Swing

M.I. Capel

ETS Ingenierías Informática
y Telecommunicación
Universidad de Granada
Email: manuelcapel@ugr.es

Desarrollo de Software

Ingeniería de Software (3er curso de Grado)

Frames & Panels

- Necesitamos siempre una instancia de Frame o de Applet como base si usamos AWT
- Después, se añaden paneles y otros componentes al código de una interfaz en Java
- La base del código utilizando Swing es programar un **JFrame** o **JApplet**

Jerarquía de objetos dentro de una instancia de JFrame

```
1 JFrame
2       JRootPane
3             glassPane
4             layeredPane
5                   [menuBar]
6                   contentPane
```

Frames & Panels

- Necesitamos siempre una instancia de Frame o de Applet como base si usamos AWT
- Después, se añaden paneles y otros componentes al código de una interfaz en Java
- La base del código utilizando Swing es programar un **JFrame** o **JApplet**

Jerarquía de objetos dentro de una instancia de JFrame

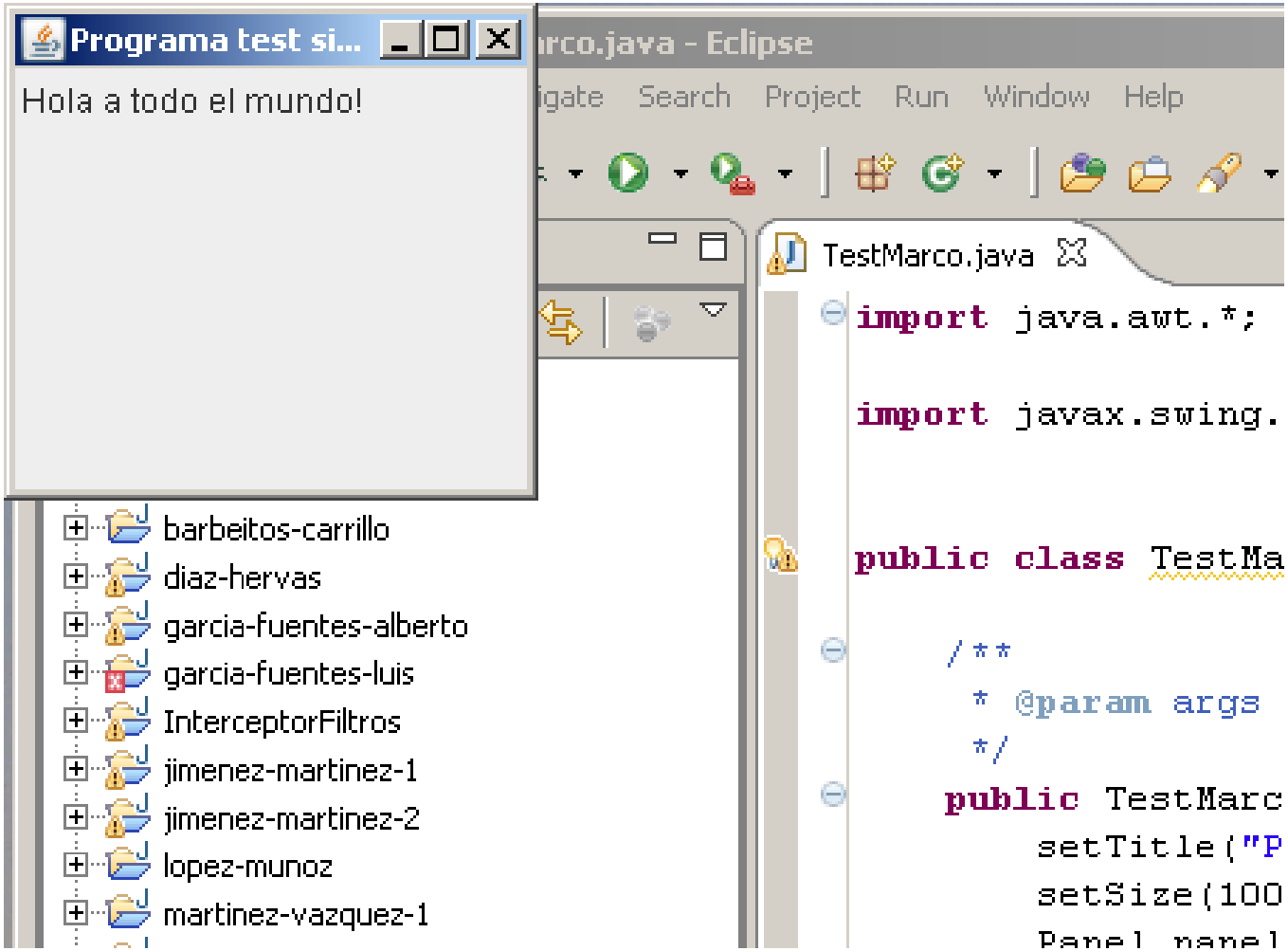
```
1 JFrame
2       JRootPane
3             glassPane
4             layeredPane
5                   [menuBar]
6                   contentPane
```

Ejemplo de código con JFrame

Necesitamos utilizar una instancia de **JFrame** para construir la ventana primaria de una aplicación mejor que usar un **applet**

```
1  import java.awt.*;//utilizando objetos Panel y Label de AWT
2  import javax.swing.*;
3  class TestFrame extends JFrame{
4      public TestFrame(){
5        setTitle("Programa test simple");
6        setSize(100,100);
7        Panel panelSuperior= new Panel();
8        panelSuperior.setLayout(new BorderLayout());
9        getContentPane().add(panelSuperior);//para acceder a
             JRootPane
10       Label etiquetaHola= new Label("Hola a todo el mundo!");
11       panelSuperior.add(etiquetaHola, BorderLayout.NORTH);
12     }
13     public void main(String args[]){
14       TestFrame marcoPrincipal= new TestFrame();
15       marcPrincipal.setVisible( true );
16     }
```

Test simple de un JFrame



Variables de **JFrame**

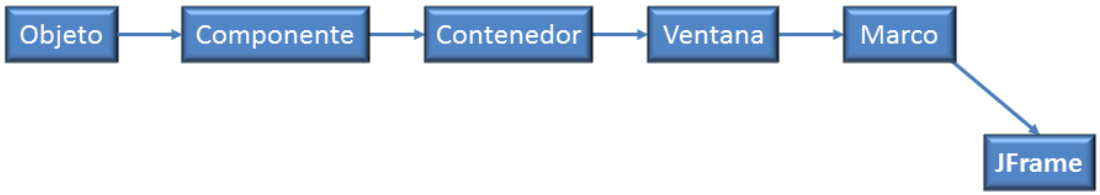


Figure: jerarquía de JFrame

Variables

- `protected JRootPane rootpane`
- `protected booolean canAdd`
- `protected AccessibleContext accessibleContext`

Constructores y otros métodos de JFrame

Constructores

- `JFrame()` y `JFrame(String titulo)`
- Crean instancias, inicialmente invisibles, de **JFrame**; el argumento proporciona un título al marco

Barra de menú de aplicación

- **JFrame** sólo soporta una barra de este tipo:

```
public void setJMenuBar( JMenuBar menu );
```

Constructores y otros métodos de JFrame

Constructores

- `JFrame()` y `JFrame(String titulo)`
- Crean instancias, inicialmente invisibles, de **JFrame**; el argumento proporciona un título al marco

Barra de menú de aplicación

- **JFrame** sólo soporta una barra de este tipo:

```
public void setJMenuBar( JMenuBar menu );
```

Constructores y otros métodos de JFrame II

Métodos

```
1 protected JRootPane createRootPane();
2 protected void setRootPane(JRootPane raiz);
3 public JRootPane getRootPane();
4 public Container getContentPane();
5 public void setLayeredPane(JLayeredPane panelInvisible);
6 public JLayeredPane getLayeredPane();
7 public void setGlassPane(Component cristal);
8 public Component getClassPane();
```

Applets

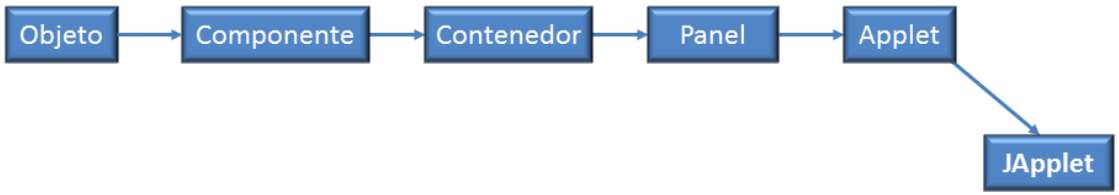


Figure: Jerarquía del elemento **JApplet**

Características

- Posee extensiones que permiten interponer entradas y otros comportamientos de dibujo complejos
- Permite la adición de barras de herramientas y de menús

