

Tecnicatura Superior en Programación

LABORATORIO III

UNIDAD V

INTERFACES GRÁFICAS

UNIDAD V: MANEJO DE INTERFAZ GRÁFICA

BIBLIOTECA JFC

Para diseñar aplicaciones que utilicen interfaces gráficas (ventanas con controles, como etiquetas, cajas de texto, botones, barras de desplazamiento, etc.), Java proporciona una biblioteca de clases denominada **JFC (Java Foundation Classes -clases base de Java)**.

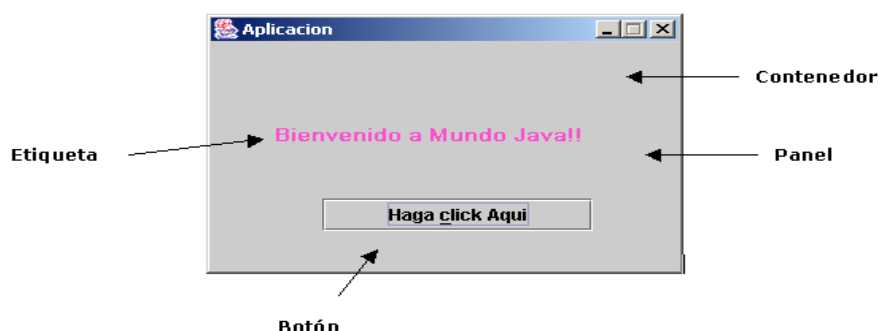
Actualmente bajo esta denominación se agrupan las siguientes APIs (interfaces para programación de aplicaciones):

- **Swing**. Conjunto de componentes escritos en Java para diseñar interfaces gráficas de usuario que se ejecutan uniformemente en cualquier plataforma nativa que soporta la máquina virtual de Java.
 - **AWT** (Abstract Window Toolkit - kit de herramientas de ventanas abstractas). Grupo de componentes, para diseñar interfaces gráficas de usuario, común a todas las plataformas, pero escritos específicamente para cada plataforma en código nativo; esto es, no están escritos en Java. Este grupo ha sido sustituido en gran medida por el conjunto de componentes Swing; muchos de éstos heredan de sus correspondientes componentes AWT.
- Accesibilidad. Ofrece soporte para usuarios con limitaciones.
 - Java 2D. Permite incorporar en los programas, gráficos 2D de alta calidad, texto e imágenes.
 - Soporte para arrastrar y colocar (drag and drop). Permite la transferencia de datos entre aplicaciones mediante la simple operación de arrastrarlos hasta el lugar de destino.

Swing o AWT? La gran diferencia entre los componentes Swing y los componentes AWT es que los primeros están implementados absolutamente con código no nativo, lo que los hace independientes de la plataforma, razón que justifica sobradamente su utilización. Además, proporcionan más capacidades que los componentes AWT. Los componentes Swing se pueden identificar porque su nombre empieza por J; por ejemplo, el componente AWTButton tiene su correspondiente componente Swing JButton. Los componentes AWT se localizan en el paquete java.awt y los componentes Swing en el paquete javax.swing. Todos los componentes Swing son subclases de la clase **JComponent**.

DISEÑO DE LA INTERFAZ GRÁFICA

Nuestro siguiente paso consistirá en añadir a la ventana de la aplicación los componentes Swing necesarios(). Este proceso requiere crear los componentes y colocarlos en el contenedor aportado por el propio formulario o en otros que nosotros creamos expresamente. Asimismo, la colocación de los componentes en un contenedor está supeditada a un administrador de diseño. Todo ello lo veremos detalladamente a continuación.





Como se muestra en la figura un aplicación gráfica consta fundamentalmente de:

- ☐ Un contenedor o marco.(**JFrame**)
- ☐ Un panel o administrador de diseño para la incorporación de los componentes Swing.
- ☐ Los controles (componentes **Swing**), como muestra la figura etiquetas, botones etc.

Una vez finalizado con el desarrollo de la parte visual de nuestra aplicación, deberemos ocuparnos del aspecto funcional del mismo.

COMPONENTES SWING MÁS COMUNES

Hay componentes Swing para cada uno de los elementos que usted ya ha visto más de una vez en alguna ventana de la interfaz gráfica de su sistema Windows, UNIX u otro. A continuación, se muestra de forma resumida, una lista de los más comunes:

- Etiquetas. Se implementan a partir de la clase **JLabel**.
- Botones. Se implementan a partir de la clase **JButton**.
- Cajas de texto. Se implementan a partir de la clase **JPasswordField** las de una sola línea de texto, y a partir de **JTextArea** las de varias líneas.
- Casillas de verificación. Se implementan a partir de la clase **JCheckBox**.
- Botones de opción. Se implementan a partir de la clase **JRadioButton**.
- Listas. Se implementan a partir de la clase **JList**.

Estos componentes enumerados en el apartado anterior son las más comunes de encontrar en cualquier aplicación, pero existen muchos otros más que no enumeramos. Y los mismos se localizan en en el paquete **javax.swing**.

A continuación se puede observar en la jerarquía de clases de Java:

- java.lang.Object
 - java.awt.Component
 - java.awt.Container
 - javax.swing.JComponent
 - java.awt. Panel
 - java.applet.Applet
 - javax.swing. JApplet
 - java.awt.Window
 - java.awt. Dialog
 - javax.swing. JDialog
 - java.awt.Frame
 - javax.swing. JFrame
 - javax.swing. JWindow

JWindow: es una ventana sin barra de título y sin los botones que permiten su manipulación, que puede visualizarse en cualquier sitio del escritorio.

JFrame es una ventana con barra de título y con los botones que permiten su manipulación, que puede visualizarse en cualquier sitio del escritorio.

JDialog permite visualizar una caja de diálogo.

JApplet permite crear un applet swing: programa que visualiza una interfaz gráfica en el contexto de una página Web.

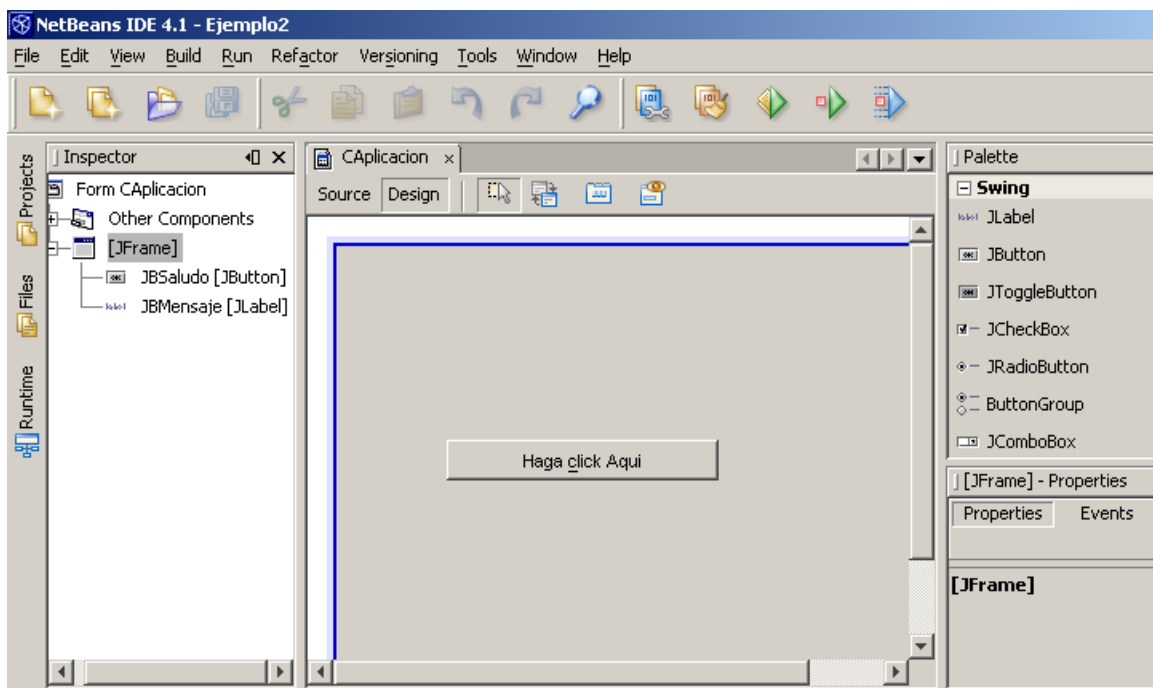
Crear un componente Swing

La forma de crear un componente Swing no difiere en nada de cómo lo hacemos con un objeto de cualquier otra clase. Se crea el componente invocando al constructor de su clase y se inician las propiedades del mismo invocando a los métodos correspondientes.

El siguiente ejemplo crea un botón de pulsación titulado "Haga clic aquí", establece como tecla de acceso la "c" y le asigna una descripción abreviada. Más tarde veremos cómo añadirlo a un contenedor.

```
// Crear un botón referenciado por jBtSaludo
jBtSaludo = new JButtonC("Haga clic aquí");
// Establecer como tecla de acceso la C. Entonces, pulsar Alt+C
// será equivalente a hacer clic sobre el botón.
jBtSaludo.setMnemonic('c');
// Asignar al botón una descripción abreviada
jBtSaludo.setToolTipText("botón de pulsación");
```

El proceso de añadir los componentes resulta muy sencillo si utilizamos un entorno de desarrollo como **netBeans de Sun Microsystems**. Simplemente tendríamos que tomar los componentes de una paleta y dibujarlos sobre el formulario utilizando el ratón. Esto haría que se añadiera automáticamente todo el código descrito anteriormente.



