Argentina programa

Desarrollo de aplicaciones JAVA

Guía II

"Interfaz Gráfica de Usuario"

GUI, SWING Y JAVA BEANS

Interfaz de Usuario Gráfica

Una interfaz gráfica de usuario, también conocida como GUI por sus siglas en inglés (Graphical User Interface), es un entorno visual que permite a los usuarios interactuar con un sistema informático mediante elementos gráficos, como ventanas, botones, menús y barras de desplazamiento. Proporciona una forma intuitiva y visualmente atractiva de interactuar con programas y aplicaciones, permitiendo a los usuarios realizar acciones y recibir información de manera eficiente y fácil de entender. En resumen, una interfaz gráfica de usuario es la representación visual y manipulable de un software o sistema que facilita la interacción entre los usuarios y la computadora.

Paquetes de Java para GUIs

Para facilitarnos la programación de las GUIS, Java nos provee de un conjunto de paquetes:

• **java.awt**: Contiene todas las clases para crear interfaces de usuario y para dibujar gráficas e imágenes.

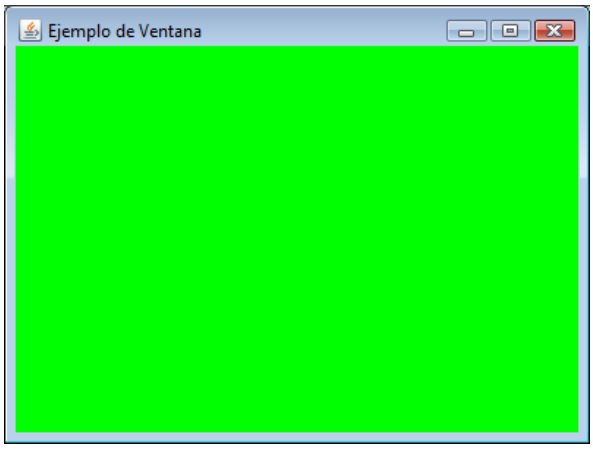
• **javax.swing**: Provee un conjunto de componentes ligeros (escritos completamente en Java) que, en lo máximo posible, trabajan de la misma manera en todas las plataformas.

• **java.awt.event**: Provee interfaces y clases para manejar los diferentes tipos de eventos disparados por los componentes AWT.

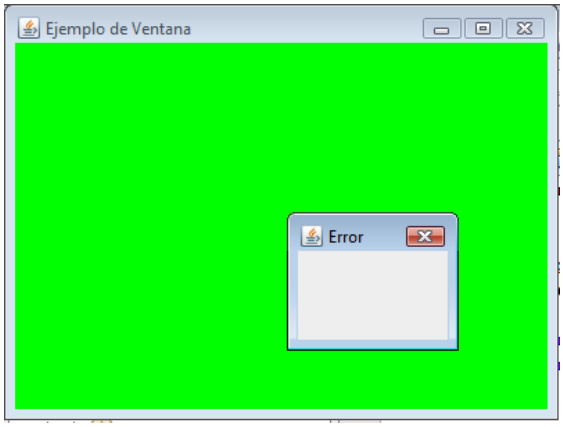
Contenedores

Los diferentes componentes que forman la interfaz gráfica de una aplicación: Etiquetas, botones, cajas de texto, menús, etc. deben agruparse en otros componentes llamados contenedores. El paquete *javax.swing* tiene los siguientes contenedores para agrupar componentes.

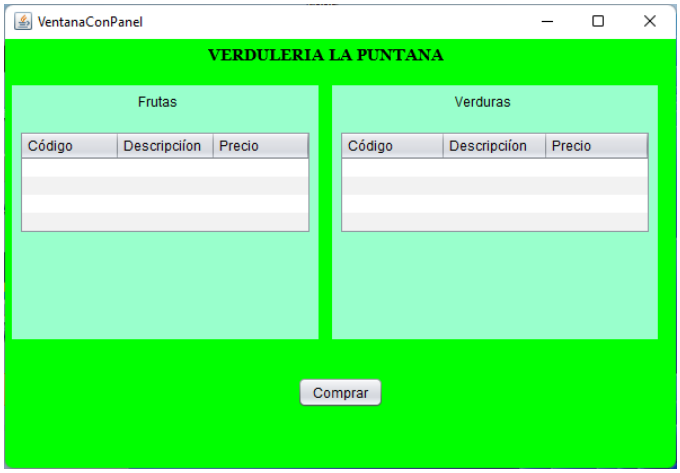
• **JFrame**: Permite crear una ventana de una aplicación. Posee título y puede tener una barra de menús, barra de herramienta, barras de desplazamiento, su propio cursor, botones para maximizar, minimizar, etc.



• **JDialog**: Permite crear cuadros de diálogo.



• **JPanel**: Permite agrupar componentes. Podemos tener paneles dentro de paneles. Tienen un manejador de diseño.