Applets

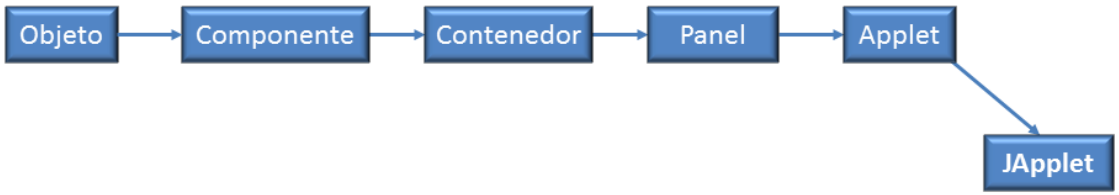


Figure: Jerarquía del elemento **JApplet**

Características

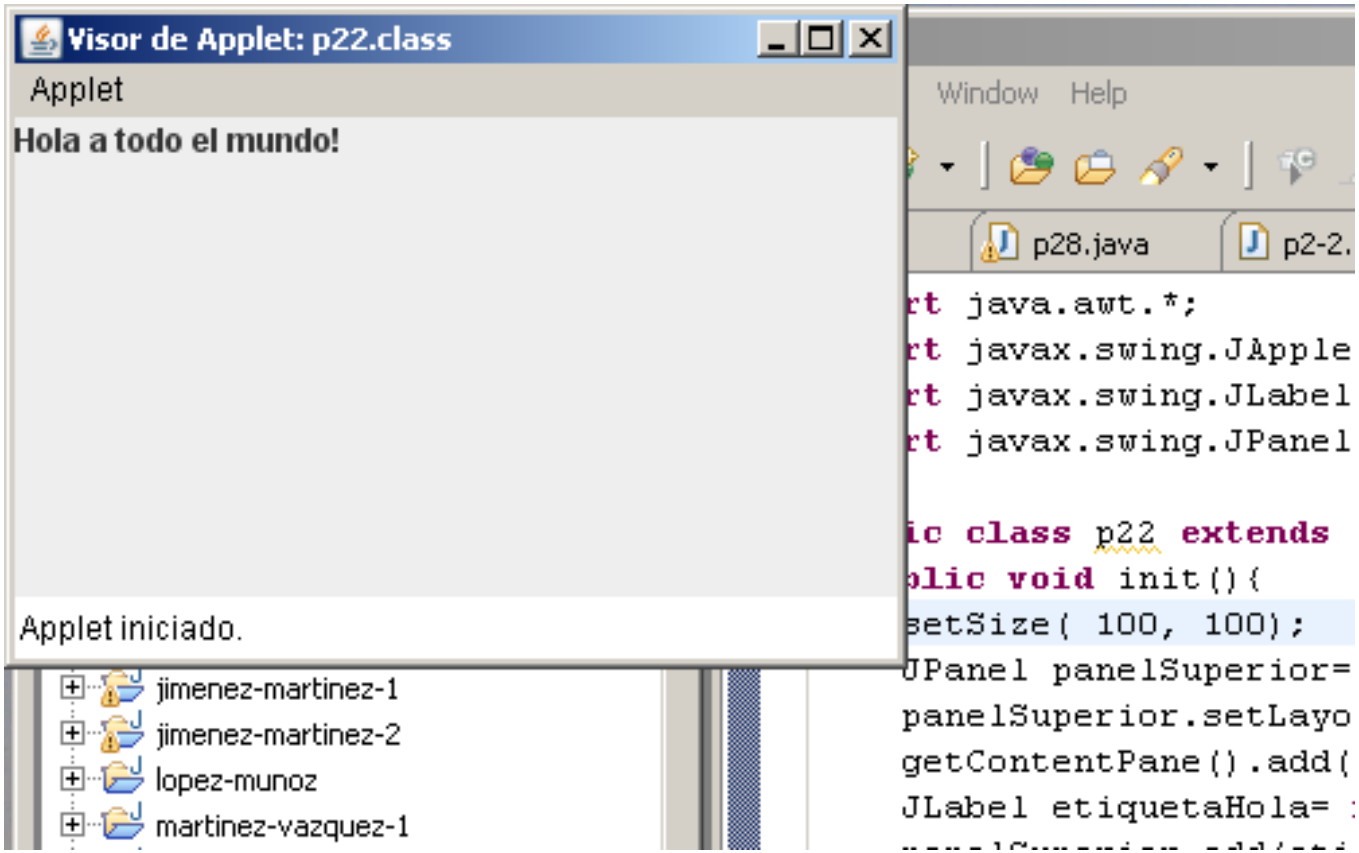
- Posee extensiones que permiten interponer entradas y otros comportamientos de dibujo complejos
- Permite la adición de barras de herramientas y de menús

Código de **applet**

Ejecutable con un AppletViewer o un navegador

```
1  import java.awt.*;//utilizando objetos Panel y Label de AWT
2  import javax.swing.*;
3  class TestApplet extends JApplet{
4     public TestApplet(){
5        }
6     public void init(){
7       setSize( 100, 100);
8       JPanel panelSuperior= new JPanel();
9       panelSuperior.setLayout(new BorderLayout());
10      getContentPane().add(panelSuperior);//para acceder a
            JRootPane
11      JLabel etiquetaHola= new JLabel("Hola a todo el mundo!");
12      panelSuperior.add(etiquetaHola, BorderLayout.NORTH);
13    }
14 }
```

Test simple de un Applet



Métodos constructor y de barras de herramientas

Para asociar barras de menús a un **Applet**

```
1 public void setJMenuBar( JMenuBar menu);
2 public JMenuBar getJMenuBar();
```

Paneles asociados a un **Applet**

```
1 public void setContentPane(Container panelContenedor);
2 public Container getContentPane();
3 public void setLayeredPane(JLayeredPane panelInvisible);
4 public JLayeredPane getLayeredPane();
5 public void setGlassPane(Component cristal);
6 public Component getClassPane();
7 protected JRootPane createRootPane();
8 protected void setRootPane(JRootPane raiz);
9 public JRootPane getRootPane();
```

Métodos constructor y de barras de herramientas

Para asociar barras de menús a un **Applet**

```
1 public void setJMenuBar( JMenuBar menu);
2 public JMenuBar getJMenuBar();
```

Paneles asociados a un **Applet**

```
1 public void setContentPane(Container panelContenedor);
2 public Container getContentPane();
3 public void setLayeredPane(JLayeredPane panelInvisible);
4 public JLayeredPane getLayeredPane();
5 public void setGlassPane(Component cristal);
6 public Component getClassPane();
7 protected JRootPane createRootPane();
8 protected void setRootPane(JRootPane raiz);
9 public JRootPane getRootPane();
```

Panels

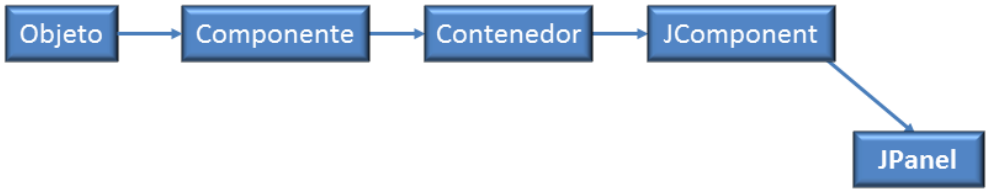


Figure: Jerarquía del elemento **JPanel**

Características

- Los JPanel poseen buferización doble por defecto
- Constructores crean **JPanel** con diseños gobernados por un `LayoutManager` y/o con doble búfer desactivado
- A los **JPanel** se le pueden asociar distintos tipos de bordes

Panels

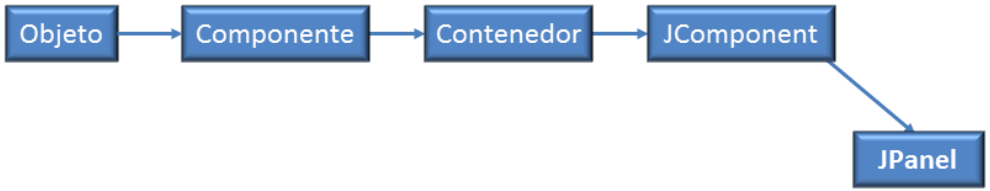


Figure: Jerarquía del elemento **JPanel**

Características

- Los JPanel poseen buferización doble por defecto
- Constructores crean **JPanel** con diseños gobernados por un `LayoutManager` y/o con doble búfer desactivado
- A los **JPanel** se le pueden asociar distintos tipos de bordes

Panels II

Constructores

- `JPanel(LayoutManager d, boolean bActivado);`
- `JPanel( LayoutManager diseno);`
- `JPanel( boolean dobleBuferActivado );`
- `JPanel();`

Tipos de bordes simples

Los bordes se pueden aplicar a cualquier componente de Swing.