Como se visualiza en la figura anterior (Entorno NetBeans) en la sección de la derecha se visualiza la Paleta con todos los componentes Swing disponibles, para el diseño de una aplicación.

Una vez que colocamos los componentes en la ventana (**JFrame**), se generará automáticamente la creación de los mismos.



CONTENEDORES

Los contenedores son componentes Swing utilizados para ubicar otros componentes. Una ventana es un contenedor Swing que a continuación vamos a desarrollar. Se trata de una ventana con una etiqueta y un botón; estos componentes pueden estar sobre el contenedor definido por la propia ventana o en cualquier otro que nosotros creamos y añadamos a dicha ventana.

En general, para ayudarnos a colocar adecuadamente los componentes, es posible utilizar una jerarquía de contenedores. La raíz de esa jerarquía siempre será el contenedor del nivel superior definido por el marco de la ventana. Pertenecen a este nivel los contenedores definidos por **JFrame**, **JDialog** y **JApplet**.

Cada contenedor define un componente denominado panel. Así, un contenedor del nivel superior en la jerarquía, define un panel denominado panel raíz. Se trata de un panel de contenido que permite ubicar, además de controles, otros paneles, formando así una jerarquía de paneles. Para acceder al panel raíz de una ventana, ésta debe invocar a su método `getContentPane`. Por ejemplo, la siguiente sentencia añade el objeto `JPanel` al panel raíz de la ventana correspondiente:

```
getContentPane().add(jPnPanel , BorderLayout.GENTER);
```

En general, un panel de contenido es un objeto de la clase **JPanel**. A diferencia del panel raíz, un objeto JPanel se corresponde con un contenedor de un nivel intermedio; su propósito es simplificar la colocación de controles. Un panel de contenido puede incluir a su vez otros paneles de contenido formando así una jerarquía de paneles. Como ejemplo, la siguiente sentencia crea un objeto `Jpanel`:

```
JPanel jPnPanel = new JPanel();
```

Para añadir un componente a un panel, utilizaremos su método **add**. Por ejemplo, el siguiente código añade una etiqueta (objeto de la clase `JLabel`) y un botón (objeto de la clase `JButton`) al panel referenciado por `jPnPanel`:

```
jPnPanel.add(etiqueta);  
jPnPanel.add(botón);
```

CAJAS DE DIÁLOGO

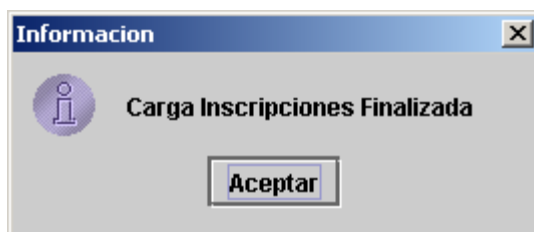
En Java tenemos principalmente dos tipos de ventana: **JFrame** y **JDialog**.

Cuando una aplicación, que muestra una interfaz gráfica, en un instante determinado de su ejecución necesita para poder continuar, aceptar nuevos datos o bien visualizarlos. Para estos casos lo que podemos hacer es utilizar una ventana que reciben el nombre de *Cajas de Diálogo*.

La funcionalidad de proporcionar cajas de diálogos es proporcionada por las clases **JOptionPane** y **JDialog**.

Hay tres formas de añadir cajas de diálogo a una aplicación.

- ❑ Cajas de Diálogo predefinidas: Estas cajas de diálogo creadas por medio de los métodos proporcionados por la clase **JOptionPane** de la biblioteca JFC, como por ejemplo **showMessageDialog**.



Ejemplo Caja de Diálogo Predefinida.(**showMessageDialog**)

- ❑ Cajas de Diálogos personalizadas: Son cajas de diálogos hecha a medida, para la cual la biblioteca JFC proporciona la clase **JDialog**.
- ❑ Cajas de Diálogo estándar: Son cajas de diálogos muy comunes: por ejemplo la caja de diálogo *Abrir* o *Guardar* proporcionada por la clase **JFileChooser** o el diálogo Color proporcionado por la clase **JColorChooser**.

A diferencia del objeto **JFrame** (ventana primaria) una caja de diálogo es una ventana secundaria que depende de otra.

ADMINISTRADORES DE DISEÑO

"Un administrador de diseño está pensado para mostrar varios componentes a la vez en un orden preestablecido".

De forma predeterminada cada contenedor tiene asignado un administrador de diseño. Por ejemplo, los contenedores del nivel superior tienen asignado de forma predeterminada el administrador de diseño **BorderLayout**.

Swing proporciona seis administradores de diseño:

BorderLayout. Diseño con límites. Divide un contenedor en cinco secciones denominadas: norte, sur, este, oeste y centro. Es el administrador de diseño que tienen asignado de forma predeterminada los contenedores del nivel superior. Los componentes se colocarán en una sección u otra según decidamos.

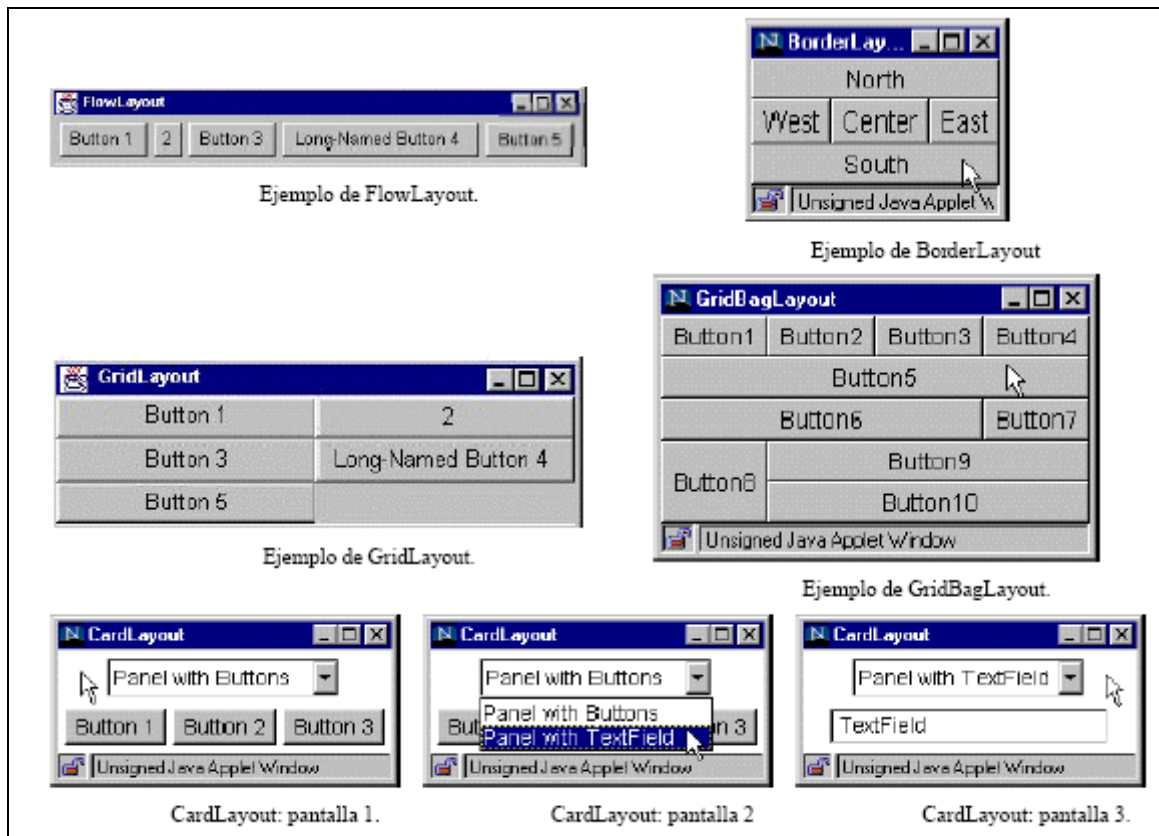
GridLayout. Diseño por rejilla. Coloca los componentes en el contenedor en filas y columnas.

GridBagLayout. Diseño tipo rejilla. Coloca los componentes en el contenedor en filas y columnas. A diferencia de GridLayout, permite que un componente pueda ocupar más de una fila y/o columna.

CardLayout. Diseño por paneles. Este administrador permite colocar en el contenedor grupos diferentes de componentes en instantes diferentes de la ejecución (similar a los paneles con pestañas).

BoxLayout. Diseño en caja. Coloca los componentes en el contenedor en una única fila o columna, ajustándose al espacio que haya.

FlowLayout. Diseño en flujo. Coloca los componentes en el contenedor de izquierda a derecha (igual que se coloca el texto en un párrafo). Es el administrador de diseño asignado de forma predeterminada a los contenedores de un nivel intermedio.

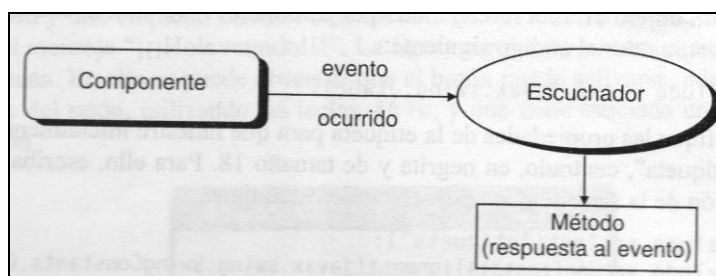


MANEJO DE EVENTOS.

