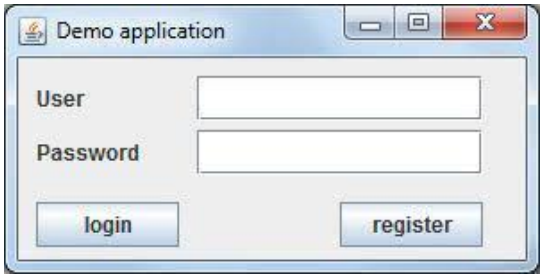
DAM
PROGRAMACIÓN

UD7- INTERFACES GRÁFICAS DE USUARIO

Introducción

- La interfaz de usuario es la parte del programa que permite a éste interactuar con el usuario.
- Tipos de interfaces de usuario:
  - Terminal de comandos.
  - Interfaces gráficas que permiten el uso del ratón y facilitan la visualización y entrada de datos.
- Las interfaces gráficas de usuario (GUI) utiliza una estructura básica de ventanas, botones, cuadros de diálogo, barras de herramientas, listas desplegables y otros elementos.
- Las interfaces gráficas permiten usar diversas alternativas para realizar la entrada de datos:
  - seleccionado una opción entre una lista,
  - pulsando botones,
  - completando formularios y
  - utilizando el ratón.



AWT y Swing

- **Swing** es un conjunto de clases desarrolladas por primera vez para **Java 1.2**,
- **Swing** mejorar el anterior paquete de interfaces gráficas de usuario, el llamado **AWT** (*Abstract Window Tools*).
- **AWT**
  - Usa clases gráficas comunes a todos los sistemas operativos gráficos.
  - El sistema es el que, en último término, crea los componentes.
- **Swing**
  - Utiliza componentes que no dependen del sistema operativo.
  - Aporta muchas más clases, consume menos recursos y construye mejor la apariencia de los programas.
- En cualquier caso, AWT no desaparece; simplemente se añade a las nuevas capacidades Swing.

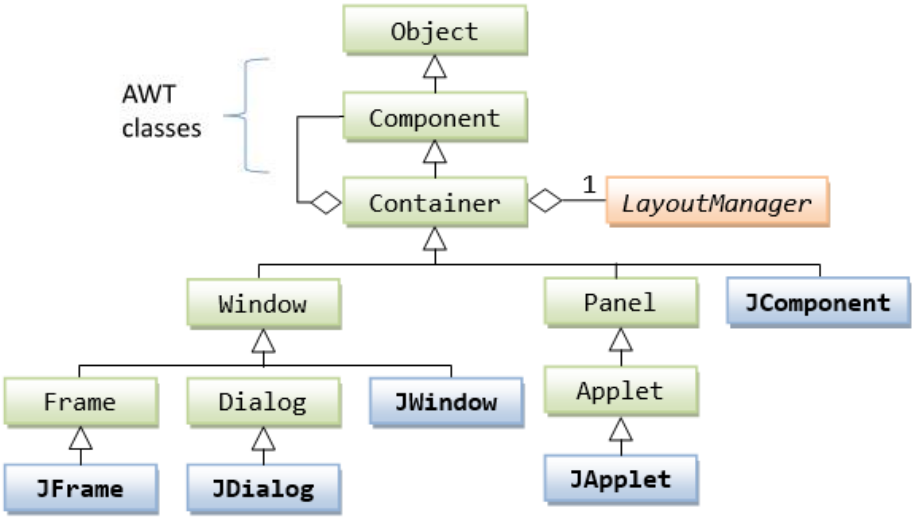
Modelo, Vista, Controlador

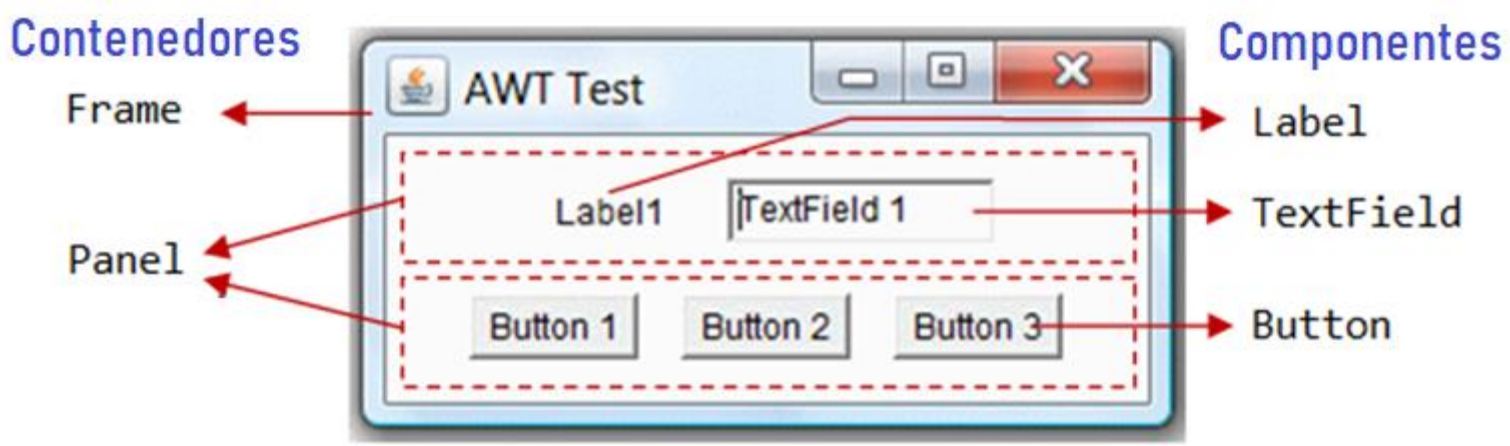
- Otra aportación de Swing es la utilización del paradigma Modelo-Vista-Controlador que facilita la implementación y reutilización del código entre distintas aplicaciones.
  - **Modelo**: recoge el origen de datos del componente.
  - **Vista**: recoge los componentes de la interfaz.
  - **Controlador**: toma la entrada del usuario en la vista y refleja los cambios en los datos del componente.
- Los componentes swing tiene el **Modelo** como elemento separado, mientras que las partes **Vista** y **Controlador** están integradas en los elementos de la interfaz de usuario.

