o la ñ) y las frases partiendo por `. ! ?` y quedándose con los trozos que tienen alguna letra, para no contar el hueco que queda tras el último punto.

Las sílabas por cada 100 palabras son las sílabas por palabra multiplicadas por 100. Las divisiones se hacen con `(double)` para no perder los decimales.

13. El CHANGELOG a partir de los commits

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

Solución (java)

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

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

        Pattern formato = Pattern.compile("^((\\w+)(?:\\(([^\\)]+\\))?)?(!)? : (.+)$");
        List<String> incompatibles = new ArrayList<>(), novedades = new ArrayList<>(), correcciones = new
            ArrayList<>();
        int sinEntrada = 0;
        for (String c : commits) {
            Matcher m = formato.matcher(c);
            if (!m.matches()) { sinEntrada++; continue; }
            String entrada = "- " + (m.group(2) != null ? "***" + m.group(2) + "***: " : "") + m.group(4);
            if (m.group(3) != null) incompatibles.add(entrada);
            else if (m.group(1).equals("feat")) novedades.add(entrada);
            else if (m.group(1).equals("fix")) correcciones.add(entrada);
            else sinEntrada++;
        }

        String[] v = version.split("\\.");
        int mayor = Integer.parseInt(v[0]), menor = Integer.parseInt(v[1]), parche =
            Integer.parseInt(v[2]);
        if (!incompatibles.isEmpty()) { mayor++; menor = 0; parche = 0; }
        else if (!novedades.isEmpty()) { menor++; parche = 0; }
        else if (!correcciones.isEmpty()) parche++;
        else {
            System.out.println("Sin cambios publicables: se mantiene la versión " + version);
            return;
        }

        System.out.println("## " + mayor + "." + menor + "." + parche);
        String[] titulos = {"Cambios incompatibles", "Novedades", "Correcciones"};
        List<List<String>> secciones = List.of(incompatibles, novedades, correcciones);
        for (int i = 0; i < 3; i++) {
            if (secciones.get(i).isEmpty()) continue;
            System.out.println();
            System.out.println("### " + titulos[i]);
            secciones.get(i).forEach(System.out::println);
        }
        System.out.println();
        System.out.println("(" + sinEntrada + " commits sin entrada en el CHANGELOG)");
    }
}
```

Una sola expresión regular separa las partes del mensaje: el tipo, el ámbito opcional entre paréntesis, el «!» opcional y la descripción. Los grupos `(?: ... )` no capturan; sirven para hacer opcional el paréntesis entero y quedarse solo con lo de dentro.

Cada commit va a una de tres listas, y la versión se decide por la lista más importante que tenga algo: un cambio incompatible sube la mayor y pone a cero las otras dos, una novedad sube la menor y pone a cero el parche, y una corrección sube el parche. Es exactamente lo que hacen herramientas como semantic-release.

Las tres secciones se imprimen con un bucle sobre dos arrays paralelos (títulos y listas) en lugar de repetir el mismo código tres veces.

14. De Javadoc a una página de documentación

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

```

import java.util.*;

public class Main {
    // Texto de una línea de Javadoc sin /**, */ ni el * del principio
    static String limpiar(String l) {
        String s = l.trim();
        if (s.startsWith("/**")) s = s.substring(3);
        if (s.endsWith("*/")) s = s.substring(0, s.length() - 2);
        s = s.trim();
        if (s.startsWith("*")) s = s.substring(1);
        return s.trim();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<String> lineas = new ArrayList<>();
        while (sc.hasNextLine()) lineas.add(sc.nextLine());

        List<String> javadoc = null; // el último comentario /** */ pendiente
        int documentados = 0;
        List<String> sinDocumentar = new ArrayList<>();
        for (int i = 0; i < lineas.size(); i++) {
            String l = lineas.get(i).trim();
            if (l.isEmpty()) continue;
            if (l.startsWith("/**")) {
                javadoc = new ArrayList<>();
                while (true) {
                    String c = lineas.get(i);
                    String texto = limpiar(c);
                    if (!texto.isEmpty()) javadoc.add(texto);
                    if (c.contains("*/")) break;
                    i++;
                }
                continue;
            }
            boolean metodoPublico = l.startsWith("public") && l.contains("(") && !l.matches("public
            .*\\b(class|interface|enum|record)\\b.*");
            if (metodoPublico) {
                String firma = l.substring("public".length()).trim();
                if (firma.contains("{") firma = firma.substring(0, firma.indexOf('{')).trim();
                String[] antes = firma.substring(0, firma.indexOf('(')).trim().split("\\s+");
                String nombre = antes[antes.length - 1];
                if (javadoc == null) {
                    sinDocumentar.add(nombre);
                } else {
                    documentados++;
                    imprimir(nombre, firma, javadoc);
                }
            }
            javadoc = null; // un Javadoc solo vale para la línea que tiene justo debajo
        }
        System.out.println("Documentados: " + documentados + " · Sin documentar: "
            + (sinDocumentar.isEmpty() ? "ninguno" : String.join(", ", sinDocumentar)));
    }

    static void imprimir(String nombre, String firma, List<String> javadoc) {
        List<String> descripcion = new ArrayList<>(), params = new ArrayList<>(), lanza = new ArrayList<>();
        String devuelve = null;
        for (String t : javadoc) {
            String[] p = t.split("\\s+", 3);
            switch (p[0]) {
                case "@param" -> params.add("- ` " + p[1] + "` : " + (p.length > 2 ? p[2] : ""));
                case "@throws", "@exception" -> lanza.add("- ` " + p[1] + "` : " + (p.length > 2 ? p[2] : ""));
                case "@return" -> devuelve = t.substring("@return".length()).trim();
                default -> descripcion.add(t);
            }
        }
    }
}

```

```

    }
    System.out.println("### " + nombre);
    System.out.println("`" + firma + "`");
    System.out.println();
    System.out.println(String.join(" ", descripcion));
    if (!params.isEmpty()) {
        System.out.println();
        System.out.println("***Parámetros**");
        params.forEach(System.out::println);
    }
    if (devuelve != null) {
        System.out.println();
        System.out.println("***Devuelve**": " + devuelve);
    }
    if (!lanza.isEmpty()) {
        System.out.println();
        System.out.println("***Lanza**");
        lanza.forEach(System.out::println);
    }
    System.out.println();
}
}

```

El programa recorre las líneas recordando el último Javadoc leído. Cuando empieza un comentario `/**`, lee hasta la línea que contiene `*/`, limpiando de cada una los delimitadores y el asterisco inicial. La primera línea con código que viene después decide qué pasa con él: si es un método público se usa, y en cualquier caso se olvida (`javadoc = null`), para que no se aplique a un método posterior.

Para la firma se quita «public» y todo lo que va desde la llave; el nombre es la última palabra antes del paréntesis. La expresión con `\b(class|interface|enum|record)\b` descarta la declaración de la clase.

El método `imprimir` reparte las líneas del comentario según su etiqueta (`@param`, `@return`, `@throws`) y el resto forma la descripción. `split("\\s+", 3)` parte solo en tres trozos, así la descripción de un parámetro conserva sus espacios.

15. Un revisor automático de manuales

Muy difícil · Documentación de Aplicaciones · apuntesdam.com/subject/desarrollo-interfaces/topic/documentacion-

Solución (java)

