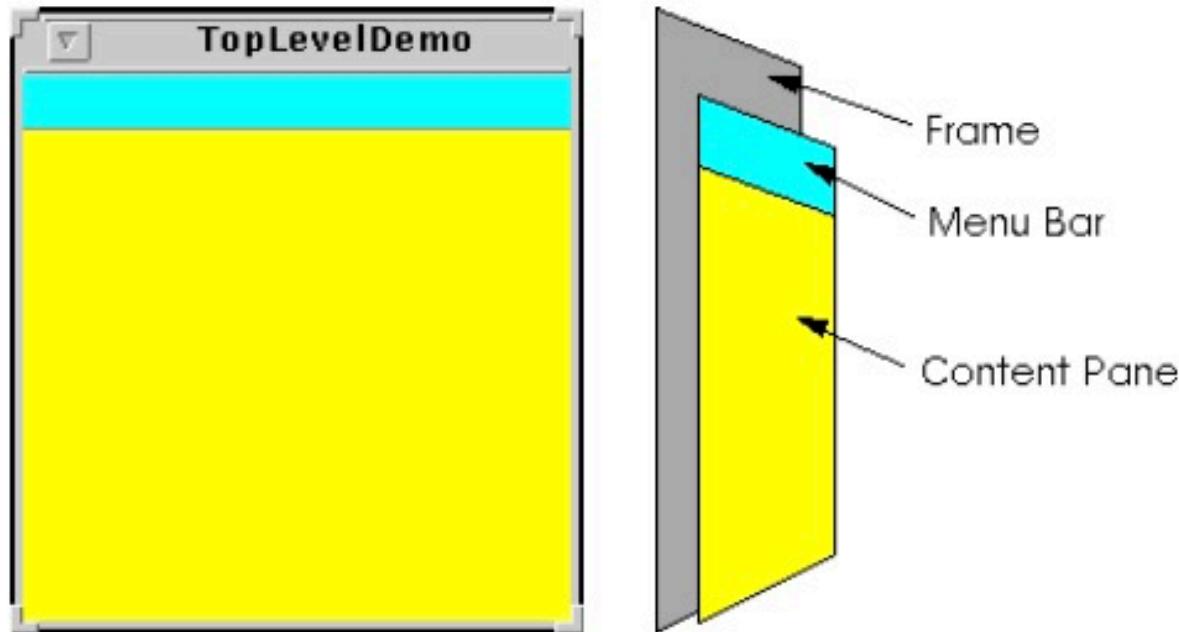


AWT y Swing

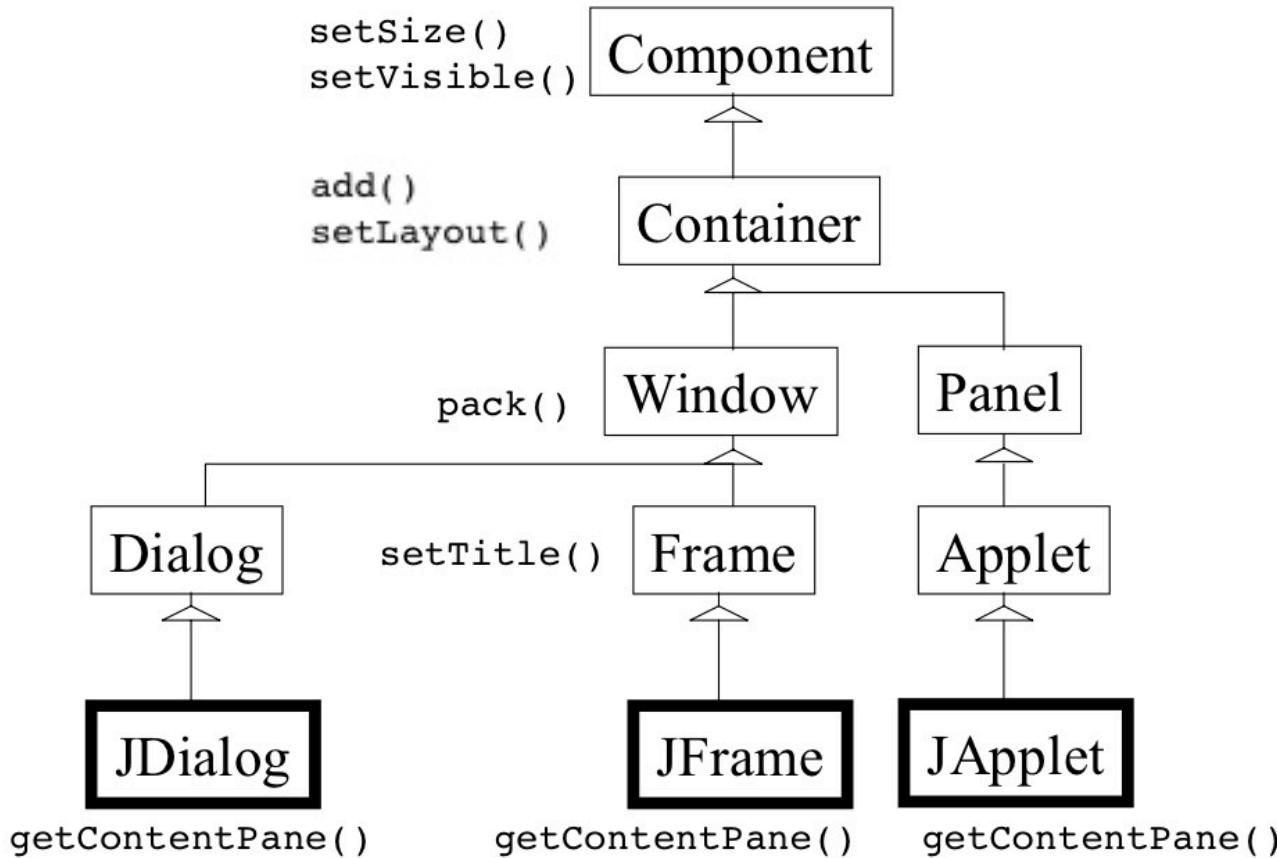
- Estructura del AWT(Abstract Windows Toolkit)
- La estructura de AWT podemos resumirla en los puntos siguientes:
- Los Contenedores contienen Componentes, que son los controles básicos.
- No se usan posiciones fijas de los Componentes, sino que están situados a través de una disposición controlada (layouts).
- El común denominador de más bajo nivel se acerca al teclado, ratón y manejo de eventos.
- Alto nivel de abstracción respecto al entorno de ventanas en que se ejecute la aplicación (no hay áreas cliente, ni llamadas a X, ni hWnds, etc.).
- Es algo dependiente de la máquina en que se ejecuta la aplicación (no puede asumir que un diálogo tendrá el mismo tamaño y aspecto en cada máquina).
- <https://www.dropbox.com/s/azeligi4txilgw/AWTaLumno.pdf?dl=0>

AWT y Swing: Contenedores



- Disponen de un panel de contenidos (contentPane)
- Pueden opcionalmente disponer de un menú

AWT y Swing : Contenedores



AWT y Swing

- Pasos a seguir para crear un GUI
 - Crear el contenedor superior y obtener un contenedor intermedio(Panel)
 - Seleccionar un Layout para los contenedores
 - Crear los componentes adecuados Agregarlos al contenedor intermedio: Método add()
 - Dimensionar el Contenedor superior, Métodos:
 - setSize() o pack()
 - Definir la escucha de eventos
 - Mostrar el contenedor, Método: setVisible(true);

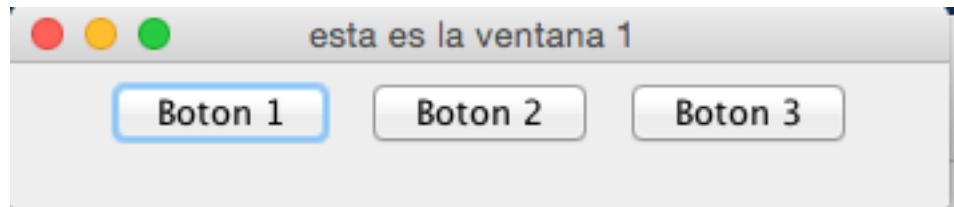
AWT y Swing

- Desarrollar GUI Layout
 - Distribuciones relativas
 - FlowLayout – Distribuye los componentes uno al lado del otro en la parte superior del contenedor. Por defecto provee una alineación centrada, pero también puede alinearlos hacia la izquierda o hacia la derecha.
 - BorderLayout – Divide el espacio del contenedor en 5 regiones: NORTH, SOUTH, EAST, WEST y CENTER (norte, sur, este, oeste y centro). Admite un único componente por región.
 - GridLayout – Divide el espacio del contenedor en una rejilla de n filas por m columnas donde todas las celdas son de igual tamaño. Admite exactamente n por m componentes, uno por celda.

AWT y Swing

- FlowLayout: Distribuye los componentes uno al lado del otro en la parte superior del contenedor. Código de una interfaz gráfica que utiliza un FlowLayout para mantener tres botones centrados en la parte superior de la ventana.

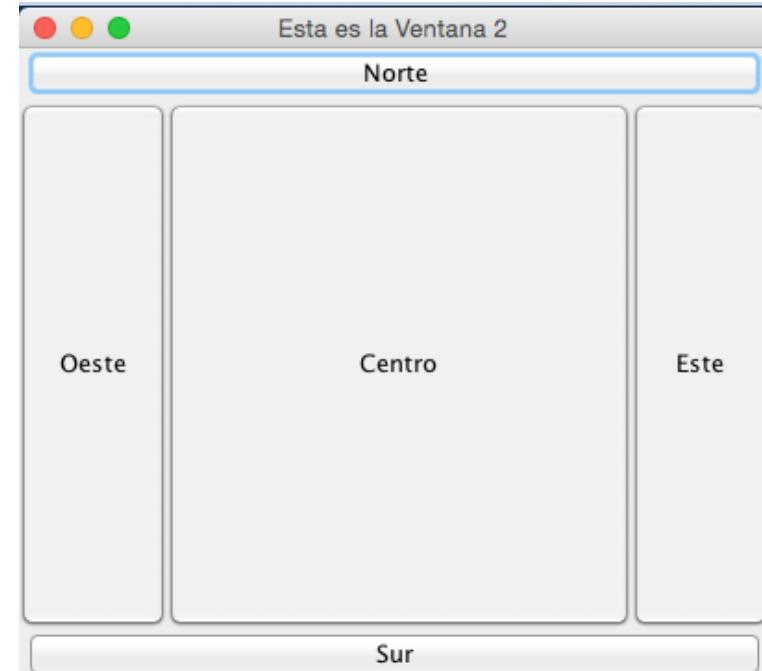
```
import java.awt.*;
public class Ventana1 extends Frame{
    // defino tres objetos Button
    private Button b1,b2,b3;
    public Ventana1()
        // el constructor del padre recibe el titulo de la ventana
        super("Esta es la Ventana 1");
        // defino el layout que va a tener la ventana: FlowLayout
        setLayout(new FlowLayout());
        // instancio el primer boton y lo agrego al contenedor
        b1=new Button("Boton 1"); add(b1);
        // instancio el segundo boton y lo agrego al contenedor
        b2=new Button("Boton 2"); add(b2);
        // instancio el tercer boton y lo agrego al contenedor
        b3=new Button("Boton 3"); add(b3);
        // defino el tamaño de la ventana y la hago visible
        setSize(300,300);
        setVisible(true); }
    public static void main(String[] args)
    {
        Ventana1 v1 = new Ventana1(); }
```



AWT y Swing

- BorderLayout : divide el espacio del contenedor en cinco regiones o cuatro bordes y una región central. Admite solo un componente por región, por lo tanto, un contenedor con esta distribución solo podrá contener a lo sumo cinco componentes.

```
import java.awt.*;
public class Ventana2 extends Frame
{private Button bNorth,bSouth,bWest,bEast, bCenter;
 public Ventana2()
 { super("Esta es la Ventana 2");
 setLayout(new BorderLayout());
 bNorth=new Button("Norte");
 add(bNorth, BorderLayout.NORTH);
 bSouth=new Button("Sur");
 add(bSouth, BorderLayout.SOUTH);
 bWest=new Button("Oeste");
 add(bWest, BorderLayout.WEST);
 bEast=new Button("Este");
 add(bEast, BorderLayout.EAST);
 bCenter=new Button("Centro");
 add(bCenter, BorderLayout.CENTER);
 setSize(300,300);
 setVisible(true); }
 public static void main(String[] args)
 {Ventana2 v2 = new Ventana2(); }
```



AWT y Swing

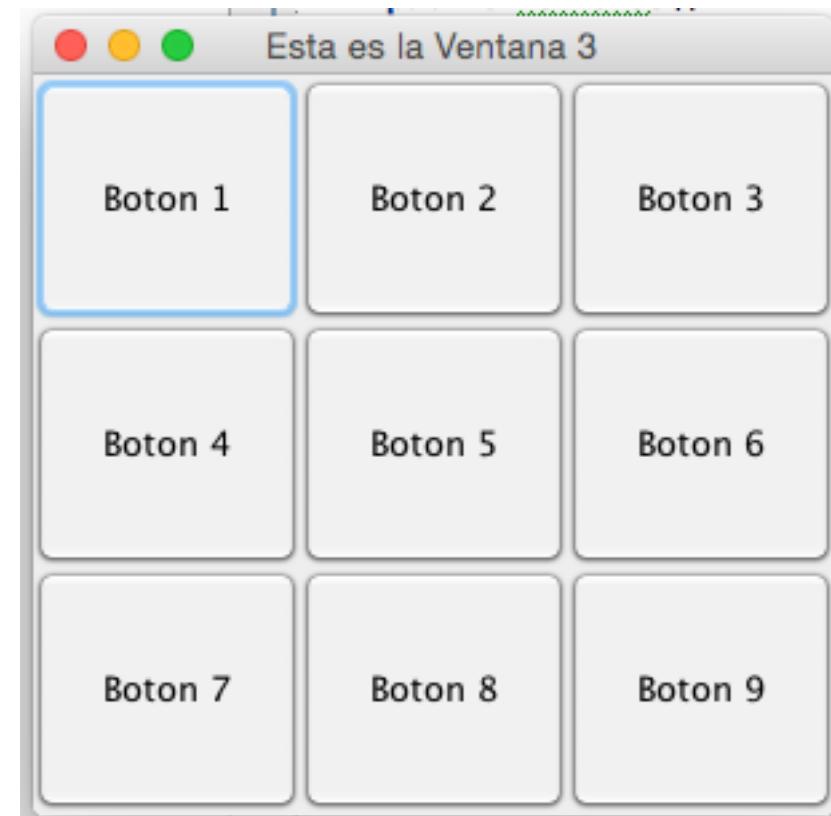
```
import java.awt.*;
public class Ventana4 extends Frame
{private Button bNorth,bSouth,bWest,bEast, bCenter; private TextField tfNick;
public Ventana4()
{super("Esta es la Ventana 4");
setLayout(new BorderLayout());
Panel pNorth = crearPNorte(); add( pNorth, BorderLayout.NORTH);
bSouth=new Button("Sur");
add(bSouth, BorderLayout.SOUTH);
bWest=new Button("Oeste");
add(bWest, BorderLayout.WEST);
bEast=new Button("Este");
add(bEast, BorderLayout.EAST);
bCenter=new Button("Centro");
add(bCenter, BorderLayout.CENTER);
setSize(300,300); setVisible(true); }
private Panel crearPNorte()
{ Panel p = new Panel(new FlowLayout(FlowLayout.LEFT));
p.add(new Label("Nombre:")); tfNick = new TextField(10);
p.add(tfNick);
bNorth=new Button("Norte");
p.add(bNorth);
return p; }
public static void main(String[] args)
{Ventana4 v2 = new Ventana4(); }
}
```