Componentes y contendores

- Los **componentes** son los elementos básicos de la programación con Swing. Todo lo que se ve en un GUI de Java es un componente.
- Los **componentes** se agrupan dentro de **contenedores**.
- Los **contenedores** contienen y organizan la posición de los componentes.
- Los **contenedores** también son componentes que pueden ser incluidos en otros contenedores.

- Cada componente y cada contenedor es una instancia de una clase.
- Todas las clases Swing están dentro del paquete **javax.swing.**





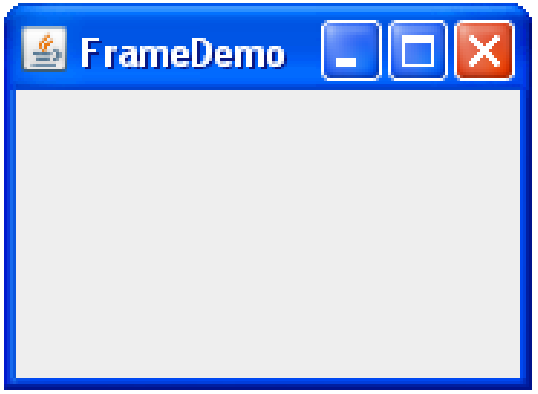
- **Contenedores**:
  - Un **frame** tiene una barra de título (que contiene un icono, un título y los botones minimizar / maximizar / cerrar), una barra de menú opcional y el área de visualización de contenido.
  - Un **panel** es un área rectangular que se utiliza para agrupar componentes de GUI relacionados en un diseño determinado.
  - El **frame** de nivel superior contiene dos paneles.
- **Componentes**:
  - Una **etiqueta** (que proporciona una descripción),
  - Un **campo de texto** (para que los usuarios ingresen texto) y
  - Tres **botones** (para que el usuario active ciertas acciones programadas).

Contenedores I

- Toda aplicación Swing debe tener al menos un contenedor. Los contenedores más comunes en Java son:

JFrame:

- Representa una ventana típica con bordes, botones de manejo de la ventana (cerrar, restaurar y maximizar)
- Es la que se utiliza como ventana principal de un aplicación Swing.
- No puede estar contenida dentro de otras ventanas.



JDialog:

- Permite crear un cuadro de diálogo común, se suele usar como ventanas emergentes.

JApplet

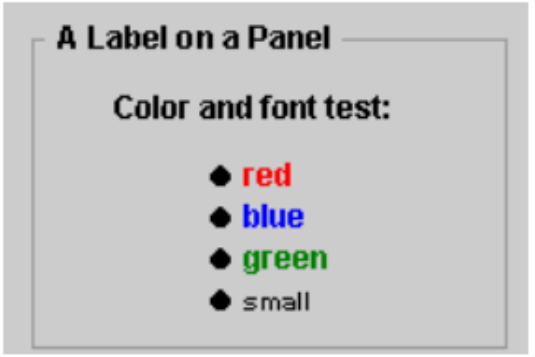
- Permite agrupar componentes que serán mostrados en un navegador.

