

# **Programación**

## **JAVA INTERFACES VISUALES**

**1º DESARROLLO DE  
APLICACIONES  
MULTIPLATAFORMA (D.A.M.)  
2021-2022**

## 30 Interfaces visuales (componentes Swing)

Hasta ahora hemos resuelto todos los algoritmos haciendo las salidas a través de una consola en modo texto. Hoy en día es muy común la necesidad de hacer la entrada y salida de datos mediante interfaces más amigables con el usuario.

En Java existen varias librerías de clase para implementar interfaces visuales. Utilizaremos los componentes Swing.

### **Problema 1:**

Confeccionar el programa "Hola Mundo" utilizando una interfaz gráfica de usuario.

## Programa:

```
import javax.swing.*;

public class Formulario extends JFrame{
    private JLabel label1;
    public Formulario() {
        setLayout(null);
        label1=new JLabel("Hola Mundo.");
        label1.setBounds(10,20,200,30);
        add(label1);
    }

    public static void main(String[] ar) {
        Formulario formulario1=new Formulario();
        formulario1.setBounds(10,10,400,300);
        formulario1.setVisible(true);
        formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Formulario.java X

```
1  import javax.swing.*;
2  public class Formulario extends JFrame{
3      private JLabel label1;
4      public Formulario() {
5          setLayout(null);
6          label1=new JLabel("Hola Mundo.");
7          label1.setBounds(10,20,200,30);
8          add(label1);
9      }
10
11      public static void main(String[] ar) {
12          Formulario formulario1=new Formulario();
13          formulario1.setBounds(10,10,400,300);
14          formulario1.setVisible(true);
15          formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16      }
17  }
```



**Hola Mundo.**

Hasta ahora habíamos utilizado la clase Scanner para hacer la entrada de datos por teclado. Dicha clase debemos importarla en nuestro programa con la sintaxis:

```
import java.util.Scanner;
```

Otra sintaxis para importarla es:

```
import java.util.*;
```

Si disponemos un \* indicamos que importe todas las clases del paquete java.util.

Ahora bien las componentes Swing hay que importarlas del paquete javax.swing. Cuando debemos importar varias componentes de un paquete es más conveniente utilizar el asterisco que indicar cada clase a importar:

```
import javax.swing.JFrame;  
import javax.swing.JLabel;
```

En lugar de las dos líneas anteriores es mejor utilizar la sintaxis:

```
import javax.swing.*;
```

La clase JFrame encapsula el concepto de una ventana. Luego para implementar una aplicación que muestre una ventana debemos plantear una clase que herede de la clase JFrame:

```
public class Formulario extends JFrame{
```

Con la sintaxis anterior estamos indicando que la clase Formulario hereda todos los métodos y propiedades de la clase JFrame.

Para mostrar un texto dentro de una ventana necesitamos de la colaboración de la clase JLabel (que tiene por objetivo mostrar un texto dentro de un JFrame)

Definimos luego como atributo de la clase un objeto de la clase JLabel:

```
private JLabel label1;
```

En el constructor de la clase llamamos al método heredado de la clase JFrame llamado `setLayout` y **le pasamos como parámetro un valor null**, con esto **estamos informando** a la clase JFrame que **utilizaremos posicionamiento absoluto para los controles visuales dentro del JFrame**.

```
public Formulario() {  
    setLayout(null);
```

Luego tenemos que crear el objeto de la clase JLabel y pasarle como parámetro al constructor el texto a mostrar:

```
label1=new JLabel("Hola Mundo.");
```

Ubicamos al objeto de la clase JLabel llamando al método **setBounds**, este requiere como parámetros la **columna, fila, ancho y alto** del JLabel. Finalmente llamamos al método `add` (metodo heredado de la clase JFrame) que tiene como objetivo añadir el control JLabel al control JFrame.

```
label1=new JLabel("Hola Mundo.");  
label1.setBounds(10,20,200,30);  
add(label1);  
}
```



Finalmente debemos codificar el main donde creamos un objeto de la clase Formulario, llamamos al método setBounds para ubicar y dar tamaño al control y mediante el método setVisible hacemos visible el JFrame:

```
public static void main(String[] ar) {  
    Formulario formulario1=new Formulario();  
    formulario1.setBounds(10,10,400,300);  
    formulario1.setVisible(true);  
    formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}
```

Llamamos al método **setDefaultCloseOperation** y le pasamos la constante definida en la clase JFrame EXIT\_ON\_CLOSE con el objetivo que **cuando se presione el botón de cerrado de la ventana se finalice la aplicación en forma completa, si hubiese varias ventanas abiertas se cerrarían todas.**

Cuando ejecutamos nuestro proyecto tenemos como resultado una ventana similar a esta:



## 31 - Swing - JFrame

El componente básico que requerimos cada vez que implementamos una interfaz visual con la librería Swing es la clase **JFrame**. Esta **clase encapsula una Ventana clásica de cualquier sistema operativo con entorno gráfico** (Windows, OS X, Linux etc.)

Hemos dicho que esta clase **se encuentra en el paquete javax.swing** y como generalmente utilizamos varias clases de este paquete luego para importarla utilizamos la sintaxis:

```
import javax.swing.*;
```

Podemos hacer una aplicación mínima con la clase JFrame:

```
import javax.swing.JFrame;
public class Formulario {
    public static void main(String[] ar) {
        JFrame f=new JFrame();
        f.setBounds(10,10,300,200);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Como vemos importamos la clase JFrame del paquete javax.swing:

```
import javax.swing.JFrame;
```

y luego en el main definimos y creamos un objeto de la clase JFrame (llamando luego a los métodos setBounds, setVisible y setDefaultCloseOperation):

```
public static void main(String[] ar) {  
    JFrame f=new JFrame();  
    f.setBounds(10,10,300,200);  
    f.setVisible(true);  
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}
```

Pero esta forma de trabajar con la clase JFrame es de poca utilidad ya que rara vez necesitamos implementar una aplicación que muestre una ventana vacía.

Lo más correcto es plantear una clase que herede de la clase JFrame y extienda sus responsabilidades agregando botones, etiquetas, editores de línea etc.

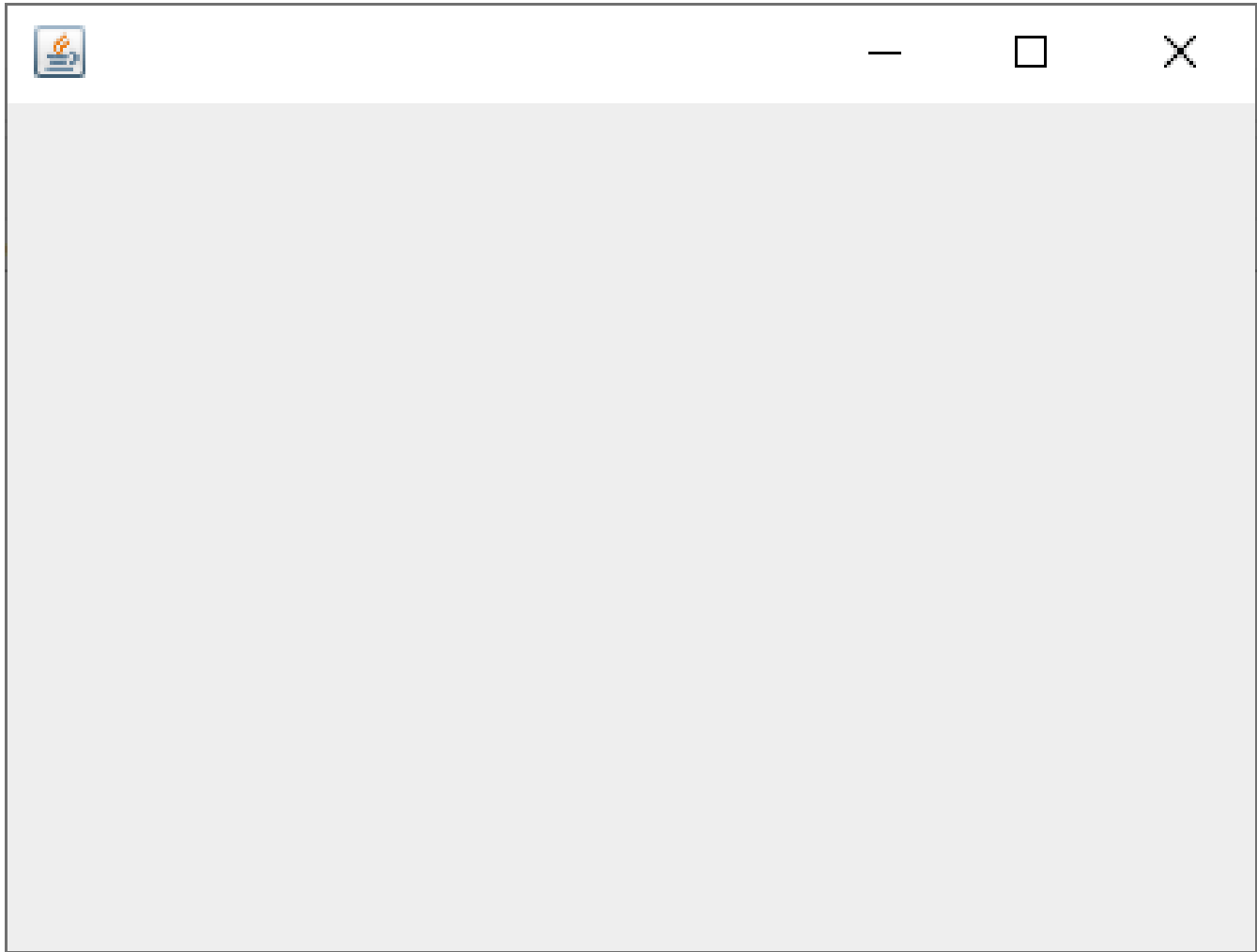
Entonces la **estructura básica** que emplearemos para crear una interfaz visual será:

## Programa:

```
import javax.swing.*;

public class Formulario extends JFrame{
    public Formulario() {
        setLayout(null);
    }

    public static void main(String[] ar) {
        Formulario formulario1=new Formulario();
        formulario1.setBounds(10,20,400,300);
        formulario1.setVisible(true);
        formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```



Importamos el paquete donde se encuentra la clase JFrame:

```
import javax.swing.*;
```

Planteamos una clase que herede de la clase JFrame:

```
public class Formulario extends JFrame{
```

En el constructor indicamos que **ubicaremos los controles visuales con coordenadas absolutas mediante la desactivación del Layout heredado** (más adelante veremos otras formas de ubicar los controles visuales dentro del JFrame):

```
    public Formulario() {  
        setLayout(null);  
    }
```

En el main creamos un objeto de la clase y llamamos a los métodos `setBounds`, `setVisible` y `setDefaultCloseOperation`:

```
public static void main(String[] ar) {  
    Formulario formulario1=new Formulario();  
    formulario1.setBounds(10,20,400,300);  
    formulario1.setVisible(true);  
    formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}
```



El método `setBounds` ubica el `JFrame` (la ventana) en la columna 10, fila 20 con un ancho de 400 píxeles y un alto de 300.

Debemos llamar al método `setVisible` y pasarle el valor `true` para que se haga visible la ventana.

Mediante el método `setDefaultCloseOperation` y pasando la constante `JFrame.EXIT_ON_CLOSE` se le indica al `JFrame` que cuando se presione el botón de cerrado de la ventana proceda a finalizar por completo la aplicación.

## **Problemas propuestos**

Crear una ventana de 1024 píxeles por 800 píxeles. Luego no permitir que el operador modifique el tamaño de la ventana. Sabiendo que hacemos visible al `JFrame` llamando el método `setVisible` pasando el valor `true`, existe otro método llamado `setResizable` que también requiere como parámetro un valor `true` o `false`.

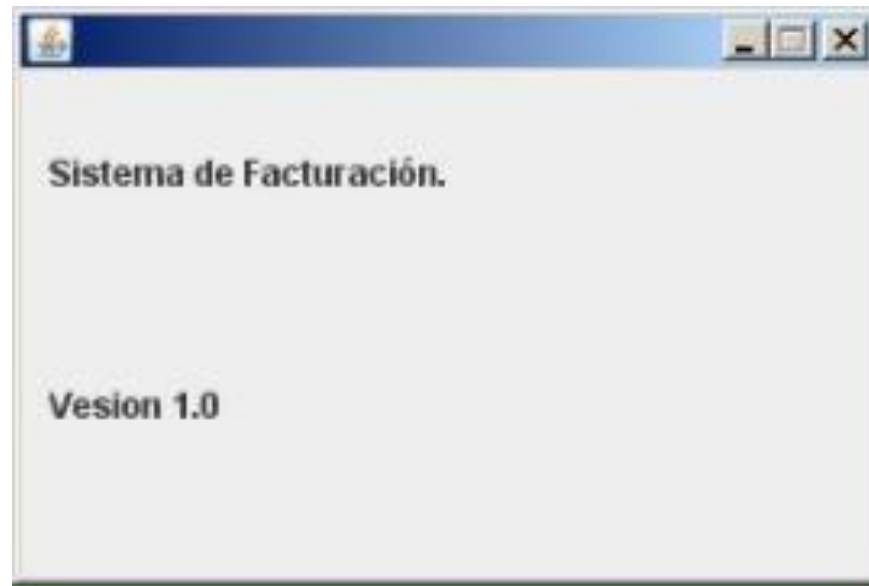
## 32 - Swing - JLabel

Veamos la segunda componente de la librería swing llamada JLabel.

La clase JLabel nos permite mostrar básicamente un texto.

### Problema 1:

Confeccionar una ventana que muestre el nombre de un programa en la parte superior y su número de versión en la parte inferior. No permitir modificar el tamaño de la ventana en tiempo de ejecución.



## Programa:

```
import javax.swing.*;

public class Formulario extends JFrame {
    private JLabel label1,label2;
    public Formulario() {
        setLayout(null);
        label1=new JLabel("Sistema de Facturación.");
        label1.setBounds(10,20,300,30);
        add(label1);
        label2=new JLabel("Vesion 1.0");
        label2.setBounds(10,100,100,30);
        add(label2);
    }

    public static void main(String[] ar) {
        Formulario formulario1=new Formulario();
        formulario1.setBounds(0,0,300,200);
        formulario1.setResizable(false);
        formulario1.setVisible(true);
        formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Formulario.java X

```
1 import javax.swing.*;
2 public class Formulario extends JFrame {
3     private JLabel label1,label2;
4     public Formulario() {
5         setLayout(null);
6         label1=new JLabel("Sistema de Facturación.");
7         label1.setBounds(10,20,300,30);
8         add(label1);
9         label2=new JLabel("Vesion 1.0");
10        label2.setBounds(10,100,100,30);
11        add(label2);
12    }
13
14    public static void main(String[] ar) {
15        Formulario formulario1=new Formulario();
16        formulario1.setBounds(0,0,300,200);
17        formulario1.setResizable(false);
18        formulario1.setVisible(true);
19        formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20    }
21 }
```



**Sistema de Facturación.**

**Vesion 1.0**

Importamos el paquete javax.swing donde se encuentran definidas las clase JFrame y JLabel:

```
import javax.swing.*;
```

Heredamos de la clase de JFrame:

```
public class Formulario extends JFrame {  
Definimos dos atributos de la clase JLabel:
```

```
    private JLabel label1,label2;
```

En el constructor creamos las dos JLabel y las ubicamos llamando al método setBounds, no hay que olvidar de llamar al método add que añade la JLabel al JFrame:

```
public Formulario() {  
    setLayout(null);  
    label1=new JLabel("Sistema de Facturación.");  
    label1.setBounds(10,20,300,30);  
    add(label1);  
    label2=new JLabel("Vesion 1.0");  
    label2.setBounds(10,100,100,30);  
    add(label2);  
}
```

Por último en el main creamos un objeto de la clase que acabamos de codificar, llamamos al `setBounds` para ubicar y dar tamaño al `JFrame`, llamamos al método `setResizable` pasando el valor `false` para no permitir modificar el tamaño del `JFrame` en tiempo de ejecución y finalmente llamamos al método `setVisible` para que se visualice el `JFrame`:

```
public static void main(String[] ar) {  
    Formulario formulario1=new Formulario();  
    formulario1.setBounds(0,0,300,200);  
    formulario1.setResizable(false);  
    formulario1.setVisible(true);  
    formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}
```



## **Problema propuesto**

Crear tres objetos de la clase JLabel, ubicarlos uno debajo de otro y mostrar nombres de colores.

## 33 - Swing - JButton

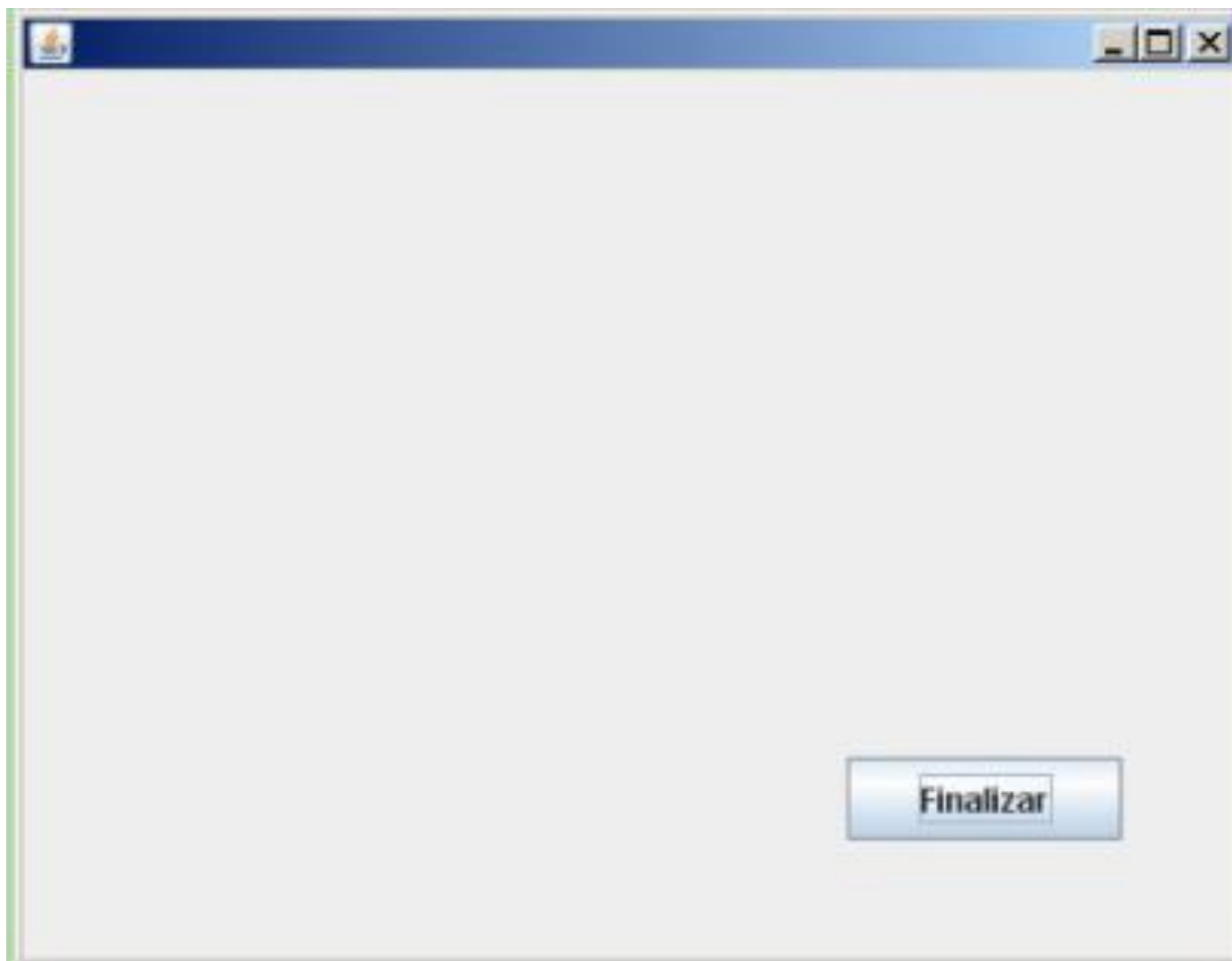
El tercer control visual de uso muy común es el que provee **la clase JButton**. Este control visual muestra un botón.

El proceso para añadir botones a un control JFrame es similar a añadir controles de tipo JLabel.

Ahora veremos la captura de eventos con los controles visuales. **Uno de los eventos más comunes es cuando hacemos clic sobre un botón.** Java implementa el concepto de interfaces para poder llamar a métodos de una clase existente a una clase desarrollada por nosotros.

### **Problema 1:**

Confeccionar una ventana que muestre un botón. Cuando se presione finalizar la ejecución del programa Java.



## Programa:

```
import javax.swing.*;
import java.awt.event.*;

public class Formulario extends JFrame implements ActionListener {
    JButton boton1;
    public Formulario() {
        setLayout(null);
        boton1=new JButton("Finalizar");
        boton1.setBounds(300,250,100,30);
        add(boton1);
        boton1.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource()==boton1) {
            System.exit(0);
        }
    }

    public static void main(String[] ar) {
        Formulario formulario1=new Formulario();
        formulario1.setBounds(0,0,450,350);
        formulario1.setVisible(true);
        formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

```
1 import javax.swing.*;
2 import java.awt.event.*;
3 public class Formulario extends JFrame implements ActionListener {
4     JButton boton1;
5     public Formulario() {
6         setLayout(null);
7         boton1=new JButton("Finalizar");
8         boton1.setBounds(300,250,100,30);
9         add(boton1);
10        boton1.addActionListener(this);
11    }
12
13    public void actionPerformed(ActionEvent e) {
14        if (e.getSource()==boton1) {
15            System.exit(0);
16        }
17    }
18
19    public static void main(String[] ar) {
20        Formulario formulario1=new Formulario();
21        formulario1.setBounds(0,0,450,350);
22        formulario1.setVisible(true);
23        formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24    }
25 }
```



**Finalizar**

La mecánica para atrapar el clic del objeto de la clase JButton se hace mediante la implementación de una interface. Una interface es un protocolo que permite la comunicación entre dos clases. Una interface contiene una o más cabeceras de métodos, pero no su implementación. Por ejemplo **la interface ActionListener tiene la siguiente estructura:**

```
interface ActionListener {  
    public void actionPerformed(ActionEvent e) {  
    }  
}
```

Luego las clases que implementen la interface ActionListener deberán especificar el algoritmo del método actionPerformed.

Mediante el concepto de interfaces podemos hacer que desde la clase JButton se puede llamar a un método que implementamos en nuestra clase.

Para indicar que una clase implementará una interface lo hacemos en la declaración de la clase con la sintaxis:

```
public class Formulario extends JFrame implements ActionListener {
```

Con esto estamos diciendo que nuestra clase implementa la interface ActionListener, luego estamos obligados a codificar el método actionPerformed.

Definimos un objeto de la clase JButton:

```
JButton boton1;
```

En el constructor creamos el objeto de la clase JButton y mediante la llamada del método addActionListener le pasamos la referencia del objeto de la clase JButton utilizando la palabra clave this (this almacena la dirección de memoria donde se almacena el objeto de la clase JFrame, luego mediante dicha dirección podemos llamar al método actionPerformed):

```
public Formulario() {  
    setLayout(null);  
    boton1=new JButton("Finalizar");  
    boton1.setBounds(300,250,100,30);  
    add(boton1);  
    boton1.addActionListener(this);  
}
```



**El método actionPerformed** (este método de la interface ActionListener **debe implementarse obligatoriamente**, en caso de no codificarlo o equivocarnos en su nombre aparecerá un mensaje de error en tiempo de compilación de nuestro programa.

**El método actionPerformed** se ejecutará cada vez que hagamos clic sobre el objeto de la clase JButton.

**La interface ActionListener** y el objeto de la clase **ActionEvent** que llega como parámetro **están definidos** en el paquete:

```
import java.awt.event.*;
```

Es decir que cada vez que se presiona el botón desde la clase JButton se llama al método actionPerformed y recibe como parámetro un objeto de la clase ActionEvent.

En el método actionPerformed mediante el acceso al método getSource() del objeto que llega como parámetro podemos analizar que botón se presionó:

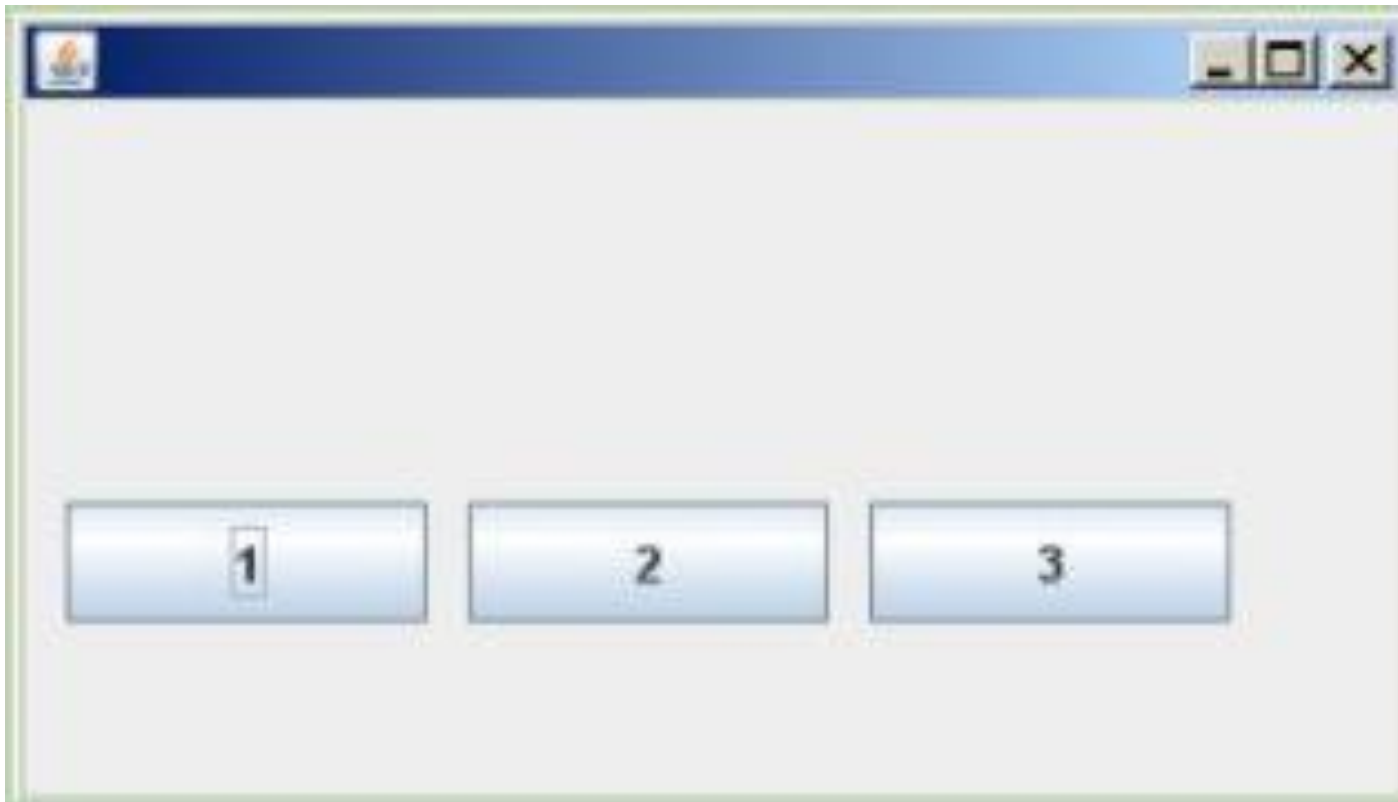
```
public void actionPerformed(ActionEvent e) {  
    if (e.getSource()==boton1) {  
        System.exit(0);  
    }  
}
```

Si se presiona el boton1, el if se verifica verdadero y por lo tanto llamando al método exit de la clase System se finaliza la ejecución del programa.

El main no varía en nada con respecto a problemas anteriores.

## **Problema 2:**

Confeccionar una ventana que contenga tres objetos de la clase JButton con las etiquetas "1", "2" y "3". Al presionarse cambiar el título del JFrame indicando cuál botón se presionó.



## Programa:

```
import javax.swing.*;
import java.awt.event.*;
public class Formulario extends JFrame implements ActionListener{
    private JButton boton1, boton2, boton3;
    public Formulario() {
        setLayout(null);
        boton1=new JButton("1");
        boton1.setBounds(10,100,90,30);
        add(boton1);
        boton1.addActionListener(this);
        boton2=new JButton("2");
        boton2.setBounds(110,100,90,30);
        add(boton2);
        boton2.addActionListener(this);
        boton3=new JButton("3");
        boton3.setBounds(210,100,90,30);
        add(boton3);
        boton3.addActionListener(this);
    }
}
```

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

```
public static void main(String[] ar){  
    Formulario formulario1=new Formulario();  
    formulario1.setBounds(0,0,350,200);  
    formulario1.setVisible(true);  
    formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}  
}
```

Debemos declarar 3 objetos de la clase JButton:

```
private JButton boton1, boton2, boton3;
```

En el constructor creamos los tres objetos de la clase JButton y los ubicamos dentro del control JFrame (también llamamos al método addActionListener para enviarle la dirección del objeto de la clase Formulario):

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

Cuando se presiona alguno de los tres botones se ejecuta el método `actionPerformed` y mediante tres `if` verificamos cual de los botones se presionó:

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

Según el botón presionado llamamos al método `setTitle` que se trata de un método heredado de la clase `JFrame` y que tiene por objetivo mostrar un `String` en el título de la ventana.

## **Problema propuesto**

Disponer dos objetos de la clase JButton con las etiquetas: "varón" y "mujer", al presionarse mostrar en la barra de títulos del JFrame la etiqueta del botón presionado.



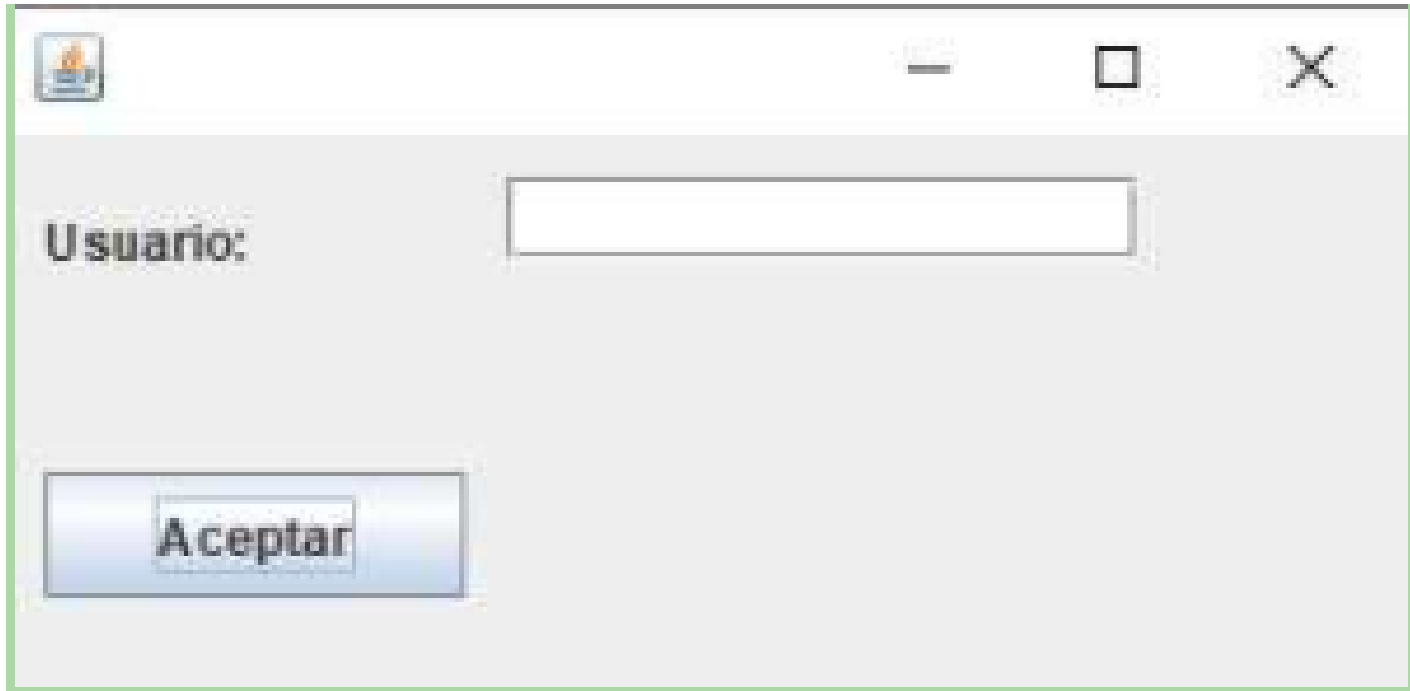
## 34 - Swing - JTextField

Así como podríamos decir que el control JLabel remplace a la salida estándar System.out.print, el control JTextField cumple la función de la clase Scanner para la entrada de datos.

El control JTextField permite al operador del programa introducir una cadena de caracteres por teclado.

### **Problema 1:**

Confeccionar un programa que permita introducir el nombre de usuario y cuando se presione un botón mostrar el valor introducido en la barra de títulos del JFrame.



A Java Swing window titled "Usuario" with a light gray background. The window has a standard title bar with a small icon on the left and three control buttons (minimize, maximize, and close) on the right. The main content area contains a label "Usuario:" followed by a rectangular text input field. Below the input field, there is a blue button with a gradient and the text "Aceptar".

Usuario:

Aceptar

## Programa:

```
import javax.swing.*;
import java.awt.event.*;
public class Formulario extends JFrame implements ActionListener{
    private JTextField textfield1;
    private JLabel label1;
    private JButton boton1;
    public Formulario() {
        setLayout(null);
        label1=new JLabel("Usuario:");
        label1.setBounds(10,10,100,30);
        add(label1);
        textfield1=new JTextField();
        textfield1.setBounds(120,10,150,20);
        add(textfield1);
        boton1=new JButton("Aceptar");
        boton1.setBounds(10,80,100,30);
        add(boton1);
        boton1.addActionListener(this);
    }
}
```

```
public void actionPerformed(ActionEvent e) {  
    if (e.getSource()==boton1) {  
        String cad=textfield1.getText();  
        setTitle(cad);  
    }  
}
```

```
public static void main(String[] ar) {  
    Formulario formulario1=new Formulario();  
    formulario1.setBounds(0,0,350,170);  
    formulario1.setVisible(true);  
    formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}  
}
```

```
1 import javax.swing.*;
2 import java.awt.event.*;
3 public class Formulario extends JFrame implements ActionListener{
4     private JTextField textfield1;
5     private JLabel label1;
6     private JButton boton1;
7     public Formulario() {
8         setLayout(null);
9         label1=new JLabel("Usuario:");
10        label1.setBounds(10,10,100,30);
11        add(label1);
12        textfield1=new JTextField();
13        textfield1.setBounds(120,10,150,20);
14        add(textfield1);
15        boton1=new JButton("Aceptar");
16        boton1.setBounds(10,80,100,30);
17        add(boton1);
18        boton1.addActionListener(this);
19    }
20
21    public void actionPerformed(ActionEvent e) {
22        if (e.getSource()==boton1) {
23            String cad=textfield1.getText();
24            setTitle(cad);
25        }
26    }
27
28    public static void main(String[] ar) {
29        Formulario formulario1=new Formulario();
30        formulario1.setBounds(0,0,350,170);
31        formulario1.setVisible(true);
32        formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
33    }
34 }
```



**Usuario:**

**Aceptar**

Definimos los tres objetos que colaboran con nuestra aplicación:

```
public class Formulario extends JFrame implements ActionListener{  
    private JTextField textfield1;  
    private JLabel label1;  
    private JButton boton1;
```

En el constructor creamos los tres objetos y los ubicamos:

