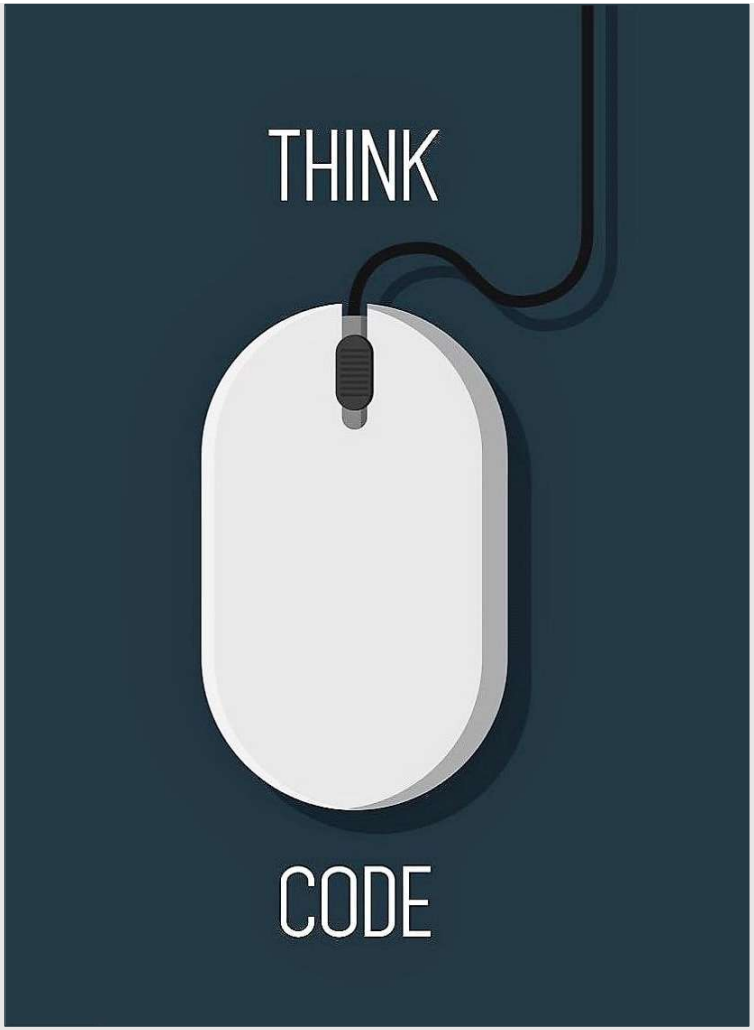
Aplicaciones gráficas Java Swing



Módulo: 2º DAM Desarrollo de Interfaces

Olga M. Moreno Martín

PROMETEO

Licencia

Toda la documentación de esta asignatura queda recogida bajo la licencia de Creative Commons

https://creativecommons.org/licenses/by-nc-nd/4.0/

En el caso de incumplimiento o infracción de una licencia Creative Commons, el autor, como con cualquier otra obra y licencia, habrá de recurrir a los tribunales. Cuando se trate de una infracción directa (por un usuario de la licencia Creative Commons), el autor le podrá demandar tanto por infracción de la propiedad intelectual como por incumplimiento contractual (ya que la licencia crea un vínculo directo entre autor y usuario/licenciatario). El derecho moral de integridad recogido por la legislación española queda protegido aunque no aparezca en las licencias Creative Commons. Estas licencias no sustituyen ni reducen los derechos que la ley confiere al autor; por tanto, el autor podría demandar a un usuario que, con cualquier licencia Creative Commons, hubiera modificado o mutilado su obra causando un perjuicio a su reputación o sus intereses. Por descontado, la decisión de cuándo ha habido mutilación y de cuándo la mutilación perjudica la reputación o los intereses del autor quedaría en manos de cCutacia Juez o Tribunal.

Índice

Introducción

Hasta ahora, todos los ejemplos de código que hemos realizado funcionan exclusivamente en modo texto. La información se visualiza en una consola y también se informa desde dicha consola. La sencillez de este modo de funcionamiento supone una ventaja innegable para el aprendizaje de un lenguaje. Sin embargo, la mayoría de los usuarios de las futuras aplicaciones seguramente esperan disponer de una interfaz un poco menos “deprimente” que una pantalla en modo texto.