Contenedores II

- Toda aplicación Swing debe tener al menos un contenedor. Los contenedores más comunes en Java son:

JPanel:

- Contenedor genérico para agrupar componentes.
- Se crea sobre un JFrame para aportar los componentes

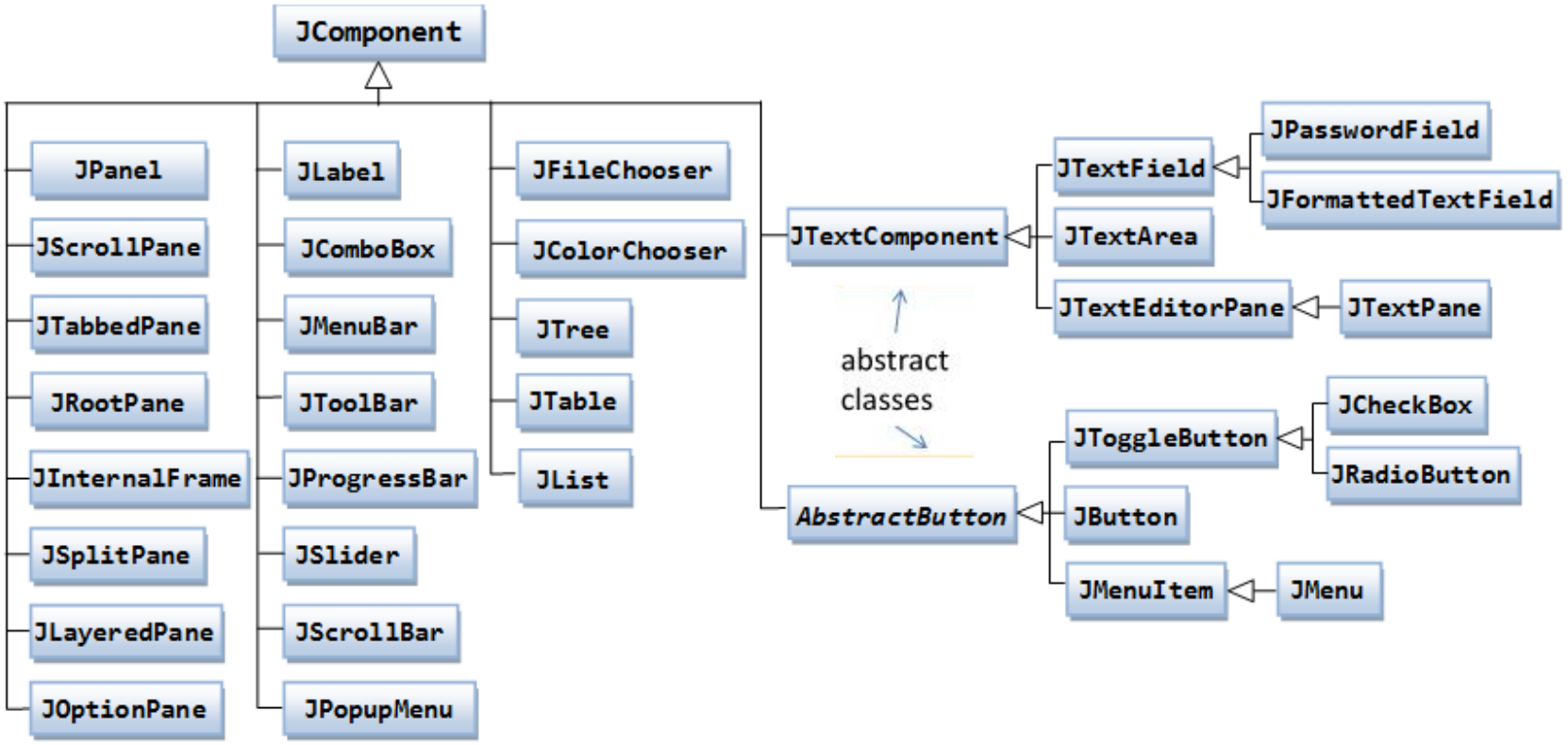


JWindows:

- Representa un panel de ventana que no tiene bordes ni elementos visibles y puede mostrarse en cualquier lugar del escritorio.

