

## Curso de Java a distancia

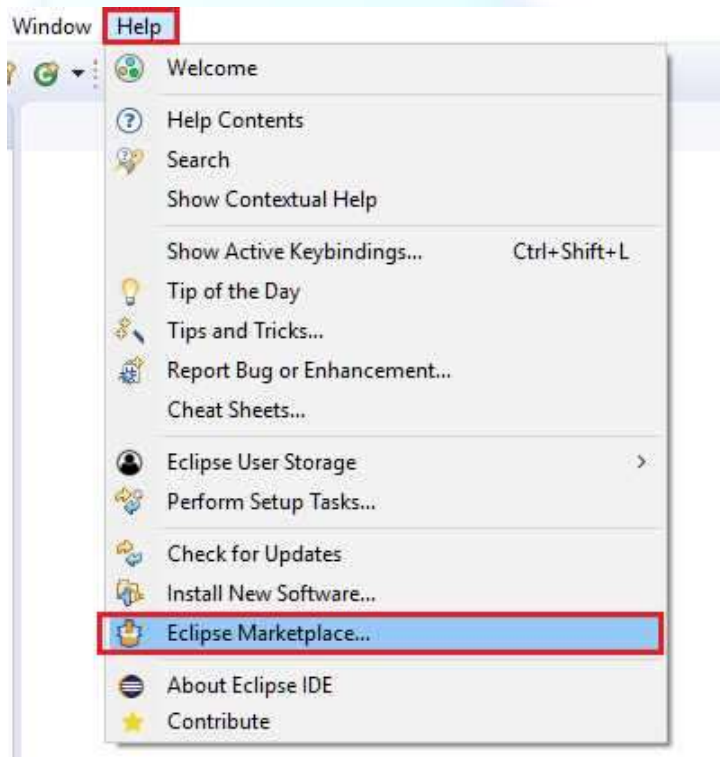
### Clase 19: Plug-in WindowBuilder para crear interfaces visuales.

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.

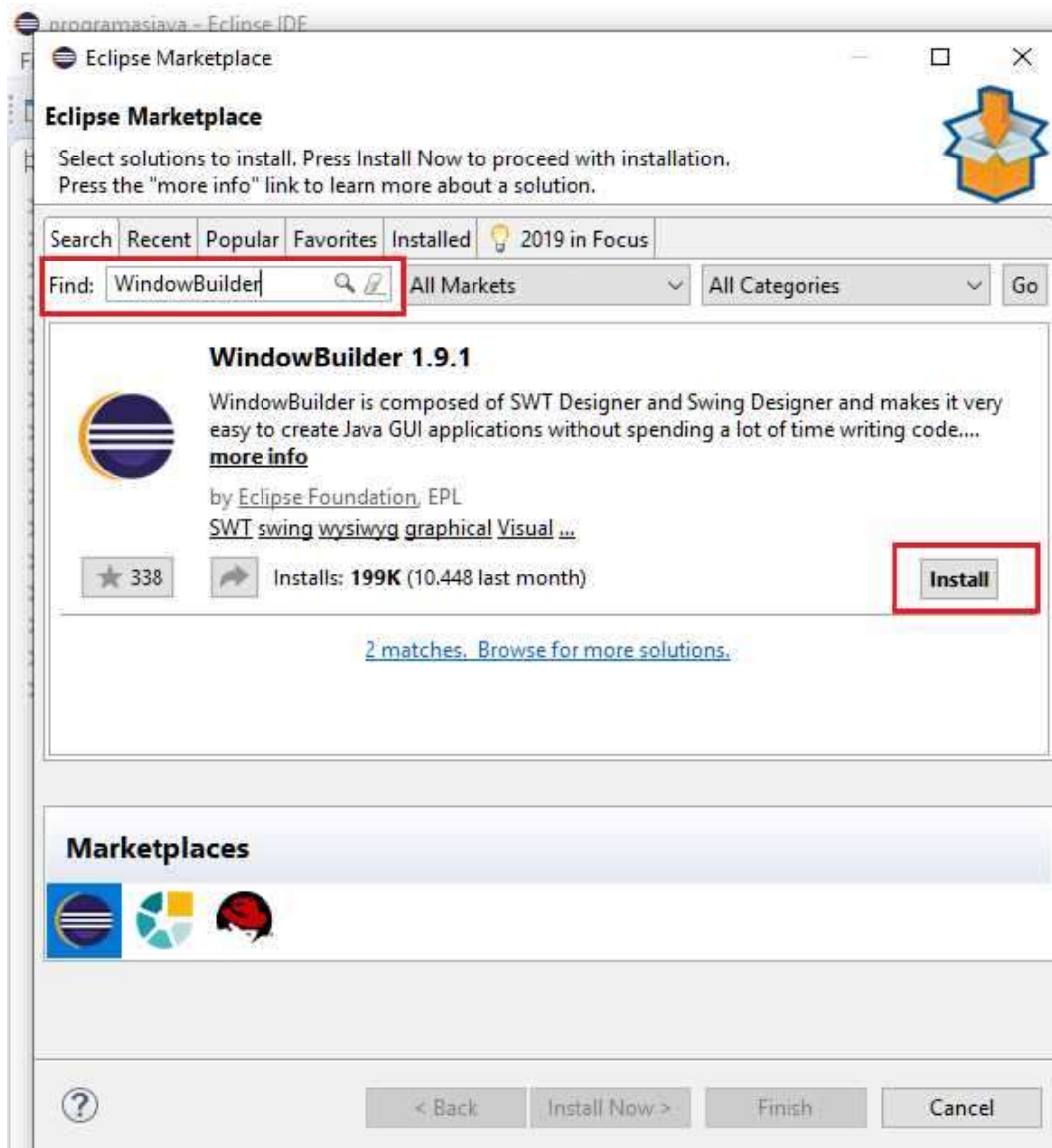
Las versiones Neon, Oxygen y la actual no trae instalado el plug-in WindowBuilder por defecto.

#### **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:



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

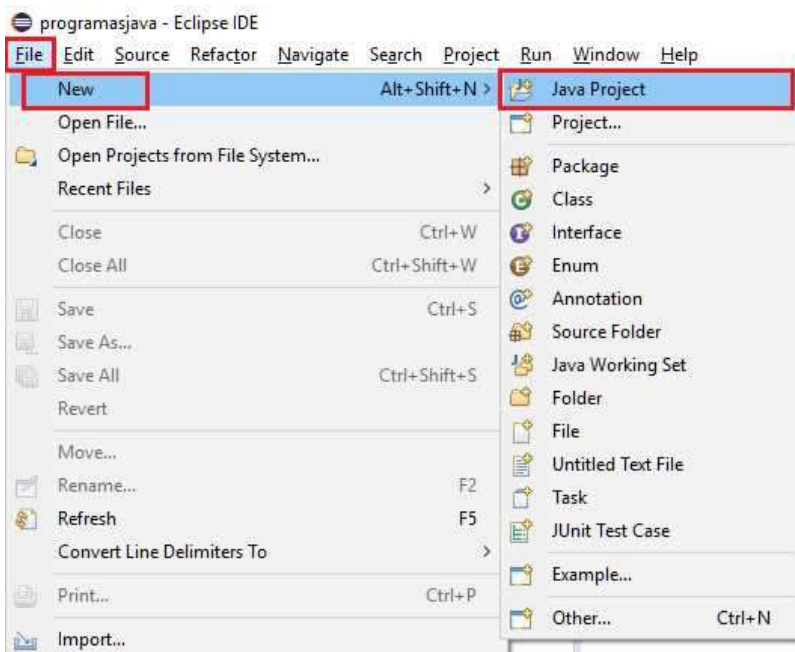
Con esto ya tenemos instalado el WindowBuilder para trabajar en los siguientes conceptos. Se debe reiniciar el entorno de Eclipse para que los cambios tengan efecto.

### Uso del WindowBuilder

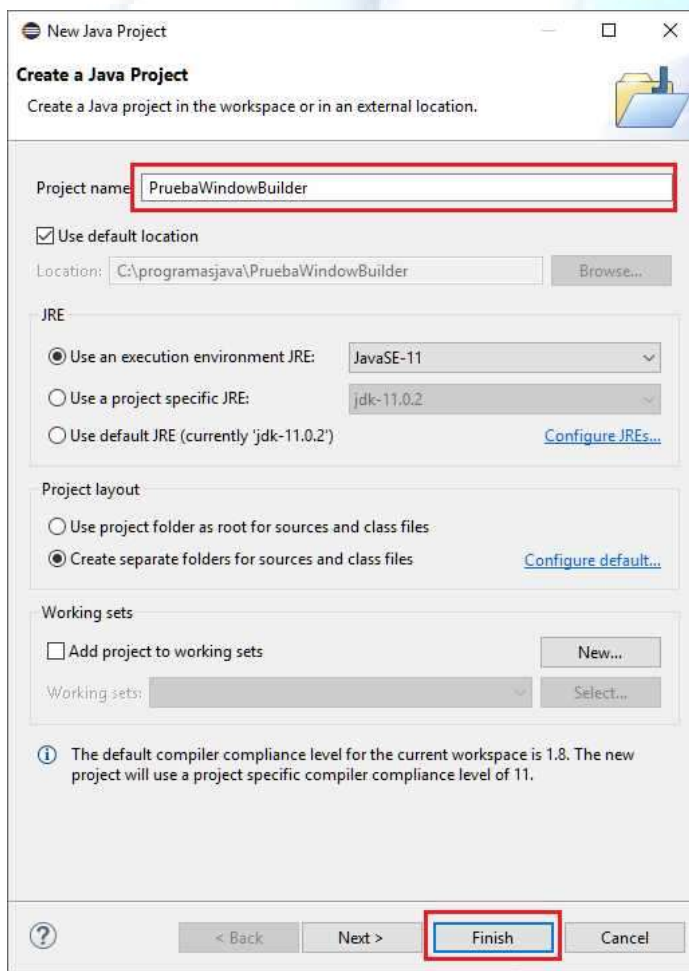
A medida que uno arrastra componentes visuales sobre un formulario se genera en forma automática el código Java, esto nos permite ser más productivos en el desarrollo de la interfaz de nuestra aplicación y nos ayuda a concentrarnos en la lógica de nuestro problema.

