MEDAC
Instituto Oficial de Formación Profesional

DESARROLLO DE INTERFACES

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Explotación de componentes visuales

05

ÍNDICE

© MEDAC
Reservados todos los derechos. Queda rigurosamente prohibida, sin la autorización escrita de los titulares del copyright, bajo las sanciones establecidas en las leyes, la reproducción, transmisión y distribución total o parcial de esta obra por cualquier medio o procedimiento, incluidos la reprografía y el tratamiento informático.

ÍNDICE

MEDAC

OBJETIVOS

Crear componentes visuales.

Determinar los eventos a los que se debe responder desde una determinada interfaz.

Asociar las acciones correspondientes a la ocurrencia de un determinado evento.

Realizar pruebas sobre los componentes.

Documentar los componentes.

Programar aplicaciones que utilizan los componentes empaquetados.

/ 1. Introducción y contextualización práctica

Una característica esencial del desarrollo de interfaces es la posibilidad de interacción entre el usuario y la aplicación. Sin esta interacción, esta asignatura no tendría ningún sentido. ¿Para qué iba a ser necesario cuidar el diseño y la atención que se le pone al desarrollo de interfaces si no existiera dicha interacción?

Por esta razón, todos los componentes descritos hasta ahora son necesarios. Sobre todo, la capacidad de interactividad que estos pueden presentar, gracias a la aparición de los eventos.

Los eventos permiten que el usuario pueda establecer una «comunicación» con cualquier aplicación para que esta se ajuste a las decisiones del usuario en lo que respecta a su recorrido por un sitio web, aplicación o herramienta.

En este tema, abordaremos diferentes elementos relativos a la creación y utilización de eventos, así como ala generación de pruebas unitarias utilizando JUnit.

Escucha el siguiente audio en el que planteamos la contextualización práctica del tema. Encontrarás su resolución en el apartado final.

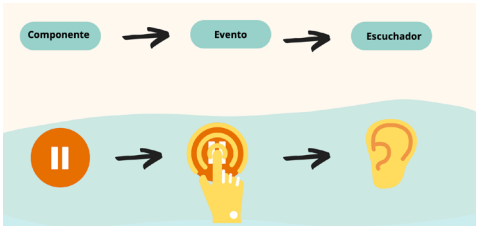

Fig. 1. Diagrama conexión tipos de ventanas.

Audio intro. "Creación y utilización de eventos y escuchadores en una interfaz"

https://bit.ly/33OhQhR

/ 2. Características fundamentales

Los componentes visuales son elementos clave para el desarrollo de interfaces. Su forma de utilización y estructura característica son esenciales para la implementación de aplicaciones a través de su uso. De una forma ágil y sencilla, liberan al desarrollador de los costes que supondría la creación de todos y cada uno de elementos visuales implicados en una sola interfaz de aplicación.

2.1. Introspección y reflexión

El lenguaje de programación Java es estático, lo que no permite alterar ciertos aspectos de los objetos durante el tiempo de ejecución. Sin embargo, existen dos conceptos que permiten introducir ciertas funcionalidades de lenguajes dinámicos: la reflexión y la introspección.

Mediante la **reflexión**, es posible recuperar y modificar de forma dinámica diferentes datos relativos a la estructura de un objeto: los métodos y propiedades de la clase, los constructores, las interfaces o el nombre original de la clase, entre otros.

La **introspección** es una de las características más importantes en el desarrollo de interfaces, sobre todo cuando se utilizan entornos visuales de diseño, puesto que permite a estas herramientas tomar de forma dinámica todos los métodos, propiedades o eventos asociados a un componente, que se colocan sobre el lienzo de diseño simplemente arrastrando y soltando el elemento.

Por lo tanto, la introspección de los componentes visuales requiere de la reflexión para funcionar correctamente. Ambas propiedades son dos características clave en el diseño de JavaBean que vimos en el tema anterior.

