

# Programación 2

Lenguaje Java - *Interfaces gráficas de usuario (GUI)*

Rodrigo Olivares

Mg. en Ingeniería Informática

`rodrigo.olivares@uv.cl`

6 de junio de 2017

# Contenido

- 1 Swing
  - Introducción
- 2 Características

# Swing

## Introducción

### Introducción

Hasta ahora hemos desarrollado programas que usan la consola para interactuar con el usuario. Esa forma de interfaz de usuario es muy simple y nos ha permitido centrarnos en todo aquello que tiene que ver tan sólo con la programación orientada a objetos con el lenguaje Java, sin tener que tratar al mismo tiempo el manejo de ventanas, botones y otros elementos similares.

# Swing

## Antes de comenzar

### Definición

La **interfaz gráfica de usuario** es un programa que actúa de intermediario entre el usuario y otro programa, utilizando un conjunto de componentes gráficos para representar la información y las acciones disponibles. Su principal uso, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una computadora.

# Swing

## Antes de comenzar

### Introducción

Normalmente las acciones se realizan mediante manipulación directa, para facilitar la interacción del usuario con la computadora. Surge como evolución de las interfaces de línea de comandos que se usaban para operar los primeros sistemas operativos y es pieza fundamental en un entorno gráfico. Como ejemplos de interfaz gráfica de usuario, cabe citar los entornos de escritorio Windows, el X-Window de GNU/Linux o el de Mac OS X.

# Swing

## Antes de comenzar

### Introducción

Las aplicaciones son conducidas por **eventos** y se desarrollan haciendo uso de las clases que para ello nos ofrece la API de Java. Esta API nos proporciona una biblioteca de clases para la manipulación de interfaces gráficas de usuario (en realidad son dos; **AWT** y **Swing**).

La biblioteca proporciona un conjunto de herramientas para la construcción de interfaces gráficas que tienen una apariencia y se comportan de forma semejante en todas las plataformas en las que se ejecuten.

# Swing

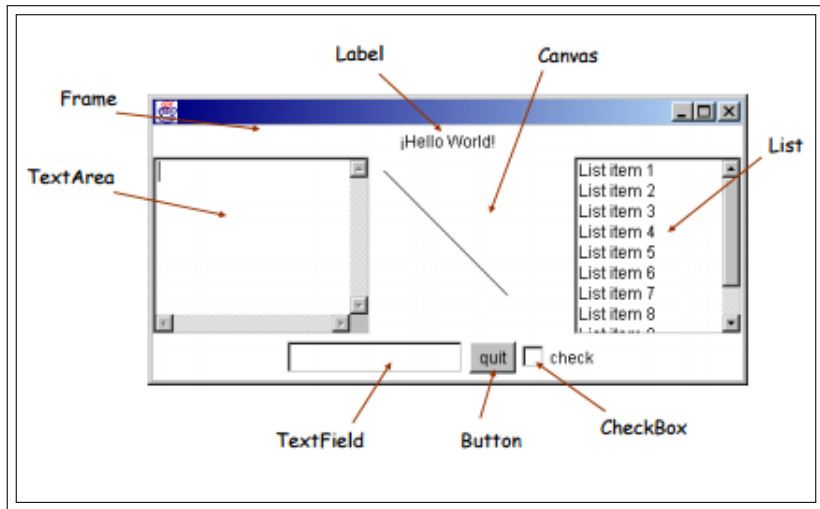
## Antes de comenzar

### Introducción

La estructura básica de la biblioteca gira en torno a **componentes** y **contenedores**. Los contenedores contienen componentes y son componentes a su vez, de forma que los eventos pueden gestionarse tanto en contenedores como en componentes.