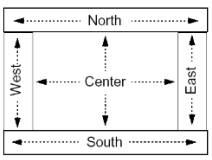


Manejadores de Diseño

Un manejador de diseño es una clase que define la forma en que se acomodan los componentes insertados en un contenedor. El paquete *javax.swing* tiene los siguientes manejadores de diseño para acomodar componentes en un contenedor.

• **FlowLayout**: Coloca los componentes de izquierda a derecha por filas, con las filas ordenadas de arriba abajo.

• **BorderLayout**: El contenedor se divide en cinco regiones: North, East, West, South y Center.



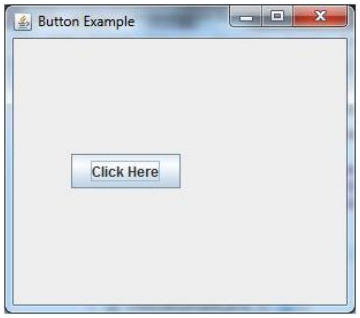
• **GridLayout**: Acomoda a los componentes en filas y columnas con todas las regiones del mismo tamaño.

• **GridBagLayout**: Acomoda a los componentes en filas y columnas que no necesariamente tienen la misma altura y ancho.

Componentes Activos

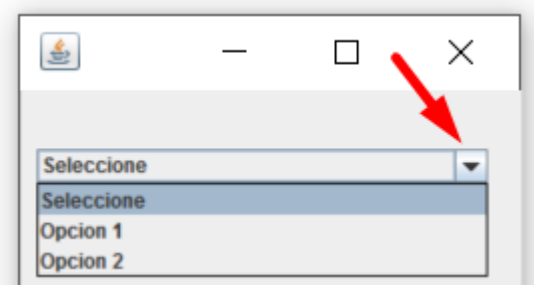
Los componentes activos del paquete Swing de Java permiten que el usuario interactúe con el programa:

• **JButton**: Implementa un botón.

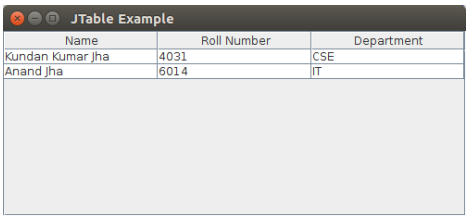


• **JCheckBox**: Implementa una casilla de verificación.

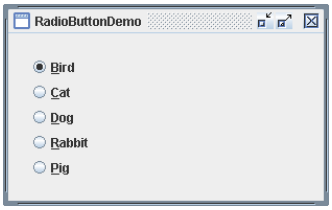
• **JComboBox**: Una componente que representa una lista desplegable de elementos.



•**JTable**: Componente que se usa para mostrar o editar datos bidimensionales que tienen filas y columnas. Es similar a una hoja de cálculo.



•**JRadioButton**: Es un widget que permite elegir una sola opción de un conjunto predeterminado de estas.

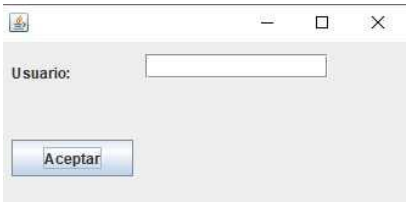


Componentes de Texto

Hay cuatro componentes que pueden usarse para el despliegue y captura de textos:

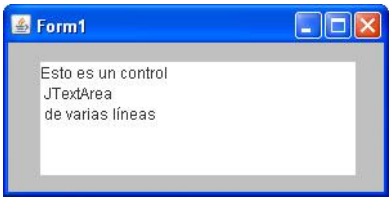
• **JLabel**: Despliega una línea de texto en la pantalla.

• **JTextField**: Un cuadro que contiene una línea de texto. El usuario puede modificarla.



• **JPassword**: Un cuadro que permite la captura de una contraseña. No hace eco de los caracteres. En lugar despliega un carácter dado.

• **JTextArea**: Un cuadro que contiene una o más líneas de texto. El usuario puede modificarlas.



Menús

El paquete java.swing de Java contiene una serie de clases que permiten crear diferentes menús:

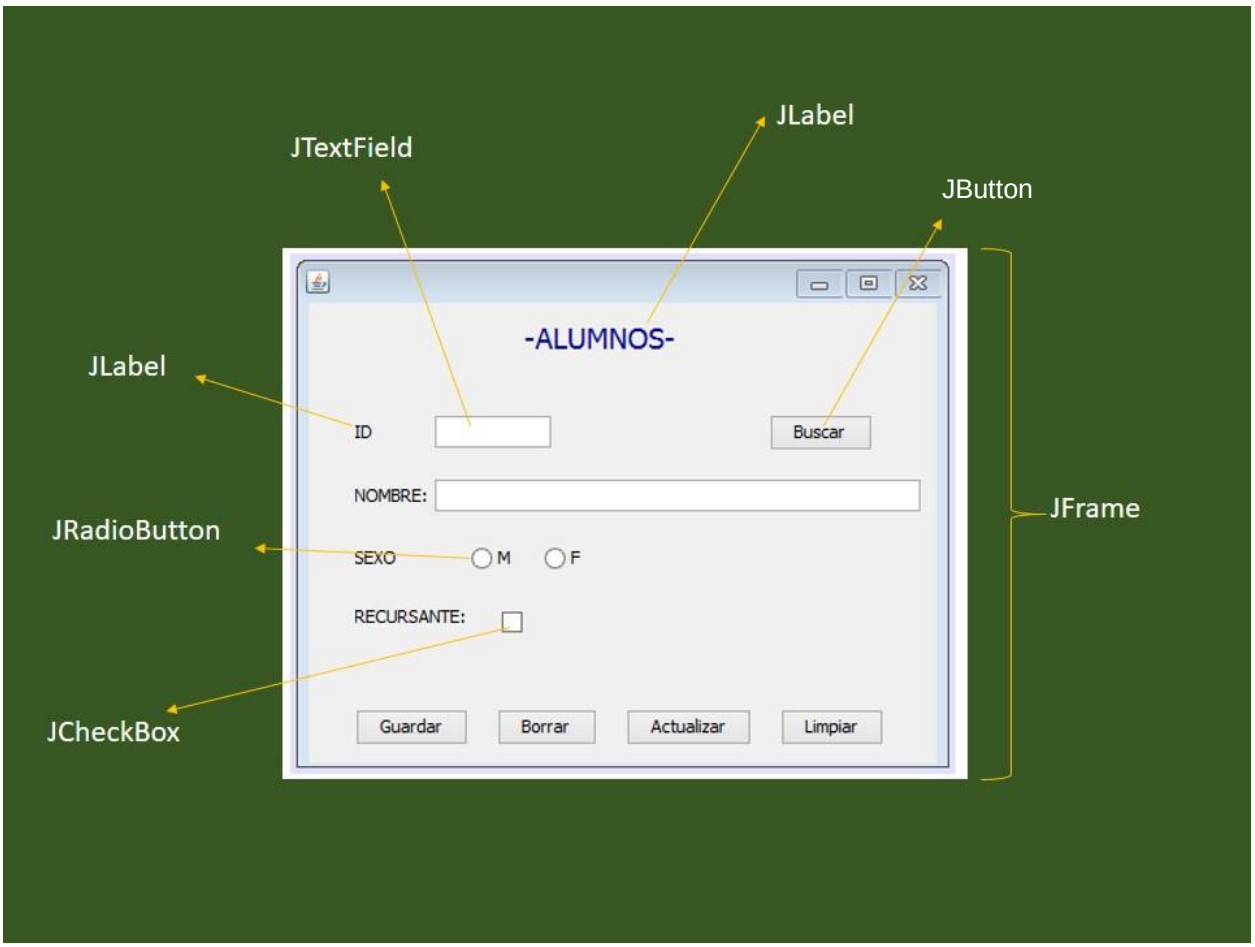
• **JMenuBar**: Permite crear una barra de menús. Contiene un conjunto de objetos de tipo JMenu y debe ser parte de una ventana, JFrame.

• **JMenu**: Permite implementar menús. Contiene una colección de objetos JMenuItem y separadores.

• **JMenuItem**: Permite implementar las opciones de un menú.

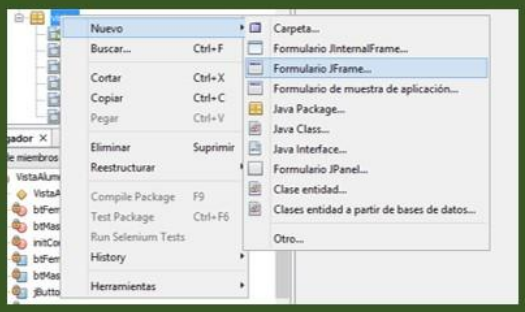
• Un **Jmenu** también puede ser un **JMenuItem** permitiendo menús jerárquicos.

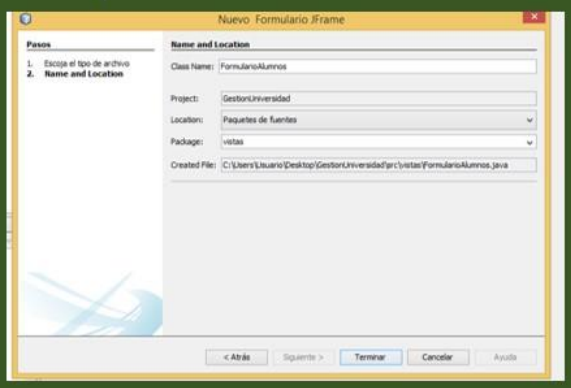
Ejemplo de Vista, utilizando el asistente de Netbeans:

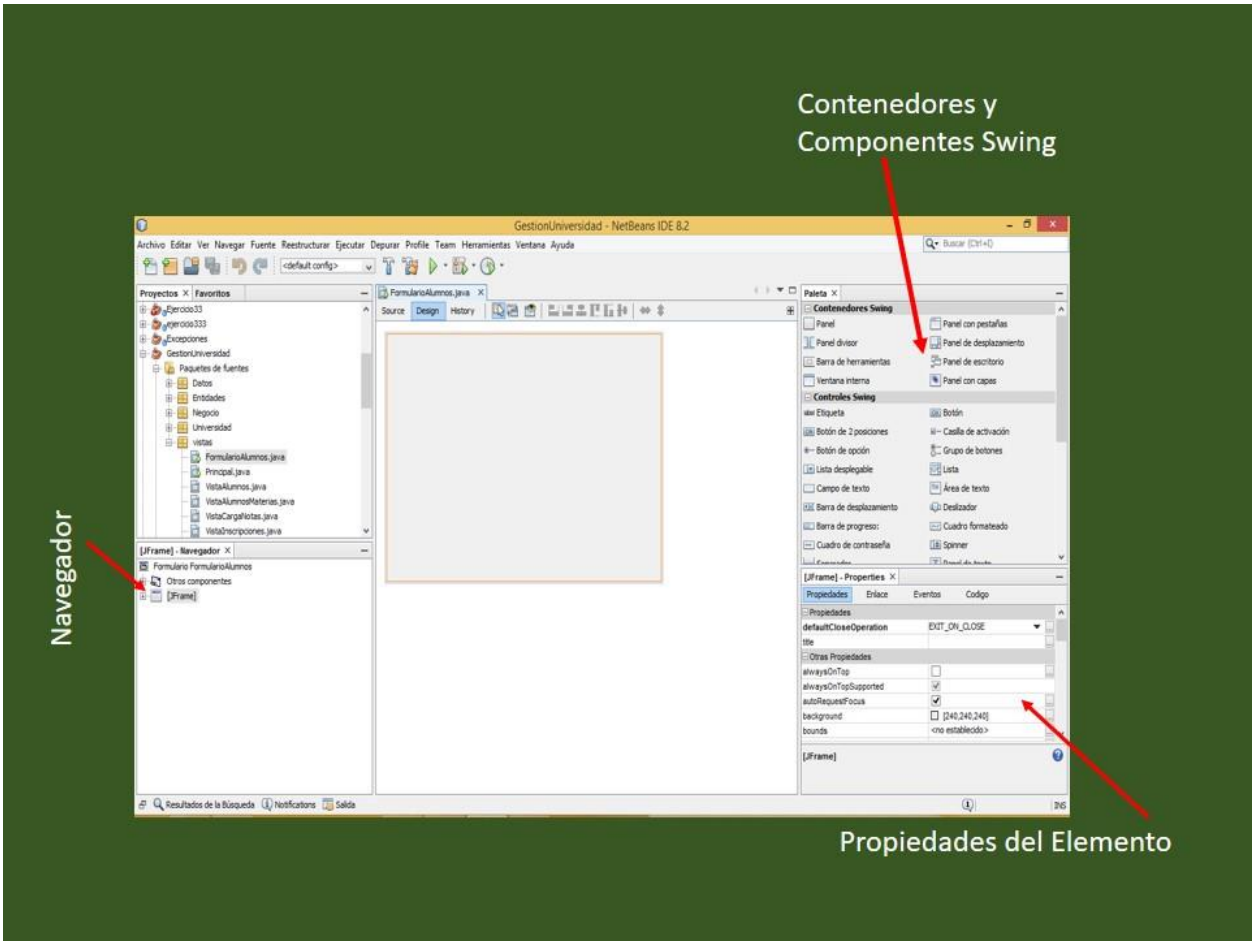


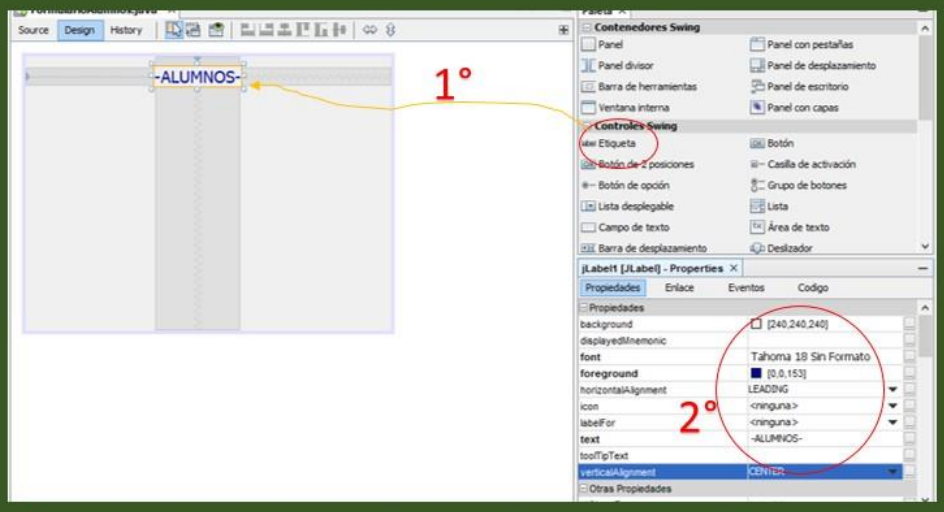
Creando nuestro Jframe:FormularioAlumnos

Sobre el paquete, dentro del cual deseamos crear nuestra clase JFrame, hacemos clic secundario…..



