UD1

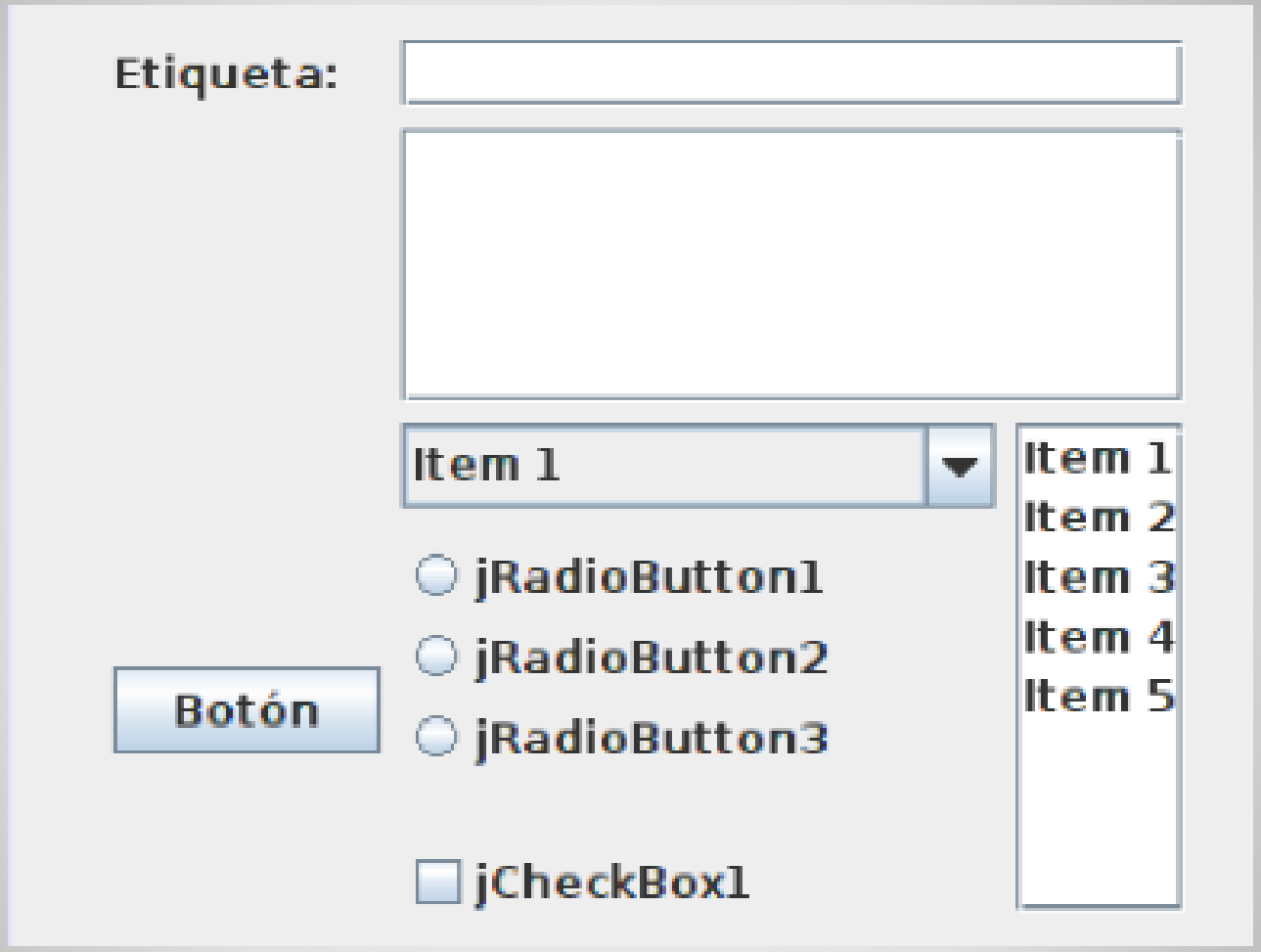
Confección de Interfaces de Usuario

¿Qué es una IU?