En esta UD vamos a estudiar cómo funcionan las interfaces gráficas con Java. El diseño de interfaces gráficas en Java no es tan sencillo y requiere escribir muchas líneas de código. En la práctica, contará con varias herramientas de desarrollo, capaces de encargarse de la generación de una gran parte de este código según el diseño gráfico de la aplicación que esté dibujando. Sin embargo, es importante entender correctamente los principios de funcionamiento de este código para intervenir en él y eventualmente optimizarlo. En este capítulo, no emplearemos ninguna herramienta específica.

Las bibliotecas gráficas

El lenguaje Java propone dos bibliotecas dedicadas al diseño de interfaces gráficas: la biblioteca AWT y la biblioteca SWING. Los fundamentos de uso son casi idénticos en ambas bibliotecas. El uso simultáneo de las dos bibliotecas en una misma aplicación puede provocar problemas de funcionamiento y por ello debería evitarse. Hay una tercera biblioteca más reciente llamada Java FX.

a. La biblioteca AWT

Esta biblioteca es la primera disponible para el desarrollo de interfaces gráficas. Contiene una multitud de clases e interfaces que permiten la definición y la gestión de interfaces gráficas. En realidad, esta biblioteca utiliza las funcionalidades gráficas del sistema operativo. Por lo tanto, no es el código presente en esta biblioteca el que asegura el resultado gráfico de los diferentes componentes. Este código hace de intermediario con el sistema operativo. Su uso ahorra bastante recursos, pero presenta varios inconvenientes:

Al estar relacionado el aspecto visual de cada componente con la representación que el sistema operativo hace de él, puede resultar delicado desarrollar una aplicación que tenga una apariencia coherente en todos los sistemas. El tamaño y la posición de los diferentes componentes son los dos elementos que se ven principalmente afectados por este problema.

Para que esta biblioteca sea compatible con todos los sistemas operativos, los componentes que contiene están limitados a los más corrientes (botones, zonas de texto, listas...).

Las bibliotecas gráficas

b. La biblioteca Swing

- Esta biblioteca se diseñó para resolver las principales carencias de la biblioteca AWT. Se obtuvo esta mejora al escribir completamente la biblioteca en Java sin apenas recurrir a los servicios del sistema operativo. Únicamente algunos elementos gráficos (ventanas y cuadros de diálogo) siguen relacionados con el sistema operativo. Para los demás componentes, es el código de la biblioteca Swing el encargado de determinar completamente su aspecto y su comportamiento.
- La biblioteca Swing contiene por lo tanto una cantidad impresionante de clases que sirven para redefinir los componentes gráficos. Sin embargo, no debemos pensar que la biblioteca Swing convierte la biblioteca AWT en algo completamente obsoleto. De hecho, Swing recupera muchos de los elementos de la biblioteca AWT. En el resto de la unidad emplearemos esencialmente esta biblioteca.

Java Swing

Olga M. Moreno Martín

PROMETEO

Diseño de una interfaz gráfica

Cualquier aplicación gráfica se compone de, al menos, un contenedor de primer nivel. La biblioteca Swing dispone de tres clases que permiten llevar a cabo este papel:

- JApplet : representa una ventana gráfica incluida en una página HTML para que un navegador se haga cargo de ella. Esta clase ya no es útil desde la plataforma Java 11 porque la tecnología de applets ha sido abandonada.
- JWindow : representa la ventana gráfica más rudimentaria que pueda existir. No dispone de ninguna barra de título, ningún menú de sistema, ningún borde: en realidad es un mero rectángulo. Esta clase se utiliza rara vez, excepto para la visualización de una pantalla de inicio en el momento del arranque de una aplicación (splash screen).
- JFrame: representa una ventana gráfica completa y plenamente funcional. Dispone de una barra de título, de un menú de sistema y de un borde. Puede fácilmente contener un menú y, por supuesto, es el elemento que vamos a emplear en la mayoría de los casos.

