

Java Foundation Classes (JFC)

Swing

Swing

- Jerarquía de componentes
- Componentes Swing
 - JText Area
 - JList
 - JTable
 - JDialog

JFC - Swing

La Java Foundation Classes o JFC es el framework para la construcción de interfaces de usuario gráficas portables provistas por JAVA. Se incorporó a la API Java a partir de la versión Java 1.2. Este framework está compuesto por un conjunto de tecnologías que permiten agregar funcionalidades gráficas avanzadas e interactividad a las aplicaciones JAVA.

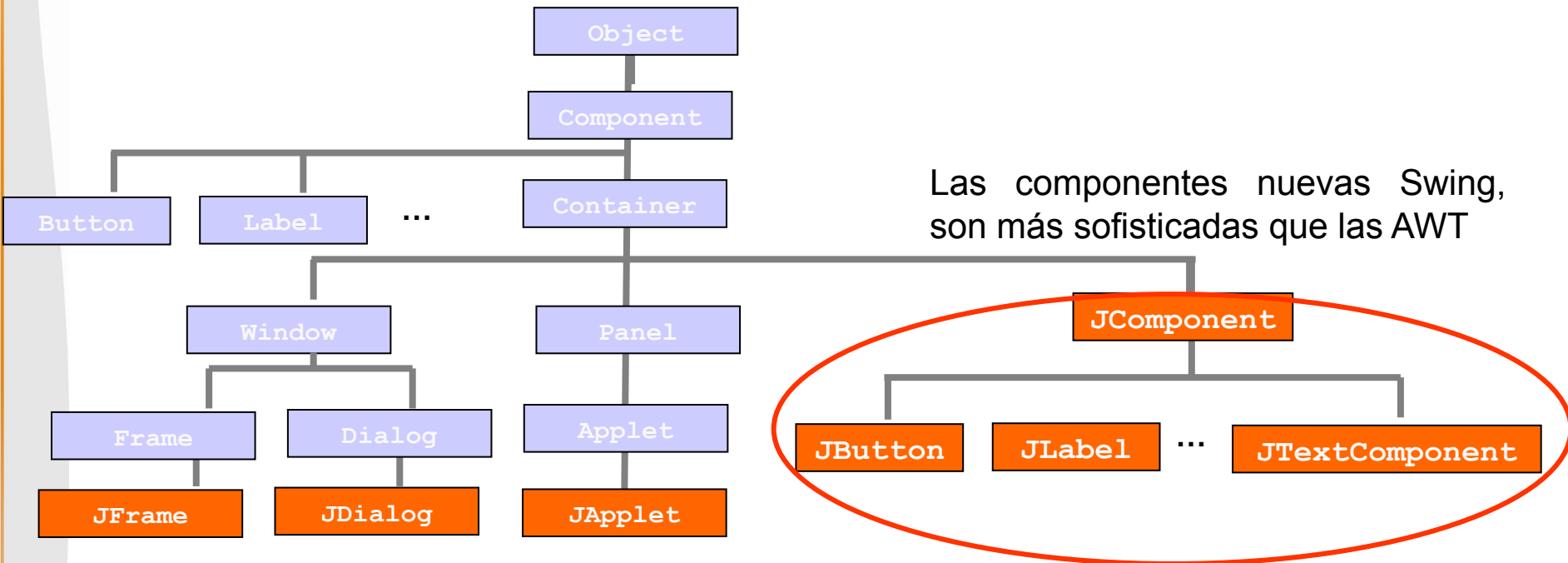
Incluye las siguientes características/capacidades:

- **Swing**: un conjunto de componentes de GUI enriquecidas, escritas en JAVA.
- **Soporte para Pluggable Look-and-Feel**: la apariencia y el comportamiento (look & feel) de las aplicaciones swing pueden ser intercambiados fácilmente. Por ejemplo el mismo programa podría lucir con un look & feel de Windows o de Mac.
- **La API para Java 2D**: es un conjunto de clases que facilitan la incorporación de gráficos, textos e imágenes 2D de alta calidad en applets y aplicaciones.
- **La API para Accesibilidad**: consiste en un conjunto de clases que permiten crear GUI para usuarios con discapacidades (paquete `javax.accessibility`).
- **Internacionalización**: permite programar aplicaciones con las que pueden interactuar usuarios de diferentes idiomas y culturas.

Swing

Jerarquía de componentes

Las componentes básicas de Swing son subclases de **Container** (las AWT son subclases de **Component**) por lo tanto están capacitadas para contener a otras componentes.

