5.2. Eventos y asociación de acciones a eventos

Índice

Objetivos

En esta lección perseguimos los siguientes objetivos:

- Comprender el concepto de evento.
- Distinguir entre fuente y auditor de eventos.
- Crear aplicaciones utilizando mecanismos de gestión de eventos.

Eventos, fuentes y auditores de eventos

Eventos

Un evento es algo así como un suceso relevante ocurrido con un objeto y al que queremos responder.

Los objetos **se disparan** cuando ocurre algo que merece atención especial y deseamos que el resto de la aplicación escuche y pueda responder en consecuencia.

El objeto que dispara el evento es el **objeto fuente.**

Los objetos que escuchan el evento son los **objetos auditores.**

Ejemplos:

- Una clase *Cuenta*, que representa una cuenta bancaria, podría disparar un evento cuando la cuenta se queda en números rojos. Otra clase *Cliente*, que construye objetos de tipo *Cuenta*, puede auditar o escuchar dichos eventos para responder en consecuencia.
- En una clase que representa una ventana en un programa de escritorio, sus objetos podrán disparar un evento cuando el usuario hace clic en un botón. Otra clase *Cliente* que construye un objeto para añadir dicha ventana a su aplicación, podrá responder a dicho evento para asociar alguna acción al hecho de hacer clic en el botón.

Tipos de eventos. Mecanismo de gestión de eventos

Mecanismo de gestión de eventos

Antes de nada, para que puedas comprender mejor los mecanismos de gestión de eventos, es conveniente que crees un proyecto nuevo en Eclipse con una clase que representará una ventana de una aplicación de escritorio.

No es objetivo de este curso profundizar en las aplicaciones de escritorio, solo veremos lo necesario para poder aplicarlo a la gestión de eventos.

Para nuestro ejemplo utilizaremos la **librería de clases *javax.swing***.

¡Manos a la obra!

1. Crea un proyecto en eclipse llamado ***ProyectoEventos***.

2. Crea una **clase** llamada ***Ventana*** con el siguiente código:

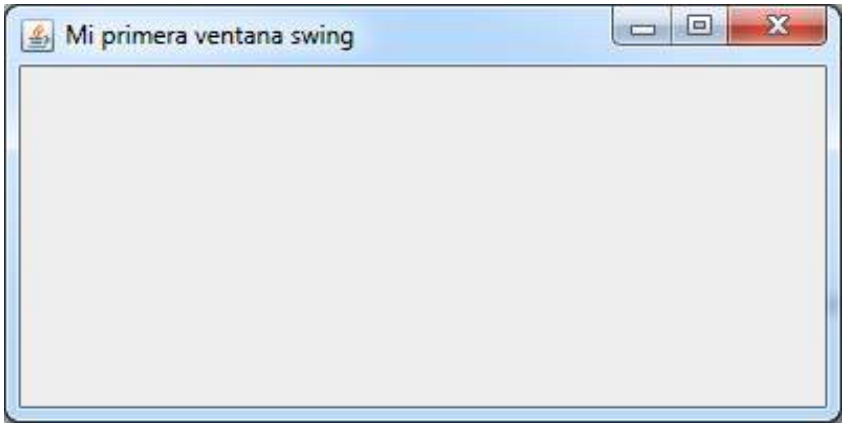
```
import javax.swing.*;

public class Ventana extends JFrame {
	private static final long serialVersionUID = 1L;

	Ventana() {
		super ("Mi primera ventana swing");
	}

	public void iniciar() {
		this.setLocation(200,200);
		this.setSize(400,200);
		this.setVisible(true);
	}
}
```

3. Crea una **clase** llamada ***Principal*** con el siguiente código:

```
public class Principal {
	public static void main(String args[]) {
		Ventana v = new Ventana();
		v.iniciar();
	}
}
```

4. Ejecuta desde la clase *Principal*.

Como resultado te aparecerá una ventana como esta:



Parece magia, ¿verdad?

Aunque no sepas nada sobre la librería ***javax.swing***, sí sabes ya mucho sobre programación orientada a objetos en Java y seguro que no te resulta complicado.

En la clase *Principal* no estamos haciendo nada que no hayamos hecho ya. Utilizamos el método *main* para construir un objeto de la clase *Ventana* y llamamos a un método, el método *iniciar()*.