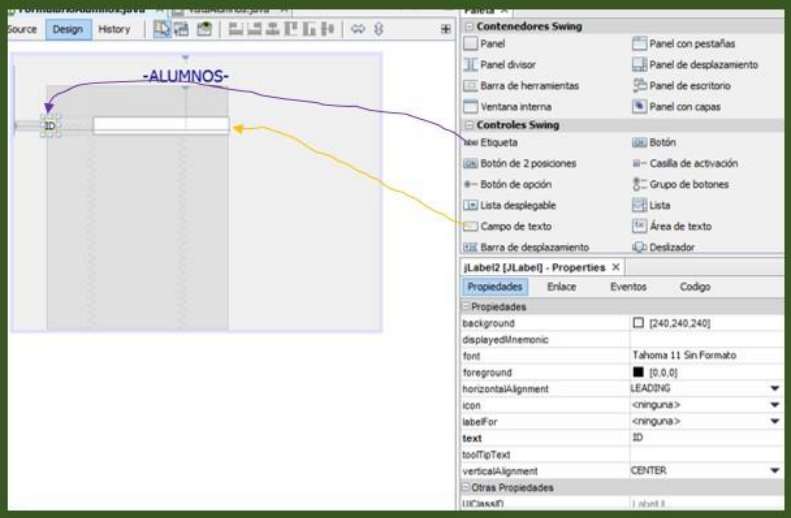






1° Arrastro del cuadro de controles Swing el componente JLabel(Etiqueta) adentro del Jframe

2° En el panel de propiedades elijo: font= “Tahoma 18”, foreground=“azul”, texto=“-ALUMNOS-”

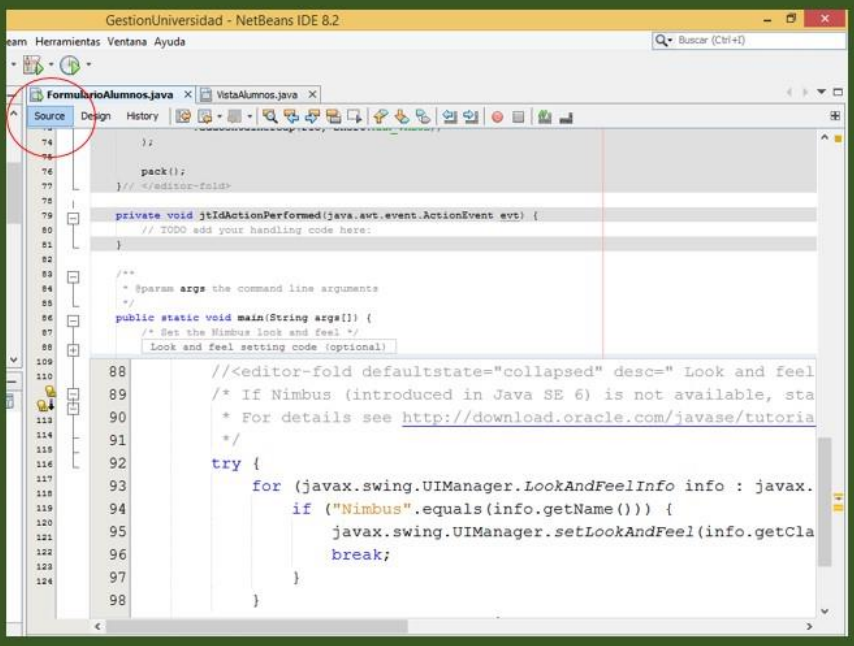


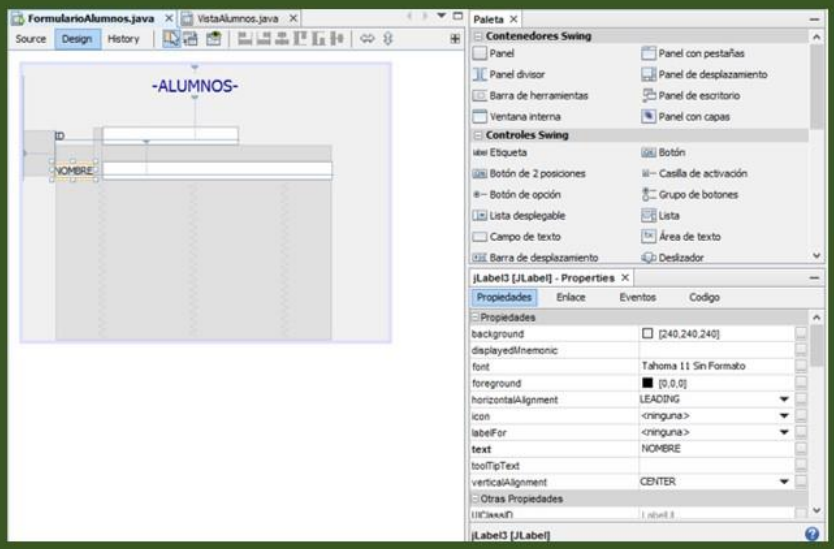
JLabel(Etiqueta):
Propiedades:
font="Tahoma 11"
Foreground="negro"
text="ID"

JTextField (Campo de Texto):
Propiedades:
font="Tahoma 11"
foreground="negro"
text=""

Codigo:
name="jtId"

Veamos el código generado hasta ahora…..

