# Java Swing

Paquete interfaz gráfica en java

# Interfaces Gráficos en JAVA

- Existen numerosas paquetes de interfaces gráficos en Java, puedes ver las diferencias entre **AWT, Swing y SWT (no es estándar) en**: http://www.slideshare.net/saratorkey/swt-vs-swing
- **Swing** es una biblioteca gráfica para Java. Incluye widgets para interfaz gráfica de usuario tales como cajas de texto, botones, desplegables y tabla. Es una de las partes, junto con Java 2D API, accessibility API, etc, del JFC (Java Foundation Classes)
- **Google Web Toolkit (GWT)** es un conjunto de herramientas de desarrollo para crear y optimizar aplicaciones creadas para navegadores. Es de código abierto y gratuita. Con GWT, puedes desarrollar y depurar aplicaciones AJAX (Asynchronous JavaScript And XML) usando el lenguaje de programación Java en el entorno de desarrollo de tu preferencia (sistema operativo e IDEs). Cuando la aplicación escrita en Java esta finalizada, GWT compila y traduce dicho programa a JavaScript y HTML compatible con cualquier navegador web.
- **JavaFX Scene Builder**: http://docs.oracle.com/javafx/scenebuilder/1/installation_1-1/jsbpub-installation_1-1.htm . JavaFX Scene Builder es una herramienta de diseño que te permite colocar diferentes componentes de un interface gráfico simplemente desplazándolos desde los disponibles de una escena JavaFX. Genera automáticamente el código FXML según vas definiendo la plantilla del interface de usuario (GUI ) de tu aplicación. JavaFX es parte del standard JDK/JRE en version 8
- También hay módulos como el windowBuilder para eclipse que permiten acelerar la generación de interfaces gráficos. En intelliJ hay plugins similares: Bing Vídeos

# Interface gráfico en java: swing

- Swing es una **ampliación del antiguo AWT** y, al igual que éste, es parte del **API standard de java**
- Usa el **paquete javax.swing** (la x se debe a que empezó siendo una ampliación de java (e**X**tended java)
- Con **swing** se pretende lograr un UI (user interface) **independiente de la plataforma** (linux, windows, mac,..). AWT es dependiente de la plataforma
- Las clases de **swing empiezan con J**: JButton, JLabel, …
- Puedes combinar componentes awt y swing (con cuidado)
- **Sistema event-driven** (basado en awt) :Al usar interfaces gráficos el control de la ejecución pasa de estar en el main a estar en los eventos producidos por el usuario sobre el entorno gráfico.
  - Para cada acción del usuario el Sist. Operativo o la JVM decide que código se debe ejecutar, quién recibirá el evento.

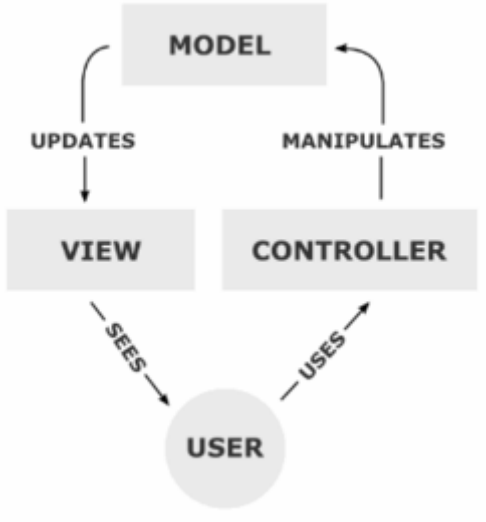
# Interface gráfico en java: swing

- En una aplicación gráfica con swing hay que definir:
  - El **contenedor** JFrame, JApplet, etc que incluirá nuestros botones, menús, etc.
  - La **forma de colocarlos**: Definida por el tipo de layout que queramos (como una matriz, como una lista, etc.)
  - El **tratamiento de los eventos**. Es decir que se debe hacer cuando se pulsa un botón, se mueve el ratón, etc.

# MVC:Modelo Vista Controlador

- Utilizado en Swing: separa los datos de su visualización:
  - Modelo  (1): contiene los datos, envía al visor la información que debe ser mostrada.
  - Vista/visor (n):  controla lo que se ve en pantalla, la presentación
  - Controlador (n): Responde a los eventos del usuario y realiza peticiones al modelo sobre lo solicitado

- Ejem: Tener 1 array de datos y dos visualizadores (con lista y con tabla) si modifico el array se modificará la vista en ambos visualizadores.
- El modelo no sabe nada sobre la presentación. Ésta puede cambiar sin que el modelo se entere.



# Orden del control en MVC

1. El usuario realiza una acción en la interfaz
2. El controlador trata el evento de entrada
   - Previamente se ha registrado
3. El controlador notifica al modelo la acción del usuario, lo que puede implicar un cambio del estado del modelo (si no es una mera consulta)
4. Se genera una nueva vista. La vista toma los datos del modelo
   - El modelo no tiene conocimiento directo de la vista
5. La interfaz de usuario espera otra interacción del usuario, que comenzará otro nuevo ciclo

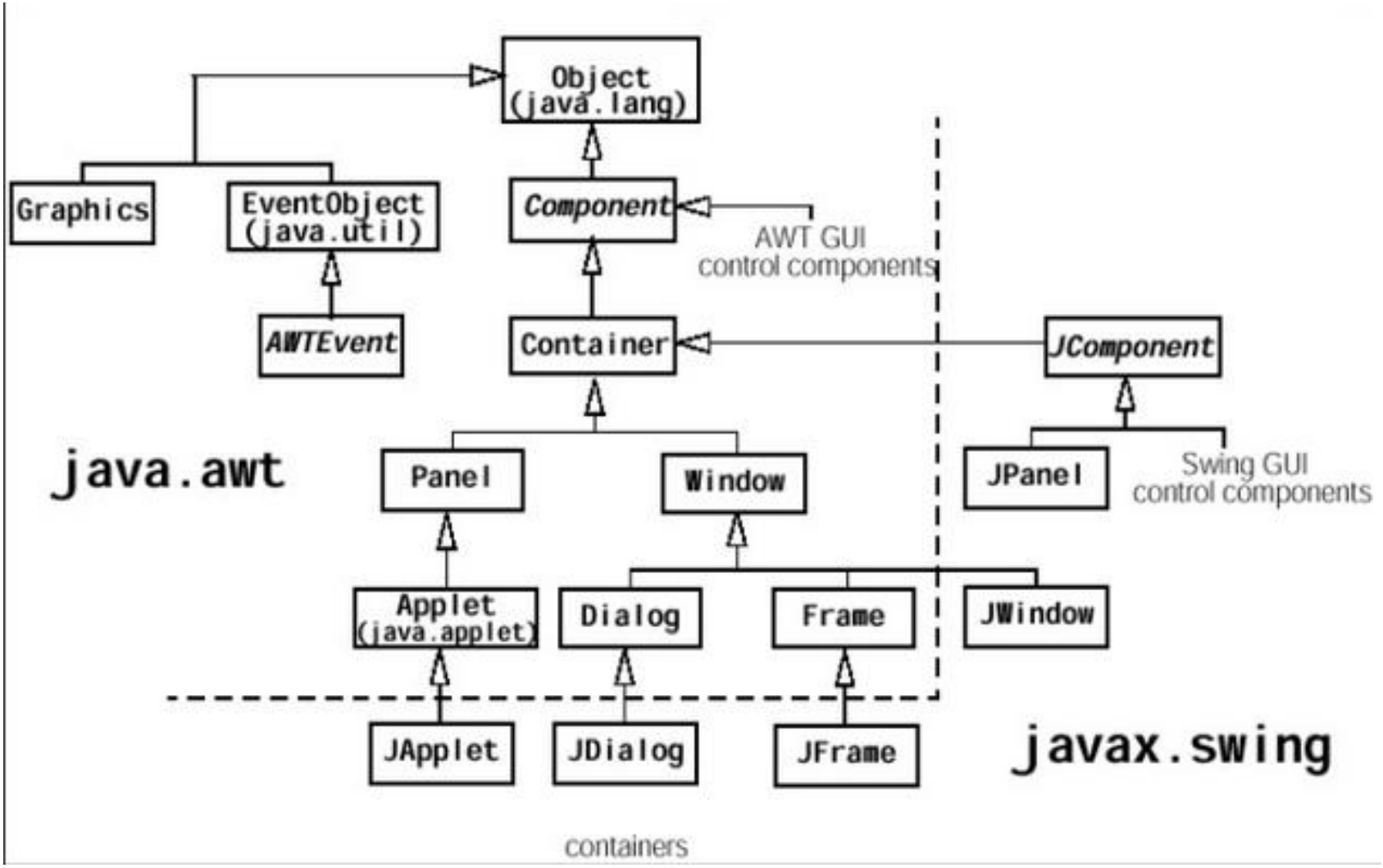
# MVC en Swing

- **Modelo**: El modelo lo realiza el desarrollador, datos almacenados en BD, fichero, xml, etc.
- **Vista**:  Conjunto de objetos de clases que heredan de javax.Swing.Component
- **Controlador**:
  - El controlador es el thread de tratamiento de eventos, que captura y propaga los eventos a la vista y al modelo
  - Clases de tratamiento de los eventos (a veces como clases anónimas) que implementan interfaces de tipo EventListener  (ActionListener, MouseListener, WindowListener, etc )