```

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

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

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<String> lineas = new ArrayList<>();
        while (sc.hasNextLine()) lineas.add(sc.nextLine());

        // Qué líneas están dentro de un bloque de código
        boolean[] codigo = new boolean[lineas.size()];
        boolean dentro = false;
        for (int i = 0; i < lineas.size(); i++) {
            if (lineas.get(i).startsWith("```")) {
                dentro = !dentro;
                codigo[i] = true;
            } else {
                codigo[i] = dentro;
            }
        }

        List<String> problemas = new ArrayList<>();
        Pattern titulo = Pattern.compile("^#{1,6}\\s+(.*)$");
        Set<String> anclas = new HashSet<>();

        // 1. Títulos
        int anterior = 0;
        boolean primera = true;
        for (int i = 0; i < lineas.size(); i++) {
            if (codigo[i] || lineas.get(i).isBlank()) continue;
            Matcher m = titulo.matcher(lineas.get(i));
            if (primera && (!m.matches() || m.group(1).length() != 1)) {
                problemas.add("Línea " + (i + 1) + ": el documento debe empezar con un título de nivel 1");
            }
            primera = false;
            if (!m.matches()) continue;
            int nivel = m.group(1).length();
            if (anterior > 0 && nivel > anterior + 1) {
                problemas.add("Línea " + (i + 1) + ": salto de título de nivel " + anterior + " a " + nivel);
            }
            anterior = nivel;
            anclas.add(ancla(m.group(2)));
        }

        // 2. Imágenes y enlaces
        Pattern enlace = Pattern.compile("(?!\\)\\[\\[([\\^\\]]*)\\]\\]\\[\\[([\\^\\]]*)\\]\\]");
        Set<String> vagos = Set.of("aquí", "aquí", "pulsa aquí", "este enlace");
        for (int i = 0; i < lineas.size(); i++) {
            if (codigo[i]) continue;
            Matcher m = enlace.matcher(lineas.get(i));
            while (m.find()) {
                String texto = m.group(2).trim(), destino = m.group(3).trim();
                if (!m.group(1).isEmpty()) {
                    if (texto.isEmpty()) problemas.add("Línea " + (i + 1) + ": la imagen " + destino + " no tiene texto alternativo");
                    continue;
                }
                if (vagos.contains(texto.toLowerCase())) {

```

```

        problemas.add("Línea " + (i + 1) + ": el texto del enlace «" + texto + "» no dice
            adónde lleva");
    }
    if (destino.startsWith("#") && !anclas.contains(destino.substring(1))) {
        problemas.add("Línea " + (i + 1) + ": el enlace " + destino + " apunta a una sección
            que no existe");
    }
}

// 3. Bloques sin cerrar
if (dentro) problemas.add("Hay un bloque de código sin cerrar");

problemas.forEach(System.out::println);
System.out.println(problemas.isEmpty() ? "Manual correcto"
    : problemas.size() + (problemas.size() == 1 ? " problema" : " problemas"));
}
}

```

Antes de nada se marca qué líneas están dentro de un bloque de código, cambiando de estado cada vez que aparece una línea con ``; así las dos pasadas pueden saltarse esas líneas sin repetir la lógica. Si al final el estado sigue «dentro», el bloque no se cerró.

Hacen falta dos pasadas porque un enlace puede apuntar a una sección que está más abajo: la primera recoge todos los títulos (y comprueba los niveles) y guarda sus anclas en un `Set`; la segunda ya puede comprobar cada enlace interno contra ese conjunto.

Una sola expresión regular reconoce imágenes y enlaces: el grupo `(!?)` dice si la marca empieza por «!», y los otros dos capturan el texto y el destino. Con `while (m.find())` se tratan todos los de una línea, en orden.

16. Versiones semánticas y rangos de dependencias

Medio · Distribución de Aplicaciones · apuntesdam.com/subject/desarrollo-interfaces/topic/distribucion-aplicaciones

Solución (java)

```
import java.util.*;

public class Main {
    // Compara dos partes de una etiqueta: las numéricas como números y menores que las de texto
    static int compararParte(String a, String b) {
        boolean na = a.matches("\\d+"), nb = b.matches("\\d+");
        if (na && nb) return Long.compare(Long.parseLong(a), Long.parseLong(b));
        if (na != nb) return na ? -1 : 1;
        return a.compareTo(b);
    }

    static int comparar(String a, String b) {
        String[] pa = a.split("-", 2), pb = b.split("-", 2);
        String[] na = pa[0].split("\\."), nb = pb[0].split("\\.");
        for (int i = 0; i < 3; i++) {
            int c = Integer.compare(Integer.parseInt(na[i]), Integer.parseInt(nb[i]));
            if (c != 0) return c;
        }
        boolean ea = pa.length > 1, eb = pb.length > 1;
        if (!ea || !eb) return Boolean.compare(!ea, !eb); // sin etiqueta es mayor
        String[] ta = pa[1].split("\\."), tb = pb[1].split("\\.");
        for (int i = 0; i < Math.min(ta.length, tb.length); i++) {
            int c = compararParte(ta[i], tb[i]);
            if (c != 0) return c;
        }
        return Integer.compare(ta.length, tb.length); // la que se acaba antes es menor
    }