2.2. Persistencia del componente

El desarrollo de componentes requiere de la persistencia. Esta característica permite que el estado de una determinada clase no varíe, lo cual es posible a través de la serialización. Por esta razón, cuando se crea un nuevo componente, será necesario implementar alguna de las interfaces siguientes en función del tipo de serialización escogida:

- **Serialización automática.** Utiliza la interfaz: java.io.Serializable
- **Serialización programada.** Utiliza la interfaz: java.io.Externalizable

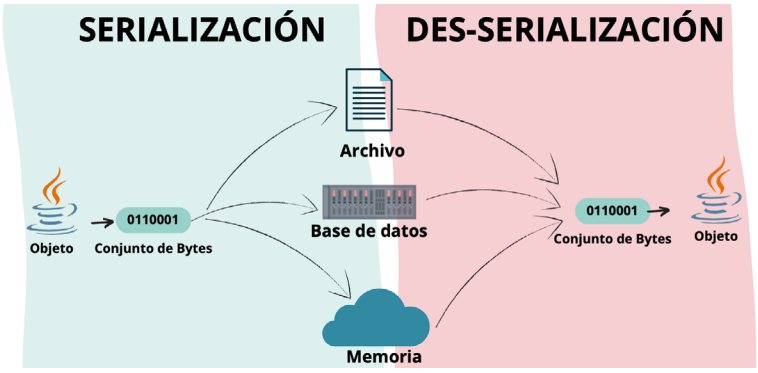

Fig. 2. Proceso serialización.

/ 3. Pruebas unitarias

El desarrollo de pruebas de software es importante en la implementación de nuevos componentes, puesto que, de esta forma, se reducen considerablemente los tiempos de desarrollo. Si se ha probado y verificado el comportamiento de un componente, esta acción ya no será necesaria, aunque sí habría que verificar cómo sería su uso en el marco de otro proyecto que lo utilice.

Fig. 3. Logo JUnit.

Las pruebas se diseñan en base al comportamiento esperado de un componente, extendiéndose a todas las casuísticas posibles.

Las pruebas unitarias son pruebas que se realizan sobre una funcionalidad concreta, es decir, sobre una parte del programa, con el objetivo de comprobar si funciona de forma correcta. Para el desarrollo de pruebas unitarias, encontramos el framework de Java, JUnit. Para intalarlo, basta con incorporar algunas librerías JAR al entorno de desarrollo.

Como resultado de las pruebas, se distinguen dos tipos de escenarios: la respuesta deseada y la respuesta real. Cuando estas no coinciden, será necesario hacer una revisión completa del código que se está probando.

Además, las pruebas unitarias deben cumplir las **características** del conocido como principio FIRST.

- Rápida ejecución
- Independencia respecto a otros test
- Se pueda repetir en el tiempo
- Cada test debe poder comprobar si es válido por sí mismo
- Es conveniente diseñar el test antes de empezar a implementar el código

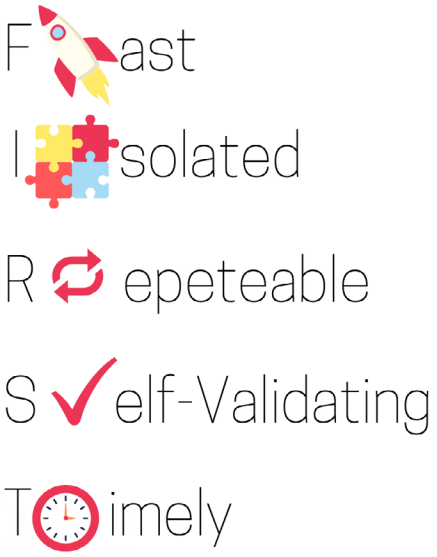

Fig. 4. Principio FIRST de pruebas unitarias.

