

PROGRAMACIÓN

TEMA 7. INTERFACE GRÁFICA

CENTRO: *CENTRO ESPECÍFICO DE EDUCACIÓN A DISTANCIA DE LA
COMUNIDAD VALENCIANA (CEEDCV)*
WEB: *<http://www.ceedcv.org>*
AÑO *2015/2016*
ACADÉMICO:

PROFESOR: *FRANCISCO ALDARIAS RAYA*
EMAIL: *paco.aldarias@ceedcv.es*

MODULO: *PROGRAMACIÓN*
CICLO FORMATIVO: *CFGS: DESARROLLO DE APLICACIONES WEB*
DEPARTAMENTO: *INFORMÁTICA*
CURSO: *1º*



Licencia de Creative Commons.

PROGRAMACIÓN

TEMA 7. INTERFACE GRÁFICA

por Francisco Aldarias Raya

es licencia bajo

<http://creativecommons.org/licenses/by-nc-sa/3.0/es/>

Creative Commons Reconocimiento-NoComercial-CompartirIgual 3.0 España License.

Creado a partir de la obra en <http://www.ceedcv.es>

Bajo las condiciones siguientes:



Reconocimiento - Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciador (pero no de una manera que sugiera que tiene su apoyo o apoyan el uso que hace de su obra).



No comercial - No puede utilizar esta obra para fines comerciales.



Compartir bajo la misma licencia - Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.

Versión 1. 2011-2012

Versión 2. 2013-2014

Versión 3. 2014-2015

Versión 4. 2015-2016

Changeset: 34:3452f54ec3cb

Ultima actualización 7 de marzo de 2016

FRANCISCO ALDARIAS RAYA

paco.aldarias@ceedcv.es

Departamento de Informática.

Centro Específico de Educación a Distancia de la Comunidad Valenciana. (CEED)

<http://www.ceedcv.org>

En Valencia (España).

Documento realizado con software libre: linux debian, L^AT_EX.

Índice general

1.	INTRODUCCIÓN A SWING.	6
1.1.	Aplicación de consola vs Aplicación con GUI	6
1.2.	Java Foundation Classes	7
1.3.	Visión general de la librería Swing	8
1.4.	Interfaces de usuario con NetBeans y Matisse	8
1.5.	Tipos de aplicaciones con GUIs	9
2.	CONTENEDORES DE ALTO NIVEL	9
3.	SWING.	11
3.1.	Los componentes de swing	11
3.2.	Los contenedores de swing	13
3.3.	Ejemplo: Contenedores	13
4.	JFRAME	14
4.1.	Hola Mundo con Swing	15
4.2.	Cerrar una Ventana con Java Swing	16
5.	JLABEL.	17
5.1.	Jerarquía de clases	18
5.2.	API	18
5.3.	Práctica	18
6.	JTEXTFIELD	19
6.1.	JTextField	19
6.2.	Ejemplo: swing6	20
6.3.	Práctica	21
6.4.	API	21
6.5.	Jerarquía de clases	23
6.6.	Eventos	23
6.7.	Verificar entradas.	24
7.	LAYOUT	25
7.1.	FlowLayout	25

7.2.	GridLayout	26
7.3.	BorderLayout	27
7.4.	Combinando	28
8.	ACTIONLISTENER	29
8.1.	Una clase normalita	30
8.2.	Una clase interna	31
8.3.	Hacer que la clase principal implemente ActionListener	32
8.4.	Clase anónima	33
8.5.	getSource	34
8.6.	getActionCommand	35
9.	EVENTOS	37
9.1.	Pasos básicos para la creación de una IGU	37
9.2.	Creación de un proyecto con IGU	38
9.3.	Eventos	43
9.4.	Práctica	45
10.	JBUTTON	46
10.1.	API	46
10.2.	Práctica	47
10.3.	Ejemplo: JButton con Eventos	47
11.	MVC. FORMULARIO	49
12.	JDIALOG	53
12.1.	API	54
12.2.	Ejemplo:	54
12.3.	Práctica	55
13.	JFRAME Y JDIALOG	55
14.	MVC. APLICACION EUROS-PESETAS	57
14.1.	Controlador	57
14.2.	Modelo	58
14.3.	Vista	59
15.	JLIST Y JCOMBOBOX.	62
15.1.	JComboboxEjemplo1	64
15.2.	JComboboxEjemplo2	65
15.3.	Práctica	66
15.4.	API	66
15.5.	Eventos	66
16.	EJERCICIOS PROPUESTOS	67

16.1.	Ejercicio0901. Conversor	68
16.2.	Ejercicio0902 DNI	69
16.3.	Ejercicio0903 Modelo Compartido	69
16.4.	Ejercicio0904 Calculadora	70
16.5.	Ejercicio0905 InputVerifier	70
17.	JPANEL	70
17.1.	Definición	70
17.2.	Creado varios tipos de paneles	71
17.3.	API	71
17.4.	Práctica: jpanel01	72
17.5.	Identificar el panel	72
17.6.	Práctica: jpanel02	73
18.	MENÚS	74
19.	JTABLE	76
19.1.	Conceptos básicos	76
19.2.	JTable y Netbeans	80
19.3.	Crear un modelo de tabla	81
19.4.	Práctica	83
19.5.	Utilizando un combobox como editor de celda	84
19.6.	Personalizando la edición de celdas	85
19.7.	Práctica	87
19.8.	Ordenación en tablas	87
19.9.	Práctica	88
19.10.	API	89
19.11.	Eventos	89
19.12.	Práctica	90
20.	JCALENDAR	90
21.	MVC. SWING	90
21.1.	Introducción	91
21.2.	Construcción Interna de Componentes.	91
21.3.	Varios componentes representan los mismos datos	94
21.4.	Eventos en el modelo	96
21.5.	Construcción de una aplicación con GUI	96
21.6.	Práctica	97
22.	JNLP	98
22.1.	Java Web Start	98