AWT y Swing

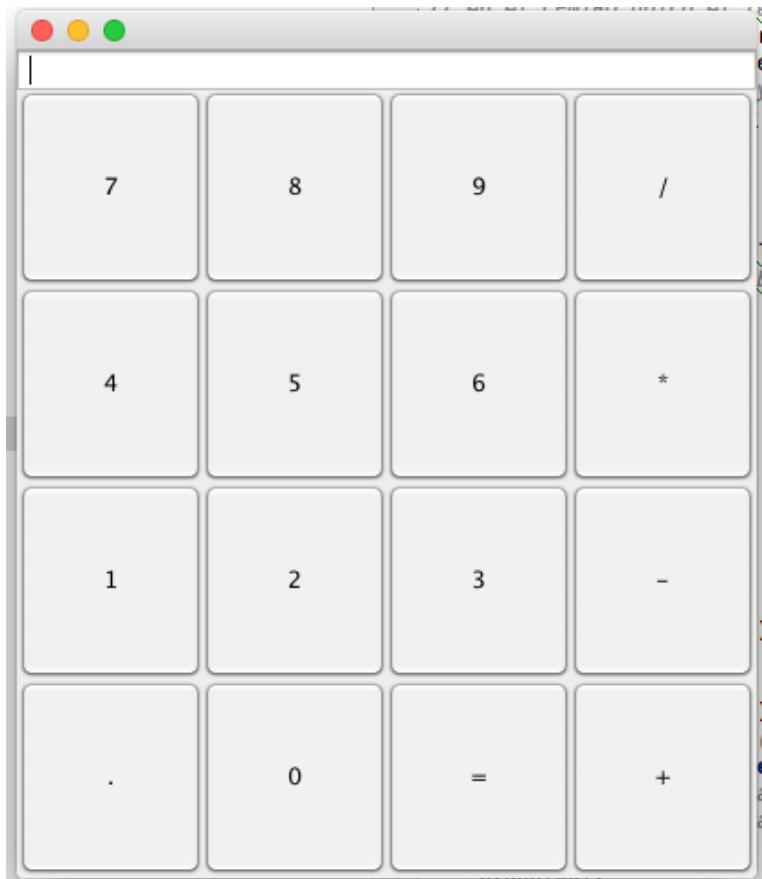
- GridLayout :divide el espacio del contenedor en una rejilla de n filas por m columnas donde todas las celdas tienen exactamente la misma dimensión.

```
import java.awt.*;
public class Ventana3 extends Frame
{ private Button b1,b2,b3,b4,b5,b6,b7,b8,b9;
  public Ventana3()
  { super("Esta es la Ventana 3");
    setLayout(new GridLayout(3,3));
    b1=new Button("Boton 1"); add(b1);
    b2=new Button("Boton 2"); add(b2);
    b3=new Button("Boton 3"); add(b3);
    b4=new Button("Boton 4"); add(b4);
    b5=new Button("Boton 5"); add(b5);
    b6=new Button("Boton 6"); add(b6);
    b7=new Button("Boton 7"); add(b7);
    b8=new Button("Boton 8"); add(b8);
    b9=new Button("Boton 9"); add(b9);
    setSize(300,300); // pack();
    setVisible(true); }
  public static void main(String[] args)
  {
    Ventana3 v3 = new Ventana3(); }
```



AWT y Swing

- Combinación de layouts : una GUI puede construirse en base a combinaciones de estos *layouts*:
- Ejemplo : una calculadora. : Esta interfaz gráfica se logra utilizando un BorderLayout que en el norte tiene una instancia de TextField (el *display*) y en el centro tiene un Panel (otro contenedor) con un GridLayout de 4 filas por 4 columnas con un Button en cada celda.



Calculadora 1

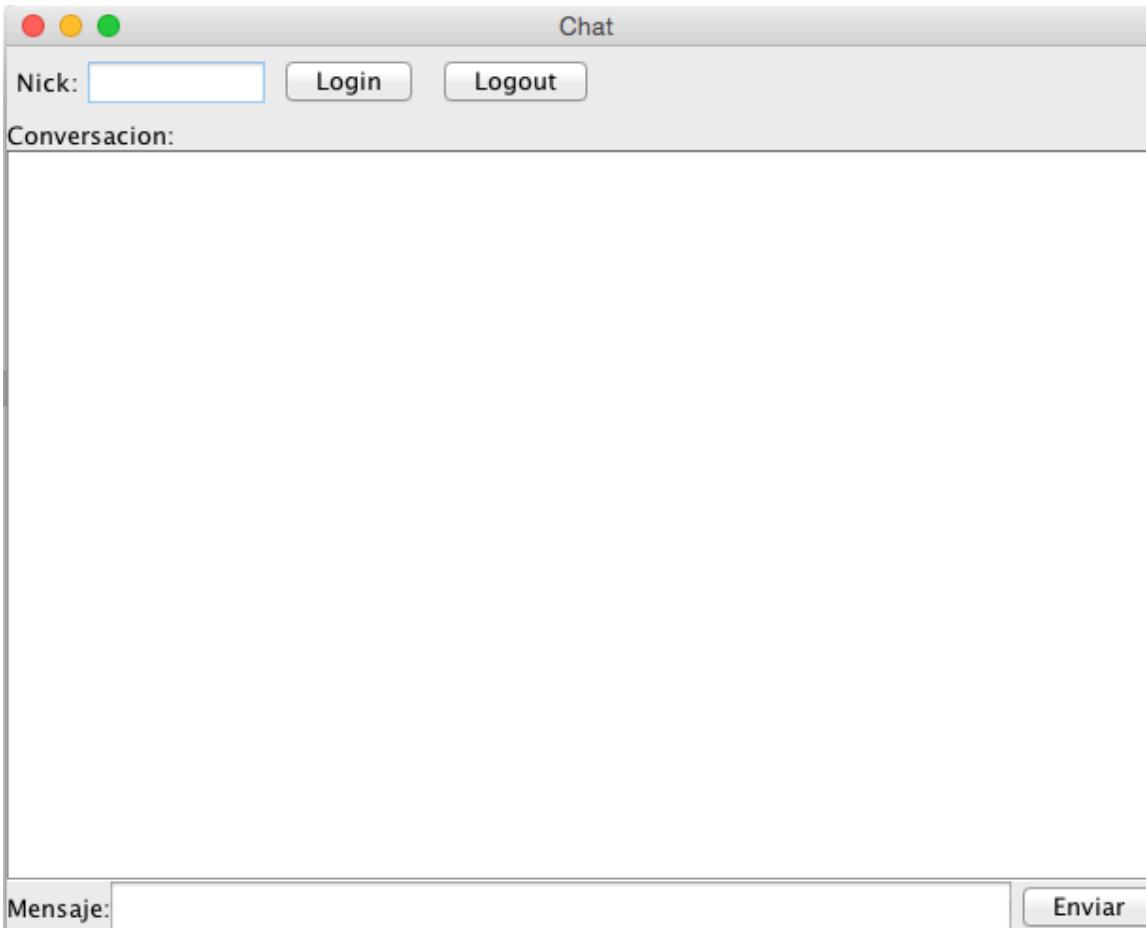
```
import java.awt.*;
public class Calculadora extends Frame
{private Button b0,b1,b2,b3,b4,b5,b6,b7,b8,b9;
private Button bDec,bMas,bMenos,bPor,bDiv,bIgual,bBorrar;
private TextField tfDisplay;
public Calculadora()
{super();
setLayout(new BorderLayout());
// en el NORTE ubico el display
tfDisplay = new TextField();
add(tfDisplay,BorderLayout.NORTH);
// en el CENTRO ubico el teclado
Panel pTeclado = crearTeclado();
add(pTeclado, BorderLayout.CENTER);
// este metodo dimensiona y configura el tamaño exacto
// necesario para contener todos los componentes del Frame
pack();
setVisible(true);
}
```

Calculadora 2

```
private Panel crearTeclado()
{ // instancio los 16 botones
b0 = new Button("0");b1 = new Button("1");
b2 = new Button("2");b3 = new Button("3");
b4 = new Button("4");b5 = new Button("5");
b6 = new Button("6");b7 = new Button("7");
b8 = new Button("8");b9 = new Button("9");
bDec = new Button(".");
bMas=new Button("+");
bMenos=new Button("-");
bPor = new Button("*");
bDiv = new Button("/");
bIgual = new Button("=");
// instancio un Panel (un container) con GridLayout de 4 x 4
Panel p = new Panel( new GridLayout(4,4) );
// Agrego los botones al panel fila 0 (la mas arriba)
p.add(b7); p.add(b8); p.add(b9); p.add(bDiv);
// fila 1 (la segunda comenzando desde arriba)
p.add(b4); p.add(b5); p.add(b6); p.add(bPor);
// fila 2 (la tercera comenzando desde arriba)
p.add(b1); p.add(b2); p.add(b3); p.add(bMenos);
// fila 3 (la cuarta comenzando desde arriba)
p.add(bDec); p.add(b0); p.add(bIgual); p.add(bMas);
// retorno el Panel con todos los botones agregados
return p; }
public static void main(String[] args)
{
Calculadora c = new Calculadora();
} }
```

AWT y Swing

- Combinación de layouts : una GUI puede construirse en base a combinaciones de estos *layouts*:
- Ejercicio : una ventana de chat.



AWT y Swing

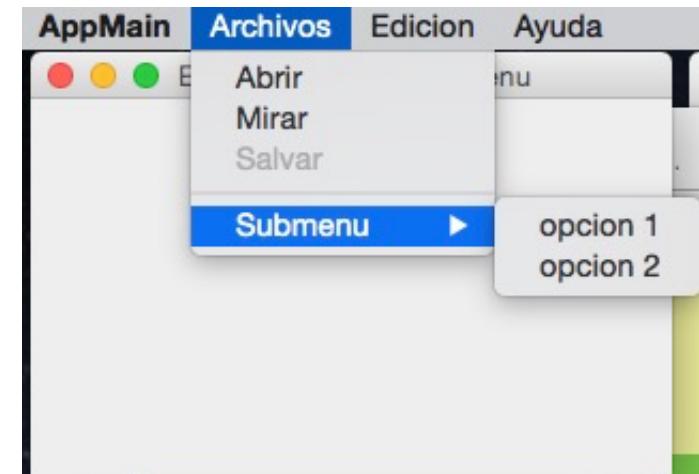
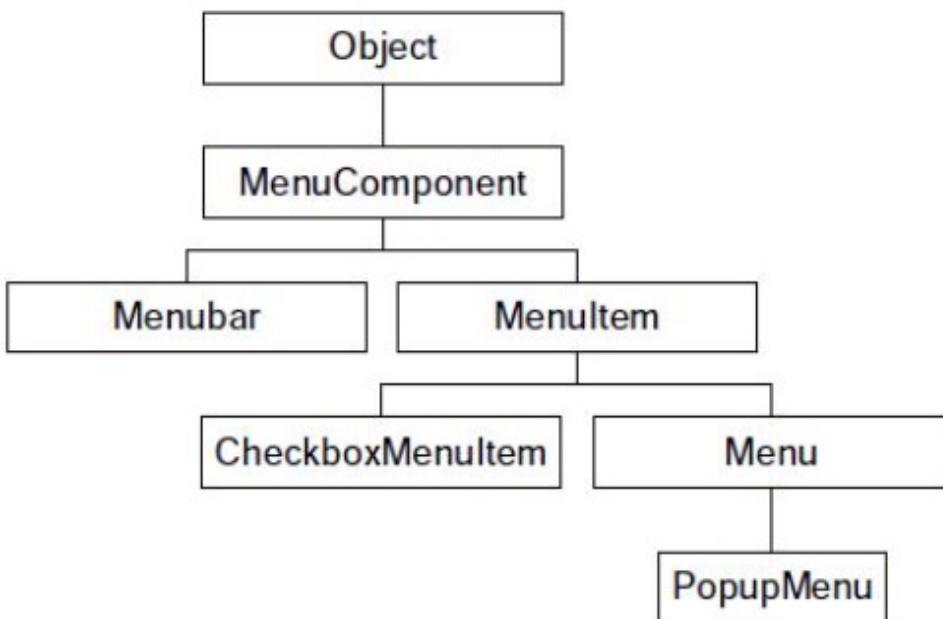
```
□ public class Chat extends Frame
{
    private TextField tfNick; private TextField tfMensaje; private Button bLogin; private Button bLogout;
    private Button bEnviar; private List lstLog;
    public Chat()
    {
        super("Chat");
        setLayout(new BorderLayout());
        // panel norte
        Panel pNorth = crearPNorte(); add( pNorth, BorderLayout.NORTH);
        // panel central
        Panel pCenter = crearPCenter(); add( pCenter, BorderLayout.CENTER);
        // panel sur
        Panel pSouth = crearPSur(); add(pSouth, BorderLayout.SOUTH);
        setSize(400,300);
        setVisible(true); }
private Panel crearPNorte()
{
    Panel p = new Panel(new FlowLayout(FlowLayout.LEFT));
    p.add(new Label("Nick:"); tfNick = new TextField(10);
    p.add(tfNick);
    bLogin=new Button("Login"); p.add(bLogin);
    bLogout = new Button("Logout"); p.add(bLogout);
    return p; }
```

AWT y Swing

```
□ private Panel crearPCenter()
{
    Panel p = new Panel(new BorderLayout());
    // norte
    p.add(new Label("Conversacion:"), BorderLayout.NORTH);
    // centro
    lstLog = new List(); p.add(lstLog,BorderLayout.CENTER);
    return p; }
private Panel crearPSur()
{
    Panel p = new Panel(new BorderLayout());
    // oeste
    p.add(new Label("Mensaje:"),BorderLayout.WEST);
    // centro
    tfMensaje = new TextField(); p.add(tfMensaje,BorderLayout.CENTER);
    // este
    bEnviar = new Button("Enviar"); p.add(bEnviar,BorderLayout.EAST);
    return p; }
public static void main(String[] args){
    Chat c = new Chat(); }
}
```

AWT y Swing: Menús

- A un contenedor principal se le pueden asociar menús en forma de “barra de menús”. Para ello hay tres clases básicas
 - **MenuBar:** Es la barra de menús
 - **Menu:** Agrupa varios **MenuItem** u otros **Menu**
 - **MenuItem:** Es la clase cuyos objetos acabarán ejecutando la acción correspondiente

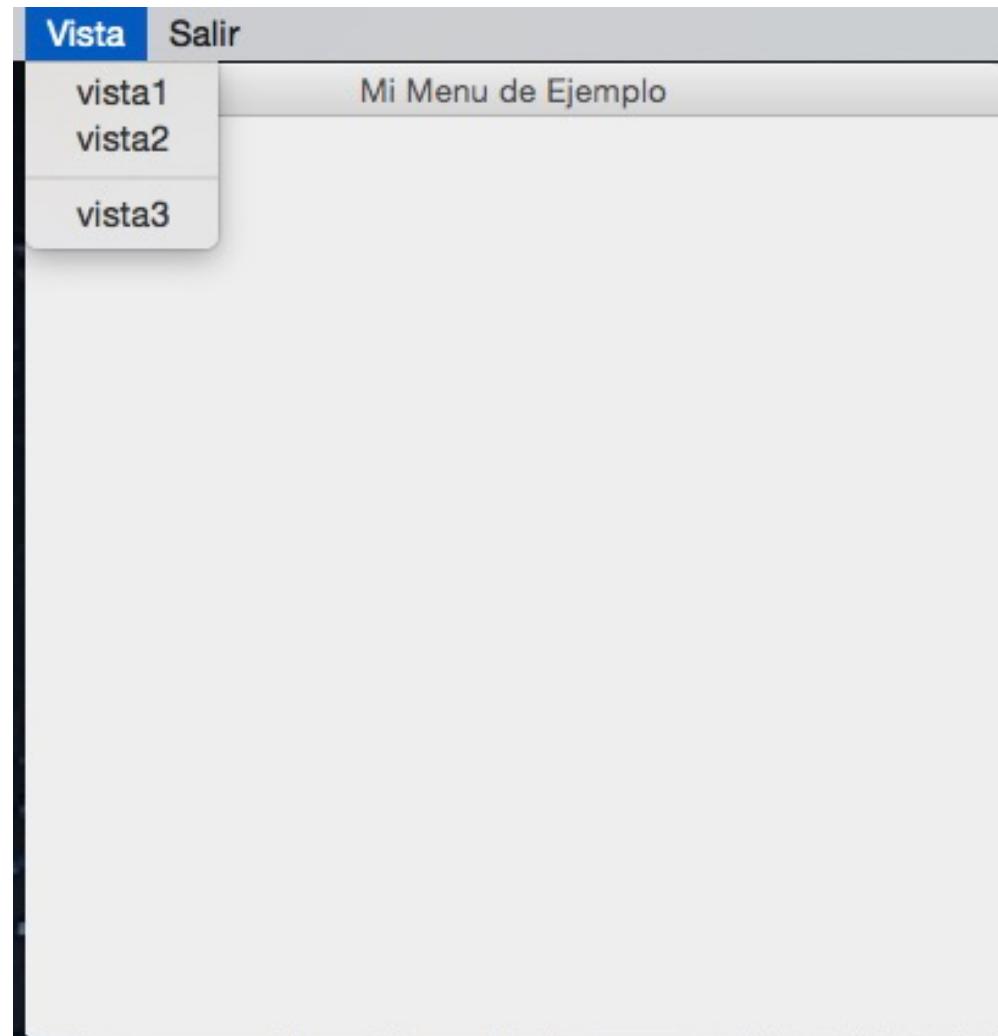


AWT y Swing: Menús

- Para tener una barra de menús ,hay que:
 - Crear los MenuItem: `MenuItem miMenuItem=new MenuItem("MenuOption");`
 - Crear los Menu: `Menu miMenu = new Menu("Menu Name");`
 - Crear laMenuBar: `MenuBar miMenuBar = newMenuBar();`
 - Asociar los MenuItem a los Menu: `miMenu.add(miMenuItem);`
 - Asociar los Menu a laMenuBar: `miMenuBar.add(miMenu);`
 - Asociar laMenuBar al contenedor principal: `miFrame.setMenuBar(miMenuBar);`
 - Asociar un ActionListener a cada MenuItem:
`miMenuItem.addActionListener
(new ActionListener()
{ public void actionPerformed (ActionEvent event) {
... // CODIGO ASOCIADO AL EVENTO }
}
);`