`BevelBorder`	Borde 3D con aspecto alzado o deprimido
`CompoundBorder`	Borde compuesto de otros anidados
`EmptyBorder`	Permite especificar un espacio reservado
`EtchedBorder`	Borde como linea grabada con un punzón
`LineBorder`	Línea borde de un color y grosor arbitrario
`MatteBorder`	Permite rodear con iconos
`TitleBorder`	Permite un “string" de título en el borde

Table: Diferentes tipos de bordes en Swing

Ejemplos de creación de bordes

Utilizando el método `setBorder(...)`

```
1 import javax.swing.*;
2 import javax.swing.border.*;
3 {
4  JPanel miPanel= new JPanel();
5  miPanel.setBorder( new BevelBorder ( BevelBorder.RAISED ));
6  }
```

Utilizando la factoría de bordes de Swing

```
1 import javax.swing.*;
2 import javax.swing.border.*;
3 {
4  JPanel miPanel= new JPanel();
5  miPanel.setBorder( BorderFactory.createRaisedBevelBorder() );
6  }
```

Gestores de diseño

Los gestores de diseño son clases proporcionadas para manejar cómo aparecen mostrados los componentes gráficos (UI) en un panel de Java

Propios de AWT

- `BorderLayout`
- Los paneles configurados con este gestor añaden componentes indicando una posición “geográfica" dentro del panel: NORTH, SOUTH, EAST, WEST, CENTER

Propios de Swing

`BoxLayout`	Alinea a lo largo del eje X ó Y de un Panel
`OverlayLayout`	Ordena los componentes uno encima de otro, alineado el punto base con una posición
`ScrollPanelLayout`	Específico para paneles desplazantes

Gestores de diseño

Los gestores de diseño son clases proporcionadas para manejar cómo aparecen mostrados los componentes gráficos (UI) en un panel de Java

Propios de AWT

- `BorderLayout`
- Los paneles configurados con este gestor añaden componentes indicando una posición “geográfica" dentro del panel: NORTH, SOUTH, EAST, WEST, CENTER

Propios de Swing

`BoxLayout`	Alinea a lo largo del eje X ó Y de un Panel
`OverlayLayout`	Ordena los componentes uno encima de otro, alineado el punto base con una posición
`ScrollPanelLayout`	Específico para paneles desplazantes

Gestores de diseño

Los gestores de diseño son clases proporcionadas para manejar cómo aparecen mostrados los componentes gráficos (UI) en un panel de Java

Propios de AWT

- `BorderLayout`
- Los paneles configurados con este gestor añaden componentes indicando una posición “geográfica" dentro del panel: NORTH, SOUTH, EAST, WEST, CENTER

Propios de Swing

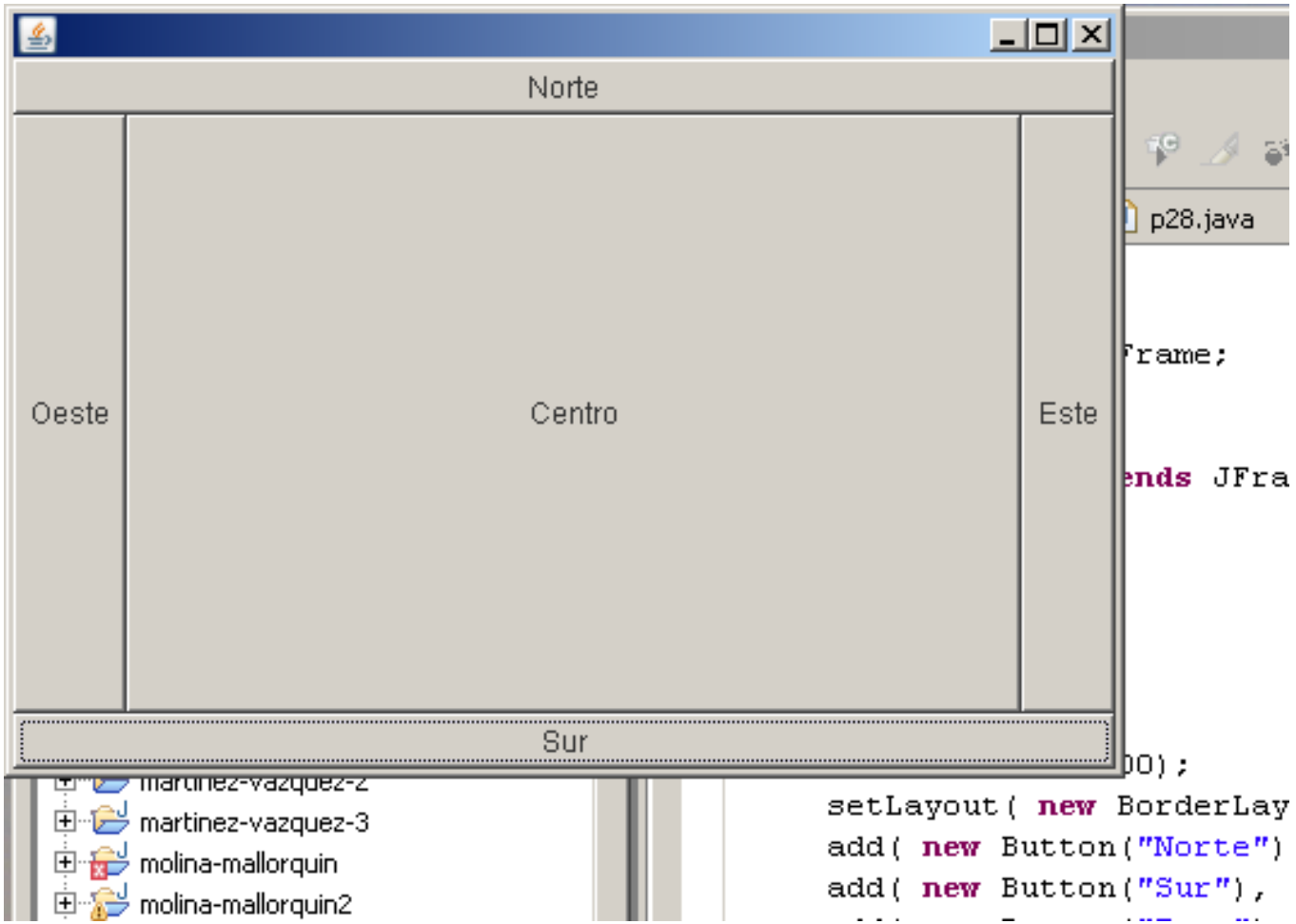
`BoxLayout`	Alinea a lo largo del eje X ó Y de un Panel
`OverlayLayout`	Ordena los componentes uno encima de otro, alineado el punto base con una posición
`ScrollPanelLayout`	Específico para paneles desplazantes

Ejemplos con gestores de diseño

5 botones con AWT en cada posición válida

```
1  import java.awt.*;
2  class TestFrame extends Frame{
3    public TestFrame(){
4      super();
5      setSize( 200, 200);
6      setLayout( new BorderLayout() );
7      add( new Button("Norte"), BorderLayout.NORTH);
8      add( new Button("Sur"), BorderLayout.SOUTH);
9      add( new Button("Este"), BorderLayout.EAST);
10     add( new Button("Oeste"), BorderLayout.WEST);
11     add( new Button("Centro"), BorderLayout.CENTER);
12   }
13 }
```

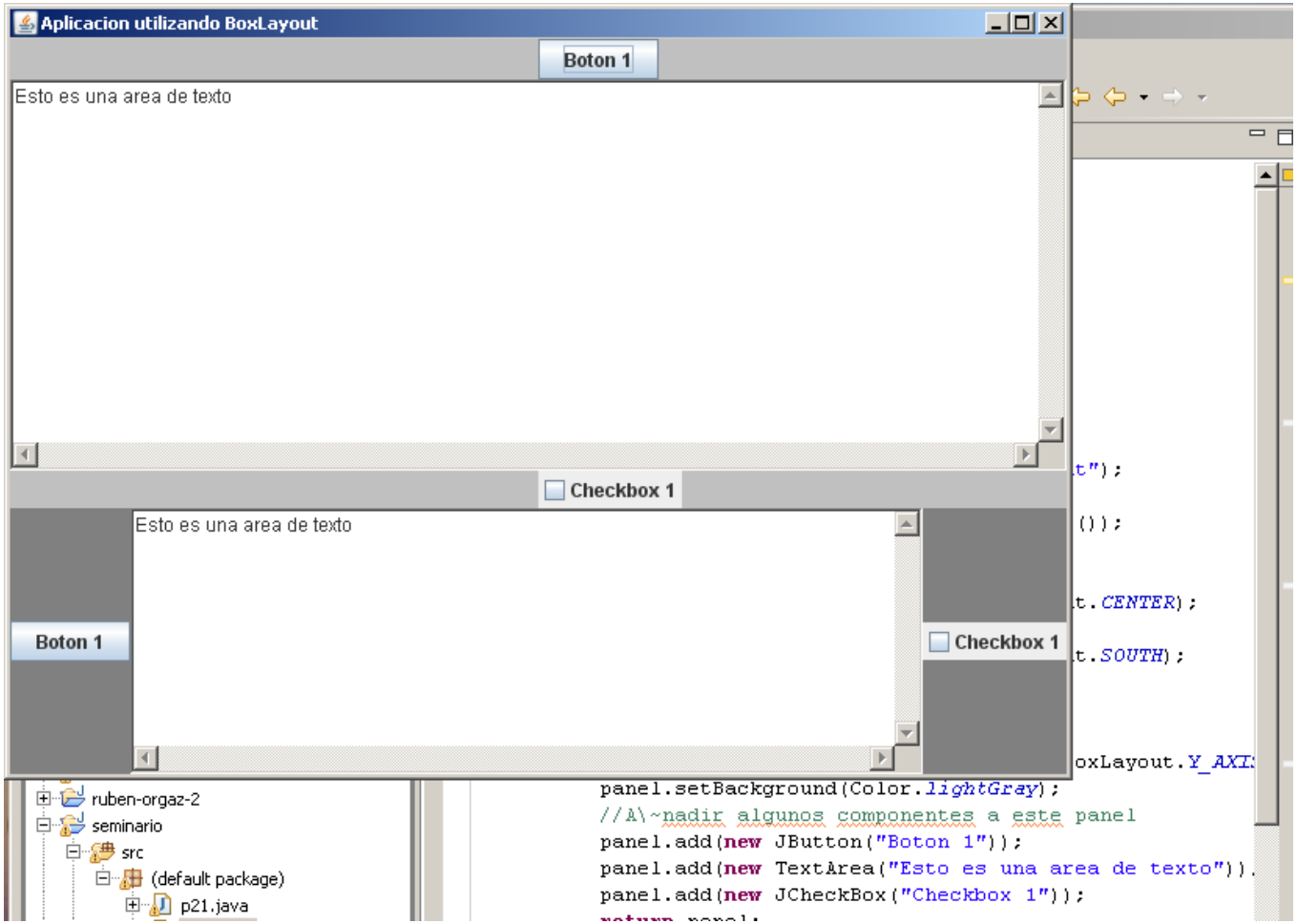
Ejemplo de panel gestionado por **BorderLayout**