- **Interfaz de Usuario**: "Elemento de una aplicación informática que permite al usuario **comunicarse** con ella"
- Tipos:
  - Textuales - órdenes
  - **Gráficas** – ratón y teclado / pantalla táctil

Bibliotecas de componentes

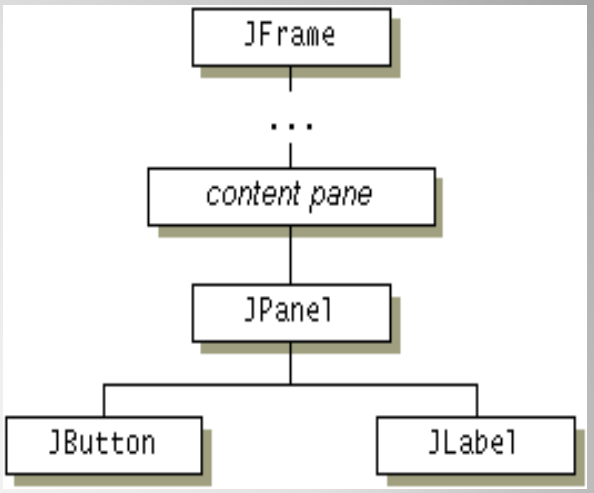
- Componentes de las **GUI** ya están implementados en clases - Agrupados en **bibliotecas**
- Opciones en **Java** (incluidas en JDK):
  - **AWT** (`java.awt.*`)**:** Primera biblioteca de de Java
    - Los componentes funcionan diferente según el SO- En contra de la filosofía de Java
  - **Swing** (`java.swing.*`)**:** Sucesor de AWT
    - Multiplataforma, personalizable
    - Más componentes y look&feels que AWT
    - **Estándar actual**
  - **JavaFX**: Posible sucesor de Swing
    - Fase inicial
    - Facilita el uso de componentes multimedia (animaciones, HTML…)
  - **Biblioteca de componentes propia**
- **Otras opciones:** Microsoft, XML, etc.


Etiqueta:
Item 1
Item 1
Item 2
Item 3
Item 4
Item 5
jRadioButton1
jRadioButton2
jRadioButton3
Botón
jCheckBox1

Ejemplo

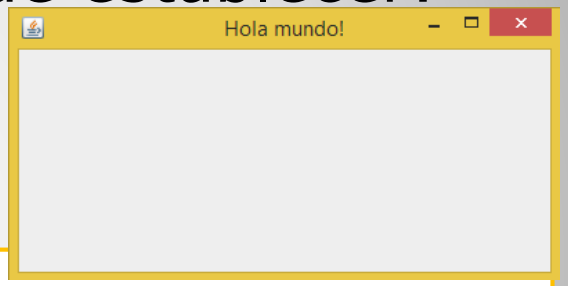
Contenedores

- Sirven para albergar componentes
  - Estructura jerárquica
    - Clase padre - **JComponent**
- Tipos:
  - Marcos – **JFrame**
  - Paneles – **JPanel**
    - De desplazamiento – **JScrollPane**
    - Pestañas -**JTabbedBar**
  - Diálogos
    - Predefinidos - **JOptionPane**
    - Personalizados – **JDialog**
    - Estándar – **JFileChooser, JColorChooser**
  - Contenedores secundarios
    - Barra de menú – **JMenuBar**
    - Barra de herramientas – **JToolBar**



Creación de ventanas: JFrame

- Constructores
  - `JFrame()`
  - `JFrame(titulo)`
- Una vez creado el objeto hay que establecer:
  - Su tamaño
  - Su acción de cierre
  - Hacerla visible




```
import javax.swing.*;
public class HolaSwingFrame extends JFrame {
    public HolaSwingFrame() {
        super("Hola mundo!");
        setSize(400, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
public static void main(String[] arguments) {
        HolaSwingFrame frSaluda = new HolaSwingFrame();
    }
}
```

