



Componentes Swing

Componentes Swing

Hay tres formas de construir interfaces gráficas en Java:

- Con las clases pertenecientes al paquete **java.AWT**
- Con las clases pertenecientes al paquete **javax.Swing**
- Con **JavaFx**

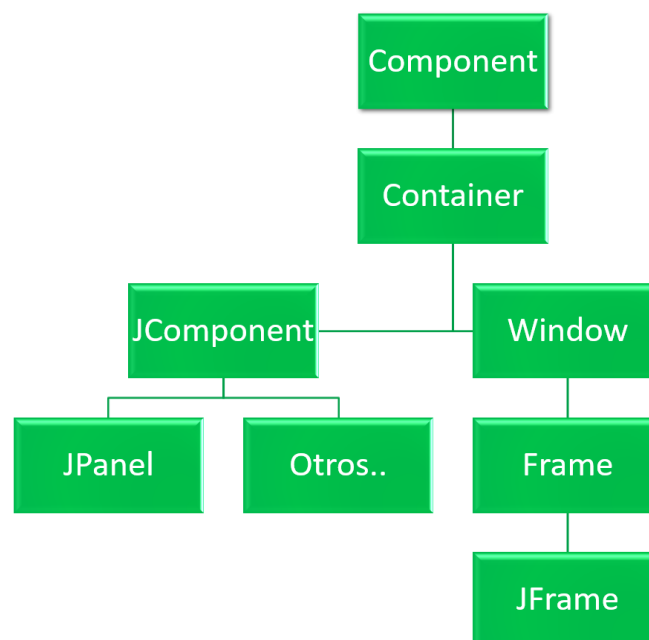
El paquete **javax.swing** es la evolución del paquete `java.awt` considerándose este obsoleto a día de hoy. JavaFx es la nueva tecnología que viene a sustituir a `javax.swing` aunque en el momento de escribir estas líneas aún no ha culminado ese “relevo generacional” y ambas bibliotecas conviven en armonía para construir interfaces gráficas en Java.

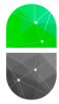
En este capítulo veremos los **componentes swing básicos** que podemos utilizar para construir objetos gráficos que permitan interactuar con el usuario.

Entendemos por componentes Swing todos aquellos elementos que forman parte de el paquete `javax.swing` con los que podemos construir una interfaz gráfica en Java.

Todas las clases del paquete `javax.swing` que permiten construir componentes comienzan por J, así que fijarnos en el nombre de la clase es una buena manera de saber si estamos construyendo con AWT o lo estamos haciendo con Swing.

Es muy importante tener siempre en mente la **jerarquía de la herencia** de los componentes Swing. Es una cadena de herencia muy larga y esto hace que las clases con las que construimos componentes tengan muchos métodos disponibles con las que poder configurar cada componente. En la imagen que aparece a continuación se observa parte de la jerarquía de herencia de algunos componentes Swing

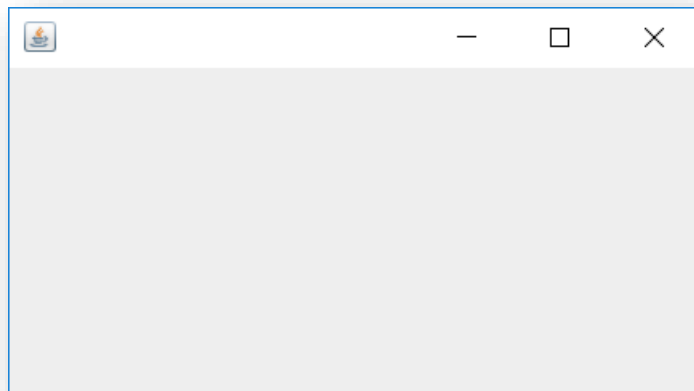




Componentes Swing básicos

- **JFrame**

Los JFrame son contenedores. Es lo que conocemos como “marcos”, “cuadros” o “frames”. En definitiva, los JFrame son los que construyen el contenedor principal de una interfaz gráfica (la ventana propiamente dicha). En la imagen siguiente se ve una ventana construida con la clase JFrame:



El código para construir el JFrame es el siguiente:

```
import javax.swing.*;

public class PrimerJFrame {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        MiJFrame miVentana=new MiJFrame();

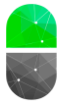
        miVentana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}

class MiJFrame extends JFrame{

    public MiJFrame(){

        setBounds(600,350,450,250); |

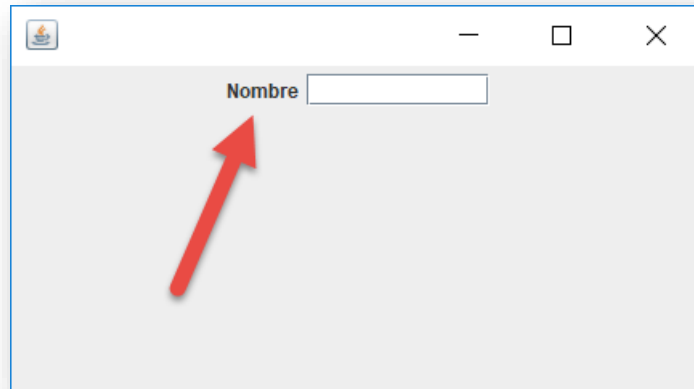
        setVisible(true);
    }
}
```



- **JLabel:**

Los JLabel son “**etiquetas**”, esto es, elementos de texto simple dentro de una interfaz gráfica. Normalmente acompañan a otros componentes como cajas de texto o áreas de texto para indicar a qué corresponden cada una, aunque se pueden utilizar para cualquier otro propósito.

En la imagen que aparece a continuación, se señala un JLabel dentro de una interfaz gráfica sencilla:



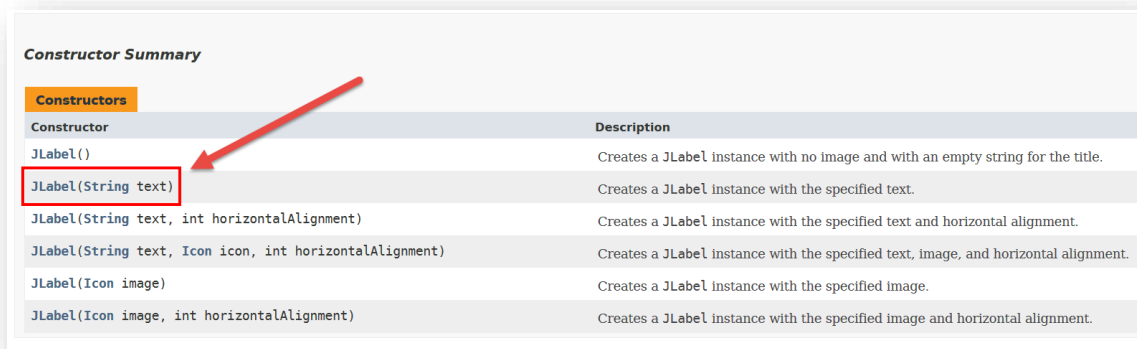
El código para construir una JLabel se señala en la siguiente imagen:

```
public class PrimerJFrame {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
  
        MiJFrame miVentana=new MiJFrame();  
  
        miVentana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    }  
}  
  
class MiJFrame extends JFrame{  
  
    public MiJFrame(){  
        setBounds(600,350,450,250);  
  
        add(new MiPanel());  
  
        setVisible(true);  
    }  
}  
  
class MiPanel extends JPanel{  
  
    public MiPanel() {  
  
        JLabel etiquetaNombre=new JLabel("Nombre");  
        JTextField cuadroNombre=new JTextField(10);  
  
        add(etiquetaNombre);  
  
        add(cuadroNombre);  
    }  
}
```