```
public Formulario() {  
    setLayout(null);  
    label1=new JLabel("Usuario:");  
    label1.setBounds(10,10,100,30);  
    add(label1);  
    textfield1=new JTextField();  
    textfield1.setBounds(120,10,150,20);  
    add(textfield1);  
    boton1=new JButton("Aceptar");  
    boton1.setBounds(10,80,100,30);  
    add(boton1);  
    boton1.addActionListener(this);  
}
```

En el método actionPerformed se verifica si se presionó el objeto JButton, en caso afirmativo extraemos el contenido del control JTextField mediante el método getText:

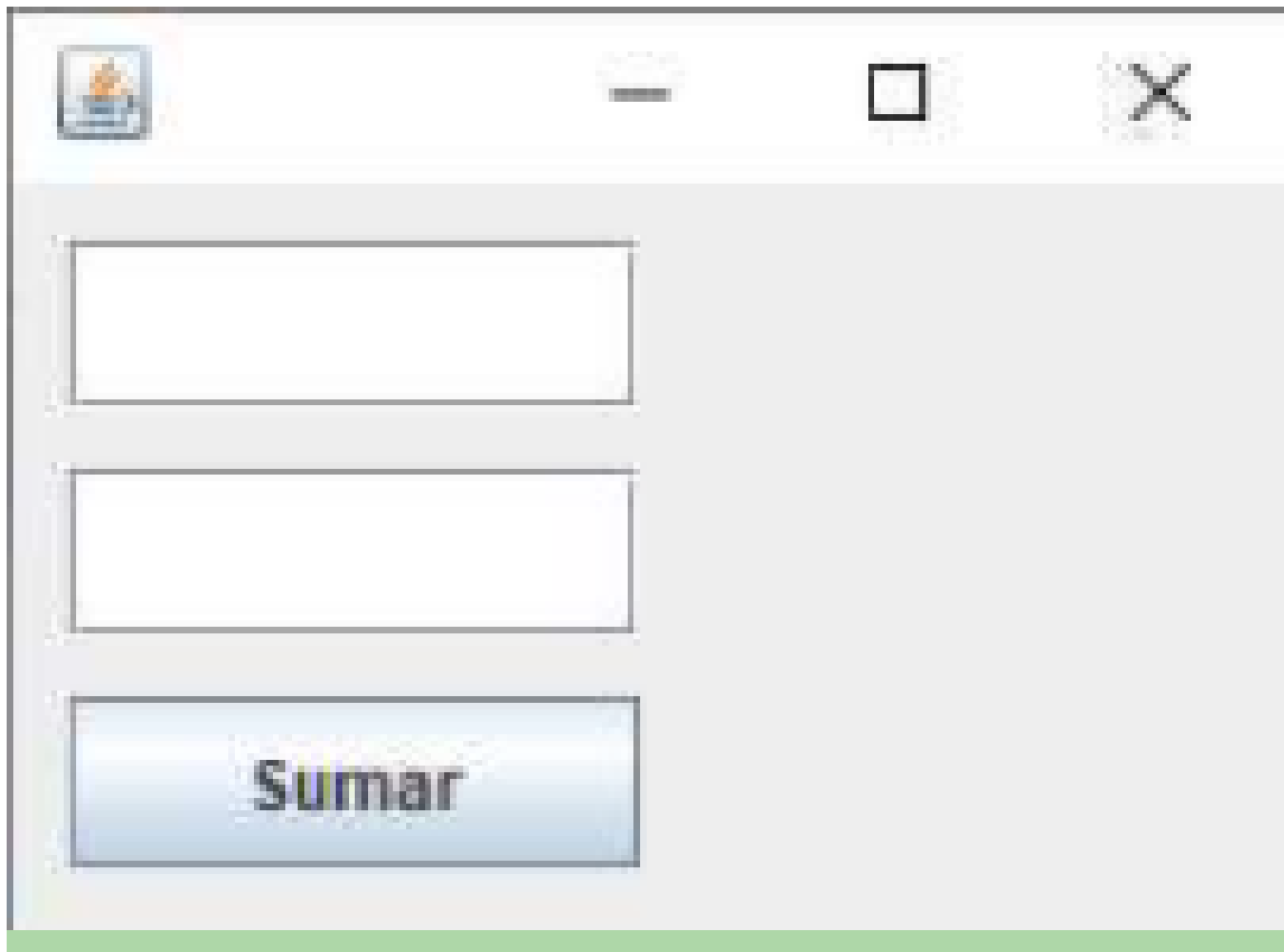
```
public void actionPerformed(ActionEvent e) {  
    if (e.getSource()==boton1) {  
        String cad=textfield1.getText();  
        setTitle(cad);  
    }  
}
```

En la variable auxiliar cad almacenamos temporalmente el contenido del JTextField y seguidamente actualizamos el título del control JFrame.

## **Problema 2:**

Confeccionar un programa que permita introducir dos números en controles de tipo JTextField, luego sumar los dos valores ingresados y mostrar la suma en la barra del título del control JFrame.





The image shows a window with a light gray title bar. On the left of the title bar is a small icon of a house with a flame. On the right are three standard window control buttons: a minus sign, a square, and an 'X'. The main area of the window is light gray. On the left side of this area, there are two empty rectangular input fields stacked vertically. Below these fields is a button with a blue gradient and the word 'Sumar' in black text. The entire window is set against a white background.


## Programa:

```
import javax.swing.*;
import java.awt.event.*;
public class Formulario extends JFrame implements ActionListener{
    private JTextField textfield1,textfield2;
    private JButton boton1;
    public Formulario() {
        setLayout(null);
        textfield1=new JTextField();
        textfield1.setBounds(10,10,100,30);
        add(textfield1);
        textfield2=new JTextField();
        textfield2.setBounds(10,50,100,30);
        add(textfield2);
        boton1=new JButton("Sumar");
        boton1.setBounds(10,90,100,30);
        add(boton1);
        boton1.addActionListener(this);
    }
```

```
public void actionPerformed(ActionEvent e) {  
    if (e.getSource()==boton1) {  
        String cad1=textfield1.getText();  
        String cad2=textfield2.getText();  
        int x1=Integer.parseInt(cad1);  
        int x2=Integer.parseInt(cad2);  
        int suma=x1+x2;  
        String total=String.valueOf(suma);  
        setTitle(total);  
    }  
}
```

```
public static void main(String[] ar) {  
    Formulario formulario1=new Formulario();  
    formulario1.setBounds(0,0,240,170);  
    formulario1.setVisible(true);  
    formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}  
}
```

```
1 import javax.swing.*;
2 import java.awt.event.*;
3 public class Formulario extends JFrame implements ActionListener{
4     private JTextField textfield1;
5     private JLabel label1;
6     private JButton boton1;
7     public Formulario() {
8         setLayout(null);
9         label1=new JLabel("Usuario:");
10        label1.setBounds(10,10,100,30);
11        add(label1);
12        textfield1=new JTextField();
13        textfield1.setBounds(120,10,150,20);
14        add(textfield1);
15        boton1=new JButton("Aceptar");
16        boton1.setBounds(10,80,100,30);
17        add(boton1);
18        boton1.addActionListener(this);
19    }
20
21    public void actionPerformed(ActionEvent e) {
22        if (e.getSource()==boton1) {
23            String cad=textfield1.getText();
24            setTitle(cad);
25        }
26    }
27
28    public static void main(String[] ar) {
29        Formulario formulario1=new Formulario();
30        formulario1.setBounds(0,0,350,170);
31        formulario1.setVisible(true);
32        formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
33    }
34 }
```

—□×

Usuario:

Aceptar

Definimos los tres objetos:

```
public class Formulario extends JFrame implements ActionListener{  
    private JTextField textfield1,textfield2;  
    private JButton boton1;
```

En el método actionPerformed es donde debemos sumar los valores ingresados en los controles de tipo JTextField. Para extraer el contenido de los controles debemos extraerlos con el método getText:

```
String cad1=textfield1.getText();  
String cad2=textfield2.getText();
```

Como debemos sumar numéricamente los valores ingresados debemos convertir el contenido de las variables cad1 y cad2 a tipo de dato int:

```
int x1=Integer.parseInt(cad1);  
int x2=Integer.parseInt(cad2);
```

El método `parseInt` de la clase `Integer` retorna el contenido de `cad1` convertido a `int` (provoca un error si introducimos caracteres en el control `JTextField` en lugar de números)

Una vez que tenemos los dos valores en formato numérico procedemos a sumarlos y almacenar el resultado en otra variable auxiliar:

```
int suma=x1+x2;
```

Ahora tenemos que mostrar el valor almacenado en `suma` en la barra de títulos del control `JFrame`, pero como el método `setTitle` requiere un `String` como parámetro debemos convertirlo a tipo `String`:

```
String total=String.valueOf(suma);  
setTitle(total);
```

Como veremos de aquí en adelante es muy común la necesidad de convertir String a enteros y enteros a String:

De String a int:

```
int x1=Integer.parseInt(cad1);
```

De int a String:

```
String total=String.valueOf(suma);
```



## Problemas propuestos

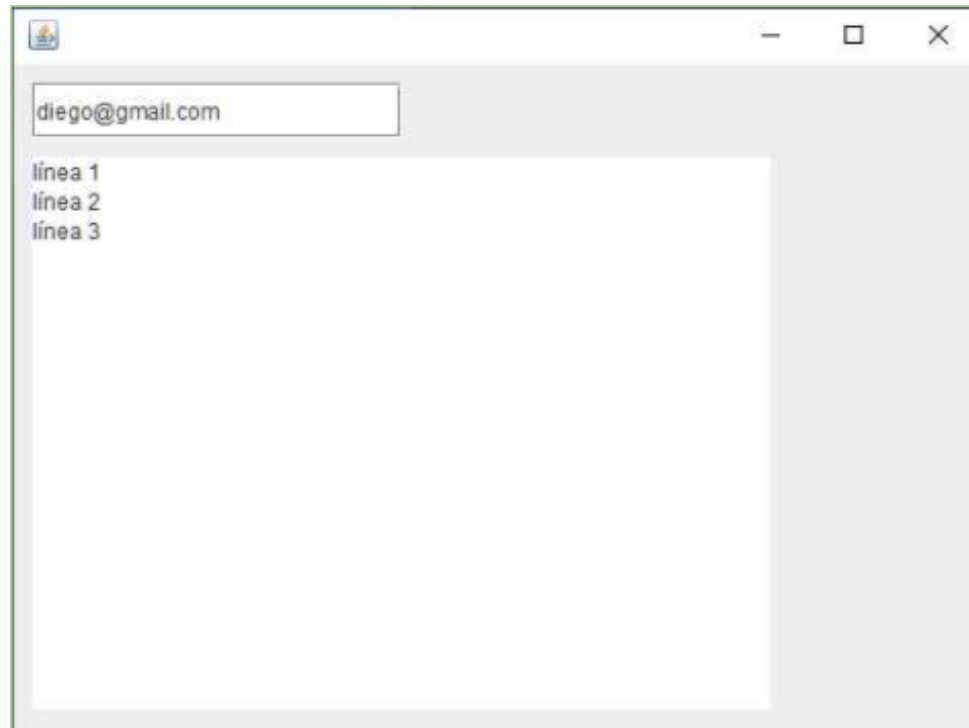
Introducir el nombre de usuario y clave en controles de tipo JTextField. Si se introducen las cadena (usuario: "juan", clave: "abc123") luego mostrar en el título del JFrame el mensaje "Correcto" en caso contrario mostrar el mensaje "Incorrecto".

## 35 - Swing - JTextArea

El control de tipo JTextArea permite ingresar múltiples líneas, a diferencia del control de tipo JTextField.

### Problema 1:

Confeccionar un programa que permita ingresar un mail en un control de tipo JTextField y el cuerpo del mail en un control de tipo JTextArea.



## Programa:

```
import javax.swing.*;

public class Formulario extends JFrame{
    private JTextField textfield1;
    private JTextArea textarea1;
    public Formulario() {
        setLayout(null);
        textfield1=new JTextField();
        textfield1.setBounds(10,10,200,30);
        add(textfield1);
        textarea1=new JTextArea();
        textarea1.setBounds(10,50,400,300);
        add(textarea1);
    }

    public static void main(String[] ar) {
        Formulario formulario1=new Formulario();
        formulario1.setBounds(0,0,540,400);
        formulario1.setVisible(true);
        formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

 \*Formulario.java ×

```
1  import javax.swing.*;
2  public class Formulario extends JFrame{
3      private JTextField textfield1;
4      private JTextArea textarea1;
5  public Formulario() {
6      setLayout(null);
7      textfield1=new JTextField();
8      textfield1.setBounds(10,10,200,30);
9      add(textfield1);
10     textarea1=new JTextArea();
11     textarea1.setBounds(10,50,400,300);
12     add(textarea1);
13 }
14
15 public static void main(String[] ar) {
16     Formulario formulario1=new Formulario();
17     formulario1.setBounds(0,0,540,400);
18     formulario1.setVisible(true);
19     formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20 }
21 }
```



l  
j  
jh  
g  
t  
n  
t  
y  
dg  
g  
g  
y  
y  
u  
j  
j  
o  
l  
l

Como vemos crear un control de tipo JTextArea es similar a la creación de controles de tipo JTextField:

```
textarea1=new JTextArea();  
textarea1.setBounds(10,50,400,300);  
add(textarea1);
```

El inconveniente que tiene este control es que si ingresamos más texto que el que puede visualizar no aparecen las barras de scroll y no podemos ver los caracteres tipeados.

Para salvar el problema anterior debemos crear un objeto de la clase JScrollPane y añadir en su interior el objeto de la clase JTextArea, luego el programa definitivo es el siguiente:

```
import javax.swing.*;

public class Formulario extends JFrame{
    private JTextField textfield1;
    private JScrollPane scrollpane1;
    private JTextArea textarea1;
    public Formulario() {
        setLayout(null);
        textfield1=new JTextField();
        textfield1.setBounds(10,10,200,30);
        add(textfield1);
        textarea1=new JTextArea();
        scrollpane1=new JScrollPane(textarea1);
        scrollpane1.setBounds(10,50,400,300);
        add(scrollpane1);
    }

    public static void main(String[] ar) {
        Formulario formulario1=new Formulario();
        formulario1.setBounds(0,0,540,400);
        formulario1.setVisible(true);
        formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

```
1  import javax.swing.*;
2  public class Formulario extends JFrame{
3      private JTextField textfield1;
4      private JScrollPane scrollpanel1;
5      private JTextArea textareal;
6      public Formulario() {
7          setLayout(null);
8          textfield1=new JTextField();
9          textfield1.setBounds(10,10,200,30);
10         add(textfield1);
11         textareal=new JTextArea();
12         scrollpanel1=new JScrollPane(textareal);
13         scrollpanel1.setBounds(10,50,400,300);
14         add(scrollpanel1);
15     }
16
17     public static void main(String[] ar) {
18         Formulario formulario1=new Formulario();
19         formulario1.setBounds(0,0,540,400);
20         formulario1.setVisible(true);
21         formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
22     }
23 }
```





J  
g  
p  
s  
ñ  
w  
q  
q  
q  
a  
s  
s  
o  
d  
d  
ñ  
f  
f  
f

Declaramos los dos objetos:

```
private JScrollPane scrollpane1;  
private JTextArea textarea1;
```

Primero creamos el objeto de la clase JTextArea:

```
textarea1=new JTextArea();
```

Seguidamente creamos el objeto de la clase JScrollPane y le pasamos como parámetro el objeto de la clase JTextArea:

```
scrollpane1=new JScrollPane(textarea1);
```

Definimos la posición y tamaño del control de tipo JScrollPane (y no del control JTextArea):

```
scrollpane1.setBounds(10,50,400,300);
```

Finalmente añadimos el control de tipo JScrollPane al JFrame:

```
add(scrollpane1);
```

## **Problema 2:**

Confeccionar un programa que permita introducir en un control de tipo JTextArea una carta. Luego al presionar un botón mostrar un mensaje si la carta contiene el String "java".

## Programa:

```
import javax.swing.*;
import java.awt.event.*;
public class Formulario extends JFrame implements ActionListener{
    private JScrollPane scrollpane1;
    private JTextArea textarea1;
    private JButton boton1;
    public Formulario() {
        setLayout(null);
        textarea1=new JTextArea();
        scrollpane1=new JScrollPane(textarea1);
        scrollpane1.setBounds(10,10,300,200);
        add(scrollpane1);
        boton1=new JButton("Verificar");
        boton1.setBounds(10,260,100,30);
        add(boton1);
        boton1.addActionListener(this);
    }
```

```
public void actionPerformed(ActionEvent e) {  
    if (e.getSource()==boton1) {  
        String texto=textarea1.getText();  
        if (texto.indexOf("java")!= -1) {  
            setTitle("Si contiene el texto \“java\“");  
        } else {  
            setTitle("No contiene el texto \“java\“");  
        }  
    }  
}
```

```
public static void main(String[] ar) {  
    Formulario formulario1=new Formulario();  
    formulario1.setBounds(0,0,400,380);  
    formulario1.setVisible(true);  
    formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}  
}
```

```
1 import javax.swing.*;
2 import java.awt.event.*;
3 public class Formulario extends JFrame implements ActionListener{
4     private JScrollPane scrollpanel1;
5     private JTextArea textareal;
6     private JButton boton1;
7     public Formulario() {
8         setLayout(null);
9         textareal=new JTextArea();
10        scrollpanel1=new JScrollPane(textareal);
11        scrollpanel1.setBounds(10,10,300,200);
12        add(scrollpanel1);
13        boton1=new JButton("Verificar");
14        boton1.setBounds(10,260,100,30);
15        add(boton1);
16        boton1.addActionListener(this);
17    }
18    public void actionPerformed(ActionEvent e) {
19        if (e.getSource()==boton1) {
20            String texto=textareal.getText();
21            if (texto.indexOf("java")!= -1) {
22                setTitle("Si contiene el texto \"java\"");
23            } else {
24                setTitle("No contiene el texto \"java\"");
25            }
26        }
27    }
28
29
30    public static void main(String[] ar) {
31        Formulario formulario1=new Formulario();
32        formulario1.setBounds(0,0,400,380);
33        formulario1.setVisible(true);
34        formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
35    }
```



Si contiene el texto "java"



asignatura de programacion veremos el lenguaje java



**Verificar**

Cuando se presiona el botón se ejecuta el método `actionPerformed` y procedemos a extraer el contenido del control `TextArea` a través del método `getText`:

```
String texto=textarea1.getText();
```

Luego mediante el método `indexOf` de la clase `String` verificamos si el `String` "java" está contenido en la variable `texto`:

```
if (texto.indexOf("java")!= -1) {  
    setTitle("Si contiene el texto \\"java\\");  
} else {  
    setTitle("No contiene el texto \\"java\\");  
}
```

Si queremos introducir una comilla doble dentro de un `String` de Java debemos antecederle la barra invertida (luego dicho caracter no se lo considera parte del `String`):

```
setTitle("Si contiene el texto \\"java\\");
```



## **Problema propuesto**

Disponer dos controles de tipo JTextArea, luego al presionar un botón verificar si tienen exactamente el mismo contenido.

## 36 - Swing - JComboBox

El control JComboBox permite seleccionar un String de una lista.

Para inicializar los String que contendrá el JComboBox debemos llamar al método addItem tantas veces como elementos queremos cargar.

Un evento muy útil con este control es cuando el operador selecciona un Item de la lista. Para capturar la selección de un item debemos implementar la interface ItemListener que contiene un método llamada itemStateChanged.

### **Problema 1:**

Cargar en un JComboBox los nombres de varios colores. Al seleccionar alguno mostrar en la barra de título del JFrame el String seleccionado.

**Programa:**

```
import javax.swing.*;
import java.awt.event.*;

public class Formulario extends JFrame implements ItemListener{
    private JComboBox<String> combo1;
    public Formulario() {
        setLayout(null);
        combo1=new JComboBox<String>();
        combo1.setBounds(10,10,80,20);
        add(combo1);
        combo1.addItem("rojo");
        combo1.addItem("verde");
        combo1.addItem("azul");
        combo1.addItem("amarillo");
        combo1.addItem("negro");
        combo1.addItemListener(this);
    }

    public void itemStateChanged(ItemEvent e) {
        if (e.getSource()==combo1) {
            String seleccionado=(String)combo1.getSelectedItem();
            setTitle(seleccionado);
        }
    }
}
```

```
public static void main(String[] ar) {  
    Formulario formulario1=new Formulario();  
    formulario1.setBounds(0,0,200,150);  
    formulario1.setVisible(true);  
    formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}  
}
```

```
1 import javax.swing.*;
2 import java.awt.event.*;
3 public class Formulario extends JFrame implements ItemListener{
4     private JComboBox<String> combo1;
5     public Formulario() {
6         setLayout(null);
7         combo1=new JComboBox<String>();
8         combo1.setBounds(10,10,80,20);
9         add(combo1);
10        combo1.addItem("rojo");
11        combo1.addItem("verde");
12        combo1.addItem("azul");
13        combo1.addItem("amarillo");
14        combo1.addItem("negro");
15        combo1.addItemListener(this);
16    }
17
18    public void itemStateChanged(ItemEvent e) {
19        if (e.getSource()==combo1) {
20            String seleccionado=(String)combo1.getSelectedItem();
21            setTitle(seleccionado);
22        }
23    }
24
25    public static void main(String[] ar) {
26        Formulario formulario1=new Formulario();
27        formulario1.setBounds(0,0,200,150);
28        formulario1.setVisible(true);
29        formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
30    }
31 }
```



rojo



rojo



Indicamos a la clase que implementaremos la interface ItemListener:

```
public class Formulario extends JFrame implements ItemListener{
```

Declaramos un objeto de la clase JComboBox:

```
private JComboBox<String> combo1;
```

Debemos anteceder el tipo de datos que administra el JComboBox <String>, luego veremos el concepto de genéricos para comprender completamente el porque anteceder entre los símbolos menor y mayor el tipo de dato String.

En el constructor creamos el objeto de la clase JComboBox:

```
combo1=new JComboBox<String>();
```

Posicionamos el control:

```
combo1.setBounds(10,10,80,20);
```

Añadimos el control al JFrame:

```
add(combo1);
```

Añadimos los String al JComboBox:

```
combo1.addItem("rojo");  
combo1.addItem("verde");  
combo1.addItem("azul");  
combo1.addItem("amarillo");  
combo1.addItem("negro");
```

Asociamos la clase que capturará el evento de cambio de item (con this indicamos que esta misma clase capturará el evento):

```
combo1.addItemListener(this);
```

El método `itemStateChanged` que debemos implementar de la interface `ItemListener` tiene la siguiente sintaxis:

```
public void itemStateChanged(ItemEvent e) {  
    if (e.getSource()==combo1) {  
        String seleccionado=(String)combo1.getSelectedItem();  
        setTitle(seleccionado);  
    }  
}
```



Para extraer el contenido del item seleccionado llamamos al método `getSelectedItem()` el cual retorna un objeto de la clase `Object` por lo que debemos indicarle que lo transforme en `String`:

```
String seleccionado=(String)combo1.getSelectedItem();
```

## **Problema 2:**

Disponer tres controles de tipo `JComboBox` con valores entre 0 y 255 (cada uno representa la cantidad de rojo, verde y azul). Luego al presionar un botón pintar el mismo con el color que se genera combinando los valores de los `JComboBox`.

## Programa:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Formulario extends JFrame implements ActionListener{
    private JLabel label1,label2,label3;
    private JComboBox<String> combo1,combo2,combo3;
    private JButton boton1;
    public Formulario() {
        setLayout(null);
        label1=new JLabel("Rojo:");
        label1.setBounds(10,10,100,30);
        add(label1);
        combo1=new JComboBox<String>();
        combo1.setBounds(120,10,50,30);
        for(int f=0;f<=255;f++) {
            combo1.addItem(String.valueOf(f));
        }
    }
}
```

```
add(combo1);  
label2=new JLabel("Verde:");  
label2.setBounds(10,50,100,30);  
add(label2);  
combo2=new JComboBox<String>();  
combo2.setBounds(120,50,50,30);  
for(int f=0;f<=255;f++) {  
    combo2.addItem(String.valueOf(f));  
}
```

```
add(combo2);
label3=new JLabel("Azul:");
label3.setBounds(10,90,100,30);
add(label3);
combo3=new JComboBox<String>();
combo3.setBounds(120,90,50,30);
for(int f=0;f<=255;f++) {
    combo3.addItem(String.valueOf(f));
}
add(combo3);
boton1=new JButton("Fijar Color");
boton1.setBounds(10,130,100,30);
add(boton1);
boton1.addActionListener(this);
}
```

```
public void actionPerformed(ActionEvent e) {  
    if (e.getSource()==boton1) {  
        String cad1=(String)combo1.getSelectedItem();  
        String cad2=(String)combo2.getSelectedItem();  
        String cad3=(String)combo3.getSelectedItem();  
        int rojo=Integer.parseInt(cad1);  
        int verde=Integer.parseInt(cad2);  
        int azul=Integer.parseInt(cad3);  
        Color color1=new Color(rojo,verde,azul);  
        boton1.setBackground(color1);  
    }  
}  
  
public static void main(String[] ar) {  
    Formulario formulario1=new Formulario();  
    formulario1.setBounds(0,0,400,300);  
    formulario1.setVisible(true);  
    formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}  
}
```

```
1 import javax.swing.*;
2 import java.awt.*;
3 import java.awt.event.*;
4 public class Formulario extends JFrame implements ActionListener {
5     private JLabel label1,label2,label3;
6     private JComboBox<String> combo1,combo2,combo3;
7     private JButton boton1;
8     public Formulario() {
9         setLayout(null);
10        label1=new JLabel("Rojo:");
11        label1.setBounds(10,10,100,30);
12        add(label1);
13        combo1=new JComboBox<String>();
14        combo1.setBounds(120,10,50,30);
15        for(int f=0;f<=255;f++) {
16            combo1.addItem(String.valueOf(f));
17        }
18        add(combo1);
19        label2=new JLabel("Verde:");
20        label2.setBounds(10,50,100,30);
21        add(label2);
22        combo2=new JComboBox<String>();
23        combo2.setBounds(120,50,50,30);
24        for(int f=0;f<=255;f++) {
25            combo2.addItem(String.valueOf(f));
26        }
27        add(combo2);
28        label3=new JLabel("Azul:");
29        label3.setBounds(10,90,100,30);
30        add(label3);
31        combo3=new JComboBox<String>();
32        combo3.setBounds(120,90,50,30);
33        for(int f=0;f<=255;f++) {
34            combo3.addItem(String.valueOf(f));
35        }
36        add(combo3);
37        boton1=new JButton("Fijar Color");
38        boton1.setBounds(10,130,100,30);
39        add(boton1);
```



Rojo:

 ▼

Verde:

 ▼

Azul:

 ▼

Fijar Color

Importamos el paquete java.awt ya que el mismo contiene la clase Color:

```
import java.awt.*;
```

Implementaremos la interface ActionListener ya que tenemos que cambiar el color del botón cuando se lo presione y no haremos actividades cuando cambiemos items de los controles JComboBox:

```
public class Formulario extends JFrame implements ActionListener{
```

Definimos los siete objetos requeridos en esta aplicación:

```
private JLabel label1,label2,label3;  
private JComboBox<String> combo1,combo2,combo3;  
private JButton boton1;
```

En el constructor creamos los objetos, primero el control label1 de la clase JLabel:

```
label1=new JLabel("Rojo:");  
label1.setBounds(10,10,100,30);  
add(label1);
```



Lo mismo hacemos con el objeto combo1:

```
combo1=new JComboBox<String>();  
combo1.setBounds(120,10,50,30);
```

Para añadir los 255 elementos del JComboBox disponemos un for y previa a llamar al método addItem convertimos el entero a String:

```
for(int f=0;f<=255;f++) {  
    combo1.addItem(String.valueOf(f));  
}  
add(combo1);
```

En el método actionPerformed cuando detectamos que se presionó el botón procedemos a extraer los tres item seleccionados:

```
public void actionPerformed(ActionEvent e) {  
    if (e.getSource()==boton1) {  
        String cad1=(String)combo1.getSelectedItem();  
        String cad2=(String)combo2.getSelectedItem();  
        String cad3=(String)combo3.getSelectedItem();
```

Los convertimos a entero:

```
int rojo=Integer.parseInt(cad1);  
int verde=Integer.parseInt(cad2);  
int azul=Integer.parseInt(cad3);
```

y creamos finalmente un objeto de la clase Color, el constructor de la clase Color requiere que le pasemos tres valores de tipo int:

```
Color color1=new Color(rojo,verde,azul);
```

Para cambiar el color de fondo del control JButton debemos llamar al método setBackground y pasarle el objeto de la clase Color:

```
boton1.setBackground(color1);
```

## **Problema propuesto**

Solicitar el nombre de una persona y seleccionar de un control JComboBox un país. Al presionar un botón mostrar en la barra del título del JFrame el nombre introducido y el país seleccionado.

## 37 - Swing - JMenuBar, JMenu, JMenuItem

Ahora veremos como crear un menú de opciones y la captura de eventos de los mismos.

Cuando necesitamos implementar un menú horizontal en la parte superior de un JFrame requerimos de un objeto de la clase JMenuBar, uno o más objetos de la clase JMenu y por último objetos de la clase JMenuItem.

Para la captura de eventos debemos implementar la interface ActionListener y asociarlo a los controles de tipo JMenuItem, el mismo se dispara al presionar con el mouse el JMenuItem.

### **Problema 1:**

Confeccionaremos un menú de opciones que contenga tres opciones que permita cambiar el color de fondo del JFrame a los colores: rojo, verde y azul.

## Programa:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Formulario extends JFrame implements ActionListener{
    private JMenuBar mb;
    private JMenu menu1;
    private JMenuItem mi1,mi2,mi3;
    public Formulario() {
        setLayout(null);
        mb=new JMenuBar();
        setJMenuBar(mb);
        menu1=new JMenu("Opciones");
        mb.add(menu1);
        mi1=new JMenuItem("Rojo");
        mi1.addActionListener(this);
        menu1.add(mi1);
        mi2=new JMenuItem("Verde");
        mi2.addActionListener(this);
        menu1.add(mi2);
        mi3=new JMenuItem("Azul");
        mi3.addActionListener(this);
        menu1.add(mi3);
    }
}
```

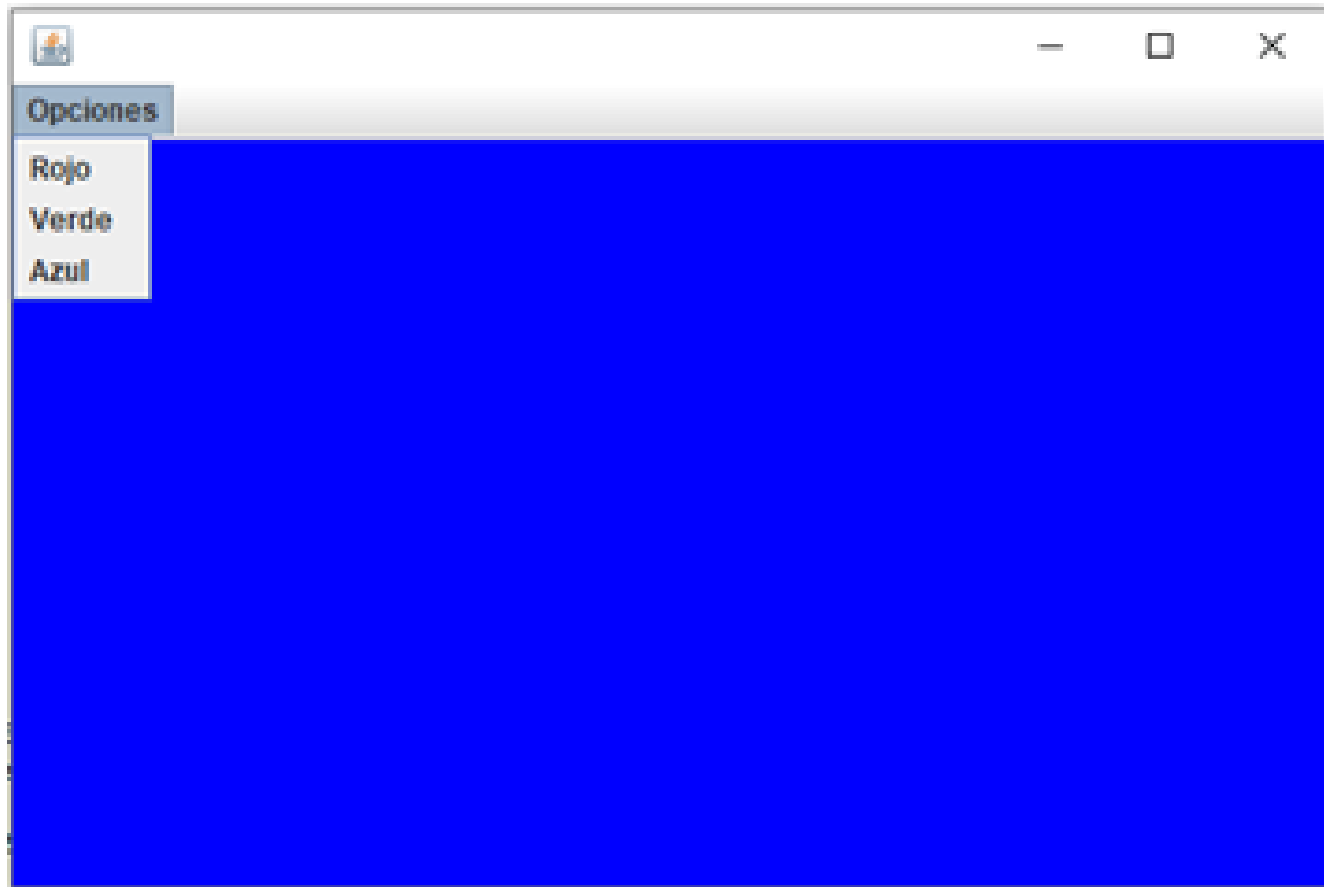
```
public void actionPerformed(ActionEvent e) {  
    Container f=this.getContentPane();  
    if (e.getSource()==mi1) {  
        f.setBackground(new Color(255,0,0));  
    }  
    if (e.getSource()==mi2) {  
        f.setBackground(new Color(0,255,0));  
    }  
    if (e.getSource()==mi3) {  
        f.setBackground(new Color(0,0,255));  
    }  
}
```

```
public static void main(String[] ar) {  
    Formulario formulario1=new Formulario();  
    formulario1.setBounds(10,20,300,200);  
    formulario1.setVisible(true);  
    formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}  
}
```

```

1 import javax.swing.*;
2 import java.awt.*;
3 import java.awt.event.*;
4 public class Formulario extends JFrame implements ActionListener {
5     private JMenuBar mb;
6     private JMenu menu1;
7     private JMenuItem mi1, mi2, mi3;
8     public Formulario() {
9         setLayout(null);
10        mb=new JMenuBar();
11        setJMenuBar(mb);
12        menu1=new JMenu("Opciones");
13        mb.add(menu1);
14        mi1=new JMenuItem("Rojo");
15        mi1.addActionListener(this);
16        menu1.add(mi1);
17        mi2=new JMenuItem("Verde");
18        mi2.addActionListener(this);
19        menu1.add(mi2);
20        mi3=new JMenuItem("Azul");
21        mi3.addActionListener(this);
22        menu1.add(mi3);
23    }
24
25    public void actionPerformed(ActionEvent e) {
26        Container f=this.getContentPane();
27        if (e.getSource()==mi1) {
28            f.setBackground(new Color(255,0,0));
29        }
30        if (e.getSource()==mi2) {
31            f.setBackground(new Color(0,255,0));
32        }
33        if (e.getSource()==mi3) {
34            f.setBackground(new Color(0,0,255));
35        }
36    }
37
38    public static void main(String[] ar) {
39        Formulario formulario1=new Formulario();

```





Importamos el paquete javax.swing ya que en el mismo se encuentran las tres clases JMenuBar, JMenu y JMenuItem:

```
import javax.swing.*;
```

Importamos java.awt donde se encuentra la clase Color:

```
import java.awt.*;
```

Para la captura de eventos mediante la interface ActionListener debemos importar el paquete java.awt.event:

```
import java.awt.event.*;
```

Declaramos la clase Formulario, heredamos de la clase JFrame e indicamos que implementaremos la interface ActionListener:

```
public class Formulario extends JFrame implements ActionListener{
```

Definimos un objeto de la clase JMenuBar (no importa que tan grande sea un menú de opciones solo se necesitará un solo objeto de esta clase):

```
private JMenuBar mb;
```

Definimos un objeto de la clase JMenu (esta clase tiene por objeto desplegar un conjunto de objetos de tipo JMenuItem u otros objetos de tipo JMenu:

```
private JMenu menu1;
```

Definimos tres objetos de la clase JMenuItem (estos son los que disparan eventos cuando el operador los selecciona:

```
private JMenuItem mi1,mi2,mi3;
```

En el constructor creamos primero el objeto de la clase JMenuBar y lo asociamos al JFrame llamando al método setJMenuBar:

```
mb=new JMenuBar();  
setJMenuBar(mb);
```

Seguidamente creamos un objeto de la clase JMenu, en el constructor pasamos el String que debe mostrar y asociamos dicho JMenu con el JMenuBar llamando al método add de objeto de tipo JMenuBar (Es decir el objeto de la clase JMenu colabora con la clase JMenuBar):

```
menu1=new JMenu("Opciones");  
mb.add(menu1);
```

Ahora comenzamos a crear los objetos de la clase JMenuItem y los añadimos al objeto de la clase JMenu (también mediante la llamada al método addActionListener indicamos al JMenuItem que objeto procesará el clic):

```
mi1=new JMenuItem("Rojo");  
mi1.addActionListener(this);  
menu1.add(mi1);
```

Lo mismo hacemos para los otros dos JMenuItem:

```
mi2=new JMenuItem("Verde");  
mi2.addActionListener(this);  
menu1.add(mi2);  
mi3=new JMenuItem("Azul");  
mi3.addActionListener(this);  
menu1.add(mi3);
```

En el método actionPerformed primero obtenemos la referencia al panel asociado con el JFrame:

```
public void actionPerformed(ActionEvent e) {  
    Container f=this.getContentPane();
```

Luego mediante if verificamos cual de los tres JMenuItem fue seleccionado y a partir de esto llamamos al método setBackground del objeto de la clase Container):

```
        if (e.getSource()==mi1) {  
            f.setBackground(new Color(255,0,0));  
        }  
        if (e.getSource()==mi2) {  
            f.setBackground(new Color(0,255,0));  
        }  
        if (e.getSource()==mi3) {  
            f.setBackground(new Color(0,0,255));  
        }  
    }
```

## **Problema 2:**

Confeccionaremos un menú de opciones que contenga además del JMenu de la barra otros dos objetos de la clase JMenu que dependan del primero.

Uno debe mostrar dos JMenuItem que permitan modificar el tamaño del JFrame y el segundo también debe mostrar dos JMenuItem que permitan cambiar el color de fondo.

## Programa:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Formulario extends JFrame implements ActionListener{
    private JMenuBar mb;
    private JMenu menu1,menu2,menu3;
    private JMenuItem mi1,mi2,mi3,mi4;
    public Formulario() {
        setLayout(null);
        mb=new JMenuBar();
        setJMenuBar(mb);
        menu1=new JMenu("Opciones");
        mb.add(menu1);
        menu2=new JMenu("Tamaño de la ventana");
        menu1.add(menu2);
        menu3=new JMenu("Color de fondo");
        menu1.add(menu3);
```

```
mi1=new JMenuItem("640*480");  
menu2.add(mi1);  
mi1.addActionListener(this);  
mi2=new JMenuItem("1024*768");  
menu2.add(mi2);  
mi2.addActionListener(this);  
mi3=new JMenuItem("Rojo");  
menu3.add(mi3);  
mi3.addActionListener(this);  
mi4=new JMenuItem("Verde");  
menu3.add(mi4);  
mi4.addActionListener(this);  
}
```



```
public void actionPerformed(ActionEvent e) {
    if (e.getSource()==mi1) {
        setSize(640,480);
    }
    if (e.getSource()==mi2) {
        setSize(1024,768);
    }
    if (e.getSource()==mi3) {
        getContentPane().setBackground(new Color(255,0,0));
    }
    if (e.getSource()==mi4) {
        getContentPane().setBackground(new Color(0,255,0));
    }
}

public static void main(String[] ar) {
    Formulario formulario1=new Formulario();
    formulario1.setBounds(0,0,300,200);
    formulario1.setVisible(true);

    formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
```

```

1 import javax.swing.*;
2 import java.awt.*;
3 import java.awt.event.*;
4 public class Formulario extends JFrame implements ActionListener {
5     private JMenuBar mb;
6     private JMenu menu1, menu2, menu3;
7     private JMenuItem mi1, mi2, mi3, mi4;
8     public Formulario() {
9         setLayout(null);
10        mb = new JMenuBar();
11        setJMenuBar(mb);
12        menu1 = new JMenu("Opciones");
13        mb.add(menu1);
14        menu2 = new JMenu("Tamaño de la ventana");
15        menu1.add(menu2);
16        menu3 = new JMenu("Color de fondo");
17        menu1.add(menu3);
18        mi1 = new JMenuItem("640*480");
19        menu2.add(mi1);
20        mi1.addActionListener(this);
21        mi2 = new JMenuItem("1024*768");
22        menu2.add(mi2);
23        mi2.addActionListener(this);
24        mi3 = new JMenuItem("Rojo");
25        menu3.add(mi3);
26        mi3.addActionListener(this);
27        mi4 = new JMenuItem("Verde");
28        menu3.add(mi4);
29        mi4.addActionListener(this);
30    }
31
32    public void actionPerformed(ActionEvent e) {
33        if (e.getSource() == mi1) {
34            setSize(640, 480);
35        }
36        if (e.getSource() == mi2) {
37            setSize(1024, 768);
38        }
39        if (e.getSource() == mi3) {

```



Definimos un objeto de la clase JMenuBar, 3 objetos de la clase JMenu y finalmente 4 objetos de la clase JMenuItem:

```
private JMenuBar mb;  
private JMenu menu1,menu2,menu3;  
private JMenuItem mi1,mi2,mi3,mi4;
```

Es importante notar el orden de creación de los objetos y como los relacionamos unos con otros. Primero creamos el JMenuBar y lo asociamos con el JFrame:

```
mb=new JMenuBar();  
setJMenuBar(mb);
```

Creamos el primer JMenu y lo pasamos como parámetro al JMenuBar mediante el método add:

```
menu1=new JMenu("Opciones");  
mb.add(menu1);
```

Ahora creamos el segundo objeto de la clase JMenu y lo asociamos con el primer JMenu creado:

```
menu2=new JMenu("Tamaño de la ventana");  
menu1.add(menu2);
```

En forma similar creamos el tercer objeto de la clase JMenu y lo asociamos con el primer JMenu creado:

```
menu3=new JMenu("Color de fondo");  
menu1.add(menu3);
```

Finalmente comenzamos a crear los objetos de la clase JMenuItem y los dos primeros los asociamos con el segundo JMenu:

```
mi1=new JMenuItem("640*480");  
menu2.add(mi1);  
mi1.addActionListener(this);  
mi2=new JMenuItem("1024*768");  
menu2.add(mi2);  
mi2.addActionListener(this);
```

También hacemos lo mismo con los otros dos objetos de tipo JMenuItem pero ahora los asociamos con el tercer JMenu:

```
mi3=new JMenuItem("Rojo");  
menu3.add(mi3);  
mi3.addActionListener(this);  
mi4=new JMenuItem("Verde");  
menu3.add(mi4);  
mi4.addActionListener(this);
```

En el método `actionPerformed` si se presiona el `mi1` procedemos a redimensionar el `JFrame` llamando al método `setSize` y le pasamos dos parámetros que representan el nuevo ancho y alto de la ventana:

```
if (e.getSource()==mi1) {  
    setSize(640,480);  
}
```

De forma similar si se presiona el segundo `JMenuItem` cambiamos el tamaño de la ventana a 1024 píxeles por 768:

```
if (e.getSource()==mi2) {  
    setSize(1024,768);  
}
```

Para cambiar de color de forma similar al problema anterior mediante el método `getContentPane` obtenemos la referencia al objeto de la clase `Container` y llamamos al método `setBackground` para fijar un nuevo color de fondo:

```
if (e.getSource()==mi3) {  
    getContentPane().setBackground(new Color(255,0,0));  
}  
if (e.getSource()==mi4) {  
    getContentPane().setBackground(new Color(0,255,0));  
}
```



## **Problema propuesto**

Mediante dos controles de tipo JTextField introducir dos números. Crear un menú que contenga una opción que redimensione el JFrame con los valores introducidos por teclado. Finalmente disponer otra opción que finalice el programa (Finalizamos un programa java llamando al método exit de la clase System: System.exit(0)).

## 38 - Swing - JCheckBox

El control JCheckBox permite implementar un cuadro de selección (básicamente un botón de dos estados)

Problema 1:

Confeccionar un programa que muestre 3 objetos de la clase JCheckBox con etiquetas de tres idiomas. Cuando se lo selecciona mostrar en el título del JFrame todos los JCheckBox seleccionados hasta el momento.

## Programa:

```
import javax.swing.*.*;
import javax.swing.event.*;
public class Formulario extends JFrame implements ChangeListener{
    private JCheckBox check1,check2,check3;
    public Formulario() {
        setLayout(null);
        check1=new JCheckBox("Inglés");
        check1.setBounds(10,10,150,30);
        check1.addChangeListener(this);
        add(check1);
        check2=new JCheckBox("Francés");
        check2.setBounds(10,50,150,30);
        check2.addChangeListener(this);
        add(check2);
        check3=new JCheckBox("Alemán");
        check3.setBounds(10,90,150,30);
        check3.addChangeListener(this);
        add(check3);
    }
}
```

```
public void stateChanged(ChangeEvent e){
    String cad="";
    if (check1.isSelected()==true) {
        cad=cad+"Inglés-";
    }
    if (check2.isSelected()==true) {
        cad=cad+"Francés-";
    }
    if (check3.isSelected()==true) {
        cad=cad+"Alemán-";
    }
    setTitle(cad);
}
```

```
public static void main(String[] ar) {
    Formulario formulario1=new Formulario();
    formulario1.setBounds(0,0,300,200);
    formulario1.setVisible(true);
    formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
```

```
1 import javax.swing.*;
2 import javax.swing.event.*;
3 public class Formulario extends JFrame implements ChangeListener{
4     private JCheckBox check1, check2, check3;
5     public Formulario() {
6         setLayout(null);
7         check1=new JCheckBox("Inglés");
8         check1.setBounds(10,10,150,30);
9         check1.addChangeListener(this);
10        add(check1);
11        check2=new JCheckBox("Francés");
12        check2.setBounds(10,50,150,30);
13        check2.addChangeListener(this);
14        add(check2);
15        check3=new JCheckBox("Alemán");
16        check3.setBounds(10,90,150,30);
17        check3.addChangeListener(this);
18        add(check3);
19    }
20
21    public void stateChanged(ChangeEvent e){
22        String cad="";
23        if (check1.isSelected()==true) {
24            cad=cad+"Inglés-";
25        }
26        if (check2.isSelected()==true) {
27            cad=cad+"Francés-";
28        }
29        if (check3.isSelected()==true) {
30            cad=cad+"Alemán-";
31        }
32        setTitle(cad);
33    }
34
35    public static void main(String[] ar) {
36        Formulario formulario1=new Formulario();
37        formulario1.setBounds(0,0,300,200);
38        formulario1.setVisible(true);
39        formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```



Inglés-Francés-Alemán-



Inglés



Francés



Alemán

Lo primero y más importante que tenemos que notar que para capturar el cambio de estado del JCheckBox hay que implementar la interface ChangeListener que se encuentra en el paquete:

```
import javax.swing.event.*;
```

y no en el paquete:

```
import java.awt.event.*
```

Cuando declaramos la clase JFrame indicamos que implementaremos la interface ChangeListener:

```
public class Formulario extends JFrame implements ChangeListener{  
Definimos tres objetos de la clase JCheckBox:
```

```
    private JCheckBox check1,check2,check3;
```

En el constructor creamos cada uno de los objetos de la clase JCheckBox y llamamos al método addChangeListener indicando quien procesará el evento de cambio de estado:

```
check1=new JCheckBox("Inglés");  
check1.setBounds(10,10,150,30);  
check1.addChangeListener(this);  
add(check1);
```



El método que debemos implementar de la interface ChangeListener es:

```
public void stateChanged(ChangeEvent e){
```

En este mediante tres if verificamos el estado de cada JCheckBox y concatenamos los String con los idiomas seleccionados:

```
String cad="";  
if (check1.isSelected()==true) {  
    cad=cad+"Inglés-";  
}  
if (check2.isSelected()==true) {  
    cad=cad+"Francés-";  
}  
if (check3.isSelected()==true) {  
    cad=cad+"Alemán-";  
}  
setTitle(cad);
```

## Problema 2:

Disponer un control JLabel que muestre el siguiente mensaje: "Esta de acuerdo con las normas del servicio?", luego un JCheckBox y finalmente un objeto de tipo JButton desactivado. Cuando se ticle el JCheckBox debemos activar el botón.

## Programa:

```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.event.*;
public class Formulario extends JFrame implements ActionListener,
ChangeListener{
    private JLabel label1;
    private JCheckBox check1;
    private JButton boton1;
    public Formulario() {
        setLayout(null);
        label1=new JLabel("Esta de acuerdo con las normas del servicio?");
        label1.setBounds(10,10,400,30);
        add(label1);
        check1=new JCheckBox("Acepto");
        check1.setBounds(10,50,100,30);
        check1.addChangeListener(this);
        add(check1);
        boton1=new JButton("Continuar");
        boton1.setBounds(10,100,100,30);
        add(boton1);
        boton1.addActionListener(this);
        boton1.setEnabled(false);
    }
}
```

```
public void stateChanged(ChangeEvent e) {  
    if (check1.isSelected()==true) {  
        boton1.setEnabled(true);  
    } else {  
        boton1.setEnabled(false);  
    }  
}
```

```
public void actionPerformed(ActionEvent e) {  
    if (e.getSource()==boton1) {  
        System.exit(0);  
    }  
}
```

```
public static void main(String[] ar) {  
    Formulario formulario1=new Formulario();  
    formulario1.setBounds(0,0,350,200);  
    formulario1.setVisible(true);  
    formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}
```

```
1⊖ import javax.swing.*;
2 import javax.swing.event.*;
3 import java.awt.event.*;
4 public class Formulario extends JFrame implements ActionListener, ChangeListener⌘
5     private JLabel label1;
6     private JCheckBox check1;
7     private JButton boton1;
8⊖ public Formulario() {
9     setLayout(null);
10    label1=new JLabel("Esta de acuerdo con las normas del servicio?");
11    label1.setBounds(10,10,400,30);
12    add(label1);
13    check1=new JCheckBox("Acepto");
14    check1.setBounds(10,50,100,30);
15    check1.addChangeListener(this);
16    add(check1);
17    boton1=new JButton("Continuar");
18    boton1.setBounds(10,100,100,30);
19    add(boton1);
20    boton1.addActionListener(this);
21    boton1.setEnabled(false);
22 }
23
24⊖ public void stateChanged(ChangeEvent e) {
25     if (check1.isSelected()==true) {
26         boton1.setEnabled(true);
27     } else {
28         boton1.setEnabled(false);
29     }
30 }
31
32⊖ public void actionPerformed(ActionEvent e) {
33     if (e.getSource()==boton1) {
34         System.exit(0);
35     }
36 }
37
38⊖ public static void main(String[] ar) {
39     Formulario formulario1=new Formulario();
```



Esta de acuerdo con las normas del servicio?



Acepto

Continuar

Importamos los paquetes donde se encuentran las interfaces para captura de eventos de objetos de tipo JButton y JCheckBox:

```
import javax.swing.event.*;  
import java.awt.event.*;
```

También importamos el paquete donde están definidas las clase JFrame, JButton y JCheckBox:

```
import javax.swing.*;
```

Como debemos implementar dos interfaces las debemos enumerar después de la palabra implements separadas por coma:

```
public class Formulario extends JFrame implements ActionListener,  
ChangeListener{
```

Definimos los tres objetos:

```
private JLabel label1;  
private JCheckBox check1;  
private JButton boton1;
```

En el constructor creamos el objeto de tipo JLabel:

```
public Formulario() {  
    setLayout(null);  
    label1=new JLabel("Esta de acuerdo con las normas del servicio?");  
    label1.setBounds(10,10,400,30);  
    add(label1);  
}
```



El objeto de tipo JCheckBox:

```
check1=new JCheckBox("Acepto");  
check1.setBounds(10,50,100,30);  
check1.addChangeListener(this);  
add(check1);
```

y también creamos el objeto de tipo JButton y llamando al método `setEnabled` con un valor `false` luego el botón aparece desactivo:

```
boton1=new JButton("Continuar");  
boton1.setBounds(10,100,100,30);  
add(boton1);  
boton1.addActionListener(this);  
boton1.setEnabled(false);
```

Cuando se cambia el estado del control JCheckBox se ejecuta el método `stateChanged` donde verificamos si está seleccionado procediendo a activar el botón en caso negativo lo desactivamos:

```
public void stateChanged(ChangeEvent e) {  
    if (check1.isSelected()==true) {  
        boton1.setEnabled(true);  
    } else {  
        boton1.setEnabled(false);  
    }  
}
```

El método `actionPerformed` se ejecuta cuando se presiona el objeto de tipo `JButton` (debe estar activo para poder presionarlo):

```
public void actionPerformed(ActionEvent e) {  
    if (e.getSource()==boton1) {  
        System.exit(0);  
    }  
}
```

## Problema propuesto

Disponer tres objetos de la clase JCheckBox con nombres de navegadores web. Cuando se presione un botón mostrar en el título del JFrame los programas seleccionados.

## 39 - Swing - JRadioButton

Otro control visual muy común es el JRadioButton que normalmente se muestran un conjunto de JRadioButton y permiten la selección de solo uno de ellos. Se los debe agrupar para que actúen en conjunto, es decir cuando se selecciona uno automáticamente se deben deseleccionar los otros.

### **Problema 1:**

Confeccionar un programa que muestre 3 objetos de la clase JRadioButton que permitan configurar el ancho y alto del JFrame.

## Programa:

```
import javax.swing.*;
import javax.swing.event.*;
public class Formulario extends JFrame implements ChangeListener{
    private JRadioButton radio1,radio2,radio3;
    private ButtonGroup bg;
    public Formulario() {
        setLayout(null);
        bg=new ButtonGroup();
        radio1=new JRadioButton("640*480");
        radio1.setBounds(10,20,100,30);
        radio1.addChangeListener(this);
        add(radio1);
        bg.add(radio1);
        radio2=new JRadioButton("800*600");
        radio2.setBounds(10,70,100,30);
        radio2.addChangeListener(this);
        add(radio2);
        bg.add(radio2);
        radio3=new JRadioButton("1024*768");
        radio3.setBounds(10,120,100,30);
        radio3.addChangeListener(this);
        add(radio3);
        bg.add(radio3);
    }
}
```

```
public void stateChanged(ChangeEvent e) {  
    if (radio1.isSelected()) {  
        setSize(640,480);  
    }  
    if (radio2.isSelected()) {  
        setSize(800,600);  
    }  
    if (radio3.isSelected()) {  
        setSize(1024,768);  
    }  
}
```

```
public static void main(String[] ar) {  
    Formulario formulario1=new Formulario();  
    formulario1.setBounds(0,0,350,230);  
    formulario1.setVisible(true);  
    formulario1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
}  
}
```

Importamos los dos paquetes donde están definidas las clases e interfaces para la captura de eventos:

```
import javax.swing.*;  
import javax.swing.event.*;
```

Heredamos de la clase JFrame e implementamos la interface ChangeListener para capturar el cambio de selección de objeto de tipo JRadioButton:

```
public class Formulario extends JFrame implements ChangeListener{
```

Definimos tres objetos de la clase JRadioButton y uno de tipo ButtonGroup:

```
    private JRadioButton radio1,radio2,radio3;  
    private ButtonGroup bg;
```

En el constructor creamos primero el objeto de la clase ButtonGroup:

```
    bg = new ButtonGroup();
```

Creamos seguidamente el objeto de la clase JRadioButton, definimos su ubicación, llamamos al método addChangeListener para informar que objeto capturará el evento y finalmente añadimos el objeto JRadioButton al JFrame y al ButtonGroup:

```
radio1=new JRadioButton("640*480");  
radio1.setBounds(10,20,100,30);  
radio1.addChangeListener(this);  
add(radio1);  
bg.add(radio1);
```

Exactamente hacemos lo mismo con los otros dos JRadioButton:

```
radio2=new JRadioButton("800*600");  
radio2.setBounds(10,70,100,30);  
radio2.addChangeListener(this);  
add(radio2);  
bg.add(radio2);  
radio3=new JRadioButton("1024*768");  
radio3.setBounds(10,120,100,30);  
radio3.addChangeListener(this);  
add(radio3);  
bg.add(radio3);
```



En el método `stateChanged` verificamos cual de los tres `JRadioButton` está seleccionado y procedemos a redimensionar el `JFrame`:

```
public void stateChanged(ChangeEvent e) {  
    if (radio1.isSelected()) {  
        setSize(640,480);  
    }  
    if (radio2.isSelected()) {  
        setSize(800,600);  
    }  
    if (radio3.isSelected()) {  
        setSize(1024,768);  
    }  
}
```

## **Problema propuesto**

Introducir dos números en controles de tipo JTextField y mediante dos controles de tipo JRadioButton permitir seleccionar si queremos sumarlos o restarlos. Al presionar un botón mostrar en el título del JFrame el resultado de la operación.

## 55 - Plug-in WindowBuilder para crear interfaces visuales.

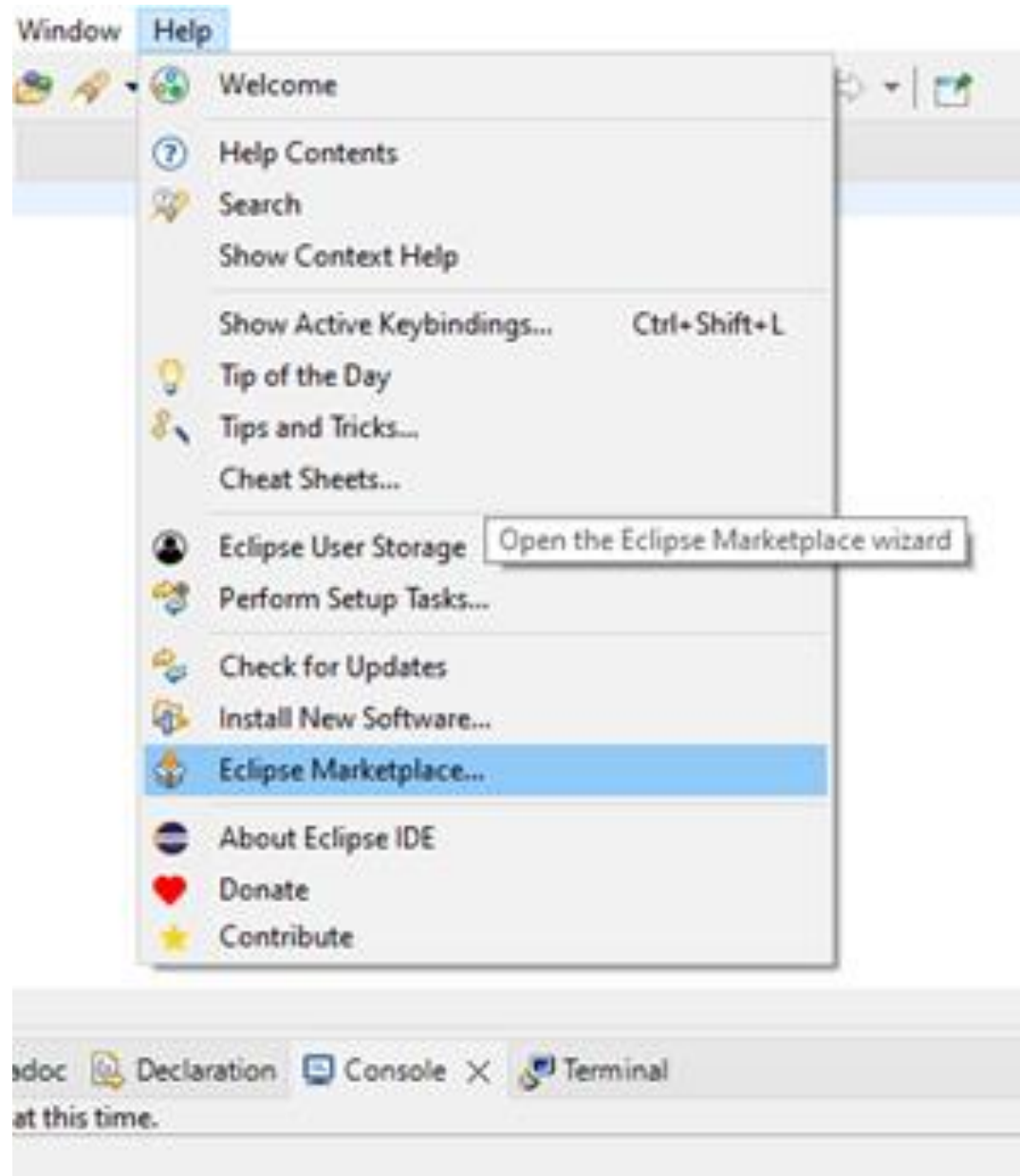
El objetivo de este concepto es conocer el empleo del plug-in WindowBuilder para el desarrollo de interfaces visuales arrastrando componentes.

Desde la versión 3.7 de Eclipse llamada Indigo(2011) hasta la versión Mars (2015) se incorpora por defecto el plug-in WindowBuilder para la implementación de interfaces visuales.

Las versiones Neon, Oxygen y la actual no trae instalado el plug-in WindowBuilder por defecto.

Instalación del plug-in WindowBuilder en las versiones nuevas.

Debemos seleccionar desde el menú de opciones de Eclipse Help->Eclipse Marketplace...:

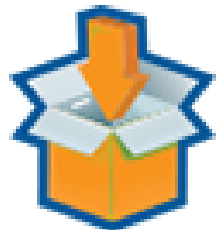


Nos aparece la tienda de complementos del entorno de Eclipse. Debemos buscar el Plug-in Windowbuilder:

 Eclipse Marketplace

## Eclipse Marketplace

Select solutions to install. Press Install Now to proceed with installation.  
Press the "more info" link to learn more about a solution.



Search

Recent

Popular

Favorites

Installed



Giving IoT an Edge

Find:



indow builder



All Markets



All Categories



Go

### WindowBuilder 1.9.8



WindowBuilder is composed of SWT Designer and Swing Designer and makes it very easy to create Java GUI applications without spending a lot of time writing code.... [more info](#)

by [Eclipse Foundation](#), EPL

[SWT swing wysiwyg graphical WindowBuilder](#)



804



Installs: **739K** (18.964 last month)

**Install**

Presionamos "Install" y aceptamos los términos y condiciones en los siguientes diálogos.

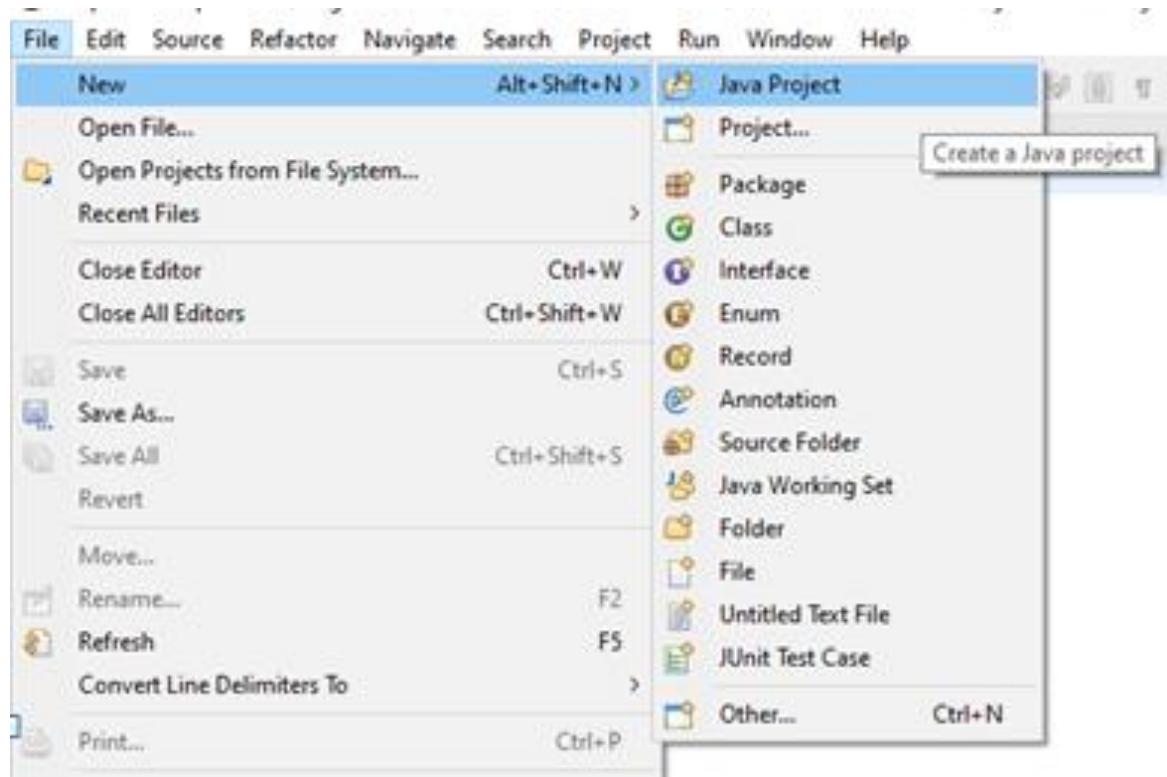
Con esto ya tenemos instalado el WindowBuilder para trabajar en los siguientes conceptos. Se debe reiniciar el entorno de Eclipse para que los cambios tengan efecto.

# Uso del WindowBuilder

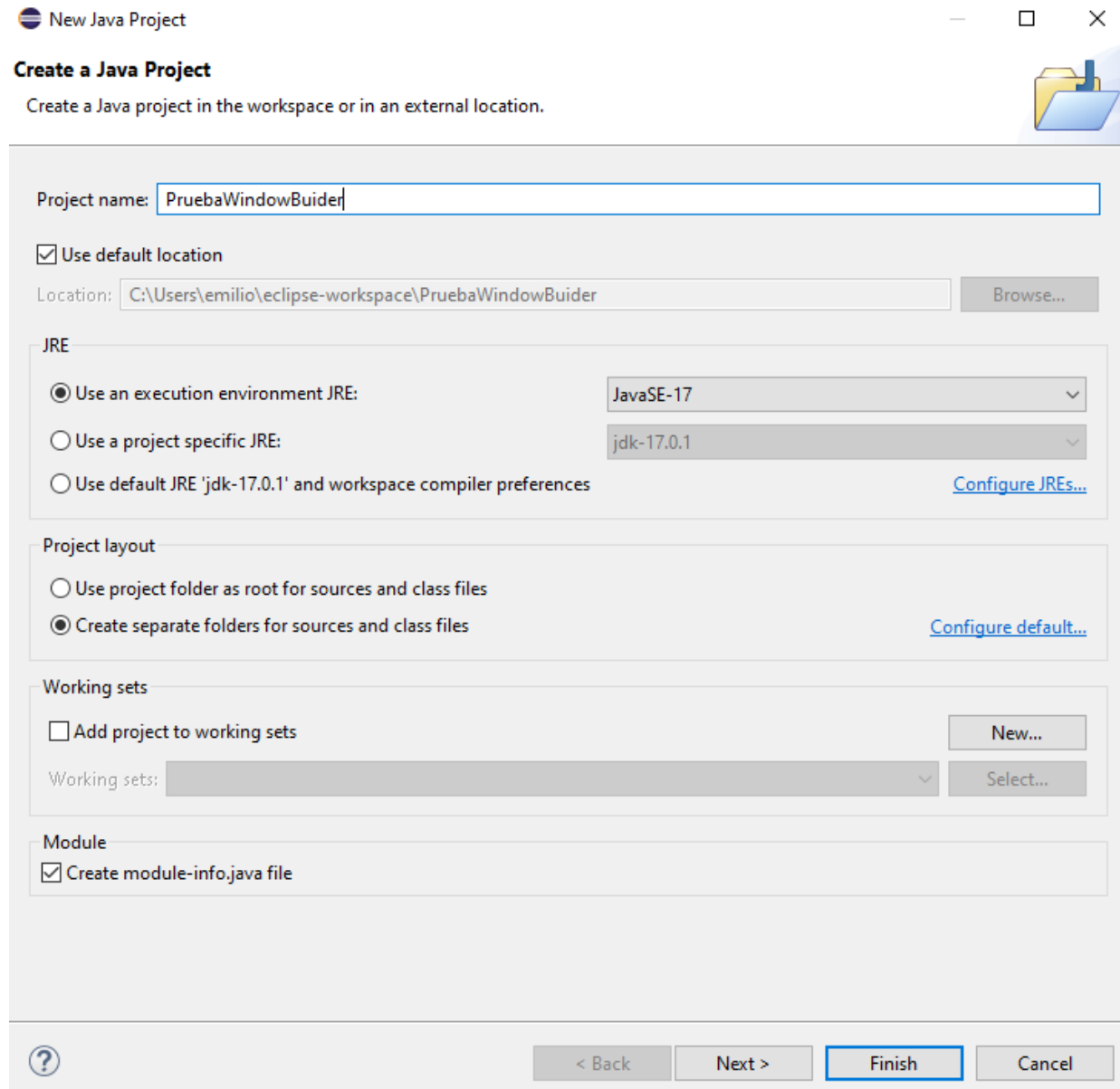
A medida que uno arrastra componentes visuales sobre un formulario se genera en forma automática el código Java, esto nos permite ser más productivos en el desarrollo de la interfaz de nuestra aplicación y nos ayuda a concentrarnos en la lógica de nuestro problema.

## Pasos para crear un JFrame con el WindowBuilder

1 - Creación de un proyecto.



2 - Seleccionamos el nombre de nuestro proyecto (lo llamaremos PruebaWindowBuilder):



The screenshot shows the 'New Java Project' dialog box in the Eclipse IDE. The title bar reads 'New Java Project'. Below the title bar, the text 'Create a Java Project' is followed by the instruction 'Create a Java project in the workspace or in an external location.' and a folder icon. The 'Project name' field contains 'PruebaWindowBuilder'. The 'Use default location' checkbox is checked, and the 'Location' field shows 'C:\Users\emilio\eclipse-workspace\PruebaWindowBuilder' with a 'Browse...' button. The 'JRE' section has three radio buttons: 'Use an execution environment JRE:' (selected), 'Use a project specific JRE:', and 'Use default JRE 'jdk-17.0.1' and workspace compiler preferences'. The first option has a dropdown menu showing 'JavaSE-17' and 'jdk-17.0.1'. A 'Configure JREs...' link is present. The 'Project layout' section has two radio buttons: 'Use project folder as root for sources and class files' and 'Create separate folders for sources and class files' (selected). A 'Configure default...' link is also there. The 'Working sets' section has an 'Add project to working sets' checkbox and a 'Working sets:' dropdown menu with 'New...' and 'Select...' buttons. The 'Module' section has a checked checkbox for 'Create module-info.java file'. At the bottom, there is a help icon, '< Back' button, 'Next >' button, 'Finish' button (highlighted with a red border), and 'Cancel' button.

**New Java Project**

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name:

☒ Use default location

Location:

**JRE**

☒ Use an execution environment JRE:

☐ Use a project specific JRE:

☐ Use default JRE 'jdk-17.0.1' and workspace compiler preferences [Configure JREs...](#)

**Project layout**

☐ Use project folder as root for sources and class files

☒ Create separate folders for sources and class files [Configure default...](#)

**Working sets**

☐ Add project to working sets

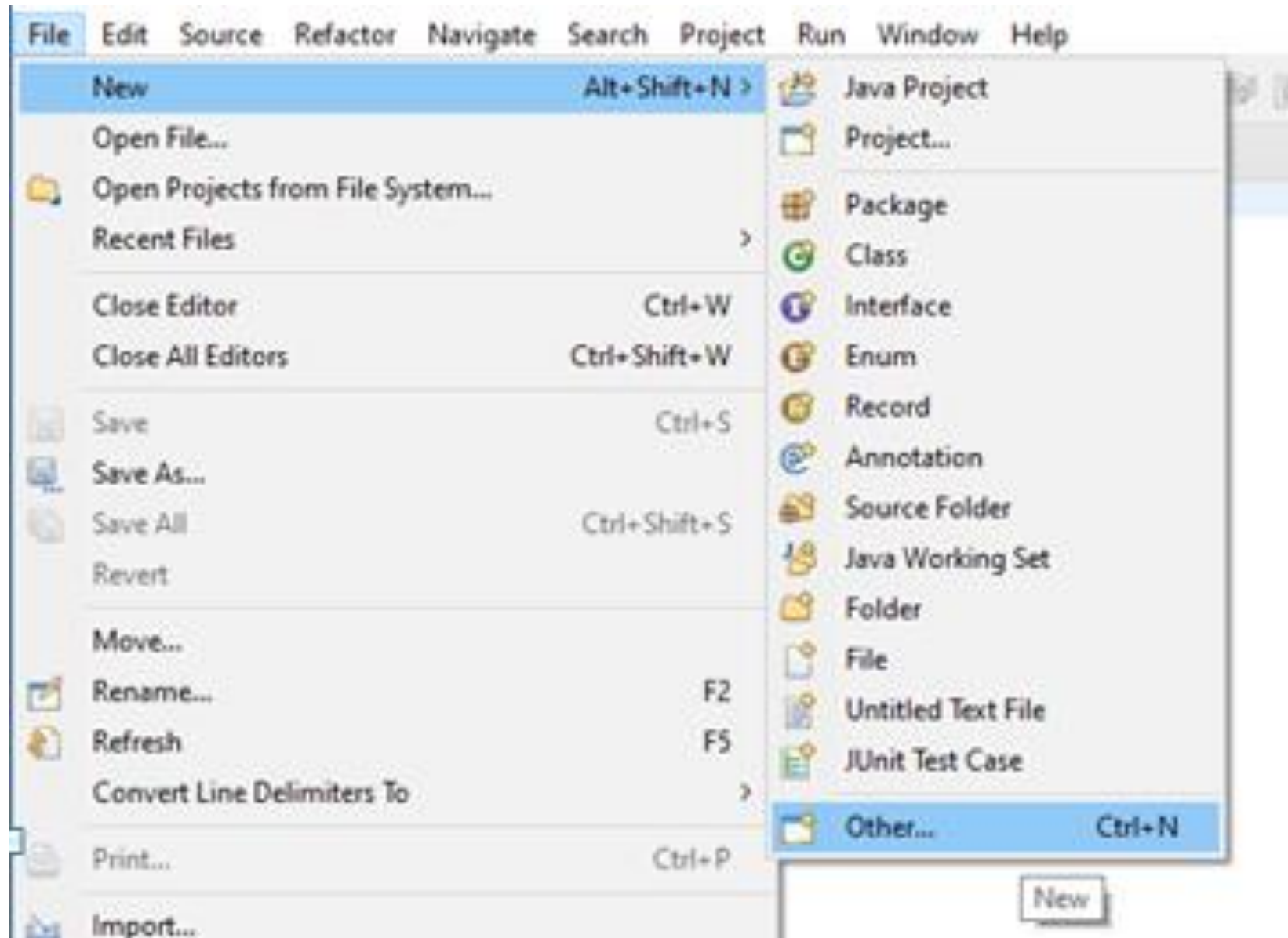
Working sets:

**Module**

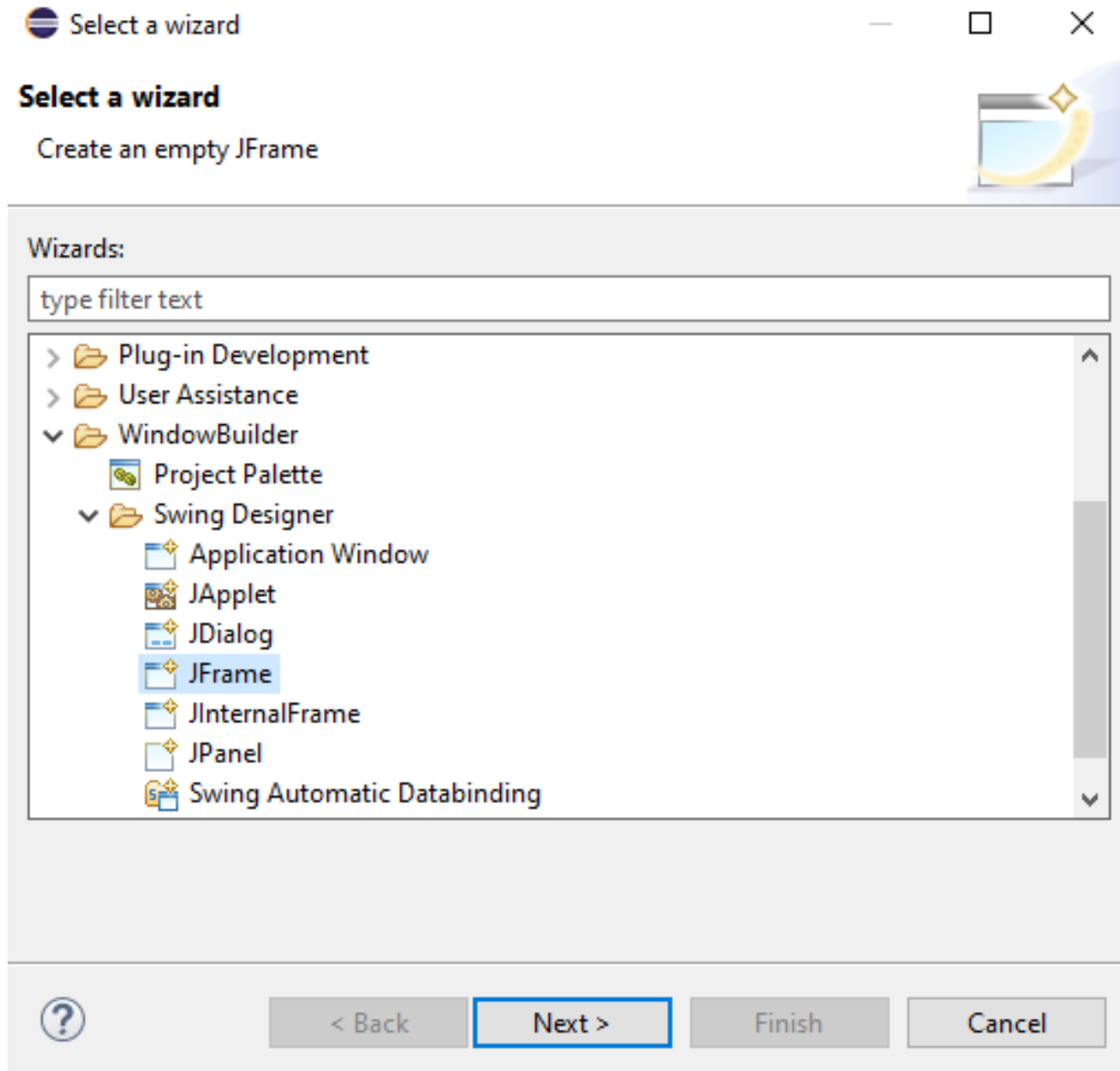
☒ Create module-info.java file



3 - Ahora seleccionamos la opción del menú File -> New -> Other ...



## 4 - Seleccionamos la opción la opción JFrame:



5 - Seguidamente presionamos el botón Next > y definimos el nombre de la clase a crear (Ventana1):

Source folder: PruebaWindowBuilder/src Browse...

Package: (default) Browse...

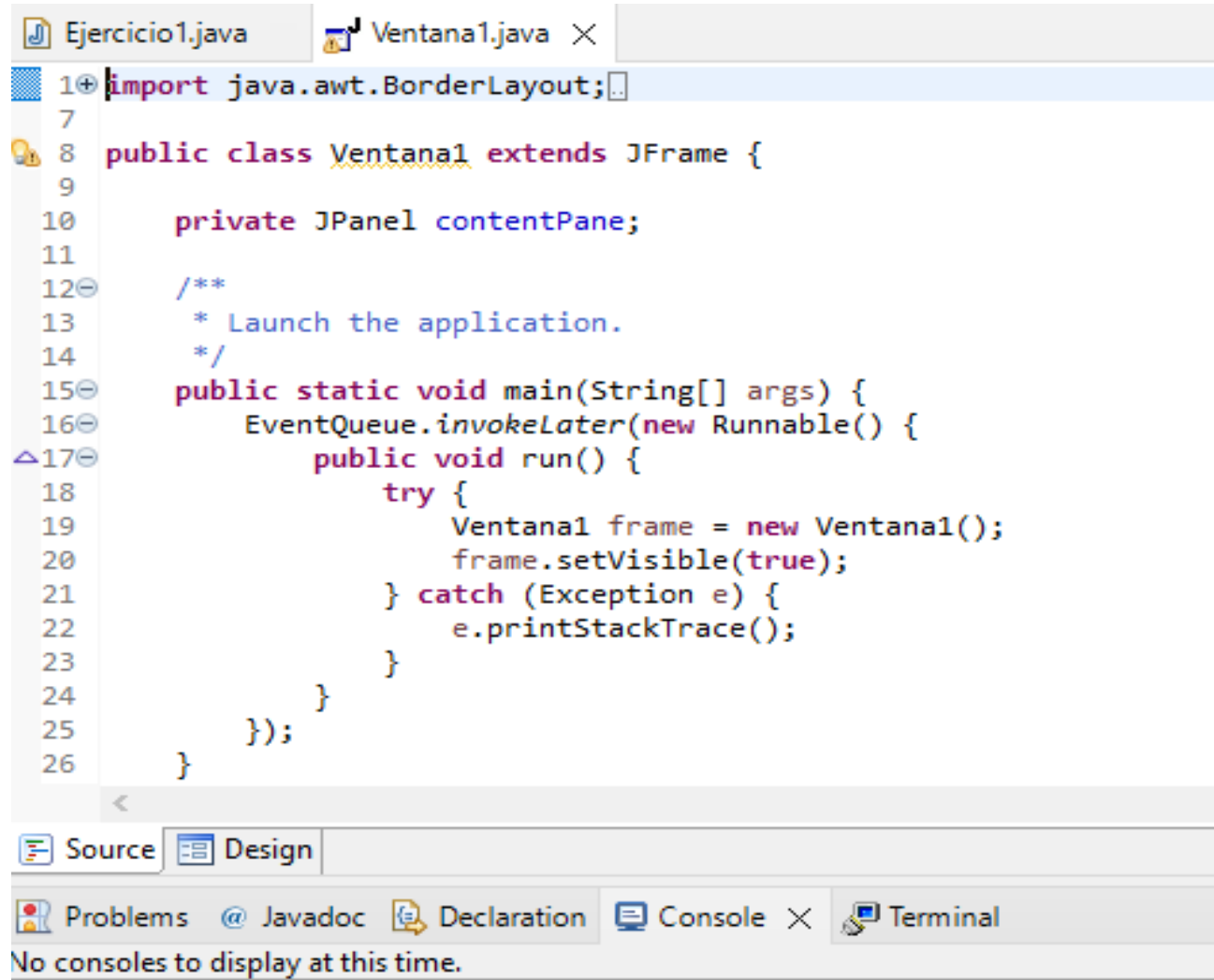
Name: Ventana1

Superclass: javax.swing.JFrame Browse...

☒ Use advanced template for generate JFrame

? < Back Next > Finish Cancel

Tenemos en este momento nuestra aplicación mínima generada por el WindowBuilder. Podemos observar que en la parte inferior de la ventana central aparecen dos pestañas (Source y Design) estas dos pestañas nos permiten ver el código fuente de nuestro JFrame en vista de diseño o en vista de código Java:



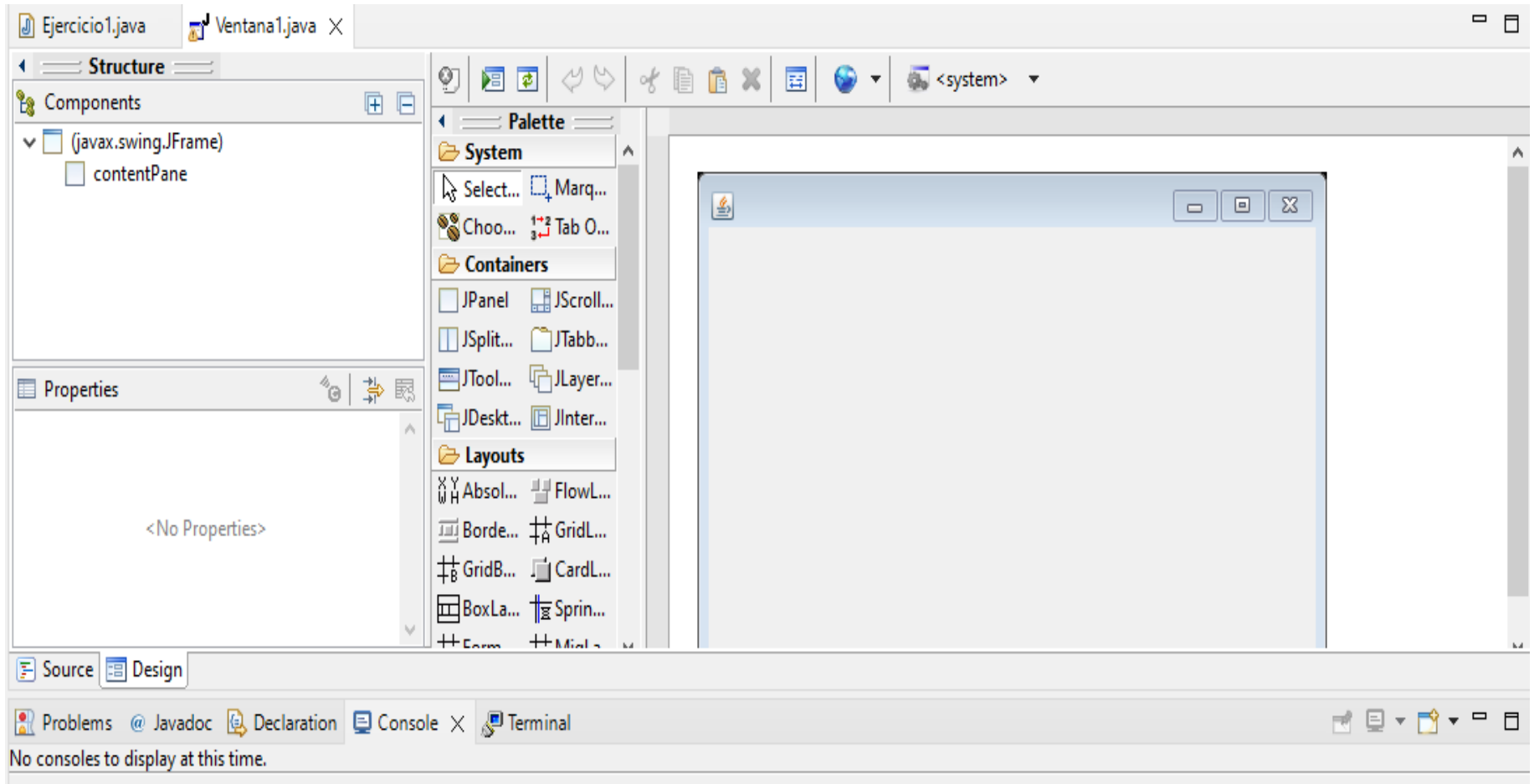
```
1+ import java.awt.BorderLayout;
7
8 public class Ventana1 extends JFrame {
9
10     private JPanel contentPane;
11
12     /**
13      * Launch the application.
14      */
15     public static void main(String[] args) {
16         EventQueue.invokeLater(new Runnable() {
17             public void run() {
18                 try {
19                     Ventana1 frame = new Ventana1();
20                     frame.setVisible(true);
21                 } catch (Exception e) {
22                     e.printStackTrace();
23                 }
24             }
25         });
26     }
```

Source Design

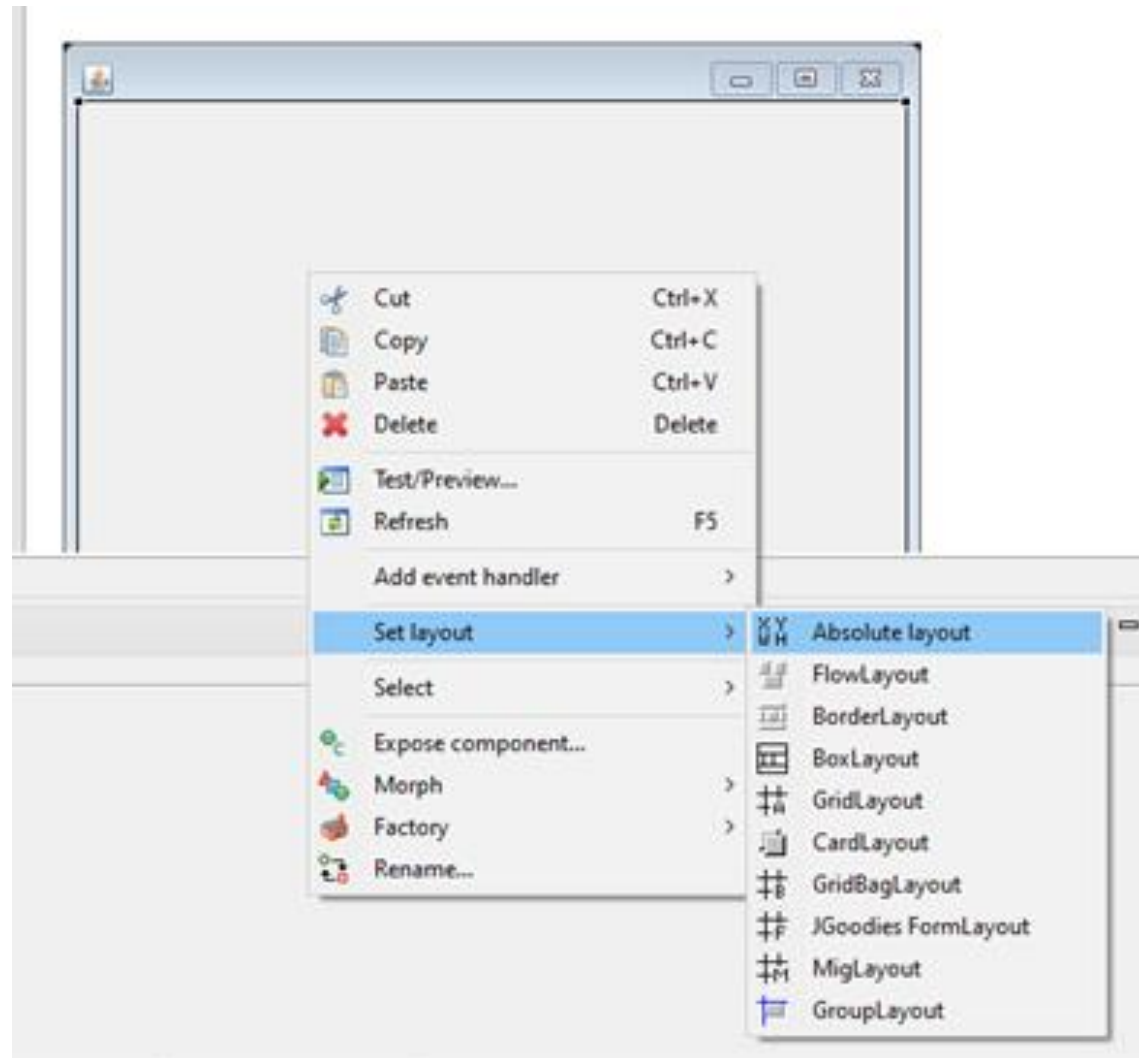
Problems Javadoc Declaration Console × Terminal

No consoles to display at this time.

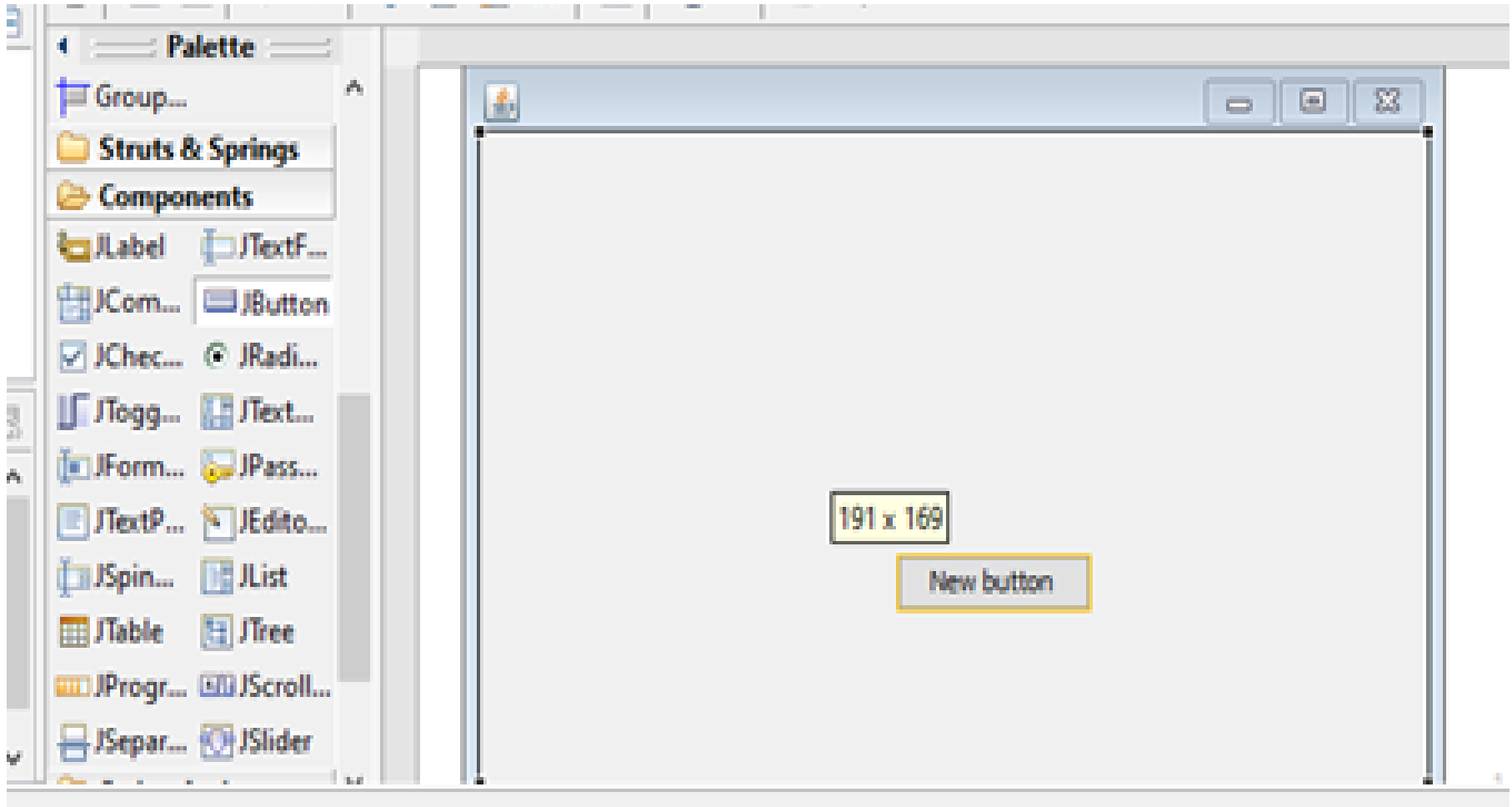
## vista de "Design":

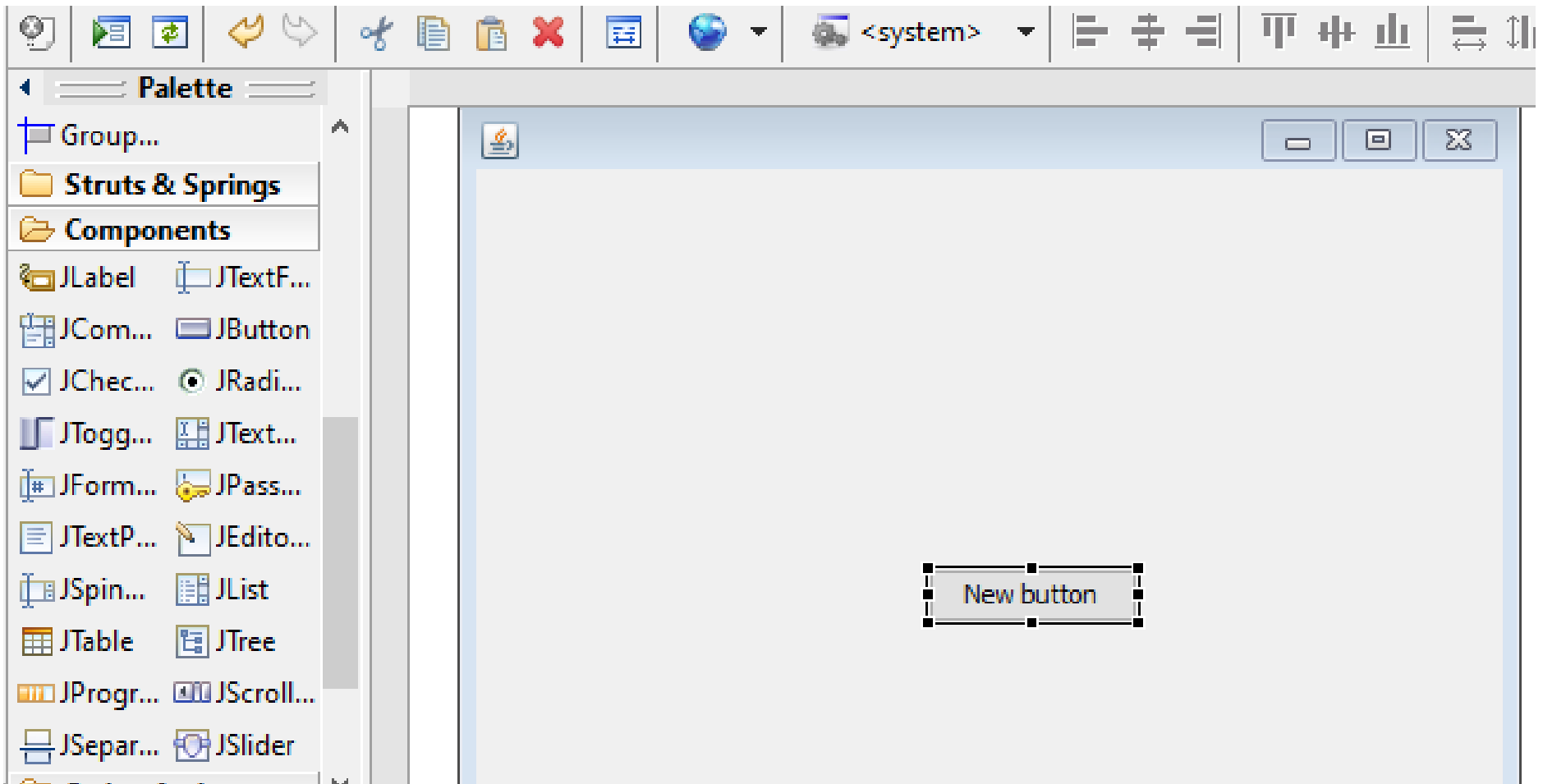


6 - Configuramos el Layout de JFrame presionando el botón derecho del mouse sobre el formulario generado y seleccionamos la opción SetLayout > Absolute layout (esto nos permite luego disponer controles visuales como JButton, JLabel etc. en posiciones fijas dentro del JFrame):



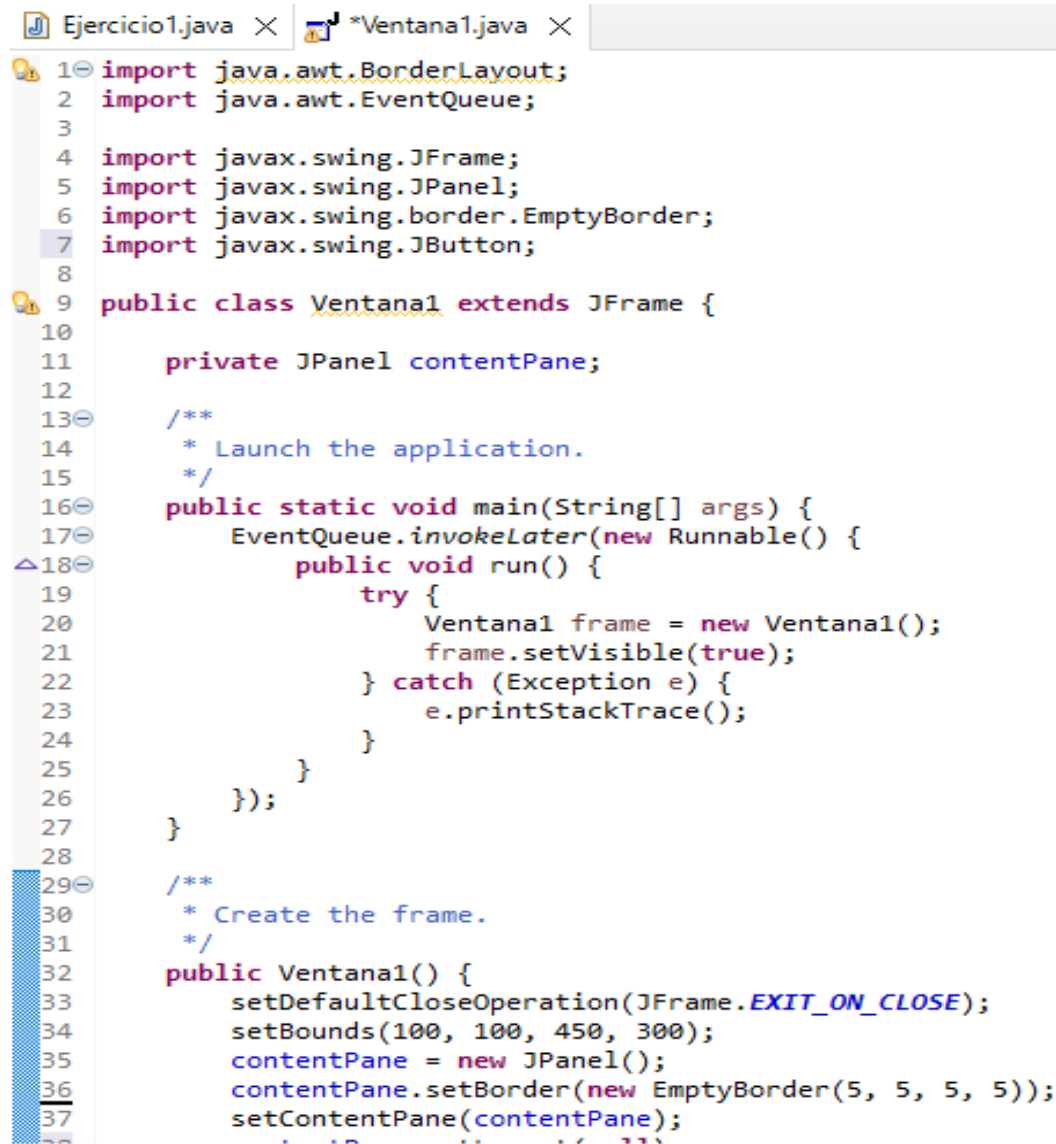
7 - De la ventana Palette seleccionamos con el mouse un objeto de la clase JButton (presionamos con el mouse dicha componente, deberá aparecer seleccionada) y luego nos desplazamos con el mouse sobre el JFrame y presionamos el botón del mouse nuevamente ( en este momento aparece el botón dentro del JFrame):







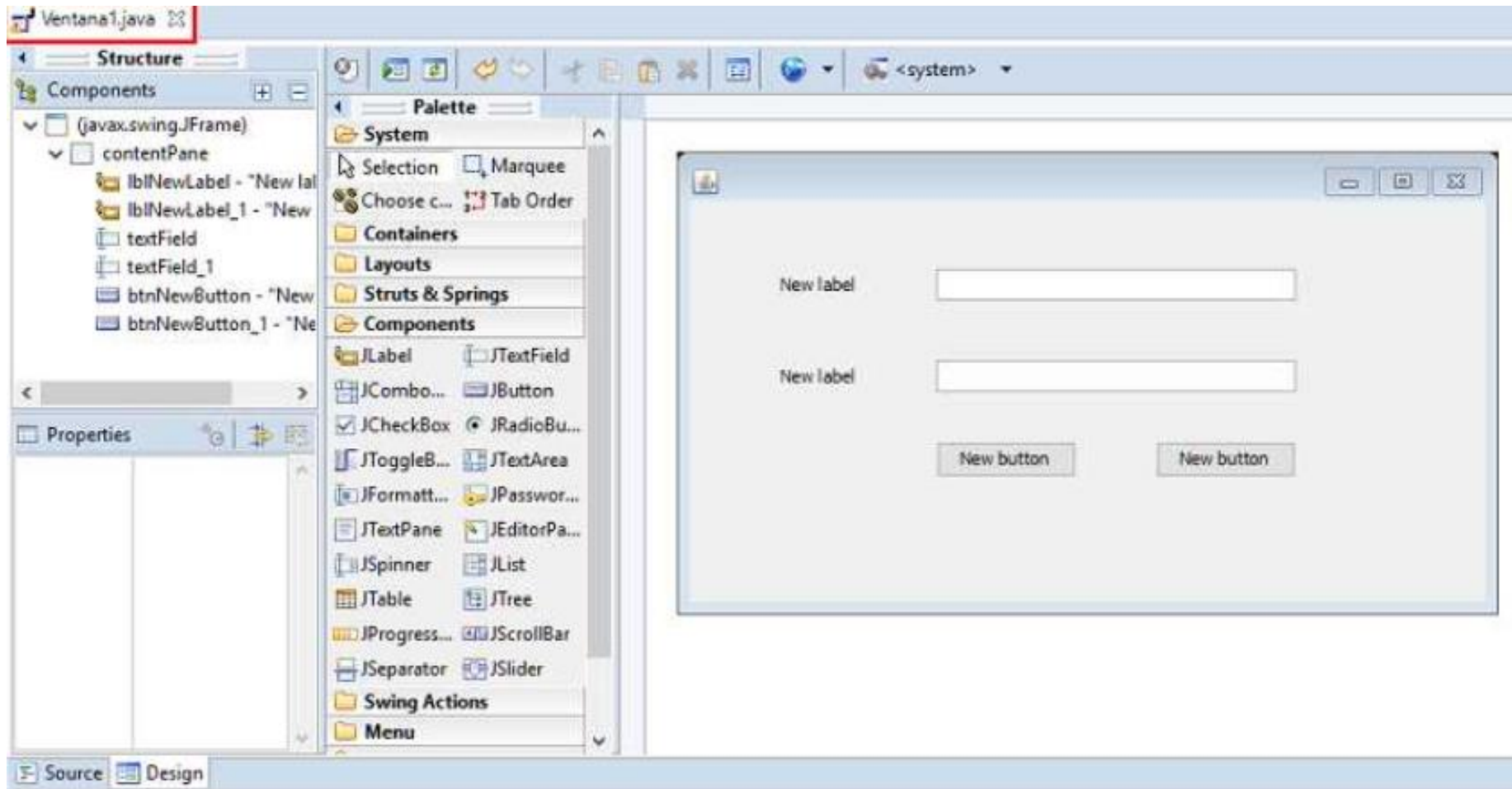
En todo momento podemos cambiar la pestaña de "Source" y "Design" para ver el código generado. Por ejemplo cuando agregamos el botón podemos ver que se agregó un objeto de la clase JButton al constructor:



```
Ejercicio1.java x *Ventana1.java x
1 import java.awt.BorderLayout;
2 import java.awt.EventQueue;
3
4 import javax.swing.JFrame;
5 import javax.swing.JPanel;
6 import javax.swing.border.EmptyBorder;
7 import javax.swing.JButton;
8
9 public class Ventana1 extends JFrame {
10
11     private JPanel contentPane;
12
13     /**
14      * Launch the application.
15      */
16     public static void main(String[] args) {
17         EventQueue.invokeLater(new Runnable() {
18             public void run() {
19                 try {
20                     Ventana1 frame = new Ventana1();
21                     frame.setVisible(true);
22                 } catch (Exception e) {
23                     e.printStackTrace();
24                 }
25             }
26         });
27     }
28
29     /**
30      * Create the frame.
31      */
32     public Ventana1() {
33         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
34         setBounds(100, 100, 450, 300);
35         contentPane = new JPanel();
36         contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
37         setContentPane(contentPane);
38     }
39 }
```

## Inicializar propiedades de los objetos.

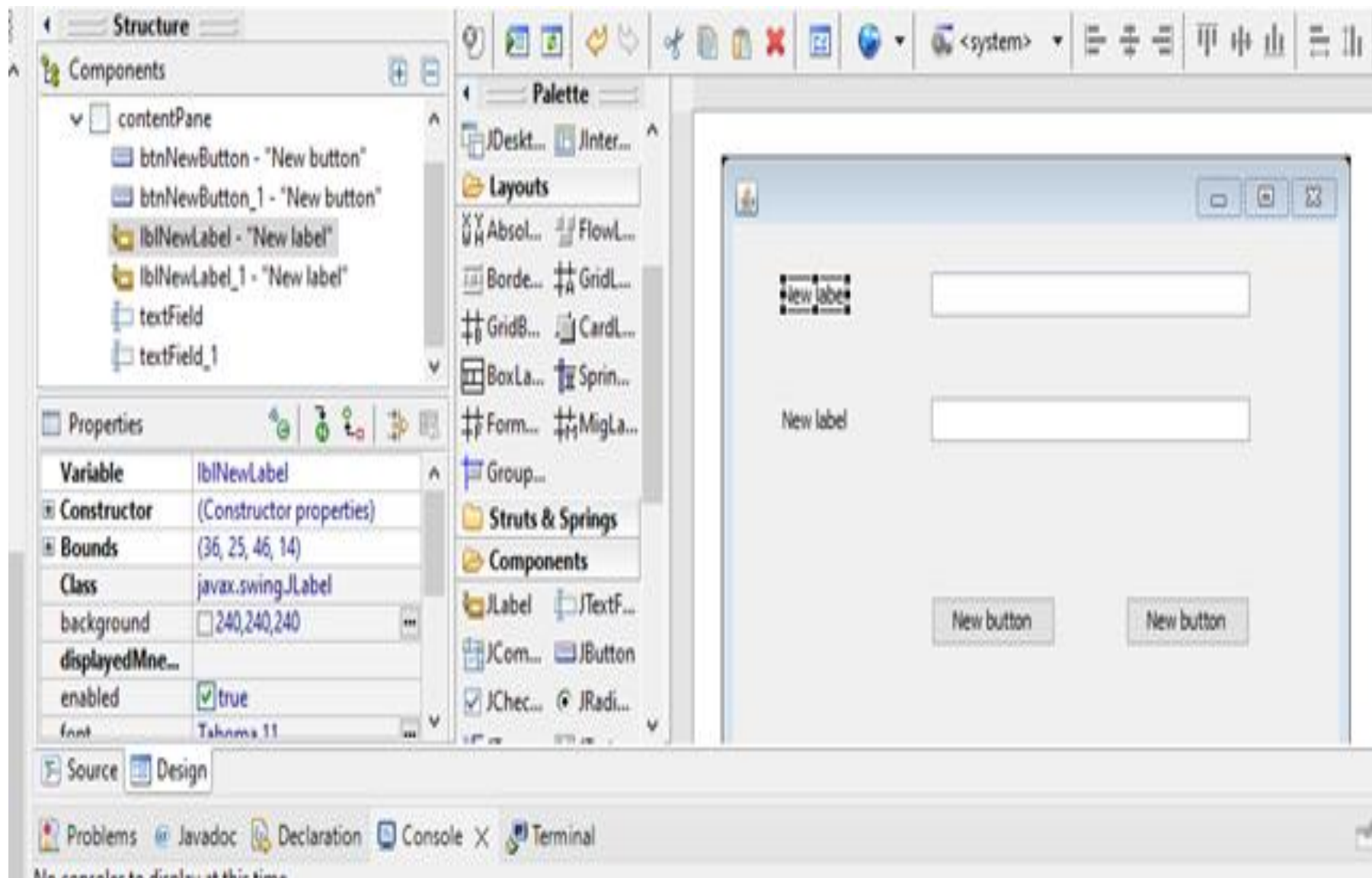
Crear un proyecto y luego un JFrame con las siguientes componentes visuales :  
Dos controles de tipo JLabel, dos JTextField y dos JButton (previamente definir el layout para el panel contenido en el JFrame de tipo Absolute Layout)



Si hacemos doble clic con el mouse en la pestaña Ventana1.java podemos maximizar el espacio de nuestro editor visual (haciendo nuevamente doble clic vuelve al tamaño anterior)

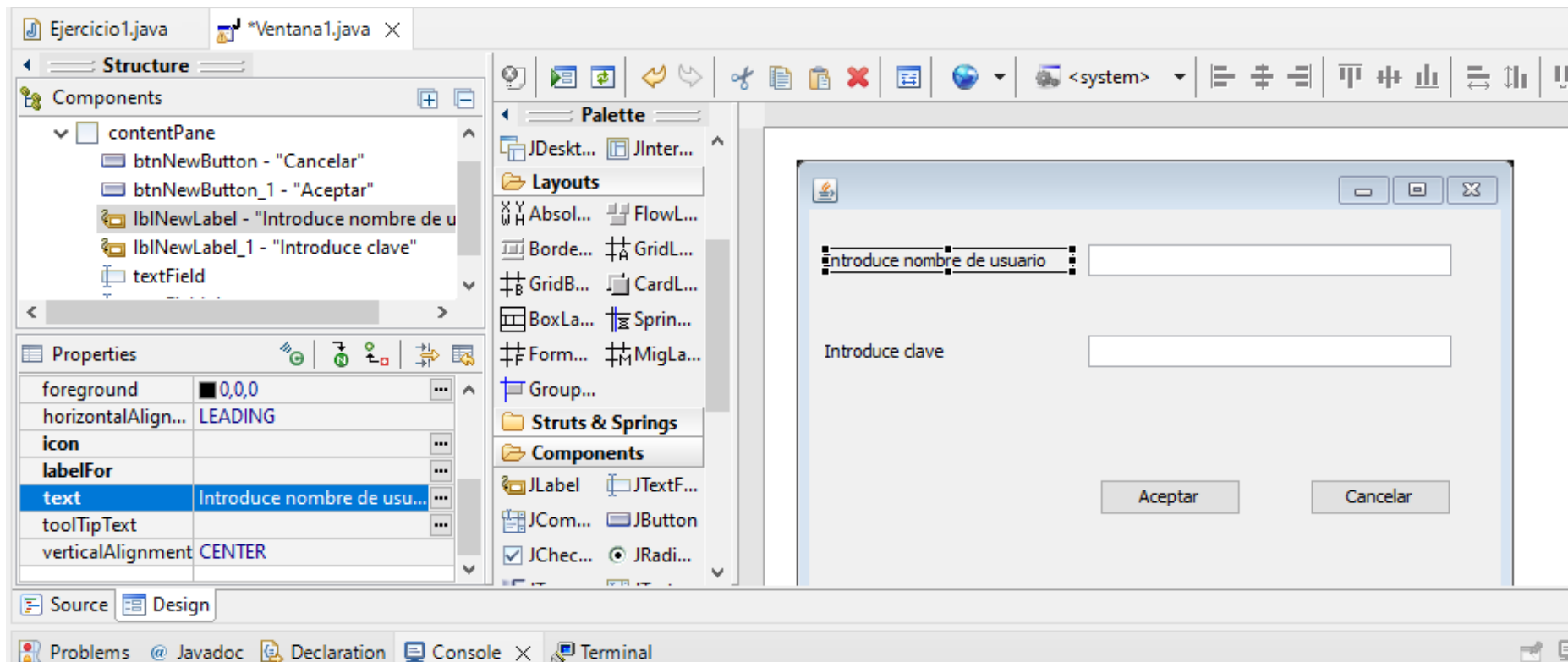
Seleccionemos el primer JLabel de nuestro formulario y observemos las distintas partes que componen el plug-in WindowBuilder. En la parte superior izquierda se encuentra la sección "Components" donde se muestran las componentes visuales agregadas al formulario. Aparece resaltada la que se encuentra actualmente seleccionada.

En la parte inferior aparece la ventana de "Properties" o propiedades del control visual seleccionado.



Veamos algunas propiedades que podemos modificar desde esta ventana y los cambios que se producen en el código fuente en java.

La propiedad text cambia el texto que muestra el objeto JLabel. Probemos disponer el texto "Introduce nombre de usuario:". De forma similar hagamos los cambios en la propiedad text de los otros controles visuales de nuestro JFrame:



Si ahora seleccionamos la pestaña inferior para ver la vista de código java: "Source" podemos ver que el WindowBuilder nos generó automáticamente el código para inicializar los textos de los controles JLabel y JButton:

```
    */
    public Ventana1() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 450, 300);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JButton btnNewButton = new JButton("Cancelar");
        btnNewButton.setBounds(314, 169, 89, 23);
        contentPane.add(btnNewButton);

        JButton btnNewButton_1 = new JButton("Aceptar");
        btnNewButton_1.setBounds(182, 169, 89, 23);
        contentPane.add(btnNewButton_1);

        JLabel lblNewLabel = new JLabel("Introduce nombre de usuario");
        lblNewLabel.setBounds(10, 25, 155, 14);
        contentPane.add(lblNewLabel);

        JLabel lblNewLabel_1 = new JLabel("Introduce clave");
        lblNewLabel_1.setBounds(10, 82, 122, 14);
        contentPane.add(lblNewLabel_1);

        textField = new JTextField();
        textField.setBounds(175, 22, 228, 20);
        contentPane.add(textField);
        textField.setColumns(10);

        textField_1 = new JTextField();
        textField_1.setBounds(175, 79, 228, 20);
        contentPane.add(textField_1);
        textField_1.setColumns(10);
    }
}
```

Como podemos observar ahora cuando se crean los objetos de la clase JLabel en el constructor se inicializan con los valores cargados en la propiedad text:

```
JLabel lblNewLabel = new JLabel("Introduce nombre de usuario:");
```

```
.....
```

```
JLabel lblNewLabel_1 = new JLabel("Introduce clave:");
```

```
.....
```

```
JButton btnNewButton = new JButton("Aceptar");
```

```
.....
```

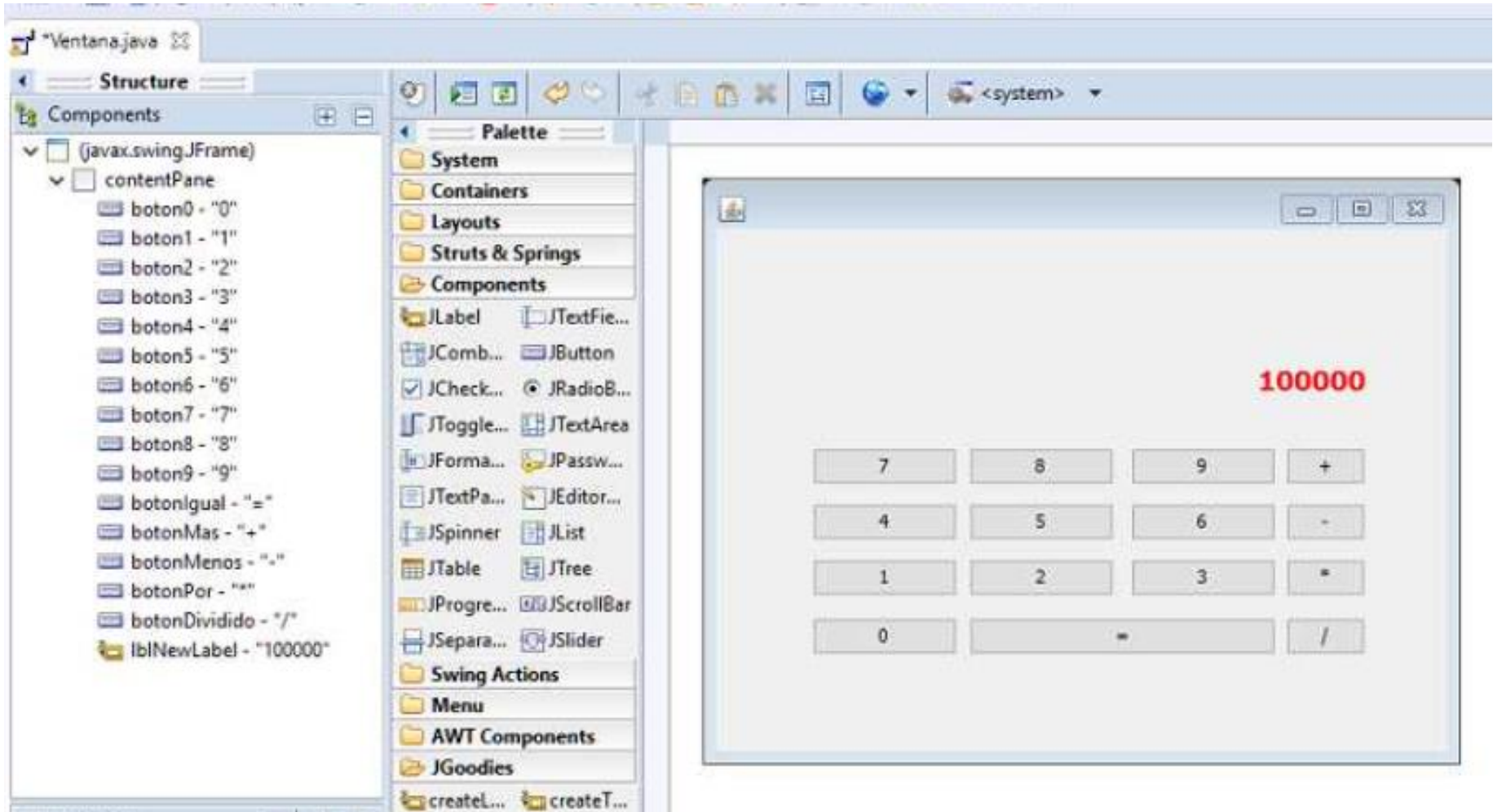
```
JButton btnNewButton_1 = new JButton("Cancelar");
```

```
.....
```



# Problema propuesto 1

Crear la interfaz visual que aparece abajo. Inicializar las propiedades 'text' y 'variable' para cada JButton y la JLabel.

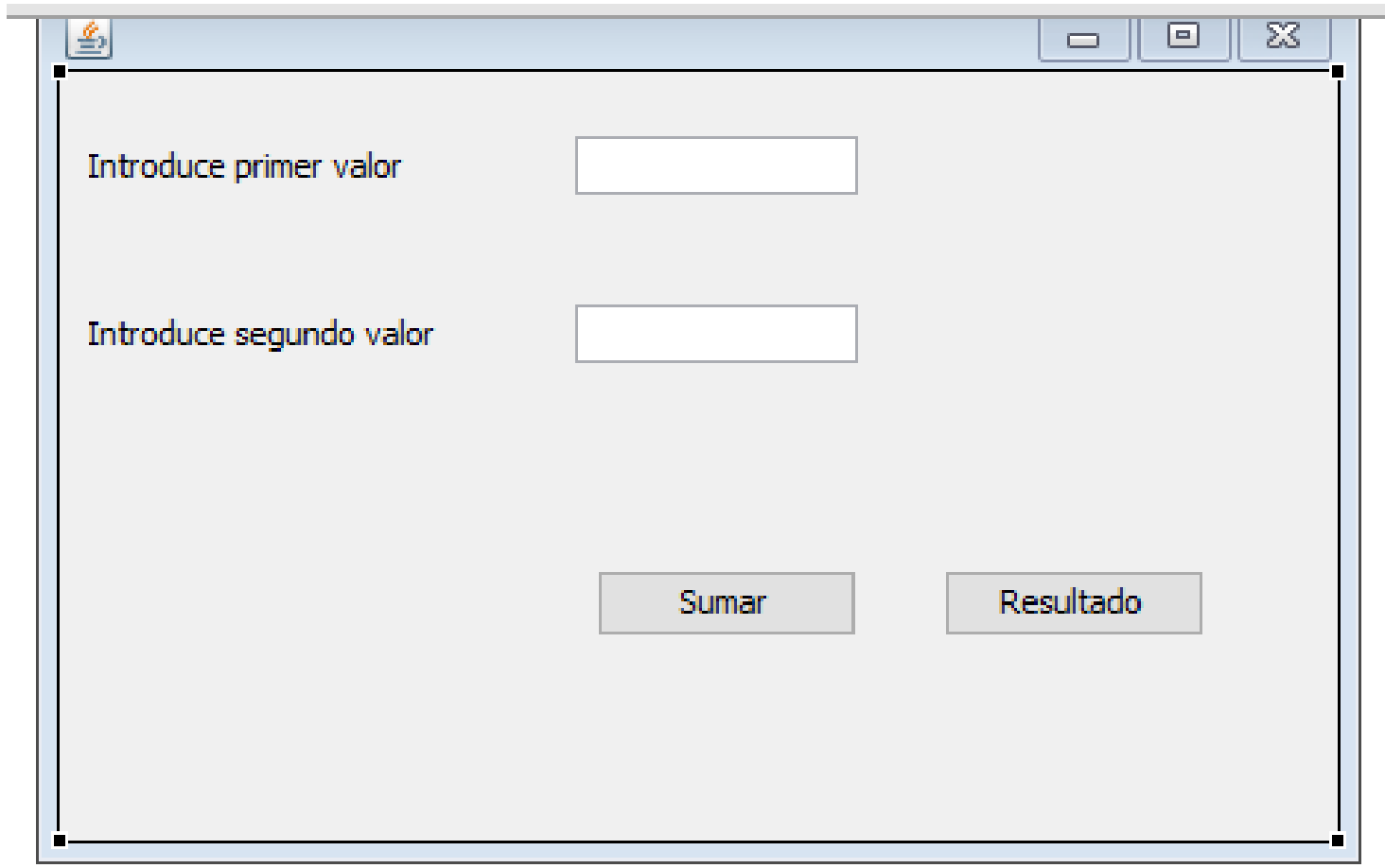




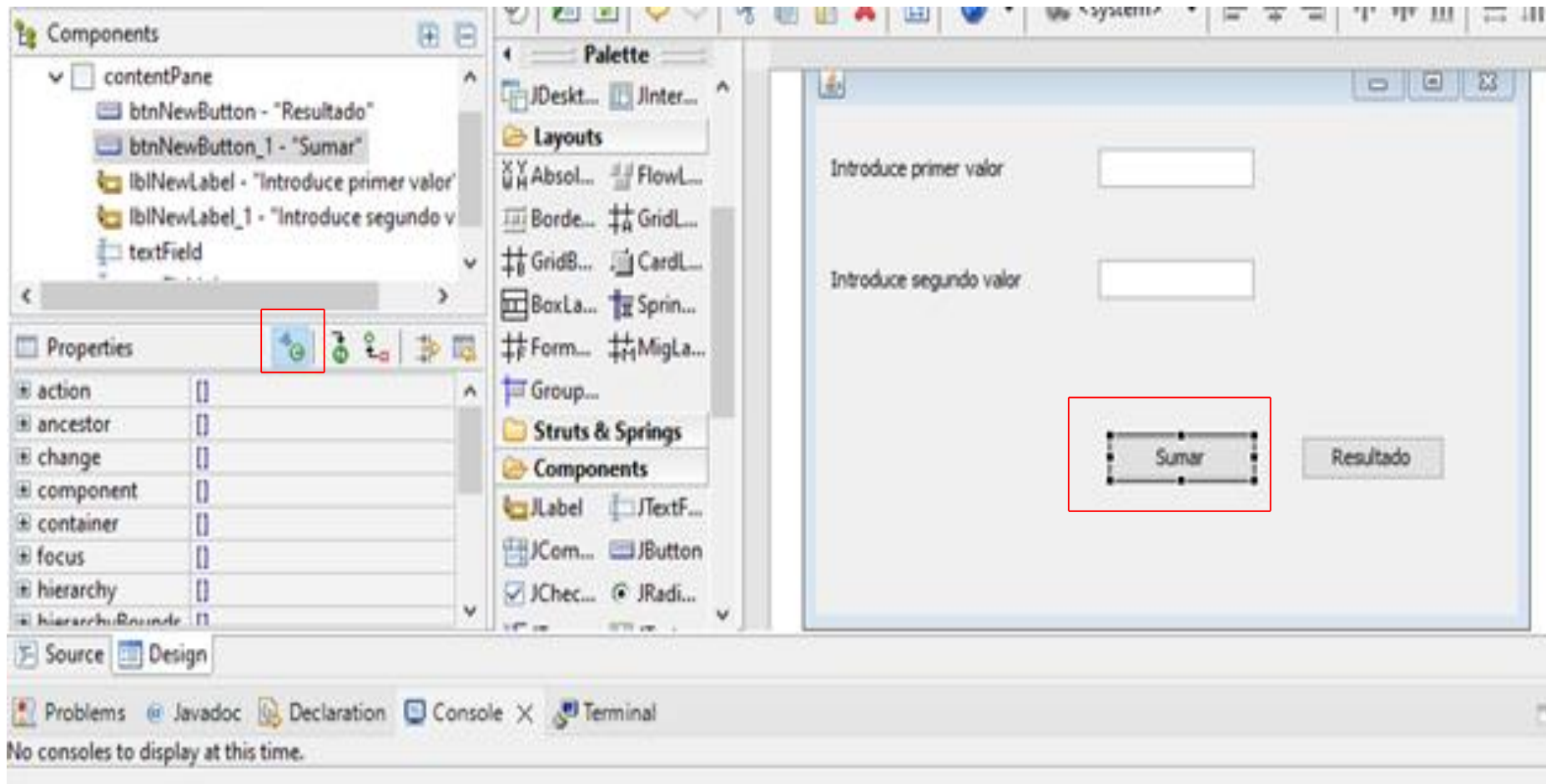
# Eventos

Para asociar eventos el plug-in WindowBuilder nos proporciona una mecánica para automatizar la generación de las interfaces que capturan los eventos de los objetos JButton, JMenuItem, JList etc.

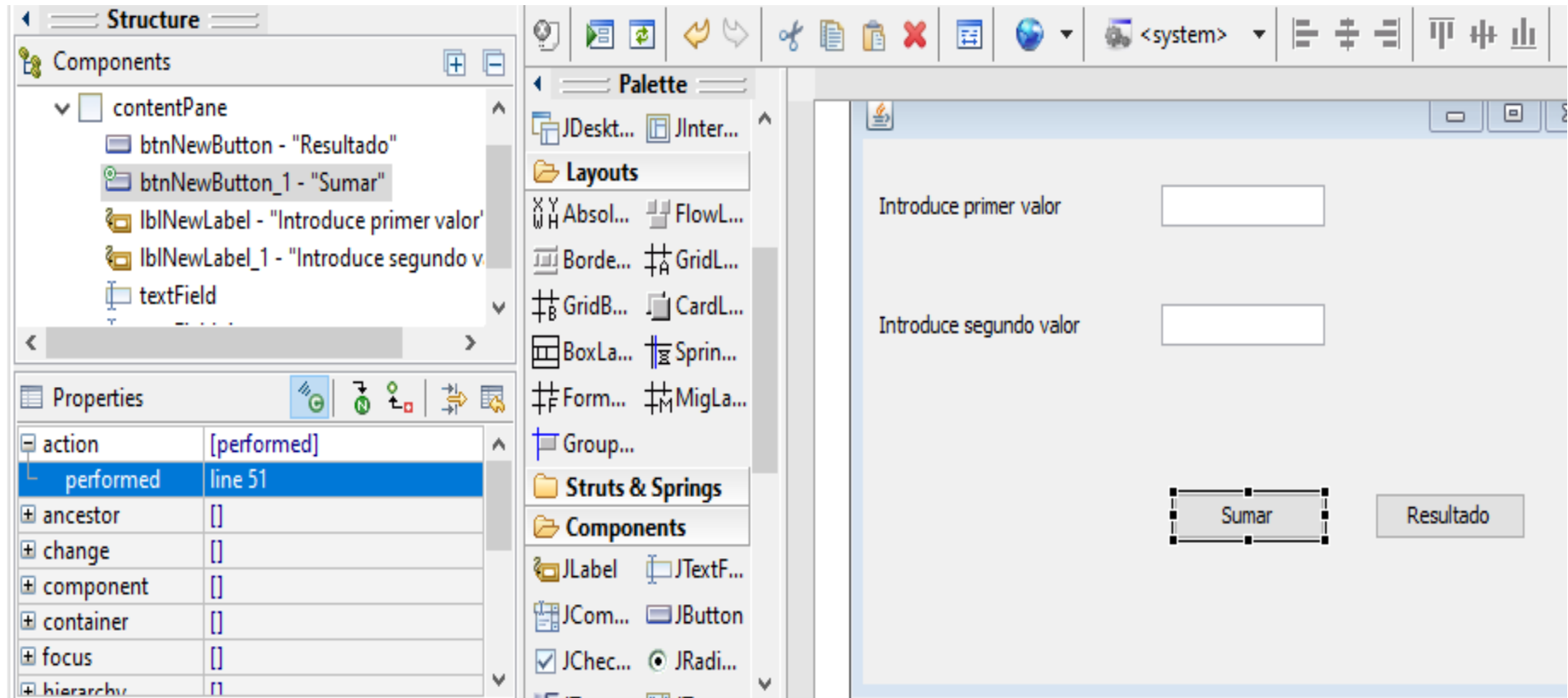
Crearemos una interfaz visual similar a esta (tres controles de tipo JLabel, dos JTextField y un JButton):



Ahora seleccionamos el control JButton y en la ventana de propiedades presionamos el icono de la parte superior:



Presionamos el + que aparece al lado de la palabra 'action' y hacemos doble clic sobre la palabra performed, vemos que se abre el editor de texto y aparece el siguiente código generado automáticamente:



```
contentPane.add(btnNewButton);
```

```
JButton btnNewButton_1 = new JButton("Sumar");
```

```
btnNewButton_1.addActionListener(new ActionListener() {
```

```
    public void actionPerformed(ActionEvent e) {
```

```
    }  
});
```

```
btnNewButton_1.setBounds(182, 169, 89, 23);
```

```
contentPane.add(btnNewButton_1);
```

```
Label lblNewLabel = new Label("Introduce primer valor");
```

En el parámetro del método `addActionListener` del botón que suma se le pasa la referencia a una interface que se crea de tipo `ActionListener` e implementa el método `actionPerformed` donde agregaremos el código necesario para responder el evento.

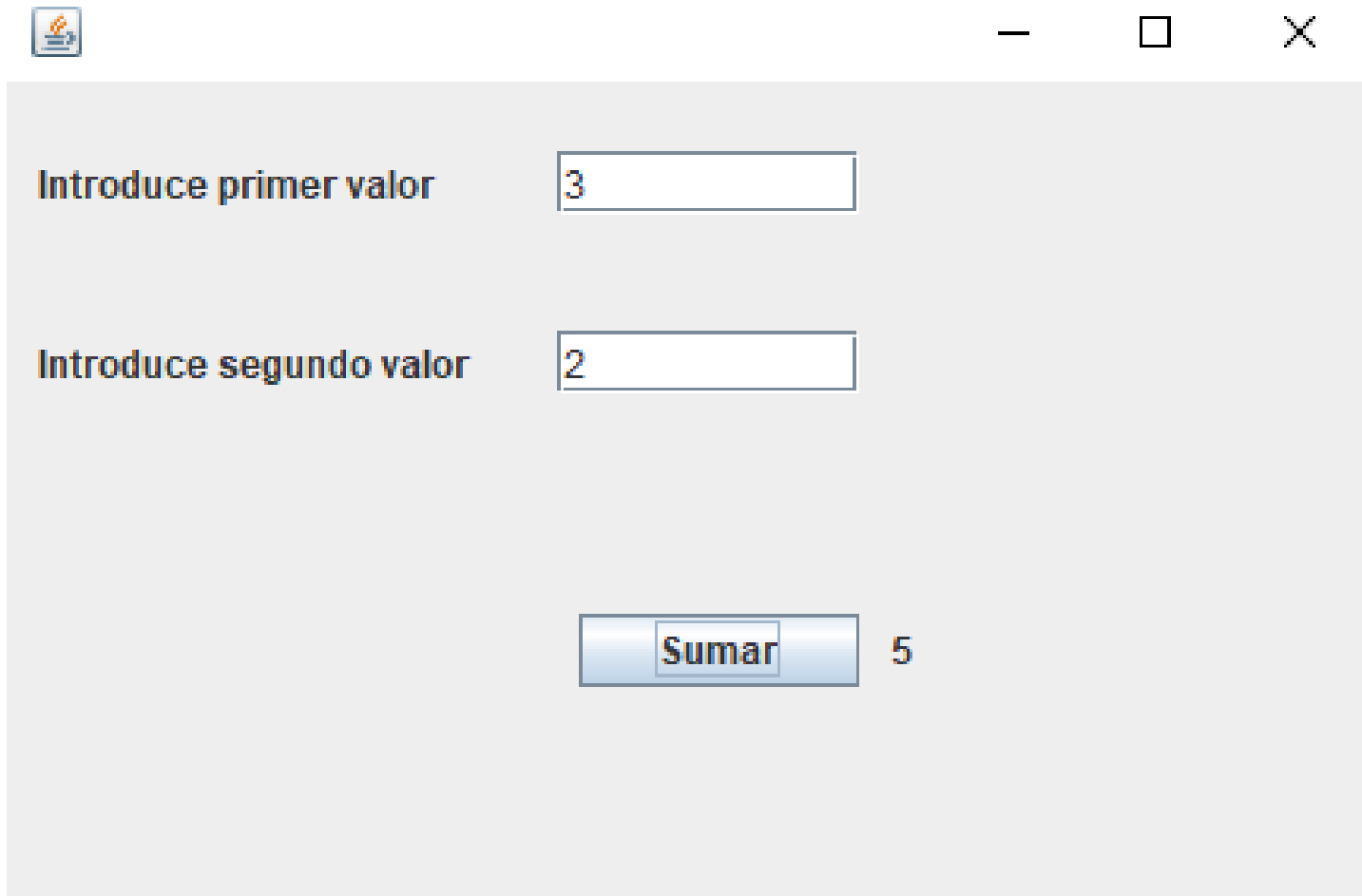
Para este problema debemos rescatar los valores almacenados en los controles de tipo `JTextField`, convertirlos a entero, sumarlos y enviar dicho resultado a una `JLabel`.

```
btnNewButton.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        int v1=Integer.parseInt(textField.getText());  
        int v2=Integer.parseInt(textField_1.getText());  
        int suma=v1+v2;  
        lblNewLabel_2.setText(String.valueOf(suma));  
    }  
});
```

Cuando compilamos vemos que no tenemos acceso al objeto lblNewLabel\_2 ya que está definido como una variable local al constructor. Si queremos que se definan como atributos de la clase debemos seleccionar la JLabel y presionar "convert Local to Field" (convertir de variable local a atributo de la clase):

Después de esto podemos acceder desde el método `actionPerformed` a la label.

Los nombres de objetos que automáticamente genera el WindowBuilder, por ejemplo `btnNewButton`, `lblNewLabel`, `lblNewLabel_1`, `lblNewLabel_2` etc. los podemos cambiar desde la ventana de properties a través de la propiedad 'variable'.

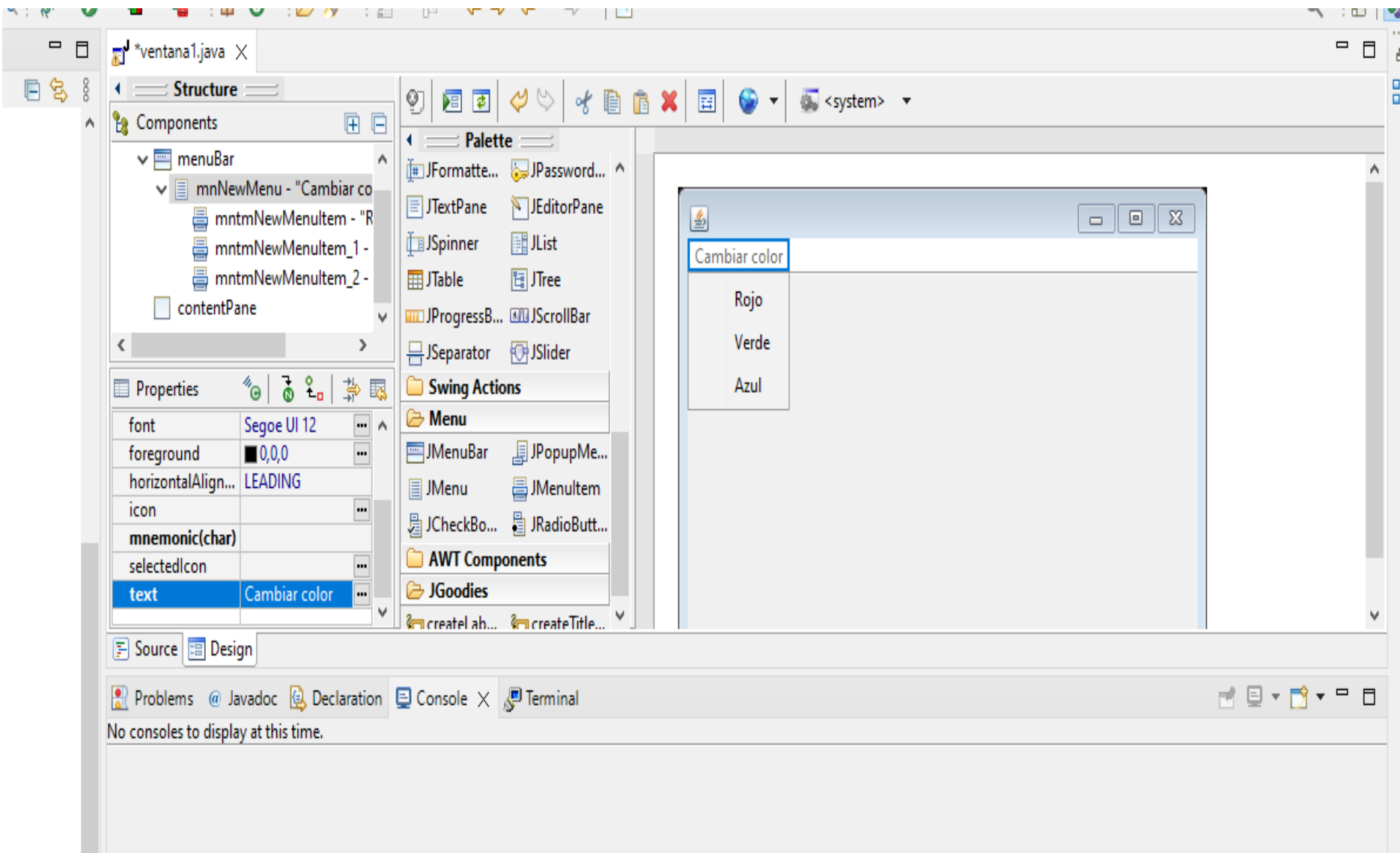


## Problema

Crear un menú de opciones que permita cambiar el color de fondo. Disponer un JMenuBar, un JMenu y 3 objetos de la clase JMenuItem. Asociar los eventos respectivos para cada control de tipo JMenuItem.

La interfaz visual debe quedar similar a la siguiente:





Para crear esta interface debemos primero seleccionar la pestaña "Menu" donde se encuentran las componentes relacionadas a la creación de menús. Debemos agregar (en este orden las siguientes componentes):

- 1 - Un JMenuBar en la parte superior.
- 2 - Un objeto de la clase JMenu en la barra del JMenuBar (podemos disponer el texto que queremos que se muestre)
- 3 - Agregamos un objeto de la clase JMenuItem en el sector donde aparece el texto: "Add items here". Los mismos pasos hacemos para agregar los otros dos JMenuItem.

Ahora debemos asociar el evento clic para cada JMenuItem. Seleccionamos primero el control de tipo JMenuItem y en la ventana de "Properties" presionamos el botón "Show events" y generamos el actionPerformed para el JMenuItem seleccionado.

Luego codificamos:

```
mntmRojo.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent arg0) {  
        contentPane.setBackground(Color.red);  
    }  
});
```

Para cambiar el color del JFrame en realidad debemos modificar el color del JPanel que cubre el JFrame. El objeto de la clase JPanel llamado contentPane tiene un método llamado setBackground que nos permite fijar el color de fondo.

De forma similar asociamos los eventos para los otros dos objetos de la clase JMenuItem:

```
JMenuItem mntmVerde = new JMenuItem("Verde");
mntmVerde.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        contentPane.setBackground(Color.green);
    }
});
mnNewMenu.add(mntmVerde);
```