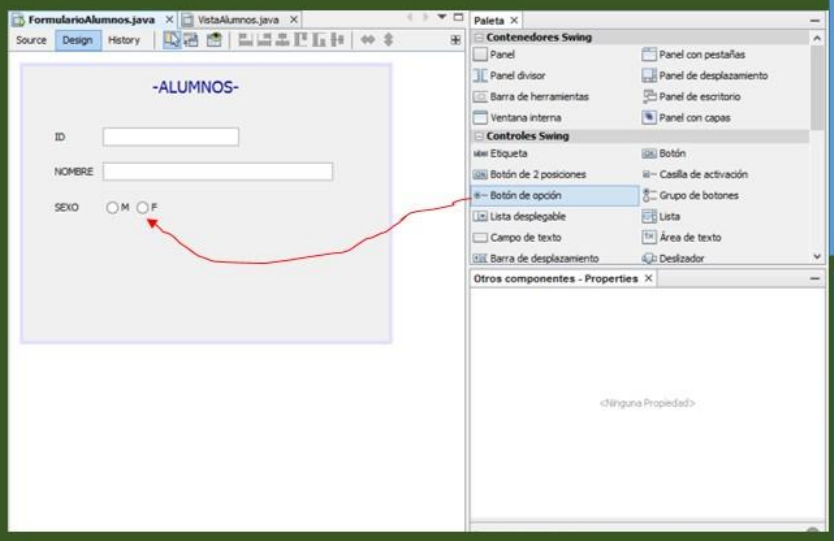




JLabel(Etiqueta):
Propiedades:
font="Tahoma 11"
Foreground="negro"
text="NOMBRE"

JTextField (Campo de Texto):
Propiedades:
font="Tahoma 11"
foreground="negro"
text=""

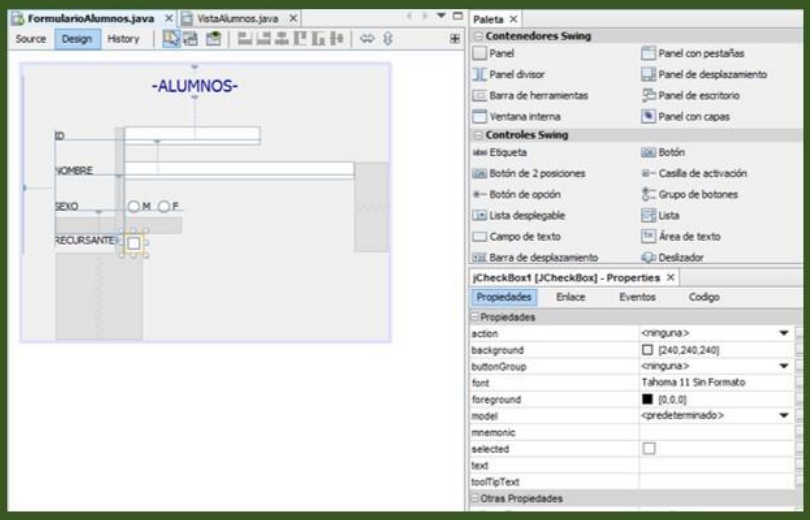
Codigo:
name="jtNombre"



JLabel(Etiqueta):
Propiedades:
font="Tahoma 11"
Foreground="negro"
text="SEXO"

JRadioButton(Botón de Opción):
Propiedades:
font="Tahoma 11"
Foreground="negro"
text="F" "M"

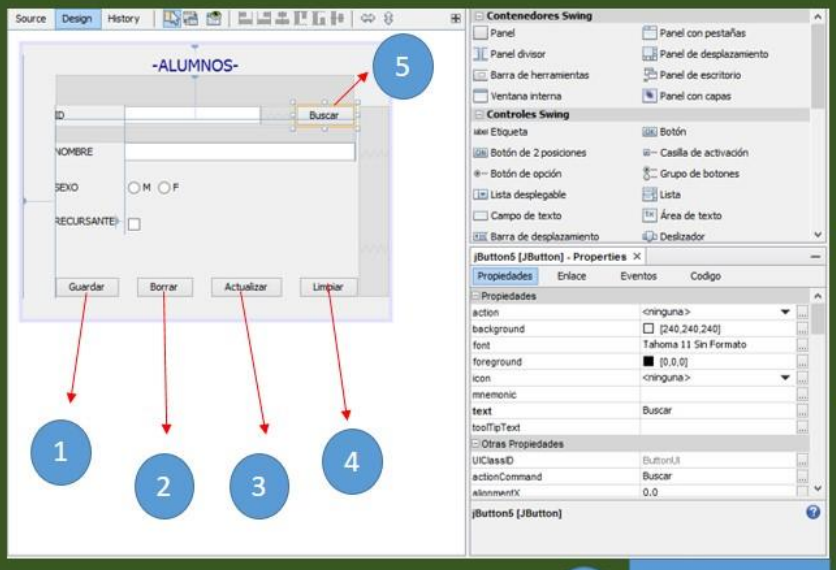
Código:
name="rbFemenino"
name="rbMasculino"