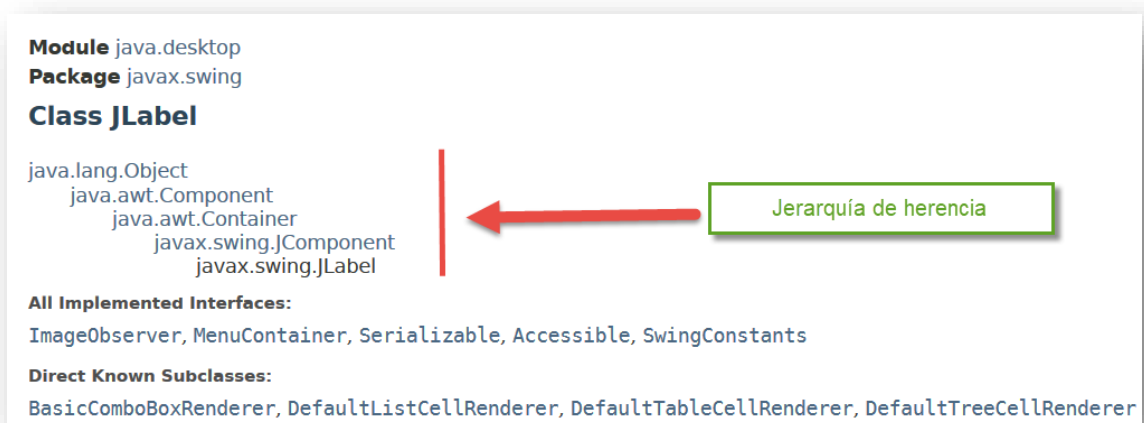
Vemos en la imagen anterior como creamos una instancia perteneciente a la clase JLabel con el nombre de “etiquetaNombre” y cómo le pasamos un parámetro al constructor de tipo String que hace referencia al texto que aparecerá finalmente en el JLabel. Es importante ir a la API de Java y ver detenidamente esta clase JLabel. El resto de clases para construir componentes Swing son muy similares y entendiendo una, se entienden todas.

En la imagen que aparece a continuación se observa la sobrecarga de constructores en clase JLabel. En el código anterior se utiliza el constructor señalado en la imagen:



Constructors	Description
JLabel()	Creates a JLabel instance with no image and with an empty string for the title.
JLabel(String text)	Creates a JLabel instance with the specified text.
JLabel(String text, int horizontalAlignment)	Creates a JLabel instance with the specified text and horizontal alignment.
JLabel(String text, Icon icon, int horizontalAlignment)	Creates a JLabel instance with the specified text, image, and horizontal alignment.
JLabel(Icon image)	Creates a JLabel instance with the specified image.
JLabel(Icon image, int horizontalAlignment)	Creates a JLabel instance with the specified image and horizontal alignment.

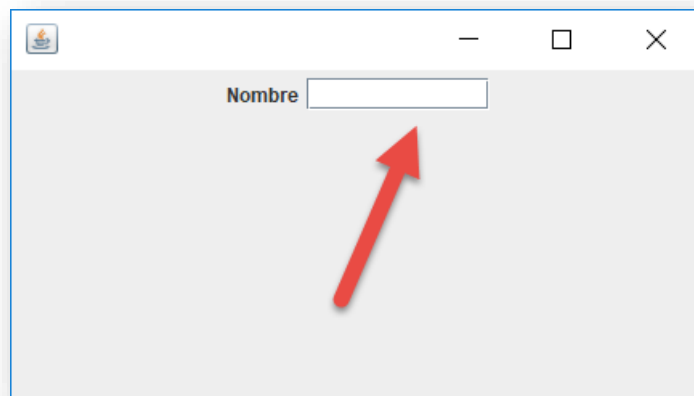
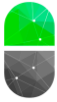
Jerarquía de herencia de la clase JLabel (muy similar en el resto de clases de componentes Swing):



- **JTextField:**

El componente JTextField construye un cuadro de texto típico donde el usuario podrá hacer clic con el ratón y escribir el texto que desee.