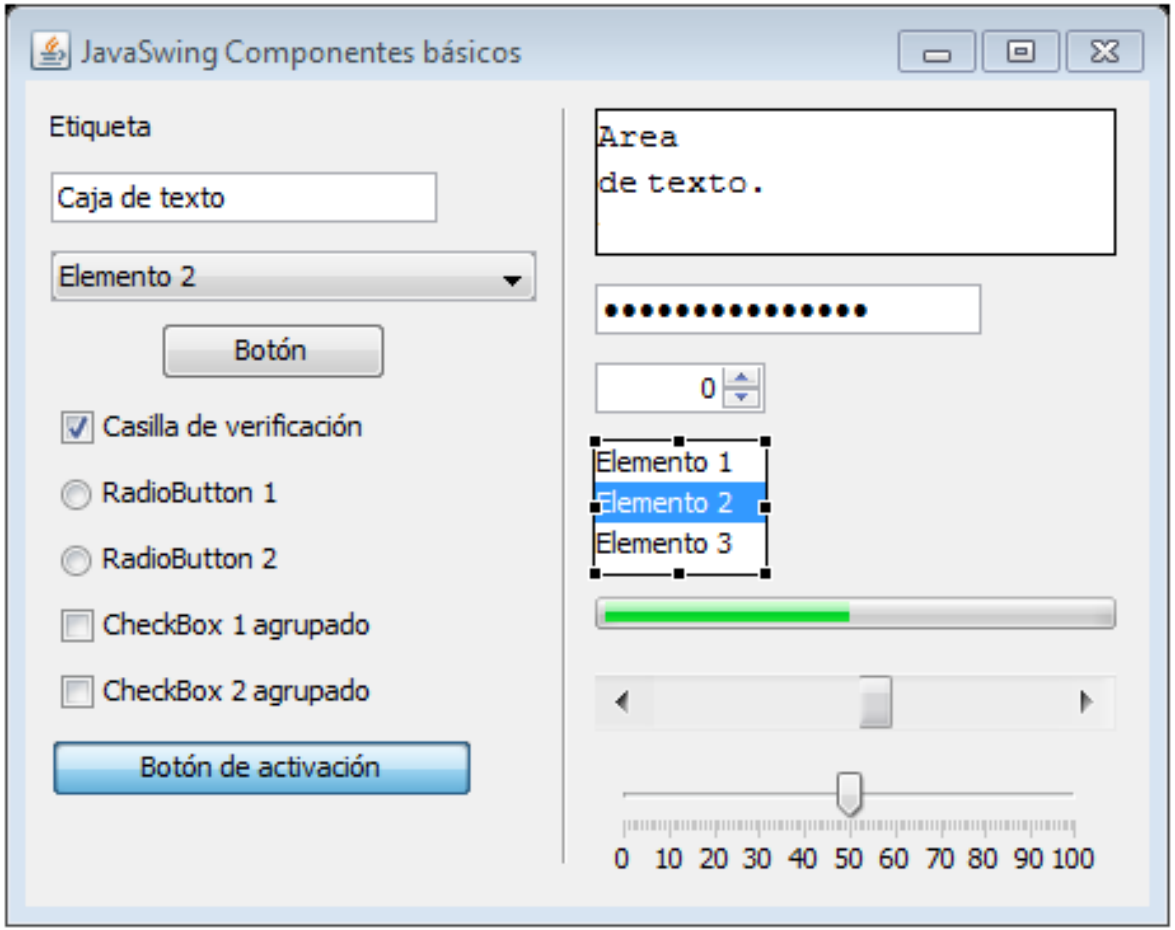
Componentes I

- Los **componentes** de **Swing** pertenecen a clases que heredan de **JComponents**.



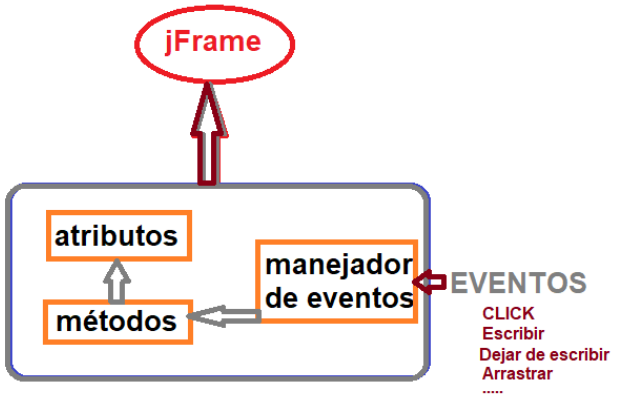
Componentes II

- Algunos de los componentes de **Swing** los vemos en las siguientes imágenes:



Eventos

- **Un evento es un suceso sobre un objeto gráfico**, como por ejemplo, escribir en un cuadro de texto, hacer *click* sobre un elemento de menú o sobre un botón, entre otros.
- El uso de GUI implica:
  - crear las interfaces gráficas y
  - programar los métodos de respuesta a los distintos eventos que se pueden producir en la interfaz gráfica.



Práctica 1. Crear un Jframe

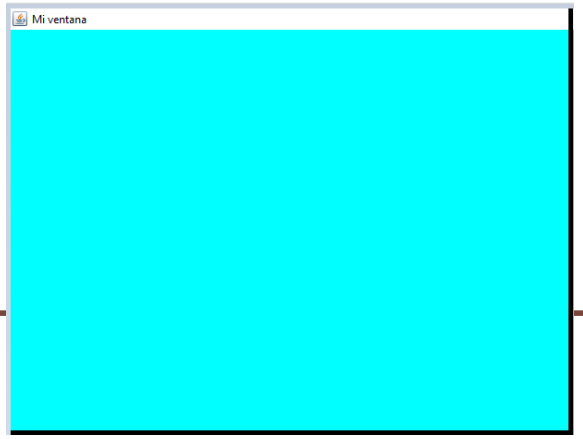
- Implementaremos una primera aplicación en la que se utilizará **Jframe**

```
public class Principal {

    public static void main(String[] args) {
        //1º Crear el objeto del tipo de la clase en en la que está
        VentanaFrame ventanaP = new VentanaFrame();
        //2º Determinar la visibilidad
        //La ventana aparecerá posicionada en la coordenada 0,0 de la ventana
        //(arriba a la izquierda)
        //Podría haber introducido este método en el construtor
        ventanaP.setVisible(true);
        //3º Acción ante el evento de cerrar la ventana --> que cierre la ventana
        //Ocultar al cerrar (JFrame.HIDE_ON_CLOSE)--> se revierte con setVisible(true)
        ventanaP.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