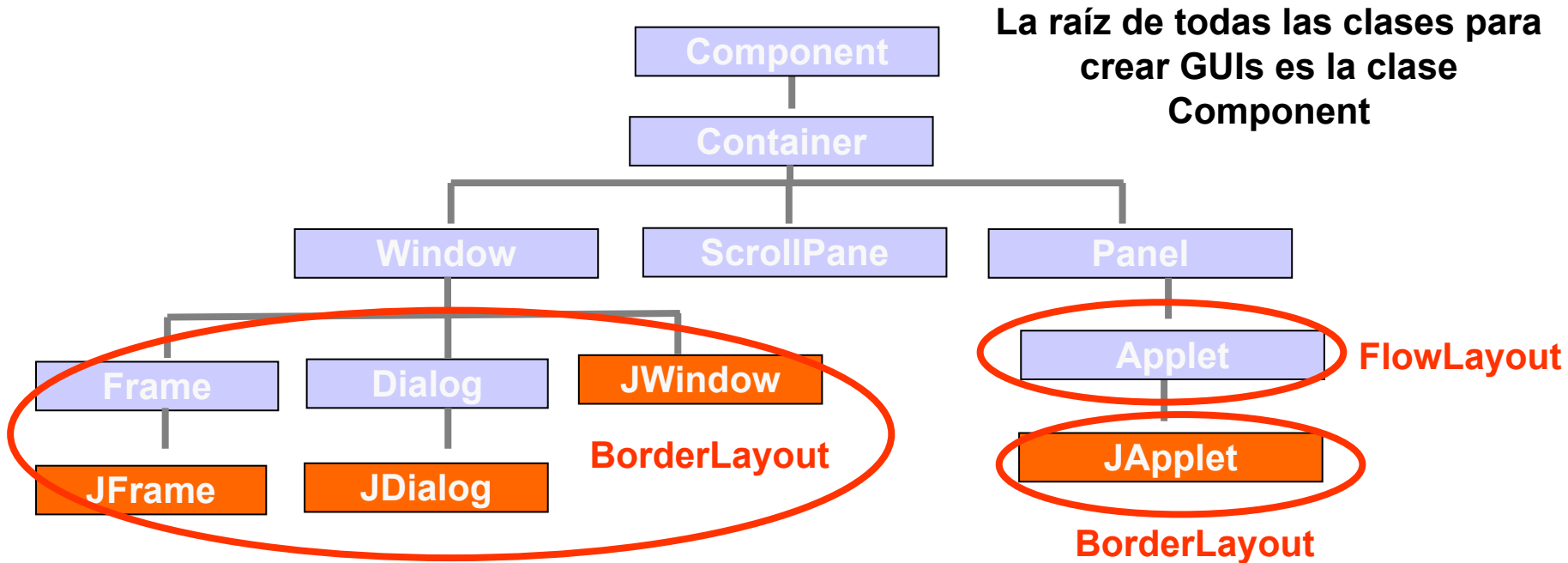


Básicamente para transformar una aplicación AWT en una aplicación Swing, se debe: cambiar la librería de AWT: `java.awt.*` por la de SWING: `javax.swing.*` y anteponer la **J** al nombre de las componentes

Swing

Contenedores

Los Layouts son los mismos que en AWT con excepción de Applet que cambió de FlowLayout a BorderLayout.



Aplicaciones Swing

Un ejemplo simple

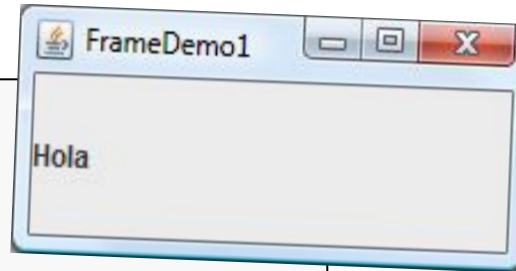
Creación de un JFrame simple

```
package componentes;

import javax.swing.JFrame;
import javax.swing.JLabel;

public class FrameDemo1 extends JFrame {
    private FrameDemo1() {
        super("FrameDemo1");
        JLabel label = new JLabel("Hola");
        this.setSize(200,100);
        this.add(label);
    }

    public static void main(String[] args){
        FrameDemo1 frame = new FrameDemo1();
        frame.setVisible(true);
    }
}
```

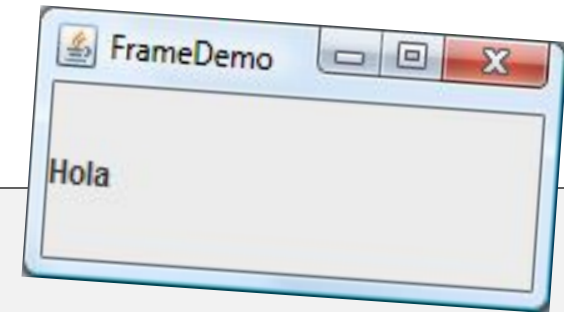


```
package componentes;

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

public class FrameDemo {
    private static void createAndShowGUI() {
        JFrame frame = new JFrame("FrameDemo");
        JLabel label = new JLabel("Hola");
        frame.add(label);
        frame.setSize(200,100);
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        FrameDemo.createAndShowGUI();
    }
}
```



Aplicaciones Swing

Un ejemplo con JTextArea

Un objeto JTextArea proporciona un área para manipular varias líneas de texto.