Diseño utilizando `BoxLayout`

```
1  import java.awt.*;
2  import javax.swing.*;
3  class TestFrame extends JFrame{
4    public TestFrame(){
5      setTitle("Aplicacion utilizando BoxLayout");
6      JPanel panelSuperior= new JPanel();
7      panelSuperior.setLayout(new BorderLayout());
8      getContentPane().add(panelSuperior);
9      JPanel ejeYPanel= createEjeYPanel();
10     panelSuperior.add(ejeYPanel, BorderLayout.CENTER);
11     JPanel ejeXPanel= createEjeXPanel();
12     panelSuperior.add(ejeXPanel, BorderLayout.SOUTH);
13   }
14   public JPanel createEjeYPanel(){ ...}
15   public JPanel createEjeXPanel(){ ...}
16   public static void main(String args[] ){
17     TestFrame marcoPrincipal= new TestFrame();
18     marcoPrincipal.pack();
19     marcoPrincipal.setVisible(true);
20   }
21 }
```

Ejemplo de panel gestionado por **BoxLayout**



JLabel

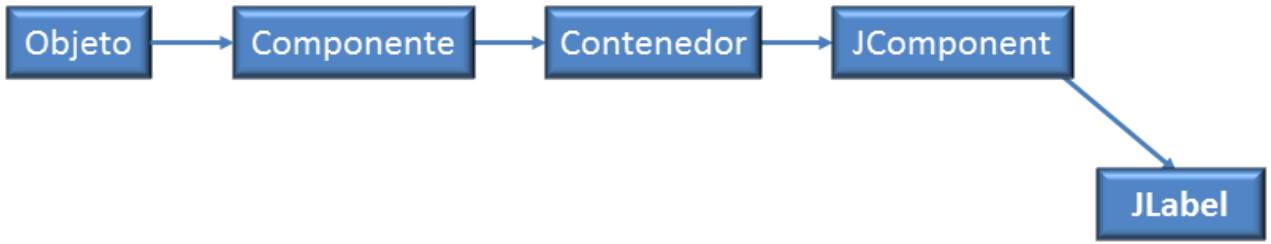


Figure: Jerarquía del elemento **JLabel**

- Es la forma más simple de componentes de una interfaz gráfica de usuario
- Son cadenas de caracteres que se usan para identificar a otros componentes de la interfaz
- **AWT** posee una clase **Label** y la clase **JLabel** de Swing es sólo un *wrapper* de la clase anterior, pero con una API más completa
- **JLabel** es una subclase de **JComponent**

JLabel

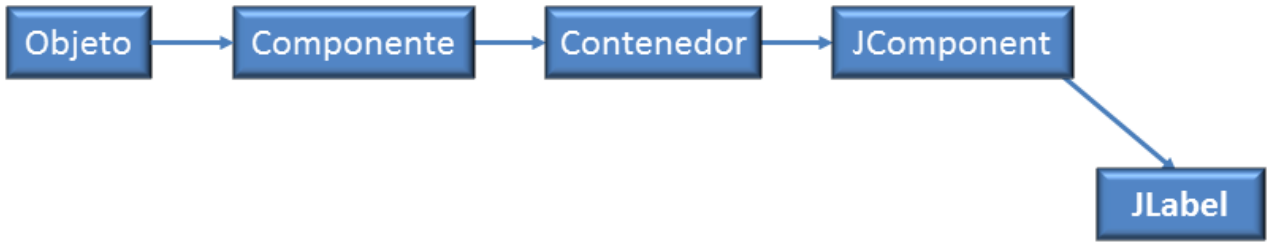


Figure: Jerarquía del elemento **JLabel**

- Es la forma más simple de componentes de una interfaz gráfica de usuario
- Son cadenas de caracteres que se usan para identificar a otros componentes de la interfaz
- **AWT** posee una clase **Label** y la clase **JLabel** de Swing es sólo un *wrapper* de la clase anterior, pero con una API más completa
- **JLabel** es una subclase de **JComponent**

Mostrar texto en componentes **JLabel**

Fuentes y colores

- El *fuente* de los caracteres de texto se cambia creando el fuente y ajustando el componente etiqueta después

```
1 etiqueta.setFont(new Font("Dialog", Font.PLAIN, 12));
```

- Los colores de fondo y de primer plano de una etiqueta también se cambian con los métodos apropiados:

```
1 etiqueta.setBackground( Color.blue );
2 etiqueta.setForeground( Color.yellow );
```

Mostrar texto en componentes **JLabel** II

Alineamiento del texto

Valor Constante	Propósito
SwingConstant.LEFT	Alineamiento horizontal izquierdo
SwingConstant.CENTER	Centrado horizontal o verticalmente
SwingConstant.RIGHT	Alineamiento horizontal derecho
SwingConstant.TOP	Alineamiento vertical arriba
SwingConstant.BOTTOM	Alineamiento vertical abajo

Campos importantes de **JLabel**

Variables de JLabel

```
1 protected Component labelFor
```

Contiene la instancia del componente asociado al objeto etiqueta

JLabel(String texto,Icon imagen, int hAlin);

crea una instancia con el texto, ícono y alineamiento horizontal que se pasa en los argumentos

JLabel();

crea una instancia sin texto, ni ícono y alineamiento horizontal por defecto, justificado a la derecha

Campos importantes de **JLabel**

Variables de JLabel

```
1 protected Component labelFor
```

Contiene la instancia del componente asociado al objeto etiqueta

JLabel(String texto,Icon imagen, int hAlin);

crea una instancia con el texto, ícono y alineamiento horizontal que se pasa en los argumentos

JLabel();

crea una instancia sin texto, ni ícono y alineamiento horizontal por defecto, justificado a la derecha

Campos importantes de **JLabel**

Variables de JLabel

```
1 protected Component labelFor
```

Contiene la instancia del componente asociado al objeto etiqueta

JLabel(String texto,Icon imagen, int hAlin);

crea una instancia con el texto, ícono y alineamiento horizontal que se pasa en los argumentos

JLabel();

crea una instancia sin texto, ni ícono y alineamiento horizontal por defecto, justificado a la derecha

Campos importantes de **JLabel** II

Variaciones de los anteriores

- **JLabel(String texto,int hAlin);**
- **JLabel(Icon imagen, int hAlin);**
- **JLabel(String texto);**
- **JLabel(Icon imagen);**