### Pasos para crear un JFrame con el WindowBuilder

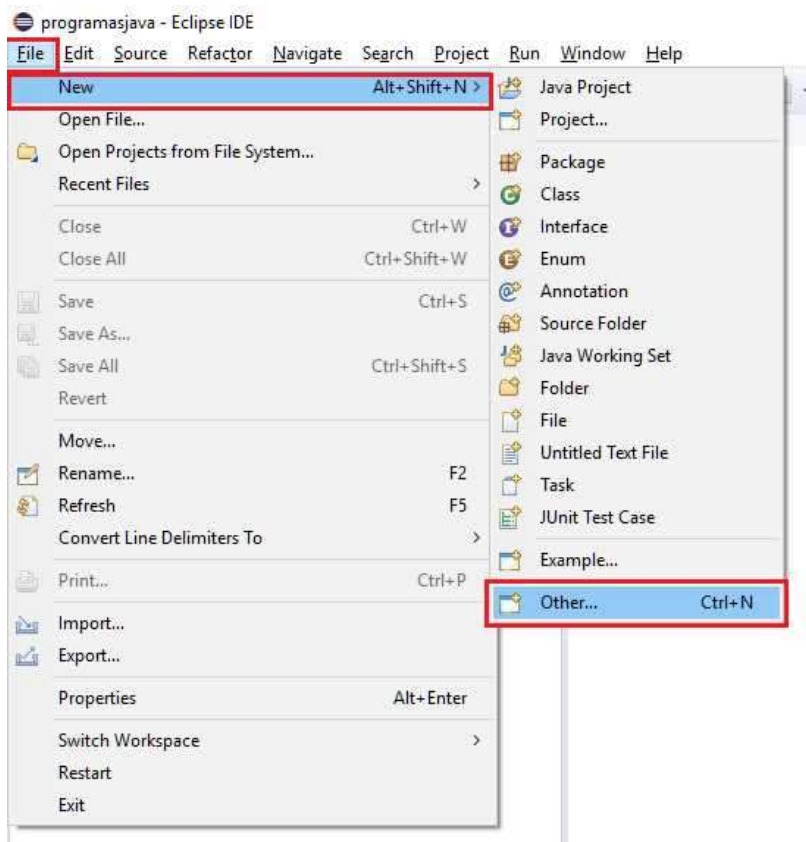
1 - Creación de un proyecto.



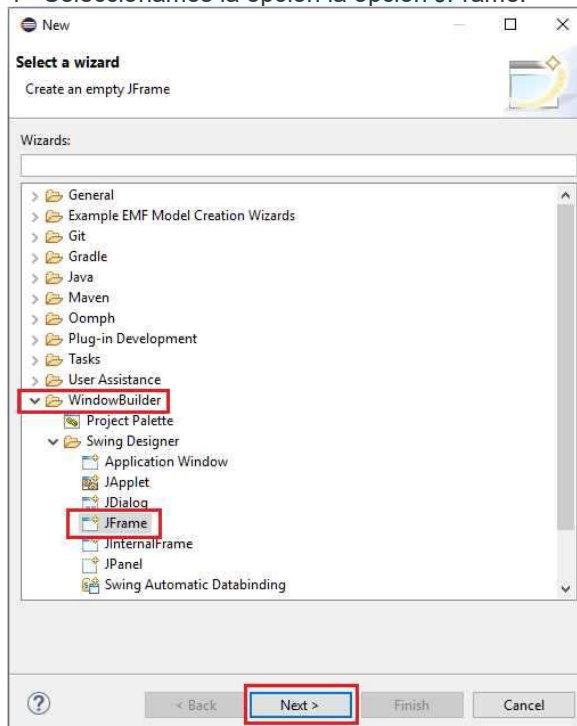
2 - Seleccionamos el nombre de nuestro proyecto (lo llamaremos PruebaWindowBuilder):



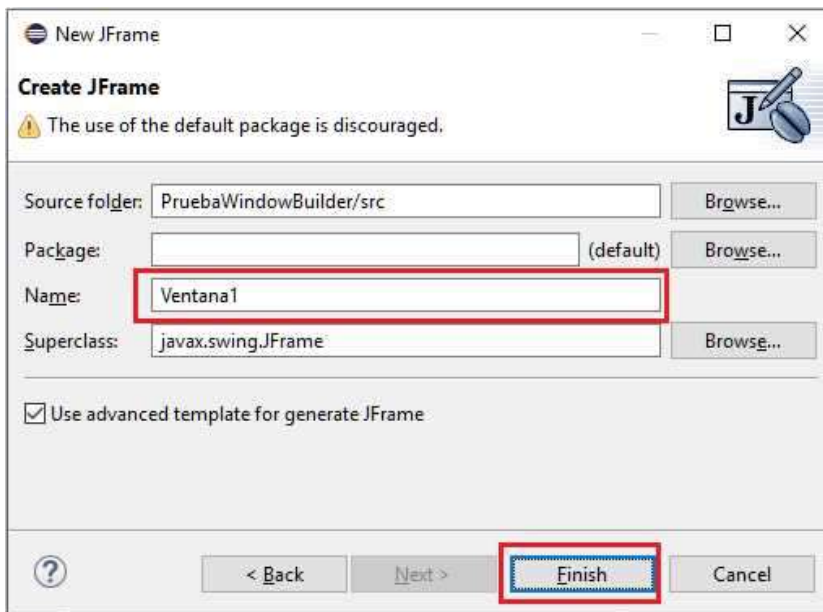
3 - Ahora seleccionamos la opción del menú File -> New -> Other ...



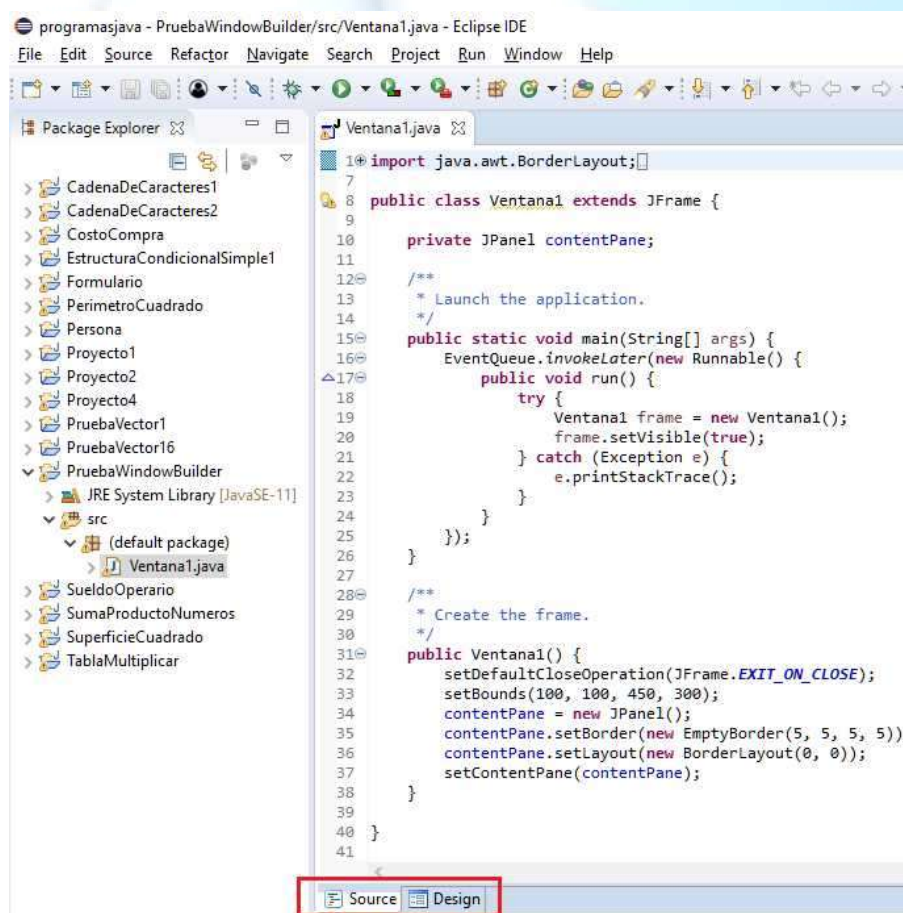
4 - Seleccionamos la opción la opción JFrame:



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

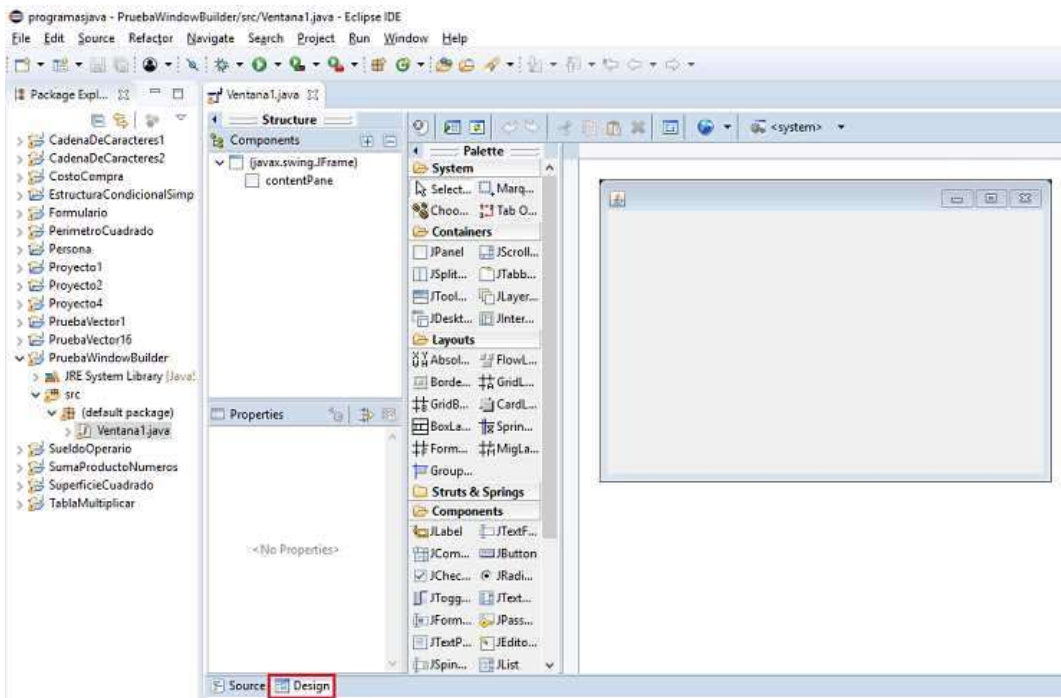


Tenemos en este momento nuestra aplicación mínima generada por el WindowBuilder. Podemos observar que en la parte inferior de la ventana central aparecen dos pestañas (Source y Design) estas dos pestañas nos permiten ver el código fuente de nuestro JFrame en vista de diseño o en vista de código Java:

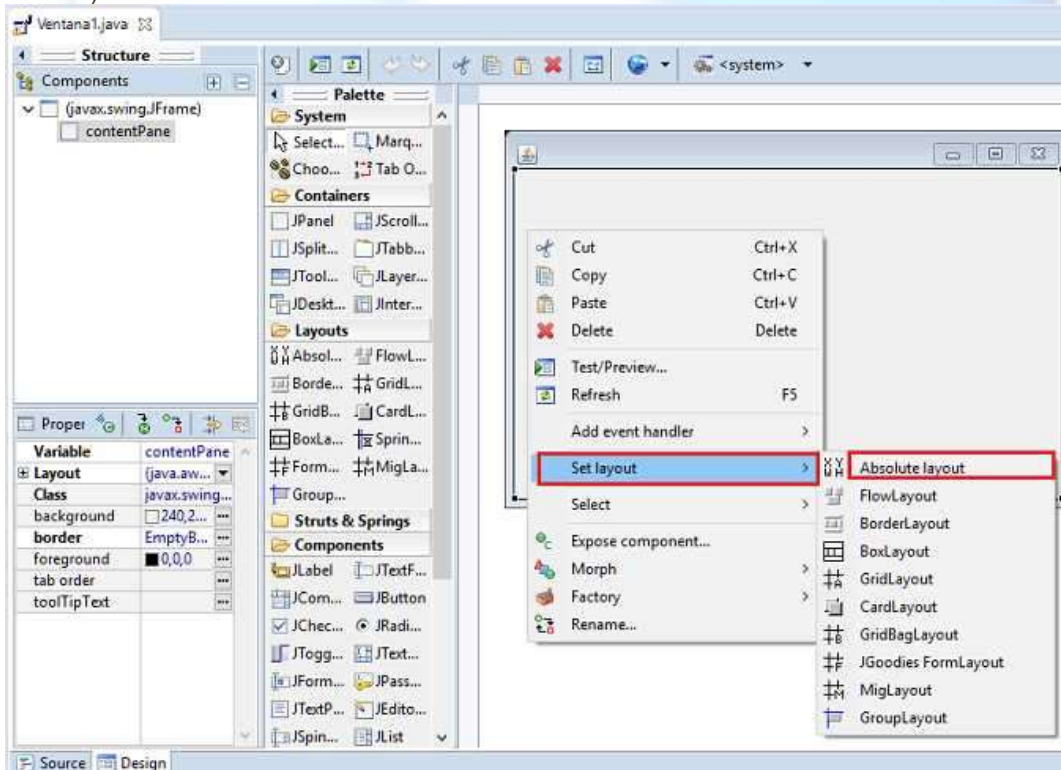


Luego en vista de "Design":

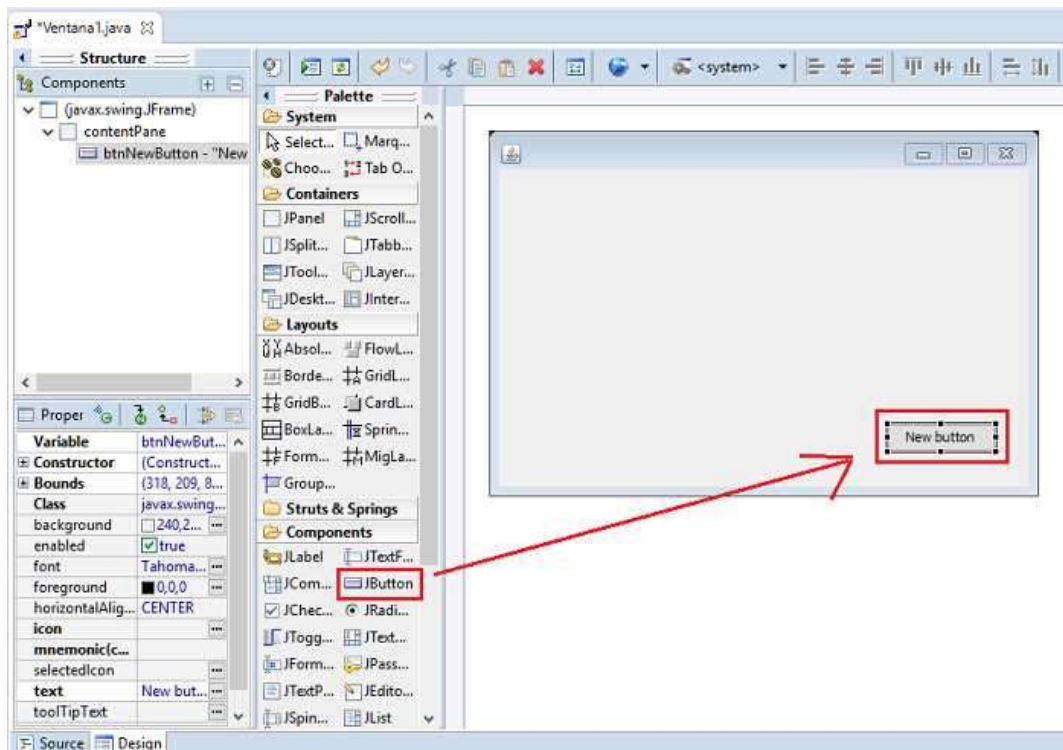




6 - Configuramos el Layout de JFrame presionando el botón derecho del mouse sobre el formulario generado y seleccionamos la opción SetLayout > Absolute layout (esto nos permite luego disponer controles visuales como JButton, JLabel etc. en posiciones fijas dentro del JFrame):



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:

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

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

```
public class Ventana1 extends JFrame {

    private JPanel contentPane;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Ventana1 frame = new Ventana1();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the frame.
     */
}
```

```

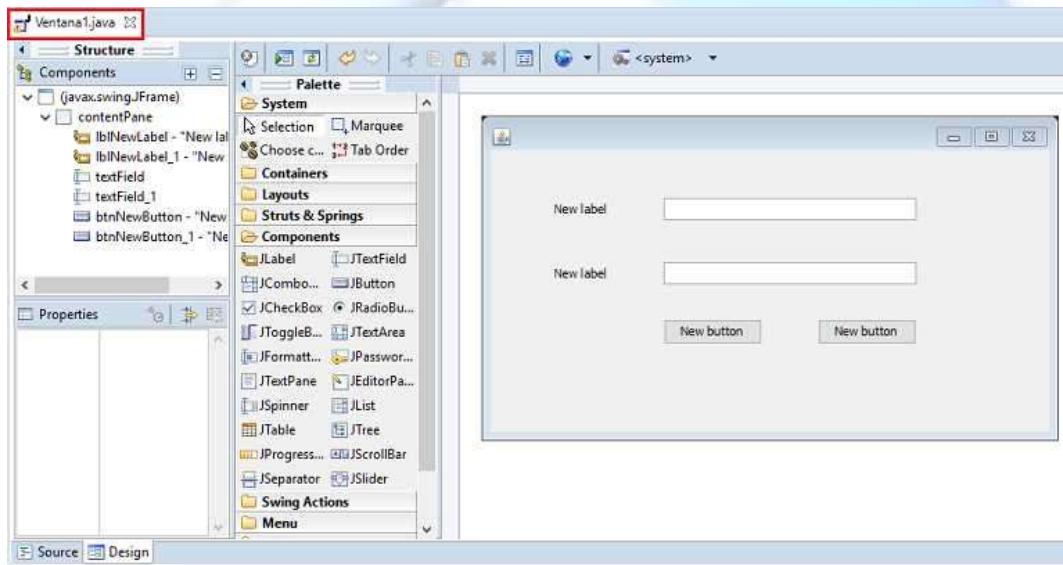
*/
public Ventana1() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 300);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(null);

    JButton btnNewButton = new JButton("New button");
    btnNewButton.setBounds(335, 228, 89, 23);
    contentPane.add(btnNewButton);
}
}

```

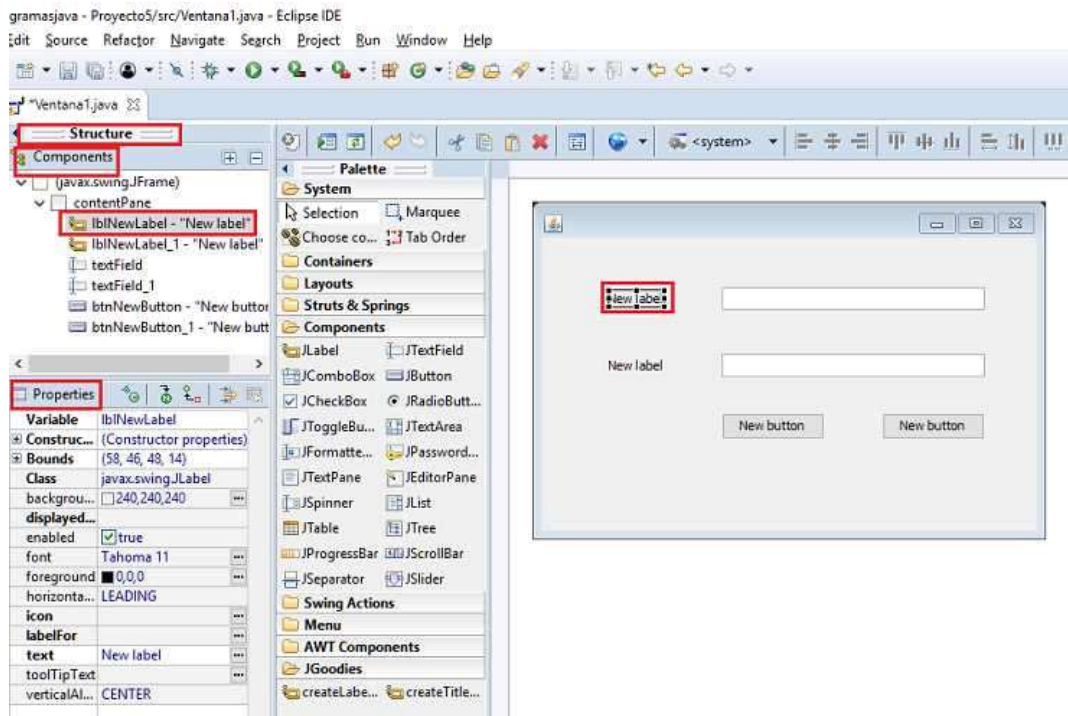
### Inicializar propiedades de los objetos.