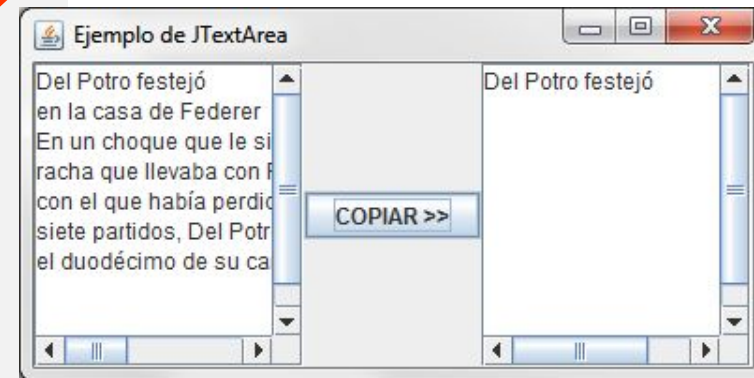
```
package textos;
import java.awt.event.*; import javax.swing.*;

public class VentanaTextos extends JFrame {
    private JTextArea area1, area2;
    private JButton copiar;
    public VentanaTextos() {
        super("Ejemplo de JTextArea");
        Box cuadro = Box.createHorizontalBox();

        String texto = "Del Potro festejó \n"
            + "en la casa de Federer \n"+...);
        area1 = new JTextArea(texto, 10, 15);
        cuadro.add(new JScrollPane(area1));
        copiar = new JButton("COPIAR >>");
        cuadro.add(copiar);
        copiar.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evento) {
                area2.setText(area1.getSelectedText());
            }
        });
        area2 = new JTextArea(10, 15);
        area2.setEditable(false);
        cuadro.add(new JScrollPane(area2));
        this.add(cuadro);
        this.setSize(400, 200);
        this.setVisible(true);
    }
}
```

Un Box es un tipo de Container que usa un layout manager.

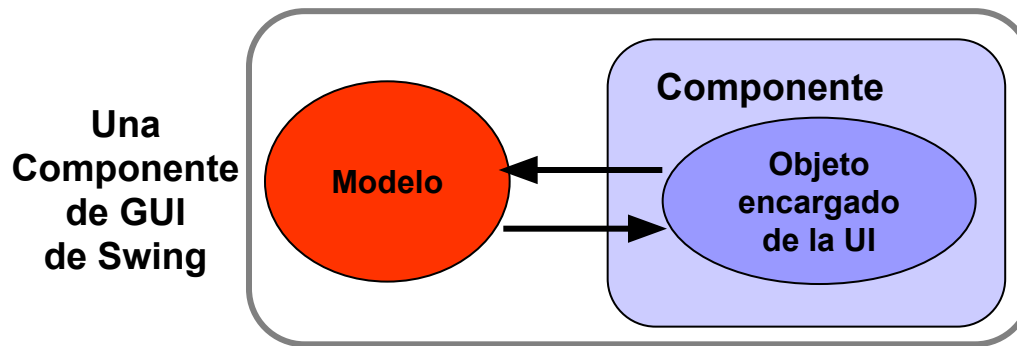
Esto lo veremos más adelante. Este código maneja el copiado de texto desde un área de texto a la otra.



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

Componentes Swing

La arquitectura de las componentes Swing responde a una **versión especializada del patrón MVC**. Está basada en 2 objetos: un objeto **Modelo** y un objeto encargado del *display* o apariencia y del manejo de los eventos, que representan la **Vista** + el **Controlador** en el patrón MVC.



El Modelo es tratado como un objeto separado

La Vista y el Controlador están acoplados en un único objeto encargado de la UI

- Cuando se crea una componente Swing, se crea automáticamente su objeto encargado de la UI, **no hay que preocuparse por eso!**
- El objeto **Modelo** es responsable de almacenar el estado de la componente. Si una aplicación, no provee un modelo explícito para una componente, Swing le crea un modelo por defecto. Swing provee un modelo por defecto para cada componente visual. Por ejemplo: **JTable -> DefaultTableModel, ...**

Componentes Swing - JList

Si la componente es creada sólo para visualizar y permitir seleccionar datos, en general alcanza con el modelo por defecto!!, pero ... si la componente es más compleja, y necesita manipular sus datos, es decir agregar/eliminarse datos del modelo, se debe crear un modelo especial!!.

A Definición de una Lista con un modelo por defecto:

La clase **JList**, tiene varios constructores:

- | | |
|---|--|
| ● <code>JList()</code> - <code>JList</code> | Si inicializamos la lista con estos constructores, podemos agregar y borrar elementos. |
| ● <code>JList(ListModel arg0)</code> - <code>JList</code> | |
| ● <code>JList(Object[] arg0)</code> - <code>JList</code> | Si inicializamos la lista con un arreglo o un vector, el constructor crea un modelo por defecto, que es IMUTABLE!!! |
| ● <code>JList(Vector<?> arg0)</code> - <code>JList</code> | |

Para trabajar con listas que sólo muestren opciones, NO es necesario crear Modelos. La componente provee uno por defecto.