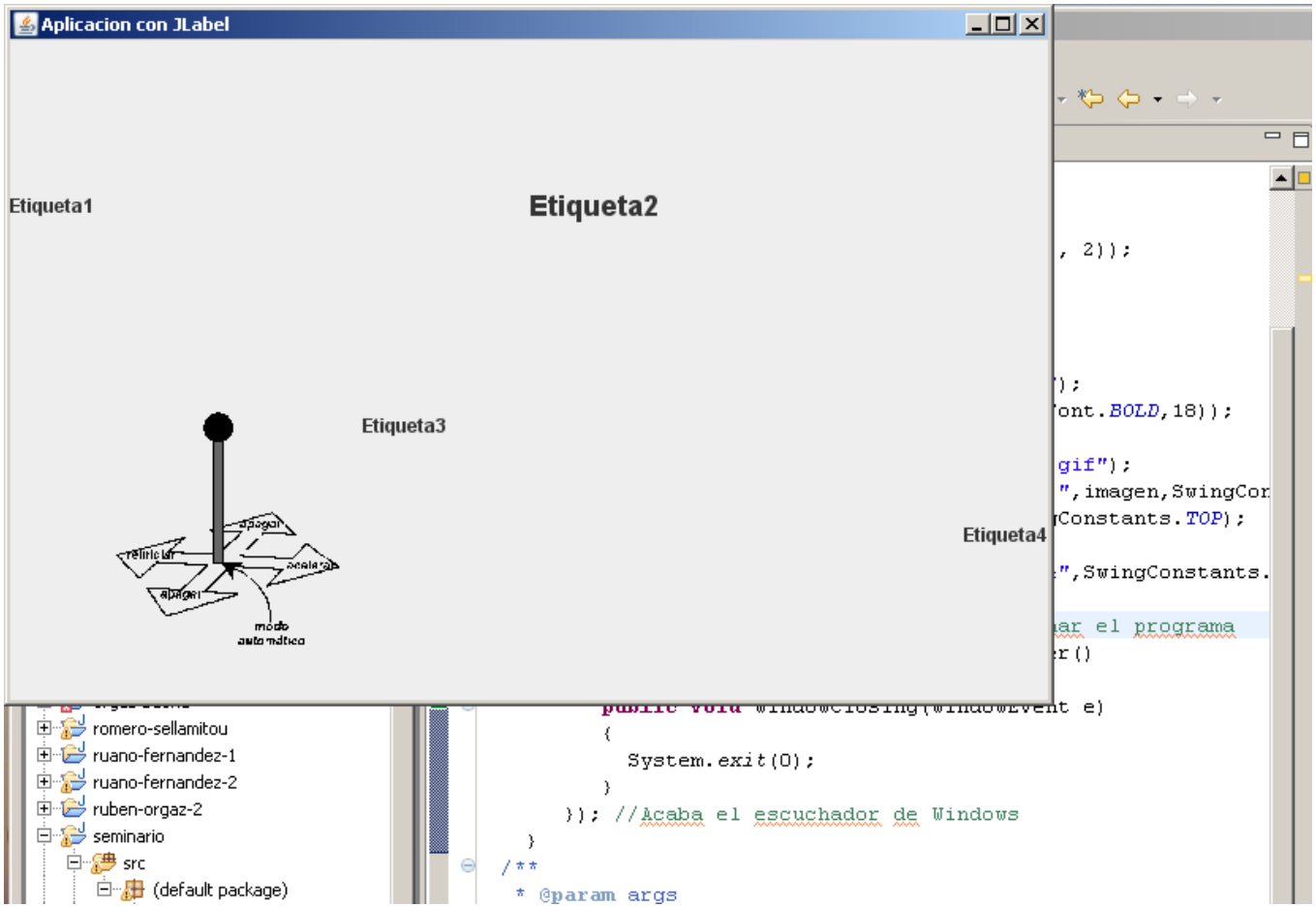
Ejemplo de programa con *etiquetas*

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  class TestFrame extends {
5    public TestFrame(){
6      setTitle("Aplicacion con JLabel");
7      setSize(300, 200);
8      JPanel panelSuperior= new JPanel();
9      panelSuperior.setLayout(new GridLayout(2, 2));
10     getContentPane().add( panelSuperior);
11     JLabel etiqueta1= new JLabel();
12     etiqueta1.setText("Etiqueta1");
13     panelSuperior.add( etiqueta1 );
14     JLabel etiqueta2= new JLabel();
15     etiqueta1.setText("Etiqueta2");
16     panelSuperior.add( etiqueta2 );
17     //... continua public TestFrame()
18   }
19 }
```

Ejemplo de programa con *etiquetas* II

```
1       Icon imagen = new ImageIcon( "icono.gif");
2       JLabel etiqueta3= new JLabel ("Etiqueta3",imagen,
            SwingConstants.CENTER);
3       etiqueta3.setVerticalTextPosition( SwingConstants.TOP);
4       panelSuperior.add( etiqueta3 );
5       JLabel etiqueta4= new JLabel ("Etiqueta3",imagen,
            SwingConstants.RIGHT);
6
7       //A\~nadir una clase an\'onima para terminar el programa
8      this.addWindowListener (new WindowAdapter(){ ...});
9     }
10    public static void main(String args[] ){
11      TestFrame marcoPrincipal= new TestFrame();
12      marcoPrincipal.setVisible(true);
13    }
14  }
```

Ejemplo de programa simple con etiquetas



Abstract Buttons

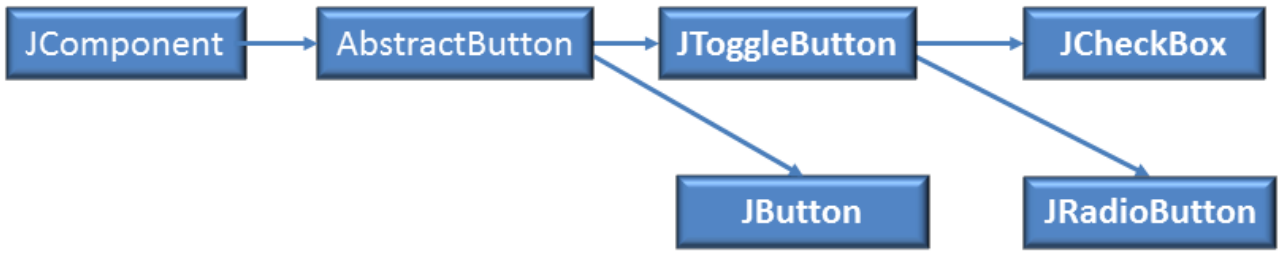


Figure: Jerarquía del elemento **AbstractButton**

- Todos los componentes de Swing derivan de **JComponent**
- Nunca se instanacia **AbstractButton** sino que implementamos con: **JButton** y **JToggleButton**
- Utilidad de **AbstractButton**:
  - Código para adjuntar íconos a los botones
  - Gestionar aceleradores del teclado
  - Alinear el texto

Abstract Buttons

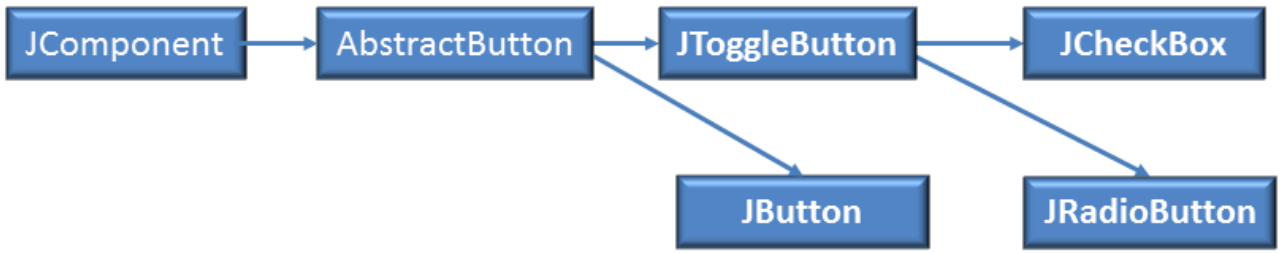


Figure: Jerarquía del elemento **AbstractButton**

- Todos los componentes de Swing derivan de **JComponent**
- Nunca se instanacia **AbstractButton** sino que implementamos con: **JButton** y **JToggleButton**
- Utilidad de **AbstractButton**:
  - Código para adjuntar íconos a los botones
  - Gestionar aceleradores del teclado
  - Alinear el texto

Escuchadores de eventos

ActionListener

- Esta interfaz escucha las pulsaciones del ratón del usuario sobre las superficies de los botones
- Se capturan las acciones programando una clase que implemente esta interfaz
- Hay que programar un método **actionPerformed()** para realizar la acción asociada a la pulsación del botón
- Opciones para los *escuchadores* (**ActionListener**):
  - Implementarlos como clases internas y anónimas: manera más eficiente de gestionar los eventos de los botones
  - En la misma clase que posee la instancia del botón: más fácil entender el código producido por otra persona

Constructores de **JButton**

JButton()

Crea una instancia sin texto ni ícono asociado

JButton(Icon icono)

Crea una instancia con el ícono asociado que se le pasa como argumento

JButton(String texto)

Crea un botón con el texto que se le pasa como argumento

JButton(String texto, Icon icono)

Crea un botón con el texto que se le pasa como argumento y el ícono asociado

Constructores de **JButton**

JButton()

Crea una instancia sin texto ni ícono asociado

JButton(Icon icono)

Crea una instancia con el ícono asociado que se le pasa como argumento

JButton(String texto)

Crea un botón con el texto que se le pasa como argumento

JButton(String texto, Icon icono)

Crea un botón con el texto que se le pasa como argumento y el ícono asociado

Constructores de **JButton**

JButton()

Crea una instancia sin texto ni ícono asociado

JButton(Icon icono)

Crea una instancia con el ícono asociado que se le pasa como argumento

JButton(String texto)

Crea un botón con el texto que se le pasa como argumento

JButton(String texto, Icon icono)

Crea un botón con el texto que se le pasa como argumento y el ícono asociado

Constructores de **JButton**

JButton()

Crea una instancia sin texto ni ícono asociado

JButton(Icon icono)

Crea una instancia con el ícono asociado que se le pasa como argumento

JButton(String texto)

Crea un botón con el texto que se le pasa como argumento

JButton(String texto, Icon icono)

Crea un botón con el texto que se le pasa como argumento y el ícono asociado

Ejemplo de clase *listener*

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  class TestFrame extends JFrame implements ActionListener{
5    private int contadorI= 0; //contar las pulsaciones
6    private JButton button=null;//un lugar donde guardar el boton
7    public TestFrame(){
8      setTitle("Aplicacion para implementar un ActionListener");
9      JPanel panelSuperior= new JPanel();
10     panelSuperior.setLayout(new FlowLayout());
11     panelSuperior.setPreferredSize(new Dimension(300,200));
12     boton= new JButton("Pulsa me");
13     panelSuperior.add(boton);
14     boton.addActionListener( this );
15   }
16   //A\~nadir una clase an\'onima para terminar el programa
17   this.addWindowListener (new WindowAdapter(){ ...});
18 }
```