    static boolean cumple(String version, String rango) {
        if (!rango.startsWith("^") && !rango.startsWith("~")) return comparar(version, rango) == 0;
        String base = rango.substring(1);
        String[] n = base.split("\\.");
        int x = Integer.parseInt(n[0]), y = Integer.parseInt(n[1]), z = Integer.parseInt(n[2]);
        String limite;
        if (rango.startsWith("~") || x == 0 && y > 0) limite = x + "." + (y + 1) + ".0";
        else if (x > 0) limite = (x + 1) + ".0.0";
        else limite = "0.0." + (z + 1);
        return comparar(version, base) >= 0 && comparar(version, limite) < 0;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        List<String> versiones = new ArrayList<>(Arrays.asList(sc.nextLine().trim().split(" ")));
        versiones.sort(Main::comparar);
        System.out.println("Ordenadas: " + String.join(" < ", versiones));
        while (sc.hasNextLine()) {
            String rango = sc.nextLine().trim();
            if (rango.isEmpty()) continue;
            String mejor = null;
            for (String v : versiones) if (cumple(v, rango) && !v.contains("-")) mejor = v; // la más alta estable
            System.out.println(rango + " → " + (mejor == null ? "ninguna" : mejor));
        }
    }
}
```

Comparar versiones como texto falla («1.10» < «1.9»), así que se separan y se comparan los tres números como enteros. Si coinciden, decide la etiqueta: sin etiqueta gana (una 2.0.0 es posterior a su 2.0.0-rc.1) y, entre dos etiquetas, se comparan parte a parte con `compararParte`, que distingue números de texto (por eso beta.2 < beta.11).

Cada rango se convierte en un intervalo [base, límite): el límite es la siguiente versión mayor para ^ (o la menor o el parche cuando las cifras de la izquierda son 0) y la siguiente menor para ~. Una vez calculado el límite como texto, basta con reutilizar `comparar`.

`Boolean.compare(!ea, !eb)` resume el caso de que alguna no tenga etiqueta: la que no la tiene vale `true`, y `true` es mayor que `false`.

17. ¿Puedo distribuir mi aplicación con estas librerías?

Difícil · Distribución de Aplicaciones · apuntesdam.com/subject/desarrollo-interfaces/topic/distribucion-aplicaciones

Solución (java)

```
import java.util.*;

public class Main {
    static final Set<String> PERMISIVAS = Set.of("MIT", "BSD-3-Clause", "Apache-2.0", "EPL-2.0");

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] proyecto = sc.nextLine().trim().split(" "); // proyecto LICENCIA (MIT, Apache-2.0,
        GPL-3.0, Propietaria)
        String licProyecto = proyecto[1];
        List<String[]> deps = new ArrayList<>(); // nombre licencia
        enlace(estatico|dinamico)
        while (sc.hasNextLine()) { String l = sc.nextLine().trim(); if (!l.isEmpty()) deps.add(l.split("
")); }

        int problemas = 0, avisos = 0;
        boolean proyectoGpl = licProyecto.startsWith("GPL");
        for (String[] d : deps) {
            String lic = d[1];
            boolean estatico = d[2].equals("estatico");
            String veredicto;
            if (PERMISIVAS.contains(lic)) {
                veredicto = lic.equals("Apache-2.0") && licProyecto.equals("GPL-2.0")
                    ? "PROBLEMA (Apache-2.0 no es compatible con GPL-2.0)"
                    : "OK (permisiva: incluye su aviso de copyright)";
            } else if (lic.equals("LGPL-3.0")) {
                if (!estatico) {
                    veredicto = "OK (LGPL enlazada dinámicamente: el usuario debe poder sustituirla)";
                    avisos++;
                } else if (proyectoGpl) {
                    veredicto = "OK (compatible con GPL)";
                } else {
                    veredicto = "PROBLEMA (LGPL enlazada estáticamente obliga a facilitar el código
                    objeto para reenlazar)";
                }
            } else if (lic.startsWith("GPL")) {
                if (lic.equals(licProyecto)) veredicto = "OK (misma licencia)";
                else if (lic.equals("GPL-2.0") && licProyecto.equals("GPL-3.0")) veredicto = "PROBLEMA
                (GPL-2.0 sola no es compatible con GPL-3.0)";
                else veredicto = "PROBLEMA (copyright fuerte: obligaría a publicar el proyecto como " +
                lic + ")";
            } else if (lic.equals("AGPL-3.0")) {
                veredicto = "PROBLEMA (AGPL: también obliga a compartir el código si se ofrece como
                servicio en red)";
            } else {
                veredicto = "REVISAR (licencia desconocida: hay que leerla)";
                avisos++;
            }
            if (veredicto.startsWith("PROBLEMA")) problemas++;
            System.out.println(d[0] + " [" + lic + ", " + d[2] + "]: " + veredicto);
        }
        System.out.println("Resumen: " + problemas + " problemas, " + avisos + " avisos → "
            + (problemas == 0 ? "se puede distribuir" : "no distribuir sin resolverlos"));
    }
}
```

Las reglas se agrupan por familias de licencias (permisivas, LGPL, GPL, AGPL y desconocidas) en una cadena de `if/else if`, y dentro de cada familia se miran el tipo de enlace y la licencia del proyecto. Así cada regla del enunciado está en un solo sitio.

Los problemas no se cuentan en cada rama: basta con mirar al final si el veredicto empieza por «PROBLEMA». Los avisos sí se suman en las dos ramas que los producen. Las permisivas van en un `Set` para comprobarlas con `contains`.

18. Un actualizador incremental con despliegue escalonado

Muy difícil · Distribución de Aplicaciones · apuntesdam.com/subject/desarrollo-interfaces/topic/distribucion-aplicaciones

Solución (java)

```
import java.util.*;

public class Main {
    static int comparar(String a, String b) {
        String[] x = a.split("\\."), y = b.split("\\.");
        for (int i = 0; i < 3; i++) {
            int c = Integer.compare(Integer.parseInt(x[i]), Integer.parseInt(y[i]));
            if (c != 0) return c;
        }
        return 0;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, String> conf = new HashMap<>();
        Map<String, String[]> local = new TreeMap<>(), remoto = new TreeMap<>(); // ruta → {tamaño, hash}
        Map<String, String[]> actual = null;
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (l.isEmpty()) continue;
            if (l.equals("[local]")) { actual = local; continue; }
            if (l.equals("[remoto]")) { actual = remoto; continue; }
            String[] p = l.split(" ");
            if (actual == null) conf.put(p[0], p[1]); // version_instalada, version_nueva, minima, porcentaje, usuario
            else actual.put(p[0], new String[]{p[1], p[2]});
        }

        String instalada = conf.get("version_instalada"), nueva = conf.get("version_nueva"), minima = conf.get("minima");
        int porcentaje = Integer.parseInt(conf.get("porcentaje"));
        if (comparar(instalada, nueva) >= 0) {
            System.out.println("Ya tienes la última versión (" + instalada + ")");
            return;
        }
        boolean obligatoria = comparar(instalada, minima) < 0;
        int grupo = Math.floorMod(conf.get("usuario").hashCode(), 100);
        if (!obligatoria && grupo >= porcentaje) {
            System.out.println("Actualización " + nueva + " aún no disponible para este usuario (grupo " + grupo + ", se publica al " + porcentaje + " %)");
            return;
        }