Las ventanas

- La clase JFrame es un elemento indispensable en cualquier aplicación gráfica. Como en el caso de una clase normal, debemos crear una instancia, modificar eventualmente las propiedades y utilizar los métodos. A continuación se muestra el código de la primera aplicación gráfica.
- Volviendo al programa, cabe admitir que es fácil de usar y muy eficaz. De hecho, es tan eficaz que no se puede parar la aplicación. En efecto, incluso si el usuario cierra la ventana, este cierre no provoca la supresión de la instancia de JFrame de la memoria. La única solución para detener la aplicación es apagar la máquina virtual Java con la combinación de teclas [Ctrl] C. Ante esto, se recomienda proporcionar otra solución para detener más fácilmente la ejecución de la aplicación, es decir, junto con el cierre de la ventana

```
public class Principal{
    public static void main(String[] args) {
        JFrame ventana;
        // creación de la instancia de la clase JFrame
        ventana=new JFrame();
        // modificación de la posición y de la tamaño de la ventana
        ventana.setBounds(0,0,300,400);
        // modificación del título de la ventana
        ventana.setTitle("primera ventana en JAVA");
        // visualizacion de la ventana
        ventana.setVisible(true);
    }
}
```

Las ventanas

- Una primera solución consiste en gestionar los eventos que se producen en el momento del cierre de la ventana y, en uno de ellos, provocar la detención de la aplicación. Se estudiará esta solución en la sección dedicada a la gestión de los eventos.
- La segunda solución utiliza comportamientos predefinidos para el cierre de la ventana. Estos comportamientos están determinados por el método setDefaultCloseOperation . Se definen varias constantes para determinar la acción emprendida al cierre de la ventana.
- DISPOSE_ON_CLOSE: esta opción provoca la detención de la aplicación en el momento del cierre de la última ventana controlada por la máquina virtual.
- DO_NOTHING_ON_CLOSE : con esta opción, no ocurre nada cuando el usuario pide el cierre de la ventana. En este caso, es obligatorio gestionar los eventos para que la acción del usuario tenga algún efecto sobre la ventana o la aplicación.
- EXIT_ON_CLOSE: esta opción provoca la detención de la aplicación incluso si otras ventanas siguen visibles.
- HIDE_ON_CLOSE : con esta opción la ventana simplemente queda oculta como consecuencia de una llamada a su método setVisible(false) .

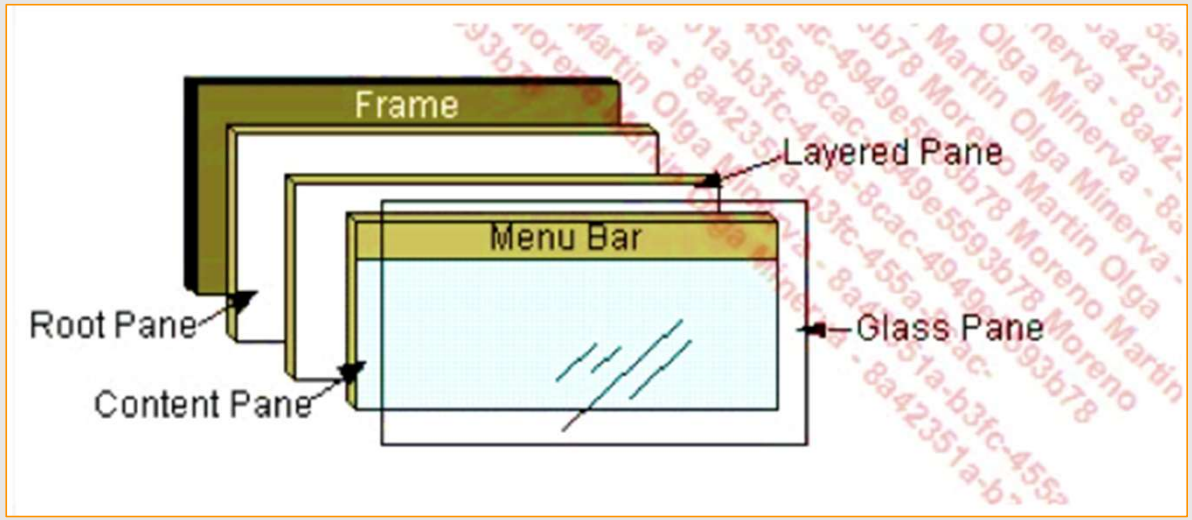
Las ventanas

- La clase JFrame se encuentra al final de una jerarquía de clases bastante importante e implementa numerosas interfaces. Por este motivo, dispone de varios métodos y atributos.