En la imagen que aparece a continuación, se señala un JTextField:



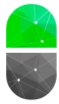
El código para construir este tipo de componente se señala en la imagen a continuación:

```
public class PrimerJFrame {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
  
        MiJFrame miVentana=new MiJFrame();  
  
        miVentana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    }  
}  
  
class MiJFrame extends JFrame{  
  
    public MiJFrame(){  
        setBounds(600,350,450,250);  
        add(new MiPanel());  
        setVisible(true);  
    }  
}  
  
class MiPanel extends JPanel{  
    public MiPanel() {  
        JLabel etiquetaNombre=new JLabel("Nombre");  
        TextField cuadroNombre=new TextField(10);  
        add(etiquetaNombre);  
        add(cuadroNombre);  
    }  
}
```

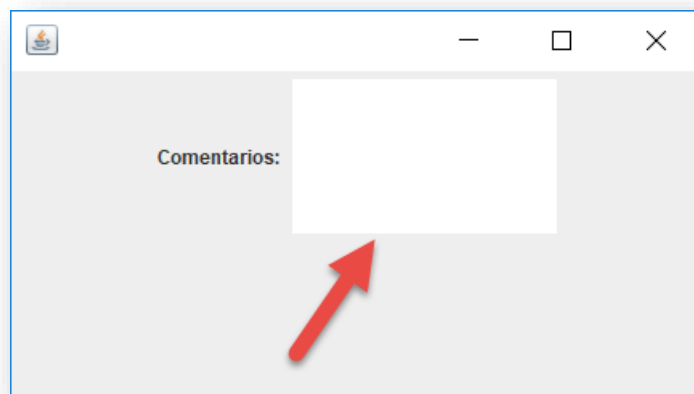
Como se observa en la imagen, el constructor recibe un parámetro de tipo entero (10) que hace referencia a la longitud del cuadro de texto. No confundir este parámetro con la cantidad de caracteres que se pueden introducir en un JTextField. A pesar de que esta clase es muy similar a la anterior, recomiendo ir a la API y echar un vistazo a la misma.

- **JTextArea:**

Este componente permite crear áreas de texto donde el usuario podrá escribir mayor cantidad de texto.



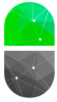
En la siguiente imagen se señala un JTextArea dentro de una interfaz gráfica:



El código que permite crear el componente se señala en la imagen a continuación:

```
public class PrimerJFrame {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
  
        MiJFrame miVentana=new MiJFrame();  
  
        miVentana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    }  
}  
  
class MiJFrame extends JFrame{  
  
    public MiJFrame(){  
        setBounds(600,350,450,250);  
        add(new MiPanel());  
        setVisible(true);  
    }  
}  
  
class MiPanel extends JPanel{  
  
    public MiPanel() {  
        JLabel etiquetaNombre=new JLabel("Comentarios: ");  
  
        JTextArea miArea=new JTextArea(6,15);  
        add(etiquetaNombre);  
        add(miArea);  
    }  
}
```

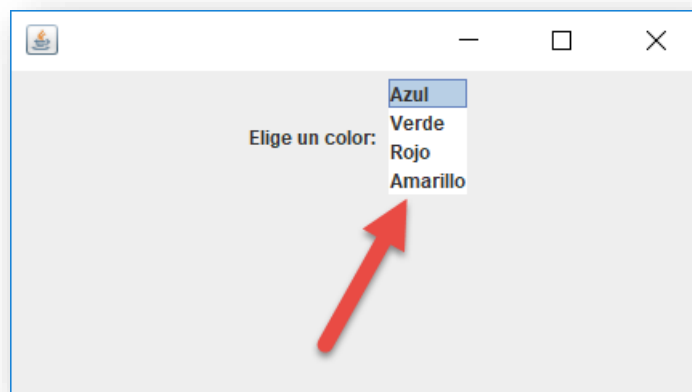
En este caso los parámetros del constructor de JTextArea hacen referencia a cuántos caracteres se visualizan en fila y cuántos en columna, marcando las dimensiones del JTextArea. En la siguiente imagen se señala en la API el constructor utilizado:



Constructor Summary	
Constructors	
Constructor	Description
JTextArea()	Constructs a new TextArea.
JTextArea(int rows, int columns)	Constructs a new empty TextArea with the specified number of rows and columns.
JTextArea(String text)	Constructs a new TextArea with the specified text displayed.
JTextArea(String text, int rows, int columns)	Constructs a new TextArea with the specified text and number of rows and columns.
JTextArea(Document doc)	Constructs a new JTextArea with the given document model, and defaults for all of the other arguments (null, 0, 0).
JTextArea(Document doc, String text, int rows, int columns)	Constructs a new JTextArea with the specified number of rows and columns, and the given model.