La clase *Ventana* extiende o hereda de la clase ***JFrame***.

```
public class Ventana extends JFrame {

}
```

Y de *JFrame* ha heredado el aspecto de ventana, no sabemos cómo ni nos interesa. Lo importante es que podemos construir una aplicación con ventanas.

En el constructor de *Ventana* tenemos que pasar un argumento al constructor de *JFrame*. *JFrame* utiliza este argumento para poner título a la ventana.

```
Ventana() {
      super ("Mi primera ventana swing");
}
```

Con el método *iniciar()* configuramos detalles de la ventana, en este caso la posición, alto, ancho y visibilidad.

¿Pero para qué queremos una ventana vacía? Para nada, hay que añadirle elementos de tipo interfaz de usuario, botones, cajas de texto, listas desplegables, menús, etc.

Puesto que nuestra clase *Ventana* tiene un método *iniciar()* que la configura, será aquí donde podamos añadir elementos que antes habrá que definir.

Modifica la clase *Ventana* de la siguiente manera:

```
public class Ventana extends JFrame {
	private static final long serialVersionUID = 1L;
	private JPanel p=new JPanel();
	private JButton botonAzul = new JButton("Azul");
	private JButton botonVerde = new JButton("Verde");
	private JButton botonRojo = new JButton("Rojo");

	Ventana() {
		super ("Mi primera ventana swing");
	}

	public void iniciar() {
		this.setLocation(200,200);
		this.setSize(400,200);
		this.setVisible(true);
		this.add(p); // Añadimos panel a la ventana.
		p.add(botonRojo);
		p.add(botonAzul);
		p.add(botonVerde);
	}
}
```

Ahora tienes tres botones, pero al hacer clic en ellos no pasa nada. Aquí es donde entran en juego los mecanismos de gestión de eventos.

Los objetos de la clase ***JButton*** **"disparan" un evento cuando el usuario hace clic en el botón**, pero la clase *Ventana* (cliente o usuaria de la clase *JButton*) no está "escuchando o auditando" dicho evento.

El modelo de eventos en Java se denomina gestión de eventos delegado. Se debe registrar específicamente si se quiere gestionar un evento, como puede ser hacer clic sobre un botón. De esta forma, Java mejora el rendimiento de las aplicaciones. Si **solo "se disparan" los eventos que necesitamos** estaremos ahorrando recursos necesarios.

Los eventos se registran implementando una interfaz de *Listener* de eventos que necesitemos.

Gestionar eventos relacionados con las ventanas

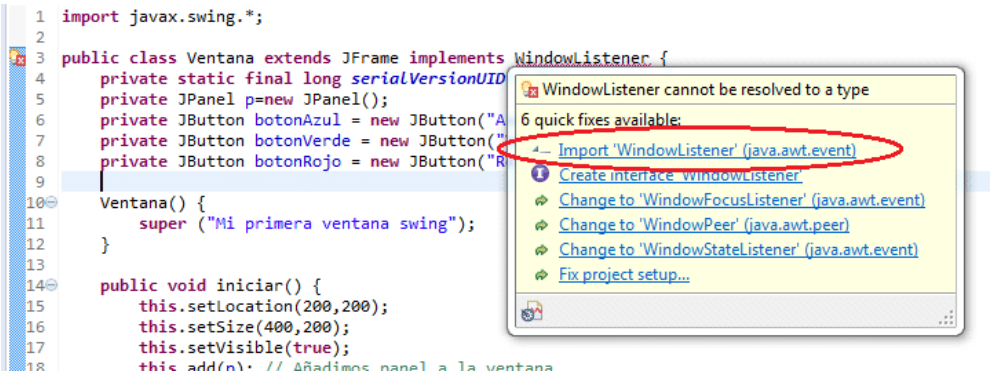
Vamos a comenzar por implementar la interfaz de *Listener* asociada con la gestión de eventos de la ventana (cargar ventana, cerrar ventana, iconizar, activar, etc.). En otro apartado pasaremos a gestionar el clic de los botones.

Tu clase *Ventana*, además de extender de JFrame, debe implementar la ***interface WindowListener***.

```
public class Ventana extends JFrame implements WindowListener {
}
```

Ahora mismo tendrás un error de compilación y la palabra *WindowListener* está subrayada en rojo. Es porque tienes que importar la librería de clases que contiene la clase *WindowListener*.