```
JMenuItem mntmAzul = new JMenuItem("Azul");
mntmAzul.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        contentPane.setBackground(Color.blue);
    }
});
```

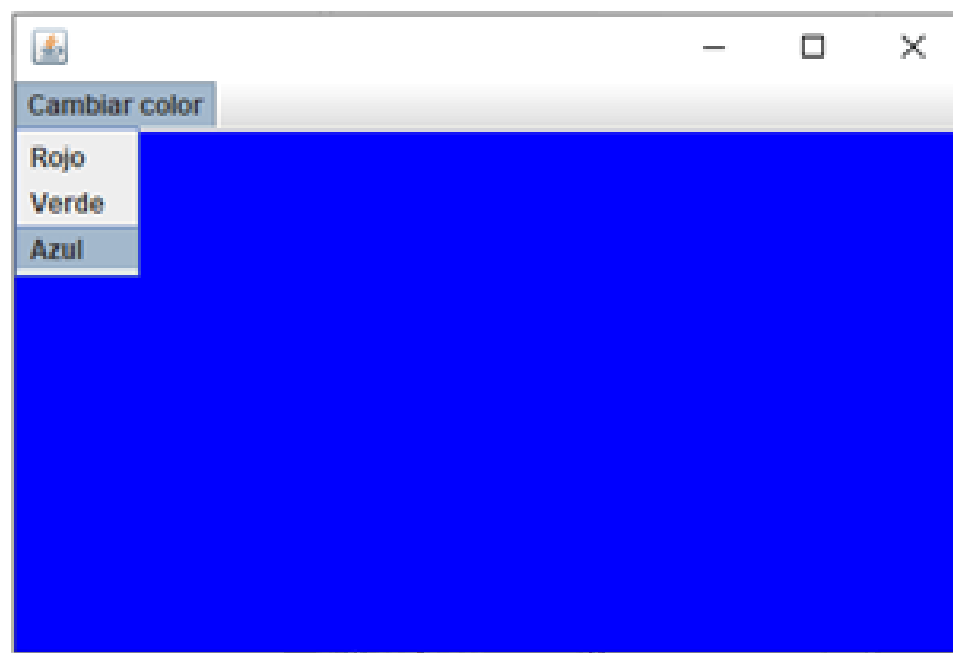
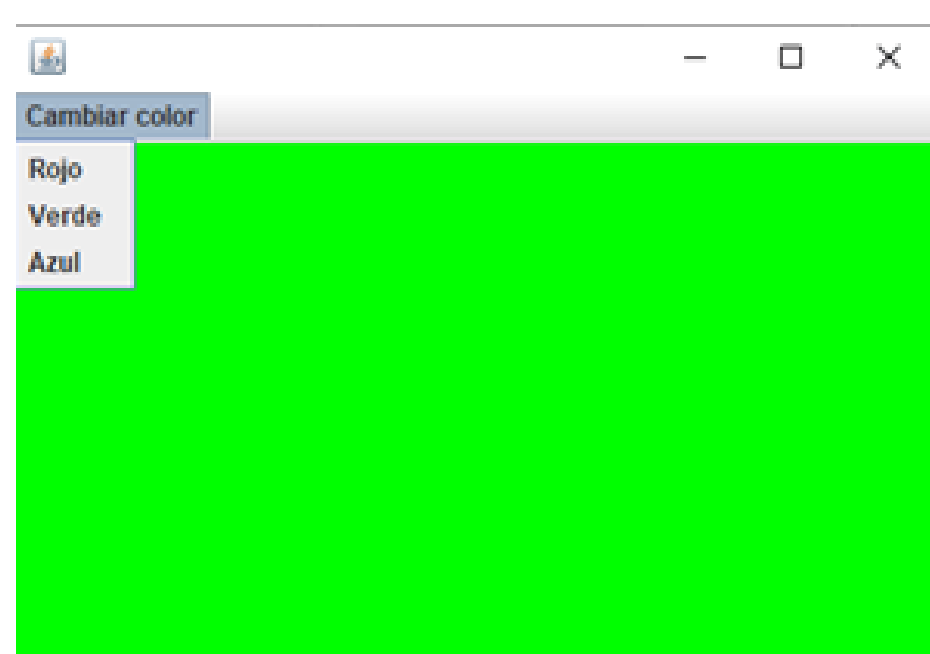
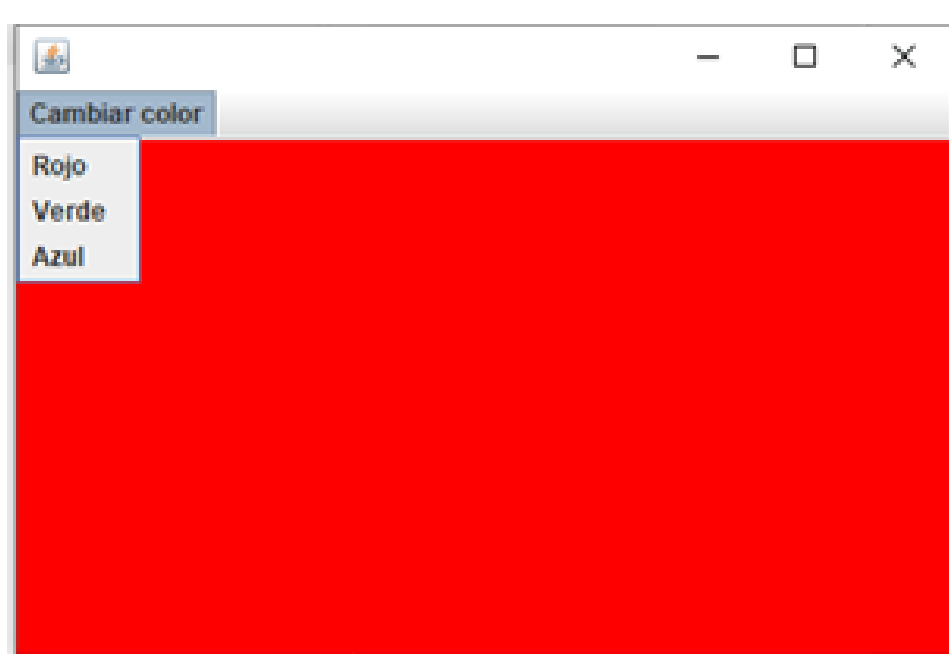
```

1  package ventana;
2
3  import java.awt.BorderLayout;
4  import java.awt.*;
5  import java.awt.EventQueue;
6
7  import javax.swing.JFrame;
8  import javax.swing.JPanel;
9  import javax.swing.border.EmptyBorder;
10 import javax.swing.JMenuBar;
11 import javax.swing.JMenu;
12 import javax.swing.JMenuItem;
13 import java.awt.event.ActionListener;
14 import java.awt.event.ActionEvent;
15
16 public class ventana1 extends JFrame {
17
18     private JPanel contentPane;
19
20     /**
21      * Launch the application.
22      */
23     public static void main(String[] args) {
24         EventQueue.invokeLater(new Runnable() {
25             public void run() {
26                 try {
27                     ventana1 frame = new ventana1();
28                     frame.setVisible(true);
29                 } catch (Exception e) {
30                     e.printStackTrace();
31                 }
32             }
33         });
34     }
35

```

```
37      * Create the frame.
38      */
39      public ventana1() {
40          setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
41          setBounds(100, 100, 450, 300);
42
43          JMenuBar menuBar = new JMenuBar();
44          setJMenuBar(menuBar);
45
46          JMenu mnNewMenu = new JMenu("Cambiar color");
47          menuBar.add(mnNewMenu);
48
49          JMenuItem mntmNewMenuItem = new JMenuItem("Rojo");
50          mntmNewMenuItem.addActionListener(new ActionListener() {
51              public void actionPerformed(ActionEvent e) {
52                  contentPane.setBackground(Color.red);
53              }
54          });
55          mnNewMenu.add(mntmNewMenuItem);
56
57          JMenuItem mntmNewMenuItem_1 = new JMenuItem("Verde");
58          mntmNewMenuItem_1.addActionListener(new ActionListener() {
59              public void actionPerformed(ActionEvent e) {
60                  contentPane.setBackground(Color.green);
61              }
62          });
63          mnNewMenu.add(mntmNewMenuItem_1);
64
65          JMenuItem mntmNewMenuItem_2 = new JMenuItem("Azul");
66          mntmNewMenuItem_2.addActionListener(new ActionListener() {
67              public void actionPerformed(ActionEvent e) {
68                  contentPane.setBackground(Color.blue);
69              }
70          });
71          mnNewMenu.add(mntmNewMenuItem_2);
72          contentPane = new JPanel();
73          contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
```

```
64
65 JMenuItem mntmNewMenuItem_2 = new JMenuItem("Azul");
66 mntmNewMenuItem_2.addActionListener(new ActionListener() {
67     public void actionPerformed(ActionEvent e) {
68         contentPane.setBackground(Color.blue);
69     }
70 });
71 mnNewMenu.add(mntmNewMenuItem_2);
72 contentPane = new JPanel();
73 contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
74 contentPane.setLayout(new BorderLayout(0, 0));
75 setContentPane(contentPane);
76 }
77
78 }
79
```



## 56 - Plug-in WindowBuilder problemas resueltos

Desarrollaremos una serie de aplicaciones que requieran componentes visuales y utilizaremos el WindowBuilder para agilizar su desarrollo.

### Problema 1

Desarrollar un programa que muestre el tablero de un ascensor:





El funcionamiento es el siguiente:

Inicialmente el ascensor está en el piso 1.

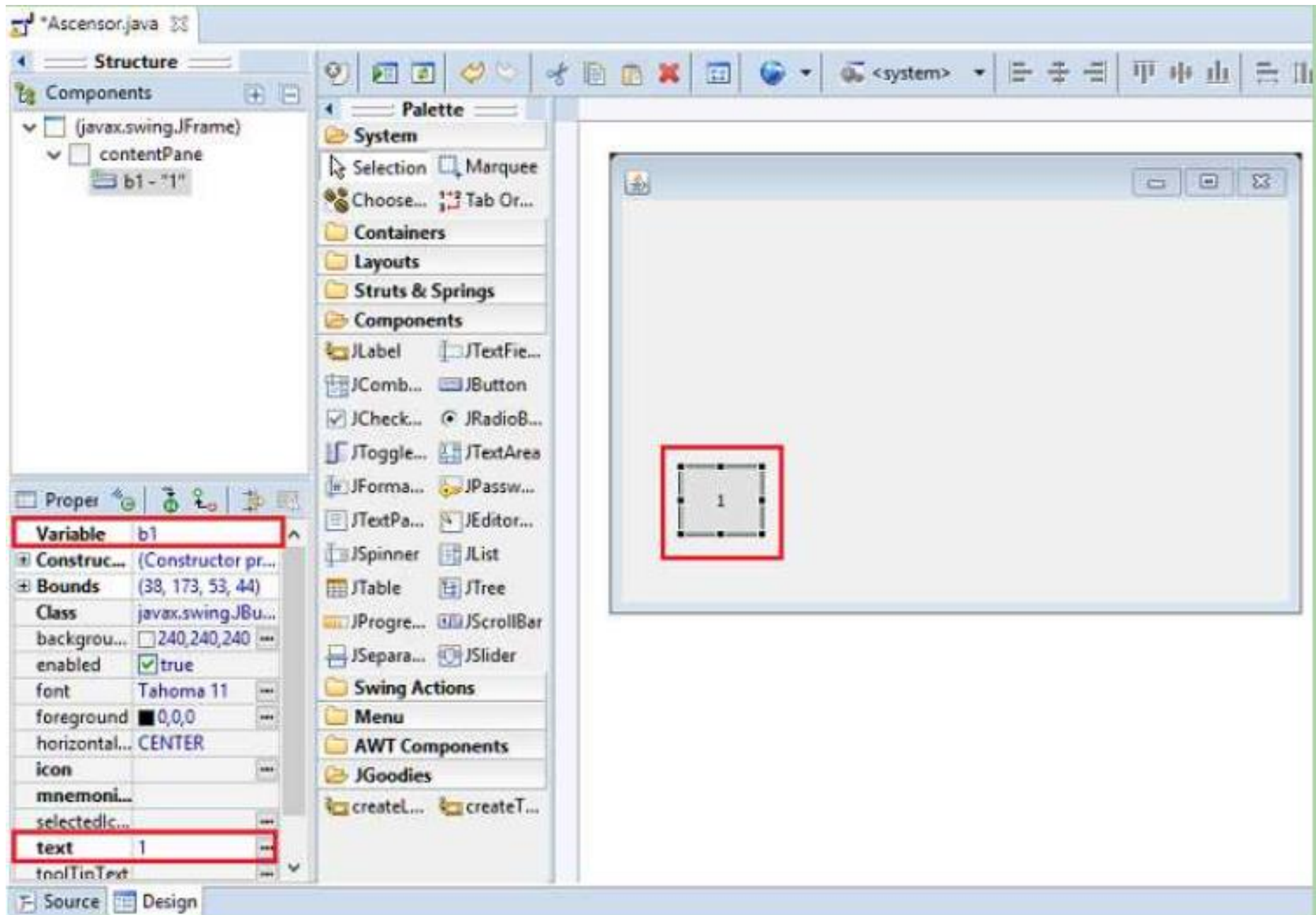
Por ejemplo: si se presiona el botón 3 se muestra en un JLabel el piso número 3 y en otra JLabel la dirección. La cadena "Sube", en caso de presionar un piso superior al actual. Mostramos la cadena "Baja" en el JLabel si se presiona un piso inferior. y si el piso donde se encuentra actualmente coincide con el presionado luego mostrar el mensaje "Piso actual".

Algunos consejos para crear la interfaz visual:

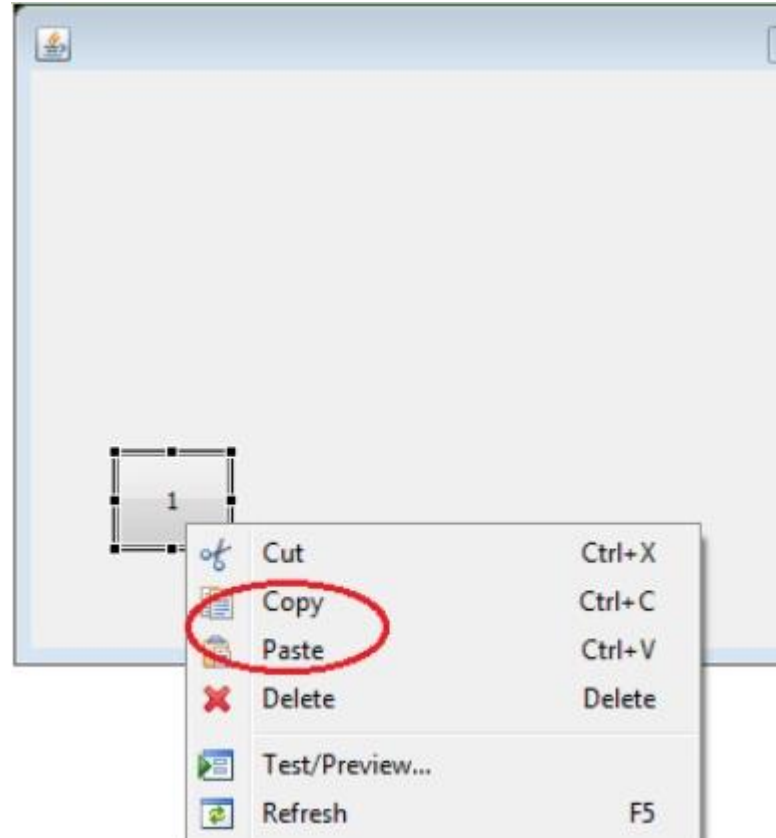
1 - Lo primero que debemos hacer cada vez que creamos un JFrame es definir el Layout a utilizar (normalmente utilizaremos "Absolute Layout", esto lo hacemos presionando el botón derecho del mouse dentro del JFrame y seleccionando la opción "Set Layout").

El tipo de layout a utilizar también se lo puede fijar seleccionando el objeto "contentPane"(este objeto es de la clase JPanel y todo JFrame lo contiene como fondo principal) y luego en la ventana de propiedades cambiamos la propiedad "Layout"

2 - Cuando creamos el primer JButton definimos el nombre del objeto cambiando la propiedad "Variable" y mediante la propiedad Text definimos el texto a mostrar (con el mouse dimensionamos el JButton):



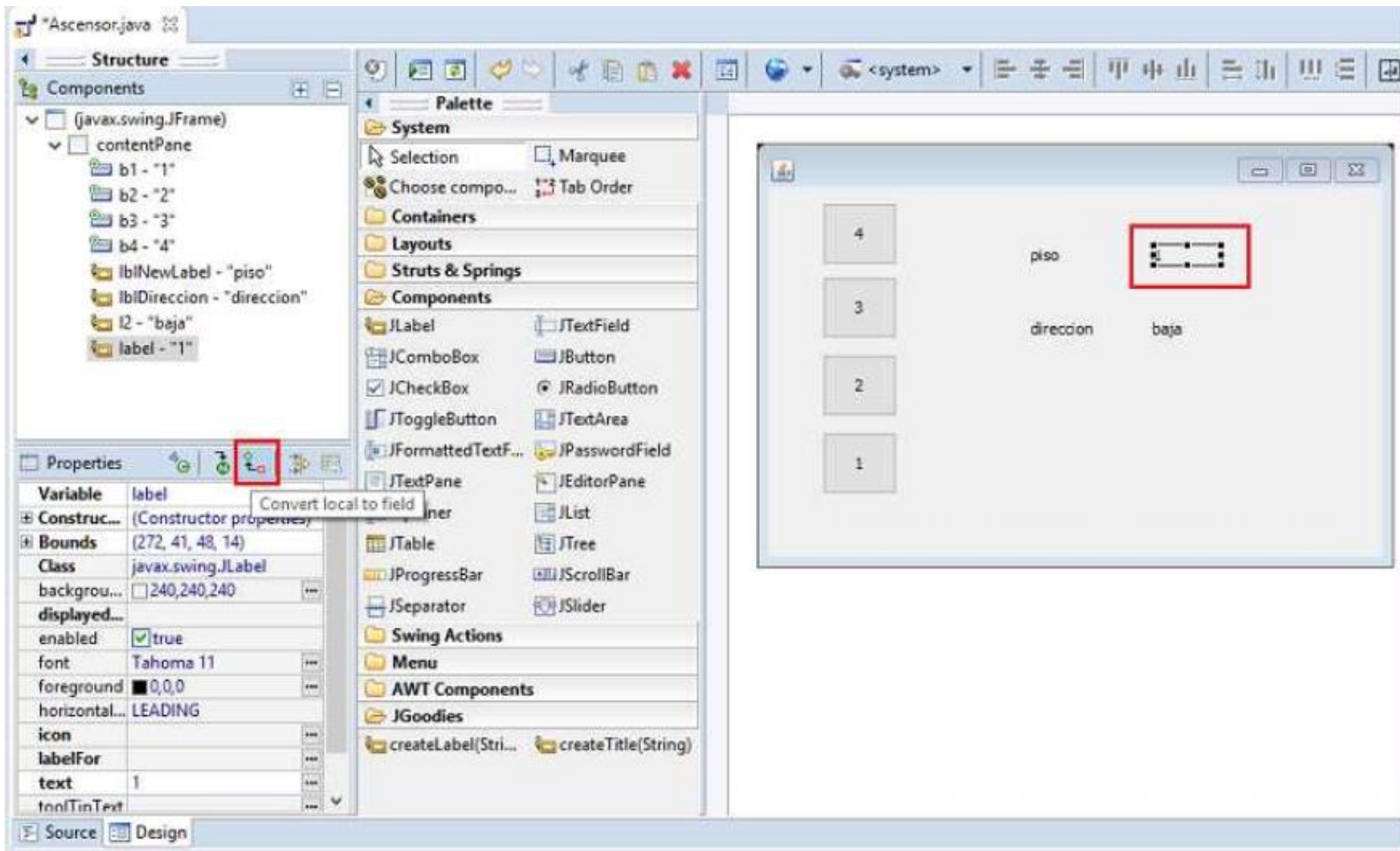
3 - Los otros botones los podemos crear de la misma manera seleccionando un objeto de la clase JButton de la "Palette" o cuando tenemos que crear otros objetos semejantes podemos presionar el botón derecho del mouse sobre el objeto a duplicar y seguidamente en el menú contextual seleccionar la opción "Copy" y seguidamente la opción "Paste" con lo que tendremos otro objeto semejante. Después, deberemos definir un nombre para el objeto (propiedad "Variable") y la propiedad "text" para la etiqueta a mostrar:



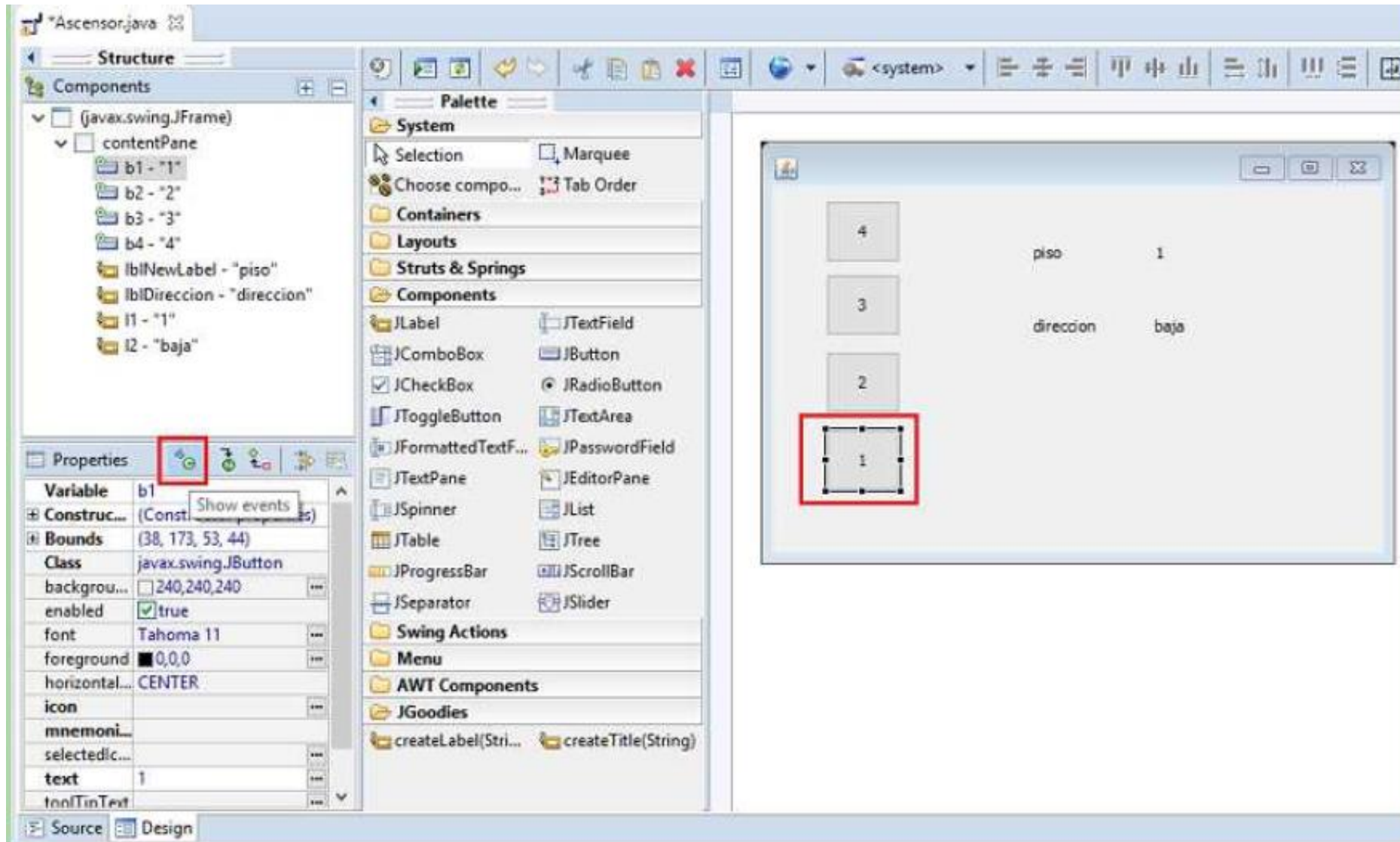
4 - Los objetos que necesitemos consultar o modificar en tiempo de ejecución debemos definirlos como atributos de clase (también llamados campos de clase)

En este problema cuando se presione alguno de los cuatro botones debemos consultar el contenido de la label que indica el piso actual y la label que muestra la dirección será modificada por otro String.

Para definir un control visual como atributo de clase debemos seleccionarlo y presionar en la ventana de propiedades el botón "Convert local to field" (en nuestro problema definamos a estos dos objetos de la clase JLabel con el nombre l1 y l2):

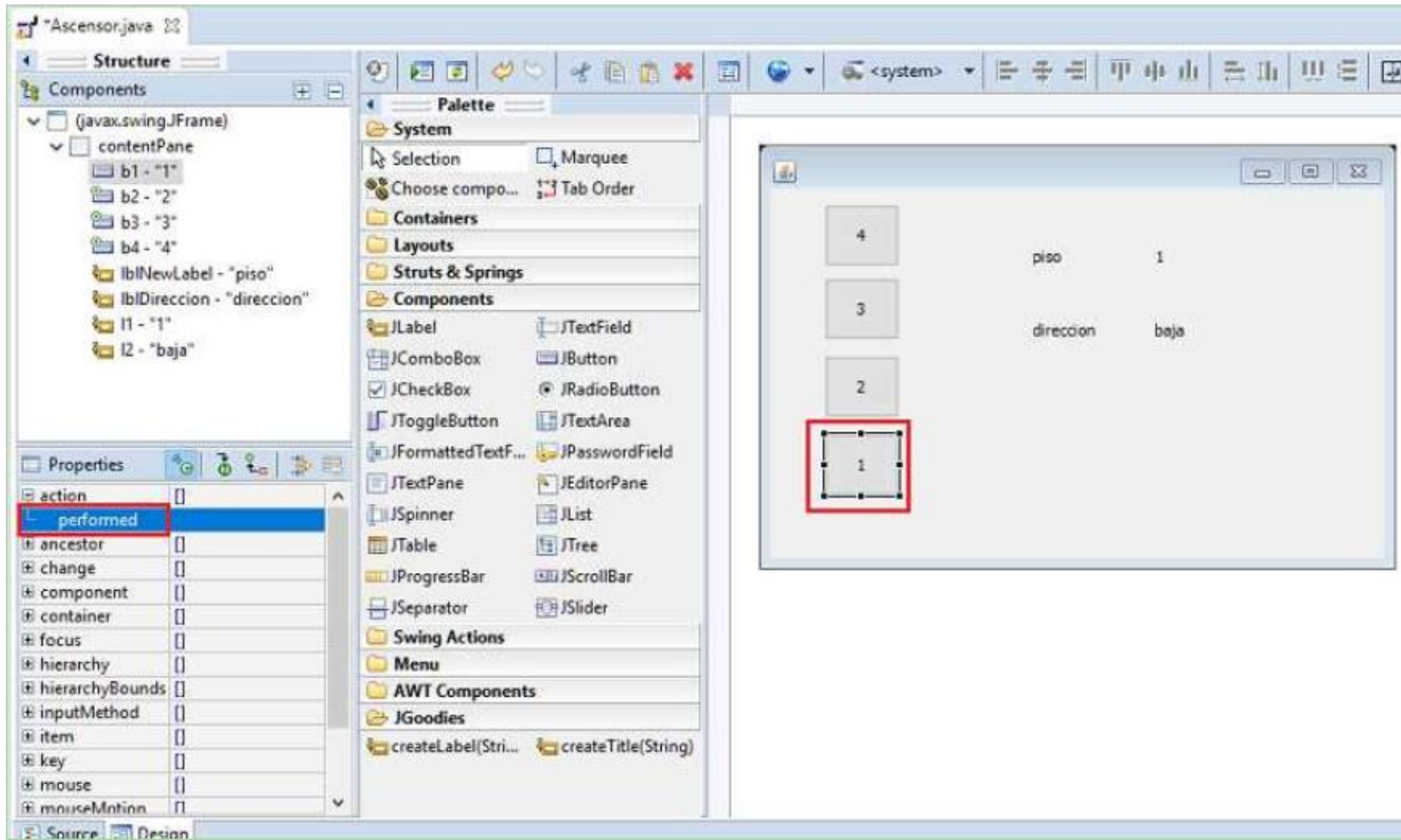


5 - Para capturar el evento clic de un objeto de la clase JButton debemos seleccionarlo y presionar el botón "Show Events":





y seguidamente hacer doble-clic sobre el evento a implementar (performed), se abre el editor y codificamos la lógica para dicho evento:



La solución a este problema es el siguiente:

```
import java.awt.EventQueue;  
import javax.swing.JFrame;  
import javax.swing.JPanel;  
import javax.swing.border.EmptyBorder;  
import javax.swing.JButton;  
import javax.swing.JLabel;  
import java.awt.event.ActionListener;  
import java.awt.event.ActionEvent;
```

```
public class Ascensor extends JFrame {
```

```
    private JPanel contentPane;  
    private JLabel l1;  
    private JLabel l2;
```



```
/**
 * Launch the application.
 */
public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                Ascensor frame = new Ascensor();
                frame.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}
```

```

/**
 * Create the frame.
 */
public Ascensor() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 300);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(null);

    JButton b1 = new JButton("1");
    b1.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            int pisoactual=Integer.parseInt(l1.getText());
            if (1<pisoactual)
                l2.setText("Baja");
            else
                l2.setText("Piso actual");
            l1.setText("1");
        }
    });
}

```

```
b1.setBounds(38, 173, 53, 44);  
contentPane.add(b1);
```

```
 JButton b2 = new JButton("2");  
 b2.addActionListener(new ActionListener() {  
     public void actionPerformed(ActionEvent e) {  
         int pisoactual=Integer.parseInt(l1.getText());  
         if (2<pisoactual)  
             l2.setText("Baja");  
         else  
             if (2>pisoactual)  
                 l2.setText("Sube");  
             else  
                 l2.setText("Piso actual");  
         l1.setText("2");  
     }  
 });
```

```
b2.setBounds(38, 118, 53, 44);  
contentPane.add(b2);
```

```
 JButton b3 = new JButton("3");  
 b3.addActionListener(new ActionListener() {  
     public void actionPerformed(ActionEvent e) {  
         int pisoactual=Integer.parseInt(l1.getText());  
         if (3<pisoactual)  
             l2.setText("Baja");  
         else  
             if (3>pisoactual)  
                 l2.setText("Sube");  
             else  
                 l2.setText("Piso actual");  
         l1.setText("3");  
     }  
 });  
 b3.setBounds(38, 63, 53, 44);  
 contentPane.add(b3);
```

```
JButton b4 = new JButton("4");
    b4.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            int pisoactual=Integer.parseInt(l1.getText());
            if (4>pisoactual)
                l2.setText("Sube");
            else
                l2.setText("Piso actual");
            l1.setText("4");
        }
    });
```

```
b4.setBounds(38, 11, 53, 44);  
    contentPane.add(b4);
```

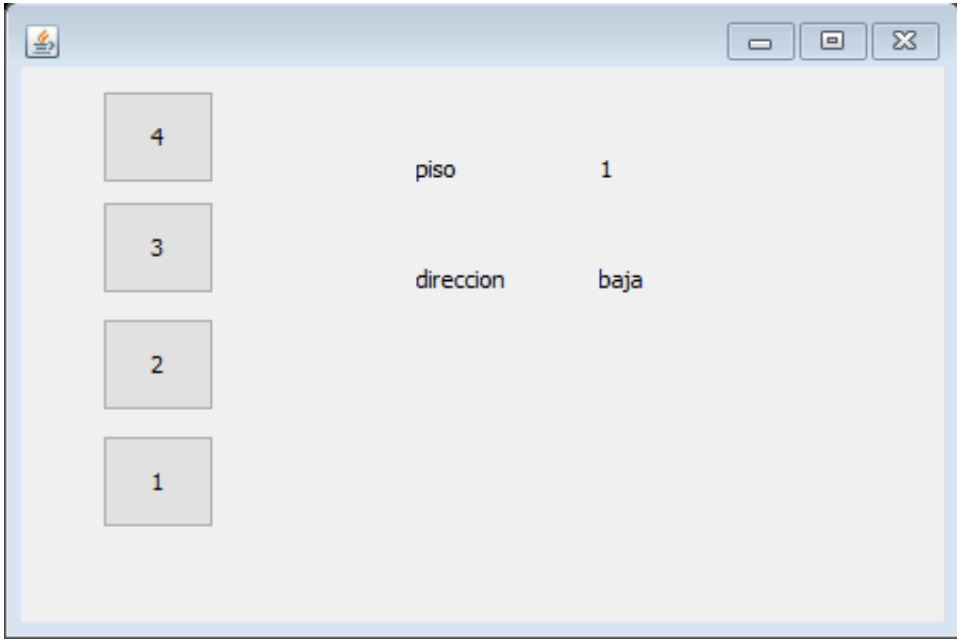
```
JLabel lblNewLabel = new JLabel("piso");  
lblNewLabel.setBounds(186, 41, 46, 14);  
contentPane.add(lblNewLabel);
```

```
JLabel lblDireccion = new JLabel("direccion");  
lblDireccion.setBounds(186, 93, 61, 14);  
contentPane.add(lblDireccion);
```

```
l1 = new JLabel("1");  
l1.setBounds(272, 41, 46, 14);  
contentPane.add(l1);
```

```
l2 = new JLabel("baja");  
l2.setBounds(272, 93, 92, 14);  
contentPane.add(l2);
```

```
    }  
}
```



Cuando se presiona el botón 1 procedemos a extraer el contenido de la label 1 que almacena el valor del piso actual, como se presionó el primer botón preguntamos si 1 es menor al piso actual, en dicho caso mostramos en la label 2 el texto "Baja" en caso contrario significa que estamos actualmente en el piso 1 (cuando se presiona el botón 1 nunca puede decir el texto sube) Luego cambiamos la etiqueta de la label 1 con el valor "1" que es el nuevo piso:

```
b1.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        int pisoactual=Integer.parseInt(l1.getText());  
        if (1<pisoactual)  
            l2.setText("Baja");  
        else  
            l2.setText("Piso actual");  
        l1.setText("1");  
    }  
});
```



El botón 4 es similar al botón 1 ya que mostraremos la etiqueta "Sube" o "Piso actual":

```
b4.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        int pisoactual=Integer.parseInt(l1.getText());  
        if (4>pisoactual)  
            l2.setText("Sube");  
        else  
            l2.setText("Piso actual");  
        l1.setText("4");  
    }  
});
```

Si se presiona el botón del segundo piso debemos verificar si 2 es menor, mayor o igual al piso actual (igual para el botón del tercer piso):