Cuando sobre un componente ocurra algún evento se espera que suceda algo. Lógicamente ese algo hay que programarlo y para poder hacerlo, hay que saber cómo manejar ese evento.

Los eventos que se producen sobre un componente y se **manipulan a través de objetos denominamos manejadores** o escuchadores de esos eventos. Según esto, un manejador de eventos es un objeto en el que un componente delega la tarea de manipular un tipo particular de eventos.

En la figura siguiente puede ver que cuando sobre un componente se produce un evento, un manejador de eventos vinculado con el componente se encarga de analizar qué evento ocurrió para responder al mismo ejecutando el método adecuado:



Un componente tendrá asociados tantos manejadores o escuchadores de eventos como tipos de eventos tenga que manejar.

Java 2 proporciona varios manejadores de eventos, cada uno de los cuales maneja un tipo particular de eventos. Por ejemplo, **WindowListener** permite a una ventana responder a las acciones que ocurren sobre ella (eventos de tipo WindowEvent), como minimizarla, abrirla, moverla, etc., y **ActionListener** permite a un componente responder a las acciones que ocurren sobre él (eventos de tipo ActionEvent), como un clic sobre un botón.

RELACIÓN ENTRE COMPONENTES Y EVENTOS

La Tabla muestra los componentes y los eventos específicos de cada uno de ellos, así como una breve explicación de en qué consiste cada tipo de evento.

Component	Eventos generados	Significado
Button	ActionEvent	Clicar en el botón
Checkbox	ItemEvent	Seleccionar o deseleccionar un ítem
CheckboxMenuItem	ItemEvent	Seleccionar o deseleccionar un ítem
Choice	ItemEvent	Seleccionar o deseleccionar un ítem
Component	ComponentEvent	Mover, cambiar tamaño, mostrar u ocultar un componente
	FocusEvent	Obtener o perder el focus
	KeyEvent	Pulsar o soltar una tecla
	MouseEvent	Pulsar o soltar un botón del ratón; entrar o salir de un componente; mover o arrastrar el ratón (tener en cuenta que este evento tiene dos Listener)
Container	ContainerEvent	Añadir o eliminar un componente de un container
List	ActionEvent	Hacer doble click sobre un ítem de la lista
	ItemEvent	Seleccionar o deseleccionar un ítem de la lista
MenuItem	ActionEvent	Seleccionar un ítem de un menú
Scrollbar	AdjustementEvent	Cambiar el valor de la scrollbar
TextComponent	TextEvent	Cambiar el texto
TextField	ActionEvent	Terminar de editar un texto pulsando Intro
Window	WindowEvent	Acciones sobre una ventana: abrir, cerrar, iconizar, restablecer e iniciar el cierre



ASIGNAR MANEJADORES DE EVENTOS A UN OBJETO.

Un objeto, generalmente un componente, tendrá asociados tantos manejadores de eventos como tipos de eventos tenga que manejar.

Por ejemplo, consideremos un botón `JbtSaludo`. Cada vez que el usuario haga clic sobre este botón, se generará un evento de acción que podrá ser recogido por un manejador de este tipo de eventos, si es que tiene uno asociado.

¿Cómo vinculamos un manejador de eventos con un componente? Lo haremos de la siguiente forma:

1. Creamos una clase **que implemente la interfaz que aporta los métodos que permiten responder al tipo de eventos que tratamos de manejar**. En nuestro caso se trata de la interfaz ***ActionListener***.

```
class ManejadorEvento implements java.awt.event.ActionListener {  
    public void actionPerformed(java.awt.event.ActionEvent evt)  
    {  
        // Responder al evento evt  
    }  
}
```

La clase `ManejadorEvento` redefine el método `actionPerformed` que se ejecutará como respuesta al evento `evt` producido.

2. Construimos un objeto de esa clase; se trata del manejador de eventos.

```
java.awt.event.ActionListener al = new ManejadorEvento( );
```

3. Vinculamos el manejador de eventos con el componente, en este caso al botón.

```
jbtSaludo.addActionListener(al);
```

Estas operaciones pueden escribirse de una forma más simplificada así:

```
java.awt.event.ActionListener al = new java.awt.event.ActionListener() { public void  
    actionPerformed(java.awt.event.ActionEvent evt)  
    {  
        // Responder al evento evt  
    }  
}; jbtSaludo.addActionListener(al);
```

El código anterior generará una clase anónima. También se puede escribir así:

```
jbtSaludo.addActionLi stener(new java.awt.event.ActionLi stenerí) { public void  
    actionPerformed(java.awt.event.ActionEvent evt)  
    {  
        // Responder al evento evt  
    }  
}
```



Observe que el método **actionPerformed** recibe como argumento un evento de tipo **ActionEvent**. Dicho argumento no es más que un objeto que proporciona información tal como: componente que originó el evento, si se pulsó simultáneamente alguna tecla (Alt, Ctrl, Shift), tipo de evento, orden asociada con ese evento, etc. Tal información la utilizaremos para responder a dicho evento. Por ejemplo, si hubiera más de un componente asociado con el manejador de eventos del ejemplo anterior, podríamos proceder así:

```
public void actionPerformed (java.awt.event.ActionEvent evt)
```

```
{ Object obj=evt.getSource(); //objeto que produjo el evento.
```

```
  if(obj==jbtSaludo) //el objeto es jtbSaludo??
```

```
    //Responder al evento.
```

```
  else if (...)
```

```
    //}
```

Otra técnica que puede ser útil es proceder en función de la clase del componente. Por ejemplo:

```
public void actionPerformed (java.awt.event.ActionEvent evt)
```

```
{ Object obj = evt.getSource(); // objeto que produjo el evento  
  // ¿El objeto es de la clase JButton?
```

```
    if (obj instanceof javax.swing.JButton) // Responder al evento evt
```

```
    // ¿El objeto es de la clase JLabel?
```

```
    else if (obj instanceof javax.swing.JLabel)
```

Además del método **getSource**, la clase **ActionEvent** proporciona el método **getActionCommand** que permite trabajar con órdenes de acción diferentes a la predeterminada (la orden predeterminada coincide con el texto asociado con el componente: título del botón, contenido de la caja de texto, etc.). Las órdenes de acción son excepcionalmente útiles cuando se escribe un programa en el que más de un componente debería provocar la misma acción (por ejemplo, los elementos de un menú). Con este fin, componentes como el botón, la caja de texto o un elemento de un menú, proporcionan el método **setActionCommand**.

Es muy importante conocer cada paso de cómo es que se asigna un manejador de eventos a un componente, aunque en la práctica veremos que es muy sencillo.

A manera de resumen se muestra en la siguiente tabla los diferentes métodos relacionados con cada evento a través de una interfaz Listener.



Evento	Interface Listener	Métodos de Listener
ActionEvent	ActionListener	actionPerformed()
AdjustementEvent	AdjustementListener	adjustementValueChanged()
ComponentEvent	ComponentListener	componentHidden(), componentMoved(), componentResized(), componentShown()
ContainerEvent	ContainerListener	componentAdded(), componentRemoved()
FocusEvent	FocusListener	focusGained(), focusLost()
ItemEvent	ItemListener	itemStateChanged()
KeyEvent	KeyListener	keyPressed(), keyReleased(), keyTyped()
MouseEvent	MouseListener	mouseClicked(), mouseEntered(), mouseExited(), mousePressed(), mouseReleased()
	MouseMotionListener	mouseDragged(), mouseMoved()
TextEvent	TextListener	textValueChanged()
WindowEvent	WindowListener	windowActivated(), windowDeactivated(), windowClosed(), windowClosing(), windowIconified(), windowDeiconified(), windowOpened()



CLASES ADAPTER

Java proporciona ayudas para definir los métodos declarados en las **interfaces Listener**. Una de estas ayudas son las **clases Adapter**, que existen para cada una de las interfaces **Listener** que tienen más de un método. Su nombre se construye a partir del nombre de la interface, sustituyendo la palabra "Listener" por "Adapter".

Hay 7 clases Adapter: ComponentAdapter, ContainerAdapter, FocusAdapter, KeyAdapter, MouseAdapter, MouseMotionAdapter y WindowAdapter.

Las clases **Adapter** derivan de **Object**, y son clases predefinidas que contienen definiciones vacías para todos los métodos de la interface. Para crear un objeto que responda al evento, en vez de crear una clase que implemente la interface Listener, basta crear una clase que derive de la clase Adapter correspondiente, y redefina sólo los métodos de interés.

RESPONDER A LOS EVENTOS.

Una vez finalizado el diseño de la interfaz gráfica, hay que asignar a cada uno de los componentes la tarea que debe desempeñar.

Por ejemplo, con respecto a la aplicación más sencilla mostrar el típico mensaje "Hola Mundo" ¿qué tiene que suceder cuando el usuario haga clic en el botón? Pues que en la etiqueta debe aparecer el mensaje "iiiHola mundo!!!".

