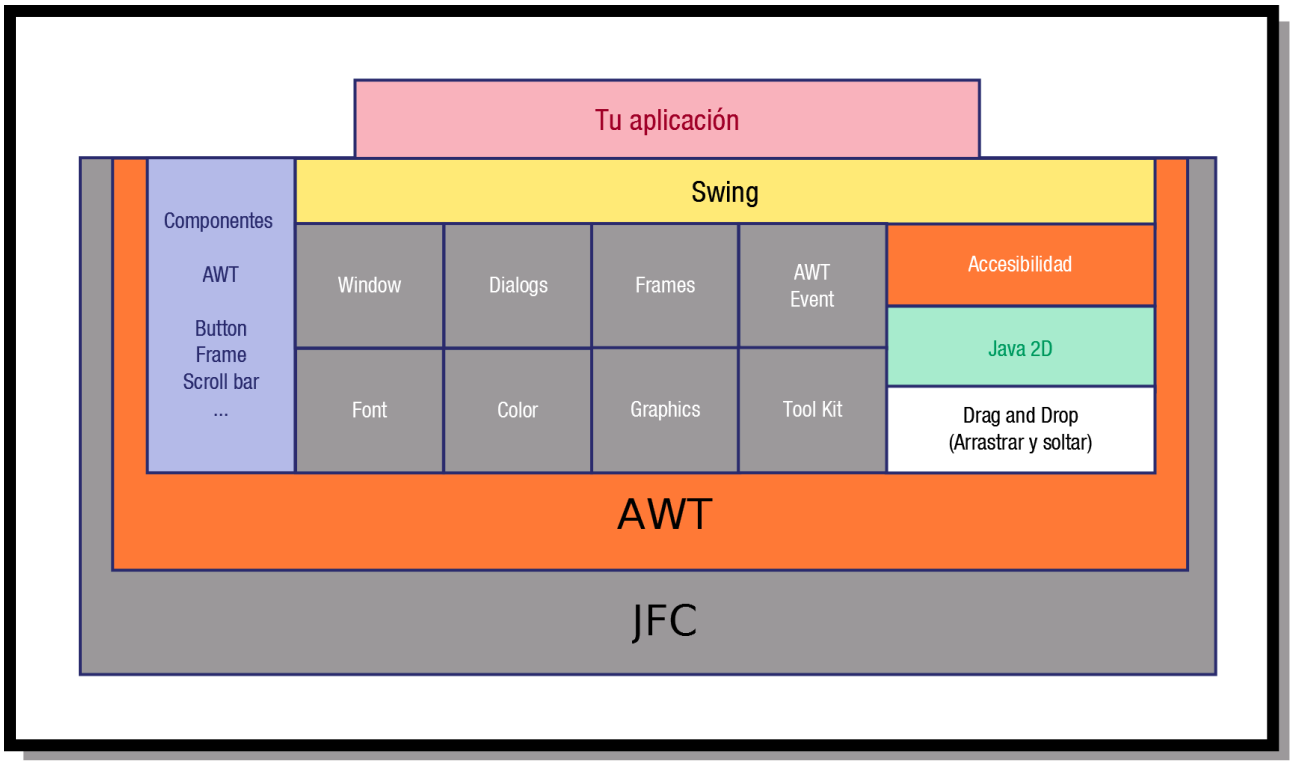
¡¡NOTA ACLARATORIA!!

Estos ficheros son un resumen literal del contenido de las unidades de trabajo expuestas en la plataforma FEDAN de educación semipresencial de la Junta de Andalucía. El motivo de sintetizarlos y subirlos a mi perfil de GitHub es conservarlos para que en un futuro pueda disponer de este contenido y consultarlo con fines didácticos.

TEMA 9

INTERFACES GRÁFICAS DE USUARIO (GUI)

En ocasiones verás otras definiciones de interfaz, como la que define una interfaz como un dispositivo que permite comunicar dos sistemas que no hablan el mismo lenguaje. También se emplea el término interfaz para definir el juego de conexiones y dispositivos que hacen posible la comunicación entre dos sistemas.