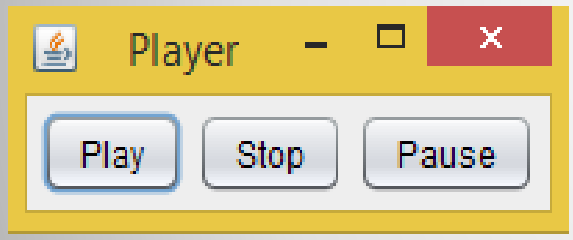
Componentes básicos

- Etiquetas - **JLabel**: Situar texto en la interfaz
- Campos de texto **- JTextField**: Introducir datos en una línea
  - Con formato
  - Password
- Áreas de texto **- JTextArea**: Datos en varias líneas
  - Asociado a JScrollPane
- Botones - **JButton**: Realizar acciones al pulsarlos
- Botones de radio - **JRadioButton**: Aspecto circular, se presentan agrupados (**ButtonGroup**) para realizar la selección de uno de ellos
- Cuadros de verificación - **JCheckBox**: Aspecto rectangular, permiten seleccionar más de un elemento
- Listas – **JList:** Colección de valores, muestra varios
- Cuadros combinados – **JComboBox:** Desplegable
- Otros: **JSlider**, etc.

Recursos

- Recursos en carpeta DENTRO del proyecto
  - Librerías
  - Imágenes
  - Etc.
- Cargar un recurso:
  - `getClass.getResource(ruta_del_fichero)`
    - La ruta comienza por / (lo que nos situa en la carpeta src)
- Imágenes
  - `ImageIcon icono = new ImageIcon(recurso)`
  - `Icono.getImage()` devuelve un objeto de tipo Image

- Crear el componente con **new**
- Añadirse al contentPane de la ventana o a un panel con el método **add**




```
import javax.swing.*;
public class Reproductor extends JFrame {
    public Reproductor() {

    JButton btnPlay = new JButton("Play");
    JButton btnStop = new JButton("Stop");
    JButton btnPause = new JButton("Pause");

    Container cp = this.getContentPane();
    cp.add(btnPlay);
    cp.add(btnStop);
    cp.add(btnPause);
    }
public static void main(String[] arguments) {
    Reproductor frReproductor = new Reproductor();
    }
}
```

Añadir componentes

Propiedades y Eventos

- Cada componente tiene una serie de **propiedades**
  - Ej. *Tamaño*, tipo de fuente, etc.
- Cuando el usuario interacciona con el usuario se generan **eventos**
  - Ej. *Pulsar un botón*

Programación orientada a eventos

Manejo de eventos en Swing

- Los componentes que tienen que desencadenar acciones al interactuar con ellos tienen que ser "escuchados"
- Existen distintos tipos de escuchadores: ratón, teclado, etc. – **EventListeners**
- Los escuchadores, cuando ocurre un evento, invocan al **manejador del evento** (código que se ha de ejecutar), pasándole información sobre el mismo
  - Ej. *`actionPerformed(ActionEvent e)`*
    - El objeto ActionEvent tiene toda la información del evento que ha ocurrido

- La clase **JButton** tiene un método para especificar el **escuchador** que manejará el evento:
  - *void addActionListener(ActionListener l)*
- Para ser escuchador hay que implementar una "interfaz escuchadora". Es típico que sea la ventana
  - *java.awt.event.ActionListener*

```
public interface ActionListener {
    void actionPerformed(ActionEvent e);
}
```

```
public class Manejador implements ActionListener
{
    void actionPerformed(ActionEvent e){
     …}
}
```

Información sobre el evento

- Cuando un usuario pulsa el botón, se ejecuta el método ***actionPerformed*** de todos los escuchadores registrados
- Métodos de **ActionEvent**:
  - `public Object getSource()`
  - `public String getActionCommand()`