Componentes Swing - JList

Creación de una lista de selección con un **modelo por defecto**.

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

```
public class ListaPlanetas extends JFrame {
    JList lista;
    JButton imprimir;
```

Se crea el arreglo con los valores

```
    public void init() {
        String[] items = { "Mercurio", "Tierra", "Saturno", "Jupiter" };
        lista = new JList(items);
        lista.setVisibleRowCount(4);
        imprimir = new JButton("Imprimir Selección");
        imprimir.addActionListener(new MyActionListener());
        this.setLayout(new FlowLayout());
        this.add(new JScrollPane(lista));
        this.add(imprimir);
    }
```

Se crea un Jlist con un modelo por defecto con los datos del arreglo

Las listas (JList) no poseen barras de desplazamiento, por lo tanto, para que dispongan de esta capacidad se las debe colocar dentro de un JScrollPane.

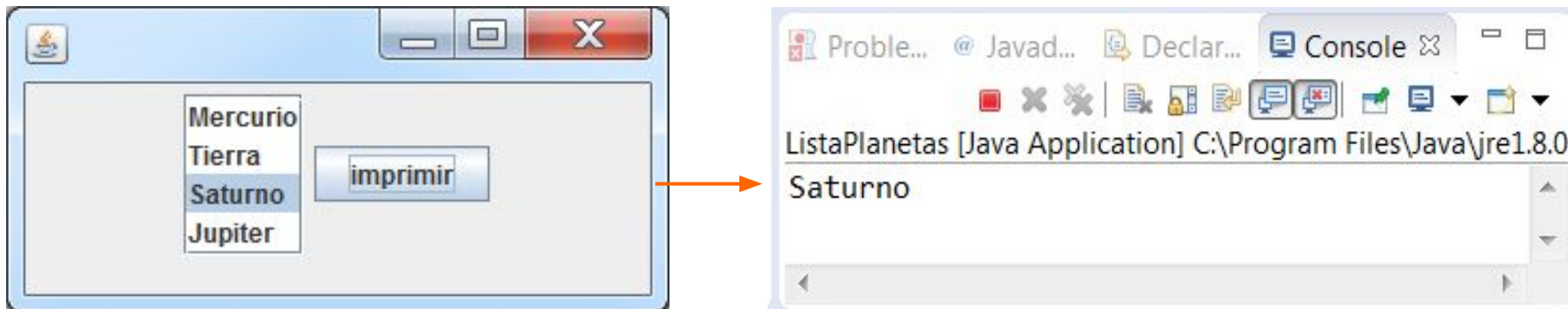
```
    private class MyActionListener implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            System.out.println(lista.getSelectedValue());
        }
    }
```

```
    public static void main(String[] args) {
        ListaPlanetas planetas = new ListaPlanetas();
        planetas.init();
        planetas.setVisible(true);
    }
}
```



Componentes Swing - JList

¿Cómo funciona la aplicación que implementamos?



Sólo se pueden visualizar datos fijos, y seleccionar items de la lista. No se pueden eliminar, reemplazar, ni agregar valores al Modelo creado por defecto.

Componentes Swing – JList

B Definición de una Lista con un modelo particular:

¿Cómo creo una lista de selección con un modelo actualizable? La idea es poder agregar y eliminar valores a la lista

Hay 2 alternativas:

- 1 Se crea usando el constructor por defecto (el que no tiene argumentos) y luego se le setea un **Modelo**:

```
JList lista = new JList();  
lista.setModel(modelo);
```

modelo es de tipo
DefaultListModel o
subclase

- 2 Se crea usando el constructor que tiene un **Modelo** como argumento:

```
JList lista = new JList(modelo);
```

Componentes Swing – JList

```
public class ListaPlanetas2 extends JFrame {
    private JList lista = new JList();
    private DefaultListModel modelo = new DefaultListModel();
    private JButton agregar = new JButton("Agregar");
    private JButton borrar = new JButton("Borrar");
    private JTextField texto = new JTextField(15);

    public void init() {
        String[] items = { "Mercurio", "Tierra", "Saturno", "Jupiter" };
        for (String item : items) {
            modelo.addElement(item);
        }
        lista.setModel(modelo);
        lista.setVisibleRowCount(4);

        agregar.addActionListener(new AgregarItemListener());
        borrar.addActionListener(new BorrarItemListener());
        this.setLayout(new FlowLayout());
        this.add(new JScrollPane(lista));
        this.add(texto);
        this.add(agregar);
        this.add(borrar);
    }
    class AgregarItemListener implements ActionListener {}
    class BorrarItemListener implements ActionListener {}

    public static void main(String[] args) {
        ListaPlanetas2 planetas = new ListaPlanetas2();
        planetas.init();
        planetas.pack();
        planetas.setVisible(true);
    }
}
```

Se usa el constructor sin argumentos!!

Se instancia un objeto **DefaultListModel**, para poder agregar y eliminar elementos de la lista.