LIBRERÍAS JAVA PARA EL DESARROLLO DE INTERFACES GRÁFICAS DE USUARIO

El API de Java proporciona librerías de clases para el desarrollo de interfaces gráficas de usuario. Esas librerías se engloban bajo los nombres de **AWT** y **Swing**, que a su vez forman parte de las **Java Foundation Classes o JFC.**

Los elementos que componen las JFC son:

- **Componentes Swing**: encontramos componentes tales como botones, cuadros de texto, ventanas o elementos de menú.
- **Soporte de diferentes aspectos y comportamientos** (**Look and Feel**): permite la elección de diferentes apariencias de entorno. Por ejemplo, el mismo programa puede adquirir un aspecto **Metal** Java (multiplataforma, igual en cualquier entorno), **Motif** (el aspecto estándar para entornos Unix) o **Windows** (para entornos Windows). De esta forma, el aspecto de la aplicación puede ser independiente de la plataforma, lo cual está muy bien para un lenguaje

que lleva la seña de identidad de multiplataforma, y se puede elegir el que se desee en cada momento.

- **Interfaz de programación Java** : permite incorporar gráficos en dos dimensiones, texto e imágenes de alta calidad.
- **Soporte de arrastrar y soltar** (Drag and Drop) entre aplicaciones Java y aplicaciones nativas. Es decir, se implementa un portapapeles. Llamamos aplicaciones nativas a las que están desarrolladas en un entorno y una plataforma concretos (por ejemplo Windows o Unix), y sólo funcionarán en esa plataforma.
- **Soporte de impresión.**
- **Soporte sonido:** captura, reproducción y procesamiento de datos de audio y MIDI
- **Soporte de dispositivos de entrada distintos del teclado**, para japonés, chino, etc.
- **Soporte de funciones de Accesibilidad, para crear interfaces para discapacitados:** permite el uso de tecnologías como los lectores de pantallas o las pantallas Braille adaptadas a las personas discapacitadas.

AWT(Abstract Window Toolkit)

Las clases AWT se desarrollaron usando código nativo (o sea, código asociado a una plataforma concreta), así que para poder conservar la portabilidad era necesario restringir la funcionalidad a los mínimos comunes a todas las plataformas donde se pretendía usar AWT. Como consecuencia, AWT es una librería con una funcionalidad muy pobre, **adecuada para interfaces gráficas sencillas**, pero no para proyectos más complejos.

La estructura básica de la librería gira en torno a **componentes** y **contenedores**. Los contenedores contienen componentes y son componentes a su vez, de forma que los eventos pueden tratarse tanto en contenedores como en componentes.

Más tarde, con **Java 2** surgió una librería más robusta, versátil y flexible: **Swing**. Pero AWT sigue siendo imprescindible, ya que todos **los componentes Swing se construyen haciendo uso de clases de AWT**. De hecho, todos los componentes Swing, como por ejemplo `JButton`, derivan de la clase `JComponent`, que a su vez deriva de la clase AWT `Container`.

Las clases asociadas a cada uno de los **componentes** AWT se encuentran en el paquete `java.awt`. Las clases relacionadas con el manejo de **eventos** en AWT están en el paquete `java.awt.event`.

AWT fue la primera forma de construir las ventanas en Java, pero:

- Limitaba la portabilidad.
- Restringía la funcionalidad.
- Requería demasiados recursos.

SWING

Swing es una librería de Java para la generación del GUI en aplicaciones.

Swing se apoya sobre AWT y añade `JComponents`. La arquitectura de los componentes de Swing facilita la personalización de apariencia y comportamiento, si lo comparamos con los componentes AWT.

Por cada componente AWT (excepto `Canvas`) existe un componente Swing equivalente, **cuyo nombre empieza por J**, que permite más funcionalidad siendo menos pesado. Así, por ejemplo, para el componente AWT `Button` existe el equivalente Swing `JButton`, que permite como funcionalidad adicional la de crear botones con distintas formas (rectangulares, circulares, etc), incluir imágenes en el botón, tener distintas representaciones para un mismo botón según esté seleccionado, o bajo el cursor, etc.

La razón por la que no existe `JCanvas` es que los paneles de la clase `JPanel` ya soportan todo lo que el componente `Canvas` de AWT soportaba. No se consideró necesario añadir un componente Swing `JCanvas` por separado.