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)

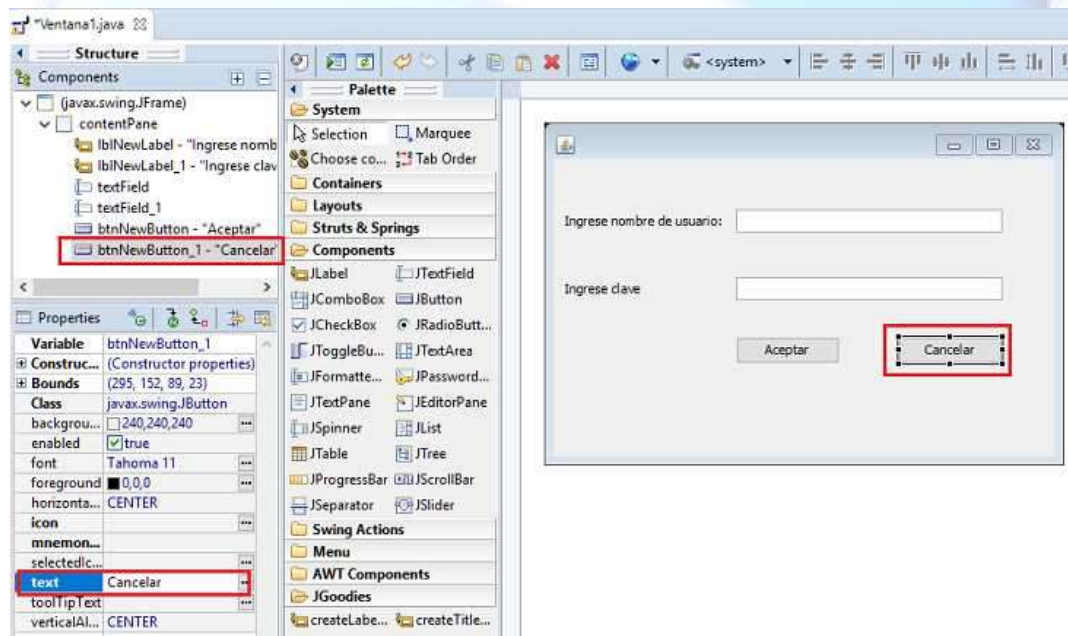


Si hacemos doble clic con el mouse en la pestaña Ventana1.java podemos maximizar el espacio de nuestro editor visual (haciendo nuevamente doble clic vuelve al tamaño anterior). Seleccionemos el primer JLabel de nuestro formulario y observemos las distintas partes que componen el plug-in WindowBuilder. 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 de disponer el texto "Ingrese 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`:

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

```
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;
```

```
public class Ventana1 extends JFrame {
```

```
    private JPanel contentPane;
    private JTextField textField;
    private JTextField textField_1;
```

```
    /**
```

```
     * Launch the application.
```

```
     */
```

```
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Ventana1 frame = new Ventana1();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

```
    /**
```

```
     * Create the frame.
```

```
     */
```

```
    public Ventana1() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 450, 203);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        JLabel lblNewLabel = new JLabel("Ingrese nombre de usuario:");
        lblNewLabel.setBounds(20, 21, 149, 14);
        contentPane.add(lblNewLabel);

        JLabel lblNewLabel_1 = new JLabel("Ingrese clave:");
        lblNewLabel_1.setBounds(20, 61, 125, 14);
        contentPane.add(lblNewLabel_1);

        textField = new JTextField();
        textField.setBounds(179, 18, 225, 20);
        contentPane.add(textField);
        textField.setColumns(10);

        textField_1 = new JTextField();
        textField_1.setBounds(179, 58, 225, 20);
        contentPane.add(textField_1);
        textField_1.setColumns(10);

        JButton btnNewButton = new JButton("Aceptar");
        btnNewButton.setBounds(178, 89, 89, 23);
    }
}
```

```

contentPane.add(btnNewButton);

JButton btnNewButton_1 = new JButton("Cancelar");
btnNewButton_1.setBounds(315, 89, 89, 23);
contentPane.add(btnNewButton_1);
}
}

```

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:

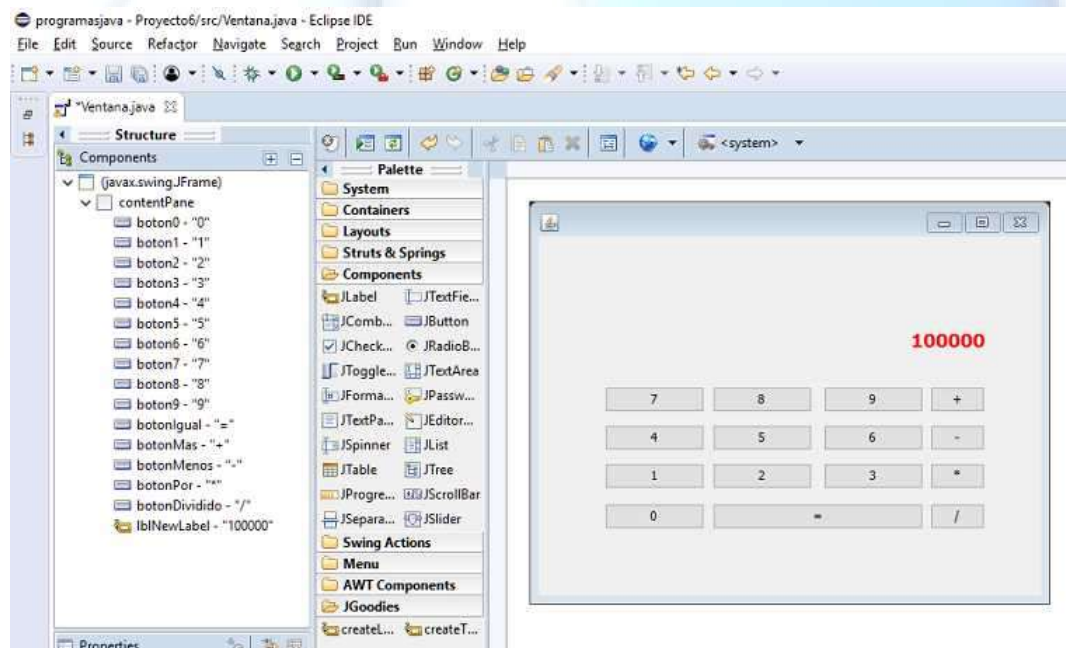
```