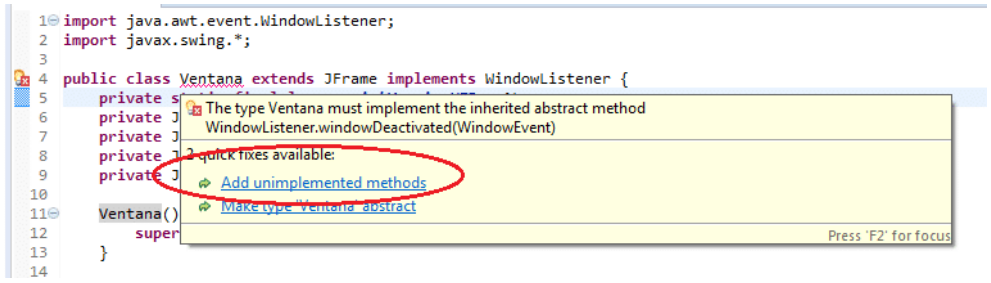
Si sitúas el puntero de ratón sobre la palabra subrayada *WindowListener* obtendrás una pequeña ventana flotante que te da pistas sobre la solución al error de compilación. En concreto la solución está en importar la librería ***java.awt.event*** que contiene todas las clases relacionadas con la gestión de eventos en ventanas.



Eclipse te ha añadido el *import* que faltaba, has solucionado un error, pero te ha aparecido otro, **ahora está subrayada en rojo la palabra *Ventana***.

Recuerda que **una clase que implementa una interfaz está obligada a implementar los métodos abstractos de dicha interfaz**. Por esa razón ahora tienes un error de compilación en la clase *Ventana*.

De nuevo puedes situar el puntero de ratón, pero ahora sobre la palabra *Ventana* para ver qué solución te da Eclipse.



Ahora la solución está en implementar los métodos de la interfaz *WindowListener* (**Add unimpemented methods**).

Si lo has hecho ya, de manera automática se han generado los siguientes métodos:

```
	@Override
	public void windowActivated(WindowEvent arg0) {

	}

	@Override
	public void windowClosed(WindowEvent arg0) {

	}

	@Override
	public void windowClosing(WindowEvent arg0) {

	}

	@Override
	public void windowDeactivated(WindowEvent arg0) {

	}

	@Override
	public void windowDeiconified(WindowEvent arg0) {

	}

	@Override
	public void windowIconified(WindowEvent arg0) {

	}

	@Override
	public void windowOpened(WindowEvent arg0) {

	}
```

Ahora puedes añadir algo de código a estos métodos manejadores de evento, solo como prueba:

```
@Override
public void windowActivated(WindowEvent arg0) {
 System.out.println("Se ha activado la ventana");
}

@Override
public void windowClosed(WindowEvent arg0) {
 System.out.println("Se ha terminado de cerrar la ventana");
}

@Override
public void windowClosing(WindowEvent arg0) {
 System.out.println("Se está comenzando a cerrar la ventana");
 }
@Override
public void windowDeactivated(WindowEvent arg0) {
 System.out.println("Se está desactivando la ventana");
}
```

```
@Override
public void windowDeiconified(WindowEvent arg0) {
 System.out.println("Se está restaurando una ventana que estaba iconizada");
}

@Override
public void windowIconified(WindowEvent arg0) {
 System.out.println("Se está iconizando la ventana");
}

@Override
public void windowOpened(WindowEvent arg0) {
 System.out.println("Se está abriendo la ventana");
}
```

Si ejecutas de nuevo, deberías estar viendo mensajes en la consola de Eclipse, pero,

¡Oh! ¡No sale ningún mensaje! ¿No funcionan los eventos?

No te preocupes, es que nos falta un paso todavía.

Recuerda que el modelo de eventos en Java se denomina gestión de eventos delegado. Se debe registrar específicamente si se quiere gestionar un evento. Justo eso es lo que nos falta. No hemos especificado en ningún momento que queremos gestionar los eventos relacionados con la ventana. Y debemos hacerlo así:

```
this.addWindowListener(this);
```

Para cada objeto cuyos eventos deseamos gestionar, hay que utilizar un método que siempre comenzará por *add*.

En este caso, el objeto es la propia ventana, es decir, el objeto actual *(this)*.