- La meta de estas diapositivas no es retomar toda la documentación del JDK, de modo que no recorreremos todos los métodos disponibles, sino sencillamente los más utilizados según las necesidades. Sin embargo, puede resultar interesante revisar la documentación antes de realizar el diseño de un método para determinar si lo que queremos diseñar no ha sido ya previsto por los diseñadores de Java. La javadoc del paquete java.swing está disponible en la dirección siguiente:
- https://docs.oracle.com/en/java/javase/18/docs/api/java.desktop/javax/swing/package-summary.html

Las ventanas

- Ahora que somos capaces de visualizar una ventana, el grueso del trabajo va a consistir en añadirle un contenido. Antes de poder añadir algo a una ventana, conviene entender bien su estructura, que resulta relativamente compleja. Un objeto JFrame se compone de varios elementos superpuestos, cada uno con un papel muy específico en la gestión de la ventana.



Las ventanas

- El elemento RootPane corresponde al contenedor de los otros tres elementos. El elemento es el responsable de la gestión de la posición de los elementos tanto en los ejes X e Y como en el eje Z, lo que permite la superposición de diferentes elementos. El elemento ContentPane es el contenedor básico de todos los elementos añadidos en la ventana. A él vamos a confiarle, por esta razón, los diferentes componentes de la interfaz de la aplicación. Por encima del ContentPane se superpone el GlassPane , como es posible hacer con un cristal sobre una foto. De hecho, presenta muchas similitudes con el cristal.
    - Es transparente por defecto.
    - Lo dibujado en el GlassPane esconde los demás elementos.
    - Es capaz de interceptar los eventos relacionados con el ratón antes de que estos hayan alcanzado los demás componentes.

Las ventanas

- De todos estos elementos, es sin duda el ContentPane el que vamos a utilizar con mayor frecuencia.
- Podemos acceder a él a través del método getContentPane de la clase JFrame. Desde el punto de vista técnico, es posible ubicar componentes directamente en el objeto ContentPane , aunque es una práctica que Oracle desaconseja. Se prefiere intercalar un contenedor intermedio que contenga los componentes y ubicarlo en el ContentPane . Para ello, se suele utilizar el componente JPanel.
- Según esto, el escenario clásico de diseño de una interfaz gráfica consiste en crear los diferentes componentes y, a continuación, ubicarlos en un contenedor y, por último, situar este contenedor en el ContentPane de la ventana. El ejemplo siguiente lo pone en práctica creando una interfaz usuario compuesta por tres botones.

Las ventanas - Ejemplo

```
public class Principal {
    public static void main(String[] args) {
        // creación de la ventana
        JFrame ventana;
        ventana=new JFrame();
        ventana.setTitle("primera ventana en JAVA");
        ventana.setBounds(0,0,300,100);
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // creación de los tres botones
        JButton b1,b2,b3;
        b1=new JButton("Rojo");
        b2=new JButton("Verde");
        b3=new JButton("Azul");
        // creación del contenedor intermedio
        JPanel pano;
        pano=new JPanel();
        // agregar los botones en el contenedor intermedio
        pano.add(b1);
        pano.add(b2);
        pano.add(b3);
        // agregar el contenedor intermedio en el ContentPane
        ventana.getContentPane().add(pano);
        // visualizacion de la ventana
        ventana.setVisible(true);
    }
}
```

El thread EDT

- La siguiente etapa de nuestro análisis nos va a permitir determinar lo que debe hacer la aplicación cuando el usuario haga clic en alguno de los botones. Pero antes de eso, es necesario comprender cómo se ejecuta una aplicación swing especificando el rol de los thread y más particularmente el thread EDT (Event Dispatching Thread).
- Cuando se ejecuta una aplicación Java sin una interfaz gráfica, se ejecutan al menos dos threads:
  - El thread principal (el thread main), cuya función es ejecutar el método main del programa. Es el punto de entrada obligatorio para ejecutar una aplicación Java SE estándar.
  - El thread recolector de basura (garbage collector) para limpiar la memoria cuando está ocupada más allá de cierto umbral.