Vídeo 1. "JUnit en Eclipse"
https://bit.ly/2XMWApd

3.1. JUnit en Eclipse

Para trabajar con JUnit desde Eclipse, lo primero que necesitamos es tener nuestro proyecto creado y añadir la librería JUnit desde Build Path. La manera más sencilla es haciendo clic con el botón derecho sobre el proyecto y seleccionando **Build Path, Add Libraries, JUnit**. Después, nos aparecerá una pantalla con las versiones de la librería disponibles, y seleccionamos **JUnit 5.**

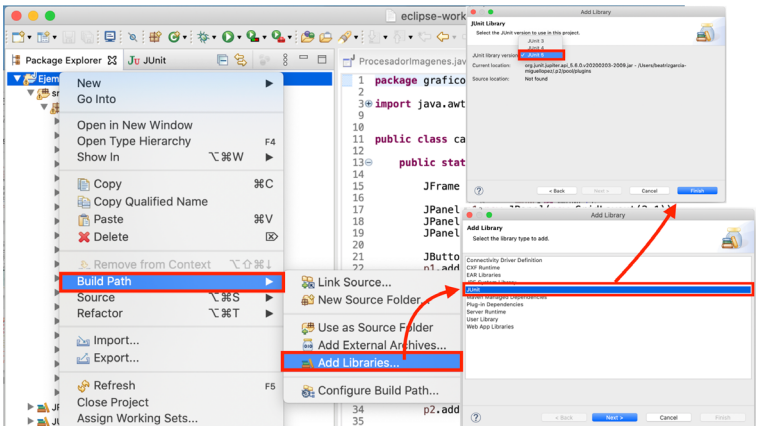

Fig. 5. Importación librería JUnit en Eclipse.

3.2. Clase prueba

Ahora crearemos nuestra clase de prueba, que se llamará como la clase que queremos probar, pero con la palabra 'test' delante. Por ejemplo, si la clase que queremos probar se llama calculadora.java, la clase de prueba se llamara **TestCalculadora.java.** Además, esta nueva clase debe extender de la clase TestCase e importar **junit.framework.TestCase.**

```
import junit.framework.TestCase;
public class TestCalculadora extends TestCase{...}
```

Código 1. Clase prueba.

3.3. Métodos de prueba

Los métodos de esta clase serán similares a los de la clase que se quiere probar, pero con la palabra 'test' delante, es decir, si el método original es sumar, el método de prueba será testSumar(). **Importante:** este método no devolverá nada, por lo que se declarará como public **void** TestSumar(). Dentro de cada método, llamaremos al constructor de nuestra clase de prueba y, para comprobar si el resultado obtenido coincide con el esperado, utilizaremos los métodos **assert.** Existen varios tipos de métodos assert, pero los más utilizados según el tipo de dato que queramos comprobar son: assertTrue, assertFalse, assertEquals y assertNull.

3.4. Comprobación de errores

Por último, comprobaremos los resultados compilando la clase prueba como: **Run As, JUnit Test.** En ese momento, aparecerá un panel llamado JUnit, en forma de árbol, que mostrará los resultados correctos en color verde y los errores en rojo.

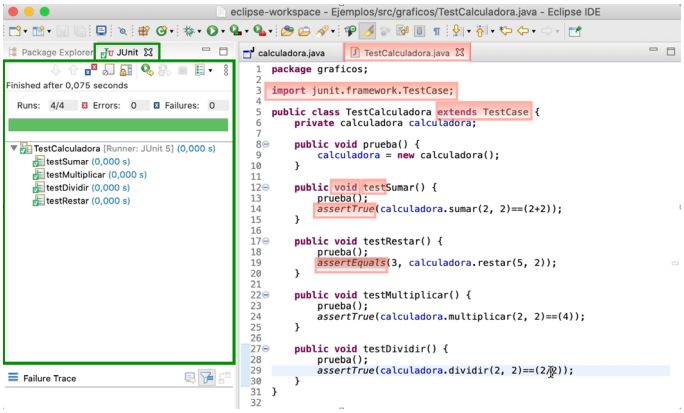

Fig. 6. Comprobación errores con JUnit.