        System.out.println("Actualizar de " + instalada + " a " + nueva
            + (obligatoria ? " (obligatoria: la versión mínima es " + minima + ")" : " (grupo " + grupo + ")");
        long descarga = 0, total = 0;
        int ficheros = 0;
        for (Map.Entry<String, String[]> e : remoto.entrySet()) {
            long tam = Long.parseLong(e.getValue()[0]);
            total += tam;
            String[] antes = local.get(e.getKey());
            if (antes != null && antes[1].equals(e.getValue()[1])) continue; // mismo hash: no se descarga
            System.out.println(" " + (antes == null ? "NUEVO " : "CAMBIA ") + e.getKey());
            descarga += tam;
            ficheros++;
        }
        for (String ruta : local.keySet()) {
            if (!remoto.containsKey(ruta)) System.out.println(" BORRAR " + ruta);
        }
        System.out.printf(Locale.US, "Descarga: %d ficheros, %.1f KB de %.1f KB (ahorro del %.0f %%)\n",
            ficheros, descarga / 1024.0, total / 1024.0, 100.0 * (total - descarga) / total);
    }
}
```

Las versiones se comparan número a número, como en cualquier actualizador. El despliegue escalonado usa `Math.floorMod(usuario.hashCode(), 100)` para asignar a cada usuario un grupo fijo del 0 al 99: si el porcentaje publicado es 10, solo lo reciben los grupos del 0 al 9, y siempre los mismos usuarios. Se usa `floorMod` y no `%` porque `hashCode` puede ser negativo.

Para saber qué descargar se comparan los dos manifiestos por ruta: lo que no está en el local es nuevo, lo que está con otro hash ha cambiado y lo que solo está en el local hay que borrarlo. Como los dos mapas son `TreeMap`, se recorren ya en orden alfabético.

Los tamaños se suman en `long` y se convierten a KB al final dividiendo entre `1024.0` (con decimales). En `printf`, `%%` escribe el símbolo del porcentaje.

19. Casos de prueba por valores límite

Medio · Pruebas de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/pruebas-interfaces

Solución (java)

```
import java.util.*;

public class Main {
    static String texto(int n) {
        if (n == 0) return "\"\"";
        if (n > 12) return "texto de " + n + " caracteres";
        return "\"\" + \"a\".repeat(n) + \"\" (" + n + (n == 1 ? " carácter" : " caracteres"));
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");    // campo tipo min max
            if (p.length < 4) continue;
            String campo = p[0], tipo = p[1];
            int min = Integer.parseInt(p[2]), max = Integer.parseInt(p[3]);

            System.out.println("== " + campo + " (" + tipo + ", " + min + ".." + max + ")");
            List<String> casos = new ArrayList<>();
            // Valores límite, sin repetir y en orden
            Set<Integer> limites = new LinkedHashSet<>(List.of(min - 1, min, min + 1, max - 1, max, max + 1));
            for (int v : limites) {
                if (tipo.equals("texto") && v < 0) continue;
                String resultado = v >= min && v <= max ? "válido" : "inválido";
                casos.add((tipo.equals("entero") ? String.valueOf(v) : texto(v)) + " → " + resultado);
            }
            if (tipo.equals("entero")) {
                int medio = (min + max) / 2;
                if (!limites.contains(medio)) casos.add(medio + " → válido");
                casos.add("abc → inválido (no es un número)");
            } else {
                casos.add("\" \" (solo espacios) → inválido (vacío tras recortar)");
            }
            casos.add("(vacío) → inválido (obligatorio)");

            for (int i = 0; i < casos.size(); i++) System.out.println(" " + campo + "-" + (i + 1) + ": " + casos.get(i));
        }
    }
}
```

Los seis valores límite se meten en un `LinkedHashSet`, que elimina los repetidos (en un rango 0..2, `min + 1` y `max - 1` son el mismo) y conserva el orden de inserción. Cada uno es válido si cae dentro del rango.

Los casos se acumulan en una lista y se numeran al final, así no hay que llevar la cuenta mientras se generan. La representación de un texto de longitud `n` está en su propio método, `texto`, con sus tres formatos.

Además de los límites, se añaden casos de clases de equivalencia inválidas que cualquier formulario debe rechazar: algo que no es un número, un texto que solo tiene espacios y el campo vacío.

20. Un mini ejecutor de pruebas unitarias

Difícil · Pruebas de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/pruebas-interfaces

Solución (java)

```
import java.util.*;

public class Main {
    static double precioFinal(double importe, String codigo, boolean socio) {
        if (importe < 0) throw new IllegalArgumentException("importe negativo");
        int descuento = 0;
        if (codigo.equals("VERANO10")) descuento = 10;
        else if (codigo.equals("VIP25") && importe >= 100) descuento = 25;
        if (socio) descuento += 5;
        descuento = Math.min(descuento, 30);
        double precio = importe * (100 - descuento) / 100;
        if (precio < 50 && importe != 0) precio += 4.95;
        return Math.round(precio * 100) / 100.0;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int pruebas = 0, pasan = 0, fallan = 0, errores = 0;
        // Cada línea: nombre_prueba importe codigo(o -) socio(si|no) esperado
        while (sc.hasNextLine()) {
            String[] p = sc.nextLine().trim().split(" ");
            if (p.length < 5) continue;
            pruebas++;
            String nombre = p[0], esperado = p[4];
            boolean esperaExcepcion = esperado.equals("excepcion");
            try {
                double obtenido = precioFinal(Double.parseDouble(p[1]), p[2], p[3].equals("si"));
                String texto = String.format(Locale.US, "%.2f", obtenido);
                if (esperaExcepcion) {
                    System.out.println("FALLA " + nombre + ": se esperaba una excepción y devolvió " +
                        texto);
                    fallan++;
                } else if (Math.abs(obtenido - Double.parseDouble(esperado)) < 0.005) {
                    System.out.println("PASA " + nombre);
                    pasan++;
                } else {
                    System.out.println("FALLA " + nombre + ": esperado " + esperado + ", obtenido " +
                        texto);
                    fallan++;
                }
            } catch (Exception e) {
                if (esperaExcepcion) {
                    System.out.println("PASA " + nombre + " (lanza " + e.getClass().getSimpleName() +
                        ")");
                    pasan++;
                } else {
                    System.out.println("ERROR " + nombre + ": " + e.getClass().getSimpleName() + ": " +
                        e.getMessage());
                    errores++;
                }
            }
        }
        System.out.println("Pruebas: " + pruebas + " · Pasan: " + pasan + " · Fallan: " + fallan + " ·
            Errores: " + errores);
    }
}
```

`precioFinal` sigue la especificación paso a paso: valida, calcula el porcentaje de descuento (con el tope del 30 %), lo aplica, suma el envío y redondea a céntimos con `Math.round(x * 100) / 100.0`.

El ejecutor distingue tres resultados como JUnit: PASA si el valor coincide (o si se esperaba una excepción y se lanzó), FALLA si el resultado es otro, y ERROR si salta una excepción que nadie esperaba. Esa diferencia es útil: un fallo suele indicar un error de lógica y un error, un caso que el código no contempla.

Los números con decimales se comparan con una tolerancia (`Math.abs(a - b) < 0.005`) y nunca con `==`, porque los `double` arrastran pequeños errores de redondeo.

21. Pruebas de interfaz guionizadas

Muy difícil · Pruebas de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/pruebas-interfaces

Solución (java)