# Jerarquía aplicaciones Swing

- Cada aplicación Swing debe tener al menos un contenedor.
- Hay 4 contenedores de alto nivel:**JFrame**, **JDialog**, **JWindow** y **JApplet**.
- Para distribuir componentes en un contenedor se usa una manejador de presentación (**Layout**) y un contenedor como **JPanel**. Cada contenedor de alto nivel contiene un contenedor intermedio llamado **content pane** sobre el que se añaden el resto de los componentes.
- Para agregar componentes se usa el método **add()**.
- Un **JFrame** inicialmente es invisible, y para hacerlo visible se usa el método **setVisible(true)**.

# Jerarquía de clases en Swing I



# Jerarquía componentes Swing II

- 1.JComponent
  - 1.1.AbstractButton
    - 1.1.1. JButton: típico botón
    - 1.1.2. JMenuItem
      - 1.1.2.1. JCheckBoxMenuItem: elemento dentro de una lista de menú que se puede seleccionar
      - 1.1.2.2. JMenu: cada una de las etiquetas que contiene un menú
      - 1.1.2.3. JRadioButtonMenuItem: elemento dentro de una lista de menú entre los cuales sólo se puede seleccionar uno simultáneamente
    - 1.1.3. JToggleButton: botón con dos estados posibles
    - 1.1.3.1. JCheckBox: elemento que puede estar seleccionado o no
    - 1.1.3.2. JRadioButton: usado con un ButtonGroup, sólo uno puede estar seleccionado
  - 1.2. JColorChooser: típico panel de selección de color
  - 1.3. JComboBox: lista desplegable de la cual se puede elegir un elemento
  - 1.4. JfileChooser: típico panel de selección de fichero
  - 1.5. JLabel: etiqueta, que no reacciona a eventos, donde se puede poner texto e
  - imágenes
  - 1.6. JList: componente que contiene una lista de la cual podemos elegir uno o más
  - componentes
  - 1.7. JMenuBar: barra superior del menú que contiene JMenu's
  - 1.8. JPanel: contendor genérico sobre el que se añaden componentes
  - 1.9. JPopUpMenu: menú emergente que aparece al pulsar con el botón derecho
  - del ratón
  - 1.10. JProgressBar: típica barra progreso de actividad no acabada

# Jerarquía componentes Swing III

- 1.JComponent
  - 1.11 JScrollBar: barra se scroll
  - 1.12. JScrollPane: panel con dos barras, vertical y horizontal, de scroll
  - 1.13. JSeparator: línea separadora dentro de un menú
  - 1.14. JSlider: típica barra de desplazamiento
  - 1.15. JSplitPane: panel dividido en dos partes
  - 1.16. JTabbedPane- Pestañas
  - 1.17. JTableHeader
  - 1.18. JTextComponent
    - 1.18.1. JEditorPane: facilita la creación de un editor
    - 1.18.2. JTextArea: área donde se puede introducir texto
    - 1.18.3. JTextField: ídem que el anterior pero de una sola línea
    - 1.18.3.1. JPasswordField: el texto se muestra con el símbolo que escojamos
  - 1.19. JToolBar: la barra con iconos que suele aparecer en la parte superior
  - 1.20. JToolTip: el texto emergente que aparece al situar el ratón sobre un componente y que aporta pistas sobre su uso
  - 1.21. JTree: se correspondería a la parte izquierda del explorar de Windows