- **JList:**

Este componente nos permite crear listas de elementos donde el usuario podrá seleccionar un ítem haciendo clic sobre él. Se ve un JList en la siguiente imagen:

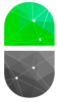


El código para crear el JList es el siguiente:

```
class MiPanel extends JPanel{  
    public MiPanel() {  
        JLabel etiquetaNombre=new JLabel("Elige un color: ");  
        String colores[]= {"Azul", "Verde", "Rojo", "Amarillo"};  
        JList misOpciones=new JList(colores);  
        add(etiquetaNombre);  
        add(misOpciones);  
    }  
}
```

Se observa en la imagen anterior cómo en este caso el constructor de JList recibe por parámetro un array de tipo String que contiene los elementos del JList.

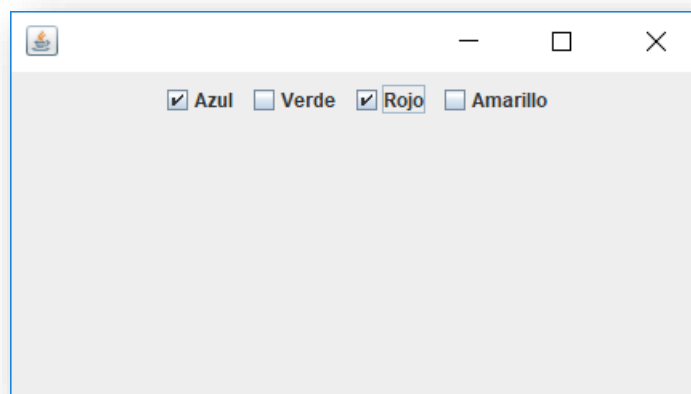
El constructor utilizado en este caso se señala en la imagen de la API a continuación:



Constructor Summary	
Constructors	
Constructor	Description
<code>JList()</code>	Constructs a <code>JList</code> with an empty, read-only, model.
<code>JList(E[] listData)</code>	Constructs a <code>JList</code> that displays the elements in the specified array.
<code>JList(Vector<? extends E> listData)</code>	Constructs a <code>JList</code> that displays the elements in the specified <code>Vector</code> .
<code>JList(ListModel<E> dataModel)</code>	Constructs a <code>JList</code> that displays elements from the specified, non-null, model.

- **JCheckBox:**

Este componente permite seleccionar varios ítems a la vez. Son las clásicas “casillas de verificación”. En la siguiente imagen se ven varios `JCheckBox` dentro de un `JFrame`:



El código para crear los `JCheckBox` de la imagen anterior es el siguiente:

```
class MiPanel extends JPanel{  
    public MiPanel() {  
        JCheckBox azul=new JCheckBox("Azul");  
        JCheckBox verde=new JCheckBox("Verde");  
        JCheckBox rojo=new JCheckBox("Rojo");  
        JCheckBox amarillo=new JCheckBox("Amarillo");  
        add(azul);  
        add(verde);  
        add(rojo);  
        add(amarillo);  
    }  
}
```

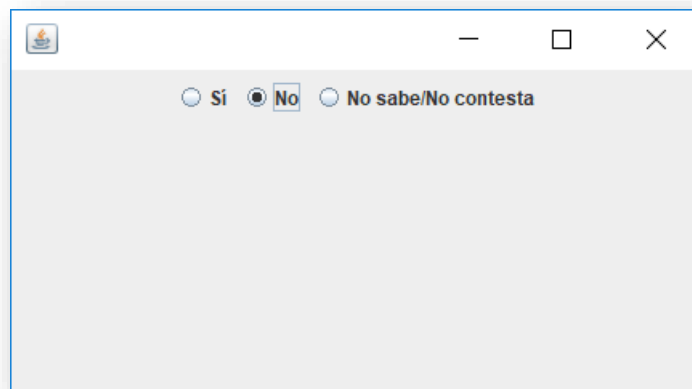



En este caso la sobrecarga de constructores de la clase `JCheckBox` es notable. Se señala en la imagen de la API el constructor escogido en el código del ejemplo anterior. El constructor señalado pide un parámetro de tipo `String` el cual hace referencia al texto que acompaña al `JCheckBox`.