```
public class VentanaFrame extends JFrame{

    public VentanaFrame(){

      //Dar el tamaño al marco teniendo en cuenta la resolución del monitor
      setSize(400,300);
      //Determinar si se quiere redimiensionar la ventana (true) o no (false)
      setResizable(true);
      //Indica la posición de la ventana
      setLocation(300,200);
      //setLocationRelativeTo(null); //Centra la ventana

      //Texto en la ventana de título
      setTitle("Mi ventana");
     //Cambiar el color de fondo de JFrame
      getContentPane().setBackground(Color.CYAN);
      //Indica que se abra a pantalla completa
       setExtendedState(Frame.MAXIMIZED_BOTH);

    }
}
```

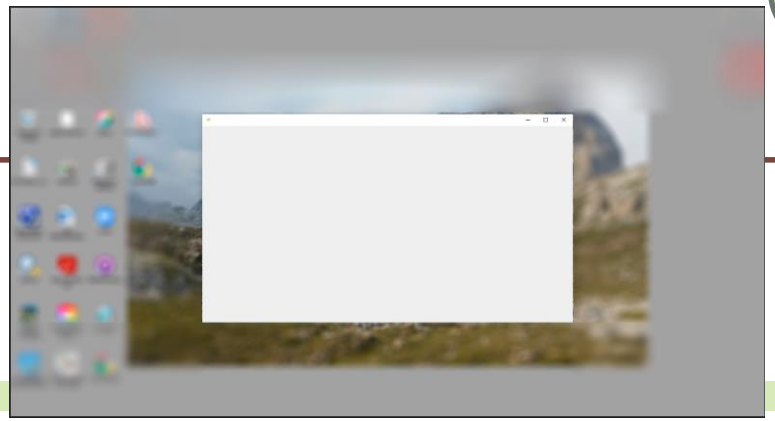
Las propiedades de la ventana se pueden indicar en el constructor o en el objeto ya creado.

```
objeto.setSize(400, 300)
```



```
public class VentanaFrame extends JFrame{
    public VentanaFrame(){
        //Devuelve el sistema nativo de ventanas donde se está ejecutando el programa
        Toolkit elToolkit = Toolkit.getDefaultToolkit();

        //Devuelve un objeto con la resolución de la pantalla
        Dimension laDimension = elToolkit.getScreenSize();

        //Crear una ventana que sea la mitad del ancho y del alto
        setSize(laDimension.width / 2, laDimension.height / 2);

        //Establece coordenadas X e Y. Divido entre 4
        //La referencia siempre es contenedor
        setLocation(laDimension.width / 4, laDimension.height / 4);

       //Modifica el icono de la ventana
        //Se aconseja un tamaño de imagen de 20x20
        Image miIcono = elToolkit.getImage("src/Images/enviar.png");
        setIconImage(miIcono);
    }
}
```

Toolkit facilita el control de la resolución donde se ejecuta la aplicación y la modificación del icono de la ventana



Práctica 2. Crear un JPanel: agregar texto, líneas y cuadros

- Los JPanel se crean y se añaden a un Jframe.
- Los contenedores JPanel contienen los componentes.

```
public class VentanaFrame extends JFrame{

    public VentanaFrame(){
       setVisible(true);
        setSize(500,300);
        setLocation(300,200);
        setTitle("Mi ventana");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //Cambiar el color de fondo de JFrame
        getContentPane().setBackground(Color.CYAN);

        //Instanciamos el panel
        LienzoPanel miPanel = new LienzoPanel();
        add(miPanel);
    }
}
```

```
public class Principal {
    public static void main(String[] args) {
        VentanaFrame miVentana =
                        new VentanaFrame();
    }
}
```

Crear un objeto **JFrame**

Agregar el **JPanel** al **JFrame**

```java
public class LienzoPanel extends JPanel {

    public LienzoPanel(){
        //Hago el panel transparente
        this.setOpaque(false);
    }

    @Override
    public void paintComponent(Graphics g) {
        //Invocar método de la clase padre
        super.paintComponent(g);

        //Cambiar tipo de letra y tamaño
        g.setFont(new Font("TimesRoman", Font.BOLD, 20));
        g.setColor(Color.red);
        g.drawString("Este es nuestro primer texto", 50, 50);
        //Dibujar un rectángulo
        g.drawRect(25, 20, 270, 50);
        /* Graphics2D que permite más operaciones:cb el grosor y el tipo de línea*/
        Graphics2D g2 = (Graphics2D) g;
        g2.setColor(Color.BLUE);
        Rectangle2D rectangulo = new Rectangle2D.Double(100, 100, 200, 150);
        //para dibujar un rectángulo no relleno --> g2.draw(rectangulo);
        g2.fill(rectangulo); //para dibujar el rectángulo relleno
    }
}
```