JLabel lblNewLabel = new JLabel("Ingrese nombre de usuario.");
.....
JLabel lblNewLabel_1 = new JLabel("Ingrese clave:");
.....
JButton btnNewButton = new JButton("Aceptar");
.....
JButton btnNewButton_1 = new JButton("Cancelar");
.....

```

### Problema propuesto 1

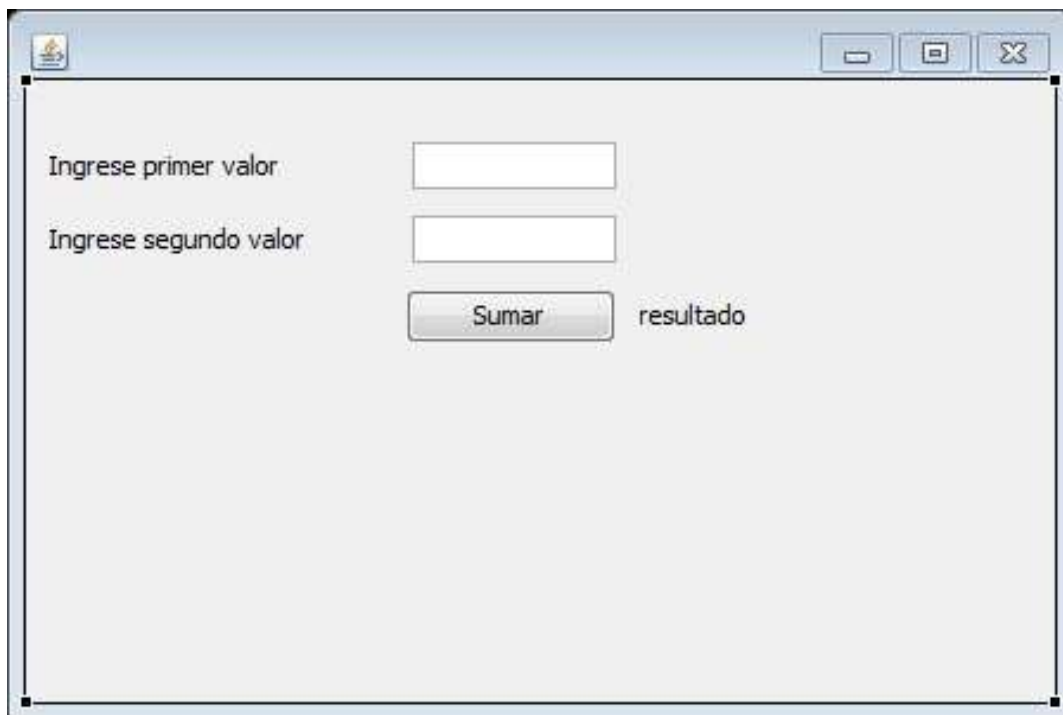
Crear la interfaz visual que aparece abajo. Inicializar las propiedades 'text' y 'variable' para cada JButton y la JLabel.



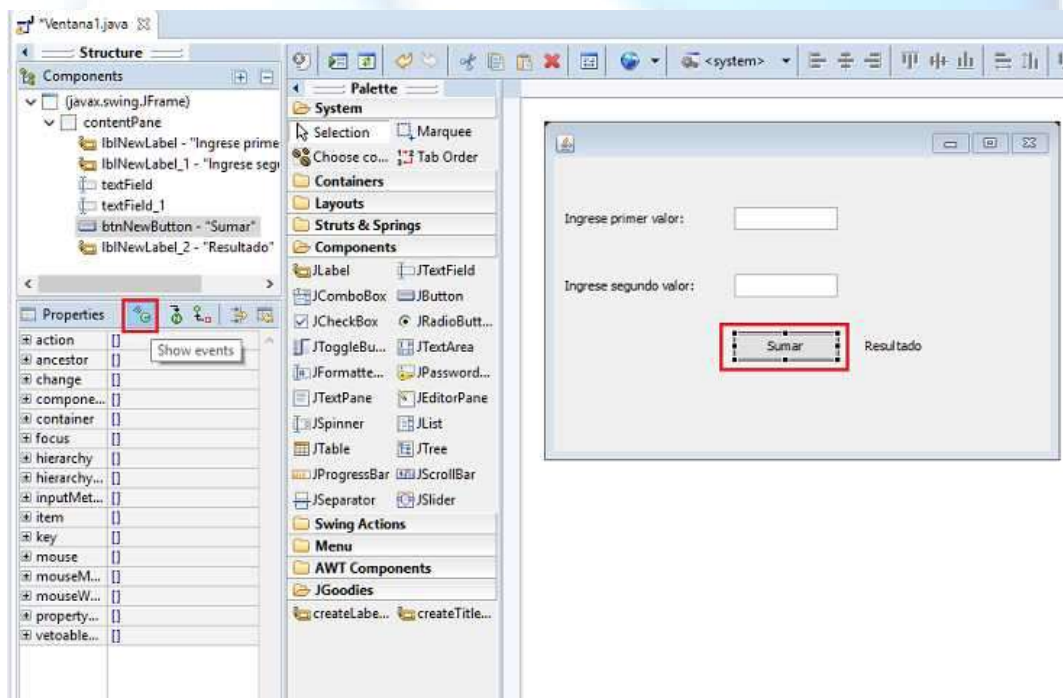
### Eventos

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:

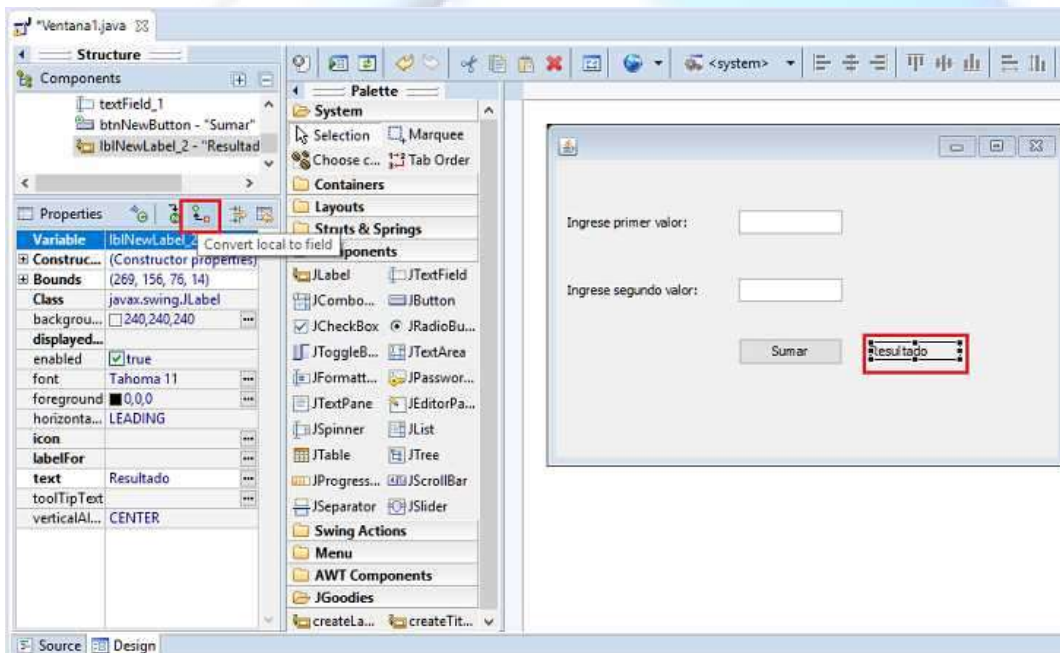
```
btnNewButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
    }
});
```

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 `TextField`, 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.

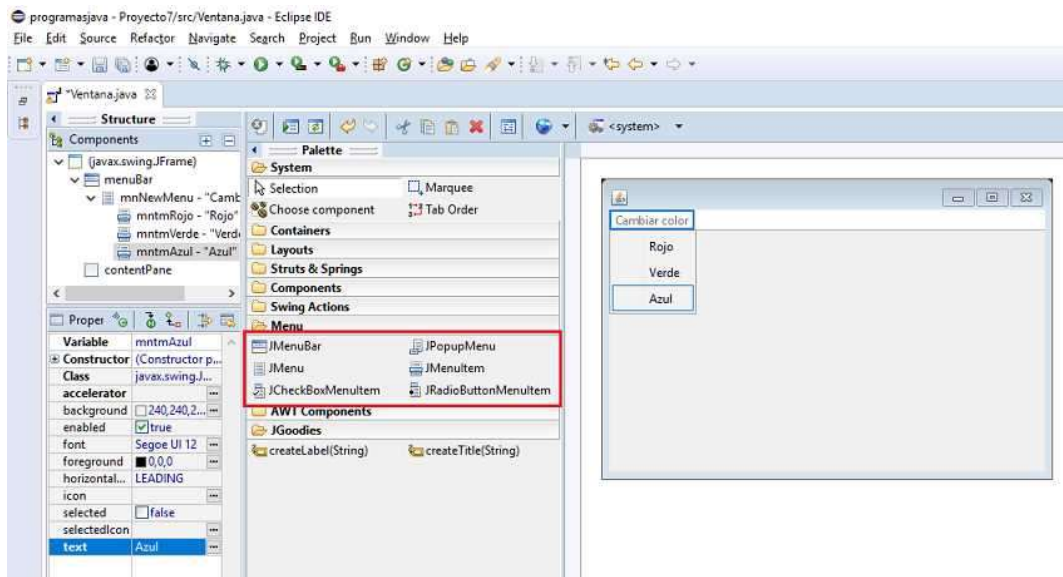
Los nombres de objetos que automáticamente genera el `WindowBuilder`, por ejemplo `btnNewButton`, `lblNewLabel`, `lblNewLabel_1`, `lblNewLabel_2` etc. los podemos cambiar desde la ventana de propiedades a través de la propiedad 'variable'.

## 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);
    }
});
```

```
});
```

### 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.

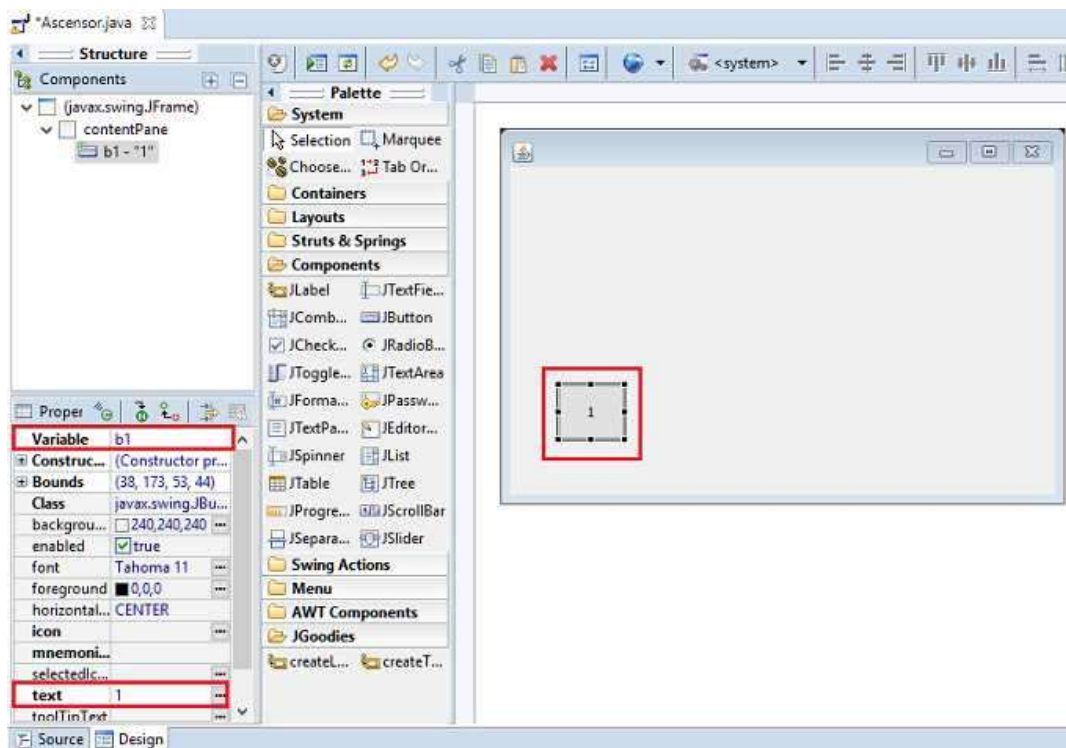
Por ejemplo: si se presiona el botón 3 se muestra en un JLabel el piso número 3 y en otra JLabel la dirección. La cadena "Sube", en caso de presionar un piso superior al actual.