# Componentes

**Pueden incluir otros componentes**

- JPanel
- JScrollPanel
- JSplitPanel
- JTabbedPane
- JToolBar

**No pueden incluir otros componentes**

**Editables**

- JButton
- JComboBox
- JList
- JMenu
- JSlider
- JSpinner
- JTextFile

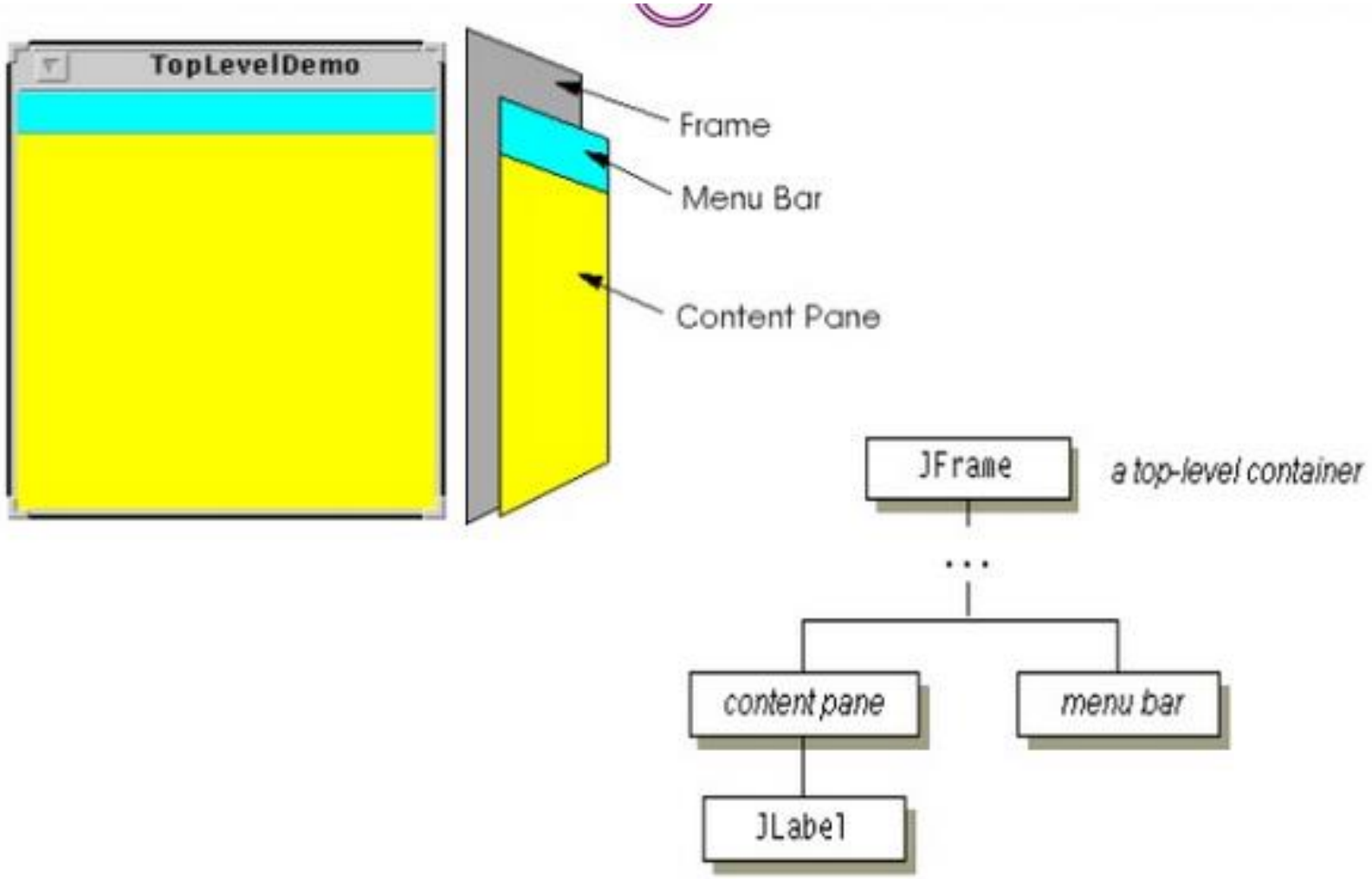
**No Editables**

- JLabel
- JProgressBar
- JToolTip

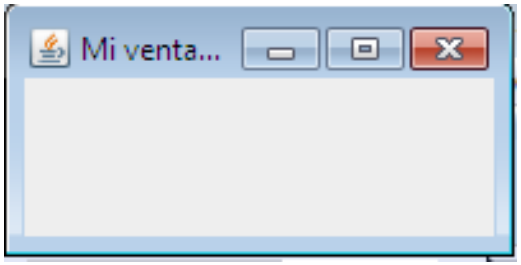
**Otros**

- JColorChooser
- JFileChooser
- JTable
- JText
- JTree

# Estructura de un JFrame



# Creación JFrame básico




```
public class Simple1 {
public static void main (String arg[]){
JFrame ventana= new JFrame ("Mi ventana de texto");
//  ventana.setSize(1400,1600); //No cabe todo el texto
  ventana.setPreferredSize(new Dimension(260,120)); //ancho, alto
  //  ventana.setMaximumSize(new Dimension(600,500));
  ventana.setMinimumSize(new Dimension(150,150));
  ventana.pack(); //Se crea una ventana del tamaño apropiado Para mostrar sus
     componentes (no su título)
  ventana.setState(Frame.NORMAL); //  ventana.setState(Frame.ICONIFIED);
  ventana.setVisible(true);
  System.out.println("Se muestra la ventana y continua la ejecución");
}
}
```

**El evento de cierre de la ventana se gestiona por defecto**

# Creación JFrame básico. Heredando de JFrame

```
import javax.swing.*;
public  class Ejem1 extends JFrame {
// Constantes y componentes (objetos)
  public Ejem1(){
        super(“Mi ventana");
        // Configurar Componentes ;
        // Configurar Manejadores Eventos ;
        setVisible(true);
        // Terminar la aplicación al cerrar la ventana.
        setDefaultCloseOperation(EXIT_ON_CLOSE); //DO_NOTHING_ON_CLOSE
        }
  public static void main(String args[]){
        Ejem1 aplicacion = new Ejem1();
  }
}
```

# Eventos

- Cuando se programa sobre un entorno gráfico, en lugar de ir realizando las acciones que indica el main de forma secuencial y esperar en los puntos que se haya decidido la entrada de datos del usuario lo que se hace es que es el propio usuario, y no el main, el que decide cuando interactúa con la aplicación.
- Cada interacción del usuario produce un evento que es notificado a la aplicación.
- Los eventos se almacenan y se produce una notificación a la aplicación para que los gestione.
- Entre las características que se almacenan de cada evento tendríamos:
  - **id** - El tipo de evento que se ha producido
  - **target** - Componente sobre el que se ha producido el evento
  - **x**, **y** - Las coordenadas en donde se ha producido el evento relativas al Componente que actualmente está procesando ese evento. El origen se toma en la esquina superior izquierda del Componente
  - **key** - Para eventos de teclado, es la tecla que se ha pulsado. Su valor será el valor Unicode del carácter que representa la tecla. Otros valores que puede tomar son los de las teclas especiales como INICIO, FIN, F1, F2, etc.
  - **when** - Instante en que se ha producido el evento
  - **modifiers** - La combinación aritmética del estado en que se encuentran las teclas modificadoras Mays, Alt, Ctrl.
  - **clickCount** - El número de clicks de ratón consecutivos. Sólo tiene importancia en los eventos MOUSE_DOWN
  - **arg** - Es un argumento dependiente del evento. Para objetos Button, este objeto *arg* es un objeto String que contiene la etiqueta de texto del botón
  - **evt** - El siguiente evento en una lista encadenada de eventos

# Eventos

- Cuando un usuario realiza una **acción** sobre el GUI (mover el ratón, pulsar un botón, seleccionar una opción del menú) se produce un **evento.**
- Al objeto/componente que recibe el evento (botón, menú, ..) se le notifica la ocurrencia del evento y si este no lo maneja se propaga hacia arriba en la estructura de componentes/contenedores.
- Para manejarlo hay que implementar la interfaz adecuada (**event handler**) y registrarla para que escuche (**event listener**) ese tipo de evento en el componente adecuado del GUI
- Los eventos están agrupados en :
  - **ActionListener**: acciones sobre componentes. Método actionPerformed
  - **WindowListener**: acciones sobre ventanas (JFrame/JDialog)
  - **MouseListener**: acciones del ratón como presionar un botón del ratón cuando el cursor está sobre un componente.
  - **MouseMotionListener**: Movimiento del cursor sobre un componente
  - **ComponentListener**: visibilidad de los componentes
  - **FocusListener**: obtención del foco del teclado
  - **ListSelectionListener**: Selección de items de una lista
  - **KeyListener**: acciones asociadas al teclado (pulsar, liberar teclas, etc.). Para asociar teclas a botones o selección de componentes mejor usar KeyBindings

# Elementos en el manejo de un evento

- Event source: componente que origina el evento
- Event listener: encargado de escuchar el evento
- Event handler: método que indica las acciones a realizar al producirse el evento:
  - Recibe un objeto evento (ActionEvent) con información sobre lo ocurrido,
  - Descifra el evento
  - Procesa el evento

# Adaptadores

- Java define un conjunto de clases adaptadoras para disminuir la cantidad de implementaciones que se deben hacer al usar una interfaz de listeners.

- Estas clases adaptadoras son abstractas y tienen implementaciones vacías de los métodos de la interfaz. Por lo tanto **se puede definir una subclase de un adaptador y definir sólo el o los métodos que se requieran**

- Por cada *ComponentListener* tenemos su correspondiente *ComponentAdapter*

# Evento básico: cierre del JFrame

- *setDefaultCloseOperation* (int)
- Usa las siguientes *WindowConstants*:
  - *DO_NOTHING_ON_CLOSE*
  - *HIDE_ON_CLOSE* : sólo oculta el frame
  - *DISPOSE_ON_CLOSE*: oculta y elimina el frame
  - *EXIT_ON_CLOSE*: cierra la aplicación (system.exit(0);)
- Uso:

ventana.setDefaultCloseOperation (WindowConstants.*DO_NOTHING_ON_CLOSE);*

# Ejem. Evento cerrar ventana I

```
public class Simple1 {
    public static void main (String arg[]) throws InterruptedException{
        JFrame ventana= new JFrame ("Mi ventana de texto");
        ventana.setVisible(true);

        ventana.setSize(200,100);
        //ventana.pack(); //Se crea una ventana del tamaño apropiado
        //Para mostrar sus componentes
        System.out.println("Se muestra la ventana y continua la ejecución");

        ventana.setDefaultCloseOperation (WindowConstants.HIDE_ON_CLOSE);

        Thread.sleep(4000);
        System.out.println("La vuelvo a hacer visible si se cerró  antes de 4 seg" );
        ventana.setVisible(true);
    }
}
```

# Añadir componentes y escuchadores

```
public class Simple1ConWindowClose {
    public static void main (String arg[]) throws InterruptedException{
        JFrame ventana= new JFrame ("Mi ventana de texto");
        ventana.setVisible(true);
        System.out.println("Se muestra la ventana (sin etiqueta) y continua");
        Thread.sleep(4000);
        JLabel e= new JLabel("Mi Etiqueta");
         ventana.getContentPane().add(e, BorderLayout.CENTER);
         ventana.pack(); //Se crea una ventana del tamaño apropiado para sus
           componentes
           System.out.println("Se refresca la ventana (con etiqueta) y continua");
           ventana.addWindowListener(new WindowAdapter() {  //Escuchador
               public void windowClosing(WindowEvent e) {
                     System.out.println("Se ha cerrado la ventana");
                        System.exit(0);
               }
           });
    }
}
```