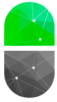
Constructor Summary	
Constructors	
Constructor	Description
<code>JCheckBox()</code>	Creates an initially unselected check box button with no text, no icon.
<code>JCheckBox(String text)</code>	Creates an initially unselected check box with text.
<code>JCheckBox(String text, boolean selected)</code>	Creates a check box with text and specifies whether or not it is initially selected.
<code>JCheckBox(String text, Icon icon)</code>	Creates an initially unselected check box with the specified text and icon.
<code>JCheckBox(String text, Icon icon, boolean selected)</code>	Creates a check box with text and icon, and specifies whether or not it is initially selected.
<code>JCheckBox(Action a)</code>	Creates a check box where properties are taken from the Action supplied.
<code>JCheckBox(Icon icon)</code>	Creates an initially unselected check box with an icon.
<code>JCheckBox(Icon icon, boolean selected)</code>	Creates a check box with an icon and specifies whether or not it is initially selected.

- **JRadioButton:**

Este componente es muy similar al `JTextBox` con la diferencia de que funcionan “en grupo”. Esto hace que solo sea posible la selección de un componente a la vez. Se utilizan cuando la elección escogida es excluyente, es decir, cuando no tiene sentido elegir más de un elemento a la vez. En la imagen a continuación se ve un ejemplo:



El código para construir los `JRadioButton` del ejemplo anterior se muestra a continuación:

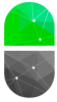


```
class MiPanel extends JPanel{  
  
    public MiPanel() {  
  
        JRadioButton si=new JRadioButton("Sí");  
  
        JRadioButton no=new JRadioButton("No");  
  
        JRadioButton noSabe=new JRadioButton("No sabe/No contesta");  
  
        ButtonGroup miGrupo=new ButtonGroup();  
  
        miGrupo.add(si);  
  
        miGrupo.add(no);  
  
        miGrupo.add(noSabe);  
  
        add(si);  
  
        add(no);  
  
        add(noSabe);  
  
    }  
}
```

En la imagen anterior vemos cómo la construcción de un componente `JRadioButton` es muy similar a la de un `JCheckBox`, sin embargo, vemos también cómo se agrega un elemento adicional: el `ButtonGroup`. El `ButtonGroup` permite agrupar los `JRadioButton` y que estos funcionen como una unidad, es decir, que la selección sea excluyente (es imposible seleccionar más de un elemento a la vez) lo cual es el sentido de este tipo de componentes.

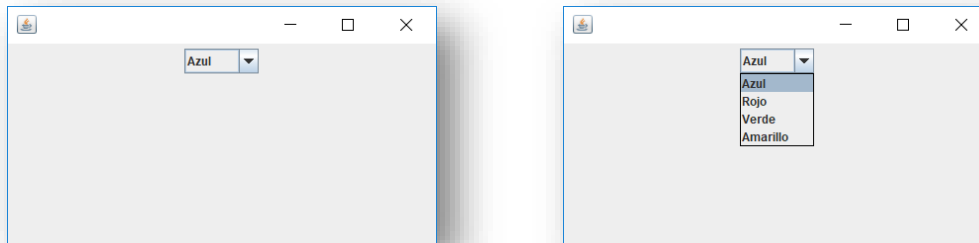
En cuanto a los constructores, son prácticamente idénticos a los vistos con el componente `JCheckBox` tal y como se aprecia en la siguiente imagen:

Constructors	
Constructor	Description
<code>JRadioButton()</code>	Creates an initially unselected radio button with no set text.
<code>JRadioButton(String text)</code>	Creates an unselected radio button with the specified text.
<code>JRadioButton(String text, boolean selected)</code>	Creates a radio button with the specified text and selection state.
<code>JRadioButton(String text, Icon icon)</code>	Creates a radio button that has the specified text and image, and that is initially unselected.
<code>JRadioButton(String text, Icon icon, boolean selected)</code>	Creates a radio button that has the specified text, image, and selection state.
<code>JRadioButton(Action a)</code>	Creates a radiobutton where properties are taken from the Action supplied.
<code>JRadioButton(Icon icon)</code>	Creates an initially unselected radio button with the specified image but no text.
<code>JRadioButton(Icon icon, boolean selected)</code>	Creates a radio button with the specified image and selection state, but no text.