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".

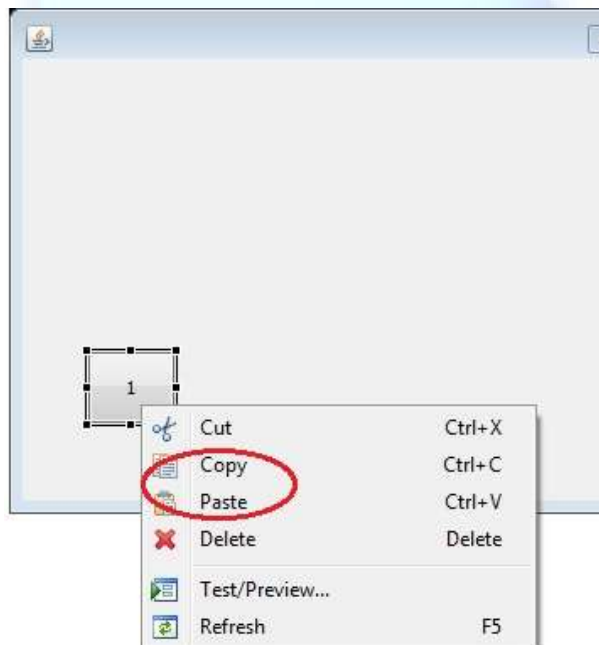
Algunos consejos para crear la interfaz visual:

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):

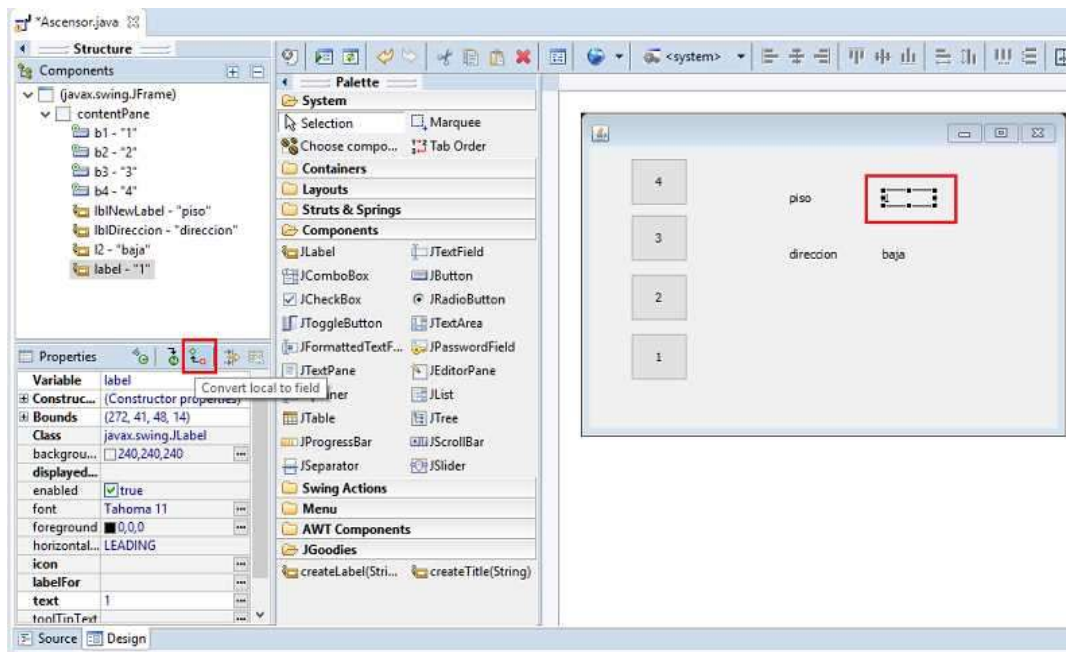


3 - Los otros botones los podemos crear de la misma manera seleccionando un objeto de la clase JButton de la "Palette" o cuando tenemos que crear otros objetos semejantes podemos presionar el botón derecho del mouse sobre el objeto a duplicar y seguidamente en el menú contextual seleccionar la opción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

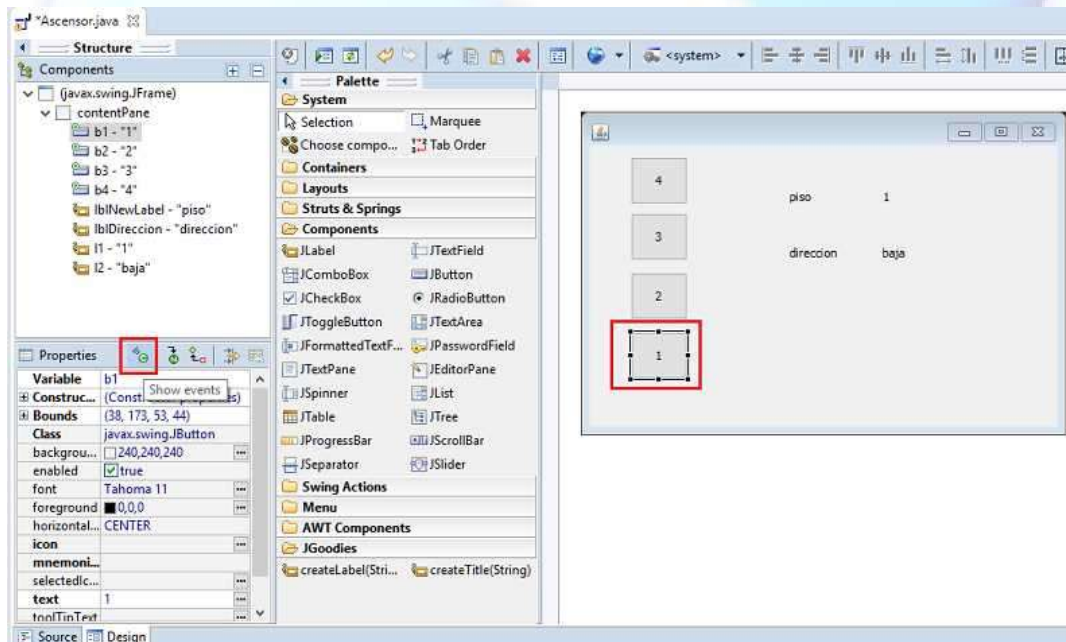


4 - Los objetos que necesitemos consultar o modificar en tiempo de ejecución debemos definirlos como atributos de clase (también llamados campos de clase). En este problema cuando se presione alguno de los cuatro botones debemos consultar el

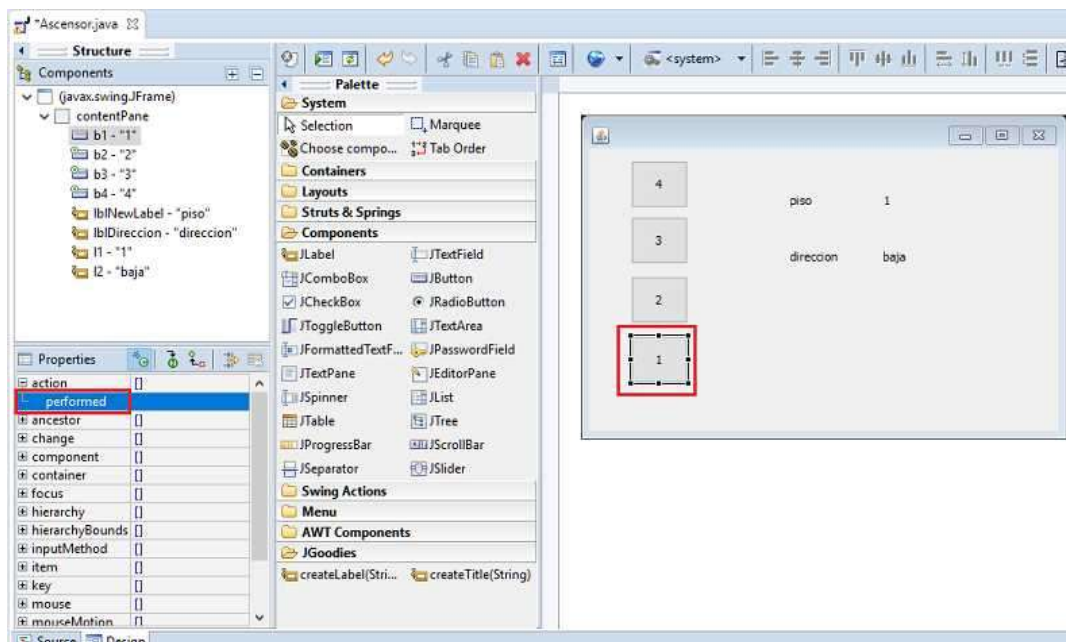
contenido de la label que indica el piso actual y la label que muestra la dirección será modificada por otro String.  
Para definir un control visual como atributo de clase debemos seleccionarlo y presionar en la ventana de propiedades el botón "Convert local to field" (en nuestro problema definamos 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:

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 comboPesos y comboCentavos)  
 En el JComboBox pesos 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 pesos 80 centavos.