El thread EDT

- Un thread es simplemente una unidad de ejecución para un programa. Un programa puede estar compuesto por uno o más thread. La ventaja de tener múltiples threads es que puede ejecutar múltiples tareas en paralelo. Por ejemplo un navegador: es muy común hacer una solicitud a un sitio y, mientras esperamos su descarga y visualización, consultar otra página, ver un vídeo, etc. Todo esto es posible gracias a uso de diferentes threads.
- En una máquina multiprocesador, los diferentes threads se pueden ejecutar en paralelo, explotando los diferentes procesadores. En una máquina con un solo procesador, es la frecuencia de transición entre los diferentes threads lo que permite al usuario pensar que las tareas se ejecutan en paralelo. De hecho, el procesador divide su tiempo entre los diferentes threads. Solo ejecuta un thread a la vez, pausando otros threads durante este tiempo. Por lo tanto, para ejecutar un thread completo, el procesador puede hacer esto varias veces pausando el thread repetidamente para ejecutar los otros threads.

El thread EDT

- Para ejecutar una aplicación Java con una interfaz gráfica, entra en juego un tercer thread: el thread EDT (Event Dispatching Thread). Este thread es responsable de administrar la interfaz gráfica y manejar los eventos (como hacer clic en un botón).
- En el ejemplo anterior, la interfaz gráfica se creó en el método main, por lo tanto, en el thread main. Así, es el thread main el «propietario» de los botones de la interfaz. Si el usuario hace clic en él, se le pide al thread EDT que reaccione a esta acción. Para hacer esto, el thread EDT a menudo necesita acceder a los gráficos. En este caso, es necesaria una operación interthreads. Y aquí es donde pueden surgir los problemas. De hecho, los componentes gráficos no son thread safe. Esto significa que acceder desde otro thread posiblemente podría causar un problema.
- Para evitar este tipo de situación, es necesario asegurarse de que solo el thread EDT manipule la interfaz gráfica, ya sea para la creación de la interfaz o para la gestión de eventos. Por lo tanto, es necesario que, desde el thread principal que ejecuta el método main, se pueda pedir al thread EDT que construya la interfaz gráfica de la aplicación. Esto es posible utilizando el método invokeLater de la clase SwingUtilities. Este método espera en el parámetro un objeto de tipo Runnable que es una interfaz funcional que contiene un único método abstracto cuya firma es la siguiente: void run();
- Este objeto (o expresión lambda) simplemente define lo que debe ser ejecutado por el thread EDT. El código de la siguiente diapositiva realiza una modificación esencial en el ejemplo anterior para garantizar el correcto funcionamiento de la aplicación en todas las situaciones.

El thread EDT

```
public class Principal {
  private JFrame ventana; //Constructor llamado por el thread EDT
  Principal(String[] args) {// Construcción de la iIHM
// creación de la ventana
      this.ventana=new JFrame();
      this.ventana.setTitle("primera ventana en JAVA");
      this.ventana.setBounds(0,0,300,100);
this.ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      // creación de los tres botones
      JButton b1,b2,b3;
      b1=new JButton("Rojo");
      b2=new JButton("Verde");
      b3=new JButton("Azul");
      // creación del contenedor intermedio
      JPanel pano;
      pano=new JPanel();
      // agregar los botones en el contenedor intermedio
      pano.add(b1);
      pano.add(b2);
      pano.add(b3);
      // agregar el contenedor intermedio en el ContentPane
      this.ventana.getContentPane().add(pano);
  }

 //Método llamado por el thread EDT
  public void mostrar() {
     // Visualización de la iIHM
    this.ventana.setVisible(true);
  }
  //Punto de entrada ejecutado por el thread main
  public static void main(final String[] args) {

// Programa una tarea para el thread EDT:
// Creación y visualizacion de la interfaz gráfica

/*SwingUtilities.invokeLater(new Runnable() {
        public void run() {
           new Principal3(args).mostrar();
        }
     });
*/
    SwingUtilities.invokeLater(()->new Principal(args).mostrar());
  }
}
```