AWT: Menus

```
public class MenuCurso extends Frame{  
    privateMenuBar barraDeMenu;  
    private Menu vista;  
    private Menu salir;  
    private MenuItem v1;  
    private MenuItem v2;  
    private MenuItem v3;  
    private MenuItem exit;
```

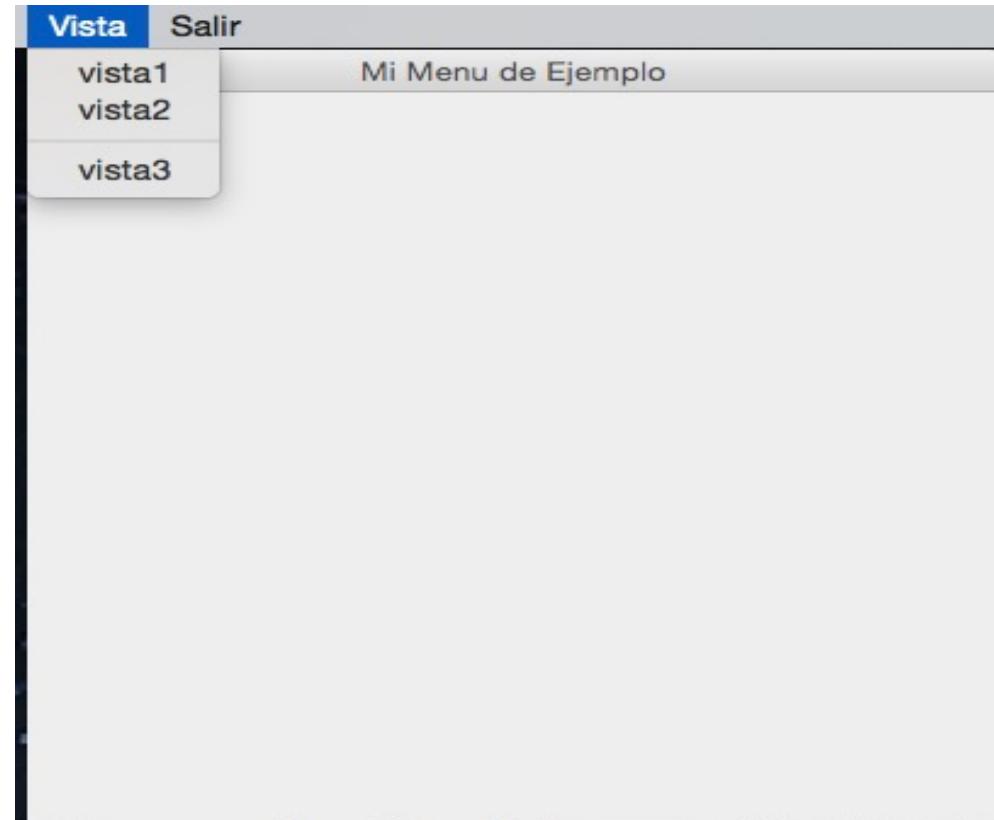


AWT: Menus

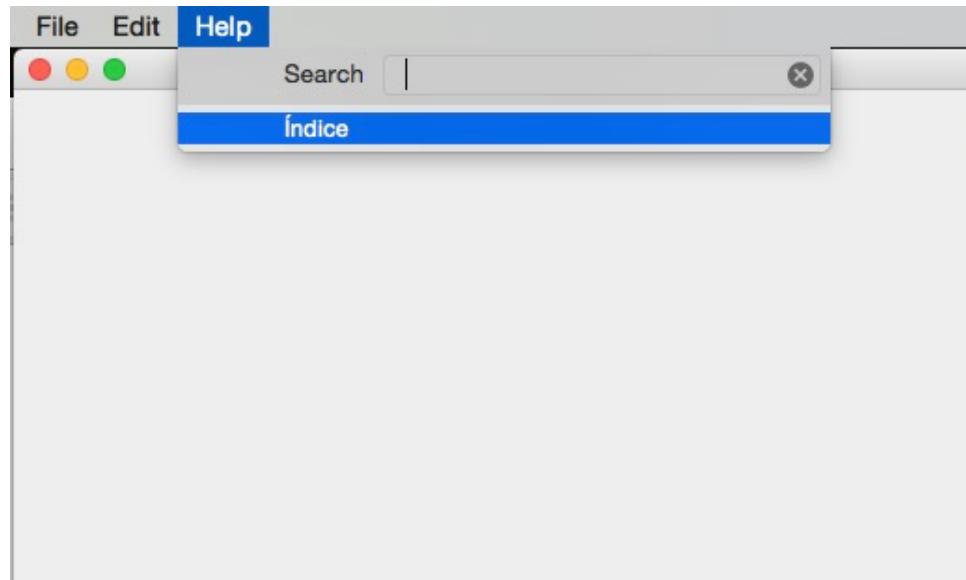
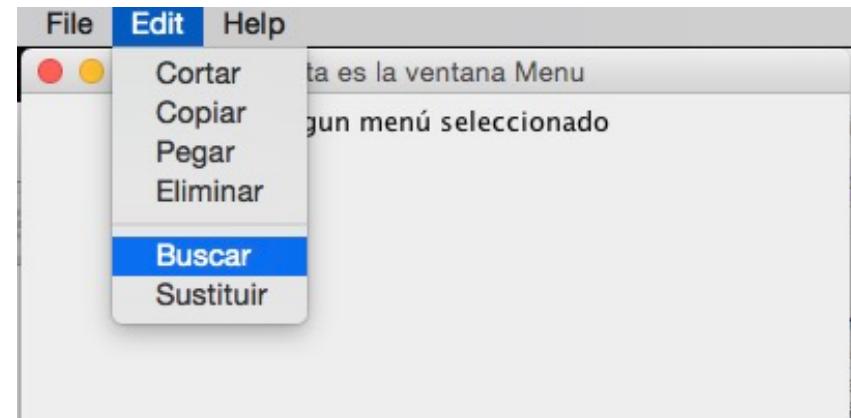
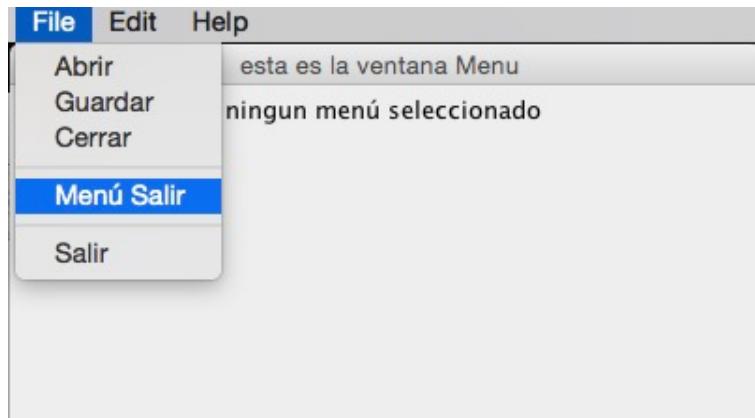
```
public MenuCurso (){
    super("Mi Menu de Ejemplo");
    setLayout(new FlowLayout());
    barraDeMenu = newMenuBar();
    vista = new Menu("Vista");
    v1= new MenuItem("vista1");
    v2= new MenuItem("vista2");
    v3= new MenuItem("vista3");
    vista.add(v1);vista.add(v2);
    vista.addSeparator();
    vista.add(v3);
    salir = new Menu("Salir");
    exit=new MenuItem("Salir");
    salir.add(exit);
    barraDeMenu.add(vista);
    barraDeMenu.add(salir);
    setMenuBar(barraDeMenu);
    setSize(400,400);
    setVisible(true); }

public static void main(String[] args)
{ MenuCurso mimenu= new MenuCurso();}

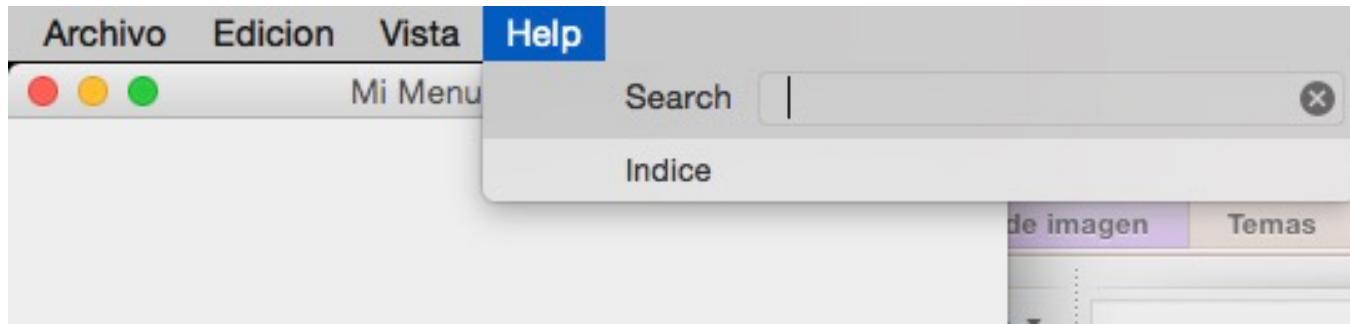
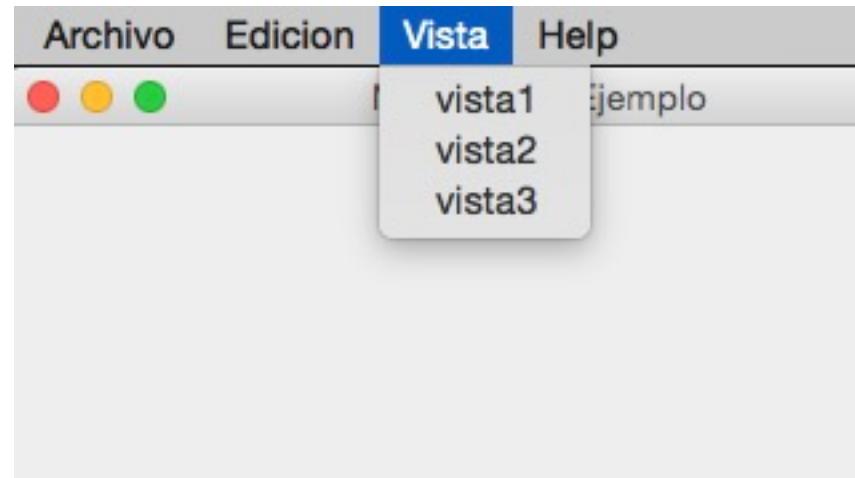
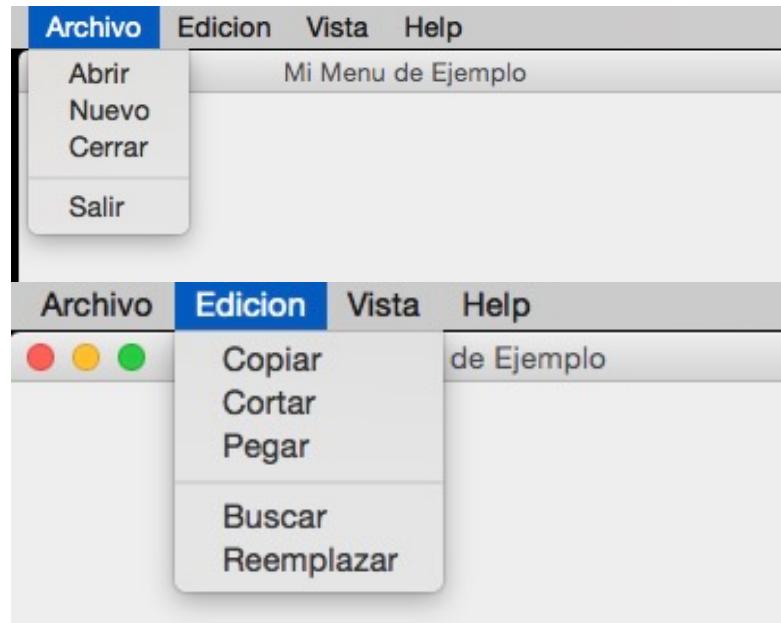
}
```



Ejercicio Menu



Ejercicio :Menu



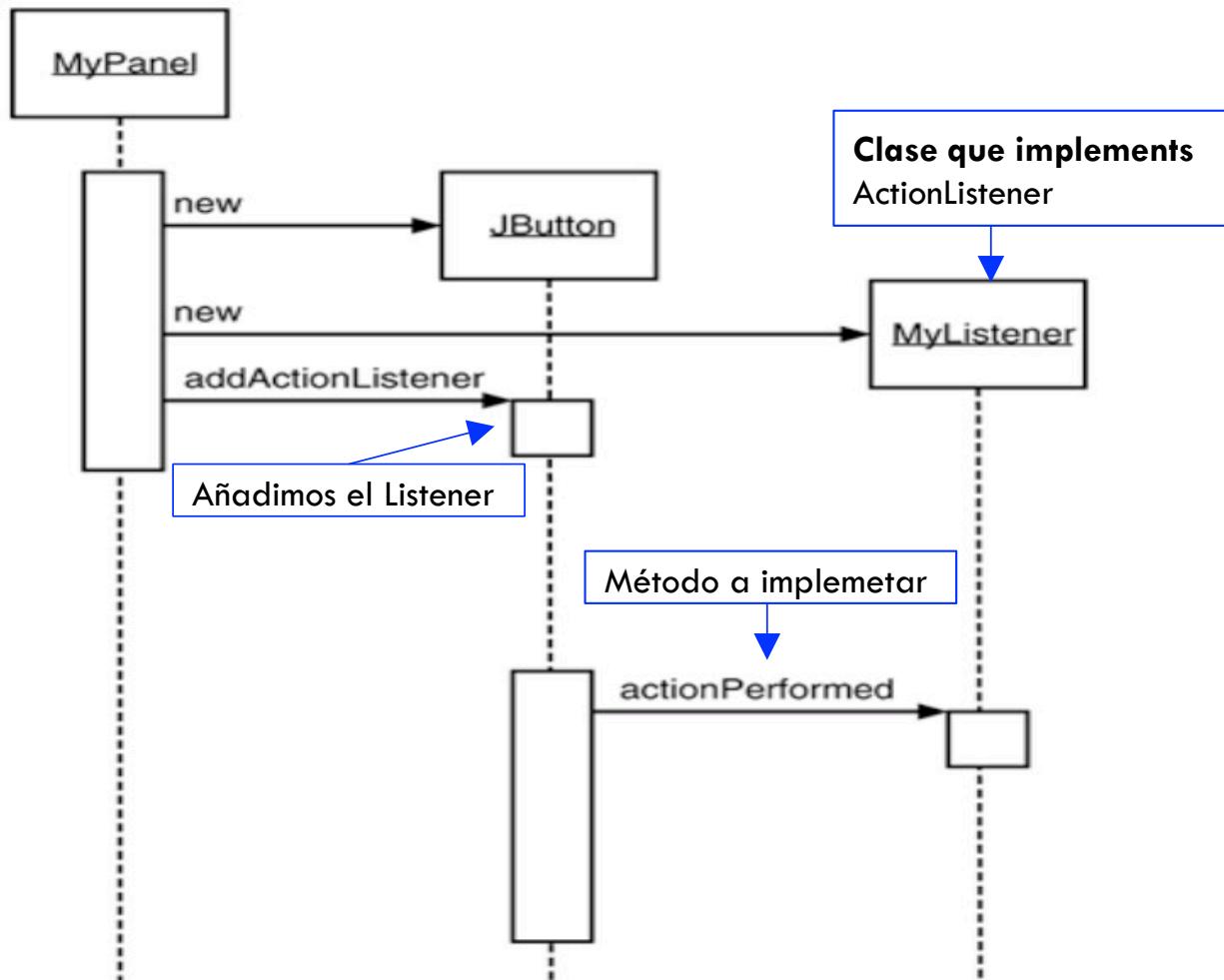
Eventos

- Cuando el usuario interactúa con la interfaz gráfica lo hace a través de sus componentes. Cada acción que realiza (esto es: cada botón que presiona, cada ítem que selecciona sobre una lista, cada carácter que escribe sobre un campo de texto, etc.) genera un evento y nosotros, como programadores, lo podemos “escuchar” para notificarnos y así poder hacer en ese momento lo que sea necesario.
- Los componentes generan eventos y nosotros podemos “escucharlos” (o notificarnos) utilizando listeners (escuchadores).
- Un listener no es más que un objeto cuya clase implementa una determinada interface, que está relacionado a un componente para que este lo notifique ante la ocurrencia de un determinado tipo de evento.