```
import java.util.*;

public class Main {
    // Modelo de una pantalla de acceso: campos usuario y clave, botón entrar y un mensaje
    static class PantallaAcceso {
        String usuario = "", clave = "", mensaje = "";
        boolean entrarHabilitado = false, dentro = false;
        int intentos = 0;
        static final Map<String, String> CUENTAS = Map.of("ana", "Segura123", "luis", "Clave2026");

        private void actualizarBoton() {
            entrarHabilitado = !usuario.isBlank() && clave.length() >= 4 && intentos < 3;
        }

        void escribir(String campo, String texto) {
            if (campo.equals("usuario")) usuario = texto;
            else if (campo.equals("clave")) clave = texto;
            actualizarBoton();
        }

        void pulsarEntrar() {
            if (!entrarHabilitado) return;
            String u = usuario.trim();
            if (clave.equals(CUENTAS.get(u.toLowerCase()))) {
                dentro = true;
                mensaje = "Bienvenida, " + u;
                return;
            }
            clave = "";
            intentos++;
            mensaje = intentos >= 3 ? "Cuenta bloqueada: demasiados intentos"
                : "Usuario o contraseña incorrectos (" + (3 - intentos) + " intentos más)";
            actualizarBoton();
        }

        String consultar(String que) {
            return switch (que) {
                case "mensaje" -> mensaje;
                case "boton" -> entrarHabilitado ? "habilitado" : "deshabilitado";
                case "clave" -> clave;
                case "pantalla" -> dentro ? "principal" : "acceso";
                default -> "?";
            };
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        PantallaAcceso pantalla = new PantallaAcceso();
        int paso = 0, fallos = 0;
        while (sc.hasNextLine()) {
            String linea = sc.nextLine();
            if (linea.isBlank() || linea.trim().startsWith("#")) continue;
            paso++;
            String[] p = linea.split(" ", 3); // orden, objeto y el resto (que puede tener espacios)
            String valor = p.length > 2 ? p[2] : "";
            switch (p[0]) {
                case "escribe" -> pantalla.escribir(p[1], valor);
                case "pulsa" -> pantalla.pulsarEntrar();
                case "espera" -> {
                    String real = pantalla.consultar(p[1]);
                    if (real.equals(valor)) {
                        System.out.println("✓ paso " + paso + ": " + p[1] + " = «" + real + "»");
                    } else {
                        System.out.println("X paso " + paso + ": " + p[1] + " esperaba «" + valor + "» pero es «" + real + "»");
                    }
                }
            }
        }
    }
}
```

```

        fallos++;
    }
    }
    default -> System.out.println("? paso " + paso + ": orden desconocida «" + p[0] + "»");
}
}
System.out.println(fallos == 0 ? "PRUEBA SUPERADA" : "PRUEBA FALLIDA (" + fallos + "
comprobaciones)");
}
}
}

```

Separar el modelo de la pantalla (su estado y sus reglas) de quien lo maneja es lo que permite probar una interfaz sin dibujarla: el guion solo escribe, pulsa y consulta, igual que haría un usuario. El método `actualizarBoton` concentra la regla de cuándo se habilita el botón y se llama después de cada cambio.

Cada línea se parte con `split(" ", 3)`: el tercer trozo es todo lo que queda, con sus espacios, que es lo que necesitan los mensajes. Por eso la línea no se recorta: en «espera clave» el valor esperado es la cadena vacía.

El método `consultar` traduce el estado interno (booleanos) a los valores del guion, de modo que comparar lo esperado con lo real es una simple comparación de cadenas.

22. Validar un NIF en un formulario

Fácil · Ejercicios de Desarrollo de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/ejercicios-desarrollo-interfaces

Solución (java)

```

import java.util.*;

public class Main {
    static final String LETRAS = "TRWAGMYFPDXBNJZSQVHLCKE";

    static boolean nifValido(String nif) {
        String s = nif.trim().replace(" ", "").replace("-", "").toUpperCase();
        if (!s.matches("\\d{8}[A-Z]")) return false;
        int numero = Integer.parseInt(s.substring(0, 8));
        return LETRAS.charAt(numero % 23) == s.charAt(8);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            String l = sc.nextLine();
            if (!l.isBlank()) System.out.println(nifValido(l) ? "válido" : "no válido");
        }
    }
}

```

Primero se limpia el NIF (sin espacios ni guiones y en mayúsculas) y después se comprueba el formato con la expresión `\\d{8}[A-Z]`: exactamente ocho dígitos y una letra. Solo si el formato es correcto se calcula la letra de control, el resto de dividir el número entre 23 usado como posición en la cadena `LETRAS`.

23. Informe agrupado con subtotales

Medio · Ejercicios de Desarrollo de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/ejercicios-

Solución (java)

```
import java.util.*;