# Swing

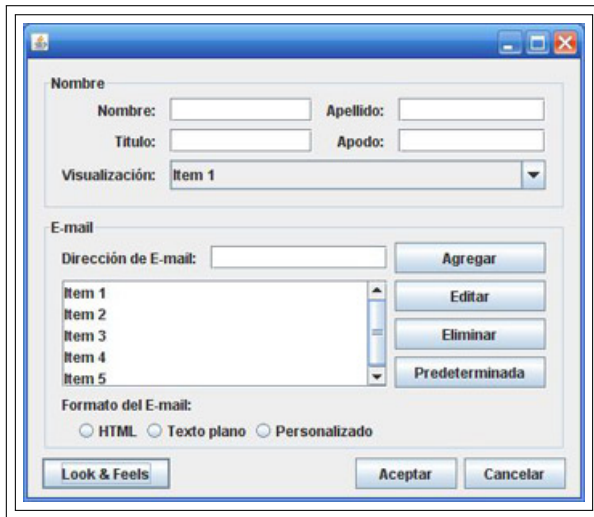
Antes de comenzar





# Swing

Antes de comenzar



# Swing

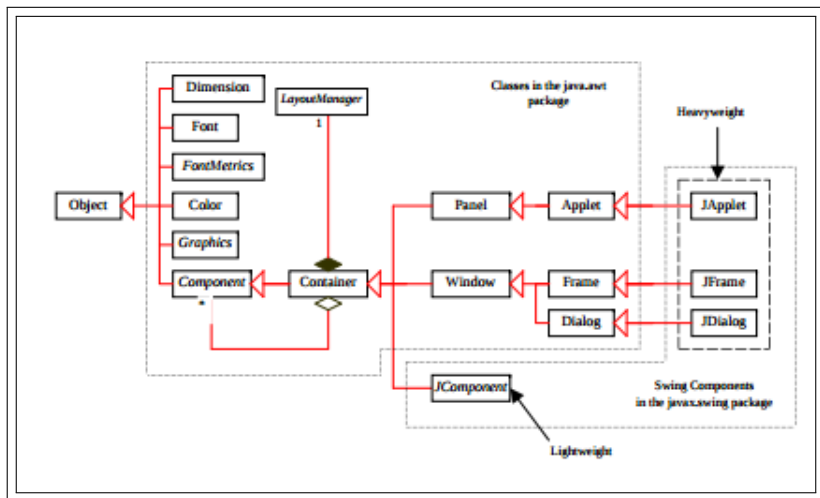
## Características

### Algunas características de Swing.

- Paquete de Java para la generación del GUI en aplicaciones reales.
- Disponible como paquete externo en Java 1.1 e integrado desde Java 1.2.
- Es una de las API de JFC (Java Foundation Classes): AWT, Java 2D, Accessibility, Drag and Drop, Swing, etc.
- Escrito totalmente en Java.
- No reemplaza a AWT. (Se apoya sobre AWT y añade JComponents).
- Utiliza el modelo de eventos de Java 1.1.
- Elección entre diferentes aspectos (look & feel).
- Arquitectura Model-View-Controller (MVC).
- Nuevos componentes (árboles, tablas, frames internos, íconos, bordes, tooltips, etc).

# Swing

## Características - Jerarquía de clases



# Swing

## Características - Jerarquía de clases

- **Component**: superclase de todas las clases de interfaz gráfica.
- **Container**: para agrupar componentes.
- **JComponent**: superclase de todos los componentes de **Swing**. Sus subclases son los elementos básicos de la GUI.
- **JFrame**: ventana que no está contenida en otras ventanas.
- **JDialog**: cuadro de diálogo.
- **JApplet**: subclase de Applet para crear applets tipo **Swing**.
- **JPanel**: contenedor invisible que mantiene componentes de interfaz y que se puede anidar, colocándose en otros paneles o en ventanas.
- **Graphics**: clase abstracta que proporciona contextos gráficos donde dibujar cadenas de texto, líneas y otras formas sencillas.
- **Color**: color de los componentes gráficos.
- **Font**: aspecto de los caracteres.
- **FontMetrics**: clase abstracta para propiedades de las fuentes.

# Swing

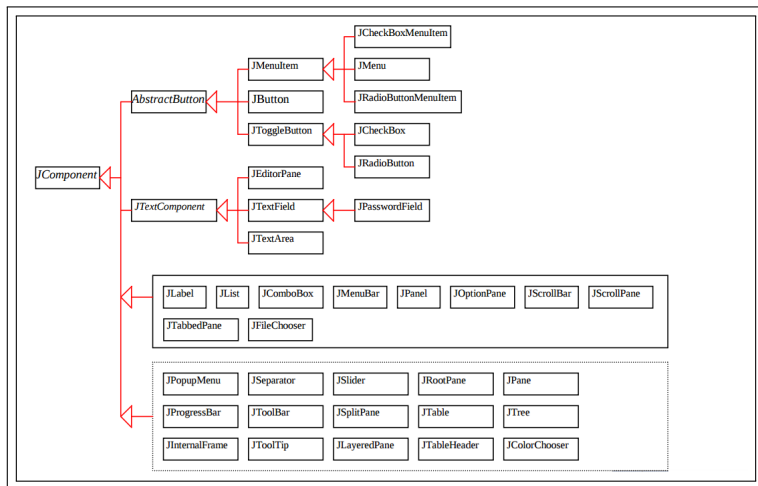
## Características - Jerarquía de clases