Algunas otras características más de Swing son:

- Es **independiente de la arquitectura** (metodología no nativa propia de Java).
- Proporciona **todo lo necesario para la creación de entornos gráficos**, tales como diseño de menús, botones, cuadros de texto, manipulación de eventos, etc**.**
- **Los componentes Swing no necesitan una ventana propia del sistema operativo cada uno**, sino que son visualizados dentro de la ventana que los contiene mediante métodos gráficos, por lo que requieren bastantes menos recursos.
- **Las clases Swing están completamente escritas en Java**, **con lo que la portabilidad es total**, a la vez que no hay obligación de restringir la funcionalidad a los mínimos comunes de todas las plataformas.
- Las clases Swing aportan una considerable gama de funciones que haciendo uso de la funcionalidad básica propia de AWT **aumentan las posibilidades de diseño** de interfaces gráficas.
- Debido a sus características, los componentes **AWT** se llaman componentes **“de peso pesado”** por la gran cantidad de recursos del sistema que usan, y los componentes **Swing** se llaman componentes **“de peso ligero”** por no necesitar su propia ventana del sistema operativo y por tanto consumir muchos menos recursos.
- Aunque todos los componentes Swing derivan de componentes AWT y de hecho **se pueden mezclar en una misma aplicación componentes de ambos tipos**, se desaconseja hacerlo. **Es preferible desarrollar aplicaciones enteramente Swing**, que requieren menos recursos y son más portables.

Los pasos para crear y ejecutar aplicaciones, se pueden resumir en:

- Crear un proyecto.
- Construir la interfaz con el usuario.
- Añadir funcionalidad, en base a la lógica de negocio que se requiera.
- Ejecutar el programa.

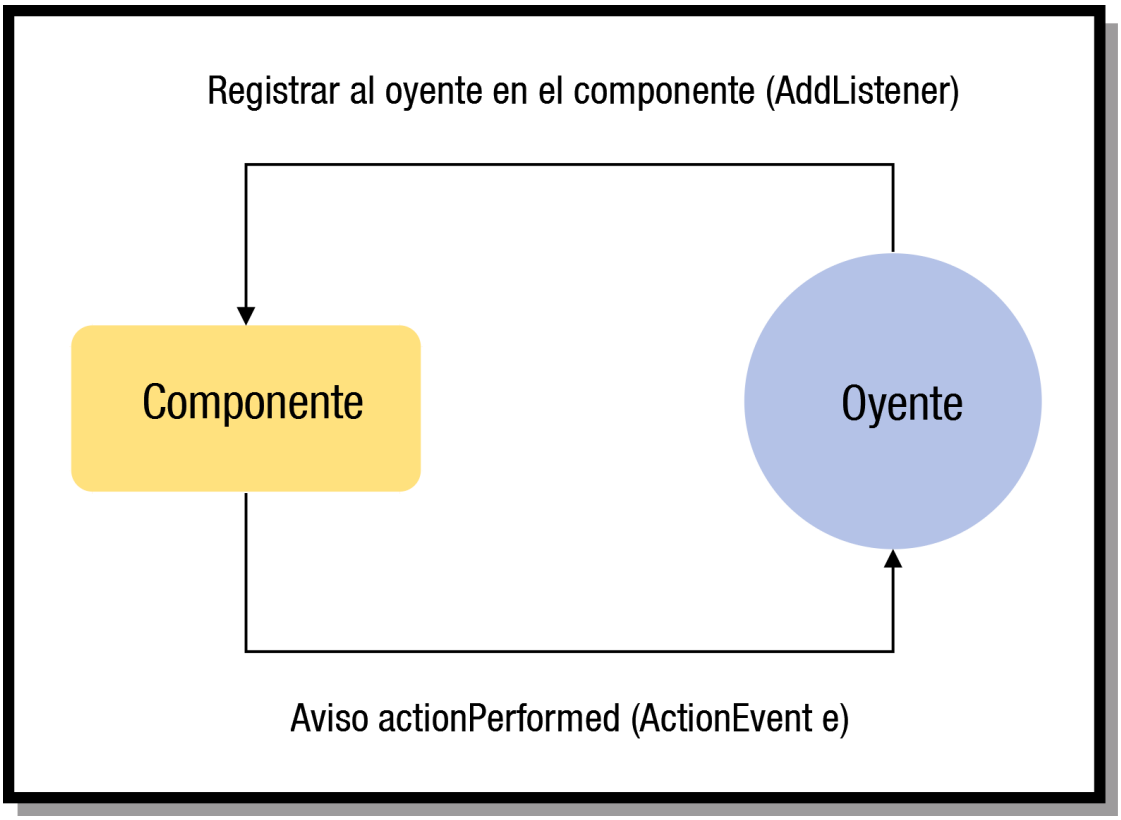
EVENTOS

Es todo hecho que ocurre mientras se ejecuta la aplicación. Normalmente, llamamos evento a cualquier interacción que realiza el usuario con la aplicación, como puede ser:

- pulsar un botón con el ratón;
- hacer doble clic;
- pulsar y arrastrar;
- etc...