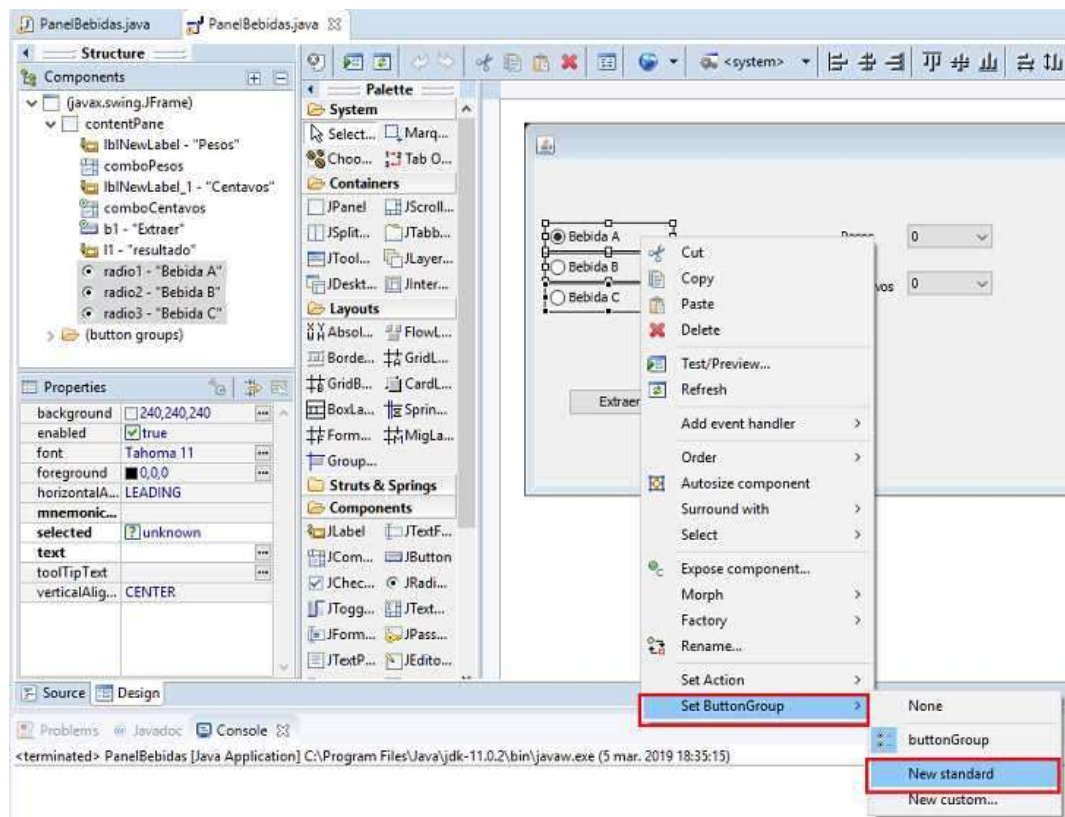
Bebida B tiene un costo de 1 peso 20 centavos.

Bebida C tiene un costo de 3 pesos 10 centavos.

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 pesos y centavos seleccionados.

Solución:

Para que todos los JRadioButton estén asociados (es decir que cuando se seleccione uno se deselectione 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 comboPesos;
    private JComboBox comboCentavos;
    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("Pesos");
    lblNewLabel.setBounds(263, 59, 46, 14);
    contentPane.add(lblNewLabel);

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

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

    comboCentavos = new JComboBox();
    comboCentavos.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
        }
    });
    comboCentavos.setModel(new DefaultComboBoxModel(new String[] {"0", "10", "20", "30",
"40", "50", "60", "70", "80", "90"}));
    comboCentavos.setBounds(319, 96, 73, 20);
    contentPane.add(comboCentavos);

    JButton b1 = new JButton("Extraer");
    b1.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
            int pesos=Integer.parseInt((String)comboPesos.getSelectedItem());
            int centavos=Integer.parseInt((String)comboCentavos.getSelectedItem());
            if (radio1.isSelected() && pesos==0 && centavos==80)
                l1.setText("Correcto");
            else
                if (radio2.isSelected() && pesos==1 && centavos==20)
                    l1.setText("Correcto");
            else
                if (radio3.isSelected() && pesos==3 && centavos==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 pesos=Integer.parseInt((String)comboPesos.getSelectedItem());
        int centavos=Integer.parseInt((String)comboCentavos.getSelectedItem());
        if (radio1.isSelected() && pesos==0 && centavos==80)
            l1.setText("Correcto");
        else
            if (radio2.isSelected() && pesos==1 && centavos==20)
                l1.setText("Correcto");
            else
                if (radio3.isSelected() && pesos==3 && centavos==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 pesos y 80 centavos, en tal caso mostramos en la label el mensaje "Correcto". La lógica es similar para las otras dos bebidas.

### Problema 3

Un embalse debe manejar la cantidad de mts3 de agua que pasa por cada compuerta. Por cada compuerta puede pasar un caudal de 100 mts3 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 mts3 x seg.) , Medio (> 100 -200 mts3. x seg.) o Alto (>200 mts3 x seg.)  
 Para la selección del caudal de cada compuerta utilizar componentes de tipo JSpinner.



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");

```

### Problema propuesto

1. 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 400.  
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 pesos. 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.



### 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:

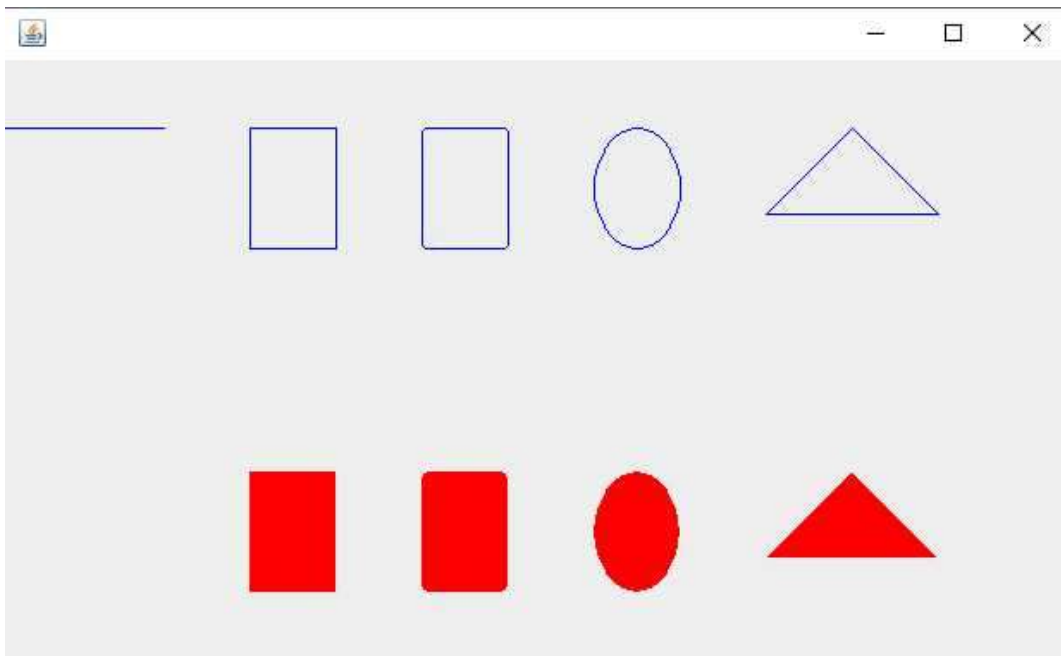
- 1) Dibujo de primitivas gráficas,
- 2) Dibujo de texto,
- 3) Presentación de imágenes en formatos \*.gif y \*.jpeg.

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 vertices 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);
}
}

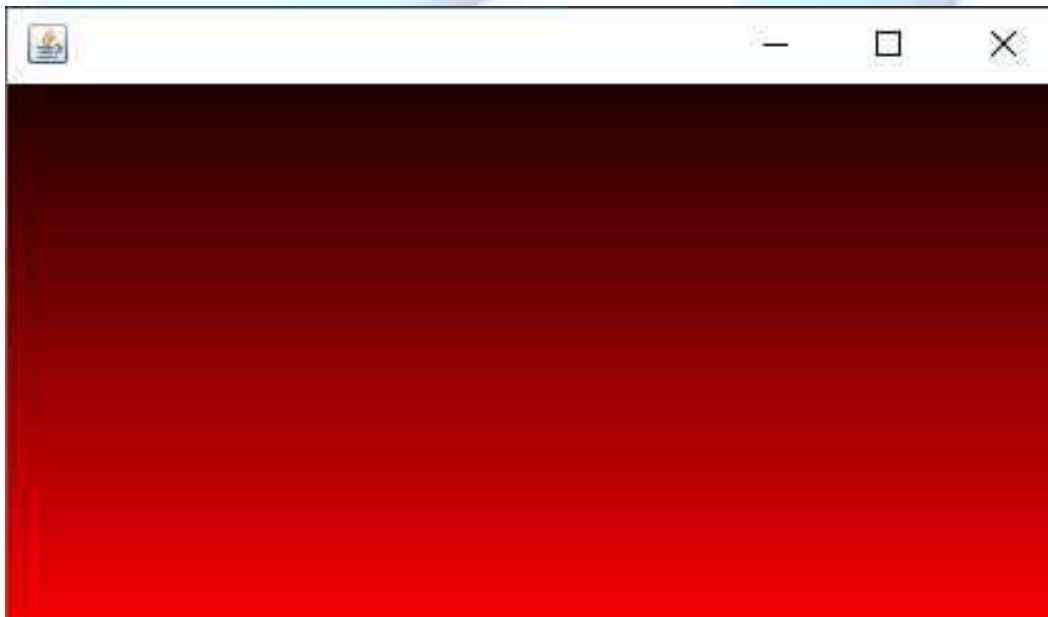
```

### 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 imágenes; 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` dentro de un `JFrame`.