- **JComboBox:**

Este componente es similar al componente JList visto anteriormente con la diferencia de que en este componente los elementos seleccionables están disponibles en un menú desplegable. En las imágenes a continuación se aprecia este componente antes de desplegar las opciones y una vez desplegadas:



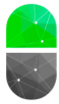
El código para construir este componente tal y como aparece en la imagen anterior, se muestra a continuación:

```
class MiPanel extends JPanel{  
    public MiPanel() {  
        String colores[]= {"Azul", "Rojo", "Verde", "Amarillo"};  
        JComboBox comboColores=new JComboBox(colores);  
        add(comboColores);  
    }  
}
```

Al igual que ocurría con el componente JList, el constructor de este componente permite por parámetro el paso de un array. Dicho array contendrá los elementos a mostrar. En la imagen de la API a continuación, se ve la sobrecarga de constructores de este componente:

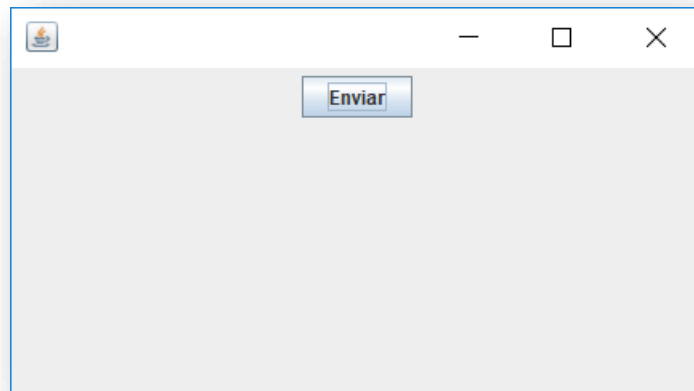
Constructor Summary	
Constructors	
Constructor	Description
JComboBox()	Creates a JComboBox with a default data model.
JComboBox(E[] items)	Creates a JComboBox that contains the elements in the specified array.
JComboBox(Vector<E> items)	Creates a JComboBox that contains the elements in the specified Vector.
JComboBox(ComboBoxModel<E> aModel)	Creates a JComboBox that takes its items from an existing ComboBoxModel.

Aparece señalado el constructor elegido en el ejemplo de código anterior.



- **JButton:**

Este componente permite construir botones clásicos en los que el usuario podrá hacer clic. En la imagen a continuación se muestra el aspecto de este componente:



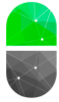
El código java del ejemplo anterior es sencillo y se muestra a continuación:

```
class MiPanel extends JPanel{  
  
    public MiPanel() {  
  
        JButton miBoton=new JButton("Enviar");  
  
        add(miBoton);  
  
    }  
}
```

En la clase JButton también hay sobrecarga de constructores tal y como se observa en la API:

Constructor Summary	
Constructors	
Constructor	Description
JButton()	Creates a button with no set text or icon.
JButton(String text)	Creates a button with text.
JButton(String text, Icon icon)	Creates a button with initial text and an icon.
JButton(Action a)	Creates a button where properties are taken from the Action supplied.
JButton(Icon icon)	Creates a button with an icon.

En la imagen anterior se señala el constructor utilizado en el ejemplo, pero observa cómo también es posible pasar al constructor un parámetro de tipo Action (esto también es posible



Swing

en componentes vistos anteriormente) lo que permite construir componentes que respondan a eventos (clic, selecciones etc.).

Hay muchísimos más componentes que nos permitirán elaborar interfaces gráficas más o menos completas como por ejemplo **JScrollBar**, **JTree**, **JTable** etc. La forma de proceder para utilizar estos elementos es la misma que hemos descrito con los ejemplos anteriores: ir a la API y ver sus constructores y métodos para ver qué posibilidades de uso nos ofrece cada uno.