Ahora nuestra clase ventana está así:

```
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import javax.swing.*;

public class Ventana extends JFrame implements WindowListener {
	private static final long serialVersionUID = 1L;
	private JPanel p=new JPanel();
	private JButton botonAzul = new JButton("Azul");
	private JButton botonVerde = new JButton("Verde");
	private JButton botonRojo = new JButton("Rojo");

	Ventana() {
		super ("Mi primera ventana swing");
	}

	public void iniciar() {
		this.setLocation(200,200);
		this.setSize(400,200);
		this.setVisible(true);
		this.add(p); // Añadimos panel a la ventana.
```

```
		p.add(botonRojo);
		p.add(botonAzul);
		p.add(botonVerde);
		this.addWindowListener(this);
	}

	@Override
	public void windowActivated(WindowEvent arg0) {
		System.out.println("Se ha activado la ventana");
	}

	@Override
	public void windowClosed(WindowEvent arg0) {
		System.out.println("Se ha terminado de cerrar la ventana");
	}

	@Override
	public void windowClosing(WindowEvent arg0) {
		System.out.println("Se está comenzando a cerrar la ventana");
	}

	@Override
	public void windowDeactivated(WindowEvent arg0) {
		System.out.println("Se está desactivando la ventana");
	}

	@Override
	public void windowDeiconified(WindowEvent arg0) {
		System.out.println("Se está restaurando una ventana
que estaba iconizada");
	}

	@Override
	public void windowIconified(WindowEvent arg0) {
		System.out.println("Se está iconizando la ventana");
	}

	@Override
	public void windowOpened(WindowEvent arg0) {
		System.out.println("Se está abriendo la ventana");
	}
}
```

Ahora, si ejecutas el proyecto, irás viendo un montón de mensajes en la consola de Eclipse mientras vas interactuando con la ventana.

Resumiendo, para gestionar los eventos relacionados con las acciones de la ventana, hemos tenido que seguir estos pasos:

- Implementar la interfaz de *listener WindowListener* añadiendo "***implements WindowListener***" a la cabecera de la clase *Ventana*.
- **Implementar los métodos abstractos** de *WindowListener* (***windowActivated***, ***windowClosed***, ***windowClosing***, **etc**.).
- **Registrar** específicamente que deseamos gestionar los eventos relacionados con la ventana añadiendo la instrucción "***this.addWindowListener(this);***"

Responder al evento clic de los botones

Ahora vamos a responder al evento clic de los botones. Para conseguirlo vamos de nuevo a realizar tres pasos similares a los anteriores con los eventos de ventana. Veamos los pasos.

1. Implementa la interfaz de *listener* ***ActionListener***

La interfaz ***ActionListener*** está pensada para gestionar los eventos de acción, tal como hacer clic sobre un botón.

```
public class Ventana extends JFrame implements WindowListener,ActionListener{
}
```

2. Implementa los **métodos abstractos**

Implementa el único método abstracto de *ActionListener* (***actionPerformed***) añadiendo las acciones que quieres que ocurran al hacer clic en cada botón.

```
@Override
      public void actionPerformed(ActionEvent e) {
            if (e.getSource() == this.botonAzul)
                  this.p.setBackground(Color.BLUE);
            else if (e.getSource() == this.botonRojo)
                  this.p.setBackground(Color.RED);
            else
                  this.p.setBackground(Color.GREEN);
      }
```

3. Indica los eventos que quieres gestionar

Registra específicamente que deseas gestionar los eventos relacionados con el clic de los botones. Para ello ejecuta el método ***addActionListener()*** sobre los objetos deseados. El método *iniciar()* quedará así:

```
public void iniciar() {
            this.setLocation(200,200);
            this.setSize(400,200);
            this.setVisible(true);
            this.add(p); // Añadimos panel a la ventana.
            p.add(botonRojo);
            p.add(botonAzul);
            p.add(botonVerde);
            this.addWindowListener(this);
            this.botonAzul.addActionListener(this);
            this.botonRojo.addActionListener(this);
            this.botonVerde.addActionListener(this);
      }
```

Los objetos Event

A estas alturas ya habrás comprobado que los métodos manejadores de eventos como *actionPerformed()*, *windowIconified()*, *windowActivated()*, etc. llevan un parámetro. Este parámetro trae información relevante sobre el evento ocurrido.