### Categorías de clases

- ✓ Contenedores:
  - **JFrame**, JApplet, JWindow, JDialog.
- ✓ Componentes intermedios:
  - **JPanel**, JScrollPane.
- ✓ Componentes:
  - JLabel, JButton, JTextField, JTextArea, etc.
- ✓ Clases de soporte:
  - Graphics, Color, Font, etc.

# Swing

## Características - Jerarquía de clases



# Swing

## Características - Jerarquía de clases

Contenedores de alto nivel:

- ✓ **JFrame**

- Habitualmente la clase JFrame se emplea para crear la ventana principal de una aplicación en Swing.

- ✓ **JDialog**

- Ventanas de interacción con el usuario.

Contenedores intermedios:

- ✓ **JPanel**

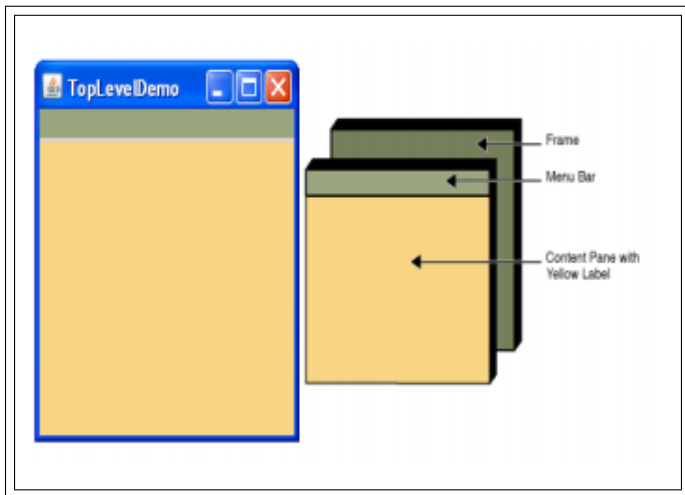
- Agrupa a otros componentes.

- ✓ **JScrollPane**

- Incluye barras de desplazamiento.

# Swing

## Características - Jerarquía de clases





# Swing

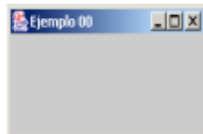
## Características - Esquema de aplicación en Swing

```
import javax.swing.*;
public class Gui00 {
    // Constantes y componentes (objetos)

    public Gui00(){
        JFrame frame = new JFrame("Ejemplo 00");

        // Configurar componentes
        // y añadirlos al panel del frame
        ...
        frame.pack();
        frame.setVisible(true);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String args[]){
        Gui00 aplicacion = new Gui00();
    }
}
```



GUI00.java

# Swing

## Características - Esquema de aplicación en Swing

### Modo 1:

- ✓ 1. Obtenemos el panel de contenido del frame:  
→ **Container** panel = **this.getContentPane();**
- ✓ 2. Añadimos componentes a dicho panel:  
→ panel.add(**unComponente**);

### Modo 2:

- ✓ A partir de 1.5 también se puede hacer directamente sobre el JFrame:  
→ add(**unComponente**);

# Swing

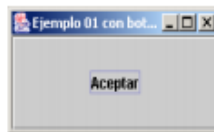
## Características - Esquema de aplicación en Swing