public class Main {
    static String importe(double v) {
        return String.format(Locale.US, "%.2f", v).replace('.', ',');
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, Double> subtotales = new TreeMap<>(); // ordenado por categoría
        double total = 0;
        while (sc.hasNextLine()) {
            String l = sc.nextLine().trim();
            if (l.isEmpty()) continue;
            String[] p = l.split(";");
            double v = Double.parseDouble(p[1]);
            subtotales.merge(p[0], v, Double::sum);
            total += v;
        }
        subtotales.forEach((cat, v) -> System.out.println(cat + ": " + importe(v)));
        System.out.println("TOTAL: " + importe(total));
    }
}
```

Un `TreeMap` agrupa y ordena a la vez: `merge(categoría, importe, Double::sum)` suma el importe al subtotal de su categoría (o lo crea), y al recorrerlo las categorías salen en orden alfabético.

Para la coma decimal se formatea con `Locale.US` (punto) y se cambia el punto por una coma, lo que da el mismo resultado en cualquier ordenador, sea cual sea su idioma.

24. Paginación de una tabla

Medio · Ejercicios de Desarrollo de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/ejercicios-desarrollo-interfaces

Solución (java)

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int tam = sc.nextInt(), pagina = sc.nextInt();
        sc.nextLine();
        List<String> elementos = new ArrayList<>();
        while (sc.hasNextLine()) {
            String l = sc.nextLine();
            if (!l.isBlank()) elementos.add(l);
        }

        if (elementos.isEmpty()) {
            System.out.println("Sin resultados");
            return;
        }
        int paginas = (elementos.size() + tam - 1) / tam; // división redondeando hacia arriba
        if (pagina < 1 || pagina > paginas) {
            System.out.println("Página no válida");
            return;
        }
        int desde = (pagina - 1) * tam, hasta = Math.min(desde + tam, elementos.size());
        elementos.subList(desde, hasta).forEach(System.out::println);
        System.out.println("Página " + pagina + " de " + paginas);
    }
}
```

El número de páginas es la división redondeando hacia arriba: $(n + \text{tam} - 1) / \text{tam}$ lo consigue con enteros (5 elementos en páginas de 3 son 2 páginas).

La página p empieza en el elemento $(p - 1) * \text{tam}$ y termina tam elementos después, salvo en la última, que puede tener menos: por eso el final se limita con `Math.min`. `subList` devuelve justo ese trozo de la lista.

25. Comparar versiones semánticas

Medio · Ejercicios de Desarrollo de Interfaces · apuntesdam.com/subject/desarrollo-interfaces/topic/ejercicios-desarrollo-interfaces

Solución (java)

```
import java.util.*;

public class Main {

    static int comparar(String a, String b) {
        String[] x = a.split("\\."), y = b.split("\\.");
        for (int i = 0; i < 3; i++) {
            int c = Integer.compare(Integer.parseInt(x[i]), Integer.parseInt(y[i]));
            if (c != 0) return c; // la primera parte distinta decide
        }
        return 0;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNext()) {
            int c = comparar(sc.next(), sc.next());
            System.out.println(c < 0 ? "<" : c > 0 ? ">" : "=");
        }
    }
}
```

`compareTo` compara texto carácter a carácter, y como «1» va antes que «9», dice que 2.10.0 es menor que 2.9.5. Hay que separar las tres partes por el punto (en `split` el punto se escapa, `\\.`, porque en una expresión regular significa «cualquier carácter») y compararlas como números.

La primera parte que es distinta decide el resultado; si las tres coinciden, las versiones son iguales.

26. Componente web accesible: valoración con estrellas

Muy difícil · JavaScript · 100 minutos · apuntesdam.com/ejercicios/javascript/componente-web-de-valoracion

Apartado 1 (javascript)

```
class ValoracionEstrellas extends HTMLElement {
  constructor() {
    super();
    // Shadow DOM abierto: el componente tiene su propio HTML y CSS, aislados de la página
    this.attachShadow({ mode: "open" });
  }

  get max() {
    return Math.max(1, parseInt(this.getAttribute("max"), 10) || 5);
  }

  connectedCallback() {
    this.render();
  }

  render() {
    this.shadowRoot.innerHTML = `<style>
      button { font-size: 1.5rem; background: none; border: 0; padding: 2px; cursor: pointer; color:
        #767676; }
      .llena { color: #b35c00; }
    </style>
    <div role="radiogroup"></div>`;
    const grupo = this.shadowRoot.querySelector("[role=radiogroup]");
    grupo.setAttribute("aria-label", this.getAttribute("etiqueta") || "Valoración");
    for (let i = 1; i <= this.max; i++) {
      const b = document.createElement("button");
      b.type = "button";
      b.textContent = "★";
      b.setAttribute("aria-label", i === 1 ? "1 estrella" : `${i} estrellas`);
      grupo.append(b);
    }
  }
}

customElements.define("valoracion-estrellas", ValoracionEstrellas);
```

Apartado 2 (javascript)

```
class ValoracionEstrellas extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: "open" });
  }

  get max() {
    return Math.max(1, parseInt(this.getAttribute("max"), 10) || 5);
  }

  get valor() {
    return Math.min(this.max, Math.max(0, parseInt(this.getAttribute("valor"), 10) || 0));
  }

  connectedCallback() {
    this.render();
  }

  render() {
    this.shadowRoot.innerHTML = `<style>
      button { font-size: 1.5rem; background: none; border: 0; padding: 2px; cursor: pointer; color:
        #767676; }
      .llena { color: #b35c00; }
    </style>
    <div role="radiogroup"></div>`;
    const grupo = this.shadowRoot.querySelector("[role=radiogroup]");
    grupo.setAttribute("aria-label", this.getAttribute("etiqueta") || "Valoración");
    for (let i = 1; i <= this.max; i++) {
      const b = document.createElement("button");
      b.type = "button";
      b.textContent = "★";
      b.setAttribute("role", "radio");
      b.setAttribute("aria-label", i === 1 ? "1 estrella" : `${i} estrellas`);
      b.setAttribute("aria-checked", String(i === this.valor));
      if (i <= this.valor) b.classList.add("llena");
      b.addEventListener("click", () => this.elegir(i));
      grupo.append(b);
    }
  }

  elegir(n) {
    this.setAttribute("valor", String(n));
    this.render();
    // composed: el evento sale del shadow DOM y llega a la página
    this.dispatchEvent(new CustomEvent("cambio", { detail: { valor: n }, bubbles: true, composed: true
    })));
  }
}

customElements.define("valoracion-estrellas", ValoracionEstrellas);
```

