JTextField y JLabel

JLabel

- Para establecer un texto por código a un **JLabel** se utiliza el siguiente método:

  setText("Correcto");

  Ejemplo:

JLabel texto= new JLabel("Texto");

...

texto.setText("Esto es una prueba de texto");

Esto es útil para sacar mensajes por pantalla en una acción concreta. Si lo que quisiéramos es hacerlo visible o ocultarlo,
utilizaríamos texto.setVisible() como en el Frame.

JTextField

Para meter texto en el *JTextField* desde código, usamos el método *setText()*. Si queremos vaciar el contenido, usaremos también *setText()*, pero pasando una cadena vacía ""

```
textField.setText("Hola");
// vaciar el JTextField
textField.setText("");
```

Para recuperar el texto, usaremos el método *getText()*. Si el *JTextField* está vacío, este método nos devolverá la cadena vacía ""

```
String texto = textField.getText();
if ("".equals(texto))
   System.out.println("El JTextField está vacío");
```

Poner y leer números en el *JTextField*

El *JTextField* sólo admite y devuelve *String*, por lo que si queremos meter u obtener números, debemos hacer fuera la conversión. Para meter un número se puede hacer así

```
int valor = 33;
textField.setText(Integer.toString(valor));
```

Para obtener el número, puesto que el usuario puede escribir lo que le de la gana, incluidas letras, debemos además hacer una comprobación

```
int valor;
String texto = textField.getText();
try {
   valor = Integer.parseString(texto);
}
catch (NumberFormatException e) {
   System.err.println("No se puede convertir a numero");
   e.printStackTrace();
}
```

De la misma forma, si quisieramos obtener *float*, *double*, etc, usaríamos las clase *Float*, *Double*, etc en vez de *Integer*.

De todas formas, para leer y escribir números en un *JTextField* es mejor usar un JFormattedTextField. El siguiente ejemplo muestra como usarlo para enteros

```
// En el constructor pasamos el tipo de valor que
queremos usar, en este
// caso un Integer
JFormattedTextField jftf = new JFormattedTextField(new
Integer(3));
...
jftf.setValue(new Integer(44)); // Le pasamos un 44
...
Integer valorRecogido = (Integer)jftf.getValue();  //
recogemos el valor escrito.
```

De esta forma nos ahorramos hacer nosotros las conversiones. Podemos usar igualmente las clases *Float*, *Double*, *Long*, *Date*, etc y en general cualquier clase que tenga un constructor que admita un *String* como parámetro (necesario para *getValue()*) y que tenga método *toString()* (necesario para *setValue()*). ¿Por qué son estos métodos necesarios?

- Cuando llamamos a *getValue()*, se recogerá el *String* del *JFormattedTextField* y ese *String* se tratará de usar en el constructor de la clase a devolver. Por ejemplo, si estamos usando *Integer* y el usuario escribe un *44*, el método *getValue()* intentará hacer algo parecido a esto

```
return new Integer("44");
```

- Cuando llamamos a *setValue(unDato)*, se llamará al método *toString()* de *unDato* y eso se meterá en el *JFormattedTextField* para que sea visible.

RadioButton, Checkbox

- Para saber si un **checkBox o un radioButton** está seleccionado o no, se utiliza el siguiente método:

  isSelected();