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

public class Gui01 extends JFrame {
    private Container panel;
    private JButton miboton;

    public Gui01() {
        super("Ejemplo 01 con botón");
        // Configurar componentes ;
        miboton = new JButton("Aceptar");
        panel = getContentPane();
        panel.add(miboton);
        . . .
        setSize(200,100);
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public static void main(String args[]) {
        Gui01 aplicacion = new Gui01();
    }
}
```



Gui01.java

# Swing

## Características - Esquema de aplicación en Swing

### Resolver combinaciones.

- ✓ 1. Con herencia de JFrame utilizando Container.
- ? 2. Con herencia de JFrame sin Container.
- ? 3. Sin herencia de JFrame con Container.
- ? 4. Sin herencia de JFrame sin Container.

# Swing

## Características - Administradores de disposición

- Los componentes se agregan al contenedor con el método **add()**.  
**JButton** unBoton = new **JButton**("Texto del botón");  
panel.add(unBoton);
- El efecto de **add()** depende del esquema de colocación o disposición (layout) del contenedor que se use.
- Existen diversos esquemas de disposición: **FlowLayout**, **BorderLayout**, **GridLayout**, etc.
- Los objetos contenedores se apoyan en objetos **LayoutManager** (administradores de disposición).
- Clases más usadas que implementa la interfaz **LayoutManager**:
  - ✓ **FlowLayout**: un componente tras otro de izquierda a derecha.
  - ✓ **BorderLayout**: 5 regiones en el contenedor (North, South, Center, West & Est).
  - ✓ **GridLayout**: contenedor en filas y columnas (grilla).

# Swing

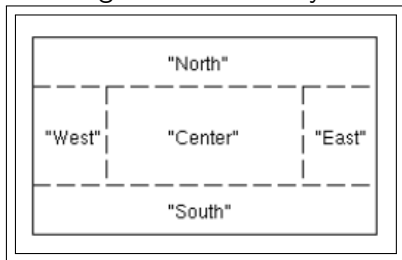
## Características - Administradores de disposición

- Para organizar el contenedor se utiliza el método **setLayout()**:  
**public void setLayout(LayoutManager mgr)**
- **setLayout(new BorderLayout());**
- **setLayout(new FlowLayout());**
- **setLayout(new GridLayout(3,4));**
- ✓ El layout manager elige la mejor posición y tamaño de cada componente de acuerdo al espacio disponible.

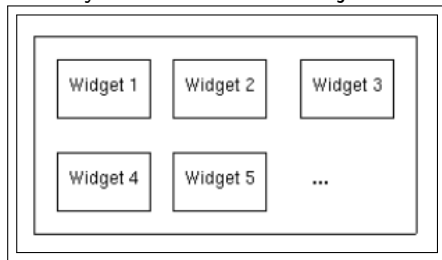
# Swing

## Características - Administradores de disposición

**BorderLayout** organiza el contenedor en 5 zonas. Cada zona se organiza en FlowLayout:



**FlowLayout** organiza los componentes de izquierda a derecha y de arriba hacia abajo:



# Swing

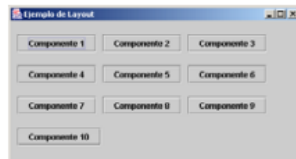
## Características - Administradores de disposición - Flow

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

public class Gui02 extends JFrame {

    public Gui02() {
        super("Ejemplo de Layout");
        // Configurar componentes ;
        // Configurar layout ;
        setLayout(new FlowLayout(FlowLayout.LEFT, 10, 20));
        for(int i = 1; i <= 10; i++)
            add(new JButton("Componente " + i));
        setSize(200,200);//pack();
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public static void main(String args[]) {
        Gui02 aplicacion = new Gui02();
    }
}
```



Cambiando el tamaño  
se redistribuyen los componentes

Gui02.java



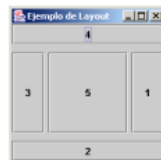
# Swing

## Características - Administradores de disposición - Border

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

public class Gui03 extends JFrame {

    public Gui03() {
        super("Ejemplo de Layout");
        // BorderLayout
        setLayout(new BorderLayout(5, 10));
        add(new JButton("1"), BorderLayout.EAST);
        add(new JButton("2"), BorderLayout.SOUTH);
        add(new JButton("3"), BorderLayout.WEST);
        add(new JButton("4"), BorderLayout.NORTH);
        add(new JButton("5"), BorderLayout.CENTER);
        setSize(200,200); //pack();
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
    ...
}
```

**Gui03.java**

# Swing

## Características - Administradores de disposición - Grid

- Crea una zona de **filas** × **columnas** componentes y éstos se van acomodando de izquierda a derecha y de arriba a abajo.
- **GridLayout** tiene otro constructor que permite establecer la separación (en pixels) ente los componentes, que es cero con el primer constructor.

Así, por ejemplo:

```
new GridLayout(3, 4, 2, 2);
```

- crea una organización de 3 filas y 4 columnas donde los componentes quedan a dos pixels de separación.

# Swing

## Características - Administradores de disposición - Grid

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

public class Gui03b extends JFrame {

    public Gui03b() {
        super("Ejemplo de Layout");

        setLayout(new GridLayout(4, 3, 5, 5));
        for(int i = 1; i <= 10; i++)
            add(new JButton(Integer.toString(i)));
        setSize(200,200);
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
    ...
}
```

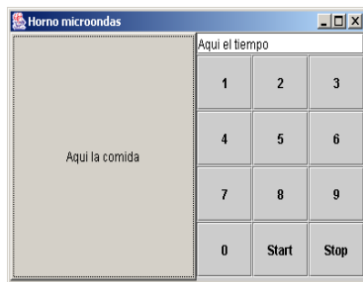
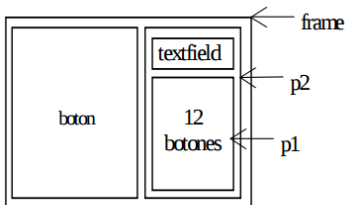


Gui03b.java

# Swing

## Características - Paneles como contenedores

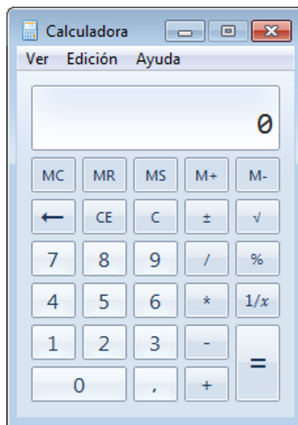
Los paneles actúan como pequeños contenedores para agrupar componentes. Colocamos los componentes en paneles y los paneles en el frame o incluso en otros paneles.



# Swing

## Características - Ejercicio

Desarrolle **sólo** la interfaz gráfica de usuario de una calculadora básica.



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

public class Gui04 extends JFrame {

    public Gui04() {
        setTitle("Horno microondas");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        JPanel p1 = new JPanel();
        p1.setLayout(new GridLayout(4, 3));
        for (int i = 1; i <= 9; i++) {
            p1.add(new JButton("" + i));
        }
        p1.add(new JButton("" + 0));
        p1.add(new JButton("Start"));
        p1.add(new JButton("Stop"));

        JPanel p2 = new JPanel();
        p2.setLayout(new BorderLayout());
        p2.add(new JTextField("Aquí el tiempo"),
            BorderLayout.NORTH);
        p2.add(p1, BorderLayout.CENTER);
        add(p2, BorderLayout.EAST);
        add(new JButton("Aquí la comida"),
            BorderLayout.CENTER);
        setSize(400, 250);
        setVisible(true);
    }