Apartado 3 (javascript)

```
class ValoracionEstrellas extends HTMLElement {
  constructor() {
    super();
    this.attachShadow({ mode: "open" });
  }

  get max() {
    return Math.max(1, parseInt(this.getAttribute("max"), 10) || 5);
  }

  get valor() {
    return Math.min(this.max, Math.max(0, parseInt(this.getAttribute("valor"), 10) || 0));
  }

  connectedCallback() {
    this.render();
  }

  render() {
    this.shadowRoot.innerHTML = `<style>
      button { font-size: 1.5rem; background: none; border: 0; padding: 2px; cursor: pointer; color:
        #767676; }
      .llena { color: #b35c00; }
    </style>
    <div role="radiogroup"></div>`;
    const grupo = this.shadowRoot.querySelector("[role=radiogroup]");
    grupo.setAttribute("aria-label", this.getAttribute("etiqueta") || "Valoración");
    for (let i = 1; i <= this.max; i++) {
      const b = document.createElement("button");
      b.type = "button";
      b.textContent = "★";
      b.setAttribute("role", "radio");
      b.setAttribute("aria-label", i === 1 ? "1 estrella" : `${i} estrellas`);
      b.setAttribute("aria-checked", String(i === this.valor));
      // Tabulador itinerante: solo la estrella elegida (o la primera) entra en el orden del tabulador
      b.tabIndex = i === (this.valor || 1) ? 0 : -1;
      if (i <= this.valor) b.classList.add("llena");
      b.addEventListener("click", () => this.elegir(i));
      b.addEventListener("keydown", (e) => this.teclado(e));
      grupo.append(b);
    }
  }

  elegir(n) {
    this.setAttribute("valor", String(n));
    this.render();
    this.dispatchEvent(new CustomEvent("cambio", { detail: { valor: n }, bubbles: true, composed: true }));
  }

  teclado(e) {
    const paso = { ArrowRight: 1, ArrowUp: 1, ArrowLeft: -1, ArrowDown: -1 }[e.key];
    if (!paso) return;
    e.preventDefault();
    const n = Math.min(this.max, Math.max(1, this.valor + paso));
    this.elegir(n);
    this.shadowRoot.querySelectorAll("button")[n - 1].focus();
  }
}

customElements.define("valoracion-estrellas", ValoracionEstrellas);
```

Apartado 4 (javascript)

```
class ValoracionEstrellas extends HTMLElement {
  // Los cambios de estos atributos desde fuera llaman a attributeChangedCallback
  static observedAttributes = ["valor", "max", "etiqueta"];

  constructor() {
    super();
    this.attachShadow({ mode: "open" });
  }

  get max() {
    return Math.max(1, parseInt(this.getAttribute("max"), 10) || 5);
  }

  get valor() {
    return Math.min(this.max, Math.max(0, parseInt(this.getAttribute("valor"), 10) || 0));
  }

  // La propiedad refleja el atributo: el.valor = 4 cambia valor="4" y el componente se redibuja
  set valor(n) {
    this.setAttribute("valor", String(n));
  }

  connectedCallback() {
    this.render();
  }

  attributeChangedCallback() {
    if (this.isConnected) this.render();
  }

  render() {
    this.shadowRoot.innerHTML = `<style>
      button { font-size: 1.5rem; background: none; border: 0; padding: 2px; cursor: pointer; color:
        #767676; }
      .llena { color: #b35c00; }
    </style>
    <div role="radiogroup"></div>`;
    const grupo = this.shadowRoot.querySelector("[role=radiogroup]");
    grupo.setAttribute("aria-label", this.getAttribute("etiqueta") || "Valoración");
    for (let i = 1; i <= this.max; i++) {
      const b = document.createElement("button");
      b.type = "button";
      b.textContent = "★";
      b.setAttribute("role", "radio");
      b.setAttribute("aria-label", i === 1 ? "1 estrella" : `${i} estrellas`);
      b.setAttribute("aria-checked", String(i === this.valor));
      b.tabIndex = i === (this.valor || 1) ? 0 : -1;
      if (i <= this.valor) b.classList.add("llena");
      b.addEventListener("click", () => this.elegir(i));
      b.addEventListener("keydown", (e) => this.teclado(e));
      grupo.append(b);
    }
  }

  elegir(n) {
    this.valor = n;
    this.dispatchEvent(new CustomEvent("cambio", { detail: { valor: n }, bubbles: true, composed: true }));
  }

  teclado(e) {
    const paso = { ArrowRight: 1, ArrowUp: 1, ArrowLeft: -1, ArrowDown: -1 }[e.key];
    if (!paso) return;
    e.preventDefault();
    const n = Math.min(this.max, Math.max(1, this.valor + paso));
    this.elegir(n);
    this.shadowRoot.querySelectorAll("button")[n - 1].focus();
  }
}
```

```
customElements.define("valoracion-estrellas", ValoracionEstrellas);
```

El componente encapsula estructura, estilo y comportamiento: la página solo escribe `<valoracion-estrellas max="5">` y escucha `cambio`. El Shadow DOM impide que el CSS de la página cambie las estrellas y que el del componente afecte a la página.

Toda la vista sale de `render()` a partir de los atributos, que son la única fuente de verdad. Con `observedAttributes` y `attributeChangedCallback`, cualquier cambio del atributo (desde el propio componente, desde otro script o desde un framework) redibuja el control.

Los roles `radiogroup` y `radio` con `aria-checked`, el tabulador itinerante y las flechas hacen que el componente se comporte exactamente como un grupo de radios nativo para el teclado y para los lectores de pantalla. Sin eso, serían diez botones sin relación entre ellos.

La propiedad `valor` refleja el atributo: es la convención de los elementos nativos (`input.value`), y permite usar el componente igual de cómodo desde JavaScript que desde HTML.

27. Informe de ventas imprimible: agrupación, formatos y CSS de impresión

Difícil · JavaScript · 75 minutos · apuntesdam.com/ejercicios/javascript/informe-de-ventas-imprimible

Apartado 1 (javascript)

```
const ventas = JSON.parse(document.getElementById("datos").textContent);

// Agrupa las ventas por comercial: { "Ana Gil": [venta, venta...], ... }
const grupos = {};
for (const v of ventas) (grupos[v.comercial] ??= []).push(v);

const tabla = document.createElement("table");
tabla.innerHTML = "<thead><tr><th>Comercial</th><th>Cliente</th><th>Fecha</th><th>Importe</th></tr>
</thead>";
const cuerpo = document.createElement("tbody");
let total = 0;

const fila = (celdas, clase) => {
  const tr = document.createElement("tr");
  if (clase) tr.className = clase;
  for (const c of celdas) {
    const td = document.createElement("td");
    td.textContent = c;
    tr.append(td);
  }
  return tr;
};

for (const comercial of Object.keys(grupos).sort((a, b) => a.localeCompare(b, "es"))) {
  const lista = grupos[comercial].sort((a, b) => a.fecha.localeCompare(b.fecha));
  let subtotal = 0;
  for (const v of lista) {
    cuerpo.append(fila([comercial, v.cliente, v.fecha, v.importe.toFixed(2)]));
    subtotal += v.importe;
  }
  cuerpo.append(fila([`Total ${comercial}`, "", "", subtotal.toFixed(2)], "subtotal"));
  total += subtotal;
}

const pie = document.createElement("tfoot");
pie.append(fila(["Total general", "", "", total.toFixed(2)], "total"));
tabla.append(cuerpo, pie);
document.getElementById("informe").append(tabla);
```