La gestión de los eventos

- Todos los SSOO que emplean una interfaz gráfica deben vigilar permanentemente los diferentes periféricos de introducción de datos para detectar las acciones del usuario y transmitirlas a las diferentes aplicaciones. Para cada acción del usuario, se crea un evento. A continuación, se envían estos eventos a cada aplicación, que determinará si le aplica el evento y determina lo que desea realizar como respuesta.
- Java se encarga de determinar qué evento acaba de producirse y sobre qué elemento. El desarrollador es responsable de configurar la sección de código que va a tratar el evento. Desde un punto de vista más técnico, el elemento origen del evento se denomina fuente de evento, y el elemento que contiene la sección de código encargada de gestionar el evento se denomina receptor de evento. Las fuentes de eventos gestionan, para cada evento que pueden activar, una lista que les permite saber qué receptores deben ser avisados si el evento se produce. Por supuesto, las fuentes de eventos y los receptores de eventos son objetos. Es necesario prever qué receptores van a gestionar los eventos que les va a transmitir la fuente de eventos.
- Para garantizar esto, a cada tipo de evento le corresponde una interfaz que debe implementar un objeto si quiere ser candidato a gestionar dicho evento.

La gestión de los eventos

- Los eventos se agrupan en categorías. El nombre de estas interfaces siempre respeta la convención siguiente:
- La primera parte del nombre representa la categoría de eventos que los objetos que implementan esta interfaz pueden gestionar. El nombre siempre termina en Listener .
- Por ejemplo, tenemos la interfaz MouseMotionListener , que corresponde a los eventos activados por los movimientos del ratón, o la interfaz ActionListener , que corresponde a un clic en un botón. En cada una de estas interfaces encontramos las firmas de los diferentes métodos asociados a cada evento.

```
public interface MouseMotionListener extends
EventListener{
      void mouseDragged(MouseEvent e);
      void mouseMoved(MouseEvent e);
}
```

La gestión de los eventos

- Cada uno de estos métodos recibe como argumento un objeto que representa el propio evento. Este objeto se crea automáticamente en el momento de la activación del evento; a continuación, se pasa como argumento al método encargado de gestionar el evento en el receptor de eventos. En general, contiene información adicional relativa al evento y es específico para cada tipo de evento.
- Es preciso crear clases que implementen estas interfaces. Desde este punto de vista, tenemos una multitud de posibilidades:
  - Crear una clase «normal» que implemente la interfaz.
  - Implementar la interfaz en una clase ya existente.
  - Crear una clase interna que implemente la interfaz.
  - Crear una clase interna anónima que implemente la interfaz.
  - Crear eventualmente una expresión lambda si la interfaz es una interfaz funcional.

La gestión de los eventos

- En algunos casos, quizá sea necesario no gestionar todos los eventos presentes en la interfaz. Sin embargo, es obligatorio escribir todos los métodos exigidos por la interfaz incluso si varios de ellos no contienen ningún código. Esto puede perjudicar la legibilidad del código. Para paliar este problema, Java proporciona para casi cada interfaz XxxListener , una clase abstracta correspondiente que implementa la interfaz y que contiene los métodos exigidos por esta. Estos métodos no contienen código alguno, ya que el tratamiento de cada evento debe ser específico a cada aplicación. Estas clases emplean la misma nomenclatura que las interfaces, excepto que se sustituye Listener por Adapter . Tenemos por ejemplo la clase MouseMotionAdapter , que implementa la interfaz MouseMotionListener . Se pueden utilizar estas clases de varias maneras:
  - Creando una clase «normal» que herede de una de estas clases.
  - Creando una clase interna que herede de una de estas clases.
  - Creando una clase interna anónima que herede de una de estas clases.
- El uso de una clase interna anónima es la solución que más se utiliza, con el pequeño inconveniente de que se obtiene una sintaxis difícil de leer si uno no está acostumbrado.