Eventos

- Los eventos los generan los componentes y los procesan objetos que implementan la interfaz EventListener
- Pasos:
 - 1. Crear los componentes
 - 2. Crear los oyentes de los eventos que queremos tratar
 - 3. Asociar los oyentes a los componentes
 - 4. Cuando un evento se genera en un componente, la aplicación llama al método correspondiente del oyente asociado a ese componente

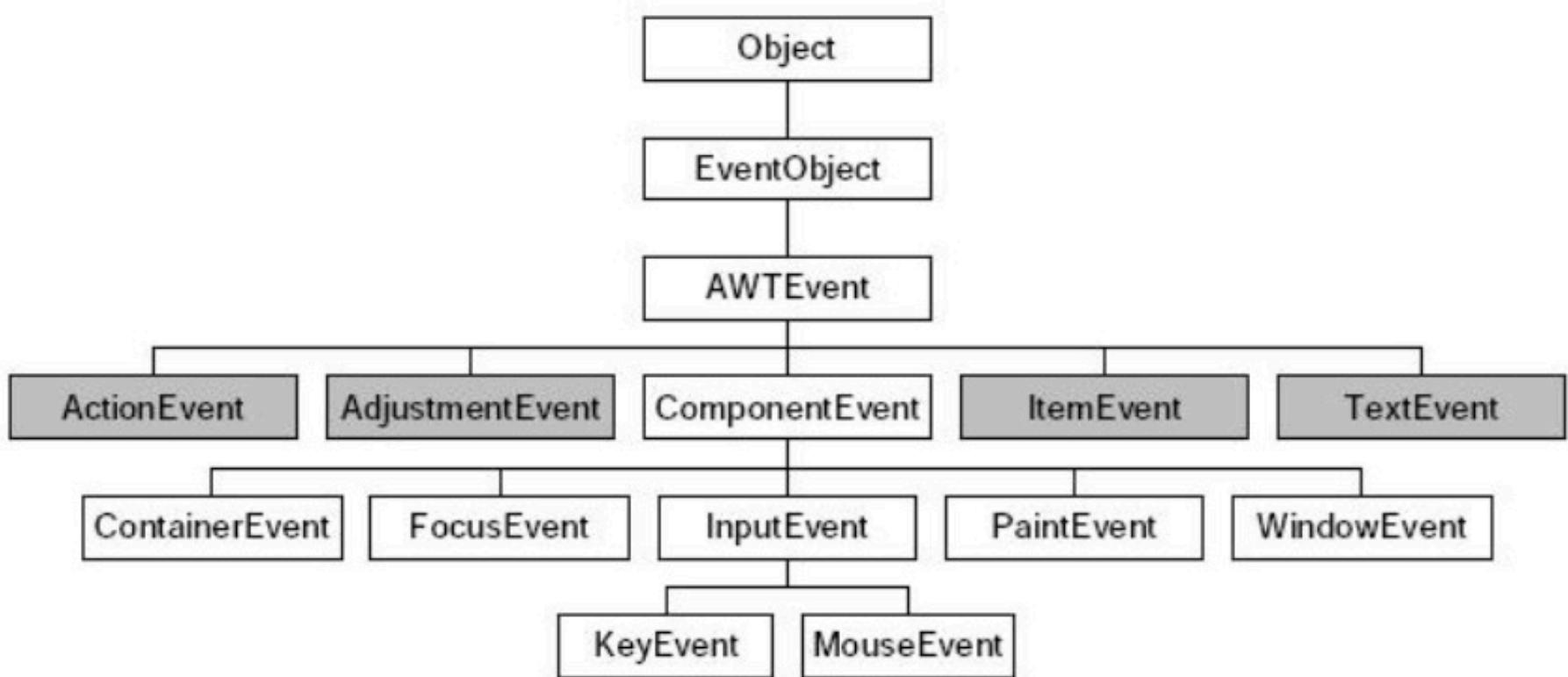
Eventos: Notificación



Eventos: Tipos

- Hay dos tipos de eventos:
 - De bajo nivel: son los que se producen con las operaciones elementales con el ratón, teclado, contenedores y ventanas, como por ejemplo:
 - Pulsar teclas
 - Mover, pulsar, arrastrar y soltar el ratón
 - Recuperar o perder el foco
 - Operaciones con ventanas: cerrar, maximizar, minimizar,... – ...
 - De alto nivel: son los que tienen un significado en sí mismo (en el contexto gráfico):
 - Hacer Click en un botón
 - Elegir un valor entre varios
 - Cambiar un texto
 - Cambiar las barras de desplazamiento

Eventos: Jerarquía y Niveles de Eventos



Los eventos de alto nivel están sombreados en gris

Eventos

- Cuando se llama a una función que procesa un evento, definida en una clase que implementa el interfaz EventListener, se pasa como parámetro información sobre el evento producido

```
public void actionPerformed(ActionEvent event)  
{ // código para tratar el evento }
```
- Para cada evento con nombre xxxEvent, se define una interfaz llamada xxxListener en la que se definen los métodos que van a manejarse relacionados con ese evento
 - ActionEvent -> ActionListener
 - MouseEvent -> MouseListener
- Los detalles del evento pueden ser obtenidos usando métodos de acceso:
 - getActionCommand()
 - getModifiers()

Eventos: ¿Dónde implementar la interfaz?

- Después de que al AWT recibe el evento transmitido desde el sistema operativo, el AWT crea un objeto de una subclase de AWTEvent. Este objeto tiene que ser transmitido a un determinado método para que lo gestione. Este método está definido en una interface Listener, que hay que asociar al componente.
 - Hay que asociar al componente un objeto que implementa una interface Listener (dependiendo del evento, será de una interface u otra)
 - El objeto tiene que definir un método que recibe como parámetro un evento, que es un objeto subclase de AWTEvent (dependiendo del evento, será de una subclase u otra)
 - Dentro del método es donde hay que poner el código a ejecutar cuando se produce el evento
- Dado que los Listener deben ser objetos de una clase que implemente la interfaz, surge la cuestión de qué clase debe implementar la interfaz
- Tres opciones:
 - Crear una clase que defina el componente e implemente el oyente
 - Crear una clase aparte
 - Crear una clase interna

Eventos: ¿Dónde implementar la interfaz?

- 1. Clase que define el componente y la interfaz, si la clase principal ya implementa la interface ActionListener, entonces basta con definir el método *actionPerformed* en la clase:

```
class MiClase implements ActionListener  
{public MiClase() {...  
    Button btn = new Button("Boton");  
    // Asocia el listener al boton  
    btn.addActionListener(this)...  
}  
...  
public void actionPerformed(ActionEvent e) {  
    // Aqui va el codigo de la accion }  
}
```

Eventos: ¿Dónde implementar la interfaz?

- 2. Clase aparte

```
class MiClase{  
    public MiClase() {  
        ...  
        Button btn = new Button("Boton");  
        btn.addActionListener(new MiOyente());  
        ...  
    } // del constructor  
} // de la clase  
class MiOyente implements ActionListener{  
    public void actionPerformed(ActionEvent e)  
    { // Aqui va el codigo de la accion  
    }  
}
```

Eventos: ¿Dónde implementar la interfaz?

- 3. Clase interna anónima

```
class MiClase implements ActionListener  
{public MiClase() {...  
    Button btn = new Button("Boton");  
    btn.addActionListener(  
        new ActionListener()  
        { public void actionPerformed(ActionEvent e)  
        { // Aqui va el codigo de la accion }  
        });  
    ...  
}  
}
```

La tabla muestra la relación entre los componentes del AWT y los eventos específicos que se pueden capturar, junto con una breve descripción.

Component	Eventos generados	Significado
Button	ActionEvent	Clicar en el botón
Checkbox	ItemEvent	Seleccionar o deseleccionar un item
CheckboxMenuItem	ItemEvent	Seleccionar o deseleccionar un item
Choice	ItemEvent	Seleccionar o deseleccionar un item
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 item de la lista
	ItemEvent	Seleccionar o deseleccionar un item de la lista
MenuItem	ActionEvent	Seleccionar un item 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

Interface	Listeners	Métodos Interface	Objetos
ActionListener		actionPerformed()	AbstractButton, Button, ButtonModel, ComboBoxEditor, JComboBox, JFileChooser, JTextField, List, MenuItem, TextField, Timer
AdjustmentListener		adjustmentValueChanged()	Adjustable
		Changed()	JScrollBar, Scrollbar
ComponentListener		componentHidden(), componentMoved(), componentResized(), componentShown()	Component
ContainerListener		componentAdded(), componentRemoved()	Container
FocusListener		focusGained(), focusLost()	Component
ItemListener		itemStateChanged()	AbstractButton, ButtonModel, Checkbox, CheckboxMenuItem, Choice, ItemSelectable, JComboBox, List
KeyListener		keyPressed(), keyReleased(), keyTyped()	Component
MouseListener		mouseClicked(), mouseEntered(), mouseExited(), mousePressed(), mouseReleased()	Component
MouseMotionListener		mouseDragged(), mouseMoved()	Component
TextListener		textValueChanged()	TextComponent
WindowListener		windowActivated(), windowClosed(), windowClosing(), windowDeactivated(), windowDeiconified(), windowIconified()	Window

La Tabla muestra la relación de todos los eventos que pueden ser generados por los diferentes componentes

AWT Components	Eventos que se pueden generar								
	ActionEvent	AdjustmentEvent	ComponentEvent	ContainerEvent	FocusEvent	ItemEvent	KeyEvent	MouseEvent	TextEvent
Button	✓	✓	✓	✓	✓	✓	✓	✓	
Canvas		✓		✓	✓	✓	✓	✓	
Checkbox		✓		✓	✓	✓	✓	✓	
Checkbox-MenuItem					✓	✓			
Choice		✓		✓	✓	✓	✓	✓	
Component		✓		✓		✓	✓	✓	
Container		✓	✓	✓		✓	✓	✓	
Dialog		✓	✓	✓		✓	✓	✓	✓
Frame		✓	✓	✓		✓	✓	✓	✓
Label		✓		✓		✓	✓	✓	
List	✓		✓		✓	✓	✓	✓	
MenuItem	✓								
Panel			✓	✓		✓	✓	✓	
Scrollbar		✓	✓		✓		✓	✓	
TextArea			✓		✓		✓	✓	✓
TextField	✓		✓		✓		✓	✓	✓
Window		✓	✓	✓		✓	✓	✓	✓

Relación entre eventos, Listeners y métodos de los Listeners

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()

Cada Listener es una interface. Eso quiere decir que cada vez que usa un Listener hay que implementar obligatoriamente los métodos definidos en ella.

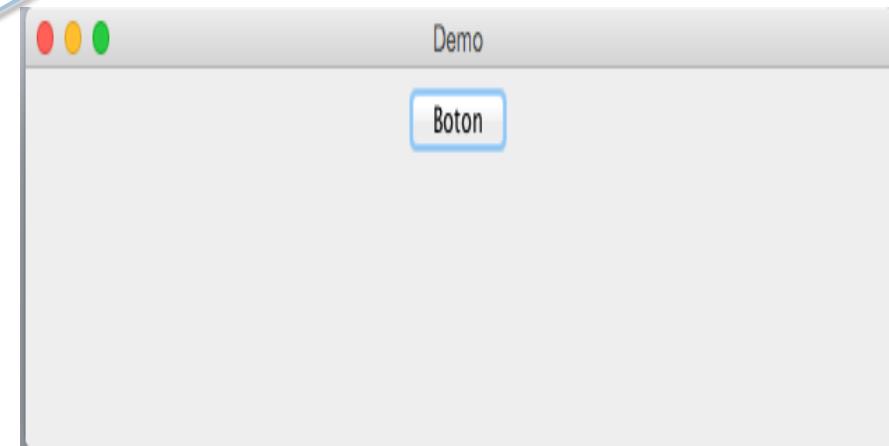
Eventos

Un evento ActionEvent se produce:
al pulsar un botón.
al hacer doble clic en un elemento de lista.
al pulsar INTRO en una caja de texto.
al elegir un menú.

```
import java.awt.*;  
import java.awt.event.*;  
public class DemoListener extends Frame  
{ private Button boton;  
  
public DemoListener()  
{super("Demo");  
setLayout(new FlowLayout());  
boton = new Button("Boton");  
// agrego un listener al boton  
boton.addActionListener(new EscuchaBoton());  
add(boton);  
setSize(200,150);  
setVisible(true); }  
  
class EscuchaBoton implements ActionListener  
{ public void actionPerformed(ActionEvent e)  
{  
System.out.println("Se presiono el boton...");}  
}  
public static void main(String[] args)  
{  
new DemoListener(); }  
}
```

En el constructor vemos que luego de instanciar el botón lo relacionamos con una instancia de la *inner class* EscuchaBoton. Esta clase implementa la *interface* ActionListener de donde hereda un único método: actionPerformed.

Cuando el botón detecte que lo están presionando notificará a la instancia de EscuchaBoton invocándole el método actionPerformed, por lo tanto, todo lo que programemos dentro de este método se ejecutará exactamente en el momento en que el usuario apriete el botón.



Se presiono el boton...

Eventos

□ Es común implementar los *listeners* como *Clase interna* de la GUI ya que, por lo general, las acciones que se programan dentro de estos son propias y exclusivas para los componentes de la interfaz gráfica, por lo que muy difícilmente puedan ser reutilizados.

un ejemplo en el que cambiamos aleatoriamente la ubicación de la ventana cada vez que se presiona el botón.

```
class EscuchaBoton implements ActionListener
{public void actionPerformed(ActionEvent e)
// dimension de la ventana
Dimension dimVentana = getSize();
// dimension de la pantalla
Dimension dimScreen = getToolkit().getScreenSize();
// nuevas coordenadas (aleatorias) para reubicar la ventana
int x= (int)(Math.random()*(dimScreen.width-dimVentana.width));
int y = (int)(Math.random()*(dimScreen.height-dimVentana.height));
// cambio la ubicacion de la ventana
setLocation(x,y); }
}
```

Eventos

- ¿Como podríamos lograr que la ventana cambie de posición cuando el *mouse* está llegando al botón para presionarlo? Es decir, que el usuario nunca pueda apretar el botón porque ni bien aproxime el *mouse* la ventana se moverá a otra posición.
- Para programar esto, tenemos que escuchar el evento de movimiento del *mouse*. Por cada movimiento del *mouse*, preguntamos sus coordenadas y las comparamos con las coordenadas del botón. Si la distancia es razonablemente corta, entonces movemos la ventana como lo hicimos en el ejemplo anterior.
- En este ejemplo escuchamos el evento que la ventana (el Frame) genera cada vez que el *mouse* se mueve dentro de su área. Por esto, le agregamos a la ventana una instancia válida de MouseMotionListener (o sea, una instancia de EscuchaMouse).

```
addMouseMotionListener(new EscuchaMouse());  
class EscuchaMouse implements MouseMotionListener  
{  
    public void mouseMoved(MouseEvent e)  
    {  
        ....  
    }  
    public void mouseDragged(MouseEvent e) {}  
}
```

EscuchaMouse es una *Clase interna* que implementa MouseMotionListener y sobrescribe los dos métodos definidos en esta *interface*. El método mouseDragged lo dejamos vacío porque no nos interesa ser notificados cuando el usuario arrastra el *mouse* (drag). Simplemente, queremos saber cuándo lo mueve. Dentro del método mouseMoved, tomamos la posición del *mouse*, la posición del botón y su dimensión para poder calcular la distancia entre el *mouse* y los cuatro lados del botón. Si la distancia es menor que distancia, entonces movemos la ventana.