Ejemplo: Pulsación de un botón

Eventos más usados

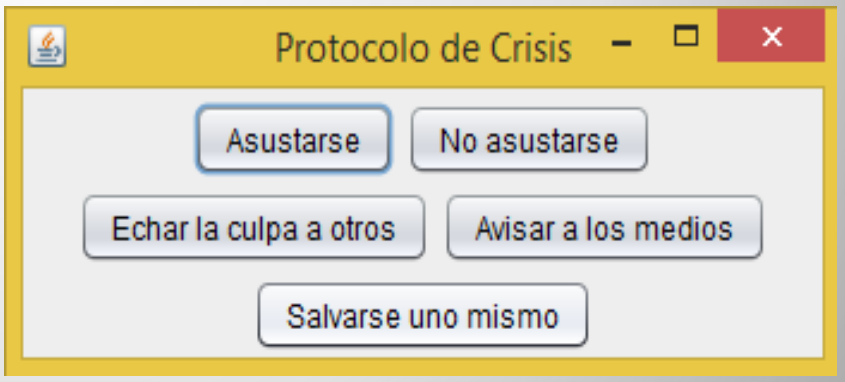
- **ActionListener:** pulsar botón, escoger elmento de menú, etc.
  - `actionPerformed(ActionEvent e)`
- **MouseListener:** Pulsaciones de botón
  - `mouseClicked(MouseEvent e)`
  - `mouseEntered(MouseEvent e)`
  - `mouseExisted(MouseEvent e)`
  - Etc.
- **KeyListener:** Pulsaciones de teclas
  - `keyPressed(KeyEvent e)`
  - `keyRealeased(KeyEvent e)`
  - Etc.
- **WindowListener**
  - `windowOpened(WindowEvent e)`
  - `windowClosed(WindowEvent e)`
  - Etc.

Administradores de diseño

- La manera de colocar componentes se denomina **Layout**
- Se puede utilizar posicionamiento absoluto (x, y) pero esto **no** es recomendable
  - *Ej. setBounds*
- Los **layout managers** se encargan de colocar los componentes en el contenedor
  - **`FlowLayout`**
  - **`GridLayout`**
  - **`BoxLayout`**
  - **`BorderLayout`**
  - **Etc.**
- Pueden aplicarse sobre la ventana o sobre un panel con el método:
  - *`void setLayout(LayoutManager lm)`*

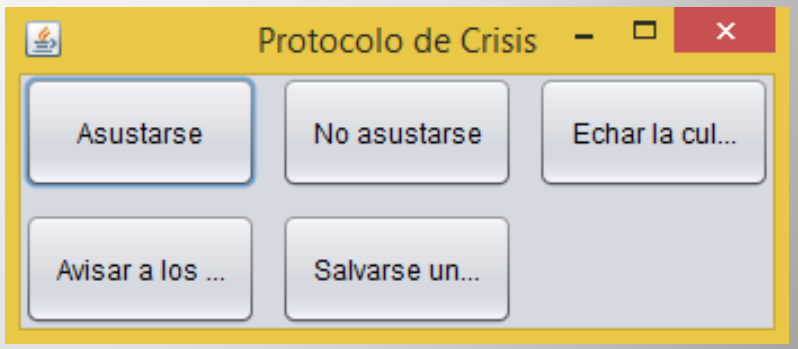
FlowLayout

- Se añaden en orden de izquierda a a derecha y de arriba abajo
- Métodos
  - `setAligment (int alineacion)`
  - `setHgap (int separacion)`
  - `setVgap (int separacion)`



GridLayout

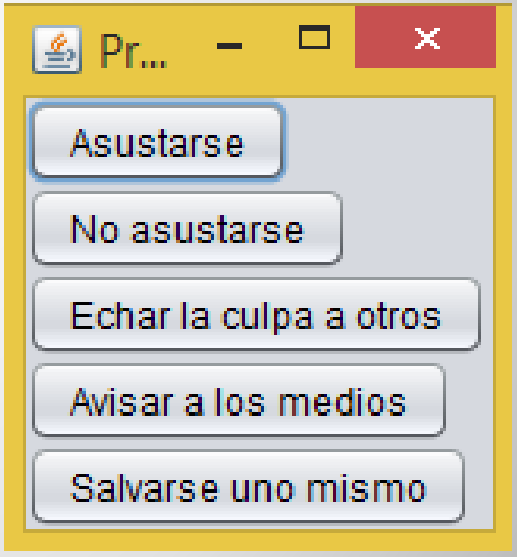
- Crea una rejilla donde coloca los componentes en el orden en el que se añaden
- Constructor
  - `GridLayout(int filas, int columnas)`
- Métodos
  - `setHgap (int separacion)`
  - `setVgap (int seperacion)`