# Gestión eventos. Con clase Propia para el Actionlistener I

```
public class BotonModificaTexto extends JFrame {
 JPanel contentPane;
 JTextField Texto1 = new JTextField();
 JButton jButton1 = new JButton();

 //Construir el marco
 public BotonModificaTexto() {
  enableEvents(AWTEvent.WINDOW_EVENT_MASK);
  try {
   inicializacion();
  }
  catch(Exception e) {
   e.printStackTrace();
  }
 }

 //Inicialización de componentes
 private void inicializacion() throws Exception  {
  contentPane = (JPanel) this.getContentPane();
  Texto1.setText("");
  Texto1.setBounds(new Rectangle(9, 12, 64, 22));
  contentPane.setLayout(null);
  this.setSize(new Dimension(400, 300));
  this.setTitle("Mi primera aplicación gráfica");
  jButton1.setBounds(new Rectangle(90, 12, 200, 24));
  jButton1.setText("Escribir");
  jButton1.addActionListener(new
      miAdaptadorParaBoton(this));
  contentPane.add(Texto1, null);
  contentPane.add(jButton1, null);
 }

 //Modificado para poder salir cuando se cierra la ventana
 protected void processWindowEvent (WindowEvent e) {
  super.processWindowEvent(e);
  if (e.getID() == WindowEvent.WINDOW_CLOSING)
      System.exit(0);
 }

 void accionARealizar(ActionEvent e) {Texto1.setText("Hola");}
}  //Fin clase BotonModificaTexto

class miAdaptadorParaBoton implements
      java.awt.event.ActionListener {
 BotonModificaTexto miVentana;

 miAdaptadorParaBoton(BotonModificaTexto
      miVentana) {
  this.miVentana = miVentana;
 }
 public void actionPerformed(ActionEvent e) {
 miVentana.accionARealizar(e);
 }
}
```

Al definir setLayout(null) puedo definir la posición en la que se colocarán los componentes (Jbutton1 y Texto1) y su tamaño utilizando el método setBounds (coordX, coordY, ancho, alto)

No es conveniente hacerlo así si se puede cambiar el tamaño de la ventana

# Gestión eventos. Con clase propia II

```
public class Principal {
  boolean packFrame = false;
  //Construir la aplicación
  public Principal() {
BotonModificaTexto contenidoVentana = new BotonModificaTexto();
    //Validar que los elementos están en su sitio, validar marcos que tienen
       tamaños preestablecidos
    //Empaquetar marcos que cuentan con información de tamaño preferente
       útil. Ej. de su diseño.
    if (packFrame)   contenidoVentana.pack();
     else                 contenidoVentana.validate();
//Centrar la ventana en la pantalla
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
 System.out.println(Toolkit.getDefaultToolkit().getScreenSize());
    Dimension frameSize = contenidoVentana.getSize();
    if (frameSize.height > screenSize.height)
      frameSize.height = screenSize.height;
    if (frameSize.width > screenSize.width)
      frameSize.width = screenSize.width;
    contenidoVentana.setLocation((screenSize.width - frameSize.width) / 2,
    (screenSize.height - frameSize.height) / 2);
    contenidoVentana.setVisible(true);
  }
```

```
  //Método Main
  public static void main(String[]
args) {
      new Principal();
  }
}
```

# Pasos para la creación de una aplicación I : Apariencia

- Especificar la apariencia (windows, motif,…:

*public static void main(String[] args) {*

*try {*

***UIManager.setLookAndFeel(***
***UIManager.getCrossPlatformLookAndFeelClassName(***
***));*** *} catch (Exception e) { }*

*…*

*//Crear y mostrar el GUI...*

*}*

*Para que sea cualquiera que se adapte al entorno actual*

# Pasos para la creación de una aplicación II :Contenedor Principal

- Cada programa con Swing GUI contiene, al menos, un contenedor Swing de alto nivel.
- La mayoría de esos contenedores son instancias de JFrame, JDialog, o JApplet.
- Cada objeto JFrame tiene una ventana principal
- El contenedor Swing de alto nivel proporciona el soporte que necesitan los componentes que añadiremos para ser mostrados y realizar el manejo de los eventos.
- Ejemplo: Con un único contenedor JFrame que finaliza la aplicación cuando usuario cierra la ventana:

# Pasos para la creación de una aplicación II :Contenedor Principal Ejemplo

```
public class SwingApplication {
//    ...
    public static void main(String[] args) {
//...
        JFrame frame = new JFrame("SwingApplication");
        //...Aquí crearemos los componentes que van en la ventana.
        //...y las colocaremos en el contentPane ...
        frame.getContentPane().add (contents, BorderLayout.CENTER);

        //Terminamos configurado y mostrando la ventana.
        frame.addWindowListener(...);
        frame.pack();
        frame.setVisible(true);
    }
}
```

# Pasos para la creación de una aplicación III : Añadiendo botones y etiquetas

- Entre los componentes que podemos añadir a nuestra ventana los más básicos serían los botones y las etiquetas
- Para inicializar un botón haremos lo siguiente:
  - Crear el botón con new Jbutton
  - Indicar la tecla que actuará como si hubiéramos hecho clic con el ratón en el botón (en nuestro ejemplo Alt+I)
  - Añadir un manejador del evento que se ejecutará cuando se pulse el botón
  - Añadir el botón al contenedor, indicando la posición.

```
JButton button = new JButton("Texto del botón");
button.setMnemonic(KeyEvent.VK_I);
button.addActionListener( Crear un manejador del evento...);
frame.getContentPane().add(button, BorderLayout.CENTER);
```

# Pasos para la creación de una aplicación III : Añadiendo botones y etiquetas

- Para darle funcionalidad a la etiqueta crearemos, como atributos de la clase principal los siguientes

```
private static String txtEtiqueta = "Número de pulsaciones en el botón: ";
  private int numClicks = 0;
```

- Crearíamos la etiqueta

```
final JLabel label = new JLabel(txtEtiqueta + "0    ");
```

- Y podríamos modificar su texto con :

```
 label.setText("Num pulsaciones:"+ numClicks);
```

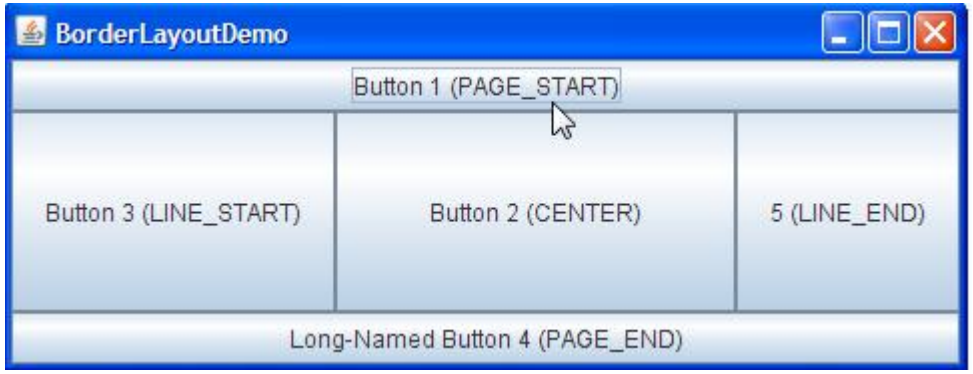
# Pasos para la creación de una aplicación IV: Manejando los eventos