```
b2.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        int pisoactual=Integer.parseInt(l1.getText());  
        if (2<pisoactual)  
            l2.setText("Baja");  
        else  
            if (2>pisoactual)  
                l2.setText("Sube");  
            else  
                l2.setText("Piso actual");  
        l1.setText("2");  
    }  
});
```

## Problema 2

Desarrollar un programa que muestre un panel para extracción de una bebida:

☒ Bebida A      Euros      0 ▾

☐ Bebida B      Centimos      0 ▾

☐ Bebida C

Extraer      resultado

Lo primero que debemos hacer cada vez que creamos un JFrame es definir el Layout a utilizar (normalmente utilizaremos "Absolute Layout", esto lo hacemos presionando el **botón derecho del mouse dentro del JFrame y seleccionando la opción "Set Layout"**).

Por un lado disponer tres objetos de la clase JRadioButton (llamarlos radio1, radio2 y radio 3), configurar el primero para que aparezca seleccionado (propiedad "selected")

Disponer dos objetos de la clase JComboBox (llamarlos comboEuros y comboCentimos)

En el JComboBox euros inicializar la propiedad model con los valores del 0 al 5 (hay que cargar un valor por cada línea en el diálogo que aparece)

En forma similar el segundo JComboBox cargamos los valores: 0,10,20,30 etc. hasta 90.

Se sabe que :

Bebida A tiene un costo de 0 euros 80 céntimos.

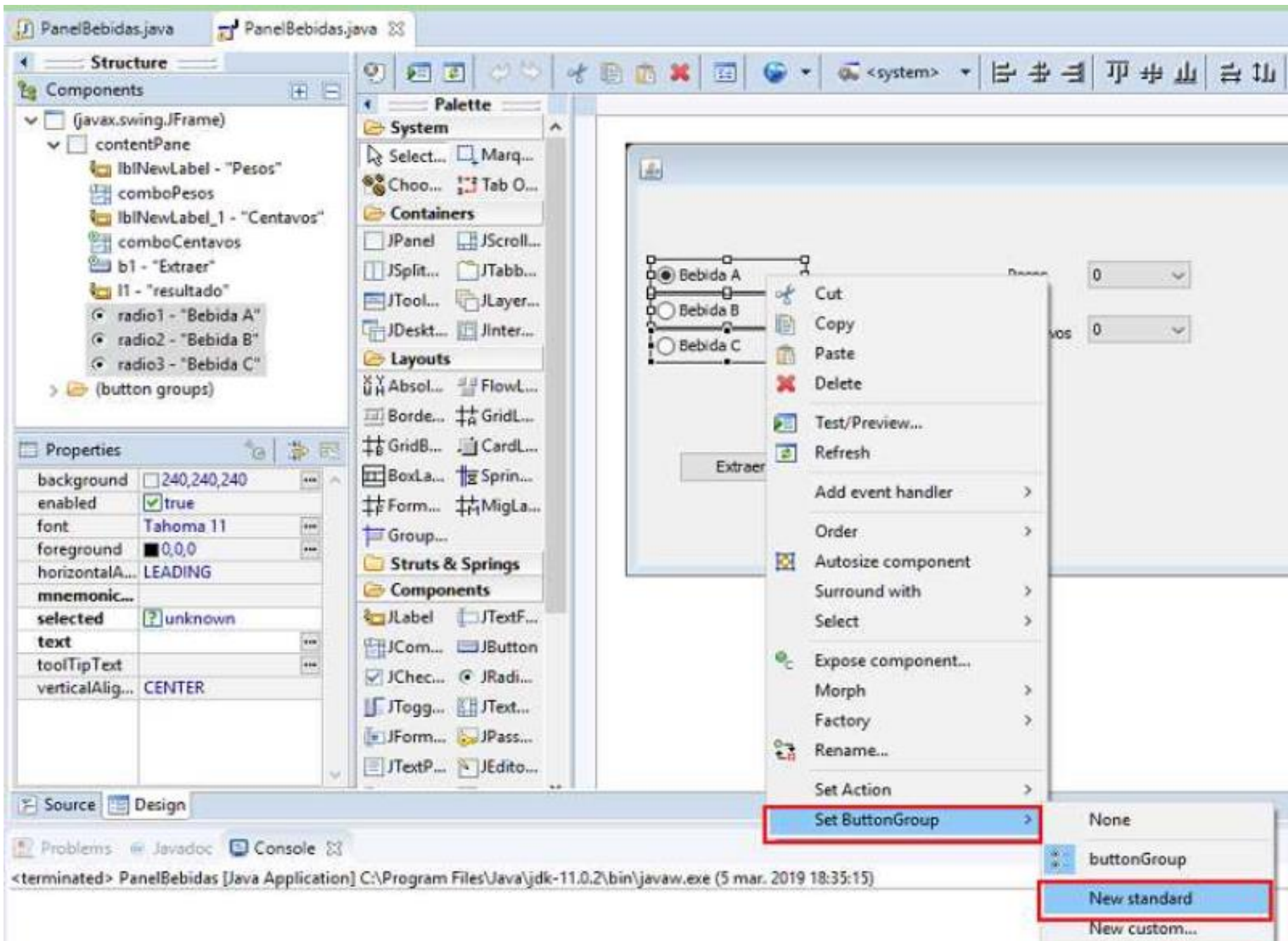
Bebida B tiene un costo de 1 euro 20 céntimos.

Bebida C tiene un costo de 3 euros 10 céntimos.

Cuando se presiona el botón extraer mostrar en la label de resultado el texto "Correcto" o "Incorrecto" dependiendo la bebida seleccionada y la cantidad de euros y céntimos seleccionados.

Solución:

Para que todos los JRadioButton estén asociados (es decir que cuando se seleccione uno se deseccione el actual lo debemos hacer en forma visual), primero seleccionamos con el mouse todos los JRadioButton (para seleccionar varios controles presionamos la tecla "Ctrl" del teclado y con el boton izquierdo del mouse seleccionamos los tres JRadioButton) y seguidamente presionamos el botón derecho del mouse y seleccionamos "New standard":



Ahora ya tenemos los tres controles de tipo JRadioButton agrupados.

El código fuente del problema es:

```
import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JRadioButton;
import javax.swing.JLabel;
import javax.swing.JComboBox;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JButton;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.ButtonGroup;
```



```
public class PanelBebidas extends JFrame {  
  
    private JPanel contentPane;  
    private JComboBox comboEuros;  
    private JComboBox comboCentimos;  
    private JRadioButton radio1;  
    private JRadioButton radio2;  
    private JRadioButton radio3;  
    private JLabel l1;  
    private final ButtonGroup buttonGroup = new ButtonGroup();
```

```
/**
 * Launch the application.
 */
public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                PanelBebidas frame = new PanelBebidas();
                frame.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}
```

```
/**
 * Create the frame.
 */
public PanelBebidas() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 600, 319);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(null);

    JLabel lblNewLabel = new JLabel("Euros");
    lblNewLabel.setBounds(263, 59, 46, 14);
    contentPane.add(lblNewLabel);

    comboEuros= new JComboBox();
    comboEuros.setModel(new DefaultComboBoxModel(new String[] {"0", "1", "2", "3",
"4", "5"}));
    comboEuros.setBounds(319, 56, 73, 20);
    contentPane.add(comboEuros);

    JLabel lblNewLabel_1 = new JLabel("Centimos");
    lblNewLabel_1.setBounds(263, 102, 58, 14);
    contentPane.add(lblNewLabel_1);
```

```
comboCentimos = new JComboBox();  
    comboCentimos.addActionListener(new ActionListener() {  
        public void actionPerformed(ActionEvent arg0) {  
        }  
    });
```

```
comboCentimos.setModel(new DefaultComboBoxModel(new String[] {"0", "10", "20",  
"30", "40", "50", "60", "70", "80", "90"}));  
    comboCentimos.setBounds(319, 96, 73, 20);  
    contentPane.add(comboCentimos);  
  
    JButton b1 = new JButton("Extraer");  
    b1.addActionListener(new ActionListener() {  
        public void actionPerformed(ActionEvent arg0) {  
            int euros=Integer.parseInt((String)comboEuros.getSelectedItem());  
            int centimos=Integer.parseInt((String)comboCentimos.getSelectedItem());  
            if (radio1.isSelected() && euros==0 && centimos==80)  
                l1.setText("Correcto");  
            else  
                if (radio2.isSelected() && euros==1 && centimos==20)  
                    l1.setText("Correcto");  
                else  
                    if (radio3.isSelected() && euros==3 && centimos==10)  
                        l1.setText("Correcto");  
                    else  
                        l1.setText("Incorrecto");  
                }  
        }  
    });
```

```
b1.setBounds(30, 196, 89, 23);
```

```
contentPane.add(b1);
```

```
l1 = new JLabel("resultado");
```

```
l1.setBounds(148, 205, 73, 14);
```

```
contentPane.add(l1);
```

```
radio1 = new JRadioButton("Bebida A");
```

```
buttonGroup.add(radio1);
```

```
radio1.setSelected(true);
```

```
radio1.setBounds(10, 55, 109, 23);
```

```
contentPane.add(radio1);
```

```
radio2 = new JRadioButton("Bebida B");
```

```
buttonGroup.add(radio2);
```

```
radio2.setBounds(10, 81, 109, 23);
```

```
contentPane.add(radio2);
```

```
radio3 = new JRadioButton("Bebida C");
```

```
buttonGroup.add(radio3);
```

```
radio3.setBounds(10, 107, 109, 23);
```

```
contentPane.add(radio3);
```

```
}
```

```
}
```

La lógica del problema se encuentra cuando se presiona el botón "Extraer":

```
b1.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent arg0) {  
        int euros=Integer.parseInt((String)comboEuros.getSelectedItem());  
        int centimos=Integer.parseInt((String)comboCentimos.getSelectedItem());  
        if (radio1.isSelected() && euros==0 && centimos==80)  
            l1.setText("Correcto");  
        else  
            if (radio2.isSelected() && euros==1 && centimos==20)  
                l1.setText("Correcto");  
            else  
                if (radio3.isSelected() && euros==3 && centimos==10)  
                    l1.setText("Correcto");  
                else  
                    l1.setText("Incorrecto");  
            }  
    }  
});
```

Extraemos los contenidos de los dos controles de tipo JComboBox y los convertimos a entero. Luego mediante tres if verificamos si el primer JRadioButton está seleccionado y el dinero seleccionado corresponde a exactamente 0 euros y 80 centimos, en tal caso mostramos en la label el mensaje "Correcto". La lógica es similar para las otras dos bebidas.





☒ Bebida A

☐ Bebida B

☐ Bebida C

Euros

1 ▼

Centimos

50 ▼

Extraer

Incorrecto



☒ Bebida A

☐ Bebida B

☐ Bebida C

Euros

0



Centimos

80



Extraer

Correcto



☐ Bebida A

☒ Bebida B

☐ Bebida C

Euros

1



Centimos

20



Extraer

Correcto



☐ Bebida A

☐ Bebida B

☒ Bebida C

Euros

Centimos

Extraer

Correcto

## Problema 3

Un embalse debe manejar la cantidad de mts de agua que pasa por cada compuerta. Por cada compuerta puede pasar un caudal de 100 mts<sup>3</sup> x seg. Cuando presionamos el botón "Actualizar caudal" mostramos el nivel de caudal actual y un mensaje que indica si el caudal es Bajo (0 a 100 mts<sup>3</sup> x seg.) , Medio (> 100 -200 mts<sup>3</sup>. x seg.) o Alto (>200 mts<sup>3</sup> x seg.) Para la selección del caudal de cada compuerta utilizar componentes de tipo JSpinner.



Compuerta 1



Compuerta 2



Compuerta 3

Actualizar Caudal

resultado

El código fuente es:

```
import java.awt.EventQueue;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;
import javax.swing.JSpinner;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.SpinnerNumberModel;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class Embalse extends JFrame {

    private JPanel contentPane;
    private JSpinner spinner1;
    private JSpinner spinner2;
    private JSpinner spinner3;
    private JLabel l1;
```

```
/**
 * Launch the application.
 */
public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                Embalse frame = new Embalse();
                frame.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}
```



```
/**
 * Create the frame.
 */
public Embalse() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 300);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(null);

    spinner1 = new JSpinner();
    spinner1.setModel(new SpinnerNumberModel(0, 0, 100, 1));
    spinner1.setBounds(31, 35, 62, 20);
    contentPane.add(spinner1);

    spinner2 = new JSpinner();
    spinner2.setModel(new SpinnerNumberModel(0, 0, 100, 1));
    spinner2.setBounds(31, 85, 62, 20);
    contentPane.add(spinner2);

    spinner3 = new JSpinner();
    spinner3.setModel(new SpinnerNumberModel(0, 0, 100, 1));
    spinner3.setBounds(31, 134, 62, 20);
    contentPane.add(spinner3);
}
```

```
JLabel lblCompuerta = new JLabel("compuerta 1");  
    lblCompuerta.setBounds(106, 38, 82, 14);  
    contentPane.add(lblCompuerta);
```

```
JLabel lblCompuerta_1 = new JLabel("compuerta 2");  
    lblCompuerta_1.setBounds(106, 88, 82, 14);  
    contentPane.add(lblCompuerta_1);
```

```
JLabel lblCompuerta_2 = new JLabel("compuerta 3");  
    lblCompuerta_2.setBounds(106, 137, 82, 14);  
    contentPane.add(lblCompuerta_2);
```

```

JButton btnNewButton = new JButton("Actualizar caudal");
btnNewButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        int v1=Integer.parseInt(spinner1.getValue().toString());
        int v2=Integer.parseInt(spinner2.getValue().toString());
        int v3=Integer.parseInt(spinner3.getValue().toString());
        int suma=v1+v2+v3;
        if (suma<=100)
            l1.setText("Bajo");
        else
            if (suma<=200)
                l1.setText("Medio");
            else
                l1.setText("Alto");
    }
});

```

```
btnNewButton.setBounds(31, 198, 157, 23);  
    contentPane.add(btnNewButton);  
  
    l1 = new JLabel("resultado");  
    l1.setBounds(218, 203, 149, 14);  
    contentPane.add(l1);  
}  
}
```

En el evento clic del JButton extraemos los tres valores almacenados en los JSpinner:

```
int v1=Integer.parseInt(spinner1.getValue().toString());  
int v2=Integer.parseInt(spinner2.getValue().toString());  
int v3=Integer.parseInt(spinner3.getValue().toString());
```

y mediante tres if valuamos si la suma es menor o igual a 100 o menor o igual a 200 o en su defecto es mayor a 200:

```
int suma=v1+v2+v3;  
if (suma<=100)  
    l1.setText("Bajo");  
else  
    if (suma<=200)  
        l1.setText("Medio");  
    else  
        l1.setText("Alto");
```



compuerta 1



compuerta 2



compuerta 3

Actualizar caudal

Bajo



100



compuerta 1

29



compuerta 2

5



compuerta 3

Actualizar caudal

Medio



100



compuerta 1

100



compuerta 2

23



compuerta 3

Actualizar caudal

Alto



# Problema propuesto

Implementar un programa para la extracción de dinero de un cajero automático.

Se debe poder fijar la cantidad de dinero a extraer:

Disponer un control de tipo JComboBox (disponer los valores: 0,50,150 etc. hasta 500)

Por otro lado poder seleccionar el tipo de cuenta (almacenar en otro JComboBox los textos "Caja de Ahorro" y "Cuenta Corriente").

Se debe tener en cuenta que:

De Caja de Ahorro se puede extraer hasta 200.

De Cuenta Corriente se puede extraer hasta 500.

Al presionar el botón extraer mostrar en una label el texto "correcto" si para el tipo de cuenta el importe está permitido.

Inicialmente el cajero tiene almacenado un monto de 3000 euros. Restar en cada extracción el monto respectivo y mostrar el mensaje "fuera de servicio" cuando se intenta extraer más del dinero que hay en el cajero.

Dinero a extraer

0



Tipo de cuenta

Caja de ahorro



Extraer

Resultado

\*CajeroAutomatico.java X

```
1 import javax.swing.JPanel;
8
9 public class CajeroAutomatico extends JPanel {
10     private JComboBox combo1;
11     private JComboBox combo2;
12     private JLabel lbl1;
13     private int dinero;
14     /**
15      * Create the panel.
16      */
17     public CajeroAutomatico() {
18         setLayout(null);
19         dinero = 3000;
20         combo1 = new JComboBox();
21         combo1.setModel(new DefaultComboBoxModel(new String[] {"0", "50", "100", "150", "200", "250", "300", "350", "400", "450", "500"}));
22         combo1.setBounds(31, 70, 141, 22);
23         add(combo1);
24
25         JLabel lblNewLabel = new JLabel("Tipo de cuenta");
26         lblNewLabel.setBounds(223, 27, 109, 18);
27         add(lblNewLabel);
28
29         JLabel lblNewLabel_1 = new JLabel("Dinero a extraer");
30         lblNewLabel_1.setBounds(31, 25, 141, 22);
31         add(lblNewLabel_1);
32
33         combo2 = new JComboBox();
34         combo2.setModel(new DefaultComboBoxModel(new String[] {"Caja de ahorro", "Cuenta corriente"}));
35         combo2.setBounds(223, 70, 109, 22);
36         add(combo2);
37
```

```

37
38 JButton btnNewButton = new JButton("Extraer");
39 btnNewButton.addActionListener(new ActionListener() {
40     public void actionPerformed(ActionEvent e) {
41         int dineroExtraer=Integer.parseInt(combo1.getSelectedItem().toString());
42         if (dineroExtraer > dinero) {
43             label1.setText("El cajero esta fuera de servicio");
44         }else {
45             String tipoDeCuenta=combo2.getSelectedItem().toString();
46             if (tipoDeCuenta.equals("Caja de Ahorro") && dineroExtraer <= 200) {
47                 label1.setText("Correcto");
48                 dinero = dinero - dineroExtraer;
49             }else {
50                 if (tipoDeCuenta.equals("Cuenta corriente") && dineroExtraer <= 500) {
51                     label1.setText("Correcto");
52                     dinero = dinero -dineroExtraer;
53                 }else {
54                     label1.setText("Incorrecto cantidad muy grande");
55                 }
56             }
57         }
58     }
59 });
60 btnNewButton.setBounds(31, 215, 89, 23);
61 add(btnNewButton);
62
63 label1 = new JLabel("Resultado");
64 label1.setBounds(175, 215, 157, 18);
65 add(label1);
66
67 }
68 }
69

```

## 57 - Clase Graphics y sus métodos

Java proporciona la clase Graphics, que permite dibujar elipses, cuadrados, líneas, mostrar texto y también tiene muchos otros métodos de dibujo. Para cualquier programador, es esencial el entendimiento de la clase Graphics, antes de adentrarse en el dibujo en Java.

La clase Graphics proporciona el entorno de trabajo para cualquier operación gráfica que se realice dentro del AWT.

Para poder pintar, un programa necesita un contexto gráfico válido, representado por una instancia de la clase Graphics. Pero esta clase no se puede instanciar directamente; así que debemos crear un componente y pasarlo al programa como un argumento al método paint().

El único argumento del método `paint()` es un objeto de esta clase. La clase `Graphics` dispone de métodos para soportar tres categorías de operaciones gráficas:

- 1) Dibujo de primitivas gráficas.
- 2) Dibujo de texto.
- 3) Presentación de imágenes en formatos `*.gif` y `*.jpeg`.

Además, la clase `Graphics` mantiene un contexto gráfico: un área de dibujo actual, un color de dibujo del `Background` y otro del `Foreground`, un `Font` con todas sus propiedades, etc. Los ejes están situados en la esquina superior izquierda. Las coordenadas se miden siempre en pixels.

## **Problema 1**

Crear una aplicación que utilice las primitivas gráficas principales que provee la clase Graphics:

```
import java.awt.BorderLayout;  
import java.awt.Color;  
import java.awt.EventQueue;  
import java.awt.Graphics;
```

```
import javax.swing.JFrame;  
import javax.swing.JPanel;  
import javax.swing.border.EmptyBorder;
```



```
public class Grafico1 extends JFrame {

    private JPanel contentPane;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Grafico1 frame = new Grafico1();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

```
/**  
 * Create the frame.  
 */  
public Grafico1() {  
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    setBounds(100, 100, 450, 300);  
    contentPane = new JPanel();  
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));  
    setContentPane(contentPane);  
    contentPane.setLayout(null);  
    setBounds(0,0,800,600);  
}
```

```
public void paint (Graphics g)
{
    super.paint(g);

    g.setColor (Color.blue);
    g.drawLine (0, 70, 100, 70);
    g.drawRect (150, 70, 50, 70);
    g.drawRoundRect (250, 70, 50, 70, 6, 6);
    g.drawOval (350, 70, 50, 70);
    int [] vx1 = {500, 550, 450};
    int [] vy1 = {70, 120, 120};
    g.drawPolygon (vx1, vy1, 3);

    g.setColor (Color.red);
    g.fillRect (150, 270, 50, 70);
    g.fillRoundRect (250, 270, 50, 70, 6, 6);
    g.fillOval (350, 270, 50, 70);
    int [] vx2 = {500, 550, 450};
    int [] vy2 = {270, 320, 320};
    g.fillPolygon (vx2, vy2, 3);
}
}
```



Sobreescribimos el método paint heredado de la clase JFrame:

```
public void paint (Graphics g)
{
```

El método paint se ejecuta cada vez que el JFrame debe ser redibujado y llega como parámetro un objeto de la clase Graphics. Este objeto nos permite acceder al fondo del JFrame y utilizando las primitivas gráficas dibujar líneas, rectángulos, elipses etc.

Lo primero que hacemos dentro de este método es llamar al método paint de la clase superior para que se pinte el fondo del JFrame y otras componentes contenidas dentro (para llamar al método paint de la clase JFrame debemos anteceder la palabra clave super y pasar el parámetro respectivo):

```
super.paint(g);
```

Mediante el método setColor activamos un color:

```
g.setColor (Color.blue);
```

Dibuja una línea desde la coordenada (0,70) es decir columna 0 y fila 70 en píxeles, hasta la coordenada (100,70). La línea es de color azul:

```
g.drawLine (0, 70, 100, 70);
```

Dibujamos un rectángulo desde la coordenada (150,70) con un ancho de 50 píxeles y un alto de 70, solo se pinta el perímetro del rectángulo de color azul):

```
g.drawRect (150, 70, 50, 70);
```

Similar a drawRect más un valor de redondeo de los vértices que le indicamos en el quinto y sexto parámetro:

```
g.drawRoundRect (250, 70, 50, 70, 6, 6);
```

Dibujamos un óvalo:

```
g.drawOval (350, 70, 50, 70);
```

Dibujamos un triángulo (debemos indicar mediante dos vectores los vértices de cada punto del triángulo), el primer punto es el (500,70) el segundo punto es el (550,120) y por último el punto (450,120):

```
int [] vx1 = {500, 550, 450};  
int [] vy1 = {70, 120, 120};  
g.drawPolygon (vx1, vy1, 3);
```

De forma similar los métodos fillRect, fillRoundRect, fillOval y fillPolygon son similares a los anteriores con la diferencia que pinta su interior con el color activo de la última llamada al método setColor:

```
g.setColor (Color.red);  
g.fillRect (150, 270, 50, 70);  
g.fillRoundRect (250, 270, 50, 70, 6, 6);  
g.fillOval (350, 270, 50, 70);  
int [] vx2 = {500, 550, 450};  
int [] vy2 = {270, 320, 320};  
g.fillPolygon (vx2, vy2, 3);
```

## Dibujar texto

La clase Graphics permite 'dibujar' texto, como alternativa al texto mostrado en los componentes JLabel, JTextField y JTextArea. El método que permite graficar texto sobre el JFrame es:

```
drawString(String str, int x, int y);
```



## Problema 2

Crear una aplicación que utilice las primitiva drawString de Java:

```
import java.awt.BorderLayout;  
import java.awt.Color;  
import java.awt.EventQueue;  
import java.awt.Graphics;  
  
import javax.swing.JFrame;  
import javax.swing.JPanel;  
import javax.swing.border.EmptyBorder;
```

```
public class Grafico1 extends JFrame {

    private JPanel contentPane;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Grafico1 frame = new Grafico1();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

```
/**
 * Create the frame.
 */
public Grafico1() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 300);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    contentPane.setLayout(new BorderLayout(0, 0));
    setContentPane(contentPane);
    setBounds(0,0,800,600);
}
```

```
public void paint (Graphics g)
{
    super.paint(g);
    g.setColor (Color.blue);
    g.drawString("Primer linea",10,200);
    g.drawString("Segunda linea",10,300);
}

}
```



Primer linea

Segunda linea

## **Clase Color**

La clase `java.awt.Color` encapsula colores utilizando el formato RGB (Red, Green, Blue). Las componentes de cada color primario en el color resultante se expresan con números enteros entre 0 y 255, siendo 0 la intensidad mínima de ese color y 255 la máxima. En la clase `Color` existen constantes para colores predeterminados de uso frecuente: `black`, `white`, `green`, `blue`, `red`, `yellow`, `magenta`, `cyan`, `orange`, `pink`, `gray`, `darkGray`, `lightGray`.

## Problema 3

Crear una aplicación que dibuje 255 líneas creando un color distinto para cada una de ellas:



```
import java.awt.BorderLayout;  
import java.awt.Color;  
import java.awt.EventQueue;  
import java.awt.Graphics;
```

```
import javax.swing.JFrame;  
import javax.swing.JPanel;  
import javax.swing.border.EmptyBorder;
```

```
public class Grafico1 extends JFrame {  
  
    private JPanel contentPane;  
  
    /**  
     * Launch the application.  
     */  
    public static void main(String[] args) {  
        EventQueue.invokeLater(new Runnable() {  
            public void run() {  
                try {  
                    Grafico1 frame = new Grafico1();  
                    frame.setVisible(true);  
                } catch (Exception e) {  
                    e.printStackTrace();  
                }  
            }  
        });  
    }  
}
```

```
/**
 * Create the frame.
 */
public Grafico1() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 300);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    contentPane.setLayout(new BorderLayout(0, 0));
    setContentPane(contentPane);
    setBounds(0,0,800,255);
}
```

```
public void paint (Graphics g)
{
    super.paint(g);
    int fila = 0;
    for (int rojo = 0 ; rojo <= 255 ; rojo++)
    {
        Color col = new Color (rojo, 0, 0);
        g.setColor (col);
        g.drawLine (0, fila, 800, fila);
        fila++;
    }
}
}
```



Dentro de un for creamos objetos de la clase Color y fijamos el color de la línea seguidamente (con esto logramos un degradé del negro al rojo):

```
int fila = 0;
for (int rojo = 0 ; rojo <= 255 ; rojo++)
{
    Color col = new Color (rojo, 0, 0);
    g.setColor (col);
    g.drawLine (0, fila, 800, fila);
    fila++;
}
```

## Presentación de imágenes

Java permite incorporar imágenes de tipo **GIF** y **JPEG** definidas en ficheros. Se dispone para ello de la **clase java.awt.Image**. Para cargar una imagen hay que indicar la localización del archivo y cargarlo mediante el **método getImage()**. Este método existe en las **clases java.awt.Toolkit**.

Entonces, para cargar una imagen hay que comenzar creando un objeto (o una referencia) Image y llamar al método getImage() (de Toolkit); Una vez cargada la imagen, hay que representarla, para lo cual se redefine el método paint() para llamar al método drawImage() de la clase Graphics. Los objetos Graphics pueden mostrar imágenes a través del método: drawImage(). Dicho método admite varias formas, aunque casi siempre hay que incluir el nombre del objeto imagen creado.

## **Clase Image**

Una imagen es un objeto gráfico rectangular compuesto por pixels coloreados. Cada pixel en una imagen describe un color de una particular localización de la imagen.

A continuación, algunos métodos de la clase Image:

La clase Graphics provee el método drawImage() para dibujar imagenes; este método admite varias formas:

- drawImage (Image i, int x, int y, ImageObserver o)
- drawImage (Image i,int x,int y,int width,int height,ImageObserver o)



## **Problema 4**

Crear una aplicación que muestre un archivo jpg, png, etc.. dentro de un JFrame.

Después de crear el proyecto debemos disponer un archivo en la carpeta raíz del proyecto (el archivo debe llamarse imagen1.jpg)

```
import java.awt.BorderLayout;  
import java.awt.EventQueue;  
import java.awt.Graphics;  
import java.awt.Image;  
import java.awt.Toolkit;
```

```
import javax.swing.JFrame;  
import javax.swing.JPanel;  
import javax.swing.border.EmptyBorder;
```

```
public class Grafico1 extends JFrame {

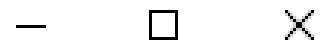
    private JPanel contentPane;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Grafico1 frame = new Grafico1();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

```
/**
 * Create the frame.
 */
public Grafico1() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 300);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    contentPane.setLayout(new BorderLayout(0, 0));
    setContentPane(contentPane);
    setBounds(0,0,800,600);
}
```

```
public void paint (Graphics g)
{
    super.paint(g);
    Toolkit t = Toolkit.getDefaultToolkit ();
    Image imagen = t.getImage ("imagen1.jpg");
    g.drawImage (imagen, 0, 0, this);
}

}
```



Creamos un objeto de la clase Toolkit llamando al método estático de la misma clase:

```
Toolkit t = Toolkit.getDefaultToolkit ();
```

Creamos un objeto de la clase Image llamando al método getImage de la clase Toolkit pasando como parámetro el archivo con la imagen:

```
Image imagen = t.getImage ("imagen1.jpg");
```

Por último llamamos al método drawImage con la referencia al objeto de tipo Image, la columna, la fila y la referencia al JFrame donde debe dibujarse:

```
g.drawImage (imagen, 0, 0, this);
```

## Método repaint()

Este es el método que con más frecuencia es llamado por el programador. El método repaint() llama lo antes posible al método paint() del componente.

El método repaint() puede ser:

```
repaint()
```

```
repaint(int x, int y, int w, int h)
```

Las segunda forma permiten definir una zona rectangular de la ventana a la que aplicar el método.



## **Problema 5**

Crear una aplicación que muestre un círculo en medio de la pantalla y mediante dos botones permitir que se desplace a izquierda o derecha.

```
import java.awt.Color;  
import java.awt.EventQueue;  
import java.awt.Graphics;
```

```
import javax.swing.JFrame;  
import javax.swing.JPanel;  
import javax.swing.border.EmptyBorder;  
import javax.swing.JButton;  
import java.awt.event.ActionListener;  
import java.awt.event.ActionEvent;
```

```
public class Grafico1 extends JFrame {

    private JPanel contentPane;

    /**
     * Launch the application.
     */

    private int columna;

    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Grafico1 frame = new Grafico1();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

```
/**
 * Create the frame.
 */
public Grafico1() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 300);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(null);

    JButton bi = new JButton("Izquierda");
    bi.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
            columna=columna-10;
            repaint();
        }
    });
    bi.setBounds(105, 482, 89, 23);
    contentPane.add(bi);
}
```

```

JButton bd = new JButton("Derecha");
    bd.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            columna=columna+10;
            repaint();
        }
    });
bd.setBounds(556, 482, 89, 23);
contentPane.add(bd);
setBounds(0,0,800,600);
columna=400;
}

```

```

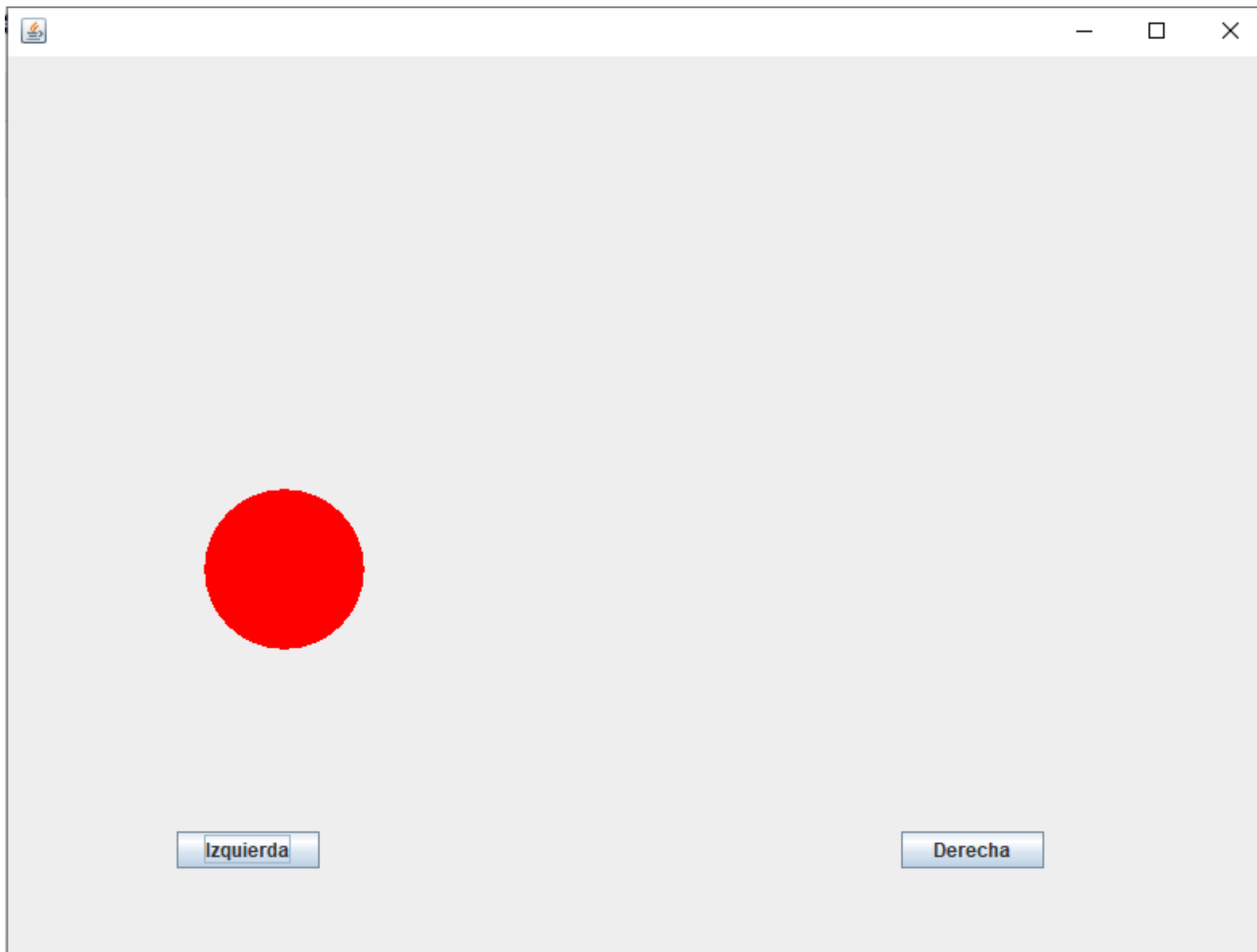
public void paint (Graphics g)
{
    super.paint(g);
    g.setColor (Color.red);
    g.fillOval (columna, 300, 100, 100);
}

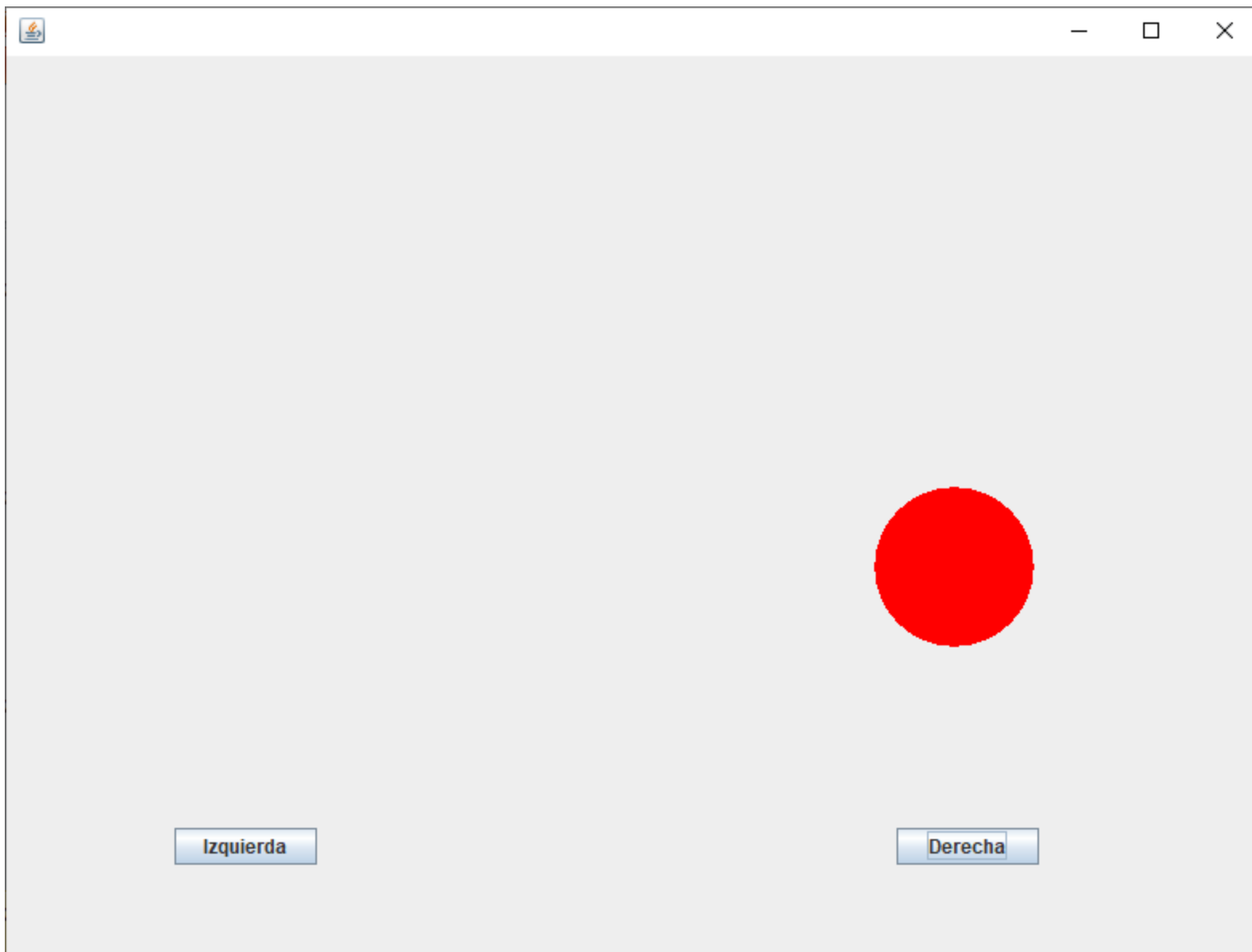
```

```

}

```





Definimos un atributo columna:

```
private int columna;
```

Cuando se presiona el botón (bi) restamos 10 al atributo columna y pedimos que se ejecute el método paint (esto último llamando al método repaint()), el método repaint borra todo lo dibujado dentro del JFrame y llama al paint:

```
bi.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent arg0) {  
        columna=columna-10;  
        repaint();  
    }  
});
```



El método paint dibuja un círculo utilizando como posición el valor del atributo columna:

```
public void paint (Graphics g)
{
    super.paint(g);
    g.setColor (Color.red);
    g.fillOval (columna, 300, 100, 100);
}
```

## Problema 6

Se debe desarrollar una pantalla para configurar ciertas características de un procesador de texto.

Debe aparecer y poder seleccionarse los márgenes superior e inferior de la página.

Los márgenes pueden ir en el rango de 0 a 10. Desplazar las líneas a medida que modificamos los márgenes.

Por otro lado tenemos la orientación de página. La misma se administra a través de un JComboBox que tiene dos valores posibles (Horizontal y Vertical). Cuando está seleccionado en el JComboBox el String Horizontal dibujar un rectángulo con base mayor a la altura, y cuando está seleccionado el String Vertical dibujar un rectángulo con una base menor.

Cuando se presiona el botón inicializar la configuración de márgenes se inicializan con 0 y se selecciona orientación horizontal.