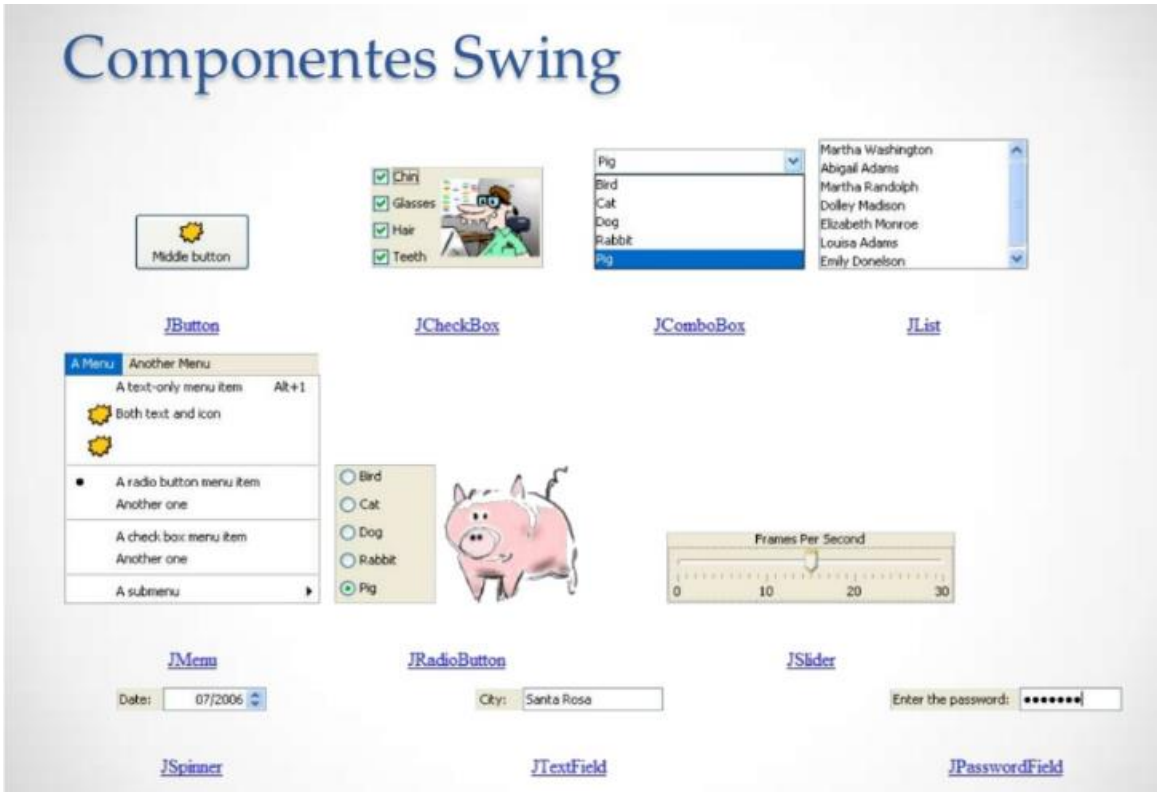
  Ejemplo:

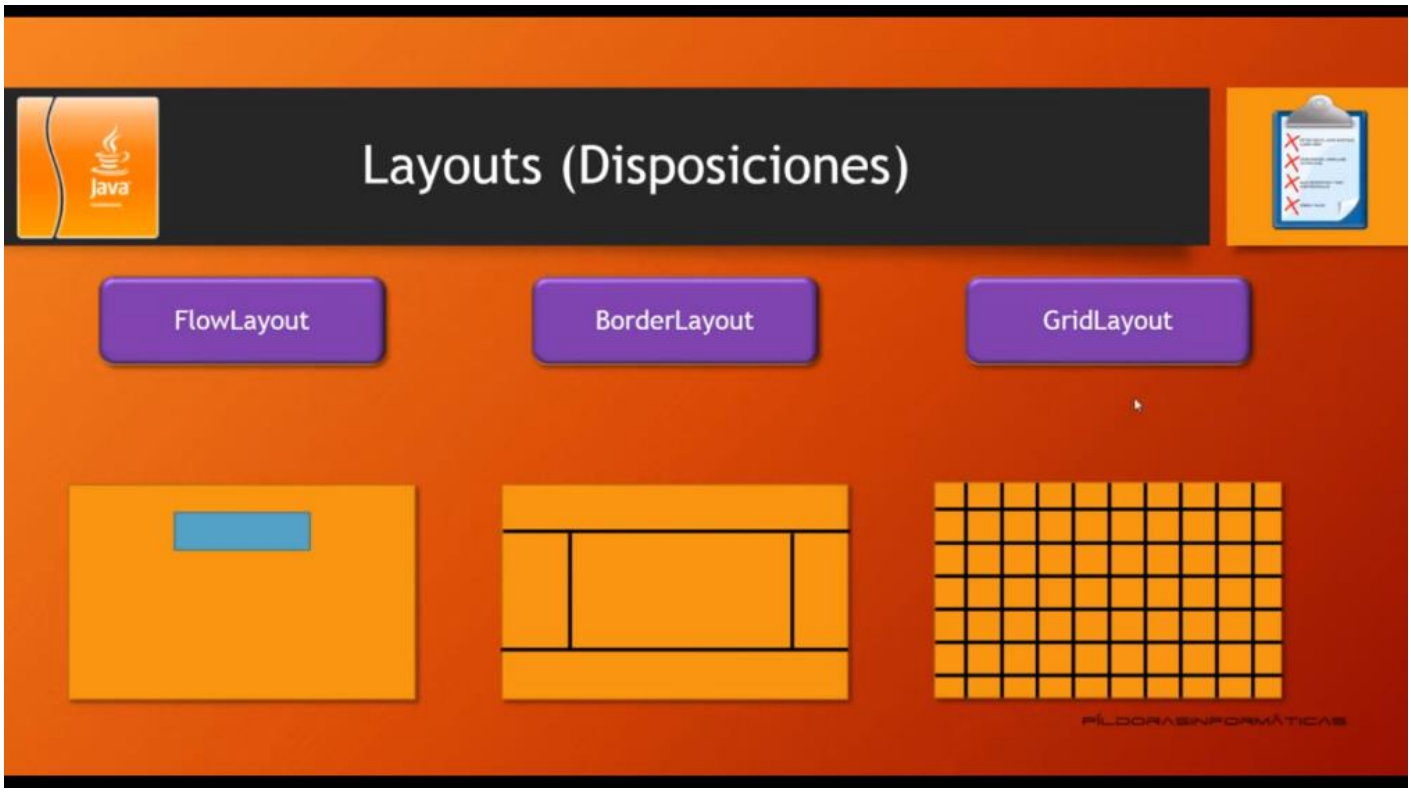
```
JCheckBox chkcheck1 = new JCheckBox("check1");

....

if (chkcheck1.isSelected()){

texto.setText("Está seleccionado");

}
```