Se crea el arreglo

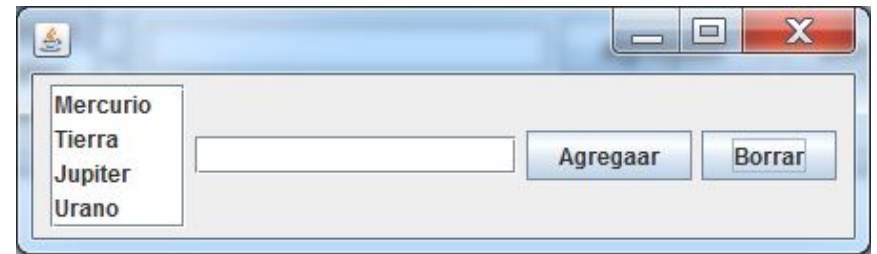
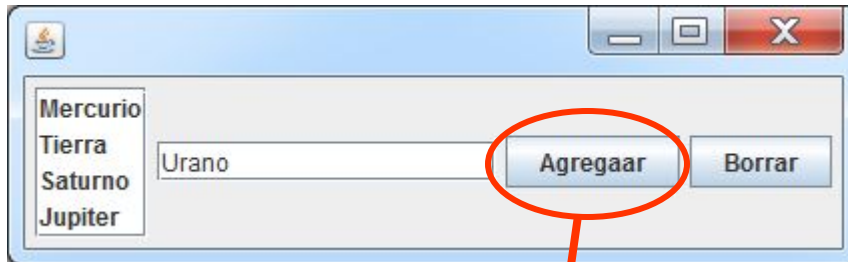
Se itera el arreglo y se guardan los valores en el **Modelo**

Se liga el **Modelo** a la componente **JList !!**

Registramos listeners en los botones

Componentes Swing – JList

¿Cómo funciona ahora la aplicación?



Al insertar en el Modelo de la componentes Swing, se actualiza la Vista. Como no pueden visualizarse todos los valores, aparecen las barras de desplazamiento.

```
class AgregarItemListener implements ActionListener{
    public void actionPerformed(ActionEvent e) {
        modelo.addElement(texto.getText());
    }
}
```

Al eliminar del Modelo el ítem seleccionado de la JList, se actualiza la Vista. Como pueden visualizarse todos los valores, desaparecen las barras de desplazamiento.

```
class BorrarItemListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        modelo.remove(lista.getSelectedIndex());
    }
}
```

Componentes Swing – JTable

La clase **JTable** es un poco más sofisticada que **JList**, pero tiene un manejo parecido.

Para trabajar con tablas, que sólo despliegue filas de datos con formato String, NO es necesario instanciar Modelos.

Para crear tablas donde se pueda agregar/borrar filas hay que usar modelos. El modelo por defecto de las tablas es **DefaultTableModel**

Cuando se crea un JTable se pueden especificar los valores de las celdas con alguno de los siguientes constructores:

● `JTable(TableModel dm) - JTable`

Si inicializamos la lista con este constructor, podemos agregar y borrar elementos.

● `JTable(Object[][] rowData, Object[] columnNames) - JTable`

● `JTable(Vector rowData, Vector columnNames) - JTable`

Si usamos estos constructores crea un modelo por defecto, que es **INMUTABLE!!!**

Componentes Swing – JTable

Definición de una tabla con un modelo particular hay dos alternativas:

- 1 Se crea usando el constructor por defecto y luego se le setea un **Modelo**:

```
JTable tabla = new JTable();  
tabla.setModel(modelo);
```

modelo es de tipo
DefaultTableModel
o una subclase

- 2 Se crea usando el constructor que tiene un **Modelo** como argumento:

```
JTable tabla = new JTable(modelo);
```