JLabel(Etiqueta):
Propiedades:
font="Tahoma 11"
Foreground="negro"
text="RECURSANTE"

JCheckBox(Casilla de Verificación):
Propiedades:
font="Tahoma 11"
Foreground="negro"
text=" "

Código:
name="chRecursante"



Jbutton(Botón):
Propiedades:
font="Tahoma 11"
Foreground="negro"

1
Propiedades:
text="Guardar"

Código:
name=btGuardar

2
Propiedades:
text="Borrar"

Código:
name=btGuardar

3
Propiedades:
text="Actualizar"

Código:
name=btActualizar

4
Propiedades:
text="Limpiar"

Código:
name=btLimpiar

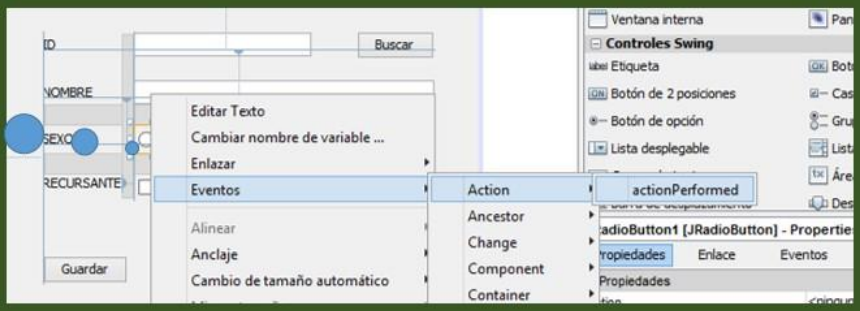
5
Propiedades:
text="Buscar"

Código:
name=btBuscar

Eventos….

Necesitamos hacer que cuando el RadioButton “F” este seleccionado el RadioButton “M” este deseleccionado y viceversa.

Hacemos clic secundario sobre el RadioButton “M”



Eventos → Action → actionPerformed

```
private void rbMasculinoActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    rbFemenino.setEnabled(false);

}
```

Y luego hacemos lo mismo con el RadioButton “F”

```
private void rbFemeninoActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    rbMasculino.setEnabled(false);
}
```

PROGRAMACIÓN POR EVENTOS

En la programación orientada por eventos el flujo de un programa no sigue una secuencia de inicio a fin sino que está controlada por eventos externos.

Los programas con una interfaz de usuario gráfica por lo general son controlados por eventos externos como hacer un clic en una opción de un menú o en un botón, arrastrar el ratón, escribir en un campo de texto, etc.

Modelo de Eventos de Java

Un programa en Java maneja los eventos externos de la siguiente manera:

Toda componente puede reaccionar ante un evento externo generando una notificación llamada evento. Se dice que el componente puede ser la fuente de un evento.

Toda clase puede recibir la notificación generada por un componente. Se dice que la clase es una oyente del evento.

Un componente que genera eventos, mantiene una lista de las oyentes interesadas en saber cuándo ocurre el evento

El evento generado por un componente sólo se envía a las oyentes registradas. Las que están en la lista del componente.

Eventos

En Java, un evento es una subclase de la clase AWTEvent que se encuentran en el paquete *java.awt.event*.

Un objeto de tipo evento contiene la información del evento externo que lo generó. Cada componente genera uno o más tipos de eventos

Oyentes

Para que una clase pueda ser oyente de un evento debe implementar una o más interfaces oyente.

Una interfaz oyente declara uno o más métodos oyentes que deben ser implementados por la clase oyente.

Cuando un componente genera un evento, invoca a un método oyente de la clase oyente y le pasa como parámetro el evento.

El método oyente contiene el código que maneja al evento generado por el componente.

Síntesis:

Fuentes de Eventos

Una fuente es un generador de eventos (Ej. JBotón, JTextfield).

Cuando el usuario hace click sobre el botón se genera una instancia ActionEvent con el botón como fuente.

La instancia ActionEvent es un objeto que contiene información acerca de los eventos que tuvieron lugar.

getActionCommand: Devuelve el nombre del comando asociado con la acción.