Para abordar el problema de tratar correctamente las interacciones del usuario con la interfaz gráfica de la aplicación recurrimos a la **programación guiada por eventos.**

MODELO DE GESTIÓN DE EVENTOS



Cada vez que el usuario realiza una determinada acción sobre una aplicación que estamos programando en Java (un clic sobre el ratón, presionar una tecla, etc.), se produce un evento que el sistema operativo transmite a Java.

Java crea un objeto de una determinada clase de evento, y este evento se transmite a un determinado método para que lo gestione.

El **modelo de eventos de Java está basado en delegación**, es decir, la responsabilidad de gestionar un evento que ocurre en un objeto fuente la tiene otro objeto **oyente**.

Las **fuentes de eventos** (event sources) son objetos que detectan eventos y notifican a los receptores que se han producido dichos eventos. Ejemplos de fuentes:

- Botón sobre el que se pulsa o pincha con el ratón.
- Campo de texto que pierde el foco.
- Campo de texto sobre el que se presiona una tecla.
- Ventana que se cierra.
- Etc.

En el apartado anterior de creación de interfaces con ayuda de los asistentes del IDE, vimos lo fácil que es realizar este tipo de programación, ya que el IDE hace muchas cosas, genera código automáticamente por nosotros.

Pero también podríamos hacerlo nosotros todo, si no tuviéramos un IDE como NetBeans, o porque simplemente nos apeteciera hacerlo todo desde código, sin usar asistentes ni diseñadores gráficos. En este caso, **los pasos a seguir** se pueden resumir en:

1. Crear la clase oyente que implemente la interfaz.
    - Ejemplo: `ActionListener`: pulsar un botón.
2. Implementar en la clase oyente los métodos de la interfaz.
    - Ejemplo: `void actionPerformed(ActionEvent)`.
3. Crear un objeto de la clase oyente y registrarlo como oyente en uno o más componentes gráficos que proporcionen interacción con el usuario.

Antes de ver el siguiente ejemplo he de hacer un inciso. Con el código que se muestra en el flash player del tema 9 apartado 4.2 Modelo de Gestión de Eventos el programa no funciona. No sé por qué, pero se arregla declarando el atributo "`etiqueta`" de tipo **JPanel** como `static` en la clase **Ventanilla** y añadiendo en la clase **OyenteAccion** la sentencia:

Ventanilla.`etiqueta`.`setText`("Botón pulsado: " + `boton`.`getText`());

Veamos el ejemplo que a mi no me funciona:

Creando el programa(I)

`Importamos las clases necesarias para implementar algunos elementos de las interfaces gráficas`

```
1  import javax.swing.JButton;
2  import javax.swing.JFrame;
3  import javax.swing.JLabel;
4  import javax.swing.JPanel;
5
6  public class Ventanilla extends JFrame {
7
8      //Declaramos los atributos
9      JLabel etiqueta;
10     JButton botonUno, botonDos;
11     JPanel panel;
12
```

Definimos los elementos que llevará nuestro diseño

```
13      /*
14       * Creamos los componentes:
15       * una etiqueta, dos botones
16       * y un panel
17       */
18      public Ventanilla() {
19
20          etiqueta = new JLabel("Mi etiqueta");
21          botonUno = new JButton("Botón 1");
22          botonDos = new JButton("Botón 2");
23          panel = new JPanel();
24
25
26          //Añadimos los componentes al panel
27          panel.add(etiqueta);
28          panel.add(botonUno);
29          panel.add(botonDos);
30
31          //Añadimos el panel al Frame
32          getContentPane().add(panel);
33
34          /*
35           * Creamos el objeto de la clase oyente
36           * para cuando se pulse el botón
37           */
38          OyenteAccion oyenteBoton = new OyenteAccion();
39
40          /*
41           * Registrar el objeto como oyente
42           * en los dos botones
43           */
44          botonUno.addActionListener(oyenteBoton);
45          botonDos.addActionListener(oyenteBoton);
46      }
47
48  }
```