Esto implica añadir un manejador que manipule el evento clic del botón. Para ello, añada el siguiente código al método **initComponents**:

```
private void initComponents()
{
    //.....

    JbtSaludo.setToolTipText("botón de pulsación");
    JbtSaludo.setText("Haga Click Aquí!");

    JbtSaludo.addActionListener(new java.awt.event.ActionListener()
    {
        public void actionPerformed(java.awt.event.ActionEvent evt)
        {
            JbtSaludoActionPerformed(evt);
        }
    });
}
```

```
private void JbtSaludoActionPerformed(java.awt.event.ActionEvent evt)
{
    //Añadir el código aquí
}
```



Entonces, lo que ahora tenemos que hacer es escribir en el método **jBtSaludoActionPerformed** el código que debe ejecutarse para responder al evento clic de este botón. En nuestro caso, lo que queremos es que la etiqueta muestre el mensaje "iiiHola mundo!!".

Sucesivas pulsaciones mostrarán el mismo mensaje pero en un color diferente. Según esto, el método podría escribirse así:

```
private void JBtSaludoActionPerformed(java.awt.event.ActionEvent evt)
{
    float rojo = (float)Math.random() ;
    float verde = (float)Math.random() ;
    float azul = (float)Math.random() ;
    jEtSaludo.setForeground(new java.awt.Color(rojo, verde, azul));
    jEtSaludo.setText("iiiHola mundo!!!");
}
```

El método **setForeground** establece el color del primer plano; en nuestro caso del texto. Observe que este método recibe como argumento un objeto de la clase **Color** que especifica las componentes rojo, verde y azul del color RGB que representa dicho objeto. Dichas componentes son obtenidas de forma aleatoria invocando al método **random**.

MÉTODOS PRINCIPALES DE LOS COMPONENTES SWING MÁS COMUNES.

CLASE WINDOW

Los objetos de la clase **Window** son ventanas de máximo nivel, pero sin bordes y sin barra de menús.

En realidad son más interesantes las clases que derivan de ella: **Frame y Dialog**. Los métodos más útiles, por ser heredados por las clases **Frame** y **Dialog**.

Métodos de Window	Función que realizan
toFront(), toBack()	Para desplazar la ventana hacia adelante y hacia atrás en la pantalla
show()	Muestra la ventana y la trae a primer plano
pack()	Hace que los componentes se reajusten al tamaño preferido

CLASE FRAME

Es una ventana con un borde y que puede tener una barra de menús. La Tabla muestra algunos métodos más utilizados de la clase **Frame**.



Métodos de Frame	Función que realiza
Frame(), Frame(String title)	Constructores de Frame
String getTitle(), setTitle(String)	Obtienen o determinan el título de la ventana
MenuBar getMenuBar(), setMenuBar(MenuBar), remove(MenuComponent)	Permite obtener, establecer o eliminar la barra de menús
Image getIconImage(), setIconImage(Image)	Obtienen o determinan el icono que aparecerá en la barra de títulos
setResizable(boolean), boolean isResizable()	Determinan o chequean si se puede cambiar el tamaño
dispose()	Método que libera los recursos utilizados en una ventana. Todos los componentes de la ventana son destruidos.

CLASE DIALOG

Un **Dialog** es una ventana que depende de otra ventana (**de una Frame**). Si una Frame se cierra, se cierran también los Dialog que dependen de ella; si se iconifica, sus Dialog desaparecen; si se restablece, sus Dialog aparecen de nuevo. Este comportamiento se obtiene de forma automática.

Un **Dialog modal** requiere la atención inmediata del usuario: no se puede hacer ninguna otra cosa hasta no haber cerrado el Dialog. Por defecto, los Dialogs son no modales. La Tabla muestra los métodos más importantes de la clase Dialog. Se pueden utilizar también los métodos heredados de sus super-clases.

Métodos de Dialog	Función que realiza
Dialog(Frame fr), Dialog(Frame fr, boolean mod), Dialog(Frame fr, String title), Dialog(Frame fr, String title, boolean mod)	Constructores
String getTitle(), setTitle(String)	Permite obtener o determinar el título
boolean isModal(), setModal(boolean)	Pregunta o determina si el Dialog es modal o no
boolean isResizable(), setResizable(boolean)	Pregunta o determina si se puede cambiar el tamaño
show()	Muestra y trae a primer plano el Dialog

CLASE LABEL

La clase Label introduce en un container **un texto no seleccionable y no editable**, que por defecto se alinea por la izquierda. La clase Label define las constantes **Label.CENTER**, **Label.LEFT** y **Label.RIGHT** para determinar la alineación del texto. La Tabla muestra algunos métodos de esta clase:

Métodos de Label	Función que realizan
Label(String lbl), Label(String lbl, int align)	Constructores de Label
setAlignment(int align), int getAlignment()	Establecer u obtener la alineación del texto
setText(String txt), String getText()	Establecer u obtener el texto del Label



La elección del font, de los colores, del tamaño y posición de Label se realiza con los métodos heredados de la clase Component: setFont(Font f), setForeground(Color), setBackground(Color), setSize(int, int), setLocation(int, int), setVisible(boolean), etc.

La clase Label no tiene más eventos que los de su super-clase Component.

CLASE BUTTON

Un componente *Button* puede recibir seis tipos de eventos, **lo más importante es que al clicar sobre él se genera un evento de la clase ActionEvent.**

Se puede cambiar el texto y la font que aparecen en el Button, así como el foreground y background color. También se puede establecer que esté activado o no. La Tabla muestra los métodos más importantes de la clase Button.

Métodos de la clase Button	Función que realiza
Button(String label) y Button()	Constructores
setLabel(String str), String getLabel()	Permite establecer u obtener la etiqueta del Button
addActionListener(ActionListener al), removeActionListener(ActionListener al)	Permite registrar el objeto que gestionará los eventos, que deberá implementar ActionListener
setActionCommand(String cmd), String getActionCommand()	Establece y recupera un nombre para el objeto Button independiente del label y del idioma

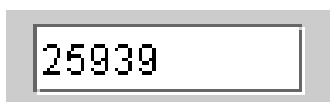


Ejemplo JButton

CLASES TEXTAREA Y TEXTFIELD

Ambas componentes heredan de la clase **TextComponent** y **muestran texto seleccionable y editable.** La diferencia principal es que **TextField sólo puede tener una línea**, mientras que **TextArea puede tener varias líneas**. Además, TextArea ofrece posibilidades de edición de texto adicionales.

Se pueden especificar el font y los colores de foreground y background. Sólo la clase **TextField** genera **ActionEvents**, pero como las dos heredan de la clase **TextComponent** ambas pueden recibir TextEvents. La Tabla muestra algunos métodos de las clases TextComponent, TextField y TextArea.



Ejemplo de TextField



Métodos heredados de TextComponent	Función que realizan
<code>String getText()</code> y <code>setText(String str)</code>	Permiten establecer u obtener el texto del componente
<code>setEditable(boolean b)</code> , <code>boolean isEditable()</code>	Hace que el texto sea editable o pregunta por ello
<code>setCaretPosition(int n)</code> , <code>int getCaretPosition()</code>	Fija la posición del punto de inserción o la obtiene
<code>String getSelectedText()</code> , <code>int getSelectionStart()</code> y <code>int getSelectionEnd()</code>	Obtiene el texto seleccionado y el comienzo y el final de la selección
<code>selectAll()</code> , <code>select(int start, int end)</code>	Selecciona todo o parte del texto
Métodos de TextField	Función que realizan
<code>TextField()</code> , <code>TextField(int ncol)</code> , <code>TextField(String s)</code> , <code>TextField(String s, int ncol)</code>	Constructores de TextField
<code>int getColumns()</code> , <code>setColumns(int)</code>	Obtiene o establece el número de columnas del TextField
<code>setEchoChar(char c)</code> , <code>char getEchoChar()</code> , <code>boolean echoCharIsSet()</code>	Establece, obtiene o pregunta por el carácter utilizado para passwords, de forma que no se pueda leer lo tecleado por el usuario
Métodos de TextArea	Función que realizan
<code>TextArea()</code> , <code>TextArea(int nfil, int ncol)</code> , <code>TextArea(String text)</code> , <code>TextArea(String text, int nfil, int ncol)</code>	Constructores de TextArea
<code>setRows(int)</code> , <code>setColumns(int)</code> , <code>int getRows()</code> , <code>int getColumns()</code>	Establecer y/u obtener los números de filas y de columnas
<code>append(String str)</code> , <code>insert(String str, int pos)</code> , <code>replaceRange(String s, int i, int f)</code>	Añadir texto al final, insertarlo en una posición determinada y reemplazar un texto determinado

CLASE LIST

La clase **List** viene definida por una zona de pantalla con varias líneas, de las que se muestran sólo algunas, y entre las que **se puede hacer una selección simple o múltiple**. Las List generan eventos de la clase **ActionEvents** (al clicar dos veces sobre un item o al pulsar return) e **ItemEvents** (al seleccionar o deseleccionar un item). Al gestionar el evento **ItemEvent** se puede preguntar si el usuario estaba pulsando a la vez alguna tecla (Alt, Ctrl, Shift), por ejemplo para hacer una selección múltiple.

Las List se diferencian de las Choices en que muestran varios items a la vez y que permiten hacer selecciones múltiples. La Tabla muestra los principales métodos de la clase List.