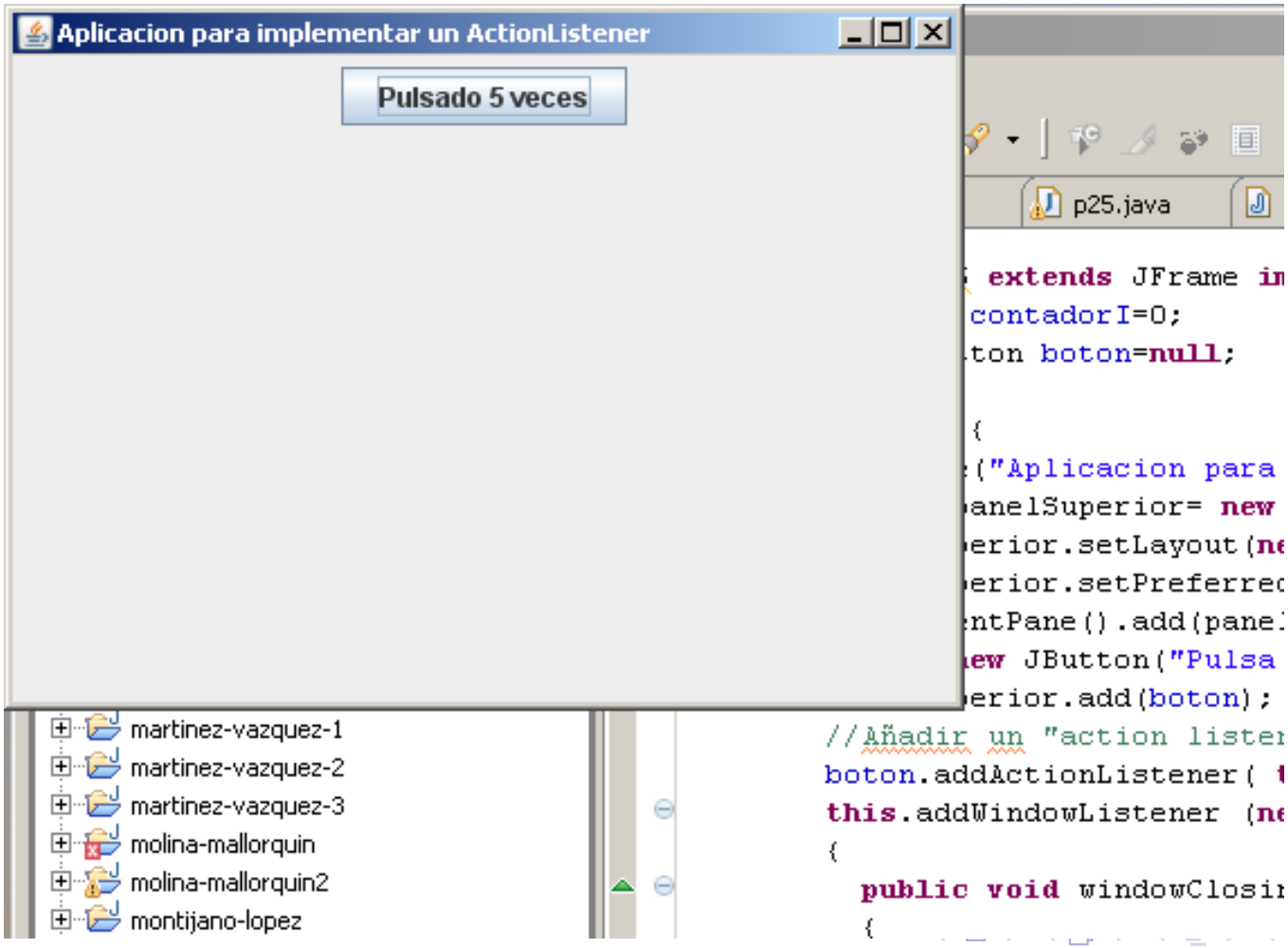
El parámetro **this** indica que el **JFrame** implementa el

Ejemplo de clase *listener*

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  class TestFrame extends JFrame implements ActionListener{
5    private int contadorI= 0; //contar las pulsaciones
6    private JButton button=null;//un lugar donde guardar el boton
7    public TestFrame(){
8      setTitle("Aplicacion para implementar un ActionListener");
9      JPanel panelSuperior= new JPanel();
10     panelSuperior.setLayout(new FlowLayout());
11     panelSuperior.setPreferredSize(new Dimension(300,200));
12     boton= new JButton("Pulsa me");
13     panelSuperior.add(boton);
14     boton.addActionListener( this );
15   }
16   //A\~nadir una clase an\'onima para terminar el programa
17   this.addWindowListener (new WindowAdapter(){ ...});
18 }
```

El parámetro **this** indica que el **JFrame** implementa el

Ejemplo de programa simple con un escuchador de botón



Toggle Buttons

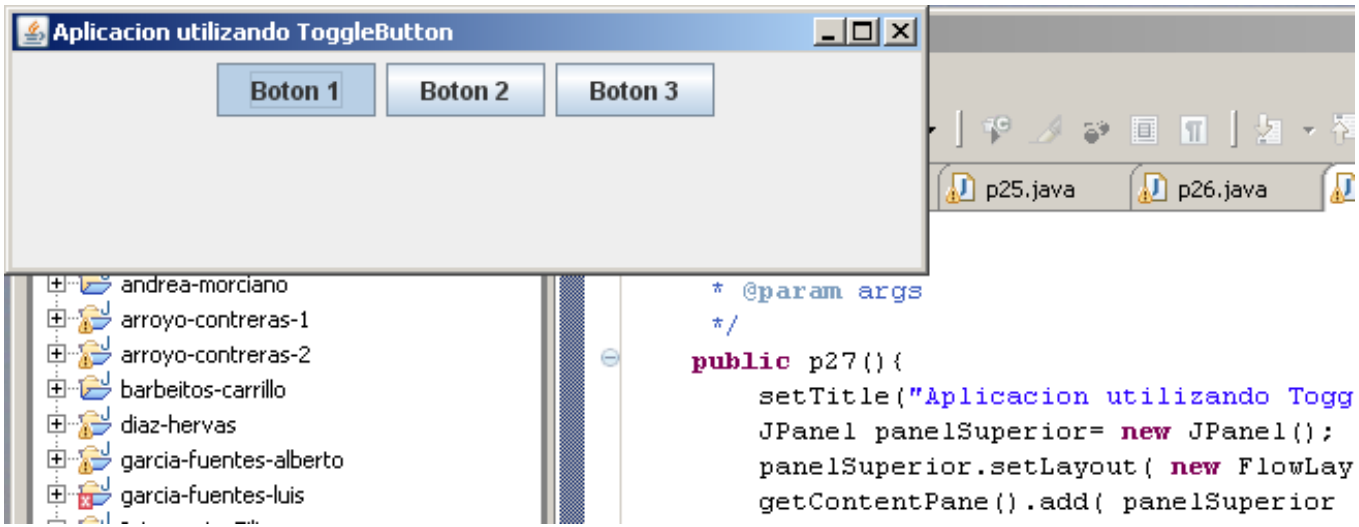
JToggleButton

- Son elementos gráficos de selección de tipo conmutador en interfaces
- Son el objeto gráfico indicado para interfaces de usuario que necesitan presentar operaciones con diferentes modos de funcionamiento
- JToggleButtons pueden incluirse dentro de un objeto **ButtonGroup**
- Los objetos de **JCheckbox** y de **JRadioButton** pertenecen a subclases de **JToggleButton**

Ejemplo de clase con botones conmutadores