    public static void main(String[] args) {
        Gui04 frame = new Gui04();
    }
}
```

# Swing

## Características - Intereacción con el usuario

Al interactuar con la aplicación, el usuario:

→ Acciona componentes (**ActionEvent**).

El usuario pulsa un botón.

El usuario termina de introducir un texto en un campo y presiona Enter.

El usuario selecciona un elemento de una lista pulsando el preferido (o de un menú).

Pulsa o suelta botones del ratón (**MouseEvent**).

→ Minimiza, cierra o manipula una ventana (**WindowEvent**).

→ Escribe con el teclado (**KeyEvent**).

→ Descubre porciones de ventanas (**PaintEvent**).

# Swing

## Características - Intereacción con el usuario

Al interactuar con la aplicación, el usuario:

Cuando el usuario de un programa mueve el ratón, cambia el foco o pulsa una tecla, genera un evento (`actionEvent`).

Los eventos son objetos de ciertas clases. Normalmente un objeto de alguna subclase de **EventObject** que indica:

- ✓ El elemento que accionó el usuario.
- ✓ La identificación del evento que indica su naturaleza.
- ✓ La posición del ratón en el momento de la interacción.
- ✓ Teclas adicionales pulsadas por el usuario, como la tecla *Control*, la tecla de cambio a mayúsculas (*shift*), etc.



# Swing

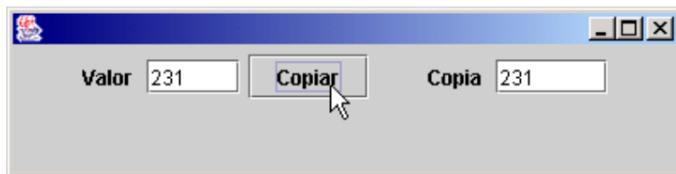
## Características - Intereacción con el usuario - Acciones

Acción	Objeto origen	Tipo de evento
Pulsar un botón	JButton	ActionEvent
Cambio del texto	JTextComponent	TextEvent
Pulsar Enter/Intro en un campo de texto	JTextField	ActionEvent
Selección de un nuevo elemento	JComboBox	ItemEvent ActionEvent
Selección de elemento(s)	JList	ListSelectionEvent
Pulsar una casilla de verificación	JCheckBox	ItemEvent ActionEvent
Pulsar un botón de radio	JRadioButton	ItemEvent ActionEvent
Selección de una opción de menú	JMenuItem	ActionEvent
Mover la barra de desplazamiento	JScrollBar	AdjustmentEvent
Abrir, minimizar, maximizar o cerrar la ventana	JWindow	WindowEvent

# Swing

## Características - Interacción con el usuario - Ejemplo

Al pulsar el botón Copiar, se debe copiar el valor del cuadro "Valor" en el cuadro "Copia".



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

public class Gui05 extends JFrame {

    JButton botonCopiar;
    JTextField campoValor, resultado;

    public Gui05() {
        setLayout(new FlowLayout());
        add(new JLabel("Valor "));
        campoValor = new JTextField(5);
        add(campoValor);
        botonCopiar = new JButton("Copiar");
        add(botonCopiar);
        botonCopiar.addActionListener(new OyenteBoton());
        add(new JLabel(" Copia "));
        resultado = new JTextField(6);
        setSize(400, 100);
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public static void main(String args[]) {
        Gui05 ventana = new Gui05();
    }

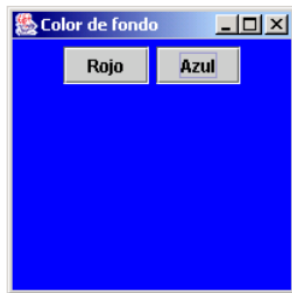
    class OyenteBoton implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {
            String valor = campoValor.getText();
            resultado.setText(valor);
        }
    }
}
```

# Swing

## Características - Intereacción con el usuario - Ejemplo

Cambiar el color del panel mediante eventos de botón.



```

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

public class Gui06 extends JFrame {

    JButton rojo = new JButton("Rojo");
    JButton azul = new JButton("Azul");
    Container p;

    public Gui06() {
        super("Color de fondo");
        p = this.getContentPane();
        setLayout(new FlowLayout());
        add(rojo);
        add(azul);
        rojo.addActionListener(new OyenteRojo());
        azul.addActionListener(new OyenteAzul());
        setSize(200, 200);
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public static void main(String args[]) {
        Gui06 ventana = new Gui06();
    }

    class OyenteRojo implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent evento) {
            p.setBackground(Color.red);
        }
    }

    class OyenteAzul implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent evento) {
            p.setBackground(Color.green);
        }
    }
}

```

# Swing

## Componentes

- ✓ **JButton**: Botón aislado. Puede pulsarse, pero su estado no cambia.
- ✓ **JToggleButton**: Botón seleccionable. Cuando se pulsa el botón, su estado pasa a seleccionado, hasta que se pulsa de nuevo (entonces se deselecciona)  
`isSelected()` permite chequear su estado.
- ✓ **JCheckBox**: Especialización de *JToggleButton* que implementa una casilla de verificación. Botón con estado interno, que cambia de apariencia de forma adecuada según si está o no está seleccionado
- ✓ **JRadioButton**: Especialización de *JToggleButton* que tiene sentido dentro de un mismo grupo de botones (`ButtonGroup`) que controla que sólo uno de ellos está seleccionado.

# Swing

## Componentes - JButton

- ✓ Constructores:

  - JButton**(String text)

  - JButton**(Icon icon)

  - JButton**(String text, Icon icon)

- ✓ Respuesta a botones:

  - Implementar la interfaz **ActionListener**.