Métodos de List	Función que realiza
List(), List(int nl), List(int nl, boolean mult)	Constructor: por defecto una línea y selección simple
add(String), add(String, int), addItem(String), addItem(String, int)	Añadir un item. Por defecto se añaden al final
addActionListener(ActionListener), addItemListener(ItemListener)	Registra los objetos que gestionarán los dos tipos de eventos soportados
insert(String, int)	Inserta un nuevo elemento en la lista
replaceItem(String, int)	Sustituye el item en posición int por el String
delItem(int), remove(int), remove(String), removeAll()	Eliminar uno o todos los items de la lista
int getItemCount(), int getRows()	Obtener el número de items o el número de items visibles
String getItem(int), String[] getItems()	Obtiene uno o todos los elementos de la lista
int getSelectedIndex(), String getSelectedItem(), int[] getSelectedIndexes(), String[] getSelectedItems()	Obtiene el/los elementos seleccionados
select(int), deselect(int)	Selecciona o elimina la selección de un elemento
boolean isIndexSelected(int), boolean isItemSelected(String)	Indica si un elemento está seleccionado o no
boolean isMultipleMode(), setMultipleMode(boolean)	Pregunta o establece el modo de selección múltiple
int getVisibleIndex(), makeVisible(int)	Indicar o establecer si un item es visible

COMPONENT CHECKBOX Y CLASE CHECKBOXGROUP.

Los objetos de la clase **Checkbox** son botones de opción o de selección con dos posibles valores: on y off. Al cambiar la selección de un **Checkbox** se produce un **ItemEvent**.



Ejemplo CheckBoxes

La clase **CheckboxGroup** permite la opción de agrupar varios Checkbox de modo que uno y sólo uno esté en on (al comienzo puede que todos estén en off).

Cuando el usuario actúa sobre un objeto Checkbox se ejecuta el método **itemStateChanged()**, que es el único método de la interface **ItemListener**. Si hay varias checkboxes cuyos eventos se gestionan en un mismo objeto y se quiere saber cuál es la que ha recibido el evento, se puede utilizar el método **getSource()** del evento **EventObject**. También se puede utilizar el método **getItem()** de **ItemEvent**, cuyo valor de retorno es un **Object** que contiene el label del componente (para convertirlo a **String** habría que hacer un cast).

Para crear un grupo o conjunto de botones de opción (de forma que uno y sólo uno pueda estar activado), se debe crear un objeto de la clase **CheckboxGroup**. Este objeto no tiene datos: simplemente sirve como identificador del grupo. Cuando se crean los objetos **Checkbox** se pasa a los constructores el objeto **CheckboxGroup** del grupo al que se quiere que pertenezcan.



Cuando se selecciona un Checkbox de un grupo se producen dos eventos: uno por el elemento que se ha seleccionado y otro por haber perdido la selección el elemento que estaba seleccionado anteriormente. Al hablar de evento ItemEvent y del método itemStateChanged() se verán métodos para determinar los checkboxes que han sido seleccionados o que han perdido la selección.

Métodos de Checkbox	Función que realizan
Checkbox(), Checkbox(String), Checkbox(String, boolean), Checkbox(String, boolean, CheckboxGroup), Checkbox(String, CheckboxGroup, boolean)	Constructores de Checkbox. Algunos permiten establecer la etiqueta, si está o no seleccionado y si pertenece a un grupo
addItemListener(ItemListener), removeItemListener(ItemListener)	Registra o elimina los objetos que gestionarán los eventos ItemEvent
setLabel(String), String getLabel()	Establece u obtiene la etiqueta del componente
setState(boolean), boolean getState()	Establece u obtiene el estado (true o false, según esté seleccionado o no)
setCheckboxGroup(CheckboxGroup), CheckboxGroup getCheckboxGroup()	Establece u obtiene el grupo al que pertenece el Checkbox
Métodos de CheckboxGroup	Función que realizan
CheckboxGroup()	Constructores de CheckboxGroup:
Checkbox getSelectedCheckbox(), setSelectedCheckbox(Checkbox box)	Obtiene o establece el componente seleccionado de un grupo:

CLASE JComboBox (LISTA DESPLEGABLES)

Una lista desplegable es un objeto de la clase JComboBox. La diferencia entre una lista simple y desplegable es que la desplegable es una combinación de una lista simple y una caja de texto, permite la selección de un solo elemento cada vez y no necesita un panel de desplazamiento.

A continuación se muestra un gráfico ejemplificando este componente.



Ejemplo JComboBox

PARTE PRACTICA

EJERCICIO 1

Diseñar la siguientes interfaces:

La pantalla que se visualiza en la figura 1, es la pantalla principal, debe incluir un menú con dos opciones: Carga, Consulta Archivo.(como se muestra en la figura 1)

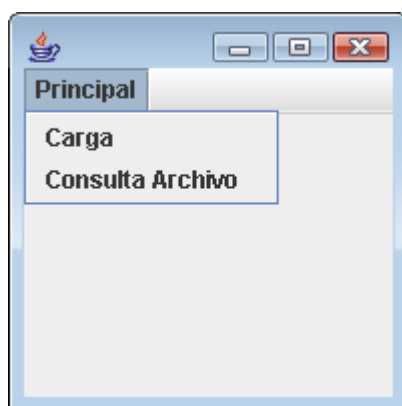


Figura 1.

Para el diseño de esta pantalla, debe tener en cuenta la pantalla que se muestra a continuación. En la misma se debe cargar los datos de alumnos y cargarlos en un archivo. El botón salir, facilita abandonar esta pantalla. La misma debe ser invocada, cuando se presiona la opción del menú **Carga**.

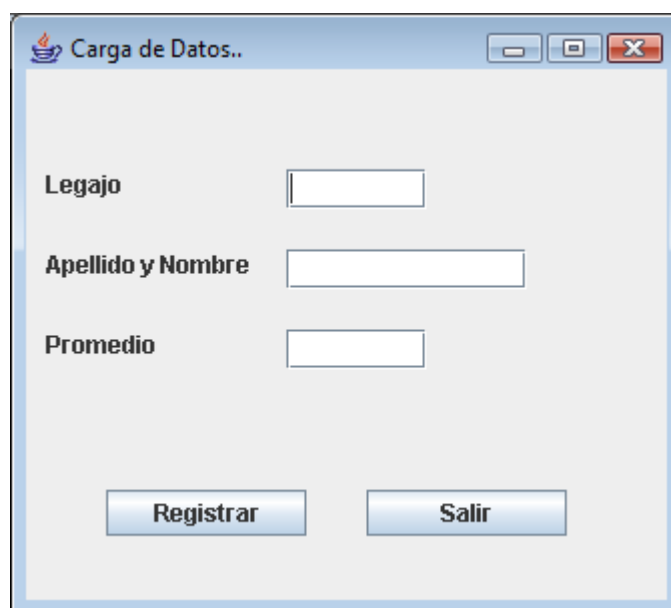


Figura 2.

La próxima pantalla debe mostrar el promedio general del curso, y deberá visualizarse cuando el usuario presione la opción **Consulta Archivo**. Para el diseño de la misma, tomar como guía la figura3.

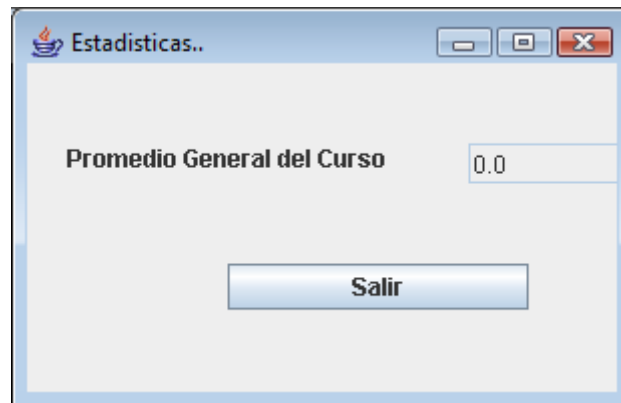


Figura 3.