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      numClicks++;
      label.setText("Num pulsaciones:"+ numClicks);
   }
});
...
     frame.addWindowListener(new WindowAdapter() {
           public void windowClosing(WindowEvent e) {
           System.exit(0); }
    });
```

# Layout

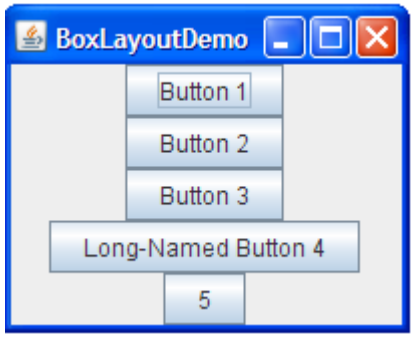
- Los Layout (manejadores de presentación) controlan la forma en la que se colocarán los diferentes componentes. La posición y el tamaño de un componente en un contenedor
- Para cambiar una presentación se puede definir el layout en el constructor (new Jpanel (new nuevoLayout() ) )o con el método **setLayout()**.
  - La presentación por default de un **JFrame/JDialog** y el **content pane contenido en applets y aplicaciones** es **BorderLayout**.
  - El layout por defecto de un **JPanel es FlowLayout**
- El tamaño y posición de un componente se puede controlar apagando el manejador de presentación (colocandolo a **null**) y manejando los métodos **setLocation()**, **setSize()** y **setBounds()**.
- Existen diferentes tipos:
- BorderLayout
- BoxLayout
- CardLayout
- FlowLayout
- GridBagLayout
- GridLayout
- GroupLayout
- SpringLayout

# Layouts I




BorderLayout: 5 posiciones:

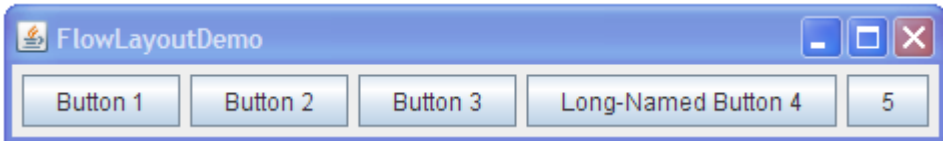
- PAGE_START
- PAGE_END
- LINE_START
- LINE_END
- CENTER




BoxLayout. Coloca los Componentes
uno detrás de otro en columna

# Layouts II

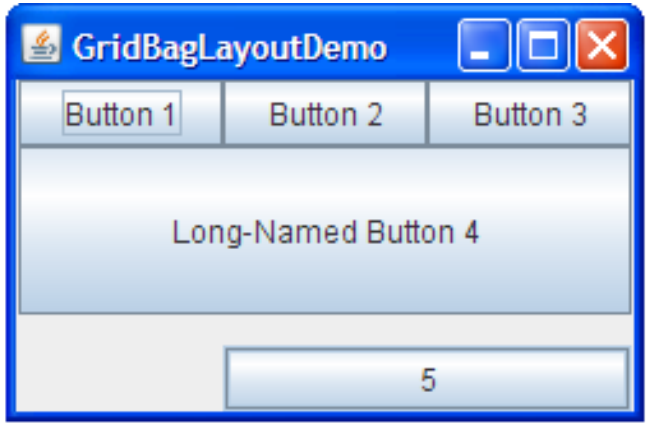

CardLayout: coloca los Componentes en varias vistas. Una está detrás de la otra y sólo es visible una de ellas


FlowLayout. Coloca los Componentes
uno detrás de otro en fila

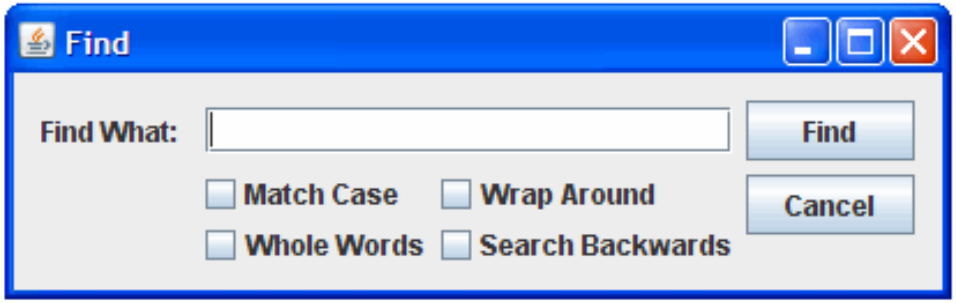
# Layouts III


GridLayout: Define una matriz de casillas de tantas columnas y filas como indiques y coloca los componentes en ellas.

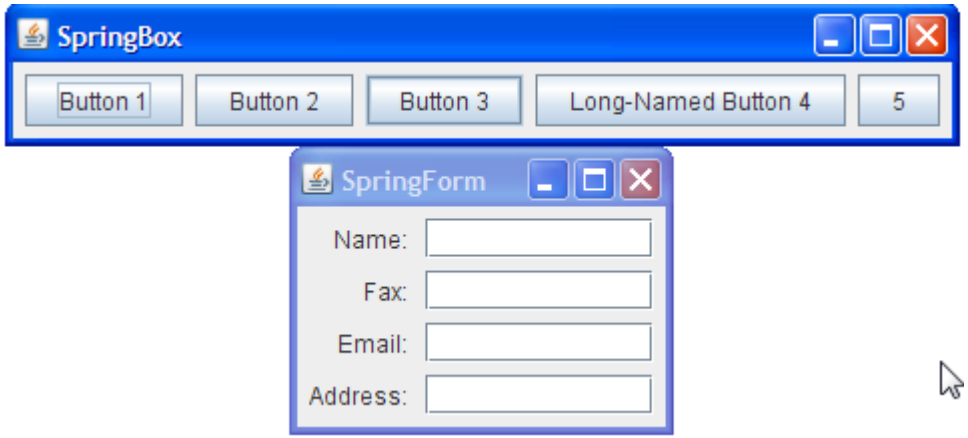



GridBagLayout. Coloca los Componentes
en una red de celtas. Es más flexible que el GridLayout

# Layouts IV




GroupLayout: Diseñado para uso en interfaces gráficas de generación automática. Permite colocar componentes de forma independiente en horizontal y vertical




SpringLayout: También fue diseñado para uso en interfaces gráficas de generación automática.
Permite definir relaciones entre las posiciones de los componentes

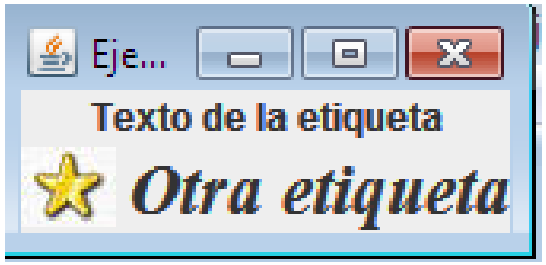
# Componentes Swing

- Existe una gran variedad de componentes.
- En las transparencias siguientes veremos algunos ejemplos de algunos de ellos.

# Clase principal para ejemplos

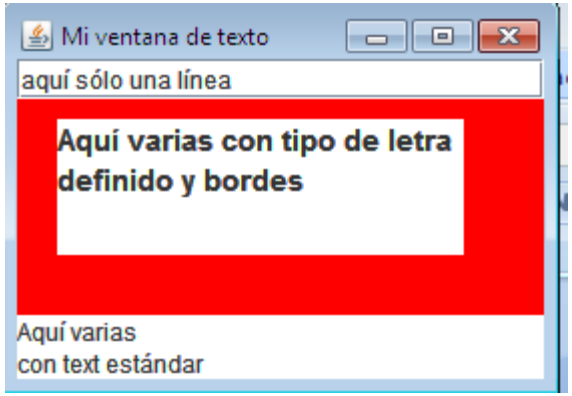
```
public class EjemBasico {
public static void main (String arg[]) throws InterruptedException{
JFrame ventana= new JFrame ("Mi ventana de texto");
//Etiqueta e= new Etiqueta();    //Boton e= new Boton("Mi botón", ventana);  //ZonaTexto e= new
      ZonaTexto();
//BarraScroll e= new BarraScroll();  //Menu e = new Menu();   //ListaOpciones e = new
      ListaOpciones();
PanelAyuda e= new PanelAyuda("Ayuda de mi botón");
   ventana.getContentPane().add(e, BorderLayout.CENTER);
   ventana.pack(); //Se crea una ventana del tamaño apropiado para sus componentes
   ventana.setVisible(true);
System.out.println("Se muestra la ventana ");
   ventana.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
        System.out.println("Se ha cerrado la ventana");
           System.exit(0);
        }
     });
}
}
```

# JLabel



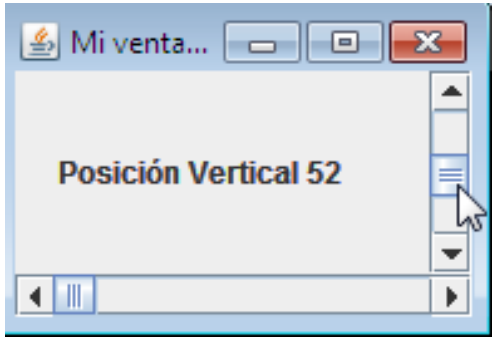

```
public class Etiqueta extends JPanel{
    public Etiqueta (){
        setLayout(new BorderLayout());
        JLabel etiqueta= new JLabel ("Texto de la etiqueta");
        etiqueta.setHorizontalAlignment(JLabel.CENTER);
        add(etiqueta,BorderLayout.NORTH);  //Añado la etiqueta al panel

        JLabel etiqueta2=new JLabel ("Otra etiqueta");
        etiqueta2.setFont(new Font("Serif", Font.BOLD|Font.ITALIC, 22));
        Icon miFoto = new ImageIcon("iconoEstrella.jpg");
        //etiqueta2.setSize(10, 20);
        etiqueta2.setIcon(miFoto);
        etiqueta2.setHorizontalAlignment(JLabel.RIGHT);
        add (etiqueta2,BorderLayout.SOUTH);
    }
}
```

# Campos de texto



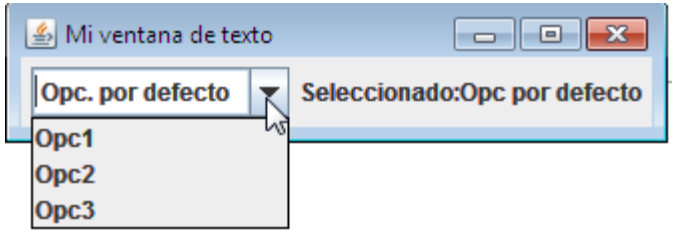