Ahora creamos la clase oyente

```
1  import java.awt.event.ActionEvent;
7
8  //Implementación de la clase oyente
9  class OyenteAccion implements ActionListener{
10
11     //Cuando se pinche el botón
12     public void actionPerformed(ActionEvent evento) {
13
14         //Obtener el botón que disparó el evento
15         JButton boton = (JButton) evento.getSource();
16
17         //Escribir en la etiqueta el botón pulsado
18         etiqueta.setText("Botón pulsado " + boton.getText());
19
20     }
```

Y a continuación el main

```
22     //Programa principal
23     public static void main(String args[]) {
24
25         //Creamos la ventana
26         Ventanilla ventana = new Ventanilla();
27
28         //Establecer título, tamaño y visibilidad
29         ventana.setTitle("Nueva Ventana");
30         ventana.setVisible(true);
31         ventana.setSize(500, 100);
32
33
34
35     }
36
37 }
```

De esta manera se muestra el código en el temario pero, como digo, en mi eclipse Oxygen no me funciona. Ciertamente, se está haciendo una llamada a un atributo de otra clase sin más que el nombre de dicho atributo, ni siquiera se declara y, como se puede comprobar, no existen métodos get con los que acceder a los atributos de la otra clase.

El código modificado, apenas sufre modificaciones salvo las ya reseñadas y quedaría como sigue:

Clase Ventanilla (I)

```
1  import javax.swing.JButton;
2  import javax.swing.JFrame;
3  import javax.swing.JLabel;
4  import javax.swing.JPanel;
5
6  public class Ventanilla extends JFrame {
7
8      //Declaramos los atributos
9      static JLabel etiqueta;
10     JButton botonUno, botonDos;
11     JPanel panel;
12
13     /*
14      * Creamos los componentes:
15      * una etiqueta, dos botones
16      * y un panel
17      */
18     public Ventanilla() {
19
20         etiqueta = new JLabel("Mi etiqueta");
21         botonUno = new JButton("Botón 1");
22         botonDos = new JButton("Botón 2");
23         panel = new JPanel();
24
25
```

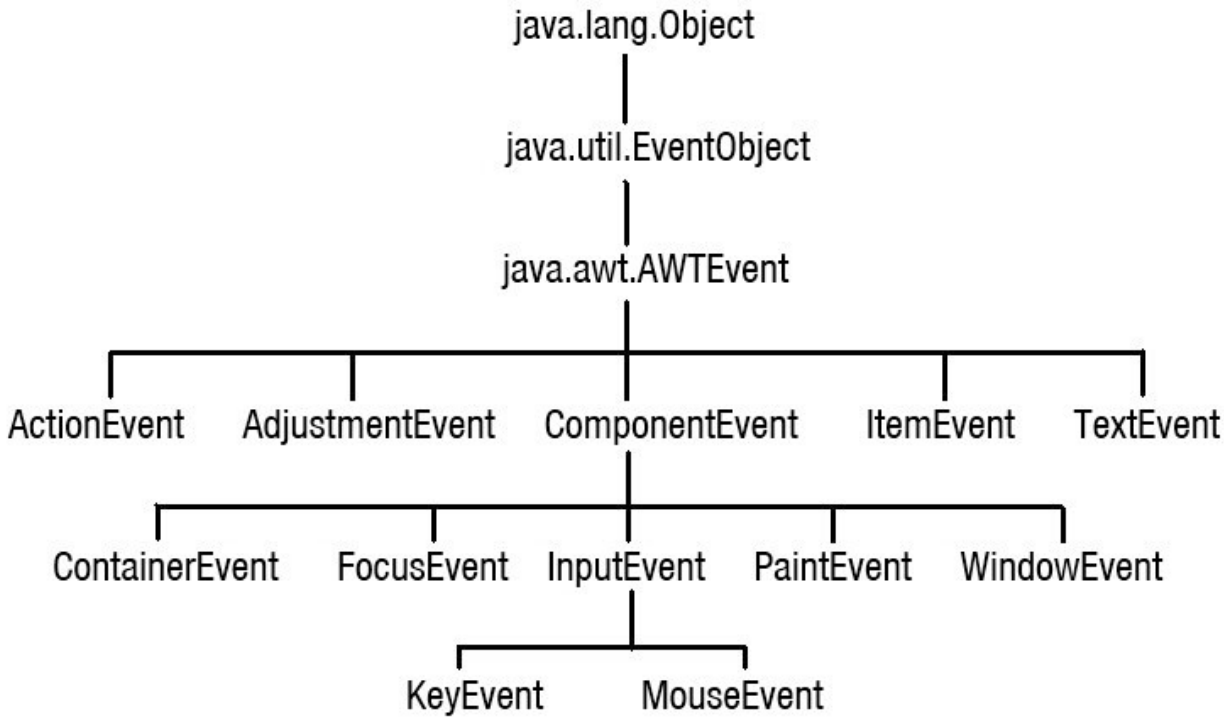
Clase Ventanilla (II)

```
26         //Añadimos los componentes al panel
27         panel.add(etiqueta);
28         panel.add(botonUno);
29         panel.add(botonDos);
30
31         //Añadimos el panel al Frame
32         getContentPane().add(panel);
33
34         /*
35          * Creamos el objeto de la clase oyente
36          * para cuando se pulse el botón
37          */
38         OyenteAccion oyenteBoton = new OyenteAccion();
39
40         /*
41          * Registrar el objeto como oyente
42          * en los dos botones
43          */
44         botonUno.addActionListener(oyenteBoton);
45         botonDos.addActionListener(oyenteBoton);
46     }
47
48 }
```