Resolución.

```
//Alumno.java
import java.io.*;

public class Alumno implements Serializable {

    private int legajo;
    private String nombre;
    private float promedio;

    public Alumno(int leg,String nom,float prom)
    {
        legajo=leg;
        nombre=nom;
        promedio=prom;
    }

    public void setLegajo(int leg)
    {legajo=leg;}

    public void setNombre(String nom)
    {nombre=nom;}

    public void setPromedio(float prom)
    {promedio=prom;}

    public int getLegajo()
    {return legajo;}

    public String getNombre()
    {return nombre;}
```



```
        public float getPromedio()
        {return promedio;}

        public String toString()
        {
            String aux="";
            aux=legajo+" "+nombre+" "+promedio;
            return aux;
        }
    }

//ArchivoAlumnos.java
import java.io.*;
public class ArchivoAlumnos {

    private FileOutputStream fileOut;
    private FileInputStream fileIn;
    private ObjectOutputStream output;
    private ObjectInputStream input;
    public ArchivoAlumnos()
    {
        fileOut=null;
        fileIn=null;
        output=null;
        input=null; }

    public void AbrirModoLectura() throws IOException
    { fileIn=new FileInputStream("c:\\Temp\\prueba.dat");
      input=new ObjectInputStream(fileIn);
    }

    public void AbrirModoEscritura() throws IOException
    { fileOut=new FileOutputStream("c:\\Temp\\prueba.dat");
      output=new ObjectOutputStream(fileOut);
    }

    public void CerrarModoLectura() throws IOException
    { if(input!=null)
      { input.close();}
    }

    public void CerrarEscritura() throws IOException
    { if(output!=null)
      { output.close();}
```



```
}

public String Leer() throws IOException,ClassNotFoundException
{
    Alumno alu=null;
    String aux="";
    AbrirModoLectura();

    if(input!=null)
    {
        try
        {
            while(true)
            {
                //Lectura de Objetos
                alu=(Alumno) input.readObject();
                aux += alu.toString() + "\n";
            }
        } catch EOFException e) {} //Fin del fichero
    }

    CerrarModoLectura();

    return aux; }

//Escribir en el fichero.
public void Escribir(Alumno alu) throws IOException
{
    if(output==null)
    {
        AbrirModoEscritura();
    }
    //Escritura del Objeto
    output.writeObject(alu);
}

public float Promedio() throws IOException,ClassNotFoundException
{
    Alumno alu=null;
    float aux=0;
    int cantidad = 0;
    AbrirModoLectura();

    if(input!=null)
```



```
{ try
{
    while(true)
    {
        //Lectura de Objetos
        alu=(Alumno) input.readObject();
        //aux += alu.toString() + "\n";
        aux += alu.getPromedio();
        cantidad += 1;
    }
} catch EOFException e) {}//Fin del fichero}

}

CerrarModoLectura();
if (cantidad!=0)
    return (float)(aux / cantidad);
else
    return 0;
}

}

//CLASES GRAFICAS
//Menu.java
public class Menu extends javax.swing.JFrame {

    /** Creates new form Menu */
    public Menu() {
        initComponents();
        this.setSize(200,200);
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc=" Generated Code ">
    private void initComponents() {
        menuBar1 = new java.awt.MenuBar();
        menu1 = new java.awt.Menu();
    }
}
```



```
jMenuBar1 = new javax.swing.JMenuBar();
jMenu1 = new javax.swing.JMenu();
jMenuItem1 = new javax.swing.JMenuItem();
jMenuItem2 = new javax.swing.JMenuItem();

menuBar1.setName("dd");
menu1.setEnabled(false);
menu1.setLabel("Menu");
menu1.setName("dddd");
menu1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        menu1ActionPerformed(evt);
    }
});

menuBar1.add(menu1);

getContentPane().setLayout(null);

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
jMenu1.setText("Principal");
jMenuItem1.setText("Carga");
jMenuItem1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jMenuItem1ActionPerformed(evt);
    }
});

jMenu1.add(jMenuItem1);

jMenuItem2.setText("Consulta Archivo");
jMenuItem2.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jMenuItem2ActionPerformed(evt);
    }
});

jMenu1.add(jMenuItem2);

jMenuBar1.add(jMenu1);

setJMenuBar(jMenuBar1);
```




```
        pack();
    }
// </editor-fold>

private void jMenuItem2ActionPerformed(java.awt.event.ActionEvent evt) {
    Resultados p1=new Resultados();
    p1.setVisible(true);
}

private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt) {

    Principal p=new Principal();
    p.setTitle("Carga de Datos..");
    p.setVisible(true);
}

private void menu1ActionPerformed(java.awt.event.ActionEvent evt) {

}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Menu().setVisible(true);
        }
    });
}

// Variables declaration - do not modify
private javax.swing.JMenu jMenuItem1;
private javax.swing.JMenuBar jMenuItemBar1;
private javax.swing.JMenuItem jMenuItem1;
private javax.swing.JMenuItem jMenuItem2;
private java.awt.Menu menu1;
private java.awt.MenuBar menuBar1;
// End of variables declaration

}
```



```
//Principal.java
import java.io.*;
import javax.swing.JOptionPane;

public class Principal extends javax.swing.JFrame {

    ArchivoAlumnos arch=new ArchivoAlumnos();
    Alumno alu=null;
    int contador=0;

    /** Creates new form Principal */
    public Principal() {
        initComponents();
        this.setSize(400, 300);
        try{
            arch.AbrirModoEscritura();} catch(IOException e){;}
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc=" Generated Code ">
    private void initComponents() {

        btnRegistrar = new javax.swing.JButton();
        jTextLegajo = new javax.swing.JTextField();
        jTextNombre = new javax.swing.JTextField();
        jTextPromedio = new javax.swing.JTextField();
        jButton1 = new javax.swing.JButton();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();

        getContentPane().setLayout(null);

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        btnRegistrar.setText("Registrar");
        btnRegistrar.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                btnRegistrarActionPerformed(evt);
            }
        });

        getContentPane().add(btnRegistrar);
        btnRegistrar.setBounds(40, 210, 100, 23);
    }
}
```



```
getContentPane().add(jTextLegajo);
jTextLegajo.setBounds(130, 50, 70, 20);

getContentPane().add(jTextNombre);
jTextNombre.setBounds(130, 90, 120, 20);

getContentPane().add(jTextPromedio);
jTextPromedio.setBounds(130, 130, 70, 20);

jButton1.setText("Salir");
jButton1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton1ActionPerformed(evt);
    }
});

getContentPane().add(jButton1);
jButton1.setBounds(170, 210, 100, 23);

jLabel1.setText("Legajo");
getContentPane().add(jLabel1);
jLabel1.setBounds(10, 50, 110, 14);

jLabel2.setText("Apellido y Nombre");
getContentPane().add(jLabel2);
jLabel2.setBounds(10, 90, 130, 14);

jLabel3.setText("Promedio");
getContentPane().add(jLabel3);
jLabel3.setBounds(10, 130, 90, 14);

pack();
}
// </editor-fold>

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
    dispose();
}

private void btnRegistrarActionPerformed(java.awt.event.ActionEvent evt) {
    //Esta vacio los campos???
```



```
        if(jTextLegajo.getText().trim().length()!=0 &&
jTextNombre.getText().trim().length()!=0 && jTextPromedio.getText().trim().length()!=0)
        { // Aca escribo la logica del boton:
            //Creo el objeto Alumno
            int l=Integer.parseInt(jTextLegajo.getText());
            String nom=jTextNombre.getText();
            float p=Float.parseFloat(jTextPromedio.getText());
            alu=new Alumno(l,nom,p);
            try{
                arch.Escribir(alu);} catch(IOException e){;}
            System.out.println("Registro grabado!!!");

            //Limpiar Controles.
            jTextLegajo.setText("");
            jTextNombre.setText("");
            jTextPromedio.setText("");
        }
        else //Algun campo para el ingreso de datos esta vacio..
        { //System.out.println("Debe ingresar datos!!!");
            JOptionPane.showMessageDialog(null,"Debe ingresar
campos","Advertencia",JOptionPane.INFORMATION_MESSAGE);
        }
    }

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Principal().setVisible(true);
        }
    });
}

// Variables declaration - do not modify
private javax.swing.JButton btnRegistrar;
private javax.swing.JButton jButton1;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JTextField jTextLegajo;
private javax.swing.JTextField jTextNombre;
```



```
private javax.swing.JTextField jTextPromedio;
// End of variables declaration

}
//Resultado.java
import java.io.*;
public class Resultados extends javax.swing.JFrame {

    /** Creates new form Resultados */
    public Resultados() {
        initComponents();
        this.setSize(310, 200);
        this.setTitle("Estadísticas..");
        ArchivoAlumnos al=new ArchivoAlumnos();
        try{
            jTextField1.setText(String.valueOf(al.Promedio()));

        }

        catch(IOException e){;}
        catch(ClassNotFoundException e1){;}
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc=" Generated Code ">
    private void initComponents() {
        jLabel1 = new javax.swing.JLabel();
        jTextField1 = new javax.swing.JTextField();
        jButton1 = new javax.swing.JButton();

        getContentPane().setLayout(null);

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        jLabel1.setText("Promedio General del Curso");
        getContentPane().add(jLabel1);
        jLabel1.setBounds(20, 40, 180, 14);

        jTextField1.setEditable(false);
        getContentPane().add(jTextField1);
```



```
        jTextField1.setBounds(220, 40, 90, 20);

        jButton1.setText("Salir");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        getContentPane().add(jButton1);
        jButton1.setBounds(100, 100, 150, 23);

        pack();
    }
// </editor-fold>

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    dispose();
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Resultados().setVisible(true);
        }
    });
}

// Variables declaration - do not modify
private javax.swing.JButton jButton1;
private javax.swing.JLabel jLabel1;
private javax.swing.JTextField jTextField1;
// End of variables declaration

}
```

EJERCICIO 2

Diseñar la siguientes interfaces:

La pantalla que se visualiza en la figura 1, es la pantalla principal, debe incluir un menú con dos opciones: Cargar Archivo, Consulta (como se muestra en la figura 1)

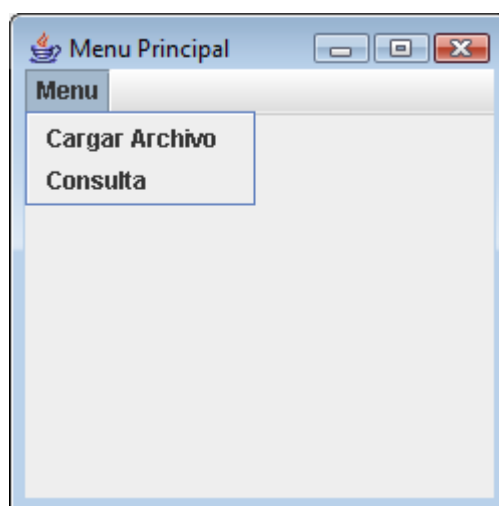


Figura 1.