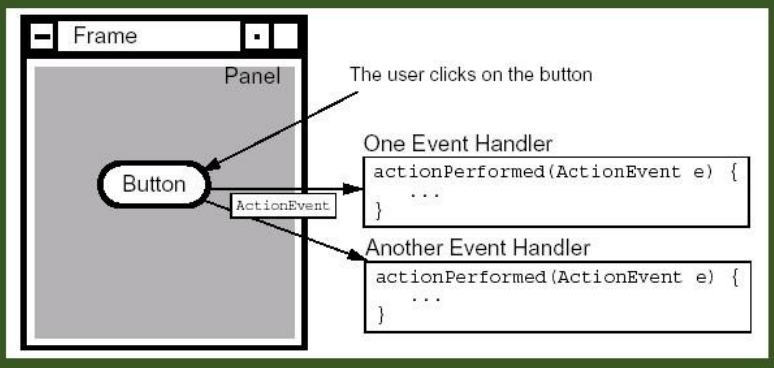
getModifiers: Devuelve cualquier modificador asociado a la acción.

Manejadores de Eventos

Es un método que recibe un objeto evento, lo descifra y procesa la interacción del usuario.

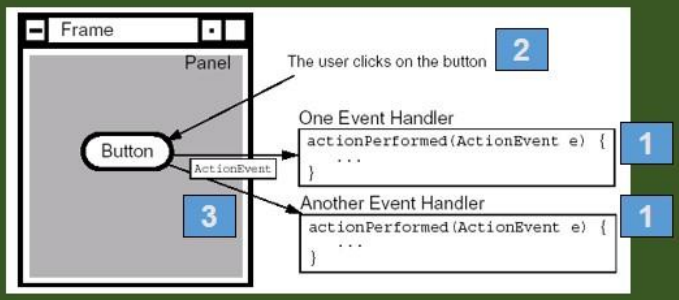
Pertenece a una clase que implementa la interfaz EventListener. Generalmente llamamos a esta clase "listener"

Un evento puede ser enviado a muchos manejadores de eventos.



Modelo Delegacional

- 1. Los listeners (Objetos EventListener) interesados se registran con el componente GUI (boton, label, Panel, etc) que ellos quieren observar.
- 2. Ocurre un evento generado por el usuario
- 3. El componente GUI delega el manejo del evento a los manejadores que se han registrado para el tipo de evento ocurrido.



Categorías de Eventos

La interfaz demanda que uno o más métodos sean definidos!!

Category	Interface Name	Methods
Action	`ActionListener`	`actionPerformed(ActionEvent)`
Item	`ItemListener`	`itemStateChanged(ItemEvent)`

Mouse	`MouseListener`	`mousePressed(MouseEvent)`
		`mouseReleased(MouseEvent)`
		`mouseEntered(MouseEvent)`
		`mouseExited(MouseEvent)`
		`mouseClicked(MouseEvent)`

Mouse motion	`MouseMotionListener`	`mouseDragged(MouseEvent)` `mouseMoved(MouseEvent)`

Key	`KeyListener`	`keyPressed(KeyEvent)`
		`keyReleased(KeyEvent)`
		`keyTyped(KeyEvent)`

Categorías de Eventos

La interfaz demanda que uno o más métodos sean definidos!!

Category	Interface Name	Methods
Focus	`FocusListener`	`focusGained(FocusEvent)` `focusLost(FocusEvent)`
Adjustment	`AdjustmentListener`	`adjustmentValueChanged(AdjustmentEvent)`

Component	`ComponentListener`	`componentMoved(ComponentEvent)`
		`componentHidden(ComponentEvent)`
		`componentResized(ComponentEvent)`
		`componentShown(ComponentEvent)`

Window	`WindowListener`	`windowClosing(WindowEvent)`
		`windowOpened(WindowEvent)`
		`windowIconified(WindowEvent)`
		`windowDeiconified(WindowEvent)`
		`windowClosed(WindowEvent)`
		`windowActivated(WindowEvent)`
		`windowDeactivated(WindowEvent)`

Container	`ContainerListener`	`componentAdded(ContainerEvent)` `componentRemoved(ContainerEvent)`
Text	`TextListener`	`textValueChanged(TextEvent)`

```
public class PrimerVentana {

   private JFrame ventana;
   private JButton bt1;

   public PrimerVentana(){

      ventana = new JFrame("Ejemplo BoxLayout");
      bt1 = new JButton("Salir 1");

   }

   public void mostrarVentana(){


      bt1.addActionListener(new ManejaBoton());


      ventana.add(bt1);
      ventana.setSize(400,300);
      ventana.setVisible(true);

   }
```

```
class ManejaBoton implements ActionListener{

      public void actionPerformed(ActionEvent e){

         System.out.println(e.getActionCommand());
         System.exit(0);

      }
      }
public static void main(String...arg){

      PrimerVentana pv = new PrimerVentana();
      pv.mostrarVentana();
   }

}
```

ManejaBoton es la clase manejadora del evento click Se registra para JButton

getActionCommand devuelve el nombre del comando asociado a la acción