Eventos

```
addMouseMotionListener(new EscuchaMouse());  
  
class EscuchaMouse implements MouseMotionListener  
{public void mouseMoved(MouseEvent e)  
{int distancia = 10;  
 Point pMouse = e.getPoint();  
 Dimension dimBoton=boton.getSize(); Point pBoton = boton.getLocation();  
 int difX1 = Math.abs(pBoton.x-pMouse.x);  
 int difX2 = Math.abs((pBoton.x+dimBoton.width)-pMouse.x);  
 int difY1 = Math.abs(pBoton.y-pMouse.y);  
 int difY2 = Math.abs((pBoton.y+dimBoton.height)-pMouse.y);  
 if(difX1<distancia||difX2<distancia||difY1<distancia||difY2 <distancia)  
 ...  
  
 setLocation(x,y); }  
 }  
public void mouseDragged(MouseEvent e) { } }
```

Eventos

- Tipos de eventos :
 - Existen eventos de tipo acción (`ActionListener`), eventos de movimiento del *mouse* (`MouseMotionListener`), eventos del teclado (`KeyListener`), eventos de la ventana (`WindowListener`), eventos de foco (`FocusListener`), etcétera.
 - Por ejemplo, si queremos detectar el evento que produce la ventana cuando el usuario la intenta cerrar debemos registrar sobre la ventana (es decir, sobre el `Frame`) un `WindowListener`. Para esto, tenemos que programar una clase que implemente esa *interface* y sobrescribir el método `windowClosing` que es el método que la ventana invocará en ese momento.

Eventos

```
public class DemoListener4 extends Frame
{public DemoListener4()
{super("Demo");
 addWindowListener(new EscuchaVentana()); // relaciono un WindowListener con el Frame
setSize(200,150);
setVisible(true); }
class EscuchaVentana implements WindowListener //clase interna
{public void windowClosing(WindowEvent e)
 { // para cerrar la ventana y finalizar el programa correctamente son tres pasos:
  // 1 – ocultar la ventana con setVisible(false)
  // 2 – liberarla con el metodo dispose
  // 3 – finalizar la aplicacion con System.exit
  System.out.println("oculto..."); setVisible(false);
  System.out.println("Libero..."); dispose();
  System.out.println("Bye bye...");
  System.exit(0);
 }
// la interface WindowListener tiene 7 metodos hay que sobrescribirlos todos aunque sea dejandolos vacios
public void windowActivated(WindowEvent e){}
public void windowClosed(WindowEvent e){}
public void windowDeactivated(WindowEvent e){}
public void windowDeiconified(WindowEvent e){}
public void windowIconified(WindowEvent e){}
public void windowOpened(WindowEvent e){}
}
```

SWING

- AWT se apoya en los componentes proporcionados por el Sistema Operativo Destino, llamados comúnmente componentes pesados, que
 - Consume muchos recursos del sistema operativo
 - No funcionan de manera exactamente igual en cada máquina, lo que da problemas de portabilidad
 - Su Aspecto "*Look&Feel*" esta ligado al Sistema donde se ejecuta (se ve diferente en cada plataforma), y no puede cambiarse
 - Swing está escrito completamente en Java y no depende tanto del Sistema (los contenedores principales en Swing son pesados, el resto no):
 - Consume menos recursos del sistema operativo
 - Mejor y mayor portabilidad
 - El *Look & Feel* puede hacerse independiente de la plataforma (para que se vea igual en todas las plataformas) o no, y se puede seleccionar
- Swing ofrece una mucho mayor gama de componentes que AWT
- Swing amplia los métodos y eventos disponibles en AWT
- Swing ofrece o amplia funcionalidades que AWT

SWING

- La metodología de trabajo con *Swing* es prácticamente idéntica a la que estudiamos con AWT. Los *listeners* trabajan de la misma manera y el funcionamiento de los *layouts* sobre los *containers* también es el mismo.
- Todos los componentes heredan de *javax.swing.Jcomponent*
- *JFrame* será la base para la aplicación principal. *JDialog* construirá los diálogos (ventanas).
- El resto de clases serán componentes simples.
- Usar en todas las clases *import javax.swing.*;*; y *import java.awt.*;*
- Los componentes que provee *Swing* son fácilmente identificables porque comienzan con el prefijo “J”. Por ejemplo: *JButton*, *JTextField*, *JPanel*, *JFrame*, *JLabel*. Los *layouts* y los *listeners* son los mismos para AWT y para *Swing*.
- <https://www.dropbox.com/s/nbpf64idfc68l9/IMAGENES.zip?dl=0>

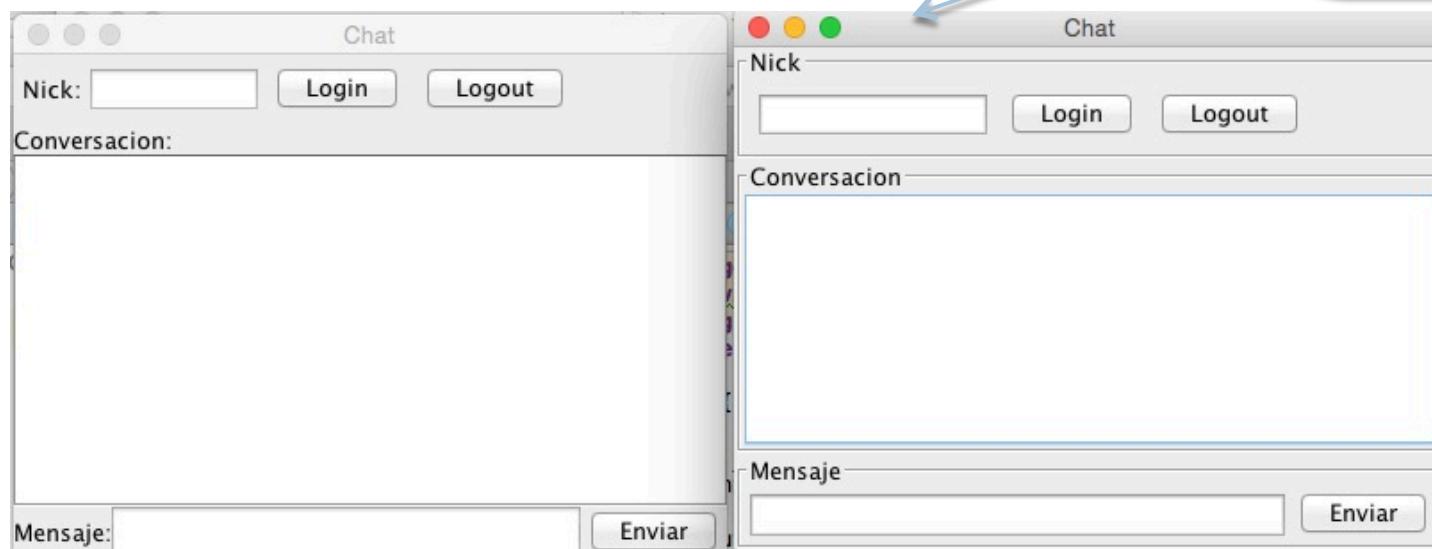
SWING

Veamos como quedaría el ejemplo del Chat con Swing.

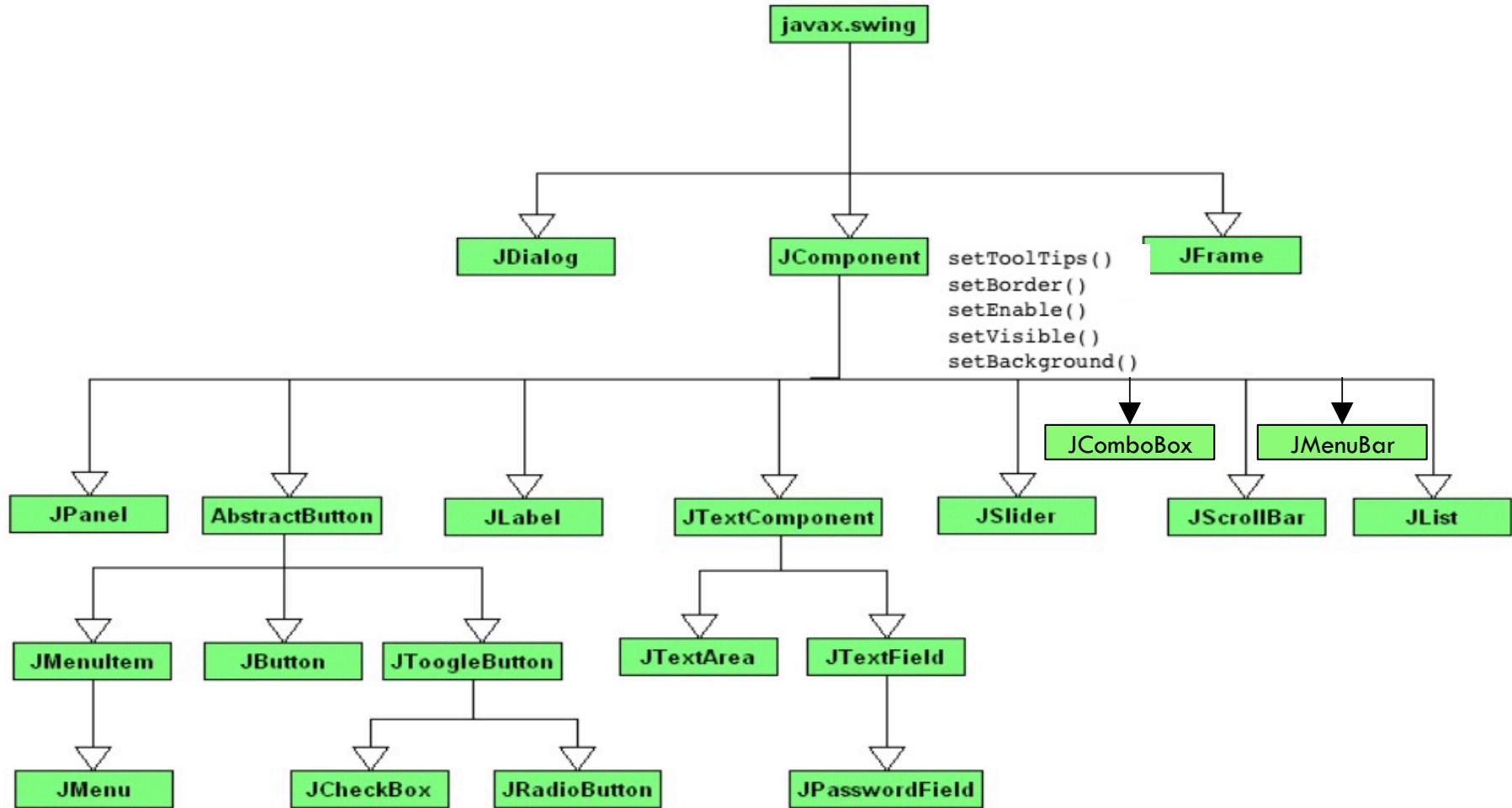
La ventana principal (una instancia de JFrame) tiene un BorderLayout. En la región norte, hay un JPanel con un FlowLayout alineado a izquierda con tres componentes: un JTextField y dos JButton.

En la región central, hay un JPanel con un BorderLayout en cuya región central hay una JList.

Por último, en la región sur hay un JPanel con un BorderLayout que tiene un JTextField en la región central y un JButton en la región este.



Clases del paquete SWING



SWING Clase JButton

- Es un botón que puede contener texto, gráficos, o ambos.

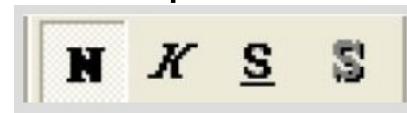


- Fijar el texto siempre centrado, en caso de contener una imagen, ha de ir a la izquierda o encima del texto.
- Métodos importantes:
 - `setText("Texto");`
`setToolTipText("Tooltip");`
`setBackground(Color.color));`
`setForeground(Color.color);`
`setIcon(new ImageIcon("ruta"));`
`setFont(new Font("tipo", estilo, tamaño));`
`setBounds(new Rectangle(posX,posY,tamX,tamY));`
 - Y sus correspondientes `get`.

SWING Clase JToggleButton

- Es un botón que representa dos estados (On y Off). Mismas características que el JButton.
Puede emplearse como dos tipos de opciones.

- Independientes (Checkboxes).



- Exclusivas (RadioButton).



- Mismos métodos que JButton, pero añadiendo algunos nuevos.

- isSelected();
 - setSelected(boolean);

SWING Clase JCheckBox

Clase JCheckBox

- Es un control que representa dos estados (On y Off).
- Métodos `isSelected()` y `setSelected(boolean)`

Clase JRadioButton

- Permiten seleccionar **una** única opción dentro de un conjunto de opciones relacionadas.

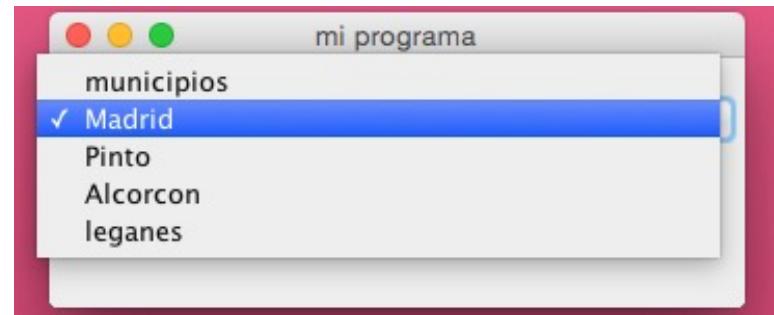
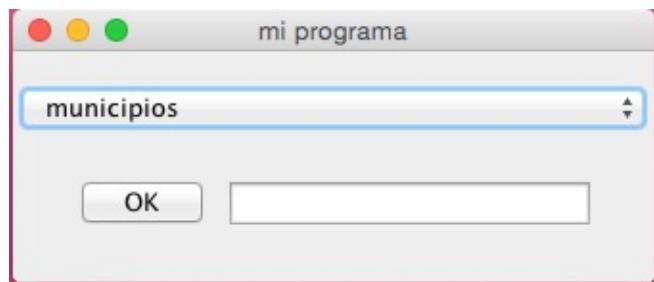
```
JRadioButton pajaroButton = new JRadioButton("pajaro");
pajaroButton.setSelected(true);
JRadioButton gatoButton = new JRadioButton("gato");
JRadioButton perroButton = new JRadioButton("perro");
JRadioButton ConejoButton = new JRadioButton("conejo");
JRadioButton caballoButton = new JRadioButton("caballo");
//Creamos un grupo y los añadimos
```

```
ButtonGroup group = new ButtonGroup();
group.add(pajaroButton);
group.add(gatoButton);
group.add(perroButton);
group.add(conejoButton);
group.add(caballoButton);
```



SWING Clase JComboBox

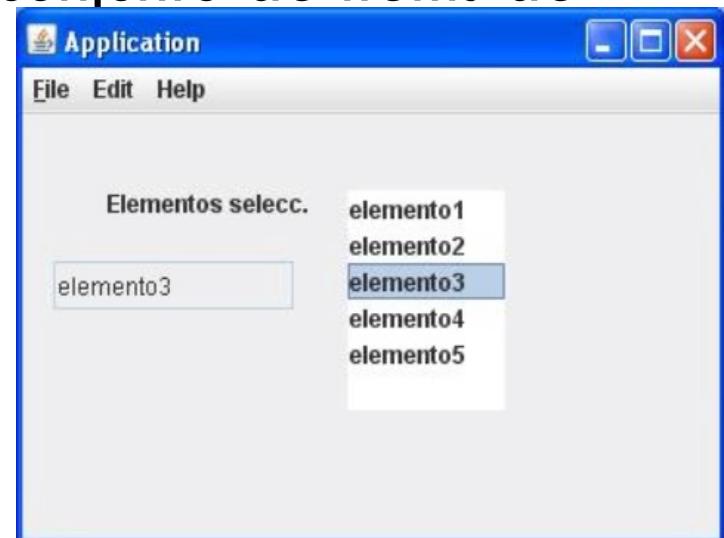
- Esta componente nos permite, al hacer click sobre ella, seleccionar una opción de entre un conjunto, todas ellas mutuamente exclusivas.



- Se pueden generalizar en dos tipos:
 - Editables No editables
- Métodos:
 - `setEditable(boolean);`
 - `addItem(Objeto);`
 - `getSelectedItem();`

SWING Clase Jlist

- Es un componente que muestra un conjunto de ítems de texto, gráfico o ambos.
- Permite tres tipos de selección:
 - Ítem único
 - Rango simple
 - Rango múltiple
- Mediante el método:
 - `.setSelectionMode(ListSelectionModel.SELECTION);`
 - Donde SELECTION puede ser:
 - `SINGLE_SELECTION` `SINGLE_INTERVAL_SELECTION`
 - `MULTIPLE_INTERVAL_SELECTION`



SWING Componentes para Texto

- JLabel Muestra texto, gráficos o ambos, sólo lectura. Ha de estar desactivado si lo está el componente.
- JTextField Muestra una linea de texto que puede ser editable. Con `setText("Texto")` se le asigna el texto.
- JPasswordField Oculta los caracteres introducidos por el usuario. `setEchoChar('char')` indica el carácter de máscara. `getPassword()` recupera el password introducido.
- JTextArea Espacio rectangular en el que ver y editar múltiples líneas de texto. Para que aparezcan barras de scroll debe ir dentro de un JScrollPane.