Componentes Swing – JTable

Creando un **JTable** con un objeto **DefaultTableModel** para poder modificar datos de la tabla

```
ReservaFrameTextual.java
package tablas;

import java.awt.BorderLayout;

public class ReservaFrameTextual {
    private JFrame f = new JFrame();
    private JTable tabla = new JTable();
    private JButton borrar = new JButton("Borrar");
    private JButton agregar = new JButton("Agregar");
    private JPanel panelBotones = new JPanel();

    private Object[] titulos = {"Titular", "Nacionalidad", "Origen", "Destino", "Fecha", "# Adul
    private DefaultTableModel modelo = new DefaultTableModel(titulos, 0);

    public ReservaFrameTextual(){
        Container contentPane = f.getContentPane();
        contentPane.setLayout(new BorderLayout());
        agregar.addActionListener(new AgregarListener());
        borrar.addActionListener(new BorrarListener());
        panelBotones.add(agregar);
        panelBotones.add(borrar);
        tabla.setModel(modelo);
        contentPane.add(panelBotones, BorderLayout.NORTH);
        contentPane.add(new JScrollPane(tabla), BorderLayout.CENTER);
        f.pack();
        f.setVisible(true);
    }

    class BorrarListener implements ActionListener{
    class AgregarListener implements ActionListener{

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

Se usa el constructor sin argumentos!!

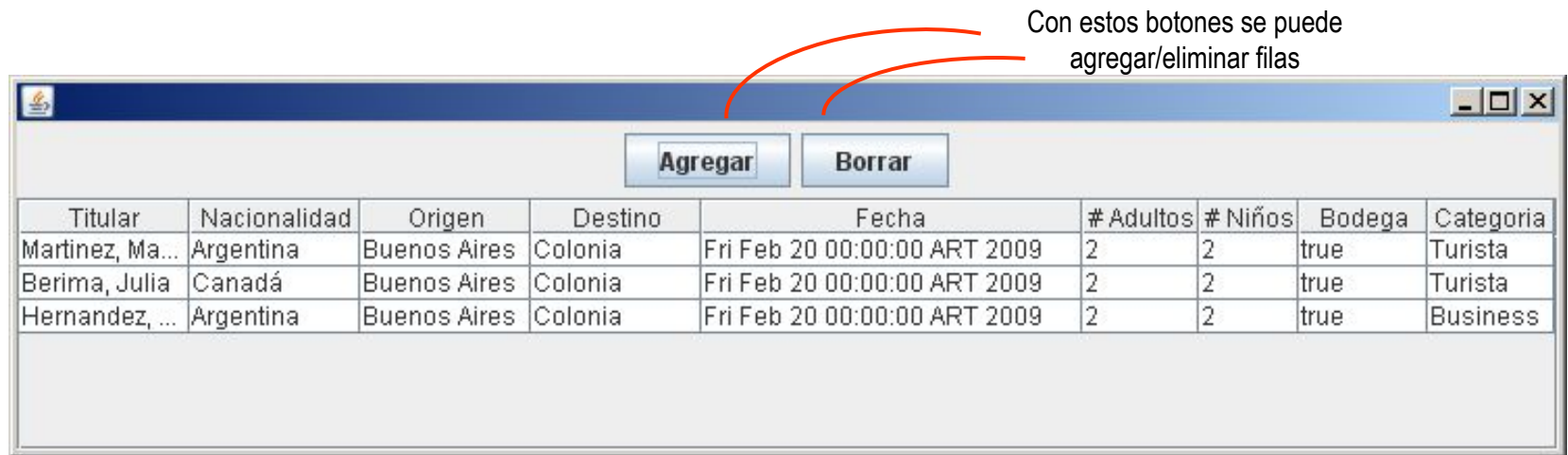
Se instancia un objeto **DefaultTableModel**, para poder agregar y eliminar elementos de la tabla y se lo liga a la tabla. Se manda como parámetro, un arreglo y la cantidad de filas que mantendrá.

Se liga el **Modelo** a la componente **JTable** !!

Si ejecutamos con el **main()**

Componentes Swing – JTable

Esta ventana contiene una tabla que fue creada con una instancia de la clase **DefaultTableModel**. La Tabla puede visualizar cualquier tipo de objeto en sus celdas, pero siempre lo muestra como un objetos de tipo String.

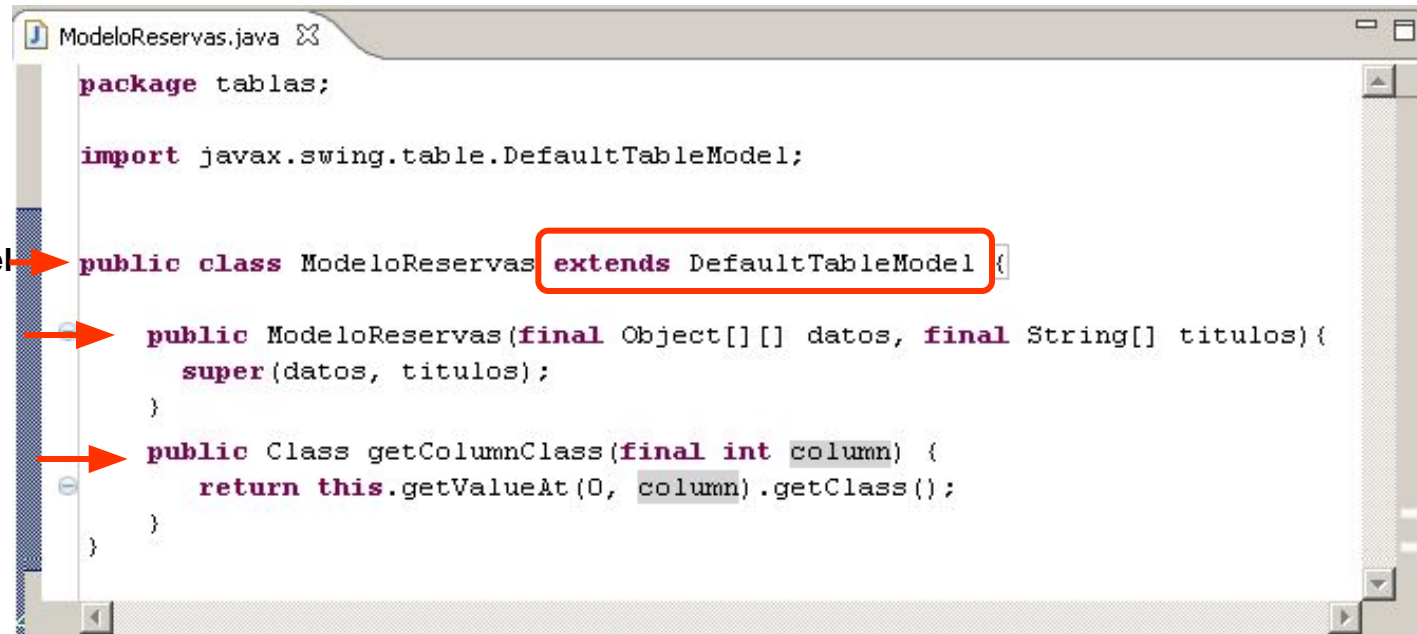


¿Puedo ver a la columna Nacionalidad como una imagen o a la columna Bodega como un objeto checkbox?

Si!!!, pero hay que trabajar un poco más con el modelo.

Componentes Swing – JTable

Los datos de la tabla son mostrados como String porque usamos como modelo una instancia de la clase **DefaultTableModel**. Esta clase tiene una manera por defecto de mostrar los datos y es en formato String. Para poder cambiar esto, debemos crear una clase que extienda a **DefaultTableModel** y sobrescribir un método para indicar el tipo de dato de cada celda.