Para el diseño de esta pantalla, debe tener en cuenta la pantalla que se muestra a continuación. En la misma se debe cargar los datos de libros y cargarlos en un archivo. El botón salir, facilita abandonar esta pantalla. La misma debe ser invocada, cuando se presiona la opción del menú **Carga Archivo**.

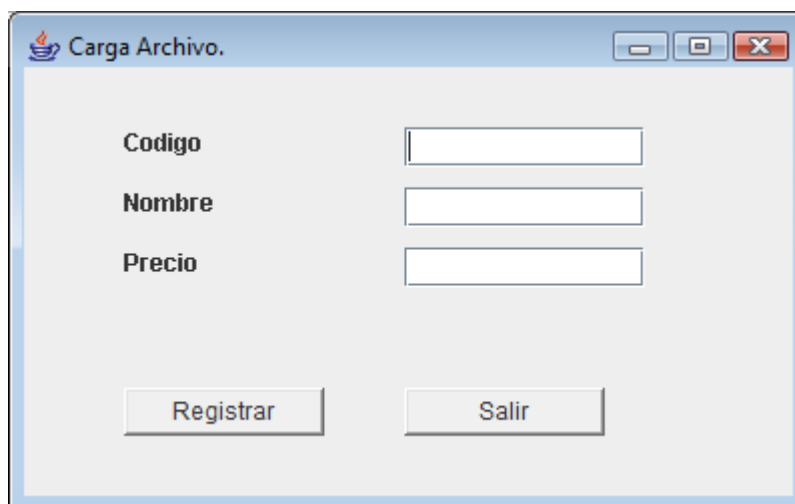


Figura 2.

La próxima pantalla debe mostrar el precio promedio de los libros registrados en el archivo, y deberá permitir la búsqueda de un libro. Al presionar el botón como se muestra en la figura, debe mostrar en la caja de texto

todos sus datos, en caso de no encontrarse mostrar "No se encontró el libro". Esto debe visualizarse cuando el usuario presione la opción **Consulta Archivo**. Para el diseño de la misma, tomar como guía la figura3.

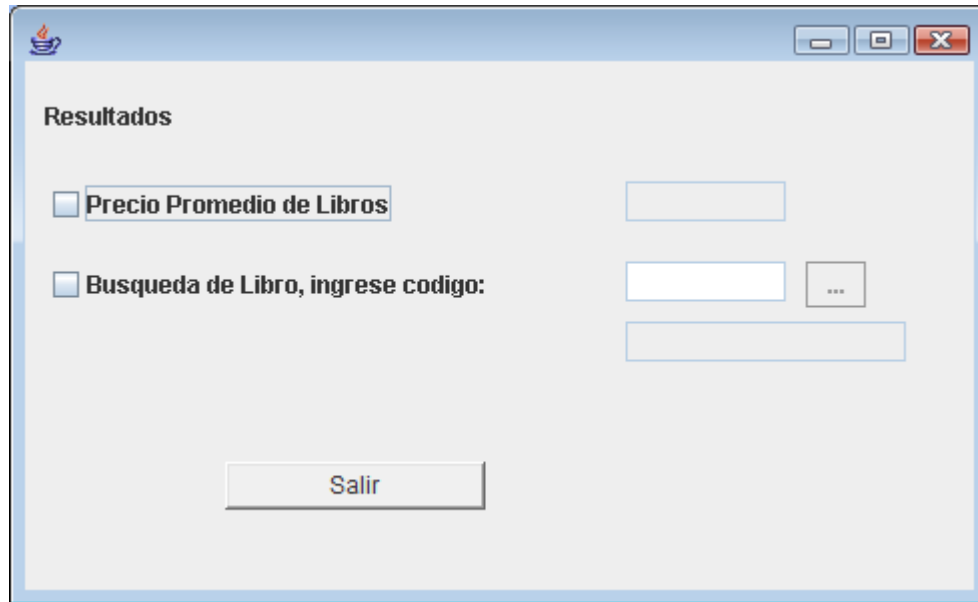


Figura 3.

```
//Libro.java
import java.io.*;

public class Libro implements Serializable{

    private int codigo;
    private String nombre;
    private float precio;
    public Libro(int cod,String nom,float prec)
    {
        codigo=cod;
        nombre=nom;
        precio=prec;
    }
    public void setCodigo(int cod)
    {codigo=cod;}
    public void setNombre(String nom)
    {nombre=nom;}
    public void setPrecio(float prec)
    {precio=prec;}
    public int getCodigo()
    {return codigo;}
    public String getNombre()
    {return nombre;}
```




```
public float getPrecio()
{return precio;}
public String toString()
{
String aux="";
aux=codigo+" "+nombre+" "+precio;
return aux;
}

}

//ArchivoDeLibros
import java.io.*;
public class ArchivoDeLibros {

    private FileOutputStream fileOutputStream;
    private FileInputStream fileInputStream;
    private ObjectOutputStream salida;
    private ObjectInputStream entrada;
public ArchivoDeLibros()
{
fileOutputStream=null;
fileInputStream=null;
salida=null;
entrada=null;
}
public void abrirModoLectura() throws IOException
{ fileInputStream=new FileInputStream("c:\\Temp\\libro.dat");
entrada=new ObjectInputStream(fileInputStream);
}
public void abrirModoEscritura() throws IOException
{ fileOutputStream=new FileOutputStream("c:\\Temp\\libro.dat");
salida=new ObjectOutputStream(fileOutputStream);
}
public void cerrarModoLectura() throws IOException
{ if(entrada!=null)
{ entrada.close();}
}
public void cerrarModoEscritura() throws IOException
{ if(salida!=null)
{ salida.close();}
}
public String mostrarLibros() throws IOException,ClassNotFoundException
```



```
{ Libro lib=null;
String aux="";
abrirModoLectura();
if(entrada!=null)
{ try
{
while(true)
{
lib=(Libro) entrada.readObject();
aux += "\n"+lib.toString();
}
} catch EOFException e) {}
}
cerrarModoLectura();
return aux;
}

public void guardarLibro(Libro lib) throws IOException
{
if(salida==null)
{
abrirModoEscritura();
}
salida.writeObject(lib);
}

public float mostrarPromedioPrecios() throws IOException, ClassNotFoundException
{ Libro lib=null;
float prec=0;
int cant = 0;
float prom=0;
abrirModoLectura();
if(entrada!=null)
{ try
{
while(true)
{
lib=(Libro) entrada.readObject();
prec += lib.getPrecio();
cant++;
}
} catch EOFException e) {}
}
cerrarModoLectura();
if (cant>0)
```



```
prom= (prec/cant);
return prom;
}

public int cantMayPrec(float prec) throws IOException,ClassNotFoundException
{ Libro lib=null;
int cant = 0;
abrirModoLectura();
if(entrada!=null)
{ try
{
while(true)
{
lib=(Libro) entrada.readObject();
if(lib.getPrecio()>prec)
cant++;
}
} catch EOFException e) {}
}
cerrarModoLectura();
return cant;
}

public String existeLibro(int cod) throws IOException,ClassNotFoundException
{ Libro lib=null;
String x="No se encontro libro";
abrirModoLectura();
if(entrada!=null)
{ try
{
while(true)
{
lib=(Libro) entrada.readObject();
if(lib.getCodigo()==cod)
{
x="Datos de libro: " +lib.toString();
break;
}
}
} //fin while
} catch EOFException e) {} //fin del try
}
cerrarModoLectura();
```



```
return x;}
```

```
}
```