SWING Menús

- Los menús han de ir en la ventana principal de la aplicación.
- Es posible asignarles un gráfico.
- Cada título de menú debe tener su mnemotécnico.
- Pueden ser de tres tipos:
 - Drop-Down son los que saldrán al hacer click en Archivo
 - Submenu: son aquellos que salen como un grupo de un elemento de menú
 - Contextuales: (clase JPopupMenu) son aplicables a la región en la que está localizado el puntero del ratón. Los mnemotécnicos de los menús contextuales han de coincidir con los del menú “Drop-Down”.
 - Son las clases *JMenuBar*, *JMenu* y *JMenuItem*.

SWING Contenedores

- Clase *JToolBar* : Son botones de comando o conmutación. Se suelen emplear gráficos.
 - Métodos *setFloatable(boolean)* y *addSeparator()*
- Clase *JPanel* : Es un contenedor que agrupa componentes dentro de una ventana.
 - Los layouts permiten un correcto posicionamiento de los componentes.
- Clase *JTabbedPane*: Es un contenedor que permite tener varios componentes separador por pestañas.

SWING Diálogos

- Son ventanas mas limitadas que los Frames, y dependientes de estos, si se destruye el Frame, también lo hace el diálogo. Pueden ser:
 - No modales: No impiden interactuar con el Frame.
 - Modales: Impiden interactuar con el resto.
- Los más importantes son: JOptionPane y JFileChooser.
- JOptionPane
 - Permite adaptar y crear varias clases de diálogos, especificando por ejemplos los iconos, el título y texto de los diálogos.
 - Los iconos estándar son: *question*, *information*, *warning* y *error*.
 - Los métodos principales son:
 - `showMessageDialog` `showConfirmDialog`
- JFileChooser: Permite navegar por el sistema de ficheros, y seleccionar uno o varios ficheros. Métodos importantes:
 - `multiSelectionEnabled(boolean)`; `getSelectedFile()`

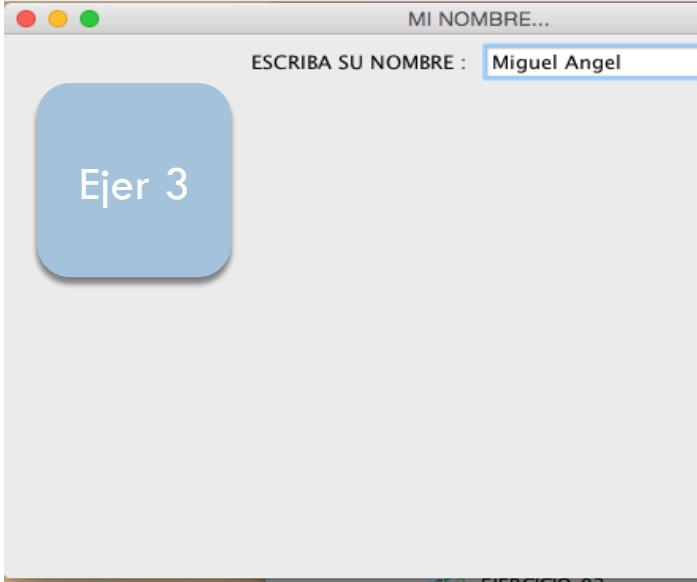
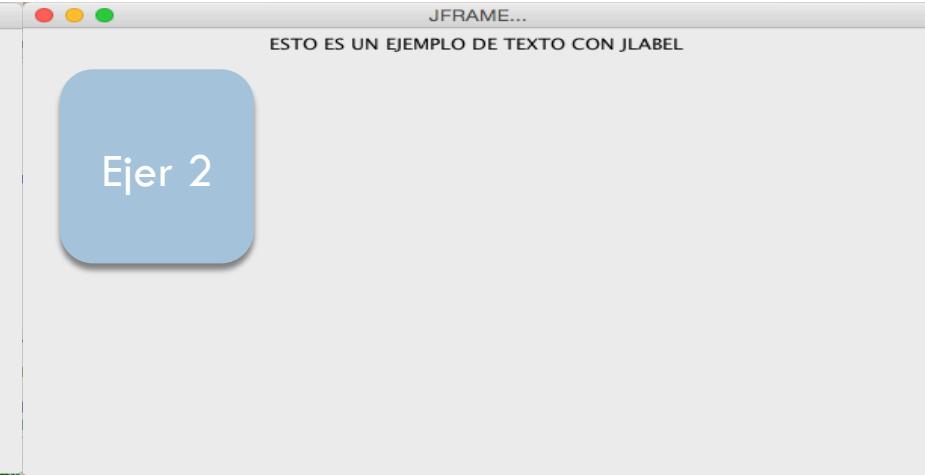
SWING Layouts

- Indican la forma de organizar los componentes dentro de un contenedor, determinando el tamaño y la posición. Para su uso:
 - Crear el contenedor.
 - Establecer el layout.
 - Agregar los componentes el contenedor.
- Tipos de layouts:
 - FlowLayout BorderLayout GridLayout BoxLayout
GridBagLayout
- Por defecto:
 - JPanel : FlowLayout
JFrame, JDialog : BorderLayout

SWING Layouts

- **FlowLayout:** Es el más simple, los componentes añadidos a un contenedor se disponen en una o mas filas, de izquierda a derecha y de arriba a abajo.
- **BorderLayout:** Utiliza 5 áreas para colocar los componentes: Norte, Sur, Este, Oeste y Centro. Si alguna no se ocupa, se expande la contigua.
- **GridLayout:**
 - El controlador se crea con un determinado numero de filas y columnas.
 - Los componentes se situan de forma secuencial, de izquierda a derecha y de arriba a abajo.
 - El tamaño de las celdas es idéntico.
- **BoxLayout:**
 - Permite organizar los componentes en una línea horizontal o vertical, sin dejar espacio entre los componentes.

SWING



Ejercicio 1

```
import javax.swing.*;  
  
public class EJERCICIO_01 extends JFrame  
{  
    //-- LE AGREGAMOS TODO A LA VENTANA MEDIANTE EL CONSTRUCTOR  
  
    public EJERCICIO_01()  
    {  
        //-- CONFIGURAMOS LA VENTANA  
  
        super("MI INTERFAZ CON JFRAME..."); //-- LE PONEMOS UN TITULO  
        this.setSize(600,400); //-- LE DAMOS UN TAMAÑO A LA VENTANA  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //-- HACEMOS QUE LA VENTANA SE CIERRE POR DEFECTO  
    }  
  
    public static void main(String[] ARGs)  
    {  
        //-- HACEMOS QUE SE INICIALICE NUESTRA VENTANA JFRAME  
  
        EJERCICIO_01 MI_INTERFAZ = new EJERCICIO_01();  
  
        //-- HACEMOS QUE NUESTRA VENTANA SE VISIBLE  
  
        MI_INTERFAZ.setVisible(true);  
    }  
}
```

Ejercicio2

```
package mad2;
import javax.swing.*;
import java.awt.*;
public class EJERCICIO_02 extends JFrame
{ public JLabel BIENVENIDA;
  public EJERCICIO_02()
  { //-- CONFIGURAMOS LA VENTANA
    super(" JFRAME..."); //-- LE PONEMOS UN TITULO
    this.setSize(600,400); //-- LE DAMOS UN TAMAÑO A LA VENTANA
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //-- HACEMOS QUE LA VENTANA SE CIERRE
    POR DEFECTO
    FlowLayout DISTRIBUCION = new FlowLayout();//-- LE AGREGAMOS UN DISTRIBUIDOR DE
    COMPONENTES
    this.setLayout(DISTRIBUCION);
    this.BIENVENIDA = new JLabel("ESTO ES UN EJEMPLO DE TEXTO CON JLABEL");
    this.add(this.BIENVENIDA);
  }
  public static void main(String[] ARGS)
  { //-- HACEMOS QUE SE INICIE NUESTRA VENTANA JFRAME
    EJERCICIO_02 INTERFAZ = new EJERCICIO_02();
    MI_INTERFAZ.setVisible(true); //-- HACEMOS QUE NUESTRA VENTANA SE VISIBLE
  }
}
```

Ejercicio3

```
import javax.swing.*; import java.awt.*;
public class EJERCICIO_03 extends JFrame
{public JLabel INDIQUE_NOMBRE;
public JTextField MI_NOMBRE;
public EJERCICIO_03()
{
    super("MI NOMBRE...");//-- CONFIGURAMOS LA VENTANA
    this.setSize(600,400);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    FlowLayout DISTRIBUIDOR = new FlowLayout();//-- LE AGREGAMOS UN DISTRIBUIDOR DE COMPONENTES
    this.setLayout(DISTRIBUIDOR);
    //-- AGREGAMOS TODOS LOS COMPONENTES
    this.INDIQUE_NOMBRE = new JLabel("ESCRIBA SU NOMBRE : ");
    this.add(this.INDIQUE_NOMBRE);
    this.MI_NOMBRE = new JTextField("SU NOMBRE AQUI...ii");
    this.add(this.MI_NOMBRE); }

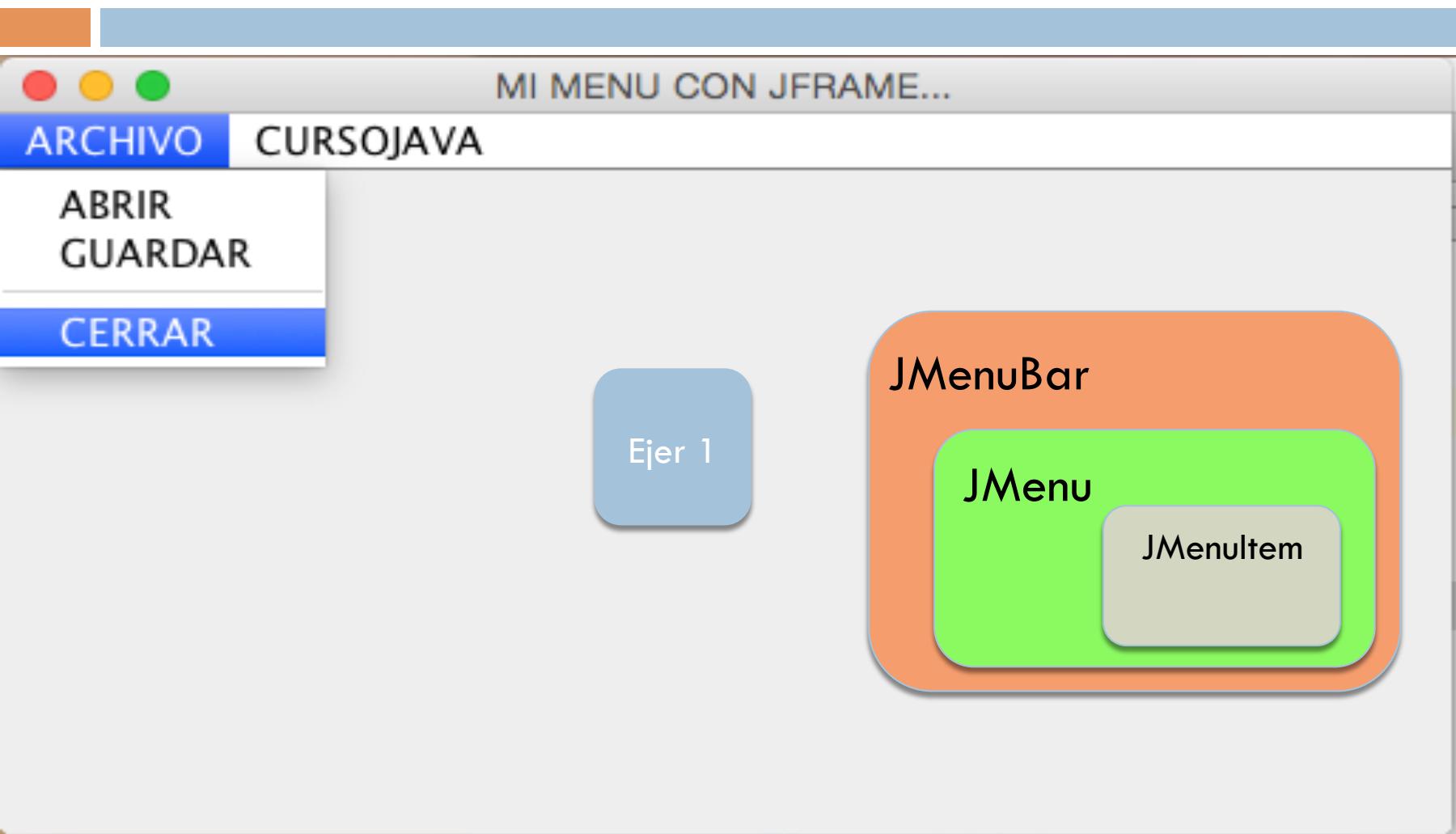
public static void main(String[] ARGS)
{ //-- HACEMOS QUE SE INICIALIZE NUESTRA VENTANA JFRAME
    EJERCICIO_03 MI_INTERFAZ = new EJERCICIO_03();
    //-- HACEMOS QUE NUESTRA VENTANA SE VISIBLE
    MI_INTERFAZ.setVisible(true);
}
}
```

Ejercicio4

```
import javax.swing.*; import java.awt.*;
public class EJERCICIO_04 extends JFrame
{ public JTextArea MI_CAJA_TEXTO; public JLabel MI_TEXTO; public JComboBox LISTA_DESPLEGABLE;
public EJERCICIO_05()
{ super("MI AREA DE TEXTO");
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
FlowLayout DISTRIBUCION = new FlowLayout();
this.setLayout(DISTRIBUCION);
this.MI_TEXTO = new JLabel();
this.MI_TEXTO.setText("ESCRIBA LO QUE SEA EN LA CAJA DE TEXTO");
this.add(this.MI_TEXTO);
this.MI_CAJA_TEXTO = new JTextArea(10,20);
this.add(MI_CAJA_TEXTO);
this.LISTA_DESPLEGABLE = new JComboBox();
this.LISTA_DESPLEGABLE.addItem("BANDAS DE ROCK");
this.LISTA_DESPLEGABLE.addItem("GUN'S ROSES");
this.LISTA_DESPLEGABLE.addItem("PINK FLOYD");
this.LISTA_DESPLEGABLE.addItem("METALLICA");
this.add(this.LISTA_DESPLEGABLE); }
public static void main(String[] ARGS)
{ EJERCICIO_05 MI_INTERFAZ = new EJERCICIO_05();
MI_INTERFAZ.pack();
MI_INTERFAZ.setVisible(true);
}
```

EL METODO PACK SIRVE PARA DAR TAMAÑO A TODOS LOS COMPONENTES POR DEFECTO

SWING Menus



Ejercicio 1

```
import javax.swing.*;
public class EJERCICIO_01 extends JFrame
{ public JMenu MENU_ARCHIVO,MENU_JAVA;
  public JMenuItem ITEM_ABRIR;
  public JMenuItem ITEM_GUARDAR;
  public JMenuItem ITEM_CERRAR;
  public JMenuItem ITEM_JAVADHC;
  public JMenuItem ITEM_MAD1;
  public JMenuItem ITEM_MAD2;
  public JMenuItem ITEM_MAD3;
  public JMenuBar MENU_BAR;
  public EJERCICIO_01()
  { super("MI MENU CON JFRAME...");
    this.setSize(1200,800);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.ITEM_ABRIR = new JMenuItem("ABRIR");
    this.ITEM_GUARDAR = new JMenuItem("GUARDAR");
    this.ITEM_CERRAR = new JMenuItem("CERRAR");
    this.ITEM_JAVADHC = new JMenuItem("CURSOJAVA");
    this.ITEM_MAD1 = new JMenuItem("MAD1");
    this.ITEM_MAD2 = new JMenuItem("MAD2");
    this.ITEM_MAD3 = new JMenuItem("MAD3");
```

Ejercicio 1

```
this.MENU_ARCHIVO = new JMenu("ARCHIVO");
this.MENU_ARCHIVO.add(this.ITEM_ABRIR);
this.MENU_ARCHIVO.add(this.ITEM_GUARDAR);
this.MENU_ARCHIVO.addSeparator();
this.MENU_ARCHIVO.add(this.ITEM_CERRAR);
this.MENU_JAVA = new JMenu("CURSOJAVA");
this.MENU_JAVA.add(this.ITEM_JAVADHC);
this.MENU_JAVA.add(this.ITEM_MAD1);
this.MENU_JAVA.addSeparator();
this.MENU_JAVA.add(this.ITEM_MAD2);
this.MENU_JAVA.add(this.ITEM_MAD3);

this.MENU_BAR = new JMenuBar();
this.MENU_BAR.add(this.MENU_ARCHIVO);
this.MENU_BAR.add(this.MENU_JAVA);

this.setJMenuBar(this.MENU_BAR);
}

public static void main(String[] ARGS)
{
    EJERCICIO_01 MI_INTERFAZ = new EJERCICIO_01();
    MI_INTERFAZ.setVisible(true);
}
```