Vamos a analizar de nuevo el código del método sobrescrito ***actionPerformed()*** en la clase *Ventana*.

```
@Override
      public void actionPerformed(ActionEvent e) {
            if (e.getSource() == this.botonAzul)
                  this.p.setBackground(Color.BLUE);
            else if (e.getSource() == this.botonRojo)
                  this.p.setBackground(Color.RED);
            else
                  this.p.setBackground(Color.GREEN);
      }
```

El **objeto *e* de tipo *ActionEvent*** es un argumento que trae información sobre el evento ocurrido y quién lo provocó.

Concretamente, estamos utilizando el método ***getSource()***, que nos devuelve el objeto que "disparó" el evento; para nuestro ejemplo solo puede ser uno de estos tres objetos: *botonAzul*, *botonRojo* o *botonVerde*.

Librerías de clases asociadas

Librerías asociadas a la gestión de eventos

Las clases implicadas en la gestión de eventos con ventanas se encuentran encerradas en el paquete ***java.awt.event***.

Recuerda que para registrar eventos en una clase primero hay que implementar la interfaz de *Listener* correspondiente. Estas son las interfaces de *Listener* disponibles:

ActionListener
Para registrar eventos de acción, como hacer clic sobre los botones.

AdjustementListener
Para gestionar eventos en los que un componente es escondido, movido, redimensionado o mostrado.

ContainerListener
Para registrar eventos relacionados con que un componente coge o pierde el foco.

ItemListener
Para gestionar eventos relacionados con el cambio de elemento seleccionado en una lista.

KeyListener
Para gestionar los eventos relacionados con el teclado.

MouseListener
Para gestionar los eventos relacionados con el ratón.

MouseMotionListener
Para gestionar los eventos relacionados con el movimiento del ratón.

TextListener
Para gestionar los eventos de cambio de valor de texto.

WindowListener
Para gestionar los casos en que una ventana está activada, desactivada, con o sin forma de icono, abierta, cerrada o se sale de ella.

Todas estas interfaces, heredan de la *interface* genérica ***EventListener***, situada en el paquete ***java.util***.

Cualquier aplicación puede crear sus propios manejadores de eventos creando interfaces personalizadas que hereden de *java.util.EventListener*.

Recuerda que cada *Listener* es una interfaz, y las clases que la implementan deben también implementar sus métodos abstractos. Cada uno de estos métodos tiene un parámetro que recibe información sobre el evento. Recuerda la estructura para el caso de ***ActionListener***:

```
public class Ventana extends JFrame implement ActionListener {

	// Método sobreescrito de ActionListener
	@Override
	public void actionPerformed(ActionEvent e) {
		// ActionEvent e es el parámetro que recibe con información
		// sobre el evento.
	}
}
```

Los parámetros que reciben los métodos gestores de eventos en las aplicaciones con ventanas serán uno de los siguientes:

ActionEvent
Hacer clic en un botón, elemento de lista u otro componente.

AdjustementEvent
Movimientos en la barra de desplazamiento.

ComponentEvent
Un componente es escondido, movido, ocultado o mostrado.

FocusEvent
Un componente coge o pierde el foco (cursor).

ItemEvent
Hacer clic en un elemento de la lista o en una casilla de verificación de un grupo.

KeyEvent
Acciones con el teclado.

MouseEvent
Acciones relacionadas con el ratón.

TextEvent
Cambios en cuadros de texto.

WindowListener
Acciones relacionadas con la ventana.

Todas estas clases son derivadas de la clase *EventObject* situada en el paquete *java.util*.

Cualquier aplicación puede crear sus propias clases gestoras de eventos personalizadas extendiendo de *java.util.EventObject*.

Despedida

Resumen

Has finalizado esta lección, veamos los puntos más importantes que hemos tratado.

La gestión de eventos en una clase Java consta de estos tres pasos:

1. Implementar la interfaz de *Listener* que necesitemos.

```
public class Ventana extends JFrame implements ActionListener  {
}
```

2. Implementar los métodos abstractos de la interfaz elegida.

```
public class Ventana extends JFrame implements ActionListener  {
      ….

      @Override
      public void actionPerformed(ActionEvent e) {
            // Acciones
      }
}
```

3. Registrar específicamente los eventos que queremos controlar en cada componente.

```
public void iniciar() {
            …..

            this.botonAzul.addActionListener(this);
            this.botonRojo.addActionListener(this);
            this.botonVerde.addActionListener(this);
      }
```