Clase OyenteAccion y main

```
1  import java.awt.event.ActionEvent;
2  import java.awt.event.ActionListener;
3
4  import javax.swing.AbstractButton;
5  import javax.swing.JButton;
6  import javax.swing.JLabel;
7
8  //Implementación de la clase oyente
9  class OyenteAccion implements ActionListener{
10
11     //Cuando se pinche el botón
12     public void actionPerformed(ActionEvent evento) {
13
14         //Obtener el botón que disparó el evento
15         JButton boton = (JButton) evento.getSource();
16
17         //Escribir en la etiqueta el botón pulsado
18         Ventanilla.etiqueta.setText("Botón pulsado: " + boton.getText());
19
20     }
21
22     //Programa principal
23     public static void main(String args[]) {
24
25         //Creamos la ventana
26         Ventanilla ventana = new Ventanilla();
27
28         //Establecer título, tamaño y visibilidad
29         ventana.setTitle("Nueva Ventana");
30         ventana.setVisible(true);
31         ventana.setSize(500, 100);
32
33
34
35     }
36
37 }
38
```

Tipos de eventos.



En la mayor parte de la literatura escrita sobre Java, encontrarás dos tipos básicos de eventos:

- **Físicos** o de **bajo nivel**: que corresponden a un evento hardware claramente identificable. Por ejemplo, se pulsó una tecla (*`KeyStrokeEvent`*). Destacar los siguientes:
  - En componentes: `ComponentEvent`. Indica que un componente se ha movido, cambiado de tamaño o de visibilidad
  - En contenedores: `ContainerEvent`. Indica que el contenido de un contenedor ha cambiado porque se añadió o eliminó un componente.
  - En ventanas: `WindowEvent`. Indica que una ventana ha cambiado su estado.
  - `FocusEvent`, indica que un componente ha obtenido o perdido la entrada del foco.
- **Semánticos** o de mayor nivel de abstracción: se componen de un conjunto de eventos físicos, que se suceden en un determinado orden y tienen un significado más abstracto. Por ejemplo: el usuario elige un elemento de una lista desplegable (*`ItemEvent`*).
  - `ActionEvent, ItemEvent, TextEvent, AdjustmentEvent`.

Los eventos en Java se organizan en una jerarquía de clases:

- La clase **`java.util.EventObject`** es la clase base de todos los eventos en Java.
- La clase **`java.awt.AWTEvent`** es la clase base de todos los eventos que se utilizan en la construcción de GUI.
- Cada tipo de evento *`loqueseaEvent`* tiene asociada una interfaz *`loqueseaListener`* que nos permite definir manejadores de eventos.
- Con la idea de simplificar la implementación de algunos manejadores de eventos, el paquete **`java.awt.event`** incluye clases *`loqueseaAdapter`* que implementan las interfaces *`loqueseaListener`*.

EVENTOS DE TECLADO

`KeyListener` **(oyente de teclas).**

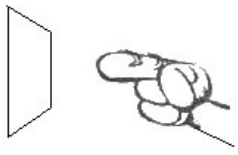
Método	Causa de la invocación
`keyPressed (KeyEvent e)`	Se ha pulsado una tecla.
`keyReleased (KeyEvent e)`	Se ha liberado una tecla.
`keyTyped (KeyEvent e)`	Se ha pulsado (y a veces soltado) una tecla.