Luego 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

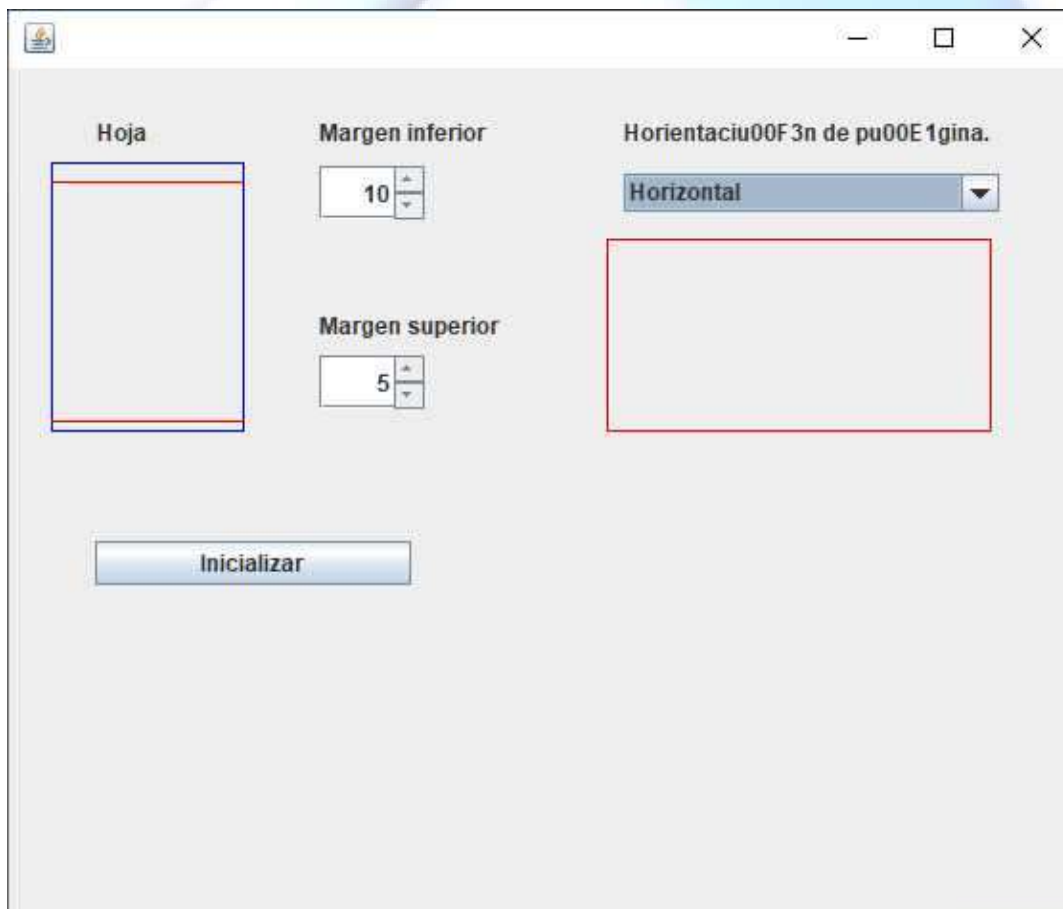
Se debe desarrollar una pantalla para configurar ciertas características de un procesador de texto.

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.



Para implementar esta aplicación con el WindowBuilder creamos la interfaz visual, disponemos 4 objetos de la clase JLabel, dos JSpinner, un JButton y un objeto de la clase JComboBox. El dibujo de la hoja con las líneas de márgenes superior e inferior como el gráfico de orientación de la hoja se hacen en el método paint.

El código fuente que resuelve esta aplicación es:

```
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("Horientaci00F3n de p00E1gina.");
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 );
    }
}

```

### 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):

```

btnIniciar.addActionListener(new ActionListener() {

```

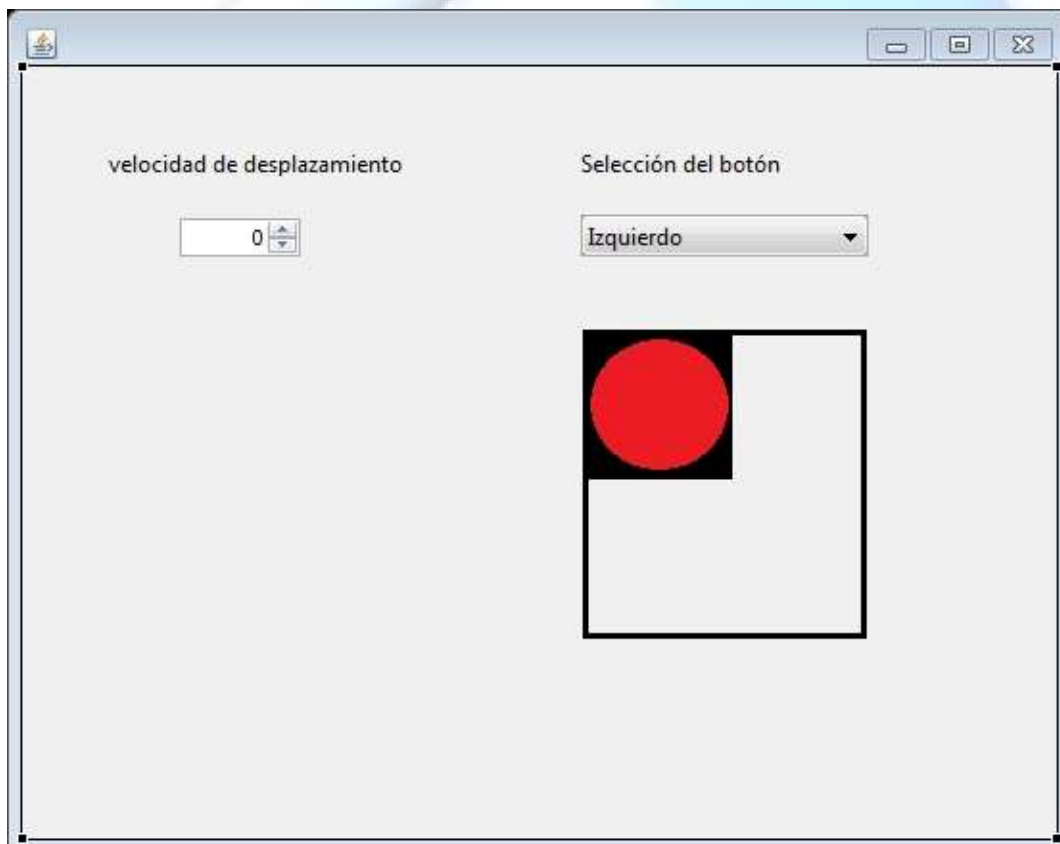
```

public void actionPerformed(ActionEvent arg0) {
    sp1.setValue(0);
    sp2.setValue(0);
    comboBox.setSelectedIndex(0);
    repaint();
}
});

```

### Problemas propuestos

1. 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)

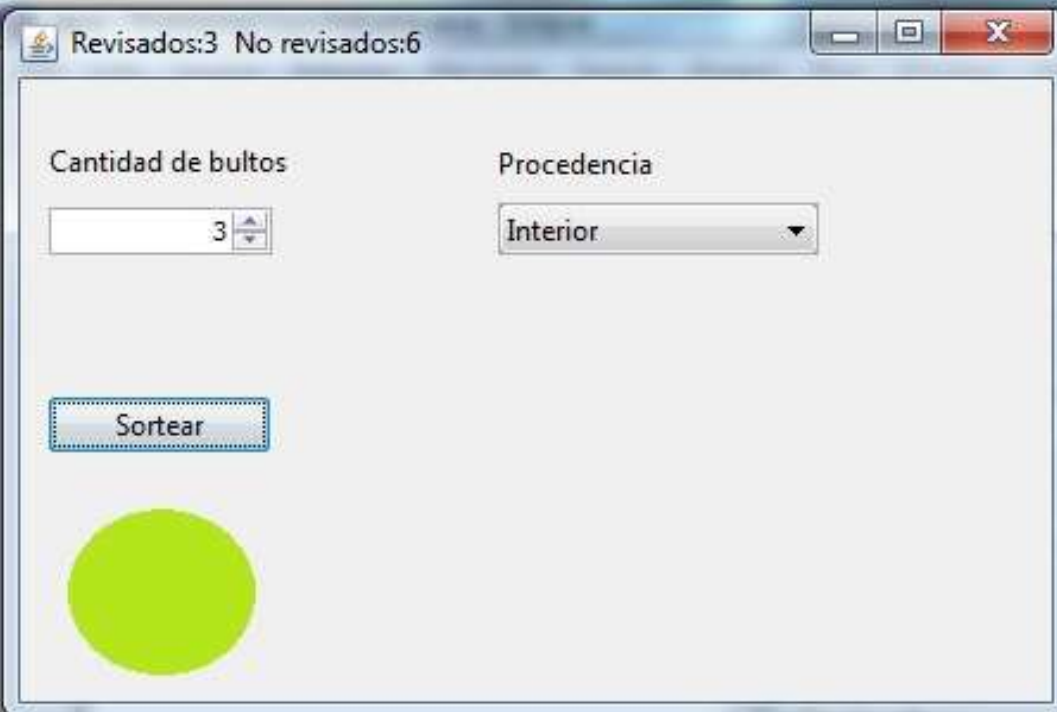


2. En una aduana hay una máquina que sorteas las personas cuyo equipaje serán revisados. La persona selecciona si viene del Interior del país o del Exterior (a través de un JComboBox), y por otro lado selecciona la cantidad de bultos (JSpinner). Luego presiona el botón sortear y aparece al lado de este botón un círculo rojo o verde. (En caso de ser rojo se revisa su equipaje, en caso de ser verde, no se revisa) Para el sorteo generar un valor aleatorio entre 1 y 3. Si se genera un 1 se revisa, si se genera un 2 o 3 no se revisa. Validar que también este seleccionado un valor distinto a cero en bultos (los valores pueden ir de 0 a 10). Si la cantidad de bultos supera a 5 se revisa siempre sus bultos (es decir que aparece

un círculo rojo).

Luego de sortear fijar en cero cantidad de bultos.

Mostrar en el título del JFrame la cantidad de bultos revisados y no revisados hasta el momento.



The screenshot shows a Java Swing window with the title bar text "Revisados:3 No revisados:6". The window contains the following elements:

- A label "Cantidad de bultos" above a spinner control showing the value "3".
- A label "Procedencia" above a dropdown menu showing "Interior".
- A button labeled "Sortear".
- A large red circle.

Muchas gracias hasta la próxima clase.

Alsina 16 [B1642FNB] San Isidro | Pcia. De Buenos Aires |Argentina |

TEL.: [011] 4742-1532 o [011] 4742-1665 |

www.institutosanisidro.com.ar [info@institutosanisidro.com.ar](mailto:info@institutosanisidro.com.ar)