  - Implementar el método **actionPerformed**(**ActionEvent** e).

# Swing

## Componentes - JButton

```
boton1 = new JButton("A Euros ");  
boton1.setIcon(new ImageIcon("flag1.gif"));
```



```
boton2 = new JButton(new ImageIcon("flag2.gif"));
```



```
boton3 = new JButton("Botón", new ImageIcon("flag8.gif"));
```





# Swing

## Componentes - JLabel

- ✓ Para texto, una imagen o ambos:

**JLabel**(String text, int horizontalAlignment)

**JLabel**(String text)

**JLabel**(Icon icon)

**JLabel**(Icon icon, int horizontalAlignment)

# Swing

## Componentes - JLabel

```
eti1 = new JLabel("Etiqueta de texto...");  
eti2 = new JLabel(new ImageIcon("flag8.gif"));
```



# Swing

## Componentes - JLabel

- ✓ Campos de texto para introducir caracteres:

**TextField**(int columns)

**TextField**(String text)

**TextField**(String text, int columns)

**TextField** text1 = new TextField("hola", 10);

- Fijar texto: text1.setText(" Adios" );
- Obtener texto: String str = text1.getText();
- Agregar al Panel: p1.add(text1);

# Swing

## Componentes - JComboBox

- ✓ Listas de elementos para seleccionar un único valor:

Creación: **JComboBox** ch1 = **new JComboBox()**;

Agregar opciones: ch1.**addItem**(Object elemento);

Registrar Evento: ch1.**addItemListener**(objeto\_oyente);

Obtener selección: **String** val = (**String**) ch1.**getSelectedItem()**;

→ Implementar la interfaz **ItemListener**.

→ Implementar el método **itemStateChanged**(ItemEvent e)

```
class OyenteItem implements ItemListener {  
    public void itemStateChanged(ItemEvent e) {  
  
        ...  
    }  
}
```

# Swing

## Componentes - JList

- ✓ Implementar la interfaz **ListSelectionListener**
- ✓ Implementar el método **valueChanged(ListSelectionEvent e)**

```
l.addListSelectionListener(new OyenteLista());
```

```
...
```

```
class OyenteLista implements ListSelectionListener {  
    public void valueChanged(ListSelectionEvent e) {  
        int[] indices = l.getSelectedIndices();  
        int i;  
        for(int i = 0; i < indices.length; i++) {  
            ...  
        }  
    }  
}
```

# Swing

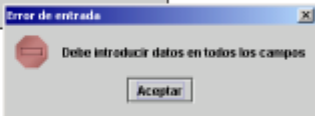
## Componentes - Interacción modal

```
JOptionPane.showMessageDialog(  
    this, // la ventana padre  
    "Texto para el mensaje", // mensaje  
    "Título", // título de la ventana de diálogo  
    JOptionPane.ERROR_MESSAGE);  
// JOptionPane.PLAIN_MESSAGE  
// JOptionPane.INFORMATION_MESSAGE  
// JOptionPane.WARNING_MESSAGE
```

```
public class Gui18 extends JFrame
```

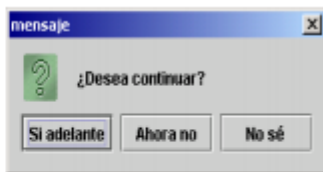


```
    public Gui18() {  
        super("Título de la ventana");  
        setLayout(new FlowLayout());  
        setSize(200, 100); // pack();  
        setVisible(true); // show();  
        setDefaultCloseOperation(EXIT_ON_CLOSE);  
        // if ocurre algo  
        JOptionPane.showMessageDialog(null,  
            "Debe introducir datos en todos los campos",  
            "Error de entrada ",  
            JOptionPane.ERROR_MESSAGE);  
    }  
  
    public static void main(String[] args) {  
        Gui18 f = new Gui18();  
    }  
}
```