La gestión de los eventos

- Para aclarar todo esto vamos a ilustrar cada una de estas posibilidades con un pequeño ejemplo que nos va a permitir terminar correctamente la aplicación en el momento del cierre de la ventana principal al invocar al método System.exit(0) . Esta solución permite realizar verificaciones antes de detener la aplicación (copia de seguridad, mostrar un mensaje de confirmación, desconectar al usuario...). Es preciso, en este caso, modificar la propiedad DefaultCloseOperation de la ventana con el valor DO_NOTHING_ON_CLOSE para que no tenga una acción por defecto.
- Se debe invocar este método durante la detección del cierre de la ventana. Para ello, debemos gestionar los eventos relacionados con la ventana y, en particular, el evento windowClosing, que se produce cuando el usuario cierra la ventana mediante el menú del sistema. La interfaz WindowListener está perfectamente adaptada para este tipo de trabajo.

La gestión de los eventos

```
public class Pantalla extends JFrame {
    public Pantalla() {
        setTitle("primera ventana en JAVA");
        setBounds(0,0,300,100);
        setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        // creación de los tres botones
        JButton b1,b2,b3;
        b1=new JButton("Rojo");
        b2=new JButton("Verde");
        b3=new JButton("Azul");
        // creación del contenedor intermedio
        JPanel pano;
        pano=new JPanel();
        // agregar los botones en el contenedor intermedio
        pano.add(b1);
        pano.add(b2);
        pano.add(b3);
        // agregar el contenedor en el ContentPane
        getContentPane().add(pano);
    }

    public void mostrar(){
      this.setVisible(true);
    }
}
```

```
public class Principal {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(()->new Pantalla().mostrar());
    }
}
```

Crear una clase «normal» que implemente la interfaz.

Si ejecutamos este código, la ventana aparece, pero ya no es posible cerrarla y aún menos detener la aplicación.

Veamos en la diapositiva siguiente cómo remediar este problema con las diferentes soluciones mencionadas anteriormente.

La gestión de los eventos

Implementar la interfaz en una clase ya existente.

```
public class Pantalla extends JFrame {
    public Pantalla() {
        setTitle("primera ventana en JAVA");
        setBounds(0,0,300,100);
        setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        // creación de los tres botones
        JButton b1,b2,b3;
        b1=new JButton("Rojo");
        b2=new JButton("Verde");
        b3=new JButton("Azul");
        // creación del contenedor intermedio
        JPanel pano;
        pano=new JPanel();
        // agregar los botones en el contenedor intermedio
        pano.add(b1);
        pano.add(b2);
        pano.add(b3);
        // agregar el contenedor en el ContentPane
        getContentPane().add(pano);
    }
    public void mostrar(){
      this.setVisible(true);
    }
}
```

```
public class OyenteVentana implements WindowListener {

    public void windowActivated(WindowEvent arg0) {}
    public void windowClosed(WindowEvent arg0) {}
    public void windowClosing(WindowEvent arg0) {
                System.exit(0);
    }
    public void windowDeactivated(WindowEvent arg0) {}
    public void windowDeiconified(WindowEvent arg0) {}
    public void windowIconified(WindowEvent arg0) {}
    public void windowOpened(WindowEvent arg0) {}
}

public class Principal {
    public static void main(String[] args){
      SwingUtilities.invokeLater(()-> {
      Pantalla pantalla = new Pantalla();
      // creación de una instancia de la clase encargada
      // de administrar los eventos
      OyenteVentana ef=new OyenteVentana();
      // referencia de esta instancia de clase
      // como listaner de evento para la ventana
      pantalla.addWindowListener(ef);
      //visualizacion de la ventana
      pantalla.mostrar();
      });
    }
}
```