```
public class ZonaTexto extends JPanel{
ZonaTexto(){
    setLayout (new BorderLayout());
    JTextField unaLinea= new JTextField ();
    JTextArea zonaTxt= new JTextArea();
    JTextPane zonaTxtConfigurable= new JTextPane();
    MutableAttributeSet atributoTxt= new SimpleAttributeSet();
    StyleConstants.setBold(atributoTxt,true);
    StyleConstants.setFontSize(atributoTxt, 15);
    zonaTxtConfigurable.setCharacterAttributes(atributoTxt, false);
    zonaTxtConfigurable.setBorder(new MatteBorder(10,20,30,40,Color.red));
    add(unaLinea, BorderLayout.NORTH);
    add(zonaTxt, BorderLayout.SOUTH);
    add(zonaTxtConfigurable, BorderLayout.CENTER);
}
}}
```

# ScrollBar

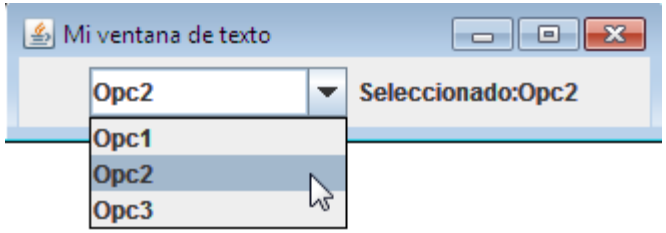



```
public class BarraScroll extends JPanel {
  JLabel posicion;
  JScrollBar BarraDesplazamientoV;
  BarraScroll (){
    setLayout (new BorderLayout());
    posicion=new JLabel ("Sin datos",JLabel.CENTER);
    add(posicion, BorderLayout.CENTER);
    BarraDesplazamientoV = new JScrollBar(JScrollBar.VERTICAL,0,5,0,100);
    add(BarraDesplazamientoV, BorderLayout.EAST);
    JScrollBar BarraDesplazamientoH = new JScrollBar(JScrollBar.HORIZONTAL,0,5,0,100);
    add(BarraDesplazamientoH, BorderLayout.SOUTH);
    BarraDesplazamientoV.addAdjustmentListener(new AdjustmentListener() {
          public void adjustmentValueChanged(AdjustmentEvent e) {
           //  label.setText("    New Value is " + e.getValue() + "      ");
           posicion.setText("Posición Vertical " + BarraDesplazamientoV.getValue() + "      ");
            repaint();
           }
     });
  }
}
```

# Menú: JComboBox

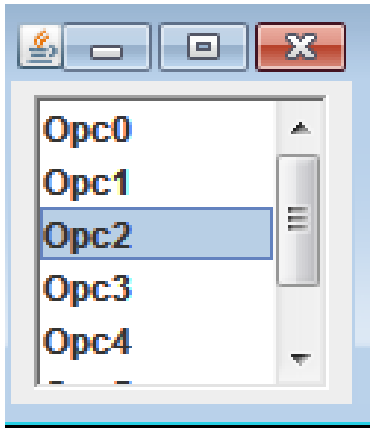



```
public class Menu extends JPanel{
String opciones[]={"Opc1", "Opc2", "Opc3"};
JLabel mensaje;
JComboBox menu1;
  Menu (){
  mensaje=new JLabel ("Seleccionado:Opc por defecto",JLabel.CENTER);
  menu1= new JComboBox ();
  for (int i=0; i<opciones.length; i++)
      menu1.addItem(opciones[i]);
  menu1.setEditable(true);
  menu1.setSelectedItem(" Opc. por defecto");
  menu1.setMaximumRowCount(4);
  menu1.addActionListener(null); JList lista
  menu1.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    mensaje.setText("Seleccionado:" +menu1.getSelectedItem().toString()); } });
  add (menu1);    add(mensaje);
  }
}
```



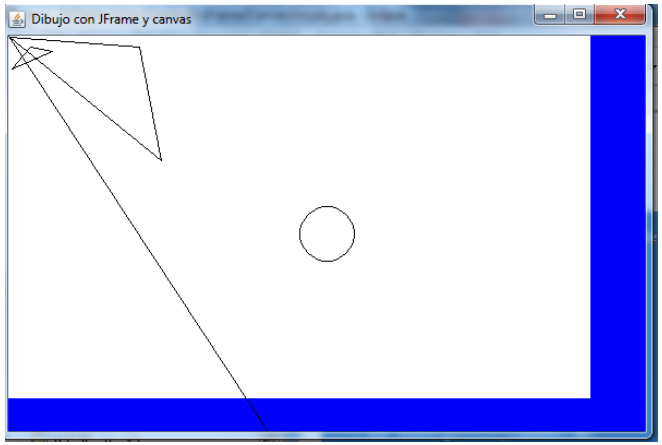
# Menú: JList

```
public class ListaOpciones extends JPanel {
String opciones[]={"Opc0","Opc1", "Opc2", "Opc3","Opc4", "Opc5", "Opc6"};
 JList lista;
ListaOpciones(){
  lista= new JList(opciones);
  lista.addMouseListener(new MouseAdapter() {
    public void mouseClicked(MouseEvent e)  {
       if (e.getClickCount() == 2){ // Sólo si hay doble click
          int posicion = lista.locationToIndex(e.getPoint());
          System.out.println("La posicion es " + posicion);   }    }   });
  lista.addMouseListener(new MouseAdapter() {
  public void mouseEntered(MouseEvent e)  {
       System.out.println("Cursor dentro de la lista");    }   });
  ScrollPane zonaConScroll= new ScrollPane ();
  zonaConScroll.add(lista);
  add(zonaConScroll);
}
}
```



# Zona de dibujo: Canvas

```
import java.awt.Color;
public class Ejem4 extends JFrame {
        CanvasSimple miObjCanvas;
public Ejem4() {
  super ("Dibujo con JFrame y canvas");
  miObjCanvas = new CanvasSimple( Color.blue );

  add( miObjCanvas,"Center" );
  setSize(600,400);
  setVisible( true );
  setDefaultCloseOperation(EXIT_ON_CLOSE);
}
  void dibujar(){
         miObjCanvas.dibujar();
  }
public static void main(String args[]) {
        Ejem4 aplicacion =new Ejem4();
        aplicacion.dibujar();
}
}
```

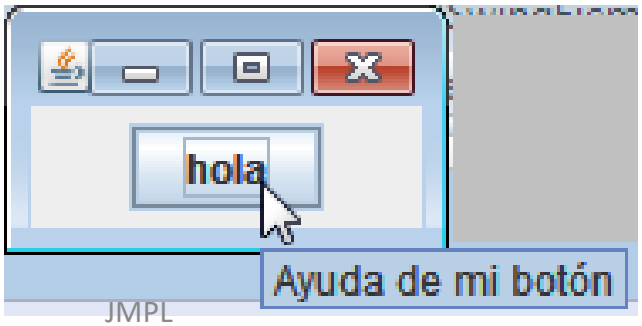



```
public class CanvasSimple extends Canvas {
   public CanvasSimple(Color color) {  //defino el color
      setBackground(color );
   }
   void dibujar( ){
    repaint();
   }

public void paint(Graphics g) {
  int radio=25;
System.out.println("usocanvas.paint");
// Creación de la zona de dibujo en blanco
  g.setColor(Color.white);
  int width=getSize().width- 50;
  int height=getSize().height-30;
  g.fillRect(0, 0, width, height);  //Pintamos de blanco un
rectángulo

  g.setColor(Color.black);
  int x1=getSize().width/2;
  int y1=getSize().height/2;
  g.drawOval(x1-radio, y1-radio, 2*radio, 2*radio);

  g.drawLine(0, 0, height-1, width-30);
  int coordX[]={3,40,20};
  int coordY[]={30,14,10};
  g.drawPolygon(coordX, coordY, 3);