SWING Menus



Ejercicio 2

```
import javax.swing.*;
import java.awt.*;
public class EJERCICIO_02 extends JFrame
{ public JButton BTN_MUSICA,BTN_VIDEO,BTN_INTERNET,BTN_JAVADHC,BTN_INFO;
  public JToolBar CAJA_HERRAMIENTA_01,CAJA_HERRAMIENTA_02;
  public ImageIcon IMG_VIDEO,IMG_MUSICA,IMG_INTERNET,IMG_JAVADHC,IMG_INFO;
  public EJERCICIO_02()
  { super("MI MENU CON JFRAME...");
    this.setSize(600,400);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.IMG_VIDEO = new ImageIcon("IMAGENES/VIDEO.PNG");
    this.IMG_MUSICA = new ImageIcon("IMAGENES/MUSICA.PNG");
    this.IMG_INTERNET = new ImageIcon("IMAGENES/INTERNET.PNG");
    this.IMG_JAVADHC = new ImageIcon("IMAGENES/JAVADHC.PNG");
    this.IMG_INFO = new ImageIcon("IMAGENES/INFO1.PNG");
    this.BTN_VIDEO = new JButton("VIDEO",this.IMG_VIDEO);
    this.BTN_MUSICA = new JButton("MUSICA",this.IMG_MUSICA);
    this.BTN_INTERNET = new JButton("INTERNET",this.IMG_INTERNET);
    this.BTN_JAVADHC = new JButton("CURSOJAVA",this.IMG_JAVADHC);
    this.BTN_INFO = new JButton("Este es el curso de ANDROID",this.IMG_INFO);
```

Ejercicio 2

```
this.CAJA_HERRAMIENTA_01 = new JToolBar();
this.CAJA_HERRAMIENTA_01.add(this.BTN_VIDEO);
this.CAJA_HERRAMIENTA_01.add(this.BTN_MUSICA);
this.CAJA_HERRAMIENTA_01.add(this.BTN_INTERNET);

this.CAJA_HERRAMIENTA_02 = new JToolBar();
this.CAJA_HERRAMIENTA_02.add(this.BTN_JAVADHC);
this.CAJA_HERRAMIENTA_02.add(this.BTN_INFO);

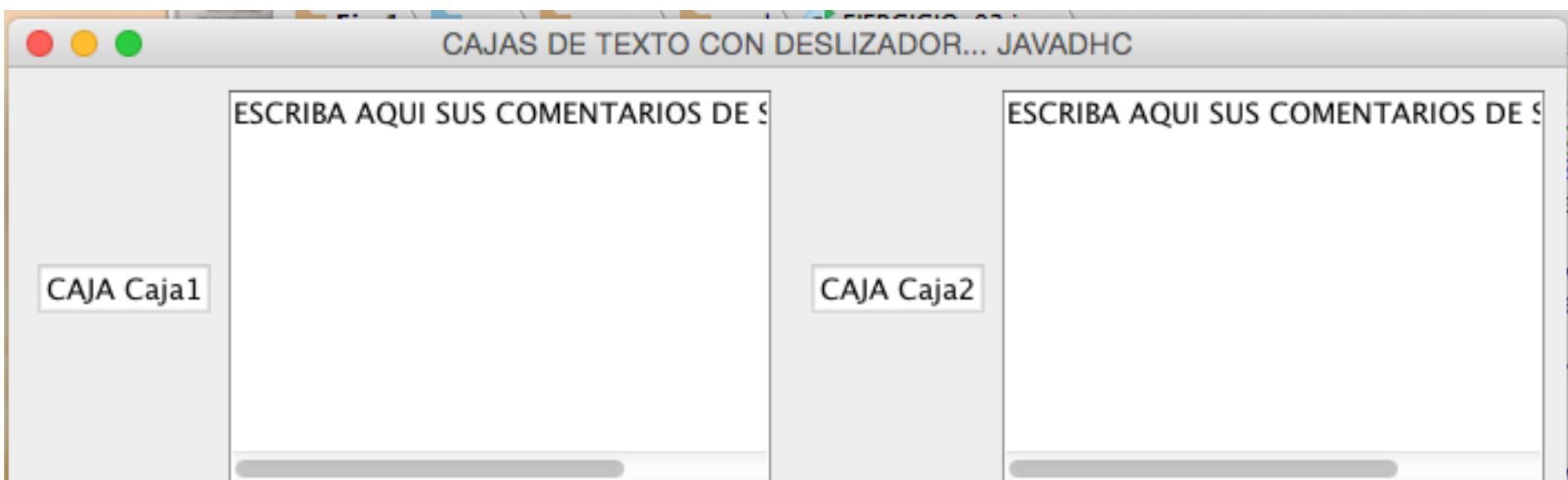
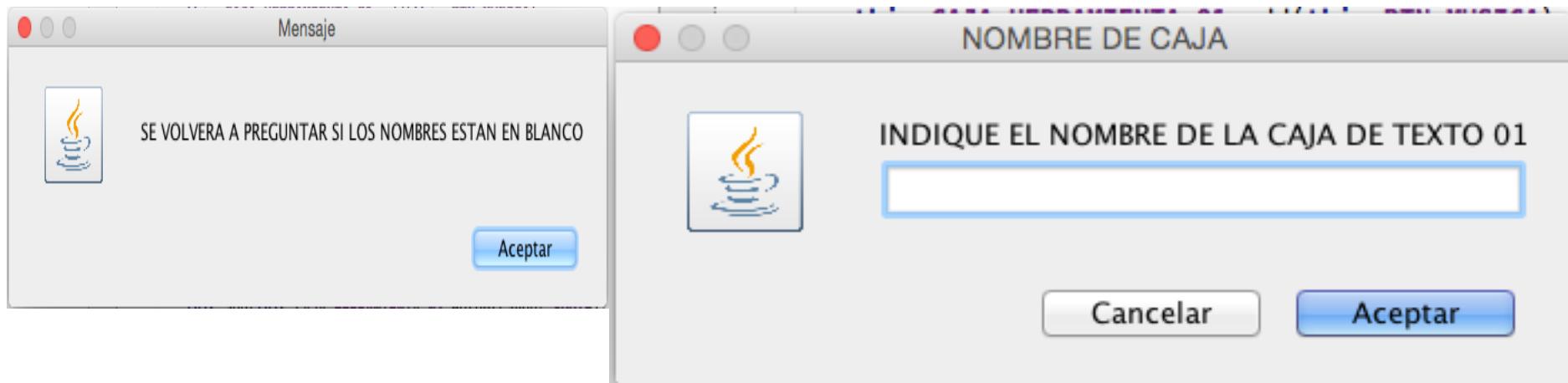
BorderLayout DISTRIBUIDOR = new BorderLayout();
this.setLayout(DISTRIBUIDOR);

this.add(this.CAJA_HERRAMIENTA_01, BorderLayout.NORTH);
this.add(this.CAJA_HERRAMIENTA_02, BorderLayout.SOUTH);
}

public static void main(String[] ARGUMENTOS)
{
    EJERCICIO_02 MI_INTERFAZ = new EJERCICIO_02();
    //MI_INTERFAZ.pack();
    MI_INTERFAZ.setVisible(true);
}
}
```

SWING Menus

Ejer 3



Ejercicio 3

```
import javax.swing.*; import java.awt.*;
public class EJERCICIO_03 extends JFrame
{    public MI_PANEL PANEL_01, PANEL_02;
    public JScrollPane DESLIZADOR;
    public EJERCICIO_03()
    { super("CAJAS DE TEXTO CON DESLIZADOR... JAVADHC");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        String NOMBRE_CAJA_01, NOMBRE_CAJA_02;
        do {JOptionPane.showMessageDialog(null,"SE VOLVERA A PREGUNTAR SI LOS NOMBRES ESTAN
EN BLANCO");
            NOMBRE_CAJA_01 = JOptionPane.showInputDialog(null,"INDIQUE EL NOMBRE DE LA CAJA
DE TEXTO 01","NOMBRE DE CAJA",JOptionPane.INFORMATION_MESSAGE);
            NOMBRE_CAJA_02 = JOptionPane.showInputDialog(null,"INDIQUE EL NOMBRE DE LA CAJA
DE TEXTO 02","NOMBRE DE CAJA",JOptionPane.INFORMATION_MESSAGE); }
        while(NOMBRE_CAJA_01.isEmpty() || NOMBRE_CAJA_02.isEmpty());
        FlowLayout DISTRIBUIDOR = new FlowLayout();
        this.setLayout(DISTRIBUIDOR);
        PANEL_01 = new MI_PANEL(NOMBRE_CAJA_01);
        PANEL_02 = new MI_PANEL(NOMBRE_CAJA_02);
        this.add(PANEL_01);
        this.add(PANEL_02);}
    public static void main(String[] ARGUMENTOS)
    { EJERCICIO_03 MI_INTERFAZ = new EJERCICIO_03();
        MI_INTERFAZ.pack();
        MI_INTERFAZ.setVisible(true);
    }
}
```

Ejercicio 3

```
class MI_PANEL extends JPanel
{
    public JScrollPane DESLIZADOR;
    public JTextArea COMENTARIOS;
    public JTextField NOMBRE_CAJA;

    public MI_PANEL(String NOMBRE)
    {
        super();

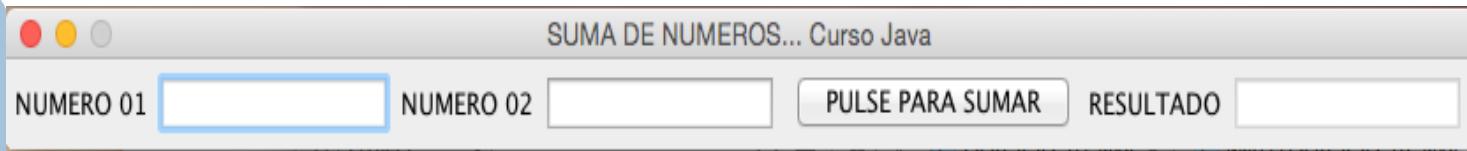
        this.NOMBRE_CAJA = new JTextField();
        this.NOMBRE_CAJA.setText("CAJA " + NOMBRE);
        this.NOMBRE_CAJA.setEditable(false);
        this.add(this.NOMBRE_CAJA);

        this.COMENTARIOS = new JTextArea(10,20);
        this.COMENTARIOS.setText("ESCRIBA AQUI SUS COMENTARIOS DE SU CAJA " + NOMBRE);

        this.DESLIZADOR = new JScrollPane(this.COMENTARIOS);
        this.add(this.DESLIZADOR);
    }
}
```

SWING Menus

Ejer 4



Ejer 5

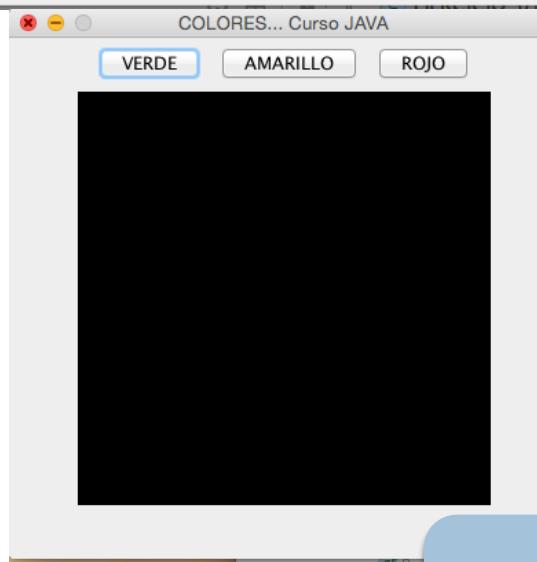
SUMA CON 4 OPERACIONES

NUMERO 01

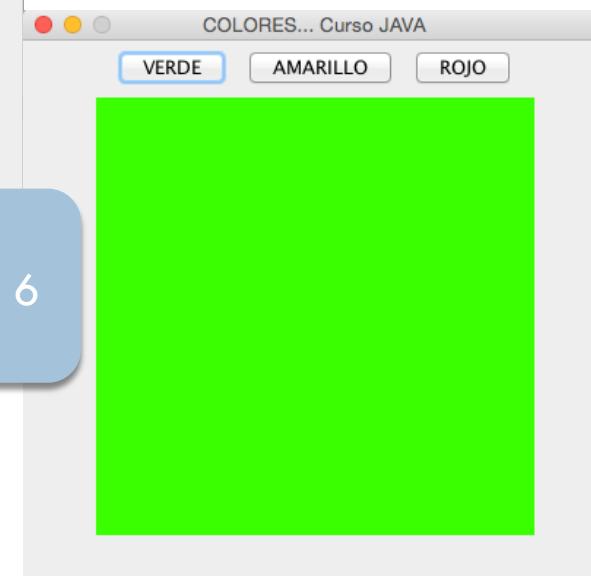
NUMERO 02

SUMAR RESTAR

MULTIPLICAR DIVIDIR



Ejer 6



Ejercicio 4

```
import java.awt.*;import java.awt.event.*;import javax.swing.*;
public class EJERCICIO_04 extends JFrame implements ActionListener
{ public JLabel LETRA_NUMERO_01,LETRA_NUMERO_02,LETRO_RESULTADO;
  public JTextField NUMERO_01,NUMERO_02,RESULTADO;
  public JButton SUMAR;
  public EJERCICIO_04()
  {   super("SUMA DE NUMEROS... Curso Java");
      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      this.setSize(310,460);
      this.LETRA_NUMERO_01 = new JLabel("NUMERO 01");
      this.LETRA_NUMERO_02 = new JLabel("NUMERO 02");
      this.LETRO_RESULTADO = new JLabel("RESULTADO");
      this.NUMERO_01 = new JTextField(10);
      this.NUMERO_02 = new JTextField(10);
      this.RESULTADO = new JTextField(10);
      this.RESULTADO.setEditable(false);
      this.SUMAR = new JButton("PULSE PARA SUMAR");
      this.SUMAR.addActionListener(this);
      FlowLayout DISTRIBUIDOR = new FlowLayout();
      this.setLayout(DISTRIBUIDOR);
      this.add(this.LETRA_NUMERO_01);
      this.add(this.NUMERO_01);
      this.add(this.LETRA_NUMERO_02);
      this.add(this.NUMERO_02);
      this.add(this.SUMAR);
      this.add(this.LETRO_RESULTADO);
      this.add(this.RESULTADO);
  }
}
```

Ejercicio 4

```
public void actionPerformed(ActionEvent EVENTO)
{ float AUX_NUMERO_01,AUX_NUMERO_02,AUX_RESULTADO;
  try
  {   AUX_NUMERO_01 = Float.parseFloat(this.NUMERO_01.getText());
      AUX_NUMERO_02 = Float.parseFloat(this.NUMERO_02.getText());
      AUX_RESULTADO = AUX_NUMERO_01 + AUX_NUMERO_02;
      this.RESULTADO.setText(String.valueOf(AUX_RESULTADO));
  }
  catch(Exception E)
  {
      this.RESULTADO.setText("ERROR AL SUMAR");
  }
}
public static void main(String[] ARGUMENTOS)
{
    EJERCICIO_04 MI_INTERFAZ = new EJERCICIO_04();
    MI_INTERFAZ.setResizable(false);

    MI_INTERFAZ.setVisible(true);
}
}
```

Ejercicio 5

```
public class EJERCICIO_05 extends JFrame implements ActionListener
{
    JRadioButton SUMAR, RESTAR, MULTIPLICAR, DIVIDIR;
    ButtonGroup OPERACIONES;
    JButton EJECUTAR_OPERACION;
    JLabel LETRA_NUMERO_01, LETRA_NUMERO_02, LETRA_RESULTADO;
    JTextField NUMERO_01, NUMERO_02, RESULTADO;

    public EJERCICIO_05()
    {
        super("SUMA CON 4 OPERACIONES");
        this.setSize(270,310);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.LETRA_NUMERO_01 = new JLabel("NUMERO 01");
        this.LETRA_NUMERO_02 = new JLabel("NUMERO 02");
        this.LETRA_RESULTADO = new JLabel("RESULTADO");
        this.NUMERO_01 = new JTextField(10);
        this.NUMERO_02 = new JTextField(10);
        this.RESULTADO = new JTextField(10);
        this.EJECUTAR_OPERACION = new JButton("EJECUTAR OPERACION");
        this.EJECUTAR_OPERACION.addActionListener(this);
    }
}
```