/ 4. Eventos: clases

La programación basada en eventos es la clave de la iteración entre el usuario y una interfaz. Sin la existencia de esta, solo tendríamos textos, imágenes o cualquier otro elemento. Este tipo de programación podría dividirse en dos grandes bloques: la detección de los eventos y las acciones asociadas a su tratamiento. Podemos decir que la programación basadas en componentes y la basada en eventos se encuentran estrechamente ligadas, puesto que la finalidad de gran parte de los componentes descritos en el capítulo anterior es la recogida de información directa a través de la comunicación e interacción con el usuario. En función del origen del evento, es decir, de dónde se haya producido, diferenciamos entre eventos internos y externos:

- **Eventos internos.** Este tipo de eventos está producido por el propio sistema.
- **Eventos externos.** Los eventos externos son aquellos producidos por el usuario, habitualmente, a través del teclado o del puntero del ratón.

Los objetos que definen todos los eventos que se verán en este tema se basan en las siguientes clases, o en varias de estas, puesto que existen relación de herencia entre ellas.

Clase	Descripción
EventObject	Clase principal de la que derivan TODOS los eventos
MouseEvent	Eventos relativos a la acción del ratón sobre el componente
ComponentEvent	Eventos relacionados con el cambio de un componente, de tamaño, posición...
ContainerEvent	Evento producido al añadir o eliminar componente sobre un objeto de tipo Container
WindowsEvent	Este tipo de eventos se produce cuando una ventana ha sufrido algún tipo de variación, desde su apertura o cierre, hasta el cambio de tamaño
ActionEvent	Evento que se produce al detectarse la acción sobre un componente. Es uno de los más comunes, puesto que modela acciones tales como la pulsación sobre un botón o el check en un menú de selección.

Tabla 1. Clases de eventos.

/ 5. Eventos: componentes

Los componentes utilizados para el desarrollo de interfaces presentan, habitualmente, un tipo de evento asociado. No es lo mismo el tipo de detección asociado a un botón o a la pulsación de una tecla, que la forma de detección de la apertura o cierre de una ventana.

Fig. 7. Asociación de un evento a un componente.

En la siguiente tabla, se muestran los componentes más habituales y el tipo de evento asociado a estos.

Nombre componente	Nombre evento	Descripción del evento
JTextField	ActionEvent	Detecta la pulsación de la tecla Enter tras completar un campo de texto
JButton	ActionEvent	Detecta la pulsación sobre un componente de tipo botón
JComboBox	ActionEvent ItemEvent	Se detecta la selección de uno de los valores del menú
JCheckBox	ActionEvent ItemEvent	Se detecta el marcado de una de las celdas de selección
JTextComponent	TextEvent	Se produce un cambio en el texto
JScrollBar	AdjustmentEvent	Detecta el movimiento de la barra de desplazamiento (scroll)

Tabla 2. Nombre de componente y nombre de evento asociado.

/ 6. Listeners: KeyListener y ActionListener

Los listeners o escuchadores permiten detectar la ocurrencia de los eventos. Se podría decir que cuando estos se definen y activan, quedan a la espera («escuchando») de si ese produce un evento. Si este se produce, se ejecutan las acciones asociadas a tal ocurrencia. Todo evento requiere de un listener que controle su activación.

A continuación, se verán todos los tipos de listeners asociados al tipo de evento al que corresponden. Como se puede observar, un mismo tipo de escuchador puede estar presente en varios eventos y componentes diferentes, aunque, normalmente, presentan un comportamiento muy similar.

6.1. KeyListener

Este evento se detecta **al pulsar cualquier tecla**. Bajo este escuchador, se contemplan varios tipos de pulsaciones, cada uno de los cuales presentará un método de control propio. **Se implementan los eventos ActionEvent.**

Modo de pulsación	Definición
keyPressed	Se produce al pulsar la tecla
keyTyped	Se produce al pulsar y soltar la tecla
keyReleased	Se produce al soltar una tecla

Tabla 3. Eventos asociados a KeyListener.

6.2. ActionListener

Este evento detecta la pulsación sobre un componente. Está presente en varios tipos de elementos y es uno de los escuchadores más comunes.

La detección tiene lugar ante dos tipos de acciones: la pulsación sobre el componente con la tecla Enter, siempre que el foco esté sobre el elemento; o la pulsación con el puntero del ratón sobre el componente. **Estos componentes implementan los eventos de tipo ActionEvent.**

Componente	Descripción
JButton	Al hacer clic sobre el botón o pulsar la tecla Enter con el foco situado sobre el componente
JTextField	Al pulsar la tecla Enter con el foco situado sobre la caja de texto
JMenuItem	Al seleccionar alguna opción del componente menú
JList	Al hacer doble clic sobre uno de los elementos del componente lista

Tabla 4. Nombre de componente asociados a ActionListener.