    //Dibujamos un triángulo.
  int coordX2[]={1,140,120};
  int coordY2[]={1,114,10};
     g.drawPolygon(coordX2, coordY2, 3);
System.out.println("usocanvas.paint- fin");
    }
}
```

# Tooltips

- Ayuda para los usuarios
- Se puede usar en cualquier JComponent:

```
public class PanelAyuda extends JPanel {
    public PanelAyuda (String msg){
        JButton miBoton= new JButton ("hola");
        miBoton.setToolTipText(msg);
        add(miBoton);  //Añado el botón al panel
    }
}
```



# JDialog

- Los dos tipos de ventanas principales que tenemos en java para aplicaciones son **JFrame** y **JDialog**.
- Si instanciamos un **JFrame**, en la barra de herramientas de windows nos aparece un nuevo icono correspondiente a nuestra aplicación. Si instanciamos un **JDialog**, no aparece nada.
- Un **JFrame** tiene un método **setIconImage()** para cambiar el icono por defecto de la taza de café. **JDialog** no tiene este método.
- Un **JDialog** admite otra ventana (**JFrame** o **JDialog**) como padre en el constructor. **JFrame** no admite padres.
- Un **JDialog** puede ser modal, un **JFrame** no.
- En resumen:

  **Un JFrame debe ser la ventana principal** de nuestra aplicación

  Todos los **JDialog** modales visibles a la vez en pantalla, deben ser unos hijos de otros en una cadena "padre-hijo-nieto" directa. No deben ser "hermanos".

# Ejemplo JDialog

```
public class VariasVentanasDialogos extends JFrame {
private JButton btnMensaje;
private JButton btnEntrada;
private JButton btnConfirma;
private JLabel lblNombre;

  public VariasVentanasDialogos()  {
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setTitle("Probando Diálogos predefinidos");
    this.setSize(new Dimension(400, 250));

    btnMensaje = new JButton("Saludar");
    btnEntrada = new JButton("Ingresar mi nombre");
    btnConfirma = new JButton("Salir");
    lblNombre = new JLabel();

    Container contenedor = this.getContentPane();

    lblNombre.setHorizontalAlignment(SwingConstants.CENTER);
    lblNombre.setFont(new Font("Arial", Font.BOLD, 16));
    lblNombre.setForeground(Color.BLUE);

    JPanel panelNorte = new JPanel();
    panelNorte.setLayout(new FlowLayout());
    panelNorte.add(btnMensaje);
    panelNorte.add(btnEntrada);
    panelNorte.add(btnConfirma);

    contenedor.add(panelNorte, BorderLayout.NORTH);
    contenedor.add(lblNombre, BorderLayout.SOUTH);

    // Manejando eventos
    ActionListener al = new ActionListener() {
      public void actionPerformed(ActionEvent evt) {
        Object obj = evt.getSource();
        if (obj == btnMensaje)
          btnMensajeActionPerformed(evt);
        else if (obj == btnEntrada)
          btnEntradaActionPerformed(evt);
        else if (obj == btnConfirma)
          btnConfirmaActionPerformed(evt);
      }
    };
    btnMensaje.addActionListener(al);
    btnEntrada.addActionListener(al);
    btnConfirma.addActionListener(al);
  }
  private void btnMensajeActionPerformed(ActionEvent evt) {
    // Centro del marco padre
    JOptionPane.showMessageDialog(this, "Hola, soy un cuadro modal",
        "Saludo", JOptionPane.INFORMATION_MESSAGE);
  }
  private void btnEntradaActionPerformed(ActionEvent evt) {
    // Centro del marco por omisión
    String nombre = JOptionPane.showInputDialog(null, "Ingrese su nombre
        por favor");
    if (nombre != null && !nombre.isEmpty())
      lblNombre.setText("Hola, " + nombre);
  }
  private void btnConfirmaActionPerformed(ActionEvent evt)  {
    int respuesta = JOptionPane.showConfirmDialog(this, "¿Está seguro que
        desea salir?", "Confirmar salida", JOptionPane.YES_NO_OPTION);
    if (respuesta == 0)
      System.exit(0);
  }
  public static void main(String[] args){
    new VariasVentanasDialogos().setVisible(true);
  }
}
```

# Modalidad en ventanas

La modalidad de una ventana se refiere a como mantiene ésta el foco respecto a las demás ventanas del sistema. De acuerdo a su modalidad se clasifican en:

- **Ventana no modal** Permite alternar el foco a cualquier otra ventana presente dentro del entorno gráfico. Es el tipo más común de modalidad.
- **Ventana modal respecto a una aplicación** Permite alternar el foco a otras ventanas del sistema, pero no a la ventana que le da origen («ventana madre») hasta que se toma una acción sobre ella. Normalmente se utilizan para confirmar una acción del usuario, un ejemplo típico sería un administrador de archivos que detiene una acción del usuario pidiendo confirmación como «¿Desea eliminar éste archivo? y las opciones de Aceptar y Cancelar (O Borrar / No Borrar)».
- **Ventana modal respecto al sistema** Similar a la anterior, pero no cede el foco a ninguna otra aplicación hasta que se toma determinada acción sobre ella. Por regla general sólo deben surgir cuando existe un evento a nivel del sistema completo que exija atención inmediata del usuario, por ejemplo «¿Desea apagar el equipo?».

Un **JDialog** puede ser modal, pasándole un **true** en el constructor en el sitio adecuado o haciéndolo modal con el método **setModal()**.

# Threads y Swing

- Cuando ejecutas una aplicación en el entorno gráfico se ejecutan 3 hilos de ejecución: el **main**, el **recolector** de basura y el **escuchador** de los eventos de la parte gráfica. Si el escuchador tiene mucho trabajo porque haces cálculos complejos o sleeps en los métodos del mismo entonces se ralentizará la realización de las acciones asociadas a los eventos.
- La mayor parte de los componentes de swing no son thread-safe
- Puede haber problemas si un thread modifica cosas que otro también está utilizando sin que ambos estén sincronizados

- Por ejemplo si un thread modifica un atributo de un componente y otro está dibujando el componente ( el color, la posición, etc.)

- La solución habitual es hacer que todo el código ejecute en un único thread pero a veces no es posible.

# Threads y Swing

- La solución más sencilla es colocar todo el código en un método estático:

```
public static void main(String[] args) {
 SwingUtilities.invokeLater (    // también se puede usar invokeAndWait
  new Runnable() {
     public void run() {
      createAndShowGUI(); // Método inicial de la aplicación
           }
  }  );
}
```

*Para threads adicionales con procesos largos de ejecutar utilizar* javax.swing.SwingWorker.

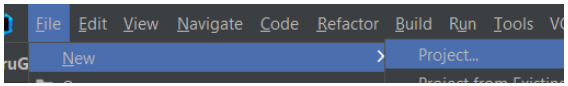
*Tutorial threads in Swing:* *https://docs.oracle.com/javase/tutorial/uiswing/concurrency/*

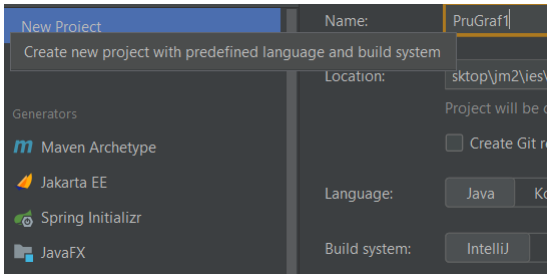
# Hola mundo