BoxLayout

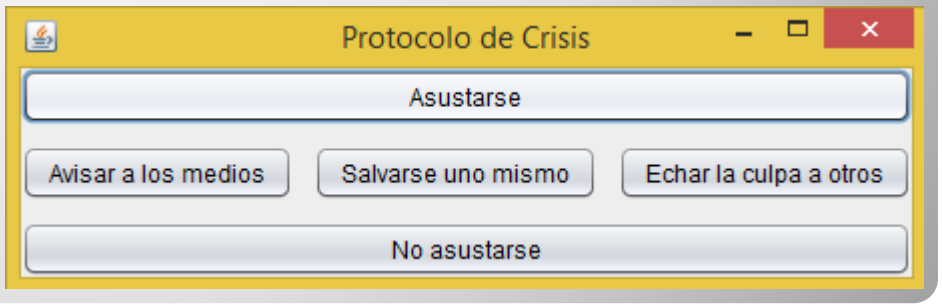
- Paquete `java.swing`
- Constructor diferente. Recibe:
  - Componente a organizar
  - Tipo de alineación: `Y_AXIS` o `X_AXIS`

```
JPanel pBotones = new JPanel();
BoxLayout admin = new
BoxLayout(pBotones,BoxLayout.Y_AXIS);
pBotones.setLayout(admin);
```



BorderLayout

- Coloca y cambia de tamaño sus componentes para que se ajusten a los bordes y parte central de la ventana
- Métodos
  - `setHgap (int separacion)`
  - `setVgap (int separacion)`
- Al añadir un elemento, hay que especificar su colocación
  - Ej. *`BorderLayout.EAST`*




Añadir usuario
Nombre:
DNI:
Fecha de nacimiento:
/
/
Aceptar
Cancelar
GridLayout
FlowLayout
FlowLayout

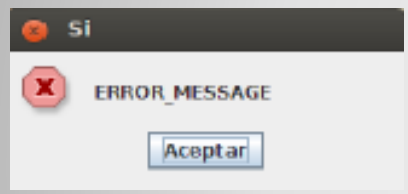
Ejemplo

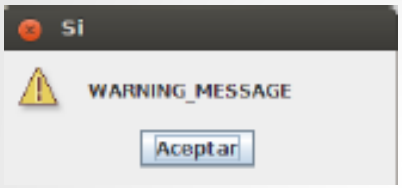
Diálogos

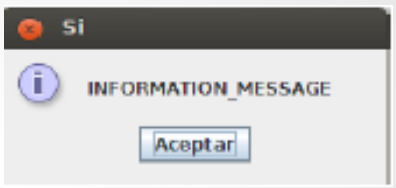
- Un diálogo es un frame que permite recolectar datos
  - clase **`JDialog`** (subclase de `JFrame`)
- Un diálogo puede ser modal o no modal
  - Si el diálog es **no modal** se abre la ventana pero el usuario puede seleccionar y trabajar con otras ventanas de la aplicación
  - Si un diálogo es **modal** cuando se activa no se puede acceder a ningún otro elemento del programa
    - En Swing si el diálogo es modal el hilo que abre el diálogo se bloquea hasta que el diálogo sea cerrado

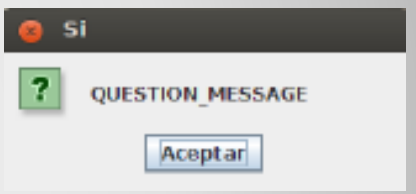
Cuadros de diálogo predefinidos

- La clase **JOptionPane** proporciona métodos para mostrar ventanas de aviso y de confirmación estándar
  - *void **showMessageDialog**(Component padre, Object mensaje, String tituloVentana, int tipoMensaje)*