```
public class LienzoPanel extends JPanel {

    private Image imagen;

    public LienzoPanel(){
      //Hago el panel transparente
      this.setOpaque(false);
    }

    @Override
    public void paintComponent(Graphics g) {
        //Mostrar una imagen en el panel
        try {

            imagen = ImageIO.read(new File("src/images/twitter.png"));

        } catch (IOException ex) {
            System.out.println("La imagen no se encuentra");
        }
        //Dibuja la imagen en el panel
        g.drawImage(imagen, 20,20,null);

    }
}
```

Podemos agregar una imagen al panel

```
public class Principal {
    public static void main(String[] args) {
        VentanaFrame miVentana = new VentanaFrame();
        //Para conocer las fuentes instaladas
        conocerFuentes();

    }

    public static void conocerFuentes() {
        String[] fuentes = GraphicsEnvironment.
                                getLocalGraphicsEnvironment().
                                        getAvailableFontFamilyNames();

        for (String fuente : fuentes) {
            System.out.println(fuente);
        }
    }
}
```

Para conocer las fuentes instaladas

Repuesta al evento ON_CLOSE de una ventana

- Al determinar la respuesta al evento ON_CLOSE de una ventana con el método

```
laVentana.setDefaultCloseOperation(respuesta_evento)
```

- El parámetro `respuesta_evento` puede ser:
  - **DISPOSE_ON_CLOSE**: cierra la ventana, pero no necesariamente finaliza la ejecución del programa.
  - **HIDE_ON_CLOSE**: hace que la ventana parezca cerrarse cambiando la propiedad de visibilidad a false. La diferencia entre HIDE_ON_CLOSE y DISPOSE_ON_CLOSE es que la última libera todos los recursos utilizados por el marco y sus componentes.
  - **EXIT_ON_CLOSE**: cierra la ventana y el programa.
  - **DO_NOTHING_ON_CLOSE**: no hace nada al presionar el botón de cierre. Esto podría ser útil, por ejemplo, si quisieras mostrar un mensaje de confirmación antes de cerrar la ventana.

Ejemplo de uso de DO_NOTHING_ON_CLOSE