/ 7. Listeners: MouseListener, FocusListener y MouseMotionListener

- **MouseListener**

Este evento se produce al hacer clic sobre algún componente. Es posible diferenciar entre distintos tipos de pulsaciones y asociar a cada una de ellas una acción diferente.

Estos componentes **implementan los eventos de tipo MouseEvent.**

Modo de pulsación	Definición
mouseClicked	Se produce al pulsar y soltar con el puntero del ratón sobre el componente
mouseExited	Se produce al salir de un componente utilizando el puntero del ratón
mousePressed	Se produce al presionar sobre el componente con el puntero
mouseReleased	Se produce al soltar el puntero del ratón
mouseEntered	Se produce al acceder a un componente utilizando el puntero del ratón

Tabla 5. Nombre de componentes asociados a MouseListener.

- **FocusListener**

Este evento se produce cuando un elemento está seleccionado o deja de estarlo, es decir, al tener el foco sobre el componente o dejar de tenerlo.

Para cualquiera de los casos, **se implementan objetos de la clase de eventos FocusEvent.**

- **MouseMotionListener**

Mientras que el caso anterior detecta la acciones con el puntero en cuanto a pulsar o soltar las teclas del ratón, este nuevo evento se produce ante la detección del movimiento del ratón.

Modo de acción	Definición
mouseMoved	Se produce al mover sobre un componente el puntero del ratón
mouseDragged	Se produce al arrastrar un elemento haciendo clic previamente sobre él

Tabla 6. Nombre de componentes asociados a MouseMotionLIstener.

Audio 1. "Diferencia entre eventos y escuchadores"

https://bit.ly/31IhMOf

/ 8. Eventos: métodos

Cada uno de los eventos detallados en los apartados anteriores utilizará un método para el tratamiento del mismo, es decir, tras enlazar al escuchador con la ocurrencia de un evento, será necesario ejecutar un método u otro en función del tipo de evento asociado.

En la siguiente tabla, se muestra la relación entre el Listener y el método propio de cada evento:

Listener	Métodos	
ActionListener	public void actionPerformed(ActionEvent e)	
KeyListener	keyPressed	public void keyPressed(KeyEvent e)
	keyTyped	public void keyTyped(KeyEvent e)
	keyRelease	public void keyReleased(KeyEvent e)
FocusListener	Obtención del foco	public void focusGained(FocusEvent e)
	Pérdida del foco	public void lostGained(FocusEvent e)
MouseListener	mouseClicked	public void mouseClicked(MouseEvent e)
	mouseExited	public void mouseExited(MouseEvent e)
	mousePressed	public void mousePressed(MouseEvent e)
	mouseReleased	public void mouseReleased(MouseEvent e)
	mouseEntered	public void mouseEntered(MouseEvent e)
MouseMotionListener	mouseMoved	public void mouseMoved(MouseEvent e)
	mouseDragged	public void mouseDragged(MouseEvent e)

Tabla 7. Relación entre Listener y los métodos de eventos.

/ 9. Asociación de acciones a eventos

Para asociar una acción o conjunto de acciones a un evento, aunque dependerá del evento y acción exacta, podemos definir, a continuación, una consecución común de pasos.

- **Creación del listener**

En primer lugar, se crea una nueva instancia del evento y este se vincula con el componente sobre el que va a actuar.

Para que se produzca esta detección, será necesario enlazar el evento con los escuchadores, elementos que se encuentren a la espera de que se produzca algún evento (Listeners). Estos elementos son interfaces que implementan un conjunto de métodos.

La implementación puede realizarse o bien añadiendo el evento directamente al componente y, posteriormente, codificando las acciones que se van a llevar a cabo (Sintaxis 1), o bien modelando primero todo el tratamiento del evento y, a continuación, asociándolo con el componente sobre el que actúa (Sintaxis 2).