```
import javax.swing.*;
public class HelloWorldSwing {
private static void crearyMostrar() {
      JFrame frame = new JFrame("Ventana inicial"); //Creamos la ventana y le ponemos título
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //Al cerrar la ventana finaliza la aplicación
      JLabel label = new JLabel("Hola Mundo"); //Creamos una etiqueta
      frame.getContentPane().add(label); //Añadimos la etiqueta a la ventana.
      frame.pack();  //Ajusta el tamaño del frame  a lo que contiene
      frame.setVisible(true); //Hacemos visible el frame
   }
   public static void main(String[] args) {
      javax.swing.SwingUtilities.invokeLater(new Runnable() {
         public void run() {
            crearyMostrar();
         }
      });
   }
}
```

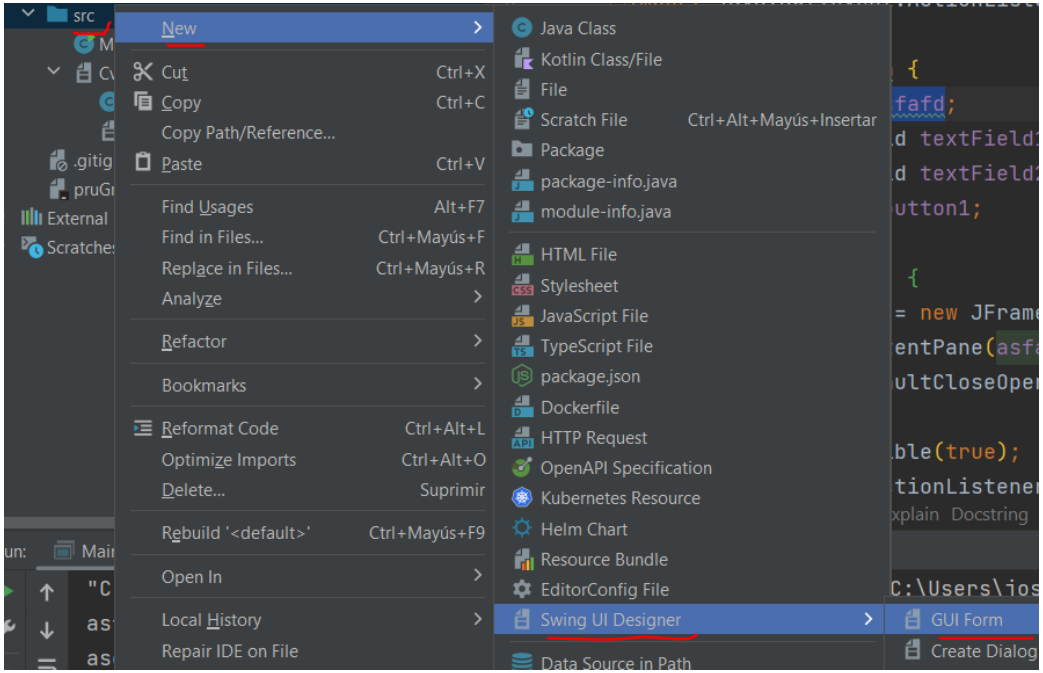
# swing con gui designer of intelliJ

## Creamos el proyecto




## *Creams la clase para la ventana*




[Crear Swing en intelliJ]

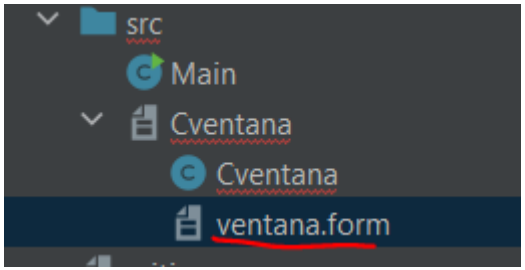
# Código

**Copiamos el código**

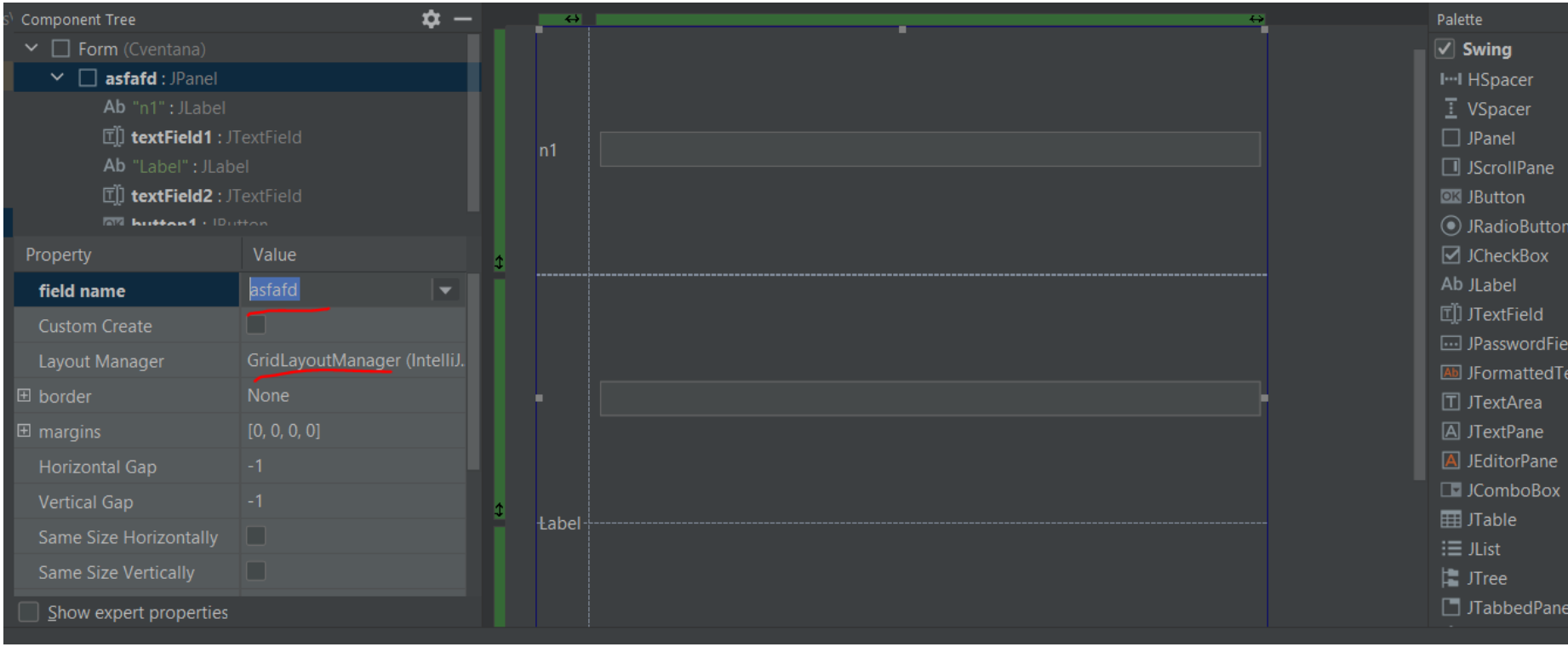
```
public class Main {
    public static void main(String[] args) {
        Cventana ventana = new Cventana();
    }
}
    public class Cventana {
    public Cventana() {
            SwingUtilities.invokeLater(new Runnable() {
                @Override
                public void run() {
                    JFrame frame = new JFrame("Cventana");
                    frame.setContentPane(asfafd);
                    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                    frame.pack();
                    frame.setVisible(true);
    }}}}
```

# Form

**Pulsamos sobre el fichero de form para abrir el GUI**

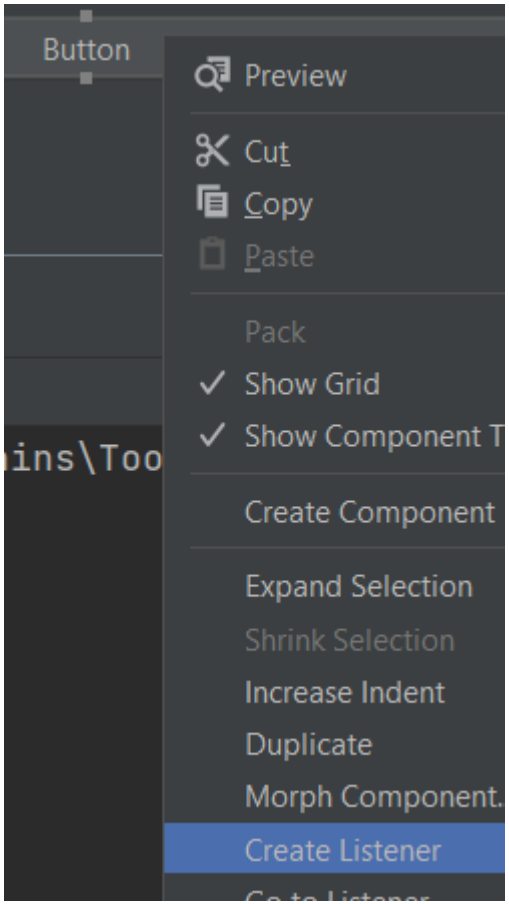



**Seleccionamos un nombre para el panel y un tipo de layout**



# Listeners

**Si añades elementos a los que añadir listener (Jbutton,..) puedes pulsar sobre ellos y elegir el tipo de listener**



# Ver código

Puedes ver el código fuente de lo generado, aunque no podrás modificarlo, seleccionando **Settings | Editor | GUI Designer y elege la opción para generar los ficheros .form en  source code.**