`KeyEvent` **(evento de teclas)**

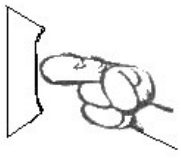
Métodos más usuales	Explicación
`char getKeyChar()`	Devuelve el carácter asociado con la tecla pulsada.
`int getKeyCode()`	Devuelve el valor entero que representa la tecla pulsada.
`String getKeyText()`	Devuelve un texto que representa el código de la tecla.
`Object getSource()`	Método perteneciente a la clase **`EventObject`**. Indica el objeto que produjo el evento.

La clase `KeyEvent`, define muchas constantes así:

- `KeyEventVK_A` especifica la tecla A.
- `KeyEventVK_ESCAPE` especifica la tecla ESCAPE.



Botón en estado normal.



Al pulsar la tecla se disparará el evento **`KeyPressed`**.

Al liberar la tecla se genera el evento **`KeyReleased`**.

EVENTOS DE RATÓN

De manera similar a los eventos de teclado, los eventos del ratón se generan como respuesta a que el usuario pulsa o libera un botón del ratón, o lo mueve sobre un componente.

Método	Causa de la invocación
`mousePressed (MouseEvent e)`	Se ha pulsado un botón del ratón en un componente.
`mouseReleased (MouseEvent e)`	Se ha liberado un botón del ratón en un componente.
`mouseClicked (MouseEvent e)`	Se ha pulsado y liberado un botón del ratón sobre un componente.
`mouseEntered (MouseEvent e)`	Se ha entrado (con el puntero del ratón) en un componente.
`mouseExited (MouseEvent e)`	Se ha salido (con el puntero del ratón) de un componente.

`MouseMotionListener` (oyente de ratón)

Método	Causa de la invocación
`mouseDragged (MouseEvent e)`	Se presiona un botón y se arrastra el ratón.
`mouseMoved (MouseEvent e)`	Se mueve el puntero del ratón sobre un componente.

`MouseWheelListener` (oyente de ratón)

Método	Causa de la invocación
`MouseWheelMoved (MouseWheelEvent e)`	Se mueve la rueda del ratón.

CREACIÓN DE CONTROLADORES DE EVENTOS

La utilidad de estos controladores o manejadores de evento es:

- Crear oyentes de evento sin tener que incluirlos en una clase propia.
- Esto aumente el rendimiento, ya que no "añade" otra clase.

Como inconveniente, destaca la dificultad de construcción: **los errores no se detectan en tiempo de compilación**, sino en tiempo de ejecución.
Por esta razón, es mejor crear controladores de evento con la ayuda de un asistente y documentarlos todo lo posible.

El uso más sencillo de `EventHandler` consiste en instalar un oyente que llama a un método, en el objeto objetivo sin argumentos. En el siguiente ejemplo creamos un `ActionListener` que invoca al método *dibujar* en una instancia de **`javax.Swing.JFrame`**.

```
miBoton.addActionListener(
        (ActionListener)EventHandler.create(ActionListener.class, frame, "dibujar"));
```

Cuando se pulse miBoton, se ejecutará la sentencia **`frame.dibujar()`**. Se obtendría el mismo efecto, con mayor seguridad en tiempo de compilación, definiendo una nueva implementación al interface `ActionListener` y añadiendo una instancia de ello al botón:

```
// Código equivalente empleando una clase interna en lugar de EventHandler.
miBoton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                frame.dibujar();
        }
});
```

Probablemente el uso más típico de `EventHandler` es extraer el valor de una propiedad de la fuente del objeto evento y establecer este valor como el valor de una propiedad del objeto destino. En el siguiente ejemplo se crea un `ActionListener` que establece la propiedad "`label`" del objeto destino al valor de la propiedad "`text`" de la fuente (el valor de la propiedad "`source`") del evento.

```
EventHandler.create(ActionListener.class, miBoton, "label", "source.text")
```

Esto correspondería a la implementación de la siguiente clase interna:

```
// Código equivalente utilizando una clase interna en vez de EventHandler.
new ActionListener {
        public void actionPerformed(ActionEvent e) {
                miBoton.setLabel(((JTextField)e.getSource()).getText());
        }
}
```