Apartado 2 (javascript)

```
const ventas = JSON.parse(document.getElementById("datos").textContent);

// Formatos de España: 1.250,50 € y 03/07/2026
const euros = new Intl.NumberFormat("es-ES", { style: "currency", currency: "EUR" });
const fechas = new Intl.DateTimeFormat("es-ES", { day: "2-digit", month: "2-digit", year: "numeric",
  timeZone: "UTC" });
const formatoFecha = (iso) => fechas.format(new Date(iso + "T00:00:00Z"));

const grupos = {};
for (const v of ventas) (grupos[v.comercial] ??= []).push(v);

const tabla = document.createElement("table");
const titulo = document.createElement("caption");
const dias = ventas.map((v) => v.fecha).sort();
titulo.textContent = `Del ${formatoFecha(dias[0])} al ${formatoFecha(dias[dias.length - 1])} .
  ${ventas.length} ventas`;
tabla.appendChild(titulo);
tabla.insertAdjacentHTML("beforeend", "<thead><tr><th scope=\"col\">Comercial</th><th
  scope=\"col\">Cliente</th><th scope=\"col\">Fecha</th><th scope=\"col\">Importe</th></tr></thead>");
const cuerpo = document.createElement("tbody");
let total = 0;

const fila = (celdas, clase) => {
  const tr = document.createElement("tr");
  if (clase) tr.className = clase;
  celdas.forEach((c, i) => {
    const td = document.createElement("td");
    td.textContent = c;
    if (i === 3) td.className = "importe";
    tr.appendChild(td);
  });
  return tr;
};

for (const comercial of Object.keys(grupos).sort((a, b) => a.localeCompare(b, "es"))) {
  const lista = grupos[comercial].sort((a, b) => a.fecha.localeCompare(b.fecha));
  let subtotal = 0;
  for (const v of lista) {
    cuerpo.appendChild(fila([comercial, v.cliente, formatoFecha(v.fecha), euros.format(v.importe)]));
    subtotal += v.importe;
  }
  cuerpo.appendChild(fila([`Total ${comercial}`, "", "", euros.format(subtotal)], "subtotal"));
  total += subtotal;
}

const pie = document.createElement("tfoot");
pie.appendChild(fila(["Total general", "", "", euros.format(total)], "total"));
tabla.appendChild(cuerpo, pie);
document.getElementById("informe").appendChild(tabla);
```

Apartado 3 (css)

```
table { border-collapse: collapse; width: 100%; font-size: 0.9rem; }
th, td { border-bottom: 1px solid #ccc; padding: 4px 8px; text-align: left; }
.importe { text-align: right; }
.subtotal td { font-weight: bold; background: #f2f2f2; }
.total td { font-weight: bold; border-top: 2px solid #000; }

@media print {
  .no-imprimir { display: none; }
  body { font-size: 10pt; color: #000; }
  /* La cabecera de la tabla se repite en cada página y ninguna fila se parte entre dos */
  thead { display: table-header-group; }
  tr { break-inside: avoid; }
  .subtotal td { background: none; }
}

@page {
  size: A4;
  margin: 2cm;
}
```

El informe separa datos, estructura y presentación: el JSON trae los datos, el JavaScript construye la tabla con su estructura semántica (`caption` , `thead` , `tbody` , `tfoot`) y el CSS decide cómo se ve en pantalla y en papel.

Agrupar con un objeto y ordenar las claves es el equivalente en JavaScript a los grupos de un generador de informes: cada grupo produce sus filas y su subtotal, y el total se acumula a la vez.

Intl evita formatear números y fechas a mano, con sus casos raros (separadores de miles, símbolos de moneda, zonas horarias). Crear la fecha en UTC y formatearla en UTC garantiza que el 3 de julio no se convierta en el 2 por la zona horaria del equipo.

La hoja de impresión es CSS normal dentro de `@media print` . Repetir la cabecera de la tabla y no partir filas son las dos reglas que más mejoran un informe largo; `@page` fija el tamaño del papel y los márgenes para quien guarde el PDF.

28. Página de un evento accesible: HTML semántico y ARIA

Medio · HTML y CSS · 60 minutos · apuntesdam.com/ejercicios/css/pagina-de-un-evento-accesible

Apartado 1 (html)

```
<a class="saltar" href="#principal">Saltar al contenido</a>
<header class="cabecera">
  
  <nav class="menu" aria-label="Principal">
    <a href="#programa">Programa</a>
    <a href="#entradas">Entradas</a>
  </nav>
</header>
<main id="principal" class="contenido">
  <h1 class="titulo">Jornadas de Software Libre 2026</h1>
  
  
  <h2 class="subtitulo" id="programa">Programa</h2>
  <table>
    <caption>Charlas del sábado</caption>
    <tr><th scope="col">Hora</th><th scope="col">Charla</th><th scope="col">Sala</th></tr>
    <tr><td>10:00</td><td>Linux en el aula</td><td>A</td></tr>
    <tr><td>11:30</td><td>Accesibilidad web</td><td>B</td></tr>
  </table>
  <a href="programa.pdf">Descarga el programa completo (PDF, 120 KB)</a>
</main>
```

Apartado 2 (javascript)

```
const boton = document.querySelector(".abrir-menu");
const menu = document.getElementById(boton.getAttribute("aria-controls"));

function abrir(abierto) {
  boton.setAttribute("aria-expanded", String(abierto));
  menu.hidden = !abierto;
}

boton.addEventListener("click", () => abrir(boton.getAttribute("aria-expanded") !== "true"));

// Escape cierra el menú y devuelve el foco al botón
document.addEventListener("keydown", (e) => {
  if (e.key === "Escape" && boton.getAttribute("aria-expanded") === "true") {
    abrir(false);
    boton.focus();
  }
});
```

Casi todas las barreras se arreglan cambiando `div` y `span` por el elemento que describe su función: `header`, `nav`, `main`, `h1`, `a`, `button`, `th`. Los navegadores y los lectores de pantalla ya saben qué hacer con ellos.

Los textos alternativos y los textos de los enlaces son contenido: deben tener sentido fuera de contexto, porque los lectores de pantalla permiten listar todos los enlaces de una página. «Pincha aquí» repetido diez veces no dice nada.

El menú desplegable usa ARIA solo para el estado (`aria-expanded`) y la relación (`aria-controls`); el resto lo da el `button`. Cerrar con Escape y devolver el foco al botón evita que el usuario de teclado se quede perdido.

Las pruebas de este ejercicio comprueban lo mismo que herramientas automáticas como axe o Lighthouse (estructura, alternativas, nombres de controles), pero ninguna herramienta sustituye a probar la página con teclado y con un lector de pantalla.