```
import java.awt.Color;  
import java.awt.EventQueue;  
import java.awt.Graphics;
```

```
import javax.swing.JFrame;  
import javax.swing.JPanel;  
import javax.swing.border.EmptyBorder;  
import javax.swing.JSpinner;  
import javax.swing.JLabel;  
import javax.swing.JComboBox;  
import javax.swing.DefaultComboBoxModel;  
import javax.swing.JButton;  
import javax.swing.SpinnerNumberModel;  
import javax.swing.event.ChangeListener;  
import javax.swing.event.ChangeEvent;  
import java.awt.event.ItemListener;  
import java.awt.event.ItemEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.ActionEvent;
```

```

public class ProcesadorTexto extends JFrame {

    private JPanel contentPane;
    private JSpinner sp1;
    private JSpinner sp2;
    private JComboBox comboBox;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    ProcesadorTexto frame = new ProcesadorTexto();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}

```

```
/**  
 * Create the frame.  
 */  
public ProcesadorTexto() {  
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    setBounds(100, 100, 573, 481);  
    contentPane = new JPanel();  
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));  
    setContentPane(contentPane);  
    contentPane.setLayout(null);  
}
```

```
sp1 = new JSpinner();
    sp1.addChangeListener(new ChangeListener() {
        public void stateChanged(ChangeEvent arg0) {
            repaint();
        }
    });
sp1.setModel(new SpinnerNumberModel(0, 0, 10, 1));
sp1.setBounds(162, 51, 55, 28);
contentPane.add(sp1);
```

```
sp2 = new JSpinner();
sp2.addChangeListener(new ChangeListener() {
    public void stateChanged(ChangeEvent e) {
        repaint();
    }
});
sp2.setModel(new SpinnerNumberModel(0, 0, 10, 1));
sp2.setBounds(162, 150, 55, 28);
contentPane.add(sp2);
```

```
JLabel lblMargenInferior = new JLabel("Margen inferior");  
    lblMargenInferior.setBounds(162, 26, 109, 14);  
    contentPane.add(lblMargenInferior);
```

```
JLabel lblMargenSuperior = new JLabel("Margen superior");  
    lblMargenSuperior.setBounds(162, 127, 109, 14);  
    contentPane.add(lblMargenSuperior);
```

```
JLabel lblHoja = new JLabel("Hoja");  
    lblHoja.setBounds(46, 26, 46, 14);  
    contentPane.add(lblHoja);
```

```
comboBox = new JComboBox();  
comboBox.addItemListener(new ItemListener() {  
    public void itemStateChanged(ItemEvent arg0) {  
        repaint();  
    }  
});  
comboBox.setModel(new DefaultComboBoxModel(new String[] {"Horizontal",  
"Vertical"}));  
comboBox.setBounds(321, 55, 196, 20);  
contentPane.add(comboBox);
```

```
JLabel lblHorientacinDePgina = new JLabel("Horientaciu00F3n de  
pu00E1gina.");
```

```
    lblHorientacinDePgina.setBounds(321, 26, 203, 14);  
    contentPane.add(lblHorientacinDePgina);
```

```
  
    JButton btnInicializar = new JButton("Inicializar");  
    btnInicializar.addActionListener(new ActionListener() {  
        public void actionPerformed(ActionEvent arg0) {  
            sp1.setValue(0);  
            sp2.setValue(0);  
            comboBox.setSelectedIndex(0);  
            repaint();  
        }  
    });  
    btnInicializar.setBounds(45, 247, 165, 23);  
    contentPane.add(btnInicializar);  
}
```



```
public void paint(Graphics g)
{
    super.paint(g);
    g.setColor(Color.blue);
    g.drawRect(30,80,100,140);
    int ms=Integer.parseInt(sp1.getValue().toString());
    int mi=Integer.parseInt(sp2.getValue().toString());
    g.setColor(Color.red);
    g.drawLine(30,80+ms,130,80+ms);
    g.drawLine(30,220-mi,130,220-mi);
    String direccion=(String)comboBox.getSelectedItem();
    if (direccion.equals("Horizontal"))
        g.drawRect(320,120,200,100 );
    else
        g.drawRect(320,120,100,200 );
}
}
```



Hoja

Margen inferior

Margen superior

Horientaciun de pagina.

Horizontal



Inicializar

## Explicación del código.

Para el evento `stateChanged` de los controles `JSpinner` se debe llamar al método `repaint()` para que se grafique nuevamente las líneas de márgenes:

```
sp1.addChangeListener(new ChangeListener() {  
    public void stateChanged(ChangeEvent arg0) {  
        repaint();  
    }  
});
```

```
sp2.addChangeListener(new ChangeListener() {  
    public void stateChanged(ChangeEvent e) {  
        repaint();  
    }  
});
```

En el método paint dibujamos primero un rectángulo de color azul que representa la hoja:

```
g.setColor(Color.blue);  
g.drawRect(30,80,100,140);
```

Extraemos los valores seleccionados de cada control JSpinner y los convertimos a tipo entero:

```
int ms=Integer.parseInt(sp1.getValue().toString());  
int mi=Integer.parseInt(sp2.getValue().toString());
```

Activamos el color rojo y dibujamos las dos líneas, la superior coincide con el comienzo del rectángulo (sumamos tantos pixeles en la fila como lo indica el primer JSpinner):

```
g.setColor(Color.red);  
g.drawLine(30,80+ms,130,80+ms);
```

La segunda línea le restamos el valor del JSpinner:

```
g.drawLine(30,220-mi,130,220-mi);
```

Para saber la orientación de la hoja debemos extraer el valor seleccionado del JComboBox y mediante un if verificar si el String seleccionado es "Horizontal":

```
String direccion=(String)comboBox.getSelectedItem();
if (direccion.equals("Horizontal"))
    g.drawRect(320,120,200,100);
else
    g.drawRect(320,120,100,200);
```

Por último cuando se presiona el botón inicializar procedemos a fijar nuevos valores a los JSpinner y al JComboBox (luego redibujamos):

```
btnInicializar.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        sp1.setValue(0);
        sp2.setValue(0);
        comboBox.setSelectedIndex(0);
        repaint();
    }
});
```

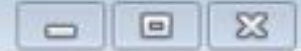
## **Problemas propuestos**

Confeccionar un programa que permita configurar las características del mouse.

Por un lado debemos seleccionar la velocidad de desplazamiento de la flecha del mouse. Disponer un JSpinner para poder seleccionarse los valores 0,25,50,75 y 100.

Por otro lado debemos poder seleccionar cual de los dos botones del mouse será el principal, tenemos para esta función un JComboBox con dos opciones: izquierdo o derecho.

Cuando se selecciona el botón (cambio en el JComboBox) actualizar el gráfico mostrando el botón del mouse seleccionado (graficar en el método paint el mouse en pantalla)



velocidad de desplazamiento


Selección del botón



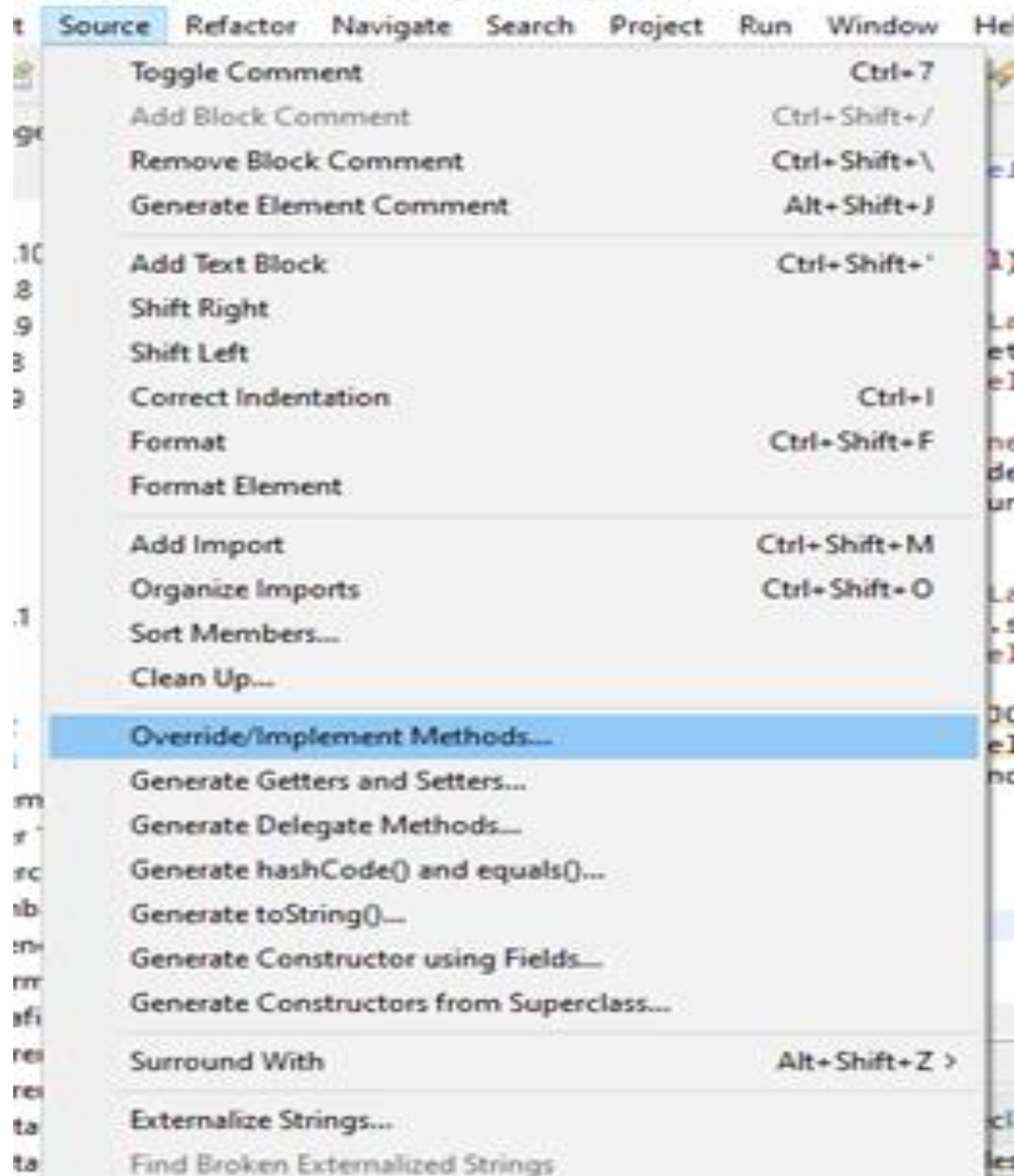
Velocidad de desplazamiento

Selección del botón





## Override/Implement Methods

Enter method name, prefix or pattern (\*, ? or camel case)

paint

Select methods to override or implement:



Select All

Deselect All

- ☐ JPanel
- ☒ JComponent
  - ☒ paint(Graphics)
  - ☐ paintBorder(Graphics)
  - ☐ paintChildren(Graphics)
  - ☐ paintComponent(Graphics)
  - ☐ paintImmediately(int, int, int, int)
  - ☐ paintImmediately(Rectangle)
- ☒ Container
  - ☐ paintComponents(Graphics)
- ☒ Component

Insertion point:

After 'Mouse()'

☐ Generate method comments

The format of the method stubs may be configured on the [Code Templates](#) preference page.

1 of 335 selected.



OK

Cancel

\*Mouse.java X

```
1 import javax.swing.JPanel;
2 import javax.swing.JLabel;
3 import javax.swing.JSpinner;
4 import javax.swing.SpinnerNumberModel;
5 import javax.swing.JComboBox;
6
7 import java.awt.Color;
8 import java.awt.Graphics;
9
10 import javax.swing.DefaultComboBoxModel;
11
12 public class Mouse extends JPanel {
13     private JComboBox combol;
14
15     /**
16      * Create the panel.
17      */
18     public Mouse() {
19         setLayout(null);
20
21         JLabel lblNewLabel = new JLabel("Velocidad de desplazamiento");
22         lblNewLabel.setBounds(32, 33, 150, 14);
23         add(lblNewLabel);
24
25         JSpinner spinner = new JSpinner();
26         spinner.setModel(new SpinnerNumberModel(0, 0, 100, 25));
27         spinner.setBounds(54, 58, 98, 20);
28         add(spinner);
29
30         JLabel lblNewLabel_1 = new JLabel("Seleccion del boton");
31         lblNewLabel_1.setBounds(245, 33, 125, 14);
32         add(lblNewLabel_1);
33     }
```

```

34     combo1 = new JComboBox();
35     combo1.setModel(new DefaultComboBoxModel(new String[] {"Izquierdo", "Derecho"}));
36     combo1.setBounds(255, 57, 91, 22);
37     add(combo1);
38
39 }
40
41 @Override
42 public void paint(Graphics g) {
43     // TODO Auto-generated method stub
44     super.paint(g);
45
46     g.setColor(Color.black);
47     g.drawRect(250, 120, 120, 230);
48     if (combo1.getSelectedItem().toString().equals("Izquierdo")) {
49         g.setColor(Color.red);
50         g.fillOval(256, 300, 60, 60);
51         g.setColor(Color.red);
52         g.fillOval(256, 300, 60, 60);
53
54     } else {
55         g.fillRect(256, 300, 60, 60);
56         g.setColor(Color.red);
57         g.fillOval(256, 300, 60, 60);
58     }
59 }
60 }
61

```

## 58 - Gráficos estadísticos

El objetivo de este concepto es la implementación de algoritmos para mostrar gráficos estadísticos.

### **Problema 1**

Crear una aplicación que solicite introducir tres valores por teclado que representen las cantidades de votos obtenidas por tres partidos políticos. Luego mostrar un gráfico de tarta:

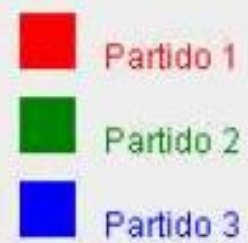
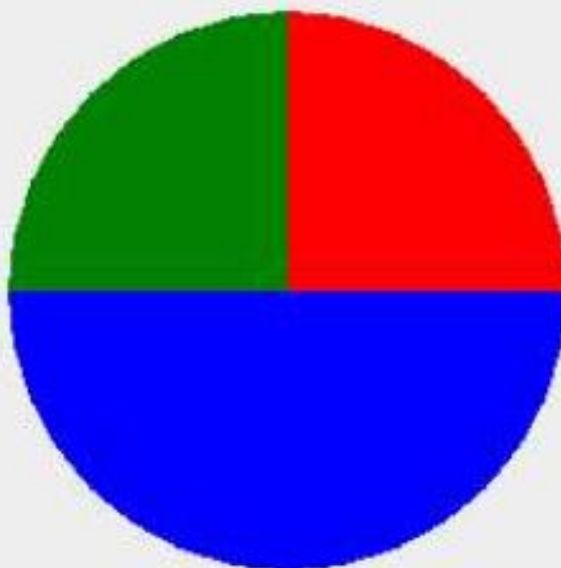


Partido 1:

Partido 2:

Partido 3:

Graficar



```
import java.awt.Color;  
import java.awt.EventQueue;  
import java.awt.Graphics;
```

```
import javax.swing.JFrame;  
import javax.swing.JPanel;  
import javax.swing.border.EmptyBorder;  
import javax.swing.JLabel;  
import javax.swing.JTextField;  
import javax.swing.JButton;  
import java.awt.event.ActionListener;  
import java.awt.event.ActionEvent;
```

```
public class GraficoTarta extends JFrame {

    private JPanel contentPane;
    private JTextField tf1;
    private JTextField tf2;
    private JTextField tf3;
    private boolean bandera=false;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    GraficoTarta frame = new GraficoTarta();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```



```
/**
 * Create the frame.
 */
public GraficoTarta() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 800, 600);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(null);

    JLabel lblPartido = new JLabel("Partido 1:");
    lblPartido.setBounds(46, 39, 61, 14);
    contentPane.add(lblPartido);
```

```
public class GraficoTarta extends JFrame {

    private JPanel contentPane;
    private JTextField tf1;
    private JTextField tf2;
    private JTextField tf3;
    private boolean bandera=false;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    GraficoTarta frame = new GraficoTarta();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

```
JLabel lblPartido_1 = new JLabel("Partido 2:");  
lblPartido_1.setBounds(46, 69, 61, 14);  
contentPane.add(lblPartido_1);
```

```
JLabel lblPartido_2 = new JLabel("Partido 3:");  
lblPartido_2.setBounds(46, 103, 61, 14);  
contentPane.add(lblPartido_2);
```

```
tf1 = new JTextField();  
tf1.setBounds(117, 36, 86, 20);  
contentPane.add(tf1);  
tf1.setColumns(10);
```

```
tf2 = new JTextField();  
tf2.setBounds(117, 66, 86, 20);  
contentPane.add(tf2);  
tf2.setColumns(10);
```

```
tf3 = new JTextField();  
tf3.setBounds(117, 97, 86, 20);  
contentPane.add(tf3);  
tf3.setColumns(10);
```

```

JButton btnGraficar = new JButton("Graficar");
    btnGraficar.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
            bandera=true;
            repaint();
        }
    });
    btnGraficar.setBounds(45, 138, 107, 37);
    contentPane.add(btnGraficar);
}

```

```
public void paint(Graphics g)
{
super.paint(g);
    if (bandera==true)
    {
        String s1=tf1.getText();
        String s2=tf2.getText();
        String s3=tf3.getText();
        int v1=Integer.parseInt(s1);
        int v2=Integer.parseInt(s2);
        int v3=Integer.parseInt(s3);
        int suma=v1+v2+v3;
        int grados1=v1*360/suma;
        int grados2=v2*360/suma;
        int grados3=v3*360/suma;

        g.setColor(new Color(255,0,0));
        g.fillArc(50,250,200,200,0,grados1);
        g.fillRect(370,250,20,20);
        g.drawString("Partido 1", 400, 270);
```

```
g.setColor(new Color(0,128,0));  
g.fillArc(50,250,200,200,grados1,grados2);  
g.fillRect(370,280,20,20);  
g.drawString("Partido 2", 400, 300);
```

```
g.setColor(new Color(0,0,255));  
g.fillArc(50,250,200,200,grados1+grados2,grados3);  
g.fillRect(370,310,20,20);  
g.drawString("Partido 3", 400, 330);
```

```
}
```

```
}
```

```
}
```

Disponemos un if en el método paint para controlar que se haya presionado el botón graficar:

```
public void paint(Graphics g)
{
    super.paint(g);
    if (bandera==true)
    {
```

El atributo bandera se inicializa cuando se define con el valor false, esto hace que cuando se ejecute por primera vez el método paint no ingrese al if:

```
private boolean bandera=false;
```

Se definen tres objetos de la clase JTextField para ingresar los tres valores por teclado:

```
private JTextField tf1;
private JTextField tf2;
private JTextField tf3;
```

Cuando se presiona el botón se cambia el estado del atributo bandera por el valor true y se llama al método repaint (recordemos que este método borra el JFrame y llama al método paint para que ahora dibuje el gráfico de tarta):

```
btnGraficar.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent arg0) {  
        bandera=true;  
        repaint();  
    }  
});
```



Lo primero que hacemos para graficar la tarta es rescatar los tres valores introducidos en los controles JTextField:

```
if (bandera==true)
{
    String s1=tf1.getText();
    String s2=tf2.getText();
    String s3=tf3.getText();
```

Los convertimos a tipo de dato entero:

```
int v1=Integer.parseInt(s1);
int v2=Integer.parseInt(s2);
int v3=Integer.parseInt(s3);
```

Sumamos los tres valores:

```
int suma=v1+v2+v3;
```

Seguidamente calculamos los grados que le corresponde a cada trozo de tarta (teniendo en cuenta que tenemos 360 grados para repartir):

Cada trozo de tarta lo obtenemos mediante la ecuación:

$\text{tamaño trozo} = \text{cantidad de votos del partido} / \text{total de votos} * 360$

si observamos la ecuación podemos imaginar que la división:

$\text{cantidad de votos del partido} / \text{total de votos}$

generará un valor menor a uno (salvo que el partido haya obtenido todos los votos de la elección en cuyo caso la división genera el valor uno)

Esta división nos genera el porcentaje de votos que le corresponde al partido y luego dicho porcentaje lo multiplicamos por la cantidad de grados a repartir (que son 360 grados)

Luego como la división generará un valor menor a uno y al tratarse de dos variables enteras el resultado será cero. Para evitar este problema procedemos primero a multiplicar por 360 y luego dividir por la variable suma:

```
int grados1=v1*360/suma;  
int grados2=v2*360/suma;  
int grados3=v3*360/suma;
```

Si quisiéramos primero dividir y luego multiplicar por 360 debemos proceder a anteceder a cada operación el tipo de resultado a obtener:

```
int grados1=(int)((float)v1/suma*360);  
int grados2=(int)((float)v2/suma*360);  
int grados3=(int)((float)v3/suma*360);
```

Como podemos ver es más complicado si queremos primero efectuar la división y luego el producto.

Procedemos ahora a graficar los trozos de tarta y leyenda con el valor introducido (activamos el color rojo y mediante el método fillArc creamos un trozo de tarta que se inicia en el grado 0 y avanza tantos grados como indica la variable grados1. Luego mediante el método drawString mostramos la leyenda del partido respectivo con un cuadradito también de color rojo ):

```
g.setColor(new Color(255,0,0));  
g.fillArc(50,250,200,200,0,grados1);  
g.fillRect(370,250,20,20);  
g.drawString("Partido 1", 400, 270);
```

El segundo trozo comienza en grados1 y avanza tantos grados como lo indica la variable grados2. Activamos el color verde para diferenciar del trozo anterior:

```
g.setColor(new Color(0,128,0));  
g.fillArc(50,250,200,200,grados1,grados2);  
g.fillRect(370,280,20,20);  
g.drawString("Partido 2", 400, 300);
```

El último trozo lo graficamos a partir de la suma de  $\text{grados1} + \text{grados2}$  y avanzamos tantos grados como indique la variable  $\text{grados3}$ :

```
g.setColor(new Color(0,0,255));  
g.fillArc(50,250,200,200,grados1+grados2,grados3);  
g.fillRect(370,310,20,20);  
g.drawString("Partido 1", 400, 330);
```

## Problema 2

Crear una aplicación que solicite introducir tres valores por teclado que representen las cantidades de votos obtenidas por tres partidos políticos. Luego mostrar un gráfico de barras horizontales:



Partido 1:

Partido 2:

Partido 3:

**Graficar**

Partido 1



Partido 2



Partido 3



```
import java.awt.Color;  
import java.awt.EventQueue;  
import java.awt.Graphics;
```

```
import javax.swing.JFrame;  
import javax.swing.JPanel;  
import javax.swing.border.EmptyBorder;  
import javax.swing.JLabel;  
import javax.swing.JTextField;  
import javax.swing.JButton;  
import java.awt.event.ActionListener;  
import java.awt.event.ActionEvent;
```



```

public class GraficoBarraHorizontal extends JFrame {

    private JPanel contentPane;
    private JTextField tf3;
    private JTextField tf1;
    private JTextField tf2;
    private boolean bandera=false;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    GraficoBarraHorizontal frame = new GraficoBarraHorizontal();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}

```

```
/**
 * Create the frame.
 */
public GraficoBarraHorizontal() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 800, 600);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(null);

    JLabel lblPartido = new JLabel("Partido 1:");
    lblPartido.setBounds(46, 39, 61, 14);
    contentPane.add(lblPartido);

    JLabel lblPartido_1 = new JLabel("Partido 2:");
    lblPartido_1.setBounds(46, 69, 61, 14);
    contentPane.add(lblPartido_1);

    JLabel lblPartido_2 = new JLabel("Partido 3:");
    lblPartido_2.setBounds(46, 103, 61, 14);
    contentPane.add(lblPartido_2);
}
```

```
tf1 = new JTextField();
    tf1.setBounds(117, 36, 86, 20);
    contentPane.add(tf1);
    tf1.setColumns(10);

    tf2 = new JTextField();
    tf2.setBounds(117, 66, 86, 20);
    contentPane.add(tf2);
    tf2.setColumns(10);

    tf3 = new JTextField();
    tf3.setBounds(117, 97, 86, 20);
    contentPane.add(tf3);
    tf3.setColumns(10);

    JButton btnGraficar = new JButton("Graficar");
    btnGraficar.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
            bandera=true;
            repaint();
        }
    });
    btnGraficar.setBounds(45, 138, 107, 37);
    contentPane.add(btnGraficar);
}
```

```
public void paint(Graphics g)
{
    super.paint(g);
    if (bandera==true)
    {
        String s1=tf1.getText();
        String s2=tf2.getText();
        String s3=tf3.getText();
        int v1=Integer.parseInt(s1);
        int v2=Integer.parseInt(s2);
        int v3=Integer.parseInt(s3);
        int mayor=retornarMayor(v1,v2,v3);

        int largo1=v1*400/mayor;
        int largo2=v2*400/mayor;
        int largo3=v3*400/mayor;

        g.setColor(new Color(255,0,0));
        g.fillRect(100,250,largo1,40);
        g.drawString("Partido 1", 10, 270);
```

```
g.setColor(new Color(0,128,0));  
g.fillRect(100,300,largo2,40);  
g.drawString("Partido 2", 10, 320);
```

```
g.setColor(new Color(0,0,255));  
g.fillRect(100,350,largo3,40);  
g.drawString("Partido 3", 10, 370);
```

```
}  
}
```

```
private int retornarMayor(int v1,int v2,int v3)
{
    if (v1>v2 && v1>v3)
        return v1;
    else
        if (v2>v3)
            return v2;
        else
            return v3;
}
}
```

La metodología es similar al problema anterior. Ahora no tenemos grados para repartir, por lo que implementaremos el siguiente algoritmo:

Consideraremos que el partido que obtuvo más votos le corresponde una barra de 400 píxeles de largo y los otros dos partidos se les entregará en forma proporcional.

Lo primero que hacemos es obtener el mayor de los tres valores introducidos por teclado, para eso implementamos un método privado que retorne el mayor de tres valores enteros:

```
private int retornarMayor(int v1,int v2,int v3)
{
    if (v1>v2 && v1>v3)
        return v1;
    else
        if (v2>v3)
            return v2;
        else
            return v3;
}
```

En el método paint llamamos al método retornarMayor:

```
int mayor=retornarMayor(v1,v2,v3);
```

La ecuación para obtener el largo de la barra será:

$\text{Largo} = \text{votos del partido} / \text{votos del partido con mas votos} * 400 \text{ píxeles.}$

Como podemos ver esta división generará un valor menor a uno salvo para el partido que tiene más votos (en este caso la división genera el valor 1) luego multiplicamos por 400.

Comenzamos a calcular el largo de cada rectángulo:

```
int largo1=v1*400/mayor;
```



Como podemos ver primero multiplicamos por 400 y lo dividimos por el mayor de los tres valores ingresados.

Nuevamente primero multiplicamos y luego dividimos con el objetivo que el resultado de la división no nos redondee a cero.

El primer trozo lo graficamos a partir de la columna 100, fila 250 y con un ancho indicado en la variable largo1 (el alto de la barra es de 40 píxeles):

```
g.setColor(new Color(255,0,0));  
g.fillRect(100,250,largo1,40);  
g.drawString("Partido 1", 10, 270);
```

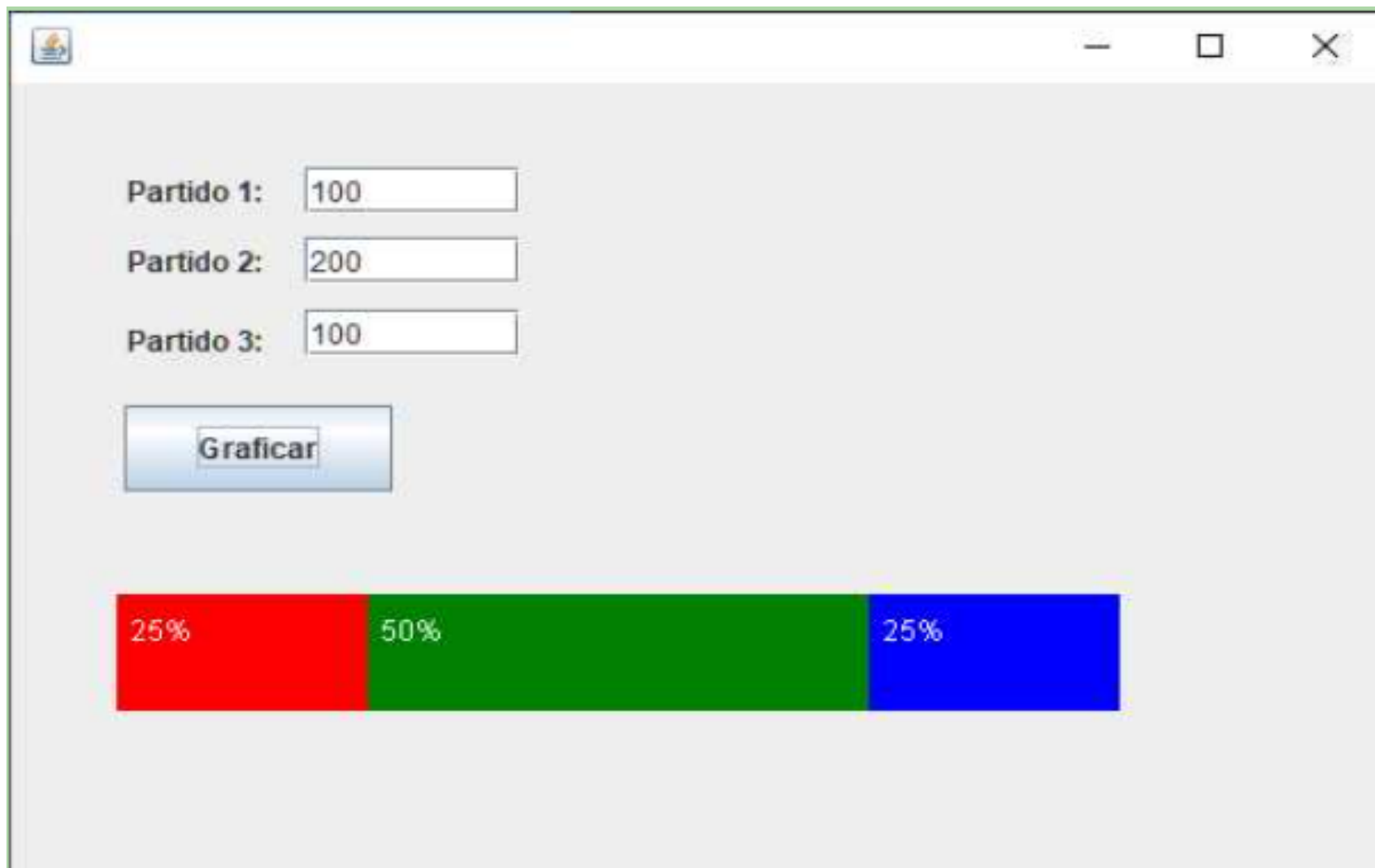
En forma similar graficamos los otros dos trozos de barra:

```
g.setColor(new Color(0,128,0));  
g.fillRect(100,300,largo2,40);  
g.drawString("Partido 2", 10, 320);
```

```
g.setColor(new Color(0,0,255));  
g.fillRect(100,350,largo3,40);  
g.drawString("Partido 3", 10, 370);
```

## Problema propuesto

Implementar un gráfico estadístico de tipo "Barra Porcentual":



```
import java.awt.Color;  
import java.awt.EventQueue;  
import java.awt.Graphics;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;
```

```
import javax.swing.JButton;  
import javax.swing.JFrame;  
import javax.swing.JLabel;  
import javax.swing.JPanel;  
import javax.swing.JTextField;  
import javax.swing.border.EmptyBorder;
```

```
public class BarraPorcentual extends JFrame {
```

```
    private JPanel contentPane;
```

```
    private JTextField tf1;
```

```
    private JTextField tf2;
```

```
    private JTextField tf3;
```

```
    private boolean bandera=false;
```

```
    /**
```

```
     * Launch the application.
```

```
     */
```

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

```
        EventQueue.invokeLater(new Runnable() {
```

```
            public void run() {
```

```
                try {
```

```
                    BarraPorcentual frame = new BarraPorcentual();
```

```
                    frame.setVisible(true);
```

```
                } catch (Exception e) {
```

```
                    e.printStackTrace();
```

```
                }
```

```
            }
```

```
        });
```

```
    }
```

```
/**
 * Create the frame.
 */
public BarraPorcentual()
{
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 800, 600);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(null);

    JLabel lblPartido = new JLabel("Partido 1:");
    lblPartido.setBounds(46, 39, 61, 14);
    contentPane.add(lblPartido);

    JLabel lblPartido_1 = new JLabel("Partido 2:");
    lblPartido_1.setBounds(46, 69, 61, 14);
    contentPane.add(lblPartido_1);

    JLabel lblPartido_2 = new JLabel("Partido 3:");
    lblPartido_2.setBounds(46, 103, 61, 14);
    contentPane.add(lblPartido_2);
}
```

```
tf1 = new JTextField();  
tf1.setBounds(117, 36, 86, 20);  
contentPane.add(tf1);  
tf1.setColumns(10);
```

```
tf2 = new JTextField();  
tf2.setBounds(117, 66, 86, 20);  
contentPane.add(tf2);  
tf2.setColumns(10);
```

```
tf3 = new JTextField();  
tf3.setBounds(117, 97, 86, 20);  
contentPane.add(tf3);  
tf3.setColumns(10);
```

```
JButton btnGraficar = new JButton("Graficar");  
btnGraficar.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent arg0) {  
        bandera=true;  
        repaint();  
    }  
});  
btnGraficar.setBounds(45, 138, 107, 37);  
contentPane.add(btnGraficar);
```

```
}
```

```
public void paint(Graphics g)
{
    super.paint(g);
    if (bandera==true)
    {
        String s1=tf1.getText();
        String s2=tf2.getText();
        String s3=tf3.getText();
        int v1=Integer.parseInt(s1);
        int v2=Integer.parseInt(s2);
        int v3=Integer.parseInt(s3);
        int suma=v1+v2+v3;
        int largo1=v1*400/suma;
        int largo2=v2*400/suma;
        int largo3=v3*400/suma;
        int porc1=v1*100/suma;
        int porc2=v2*100/suma;
        int porc3=v3*100/suma;

        g.setColor(new Color(255,0,0));
        g.fillRect(50,250,largo1,50);
        g.setColor(new Color(255,255,255));
        g.drawString(porc1+"%",55,270);
```



```
g.setColor(new Color(0,128,0));  
g.fillRect(50+largo1,250,largo2,50);  
g.setColor(new Color(255,255,255));  
g.drawString(porc2+"%",55+largo1,270);
```

```
g.setColor(new Color(0,0,255));  
g.fillRect(50+largo1+largo2,250,largo3,50);  
g.setColor(new Color(255,255,255));  
g.drawString(porc3+"%",55+largo1+largo2,270);
```

```
}
```

```
}
```

```
}
```