Gui18.java

```
public class Gui19 extends JFrame {  
    public Gui19() {  
        super("Título de la ventana");  
        p = getContentPane();  
        setLayout(new FlowLayout());  
        setSize(200, 100);  
        setVisible(true);  
        setDefaultCloseOperation(EXIT_ON_CLOSE);  
        Object[] textoOpciones={"Si adelante","Ahora no","No  
sé"};  
        int opcion = JOptionPane.showOptionDialog(null,  
            "¿Desea continuar?", "mensaje",  
            JOptionPane.YES_NO_CANCEL_OPTION,  
            JOptionPane.QUESTION_MESSAGE, null, textoOpciones,  
            textoOpciones[0]);  
    }  
    ...  
}
```



Gui19.java

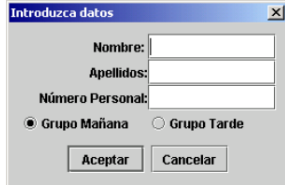


```
class PanelDatos extends JPanel {
    public PanelDatos() {
        setLayout(new GridLayout(4, 2));
        JLabel etiquetaNombre = new JLabel("Nombre: ", JLabel.RIGHT);
        JTextField campoNombre = new JTextField();
        add(etiquetaNombre);
        add(campoNombre);
        JLabel etiquetaApellidos = new JLabel("Apellidos:", JLabel.RIGHT);
        JTextField campoApellidos = new JTextField();
        add(etiquetaApellidos);
        add(campoApellidos);
        JLabel etiquetaNP = new JLabel("Número Personal:", JLabel.RIGHT);
        JTextField campoNP = new JTextField();
        add(etiquetaNP);
        add(campoNP);
        ButtonGroup grupoBotones = new ButtonGroup();
        JRadioButton mañana = new JRadioButton("Grupo Mañana", true);
        JRadioButton tarde = new JRadioButton("Grupo Tarde");
        grupoBotones.add(mañana);
        grupoBotones.add(tarde);
        add(mañana);
        add(tarde);
    }
}
```

```
public class Gui20 extends JFrame {
    public Gui20() {
        super("Título de la ventana");
        setLayout(new FlowLayout());

// Cuando necesitamos el cuadro de diálogo...
        PanelDatos pd = new PanelDatos();
        if(JOptionPane.showConfirmDialog(this, pd,
            "Introduzca datos",
            JOptionPane.OK_CANCEL_OPTION,
            JOptionPane.PLAIN_MESSAGE)
            == JOptionPane.OK_OPTION) {
            // ... tratamiento
        }
    }

    public static void main(String[] args) {
        Gui20 f = new Gui20();
    }
}
```



A screenshot of a Java dialog box titled "Introduzca datos". It contains three text input fields labeled "Nombre:", "Apellidos:", and "Número Personal:". Below these fields are two radio buttons: "Grupo Mañana" (which is selected) and "Grupo Tarde". At the bottom of the dialog are two buttons: "Aceptar" and "Cancelar".

Gui20.java

# Preguntas

Preguntas ?

# Tarea

## Transformación Números Romanos

El sistema de numeración romana se desarrolló en la antigua Roma y se utilizó en todo su imperio. Es un sistema de numeración no posicional, en el que se usan algunas letras mayúsculas como símbolos para representar los números. La siguiente tabla muestra los símbolos válidos en el sistema de numeración romano, y sus equivalencias en el sistema decimal:

Romano	Decimal	Nota
I	1	Unus
V	5	Quinque
X	10	Decem
L	50	Quinquaginta
C	100	Centum
D	500	Quingenti
M	1000	Mille

# Tarea

## Transformación Números Romanos

Desarrolle un programa con interfaz de usuario que convierta un entero positivo (ingresado en un campo de texto), en un número romano. Las reglas para construir un número romano son las siguientes:

- Los símbolos con un valor grande usualmente aparecen antes que los con menor valor.
- El valor de un número romano es, en general, la suma de los valores de los símbolos. Por ejemplo, II es 2, VIII es 8.
- Si un símbolo de menor valor aparece antes de uno de mayor valor, el valor de los dos símbolos es la diferencia entre ambos. Por ejemplo, IV es 4, IX es 9, y LIX es 59.
- Note que no hay cuatro símbolos consecutivos iguales. Por ejemplo, IV, pero no IIII, es el número 4.
- Los números romanos contruidos de esta forma pueden no ser únicos. Por ejemplo, MCMXC y MXM son válidos para 1990. Aunque el número romano generado por su programa no debe necesariamente ser el más corto, **nunca** use VV para 10, VVV para 15, LL para 100, DD para 1000, etc.