Sintaxis 1:

```
nombreComponente.addtipoEventoListener(new tipoEventoListener()
{...}
```

Código 2. Sintaxis 1 creación de Listener.

Sintaxis 2:

```
tipoEventoListener nombreEvento = new tipoEventoListener (){...}
nombreComponente.addtipoEventoListener(nombreEvento);
```

Código 3. Sintaxis 2 creación de Listener.

El valor de tipoEventoListener se obtiene de las tablas del apartado 4, escogiéndolo en función del evento que se vaya a tratar.

- **Asociación de la acción al evento**

Cuando se activa y vincula un escuchador o listener a un componente y, por tanto, a la ocurrencia de un evento, los componentes no realizan un filtrado previo de los eventos para determinar si los manejan o no, sino que los reciben todos. A través de la asociación de la acción al evento se determinará si se maneja el evento o no.

A continuación, se implementa el método bajo el cual se desarrolla la acción, que se ejecutará tras la ocurrencia del evento, ya sea la pulsación sobre un botón, selección con el ratón, marcado en una lista de componentes o escritura a través del teclado, entre otros.

En el lenguaje de programación Java, cada evento está asociado a un objeto de la clase EventObject y, por lo tanto, a un método concreto. La estructura general para la definición de este método sigue el siguiente patrón:

```
public void métodoDeEvento(TipoEvento e){...}
```

Código 4. Sintaxis métodos de acciones.

Los métodos relativos a cada evento, prestando especial atención a las diferentes casuísticas que presentan algunos eventos, son los estudiados en el apartado 4.

9.1. Flujo completo de diseño

Por ejemplo, en el siguiente fragmento de código, se quiere implementar la detección de la pulsación sobre un botón. Cuando esto ocurra, se debe mostrar el mensaje «Hola Mundo» en la misma ventana en la que se encuentra el botón.

En primer lugar, se crea una nueva clase JFrame y se inserta un panel JPanel para ubicar encima el resto de elementos.

Se coloca una etiqueta y un botón, en la distribución que se desee, desde la vista de diseño. En la parte del código, de forma automática, se habrá generado:

```
//Se crea un nuevo botón
JButton btnNewButton = newJButton("Pulsa aquí");
//Se añade al JPanel
panel.add(btnNewButton);
//Se coloca una etiqueta también en Panel
JLabel label = new JLabel("...");
panel.add(label);
```

Código 5. Creación del botón y la etiqueta.

Y ahora, ¿cómo se crea el código relativo a la producción y detección de eventos para cada componente?

Desde la vista de diseño, hacemos doble clic sobre el botón, lo que nos lleva al código, donde ahora aparecen algunas líneas nuevas.

```
btnNewButton.addActionListener(new ActionListener({
```

Código 6. Escuchador del evento.

De forma automática, se implementa tanto el escuchador vinculado al botón (ActionListener) como el método dentro del cual se desarrollan las acciones desencadenadas por el evento (actionPerformed), que recibe por parámetro un ObjectEvent de tipo ActionEvent.

Finalmente, solo quedará colocar el código que envía a la etiqueta de texto creada en el inicio el mensaje «Hola Mundo» y la muestra por pantalla.

```
public void actionPerformed(ActionEvent e) {
                lblNewLabel.setText("Hola Mundo");
    }
});
```

Código 7. Acción asociada al evento.

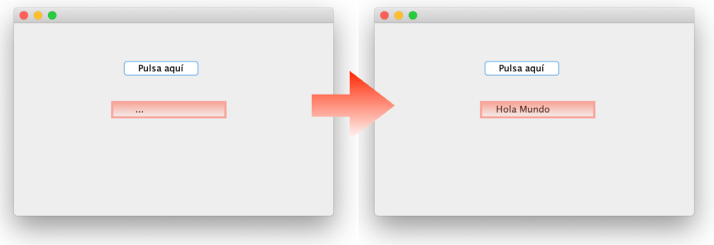

Fig. 8. Resultado de la detección de la pulsación de un botón.