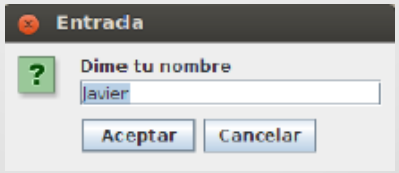


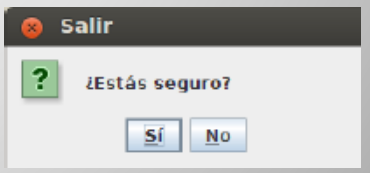






  - *String **showInputDialog**(Component padre, Object mensaje, Object valorDefecto)*
  - *int **showConfirmDialog**(Component padre, Object mensaje, String titulo, int tipoOpciones, int tipoMensaje)*





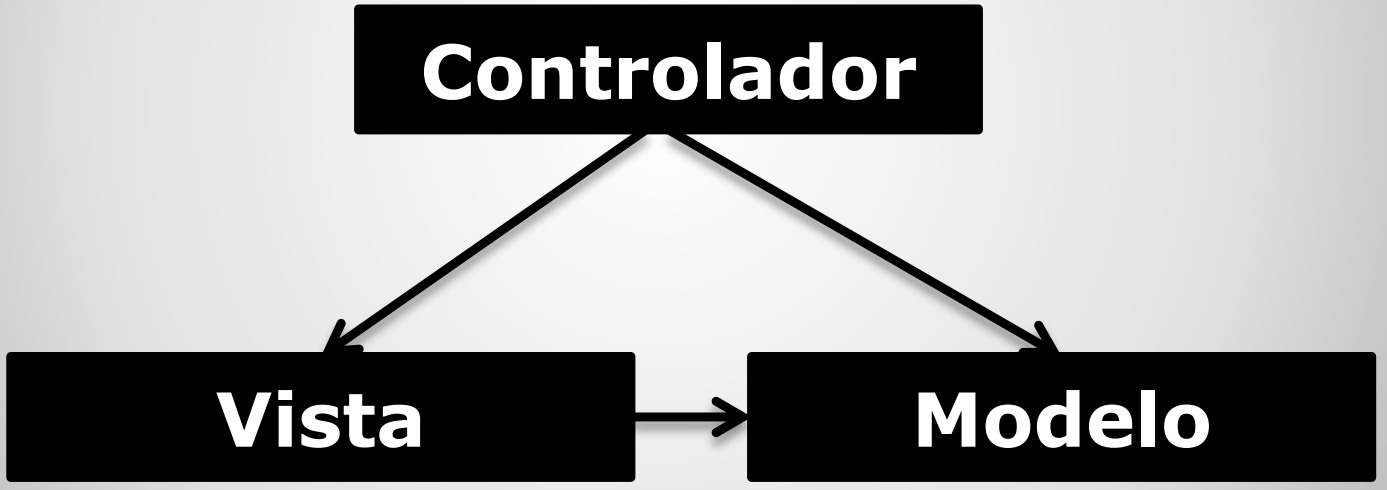
Diálogos personalizados

- Heredan de **JDialog**
- Constructor: Título y si es modal

```
public class DialogoDatos extends JDialog {
   JTextField nombre;
   ...
   public DialogoDatos(JFrame frame) {
      super(frame, “Título”, true);
      ContentPane cp = (ContentPane) getContentPane();
      ...
   }
}
```

Arquitectura MVC

- MVC = **Modelo – Vista – Controlador**
- Patrón de diseño que separa los datos de la aplicación, la interfaz de usuario y la lógica de negocio



Creación de una GUI simple

- Crear una clase que extienda de **`JFrame`**
- Declarar los componentes necesarios como atributos
- En el constructor
  - Establecer las propiedades de la ventana
  - Obtener elcontent pane y fijar Layout
  - Instanciar los componentes necesarios y establecer sus propiedades
  - Añadir los componentes al content pane
  - Añadir los listeners de los componentes y definir código del método manejador
- Método `main()` que instancie el objeto creado

Constructores de interfaces

- Permiten construir interfaces de usuario interactivamente
  - *Ejemplo:* ***Netbeans*** *(netbeans.org)*
- Pasos:
  - Añadir componentes con drag&drop
  - Modificar propiedades
  - Modificar eventos
- Ventajas:
  - Curva de aprendizaje corta
  - Facilita tareas tediosas (layout…)
- Desventajas:
  - Difícil de mantener
    - Es necesario conocer el código
  - Dependiente de la herramienta