```
frame.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
frame.addWindowListener(new WindowAdapter() {
        @Override
        public void windowClosing(WindowEvent e) {
                // Se pide una confirmación antes de finalizar el programa
                int option = JOptionPanel.showConfirmDialog(
                        ventana,
                        "¿Estás seguro de que quieres cerrar la aplicación?",
                        "Confirmación de cierre",
                        JOptionPanel.YES_NO_OPTION,
                        JOptionPanel.QUESTION_MESSAGE);
                if (option == JOptionPane.YES_OPTION) {
                // Si se responde YES el programa se cierra
                        System.exit(0);
                }
        }
});
```

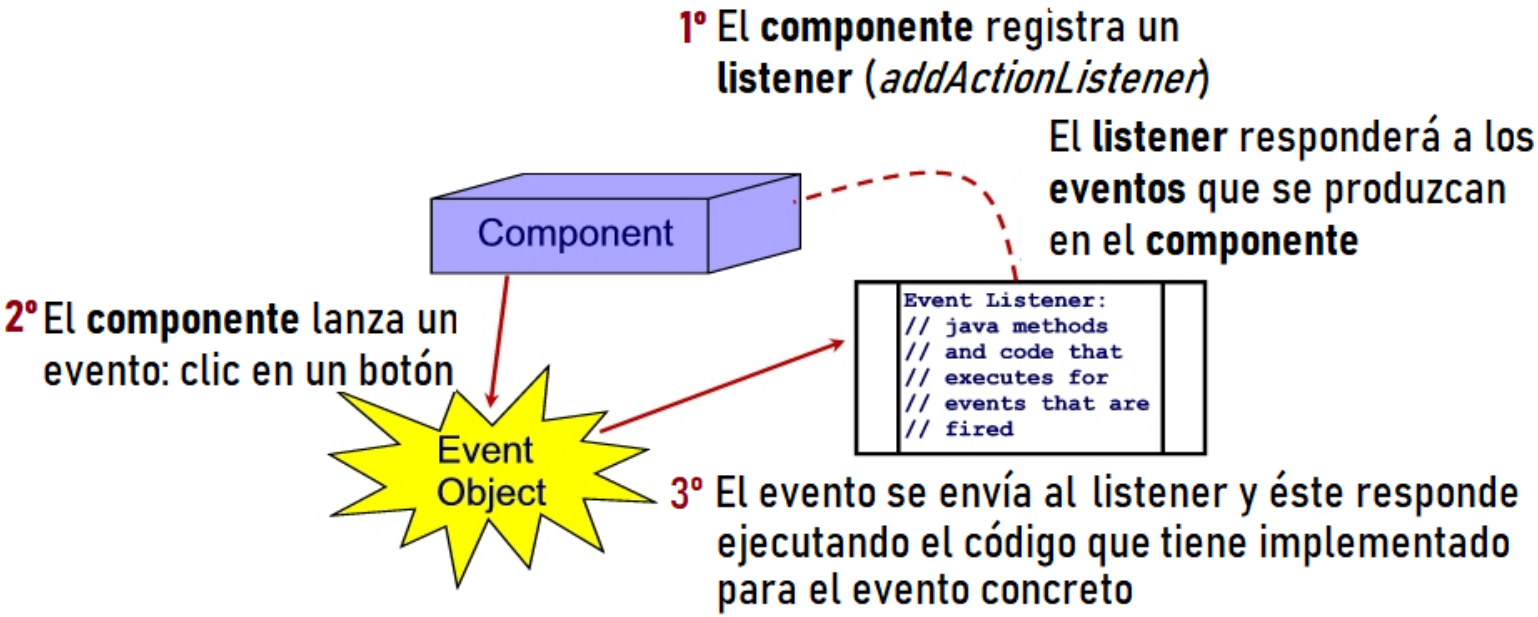
Evento

- Podemos definir un **evento** como un **suceso que puede desencadenar acciones**.
- En **Java** los **eventos** (*clic, cerrar ventana, etc.*) **son objetos** que se generan al producirse determinados sucesos sobre otros objetos (botones, ventanas, etc.).
- Las **acciones** que se producen a partir de un **evento** tienen que **ser recibidas** por un objeto ***listener*** que recibe el evento y ejecuta la acción correspondiente.
- **Evento**:
  - Es un objeto
  - Acción que realiza el usuario o el software.
  - Está asociado a un componente concreto.
  - Cuando se produce un evento se provocan acciones.
  - **Ejemplos**:
    - Al pulsar un botón.
    - Al cerrar una ventana.
    - El entrar en un campo de texto.

Listener

- **Es una interfaz** (*java.util.EventListener*).
- **Capturan los eventos**.
- **Están atentos** (*listener*) **a que el evento se produzca**, cuando esto suceden los capturan y ejecutan las operaciones implementados en el método que corresponda.
- Cada *listener* tiene unos **métodos que debemos implementar.**
- **Para cada tipo de evento existe una interfaz de escucha**.
  - **Para los eventos de tipo ActionEvent existe la interface ActionListener.**
    - Representa una acción del usuario en el interfaz gráfica, como pulsar un botón de la pantalla, elegir una casilla de verificación, etc.
  - **Para los eventos de tipo WindowEvent existe la interface WindowListener.**
    - Se genera cuando una ventana se activa, se cierra, se desactiva, se minimiza, se maximiza, se abre, o se sale de ella.
  - **Para los eventos de tipo MouseEvent existe la interface MouseListener.**
    - Representa las acciones del ratón (mover, arrastrar, hacer clic, etc.).
  - **Para los eventos de tipo KeyEvent existe la interface KeyListener.**
    - Representa las acciones sobre el teclado al pulsar las teclas.

Eventos. Mecanismo de funcionamiento



Un componente (botón, frame, panel, etc.) debe registrar *(add)* un detector *(listener)* de eventos para activar el mecanismo de respuesta ante distintos eventos.
Cuando se dispara un evento en un componente se crea un objeto **evento** y el objeto se pasa por parámetro al objeto registrado como *listener* donde tiene que estar implementado un método específico para el evento producido.

Eventos. Mecanismo de funcionamiento

```
public class LienzoPanel extends JPanel implements ActionListener {
  //Crear un componente botón con el texto que mostrará
  JButton elBoton = new JButton("Prueba evento-emitir sonido");

  public LienzoPanel() {
      add(elBoton); //Agrega un componente
      elBoton.addActionListener(this);
  }

  @Override
  public void actionPerformed(ActionEvent e) {
   //Runnable es una interfaz que permite ejecutar hilos
   final Runnable runnable=
                (Runnable) Toolkit.getDefaultToolkit().
                                   getDesktopProperty("win.sound.exclamation");
     if (runnable != null) { runnable.run();}
  }
}
```

El botón tendrá un *listener* que envía un evento al panel. El evento que provocará el ratón es de tipo *ActionEvent*

Este método pertenece a la interfaz *ActionListener*. Determina qué debe hacer el programa cuando se pulse el botón. Como parámetro recibe el evento producido (*ActionEvent e*)

Ejercicio 1

- **Al pulsar un botón se modifique el color de fondo el panel.**
- En la clase que extienda de **JPanel** se debe incorporar el siguiente código como acción ante el evento de pulsar un botón.