- Para crear un grupo de radioButtons:

```
ButtonGroup buttonGroup = new BottonGroup();
JRadioButton btn01 = new JRadioButton("btn 1");
buttonGroup.add(btn01);
JRadioButton btn02 = new JRadioButton("btn 2");
buttonGroup.add(btn02);
JRadioButton btn03 = new JRadioButton("btn 3");
buttonGroup.add(btn03);
// gets the selected radio button
if(buttonGroup.getSelection().equals(btn01.getModel())) {
 // código que se tenga que ejecutar si el seleccionado es ese radioButton
}
```

En el ejemplo vemos como definimos tres **JRadioButtons**, luego creamos un objeto del tipo **ButtonGroup** con el cual vamos a proceder a añadir los botones que habíamos creado, esto hace que al estar agrupados cuando seleccionemos una opción inmediatamente la otra sea deseleccionada.


Componentes Swing
Middle button
Chin
Glasses
Hair
Teeth
Pig
Bird
Cat
Dog
Rabbit
Pig
Martha Washington
Abigail Adams
Martha Randolph
Dolley Madison
Elizabeth Monroe
Louisa Adams
Emily Donelson
JButton
JCheckBox
JComboBox
JList
A Menu
Another Menu
A text-only menu item
Alt+1
Both text and icon
A radio button menu item
Another one
A check box menu item
Another one
A submenu
Bird
Cat
Dog
Rabbit
Pig
Frames Per Second
0
10
20
30
JMenu
JRadioButton
JSlider
Date:
07/2006
City:
Santa Rosa
Enter the password:
JSpinner
JTextField
JPasswordField


Java
Layouts (Disposiciones)
FlowLayout
BorderLayout
GridLayout

Acciones al cerrar una ventana

Establecer la operación predeterminada de cierre

Para establecer la operación predeterminada de cierre tienes que usar el método *setter* que se encuentra dentro de la clase `JFrame`, `setDefaultCloseOperation`, que determina qué es lo que ocurre al hacer clic en el botón de cierre. Este método toma los siguientes parámetros:

- `JFrame.EXIT_ON_CLOSE`: cierra el marco y finaliza la ejecución del programa.
- `JFrame.DISPOSE_ON_CLOSE`: cierra el marco, pero no necesariamente finaliza la ejecución del programa.
- `JFrame.HIDE_ON_CLOSE`: hace que el marco parezca cerrarse cambiando la propiedad de visibilidad a "falso". La diferencia entre `HIDE_ON_CLOSE` y `DISPOSE_ON_CLOSE` es que la última libera todos los recursos utilizados por el marco y sus componentes.
- `JFrame.DO_NOTHING_ON_CLOSE`: no hace nada al presionar el botón de cierre. Esto podría ser útil, por ejemplo, si quisieras mostrar un mensaje de confirmación antes de cerrar la ventana. En ese caso, puedes agregarle un `WindowListener` al marco y sobrescribir el método `windowClosing`.

En la siguiente tabla se muestra un resumen de los *listener*, los componentes y las acciones a las que responden:

Tabla 6.12. Listeners asociados a componentes

Listener	Componentes	Acción a la que responden
ActionListener	1. JButton 2. JTextField 3. JComboBox 4. ...	1. Presionar el botón. 2. Pulsar intro. 3. Elegir una opción. 4. ...
AdjustmentListener	1. JScrollBar 2. ...	Mover la barra de desplazamiento. ...
FocusListener	1. JButton 2. JTextField 3. JComboBox 4. ...	Las acciones de este listener son obtener y perder el foco (colocarnos en el componente e irnos del mismo cuando estaba activo).
ItemListener	1. JCheckbox 2. ...	1. Seleccionar y deseleccionar la opción.
KeyListener	1. JTextField 2. JTextArea 3. ...	Pulsar una tecla cuando el componente tiene el foco.
MouseListener	Múltiples componentes	Acciones como presionar el botón del ratón.
MouseMotionListener	Múltiples componnentes	Acciones como arrastrar (*drag*) o pasar por encima del objeto.
WindowListener	1. JFrame	Acciones relativas a la ventana como por ejemplo cerrarla.