Vídeo 2. "Añadir eventos desde Design en Eclipse"
https://bit.ly/2F65pnm

/ 10. Caso práctico 1: "Configuración del ratón"

Planteamiento: En este ejercicio, se implementará una interfaz para configurar la velocidad de desplazamiento del ratón; también se configurará cuál de los dos botones será el principal. Para ello, se utilizarán los componentes gráficos:

- Un JSpinner, para poder seleccionar los valores 0, 25 y 50.
- Un JComboBox, con la opción izquierdo y derecho.
- Una barra de progreso que muestre la velocidad.
- Un ratón que indique el botón seleccionado.

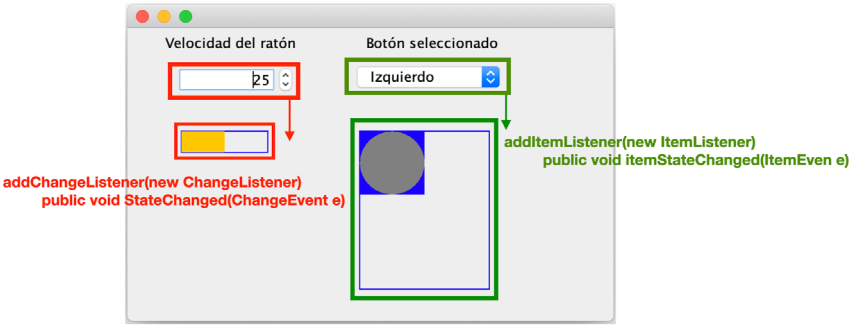

Fig. 9. Diseño a implementar Caso Práctico 1.

Nudo: Para implementar los eventos escuchadores asociados al JSpinner y al JComboBox. será necesario utilizar:

- En el caso del JSpinner. para capturar la velocidad establecida por el usuario. se utilizará el evento de cambio de estado:

```
spinner1.addChangeListener(new ChangeListener() {
public void stateChanged(ChangeEvent e) {
```

- Para el JComboBox. se utilizará el evento de ítem seleccionado:

```
combo1.addItemListener(new ItemListener() {
    public void itemStateChanged(ItemEvent e) {
```

- Cuando se detecte el cambio de estado del componente JSpinner, se llamará al método de 'pintar' para dibujar la barra de progreso con el color y longitud apropiados. Si la velocidad seleccionada es 0, la longitud de la barra será de 20px y el color será el rojo. Si es 25, será de color naranja y longitud 40px. Por último, en el caso de ser velocidad 50, la barra será de longitud 60px y de color verde.

```
if(spinner1.getValue().equals(0)) {
        g.setColor(Color.red);
        g.fillRect(50,120, 20, 20);
```

- Con la detección del evento del botón seleccionado en el JComboBox, se llamará también al método 'pintar', que dibujará el botón del ratón seleccionado. En el caso de que sea el botón izquierdo, dibujará desde el píxel 215 del eje 'x', que es donde empieza el rectángulo del botón. Si es el botón derecho, se dibujará ese mismo círculo desde la coordenada 'x' de 275 px.

```
if(combo1.getSelectedItem().toString().equals("Izquierdo")) {
        g.fillRect(215,120, 60, 60);
        g.setColor(Color.gray);
        g.fillOval(215,120,60,60);
}
else {
        g.fillRect(275,120, 60, 60);
        g.setColor(Color.gray);
        g.fillOval(275,120,60,60);
}
```

Desenlace:

El código completo de la clase raton.java se encuentra en el **Anexo del tema.**

/ 11. Caso práctico 2: "Teclado de piano"

Planteamiento: Se desea desarrollar la interfaz gráfica del teclado de un piano que permita tocar la escala musical. Para ello, se utilizarán siete botones, a modo de teclas, para tocar las notas musicales y los eventos escuchadores adecuados.

Nudo: Para realizar esta propuesta, los pasos a seguir son:

- Importar la librería JFugue.jar, que nos permitirá usar las notas.
- Instanciar un objeto de tipo Player.
- Crear los siete botones de tipo JButton.
- Escuchar el evento actionPerformed.
- Cuando suceda ese evento, llamar al método play.