```
ModeloReservas.java
package tablas;

import javax.swing.table.DefaultTableModel;

public class ModeloReservas extends DefaultTableModel {

    public ModeloReservas(final Object[][] datos, final String[] titulos){
        super(datos, titulos);
    }

    public Class getColumnClass(final int column) {
        return this.getValueAt(0, column).getClass();
    }
}
```

Se extiende el DefaultListModel

Se define un constructor que recibe como parámetro los títulos y datos.

Este método se debe sobrescribir para indicar de qué tipo es cada columna. De lo contrario siempre se muestran como String

Componentes Swing – JTable

Creando un **JTable** con objeto **customizado** de **DefaultListTable** para poder modificar la visibilidad de los datos.

```
ReservaFrame.java ✕  
  
public class ReservaFrame {  
    private JFrame f = new JFrame();  
    private JTable tabla = new JTable(); ← Se usa el constructor sin argumentos!!  
    private JButton borrar = new JButton("Borrar");  
    private JButton agregar = new JButton("Agregar");  
    private JPanel panelBotones = new JPanel();  
  
    private String[] titulos={"Titular","Nacionalidad","Origen","Destino","Fecha","# Adultos","# Niños","Bodega","Cate  
    private Object[][] datos={  
        {"Martinez, María", new ImageIcon("Argentina.gif"), "Buenos Aires", "Colonia", new Date(), 2,2,true,"Turista"},  
        {"Berima, Julia", new ImageIcon("Canada.gif"), "Buenos Aires", "Colonia", new Date(), 2,2,true,"Turista"},  
        {"Hernandez, Juan", new ImageIcon("Argentina.gif"), "Buenos Aires", "Colonia", new Date(), 2,2,true,"Business"},  
    };  
    public ReservaFrame(){  
        Container contentPane = f.getContentPane();  
        contentPane.setLayout(new BorderLayout());  
        tabla.setModel(new ModeloReservas(datos, titulos)); ← Se crea una instancia del modelo customizado para  
        agregar.addActionListener(new AgregarListener()); ← visualizar cada datos según su tipo y se lo liga a la tabla.  
        borrar.addActionListener(new BorrarListener());  
        panelBotones.add(agregar);  
        panelBotones.add(borrar);  
        contentPane.add(panelBotones, BorderLayout.NORTH);  
        contentPane.add(new JScrollPane(tabla), BorderLayout.CENTER);  
        f.pack();  
        f.setVisible(true);  
    }  
    public static void main(String[] args) {  
        ReservaFrame vent = new ReservaFrame();  
    }  
    class BorrarListener implements ActionListener{  
    class AgregarListener implements ActionListener{
```

Componentes Swing – JTable

La tabla tiene la siguiente apariencia. Ahora cada dato se visualiza diferente, según la clase a la que pertenece:



Titular	Nacionalidad	Origen	Destino	Fecha	# Adultos	# Niños	Bodega	Categoria
Martinez, María		Buenos Aires	Colonia	20-feb-2009	2	2	<input checked="" type="checkbox"/>	Turista
Berima, Julia		Buenos Aires	Colonia	20-feb-2009	2	2	<input checked="" type="checkbox"/>	Turista
Hernandez, Ju...		Buenos Aires	Colonia	20-feb-2009	2	2	<input checked="" type="checkbox"/>	Business

Instancia de
ImageIcon

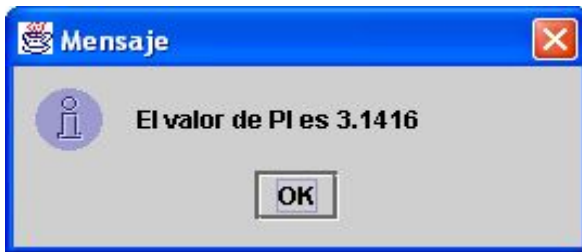
Instancia de
Boolean

Las tablas pueden ser provistas de más características, se puede configurar tamaño fijo o variable para las columnas, se puede permitir o no la edición de una o más columnas, se puede fijar una columna y permitir el *scroll* del resto, etc. Las analizadas son algunas de las funcionalidades más utilizadas con los objetos JTable.

Componentes Swing

Ventanas de Diálogo

Swing también provee una clase que automatiza muchas de las actividades que un programador haría para crear ventanas de diálogo: **JOptionPane**. Esta clase tiene muchos métodos de clase que permiten crear diálogos con íconos, mensajes, campos de entrada y botones.