```
1  import java.awt.*;
2  import javax.swing.*;
3  class TestFrame extends JFrame{
4    public TestFrame(){
5      setTitle("Aplicacion utilizando ToggleButton");
6      JPanel panelSuperior= new JPanel();
7      panelSuperior.setLayout( new FlowLayout() );
8      getContentPane().add( panelSuperior );
9      JTogglebutton boton1= new JToggleButton("Boton 1",true);
10     panelSuperior.add(boton1);
11     JTogglebutton boton2= new JToggleButton("Boton 2",false);
12     panelSuperior.add(boton2);
13     JTogglebutton boton1= new JToggleButton("Boton 3",false);
14     panelSuperior.add(boton3);
15     //A\~nadir una clase an\'onima para terminar el programa
16   this.addWindowListener (new WindowAdapter(){ ...});
17   }
18 }
```

Ejemplo de programa simple con botones de conmutación



Constructores de **JToggleButton**

JToggleButton()

Crea una instancia sin texto ni ícono asociado

JToggleButton(Icon i, boolean b)

Lo crea con un ícono asociado y un estado inicial de conmutación (sí||no)

JToggleButton(String texto)

Crea un componente *botón conmutable* con el texto que se le pasa como argumento

Combinaciones de los anteriores

- **JToggleButton(String texto, boolean bSeleccionado);**
- **JToggleButton(String texto, Icon icono, boolean bSel);**

Constructores de **JToggleButton**

JToggleButton()

Crea una instancia sin texto ni ícono asociado

JToggleButton(Icon i, boolean b)

Lo crea con un ícono asociado y un estado inicial de conmutación (sí||no)

JToggleButton(String texto)

Crea un componente *botón conmutable* con el texto que se le pasa como argumento

Combinaciones de los anteriores

- **JToggleButton(String texto, boolean bSeleccionado);**
- **JToggleButton(String texto, Icon icono, boolean bSel);**

Constructores de **JToggleButton**

JToggleButton()

Crea una instancia sin texto ni ícono asociado

JToggleButton(Icon i, boolean b)

Lo crea con un ícono asociado y un estado inicial de conmutación (sí||no)

JToggleButton(String texto)

Crea un componente *botón conmutable* con el texto que se le pasa como argumento

Combinaciones de los anteriores

- JToggleButton(String texto, boolean bSeleccionado);
- JToggleButton(String texto, Icon icono, boolean bSel);

Constructores de **JToggleButton**

JToggleButton()

Crea una instancia sin texto ni ícono asociado

JToggleButton(Icon i, boolean b)

Lo crea con un ícono asociado y un estado inicial de conmutación (sí||no)

JToggleButton(String texto)

Crea un componente *botón conmutable* con el texto que se le pasa como argumento

Combinaciones de los anteriores

- **JToggleButton(String texto, boolean bSeleccionado);**
- **JToggleButton(String texto, Icon icono, boolean bSel);**

Check Boxes

JCheckBox

- Extiende a **JToggleButton** para ofrecer a las aplicaciones gráficas el componente “casilla de verificación"
- Determinación y cambio del estado de una *casilla de verificación*:

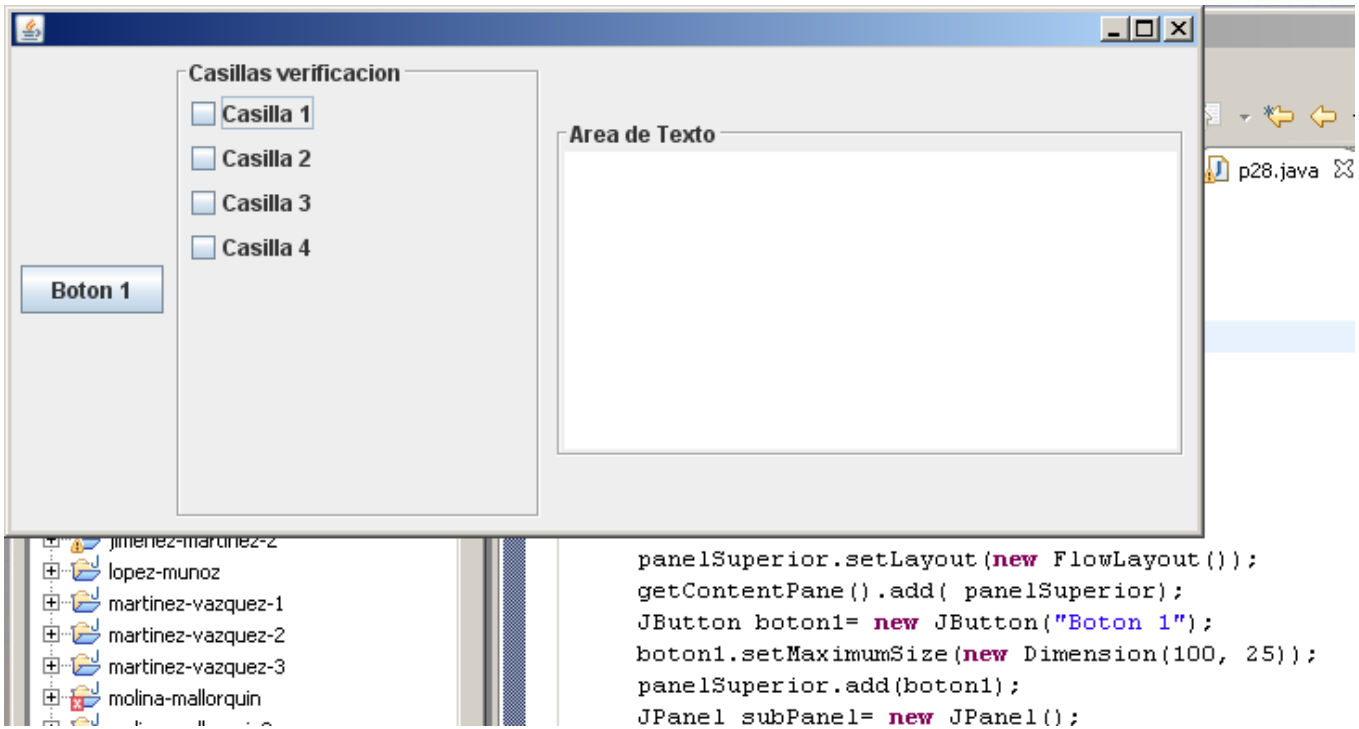
```
1 boolean valorB= checkbox.isSelected();
2 checkbox.setSelected(valorB);
```

  - Utilizar cuando queremos diseñar una interfaz que contenga estados múltiples simultáneos
- El gestor de diseño **BoxLayout** simplica enormemente el mostrar *casillas* dentro de un formato encolumnado
- Para mezclar grupos de *casillas* con *textos*:
  - Crear un subpanel de **JPanel**
  - Utilizar **BoxLayout** para las *casillas de verificación*

Ejemplo de clase con *casillas de verificación*

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import javax.swing.border.*;
5  class TestFrame extends {
6    public TestFrame(){
7     JPanel panelSuperior= new JPanel();
8     panelSuperior.setLayout(new FlowLayout());
9     getContentPane().add( panelSuperior);
10    JButton boton1= new JButton("Boton 1");
11    boton1.setMaximumSize(new Dimension(100, 25));
12    panelSuperior.add(boton1);
13    JPanel subPanel= new JPanel();
14    subPanel.setLayout(new BoxLayout(subPanel,BoxLayout.Y_AXIS));
15    subPanel.setPreferredSize(new Dimension(100,200));
16    subPanel.setBorder(new TitleBorder(
17           new EtchedBorder(), "Casillas verificacion"));
18    panelSuperior.add(subPanel);
19    //creaci\'on de los componentes--casillas
20    //creaci\'on de un panel de texto
21     //A\~nadir una clase an\'onima para terminar el programa
```

Ejemplo de programa simple con casillas de verificación



Constructores de **JCheckBox**

JCheckBox()

Crea una instancia sin texto ni ícono asociado

JCheckBox(Icon i, boolean b)

Lo crea con un ícono asociado y un estado inicial de selección

JCheckBox(String texto)

Crea un componente *casilla de verificación* con el texto que se le pasa como argumento

Combinaciones de los anteriores

- **JCheckBox(String texto, boolean bSeleccionado);**
- **JCheckBox(String texto, Icon icono, boolean bSel);**
- **JCheckBox(String texto, Icon icono);**

Constructores de **JCheckBox**

JCheckBox()

Crea una instancia sin texto ni ícono asociado

JCheckBox(Icon i, boolean b)

Lo crea con un ícono asociado y un estado inicial de selección

JCheckBox(String texto)

Crea un componente *casilla de verificación* con el texto que se le pasa como argumento

Combinaciones de los anteriores

- **JCheckBox(String texto, boolean bSeleccionado);**
- **JCheckBox(String texto, Icon icono, boolean bSel);**
- **JCheckBox(String texto, Icon icono);**

Constructores de **JCheckBox**

JCheckBox()

Crea una instancia sin texto ni ícono asociado

JCheckBox(Icon i, boolean b)

Lo crea con un ícono asociado y un estado inicial de selección

JCheckBox(String texto)

Crea un componente *casilla de verificación* con el texto que se le pasa como argumento

Combinaciones de los anteriores

- **JCheckBox(String texto, boolean bSeleccionado);**
- **JCheckBox(String texto, Icon icono, boolean bSel);**
- **JCheckBox(String texto, Icon icono);**

Constructores de **JCheckBox**

JCheckBox()

Crea una instancia sin texto ni ícono asociado

JCheckBox(Icon i, boolean b)

Lo crea con un ícono asociado y un estado inicial de selección

JCheckBox(String texto)

Crea un componente *casilla de verificación* con el texto que se le pasa como argumento

Combinaciones de los anteriores

- **JCheckBox(String texto, boolean bSeleccionado);**
- **JCheckBox(String texto, Icon icono, boolean bSel);**
- **JCheckBox(String texto, Icon icono);**