Ejercicio 5

```
this.SUMAR = new JRadioButton("SUMAR",true);
this.RESTAR = new JRadioButton("RESTAR",false);
this.MULTIPLICAR = new JRadioButton("MULTIPLICAR",false);
this.DIVIDIR = new JRadioButton("DIVIDIR",false);
this.OPERACIONES = new ButtonGroup();
this.OPERACIONES.add(this.SUMAR);
this.OPERACIONES.add(this.RESTAR);
this.OPERACIONES.add(this.MULTIPLICAR);
this.OPERACIONES.add(this.DIVIDIR);
FlowLayout DISTRIBUIDOR = new FlowLayout();
this.setLayout(DISTRIBUIDOR);
this.add(this.LETRA_NUMERO_01);
this.add(this.NUMERO_01);
this.add(this.LETRA_NUMERO_02);
this.add(this.NUMERO_02);
this.add(this.SUMAR);
this.add(this.RESTAR);
this.add(this.MULTIPLICAR);
this.add(this.DIVIDIR);
this.add(this.EJECUTAR_OPERACION);
this.add(this.LETRA_RESULTADO);
this.add(this.RESULTADO);
}
```

Ejercicio 5

```
public void actionPerformed(ActionEvent EVENTO)
{float AUX_NUMERO_01,AUX_NUMERO_02,AUX_RESULTADO;
 try{AUX_RESULTADO = 0;
    AUX_NUMERO_01 = Float.parseFloat(this.NUMERO_01.getText());
    AUX_NUMERO_02 = Float.parseFloat(this.NUMERO_02.getText());
    if(this.SUMAR.isSelected())
    {AUX_RESULTADO = AUX_NUMERO_01 + AUX_NUMERO_02; }
    else if(this.RESTAR.isSelected())
    {AUX_RESULTADO = AUX_NUMERO_01 - AUX_NUMERO_02; }
    else if(this.MULTIPLICAR.isSelected())
    {AUX_RESULTADO = AUX_NUMERO_01 * AUX_NUMERO_02; }
    else if(this.DIVIDIR.isSelected())
    {AUX_RESULTADO = AUX_NUMERO_01 / AUX_NUMERO_02; }
    this.RESULTADO.setText(String.valueOf(AUX_RESULTADO));
    }
 catch(Exception E)
 { this.RESULTADO.setText("ERROR AL EJECUTAR"); }
}
public static void main(String[] ARGUMENTOS)
{EJERCICIO_05 MI_INTERFAZ = new EJERCICIO_05();
 MI_INTERFAZ.pack();
 MI_INTERFAZ.setResizable(false);
 MI_INTERFAZ.setVisible(true); }
```

Ejercicio 6

```
public class EJERCICIO_06 extends JFrame implements ActionListener
{
    public JButton VERDE,AMARILLO,ROJO;
    public JPanel COLOR_PANEL;
    public EJERCICIO_06()
    {
        super("COLORES... Curso JAVA");
        this.setSize(400,400);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.VERDE = new JButton("VERDE");
        this.VERDE.setBackground(Color.GREEN);
        this.VERDE.addActionListener(this);
        this.AMARILLO = new JButton("AMARILLO");
        this.AMARILLO.setBackground(yellow);
        this.AMARILLO.addActionListener(this);
        this.ROJO = new JButton("ROJO");
        this.ROJO.setBackground(Color.RED);
        this.ROJO.addActionListener(this);
        this.COLOR_PANEL = new JPanel();
        this.COLOR_PANEL.setPreferredSize(new Dimension(300,300));
        this.COLOR_PANEL.setBackground(Color.BLACK);
        FlowLayout DISTRIBUIDOR_FRAME = new FlowLayout();
        this.setLayout(DISTRIBUIDOR_FRAME);
        this.add(this.VERDE);
        this.add(this.AMARILLO);
        this.add(this.ROJO);
        this.add(this.COLOR_PANEL);
    }
}
```

Ejercicio 6

```
public void actionPerformed(ActionEvent EVENTO)
{
    Object BOTON_SELECCIONADO = EVENTO.getSource();

    if(BOTON_SELECCIONADO == this.VERDE)
    {
        this.COLOR_PANEL.setBackground(Color.GREEN);
    }
    else if(BOTON_SELECCIONADO == this.AMARILLO)
    {
        this.COLOR_PANEL.setBackground(Color.YELLOW);
    }
    else if(BOTON_SELECCIONADO == this.ROJO)
    {
        this.COLOR_PANEL.setBackground(Color.RED);
    }
}

public static void main(String[] ARGUMENTOS)
{
    EJERCICIO_06 MI_INTERFAZ = new EJERCICIO_06();
    MI_INTERFAZ.setResizable(false);
    MI_INTERFAZ.setVisible(true);
}
```

SWING

El código del chat empleando Swing sería el siguiente:

```
import java.awt.*;
import javax.swing.*; import javax.swing.border.*;
public class ChatSwing extends JFrame {
    private JTextField tfNick;
    private JTextField tfMensaje;
    private JButton bLogin;
    private JButton bLogout;
    private JButton bEnviar;
    private JList lstLog;
    private Border border;
    public ChatSwing() {
        super("Chat");
        Container content = getContentPane();
        content.setLayout(new BorderLayout());
        border = BorderFactory.createEtchedBorder(EtchedBorder.LOWERED);
        JPanel pNorth = _crearPNorte();
        content.add(pNorth, BorderLayout.NORTH);
        JPanel pCenter = _crearPCenter();
        content.add(pCenter, BorderLayout.CENTER);
        JPanel pSouth = _crearPSur();
        content.add(pSouth, BorderLayout.SOUTH);
        setSize(400, 300);
        setVisible(true);
    }
}
```

En Swing no trabajamos sobre el JFrame sino que lo hacemos sobre un panel contenedor que se llama content y que lo obtenemos con el método getContentPane.

El layout de la ventana principal lo definimos sobre este content. Los componentes que contendrá la ventana principal también los agregamos sobre content

pedimos el "panel contenedor" al JFrame

configuramos el layout

este será el tipo de borde que utilizamos en todos los paneles

creamos el panel norte, central y sur

SWING

Con respecto de la versión de AWT instanciamos un **TitledBorder** (a través de la clase **BorderFactory**) para asignarlo al **JPanel**.

```
private JPanel _crearPNorte() {  
    JPanel p = new JPanel(new FlowLayout(FlowLayout.LEFT));  
    // que cree en el constructor  
    TitledBorder titulo = BorderFactory.createTitledBorder(border, "Nick");  
    p.setBorder(titulo);  
    tfNick = new JTextField(10);  
    p.add(tfNick);  
    bLogin = new JButton("Login");  
    p.add(bLogin);  
    bLogout = new JButton("Logout");  
    p.add(bLogout);  
    return p;  
}
```

instancio un **TitledBorder** con el **titulo** y el objeto **border** creado en el constructor

añado el **JtextField** con el metodo **add** del **Jpanel**

```
private JPanel _crearPCenter() {  
    JPanel p = new JPanel(new BorderLayout());  
    TitledBorder titulo = BorderFactory.createTitledBorder(border, "Conversacion");  
    p.setBorder(titulo);  
    lstLog = new JList();  
    JScrollPane scroll = new JScrollPane(lstLog);  
    p.add(scroll, BorderLayout.CENTER);  
    return p;  
}
```

SWING

```
private JPanel _crearPSur() {
    JPanel p = new JPanel(new BorderLayout());
    TitledBorder titulo = BorderFactory.createTitledBorder(border, "Mensaje");
    p.setBorder(titulo);
    tfMensaje = new JTextField();
    p.add(tfMensaje, BorderLayout.CENTER);
    bEnviar = new JButton("Enviar");
    p.add(bEnviar, BorderLayout.EAST);
    return p;
}

public static void main(String args[]) throws Exception {
    ChatSwing c = new ChatSwing();
}
}
```

SWING

- Cambiar el Aspecto del GUI: Swing tiene la posibilidad de definir un aspecto propio e independiente de la plataforma. En otras palabras, con Swing una misma GUI puede verse con distintos estilos. Para esto, Java provee las clases WindowLookAndFeel, MotifLookAndFeel y MetalLookAndFeel.
- Podemos cambiar el aspecto. Lo hacemos agregando una línea en el método main. Para ello hay que importar el paquete javax.swing.*

```
public static void main(String args[]) throws Exception
```

```
{  
    UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");  
    ChatSwing c = new ChatSwing();  
}
```

- Para obtener los diferentes aspectos disponibles en nuestra plataforma se puede usar el siguiente código:

```
Import javax.swing.UIManager.*;  
LookAndFeelInfo[] lista = UIManager.getInstalledLookAndFeels();  
for (int i = 0; i < lista.length; i++) {  
    System.out.println(lista[i].getClassName());
```

SWING

- Hacer una calculadora:



THREADS

- Un proceso es un programa ejecutándose de forma independiente y con un espacio propio de memoria. Un Sistema Operativo multitarea es capaz de ejecutar más de un proceso simultáneamente. Un thread o hilo es un flujo secuencial simple dentro de un proceso. Un único proceso puede tener varios hilos ejecutándose.
- Un sistema multitarea da realmente la impresión de estar haciendo varias cosas a la vez y eso es una gran ventaja para el usuario. Sin el uso de threads hay tareas que son prácticamente imposibles de ejecutar, particularmente las que tienen tiempos de espera importantes entre etapas.
- Los threads o hilos de ejecución permiten organizar los recursos del ordenador de forma que pueda haber varios programas actuando en paralelo. Un hilo de ejecución puede realizar cualquier tarea que pueda realizar un programa normal y corriente. Bastará con indicar lo que tiene que hacer en el método run(), que es el que define la actividad principal de las threads.
- Los threads pueden ser daemon o no daemon. Son daemon aquellos hilos que realizan en background (en un segundo plano)
- Un programa de Java finaliza cuando sólo quedan corriendo threads de tipo daemon. Por defecto, y si no se indica lo contrario, los threads son del tipo no daemon.

CREACIÓN DE THREADS

- En Java hay dos formas de crear nuevos threads.
 - La primera de ellas consiste en crear una nueva clase que herede de la clase `java.lang.Thread` y sobrecargar el método `run()` de dicha clase.
 - El segundo método consiste en declarar una clase que implemente la interface `java.lang.Runnable`, la cual declarará el método `run()`; posteriormente se crea un objeto de tipo `Thread` pasándole como argumento al constructor el objeto creado de la nueva clase (la que implementa la interface `Runnable`)

CREACIÓN DE THREADS

□ Creación de threads derivando de la clase Thread

```
public class SimpleThread extends Thread { // constructor  
public SimpleThread (String str) { super(str); }  
public void run() { // redefinición del método run()  
for(int i=0;i<10;i++)  
System.out.println("Este es el thread : " + getName());  
} }
```

- Para poner en marcha este nuevo thread se debe crear un objeto de la clase SimpleThread, y llamar al método start(), heredado de la super-clase Thread, que se encarga de llamar a run(). Por ejemplo:

```
SimpleThread miThread = new SimpleThread("Hilo de prueba");  
miThread.start();
```

CREACIÓN DE THREADS

□ Creación de threads implementando la interface Runnable

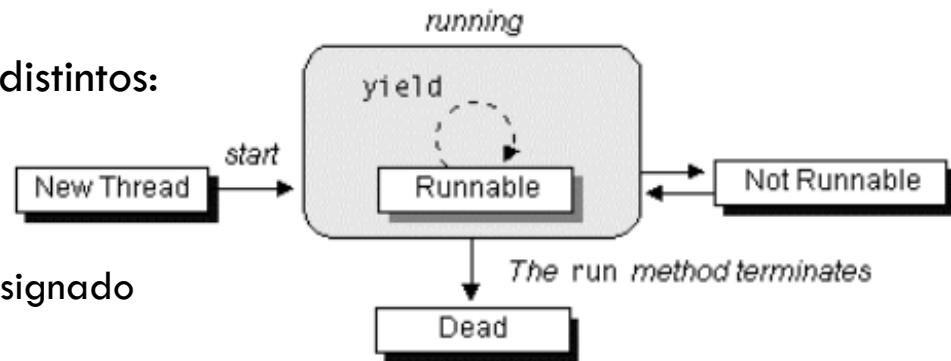
```
public class SimpleRunnable implements Runnable { // se crea un nombre  
    String nameThread;  
public SimpleRunnable (String str) // constructor  
{ nameThread = str; }  
public void run() // definición del método run()  
{for(int i=0;i<10;i++) System.out.println("Este es el thread: " + nameThread);  
}
```

□ código crea un nuevo thread :

```
SimpleRunnable p = new SimpleRunnable("Hilo de prueba");  
// se crea un objeto de la clase Thread pasándolo el objeto Runnable como argumento Thread  
miThread = new Thread(p);  
// se arranca el objeto de la clase Thread  
miThread.start();
```

- Un thread puede presentar cuatro estados distintos:

- Nuevo (New): El thread ha sido creado pero no inicializado.
- Ejecutable (Runnable): El thread puede estar ejecutándose, siempre y cuando se le haya asignado un determinado tiempo de CPU.
- Bloqueado (Blocked o Not Runnable): El thread podría estar ejecutándose, pero hay alguna actividad interna suya que lo impide, como por ejemplo una espera producida por una operación de escritura o lectura de datos por teclado (E/S). Si un thread está en este estado, no se le asigna tiempo de CPU.
- Muerto (Dead): La forma habitual de que un thread muera es finalizando el método run(). También puede llamarse al método stop() de la clase Thread, aunque no es aconsejable



THREADS

- Threads e interfaz gráfica
- Cuando en una GUI alguno de los componentes da origen a un proceso que puede llegar a demorar, tenemos que identificarlo y lanzarlo en su propio hilo de ejecución ya que de lo contrario toda la interfaz gráfica quedará bloqueada e inutilizada mientras que el proceso iniciado no finalice, lo que puede dar al usuario una idea del mal funcionamiento general.
- Creamos una interfaz gráfica que tiene un botón y un choice (una de lista de donde se puede seleccionar un ítem). Cuando se presiona el botón “dor- mimos” 10 segundos simulando que se invocó a un proceso que demora ese tiempo (podría ser una consulta a una base de datos por ejemplo).
- Podrá verificar (si ejecuta este programa) que una vez que se presiona el botón, durante los siguientes 10 segundos, la GUI queda inutilizada y la sensación será de que algo no está funcionando bien.
- Para evitar esto , en el método actionPerformed instanciamos y lanzamos un thread de la *inner class* TareaBoton en cuyo método run hacemos lo mismo que hacíamos antes en el método actionPerformed.

Ejemplo GUI en espera

```
□ import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class VentanaDemora extends Frame
{private Button boton; private Choice combo;
 public VentanaDemora()
 { setLayout(new FlowLayout());
  add( boton = new Button("Esto va a demorar...") );
  boton.addActionListener(new EscuchaBoton());
  add( combo = new Choice() ); combo.addItem("Item 1"); combo.addItem("Item 2");
  combo.addItem("Item 3");
  setSize(300,300);
  setVisible(true); }
 class EscuchaBoton implements ActionListener
 {public void actionPerformed(ActionEvent e)
  {try
   { Thread.sleep(10000);
    System.out.println("Termino la espera...!");
   }
  catch(Exception ex)
  {ex.printStackTrace();
   throw new RuntimeException(ex); }
  }
 }
 public static void main(String[] args)
 { new VentanaDemora(); }
}
```

Ejemplo GUI en Thread

```
□ public class VentanademoraMulti extends Frame {  
    private Button boton;  
    private Choice combo;  
    public VentanademoraMulti() {  
        setLayout(new FlowLayout());  
        add(boton = new Button("Esto va a demorar..."));  
        boton.addActionListener(new EscuchaBoton());  
        add(combo = new Choice());  
        combo.addItem("Item 1");  
        combo.addItem("Item 2");  
        combo.addItem("Item 3");  
        setSize(300, 300);  
        setVisible(true);  
    }  
  
class EscuchaBoton implements ActionListener {  
    public void actionPerformed(ActionEvent e) // inhabilito el boton mientras dure el proceso  
    {boton.setEnabled(false);  
        TareaBoton t = new TareaBoton(); // instancio y lanzo el thread que lleva a cabo la tarea  
        t.start();  
    }  
}
```

Ejemplo GUI en Thread

```
□ class TareaBoton extends Thread {  
    public void run() {  
        try { // hago aqui lo que antes hacia en el actionPerformed  
            Thread.sleep(10000);  
            System.out.println("Termino la espera...!");  
            // cuando finalizo la tarea vuelvo a habilitar el // boton  
            boton.setEnabled(true);  
        } catch (Exception ex) {  
            ex.printStackTrace();  
            throw new RuntimeException(ex);  
        }  
    }  
}  
  
public static void main(String[] args) {  
    new VentanademoraMulti();  
}  
}
```