```
JOptionPane.showMessageDialog(  
    null,                //padre  
    "El valor de PI es "+pi, //mensaje  
    "Mensaje",           // título  
    JOptionPane.INFORMATION_MESSAGE); //hay 5 tipos de mensajes  
}
```



```
JOptionPane.showConfirmDialog(  
    boton,                //padre  
    "¿Salva antes de salir?", //mensaje  
    "Mensaje",           //título  
    JOptionPane.YES_NO_CANCEL_OPTION, //tipo de opción  
    JOptionPane.WARNING_MESSAGE,    // tipo de mensaje  
    new ImageIcon("duke.gif"));      //ícono
```

Todos los diálogos creados con **JOptionPane**, son modales.

Aplicaciones Swing

Un ejemplo simple de pintado

De la misma forma que en AWT, las componentes Swing son mostradas en la pantalla invocando el método `paint(Graphics g)` de la clase `javax.swing.JComponent`.

El método `paint(Graphics g)` de `JComponent` considera que la componente puede ser *double buffer*, puede tener borde y que puede contener otras componentes. De esta manera, internamente invocará a otros métodos:

- `protected void paintComponent(Graphics g)`
- `protected void paintBorder(Graphics g)`
- `protected void paintChildren(Graphics g)`

Sin embargo cuando se trabaja con Swing solamente es necesario sobrescribir el método `paintComponent()` (en AWT se debe sobrescribir el `paint(Graphics g)`).

El método `repaint()` controla el pintado invocando la secuencia: `update()` □ `paintComponent()`.

Invocar al `repaint()` para pedir que la componente sea repintada/redibujada.

Aplicaciones Swing

Un ejemplo simple de pintado

Cuando se necesite repintar **el sistema AWT** invoca automáticamente al método `repaint()`, el cual primero invoca al `update()` para limpiar la componente y luego al `paint()` de `Component/JComponent` para que se repinte.

Pintado en AWT

```
package interfaces.graficas;
import java.awt.Graphics;
import java.awt.Frame;
...
public class LoadImageDemo extends Frame {
    private String imgFileName = "imagenes/duke.jpg";
    private Image img;      // es un objeto BufferedImage
    public LoadImageDemo() { . . . }
    public void dibujar() {
        Graphics gr = img.getGraphics();
        gr.drawLine(150, 150, 280, 150);
        gr.setColor(Color.RED);
        gr.setFont(new Font(Font.DIALOG, Font.BOLD, 30));
        gr.drawString("Soy DUKE", 30, 30);
        gr.fillRect(480, 290, 20, 20);
        this.getGraphics().drawImage(img, 50, 50, null);
    }
    @Override
    public void paint(Graphics g) {
        super.paintComponents(g);
        dibujar();
    }
    public static void main(String[] args) {
        LoadImageDemos app = new LoadImageDemo();
    }
}
```

Pintado en SWING

```
package interfaces.graficas;
import java.awt.Graphics;
import javax.swing.JFrame;
...
public class LoadImageDemo extends JFrame {
    private String imgFileName = "imagenes/duke.jpg";
    private Image img;      // es un objeto BufferedImage
    public LoadImageDemo() { . . . }
    public void dibujar() {
        Graphics gr = img.getGraphics();
        gr.drawLine(150, 150, 280, 150);
        gr.setColor(Color.RED);
        gr.setFont(new Font(Font.DIALOG, Font.BOLD, 30));
        gr.drawString("Soy DUKE", 30, 30);
        gr.fillRect(480, 290, 20, 20);
        this.getGraphics().drawImage(img, 50, 50, null);
    }
    @Override
    public void paintConponent(Graphics g) {
        super.paintComponents(g);
        dibujar();
    }
    public static void main(String[] args) {
        LoadImageDemos app = new LoadImageDemo();
    }
}
```

Referencias

- **Pensando en Java 3a edición Español, Bruce Eckel. Capítulo 14, Creando Windows & Applets.**

- **Java tutorial: Creating a GUI With JFC/Swing**

<http://docs.oracle.com/javase/tutorial/uiswing/index.html>

- **Java accessibility**

<https://docs.oracle.com/javase/tutorial/uiswing/misc/access.html>