Botones Radio

JRadioButton

- Se trata de arrays de botones que se utilizan para seleccionar el modo de funcionamiento de una función específica de una aplicación
- Por tanto, los *botones radio* estarán siempre asociados con una instancia de **ButtonGroup**
- La decisión acerca de cuándo utilizarlos depende de si necesitamos representar estados múltiples excluyentes en una interfaz para que el usaurio seleccione uno de ellos, que deselecciona a los otros
- Dentro de un mismo panel con gestor de diseño **BoxLayout**, podemos mezclar *botones de radio*, *casillas de verificación* y *botones de conmutación* si lo necesitamos

Constructores de **JRadioButton**

JRadioButton()

Crea una instancia sin texto ni ícono asociado

JRadioButton(Icon i, boolean b)

Lo crea con un ícono asociado y un estado inicial de selección del botón de radio

JRadioButton(String texto)

Crea un componente *casilla de verificación* con el texto que se le pasa como argumento

Combinaciones de los anteriores

- **JRadioButton(String texto, boolean bSeleccionado);**
- **JRadioButton(String texto, Icon icono, boolean bSel);**

Constructores de **JRadioButton**

JRadioButton()

Crea una instancia sin texto ni ícono asociado

JRadioButton(Icon i, boolean b)

Lo crea con un ícono asociado y un estado inicial de selección del botón de radio

JRadioButton(String texto)

Crea un componente *casilla de verificación* con el texto que se le pasa como argumento

Combinaciones de los anteriores

- **JRadioButton(String texto, boolean bSeleccionado);**
- **JRadioButton(String texto, Icon icono, boolean bSel);**

Constructores de **JRadioButton**

JRadioButton()

Crea una instancia sin texto ni ícono asociado

JRadioButton(Icon i, boolean b)

Lo crea con un ícono asociado y un estado inicial de selección del botón de radio

JRadioButton(String texto)

Crea un componente *casilla de verificación* con el texto que se le pasa como argumento

Combinaciones de los anteriores

- **JRadioButton(String texto, boolean bSeleccionado);**
- **JRadioButton(String texto, Icon icono, boolean bSel);**

Constructores de **JRadioButton**

JRadioButton()

Crea una instancia sin texto ni ícono asociado

JRadioButton(Icon i, boolean b)

Lo crea con un ícono asociado y un estado inicial de selección del botón de radio

JRadioButton(String texto)

Crea un componente *casilla de verificación* con el texto que se le pasa como argumento

Combinaciones de los anteriores

- **JRadioButton(String texto, boolean bSeleccionado);**
- **JRadioButton(String texto, Icon icono, boolean bSel);**

Componente Interfaz de la práctica 2

```
1  public class Interfaz extends javax.swing.JApplet {
2  //Constructor: Inicializa objetos y lanza las 3 hebras
3    public Interfaz() {
4     PanelBotones Panel1= new PanelBotones();
5     PanelPantalla Panel2= new PanelPantalla(this);
6     ControlVelocidad control=new ControlVelocidad();
7    //Almacenamiento, Eje, Monitorizacion etc., etc.
8     Monitorizacion monot=new Monotorizacion(almacen,eje);
9     Simulacion simulacion=new Simulacion(Panel2,Panel1);
10    Panel1.annadirSimulacion(simulacion);
11    Panel2.annadirSimulacion(simulacion);
12    Panel1.annadirControl(control);
13    Panel2.annadirControl(control);
14    Panel1.annadirMonotorizacion(monot);
15    Panel2.annadirMonotorizacion(monot);
16    Reloj reloj=control.enviarReloj();
17 //... faltan cosas
18       simulacion.start();
19       reloj.start();
20       relojM.start();
21     }
```

Panel de botones

```
1  public class PanelBotones extends javax.swing.JPanel implements
       Observador{
2  //Miembro del paquete Controlador para la pantalla
3    Simulacion simulacion;
4  //Miembros del paquete Controlador para los botones
5      ControlVelocidad control; Freno freno; //etc., etc.
6  //A este metodo lo llama el Constructor
7   private void initComponents() {
8  GrupoPalanca = new javax.swing.ButtonGroup();
9      BotonEncender = new javax.swing.JToggleButton();
10     BotonAcelerar = new javax.swing.JButton();
11     BotonReiniciando = new javax.swing.JRadioButton(); //etc.,
           etc.
12     BotonAcelerar.setText("Acelerar");
13         BotonAcelerar.addActionListener(new java.awt.event.
               ActionListener() {
14             public void actionPerformed(java.awt.event.
                   ActionEvent evt) {
15                 BotonAcelerarActionPerformed(evt); }
16         });
17     add(BotonAcelerar);
```

Panel de botones II

```
1  // Recogedor de eventos del boton acelerar
2  synchronized private void
3     BotonAcelerarActionPerformed(java.awt.event.ActionEvent evt)
          {
4
5           // Incrementamos el acelerador y ponemos a cero el freno
6           acelerador.incrementar();
7           freno.soltarFreno();
8           BotonApagado.setSelected(true);
9          EtiqMostrarEstado.setForeground(new java.awt.Color(255, 0,
               0));
10          EtiqMostrarEstado.setText("APAGADO");
11          palanca.cambiarEstado(palanca.APAGADO);
12       }
```

Panel de botones III

```
1  // Metodo que actualiza desde Observable los parametros del
       panel de botones
2  synchronized public void manejadorEvento(){
3     // Solo activa los botones si motor esta encendido
4   if(motor.leerEstado()){
5                   activar();
6                   apagarParado();
7                   // Actualiza todos los aspectos de la simulacion
8                   double a=acelerador.leerEstado();
9                   double f=freno.leerEstado();
10                  eje.incrementarVueltas(simulacion.
                        calcularRevoluciones(a,f));
11                  acelerador.actualizarAcelerador();
12                  freno.actualizarFreno();
13                  if(palanca.leerEstado()==palanca.MANTENIENDO)
14          }else{
15                  desactivar();
16            }
17      }
```

Simulador

```
1  public class Simulacion extends Thread{
2   private final int INTERVALO=250;
3   private Observable observableEtiquetas;
4   private Observable observableBotones; //etc., etc.
5   public Simulacion(Observador etiquetas,Observador botones) {
6          revolAcum=0;
7          motorEncendido=false;
8          observableEtiquetas=new Observable(etiquetas);
9          observableBotones=new Observable(botones);
10  }
11  public void run(){
12       while(true){
13         try{ sleep(INTERVALO);
14         }catch(java.lang.InterruptedException e){e.
               printStackTrace();}
15         //Actualización de las revoluciones
16         //actualización de las etiquetas
17         observableBotones.notificarObservador();
18         observableEtiquetas.notificarObservador();
19       }
20     }
```

Panel de Etiquetas I

```
1  public class PanelPantalla extends javax.swing.JPanel implements
       Observador{
2    Interfaz interfaz;
3    Simulacion simulacion;
4    //Miembros del paquete Controlador para las etiquetas
5    ControlVelocidad control;
6    Monotorizacion monot;
7    Deposito deposito;
8    Notificaciones notif; //etc. etc.
9    //Constructor
10   public PanelPantalla(Interfaz interfaz) {
11      this.interfaz=interfaz;
12      initComponents();
13    }
```

Panel de Etiquetas II

```
1  private void initComponents() {
2   EtiquetaVelocidad = new javax.swing.JLabel();
3   EtiquetaRevolucion = new javax.swing.JLabel();
4           //etc.etc.
5  setLayout(null);
6  setBackground(new java.awt.Color(0, 255, 255));
7  EtiquetaVelocidad.setFont(new java.awt.Font("MS Sans Serif", 1,
       14));
8  EtiquetaVelocidad.setText("Velocidad");
9  add(EtiquetaVelocidad);
10 EtiquetaVelocidad.setBounds(40, 10, 110, 19);
11  //etc.etc.
12 }
```

Panel Etiquetas III

```
1  synchronized public void manejadorEvento(){
2  // Actualizar barras de aceleracion y frenado
3   BarraAceleracion.setValue(10*((int)acelerador.leerEstado()));
4   BarraFrenado.setValue(10*((int)freno.leerEstado()));
5   // Mostramos todos los datos en la pantallas
6   mostrarRevoluciones();
7   mostrarVelocidad();
8   mostrarMedias();
9   mostrarNotificaciones();
10 // Refrescamos la pantalla
11         interfaz.repaint();
12     }
```

Bibliografía Fundamental


Arnold, K. (1998).
The Java Programming Language, volume 2nd edition.
Addison-Wesley, Reading, Mass.


Booch, G. (2008).
Handbook of Software Architecture.
http://www.booch.com/systems.jsp.


Bruegge, B. and Dutoit, A. (2013).
Object–Oriented Software Engineering using UML patterns and Java.
Prentice-Hall, Upper Saddle River, NJ.


Flanagan, D. (2005).
Java in a nutshell, volume 5th edition.
O'Reilly, Sebastopol, CA.


Fowler, M. (2003).
Patterns of enterprise application architecture.
Addison–Wesley, Boston, Mass.


Gutz, S. (2000).
Up to speed with Swing, volume 2nd edition.
Manning, Greenwich, CT.


Larman, C. (2010).
UML y patrones : una introducción al análisis y diseño orientado a objetos y al proceso unificado.
Pearson Education.