Por último, repetimos el proceso con los seis botones restantes, asignándole a cada uno de ellos la nota inglesa correspondiente (do=C, re=D, mi=E, fa=F, sol=G, la=A, si=B).

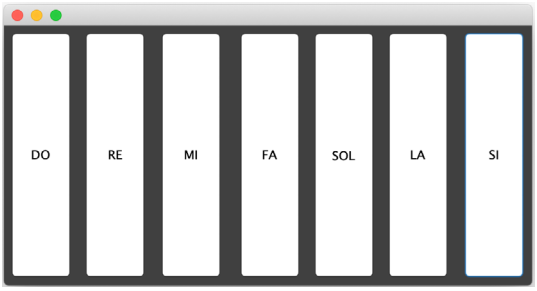

Fig. 10. Diseño a implementar Caso Práctico 2.

Desenlace:

```
import org.jfugue.player.Player;
public class interfazTeclado extends JFrame {
 private JPanel contentPane;
 public Player player=new Player();
 public static void main(String[] args) {
  EventQueue.invokeLater(new Runnable() {
    public void run() {
     try {
interfazTeclado frame = new interfazTeclado();
 frame.setVisible(true);
     } catch (Exception e) {
       e.printStackTrace();
     }
   }
  });
 }
  public interfazTeclado() {
   setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 549, 300);
    contentPane = new JPanel();
contentPane.setBackground(Color.DARK_GRAY);
    contentPane.setBorder(new EmptyBorder(5,5,5,5));
    setContentPane(contentPane);
    contentPane.setLayout(null);
    JButton A = new JButton("DO");
    A.addActionListener(new ActionListener() {
     public void actionPerformed(ActionEvent e) {
              player.play("C");
     }
    });
```

Código 8. Código desenlace Caso Práctico 2.

/ 12. Resumen y resolución del caso práctico de la unidad

A lo largo de este tema, hemos trabajado la creación de eventos, elementos clave en la implementación y desarrollo de interfaces visuales, sobre todo para la creación de componentes integrados en estas interfaces, que proporcionan a los usuarios una experiencia de interacción.

El proceso de creación y asociación de eventos se basa en los llamados listeners (escuchadores), que se enlazan a los componentes «escuchando» si se producen cambios asociados a la integración entre el usuario y la interfaz, y si es así, se ejecutan las acciones implementadas.

Como hemos visto en las tablas de este tema, debemos diferenciar entre los escuchadores, los métodos que tratan las acciones y los eventos. Escogiendo la combinación adecuada, podremos desarrollar prácticamente cualquier tipo de interfaz.

Por último, en este tema también hemos visto la generación y aplicación de pruebas unitarias, imprescindibles para garantizar la implantación correcta de cualquier tipo de desarrollo.

Resolución del caso práctico de la unidad.

Ahora que tenemos un diseño bastante avanzado de la base de datos para la tienda de nuestro amigo, nos plantea una nueva cuestión en con la que podemos ayudarle.

Le gustaría saber cómo gestionar la información de la base de datos, modificando, añadiendo y eliminando elementos sobre las tablas que la componen (Clientes, Almacén, Empleados...).

- **¿Cómo podríamos ayudarle a realizarlo?**

En el ámbito de la programación, los eventos están asociados a las acciones que realizan los usuarios para interactuar con la interfaz o bien a los sucesos que se quieren controlar para dar respuesta a ellos.

Existen múltiples tipos de eventos. Además de hacer clic con el ratón, otros eventos muy utilizados son el movimiento del ratón y escribir con el teclado.

La implementación de interfaces con Java tiene asociada una funcionalidad que permite detectar diferentes tipos de eventos. Estos programas pueden detectar desde el tipo de evento (clic, desplazamiento del ratón, pulsación del teclado...) hasta el componente con el que se ha interactuado (botón, menú desplegable, completar el campo de un formulario...).

/ 13. Bibliografía

Fernández, A., García-Miguel, B. y García-Miguel, D. (2020). Desarrollo de Interfaces. (1.a ed.). Madrid: Síntesis.