```
 static boolean cambio = true;
…….
 @Override
 public void actionPerformed(ActionEvent e) {
   if (cambio) {
      setBackground(Color.red);
      cambio = false;
   } else {
      setBackground(Color.orange);
      cambio = true;
   }
…….
```

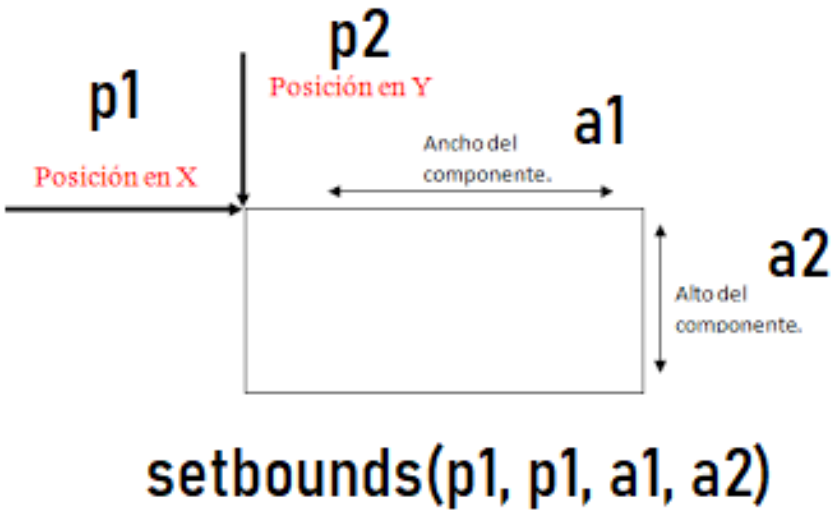
Ejercicio 2

- Cambiar el funcionamiento del botón para que al pulsarlo se cierre el programa después de 3 segundos.

```
try {
    Thread.sleep(3000);
    System.exit(0);
} catch (Exception excep) {
    System.exit(0);
}
```

Posicionar elementos en pantalla

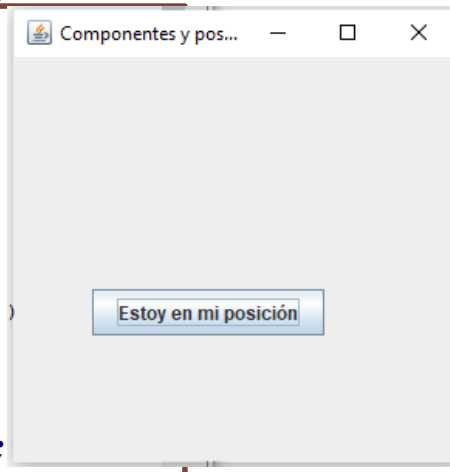
- Se pueden utilizar controladores de posicionamientos (***layouts***) para ordenar los componentes en un contenedor.
- Para posicionar un elemento en la pantalla, sin utilizar ***layouts***, podemos utilizar el método **setbounds()** para determinar:
  - La posición de la esquina superior derecha del elemento.
  - El ancho del elemento.
  - La altura del elemento.
- Al utilizar **setbounds()** con los componentes debemos indicar previamente que no utilizaremos ningún ***layout*** con el método **setLayout(null)** en el contenedor.



Ejemplo

```
public class VentanaFrame extends JFrame{

  JButton boton;

  public VentanaFrame(){
    setLayout(null);
    //Botón
    boton=new JButton("Estoy en mi posición");
    boton.setBounds(50,150,150,30);
    add(boton);
    //JFrame
    setVisible(true);
    setBounds(700,300,300,300);
    setTitle("Componentes y posición");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
}
```

Creamos un botón directamente en el *JFrame* sin usar un Panel



Situamos el botón en la ventana del *JFrame*

Ejemplo: Eventos: Detectar el mismo evento para distintos objetos

```
public VentanaFrame() {
    //Inicializo la ventana
    setVisible(true);
    setBounds(700, 300,325, 300);
    setTitle("Mismo evento en distintos componentes");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    setLayout(null);
    boton1 = new JButton("1");
    boton1.setBounds(10, 100, 90, 30);
    add(boton1);
    boton1.addActionListener(this);
    boton2 = new JButton("2");
    boton2.setBounds(110, 100, 90, 30);
    add(boton2);
    boton2.addActionListener(this);
    boton3 = new JButton("3");
    boton3.setBounds(210, 100, 90, 30);
    add(boton3);
    boton3.addActionListener(this);
}
```

getSource() del evento permite conocer el objeto fuente del evento

```
@Override
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == boton1) {
        setTitle("boton 1");
    }else if (e.getSource() == boton2) {
        setTitle("boton 2");
    }else if (e.getSource() == boton3) {
        setTitle("boton 3");
    }
}
```

```
public class VentanaFrame extends JFrame implements ActionListener {

    JButton boton1, boton2, boton3, boton4;

    public VentanaFrame() {...22 lines }

    @Override
    public void actionPerformed(ActionEvent e) {...11 lines }
}
```

Árbol de herencia de los eventos