Interfaces gráficas.

```
//Menu.java
public class Menu extends javax.swing.JFrame {

    /** Creates new form Menu */
    public Menu() {
        initComponents();
        this.setSize(250,250);
        this.setTitle("Menu Principal");
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc=" Generated Code ">
    private void initComponents() {
        jMenuItem1 = new javax.swing.JMenuItem();
        jMenuItem2 = new javax.swing.JMenuItem();

        getContentPane().setLayout(null);

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        jMenuItem1.setText("Menu");
        jMenuItem1.setText("Cargar Archivo");
        jMenuItem1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItem1ActionPerformed(evt);
            }
        });

        jMenuItem1.add(jMenuItem1);

        jMenuItem2.setText("Consulta");
        jMenuItem2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItem2ActionPerformed(evt);
            }
        });
    }
}
```



```
jMenu1.add(jMenuItem2);

jMenuBar1.add(jMenu1);

setJMenuBar(jMenuBar1);

pack();
}
// </editor-fold>

private void jMenuItem2ActionPerformed(java.awt.event.ActionEvent evt) {
//Invoco la pantalla de resultados..
    Resultados resu=new Resultados();
    this.setSize(250,250);
    this.setTitle("Procesamiento de archivo de libros");
    resu.setVisible(true);
}

private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt) {
//Invoco la pantalla de Carga Archivo.
    CargaDatos cd=new CargaDatos();
    cd.setVisible(true);
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Menu().setVisible(true);
        }
    });
}

// Variables declaration - do not modify
private javax.swing.JMenu jMenu1;
private javax.swing.JMenuBar jMenuBar1;
private javax.swing.JMenuItem jMenuItem1;
private javax.swing.JMenuItem jMenuItem2;
// End of variables declaration
```



```
}
//CargaDatos.java

import java.io.*;
import javax.swing.JOptionPane;
public class CargaDatos extends javax.swing.JFrame {
    ArchivoDeLibros arch=new ArchivoDeLibros();
    Libro lib=null;

    /** Creates new form CargaDatos */
    public CargaDatos() {
        initComponents();
        this.setSize(400,250);
        this.setTitle("Carga Archivo.");
        try{
            arch.abrirModoEscritura();} catch(IOException e){}

    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc=" Generated Code ">
    private void initComponents() {
        jTextFieldCodigo = new javax.swing.JTextField();
        jTextFieldNombre = new javax.swing.JTextField();
        jTextFieldPrecio = new javax.swing.JTextField();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        button1 = new java.awt.Button();
        button2 = new java.awt.Button();

        getContentPane().setLayout(null);

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        getContentPane().add(jTextFieldCodigo);
        jTextFieldCodigo.setBounds(190, 30, 120, 20);

        getContentPane().add(jTextFieldNombre);
        jTextFieldNombre.setBounds(190, 60, 120, 20);
```



```
getContentPane().add(jTextPrecio);
jTextPrecio.setBounds(190, 90, 120, 20);

jLabel1.setText("Codigo");
getContentPane().add(jLabel1);
jLabel1.setBounds(50, 30, 100, 14);

jLabel2.setText("Nombre");
getContentPane().add(jLabel2);
jLabel2.setBounds(50, 60, 100, 14);

jLabel3.setText("Precio");
getContentPane().add(jLabel3);
jLabel3.setBounds(50, 90, 90, 14);

button1.setLabel("Registrar");
button1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        button1ActionPerformed(evt);
    }
});

getContentPane().add(button1);
button1.setBounds(50, 160, 100, 24);

button2.setLabel("Salir");
button2.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        button2ActionPerformed(evt);
    }
});

getContentPane().add(button2);
button2.setBounds(190, 160, 100, 24);

pack();
}
// </editor-fold>

private void button1ActionPerformed(java.awt.event.ActionEvent evt) {

    //Esta vacio los campos???
```




```
        if(jTextCodigo.getText().trim().length()!=0 &&
jTextNombre.getText().trim().length()!=0 && jTextPrecio.getText().trim().length()!=0)
        { // Aca escribo la logica del boton:
            //Creo el objeto Alumno
            int l=Integer.parseInt(jTextCodigo.getText());
            String nom=jTextNombre.getText();
            float p=Float.parseFloat(jTextPrecio.getText());
            lib=new Libro(l,nom,p);
            try{
                arch.guardarLibro(lib);} catch(IOException e){;}
            System.out.println("Registro grabado!!!");

            //Limpiar Controles.
            jTextCodigo.setText("");
            jTextNombre.setText("");
            jTextPrecio.setText("");
        }
        else //Algun campo para el ingreso de datos esta vacio..
        { //System.out.println("Debe ingresar datos!!!");
            JOptionPane.showMessageDialog(null,"Debe ingresar
campos","Advertencia",JOptionPane.INFORMATION_MESSAGE);
        }

    }

private void button2ActionPerformed(java.awt.event.ActionEvent evt) {
    //Aca cierro la pantalla
    this.dispose();
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new CargaDatos().setVisible(true);
        }
    });
}

// Variables declaration - do not modify
private java.awt.Button button1;
```



```
private java.awt.Button button2;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JTextField jTextFieldCodigo;
private javax.swing.JTextField jTextFieldNombre;
private javax.swing.JTextField jTextFieldPrecio;
// End of variables declaration

}

//Resultados.java
import java.io.*;
import javax.swing.JOptionPane;
public class Resultados extends javax.swing.JFrame {
    ArchivoDeLibros al=new ArchivoDeLibros();
    /** Creates new form Resultados */
    public Resultados() {
        initComponents();
        this.setSize(490,300);
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc=" Generated Code ">
    private void initComponents() {
        jLabel1 = new javax.swing.JLabel();
        button1 = new java.awt.Button();
        jCheckBox1 = new javax.swing.JCheckBox();
        jCheckBox2 = new javax.swing.JCheckBox();
        jTextFieldCodigo = new javax.swing.JTextField();
        jTextFieldPromedio = new javax.swing.JTextField();
        jTextFieldDatos = new javax.swing.JTextField();
        jButtonBuscar = new javax.swing.JButton();

        getContentPane().setLayout(null);

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
    }
}
```



```
jLabel1.setText("Resultados");
getContentPane().add(jLabel1);
jLabel1.setBounds(10, 20, 240, 14);

button1.setLabel("Salir");
button1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        button1ActionPerformed(evt);
    }
});

getContentPane().add(button1);
button1.setBounds(100, 200, 130, 24);

jCheckBox1.setText("Precio Promedio de Libros");
jCheckBox1.addItemListener(new java.awt.event.ItemListener() {
    public void itemStateChanged(java.awt.event.ItemEvent evt) {
        jCheckBox1ItemStateChanged(evt);
    }
});
jCheckBox1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jCheckBox1ActionPerformed(evt);
    }
});

getContentPane().add(jCheckBox1);
jCheckBox1.setBounds(10, 60, 220, 23);

jCheckBox2.setText("Busqueda de Libro, ingrese codigo:");
jCheckBox2.addItemListener(new java.awt.event.ItemListener() {
    public void itemStateChanged(java.awt.event.ItemEvent evt) {
        jCheckBox2ItemStateChanged(evt);
    }
});

getContentPane().add(jCheckBox2);
jCheckBox2.setBounds(10, 100, 280, 23);

jTextCodigo.setEnabled(false);
getContentPane().add(jTextCodigo);
jTextCodigo.setBounds(300, 100, 80, 20);
```



```
jTextPromedio.setEditable(false);
getContentPane().add(jTextPromedio);
jTextPromedio.setBounds(300, 60, 80, 20);

jTextDatos.setEditable(false);
getContentPane().add(jTextDatos);
jTextDatos.setBounds(300, 130, 140, 20);

jBBuscar.setText("jButton1");
jBBuscar.setEnabled(false);
jBBuscar.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jBBuscarActionPerformed(evt);
    }
});

getContentPane().add(jBBuscar);
jBBuscar.setBounds(390, 100, 30, 23);

pack();
}
// </editor-fold>

private void jCheckBox2ItemStateChanged(java.awt.event.ItemEvent evt) {
//Aca habilito las caja de texto y el boton..
if(evt.getStateChange()==java.awt.event.ItemEvent.SELECTED)
{
    jTextCodigo.setEnabled(true);
    jTextCodigo.grabFocus();
    jBBuscar.setEnabled(true);}
else
{ jTextCodigo.setEnabled(false);
  jTextCodigo.setText("");
  jBBuscar.setEnabled(false);
  jTextDatos.setText(" ");
  }
}

private void jCheckBox1ItemStateChanged(java.awt.event.ItemEvent evt) {

//Si alguien presiono esta casilla de verificacion.
if(evt.getStateChange()==java.awt.event.ItemEvent.SELECTED)
{try{
```



```
        jTextPromedio.setText(String.valueOf(al.mostrarPromedioPrecios()));

    }

    catch(IOException e){;}
    catch(ClassNotFoundException e1){;}}

    else
        {jTextPromedio.setText("0.00");}
}

private void jCheckBox1ActionPerformed(java.awt.event.ActionEvent evt) {

}

private void jBBuscarActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
    //Valido de que se escriba algo en la caja de texto.
    if(jTextCodigo.getText().trim().length() !=0)
    {
        try{

jTextDatos.setText(String.valueOf(al.existeLibro(Integer.valueOf(jTextCodigo.getText()))));

        }

        catch(IOException e){;}
        catch(ClassNotFoundException e1){;}
    }
    else
    {    JOptionPane.showConfirmDialog(null, "Ingrese codigo!!", "Advertencia",
JOptionPane.INFORMATION_MESSAGE);
        //Hago que tomo el foco..
        jTextCodigo.grabFocus();
    }
}

private void button1ActionPerformed(java.awt.event.ActionEvent evt) {
// Cierro la pantalla.
    this.dispose();
}

/**
```



```
* @param args the command line arguments
*/
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Resultados().setVisible(true);
        }
    });
}

// Variables declaration - do not modify
private java.awt.Button button1;
private javax.swing.JButton jButton1;
private javax.swing.JCheckBox jCheckBox1;
private javax.swing.JCheckBox jCheckBox2;
private javax.swing.JLabel jLabel1;
private javax.swing.JTextField jTextField1;
private javax.swing.JTextField jTextField2;
private javax.swing.JTextField jTextField3;
// End of variables declaration

}
```



BIBLIOGRAFÍA

- ❖ Aprenda Java como si estuviera en primero. Marzo 1999 Autores: Javier Garcia de Jalón, José Ignacio Rodríguez, Iñigo Mingo, Aitor Imaz, Alfonso Brazales, Alberto Larzabal, Jesús Calleja, Jon García.
- ❖ Java 2: Interfaces gráficas y Aplicaciones para Internet. Autor: F. J. Ceballos/ Ra-Ma.
- ❖ "Tutorial del Programa ComboBox", Osvaldo Gutiérrez
<http://www.gfc.edu.co/estudiantes/anuario/2003/sistemas/orlando/SGML/Jcombobox/c22.html>
- ❖ Ejemplo Java y C/Linux.
<http://www.chuidiang.com/java/index.php>