22.2.	JNLP	98
22.3.	Ejemplo	99
22.4.	Qué hace	99
22.5.	Cómo funciona	100
22.6.	Estructura de un archivo JNLP	100
22.7.	Ejemplo	100
22.8.	JNLP con Netbeans	101
22.9.	Práctica	102
23.	JAPPLET	102
23.1.	Primer Applet	103
23.2.	Appletviewer	105
23.3.	Applet con Netbeans	106
24.	ORIENTACIÓN	108
24.1.	Objetivos	108
24.2.	Prácticas	108
24.3.	Código Fuente	108
24.4.	Software	109
24.5.	Datos bibliográficos	109
24.6.	Recursos en Internet	109
24.7.	Temporalización	110

1. INTRODUCCIÓN A SWING.

Este apartado pretende dar una visión general de las interfaces gráficas de usuario. Para ello, entre otras cosas, describe de forma breve qué se entiende por interfaz de usuario e interfaz gráfica de usuario. Veremos los conceptos generales sobre elementos gráficos y eventos, e indica los pasos básicos a seguir en la construcción de una aplicación con interfaz gráfica.

1.1. Aplicación de consola vs Aplicación con GUI

Los elementos que componen la interfaz gráfica son elementos gráficos, y a través de ellos el usuario puede interactuar con la aplicación. En esta interacción el usuario introduce datos que el programa necesita para llevar a cabo su funcionalidad y obtiene los resultados de procesar dichos datos. Por ejemplo, las ventanas, los botones, las imágenes, etc. son elementos gráficos.

Una diferencia clara entre una aplicación de consola y una aplicación con interfaz gráfica de usuario, es que la primera no tiene ningún elemento gráfico, mientras que en la segunda éstos si existen. Por otra parte, un evento es la notificación que hace un elemento gráfico cuando el usuario interactúa con él.

Por lo tanto, si se realiza alguna acción sobre algún elemento de la interfaz, se dice que se ha generado un evento en dicho elemento.

Otra diferencia destacable entre una aplicación de consola y una con interfaz gráfica de usuario está relacionada con los eventos y su gestión, y afecta notablemente a la hora de programar la aplicación. En una aplicación de consola el programa decide cuándo necesita datos del usuario, y es en ese momento cuando los lee de la cadena de entrada. Sin embargo, una aplicación con interfaz gráfica siempre se encuentra a la espera de una entrada de datos por parte del usuario. Éste, en cualquier momento, puede realizar alguna acción sobre algún elemento de la interfaz (por ejemplo, pulsar con el ratón sobre un botón, introducir un carácter por teclado, etcétera).

Para poder atender las acciones realizadas sobre los elementos de la interfaz gráfica de usuario, es necesario asociar código a los eventos que se puedan generar como consecuencia de dichas acciones. De esta manera, si se asocia código a un evento concreto, éste se ejecutará cuando se realice la acción que genera el evento sobre un elemento de la interfaz. Al código asociado a los eventos se le denomina código de gestión de eventos.

A la hora de programar una aplicación, dependiendo si ésta es de consola o con interfaz gráfica, dadas las diferencias existentes entre ellas (existencia o no de elementos gráficos y tratamiento de eventos), los pasos a seguir serán bastante distintos.

En las aplicaciones de consola, básicamente lo que se le puede mostrar al usuario son cadenas de caracteres, pudiéndose definir, en algunos casos, la posición donde se desea que éstas aparezcan. Sin embargo, en las aplicaciones con interfaz gráfica de usuario, se puede hacer uso de un conjunto de elementos gráficos que permiten una mejor interacción del usuario con la aplicación.

1.2. Java Foundation Classes

Las Java Foundation Classes (JFC, en castellano Clases Base Java) son un framework gráfico para construir interfaces gráficas de usuario portables basadas en Java. JFC se compone de Abstract Window Toolkit (AWT), Swing y Java 2D. Juntas, suministran una interfaz de usuario consistente para programas Java, tanto si el sistema de interfaz de usuario subyacente es Windows, Mac OS X o Linux.

AWT es la más antigua de las dos APIs de interfaz, y fue criticada duramente por ser poco más que una envoltura alrededor de las capacidades gráficas nativas de la plataforma anfitrión. Esto significa que los widgets estándar en la AWT confían en esas capacidades de los widgets nativos, requiriendo que el desarrollador también esté prevenido de las diferencias entre plataformas anfitrión.

Una API de gráficos alternativa llamada Internet Foundation Classes fue desarrollada en código más independiente de la plataforma por Netscape. Ultimamente, Sun mezcló la JFC con otras tecnologías bajo el nombre "Swing", añadiendo la capacidad para un look and feel enchufable de los widgets. Esto permite a los programas Swing mantener la base del código independiente de la plataforma, pero imita el look de la aplicación nativa.

1.3. Visión general de la librería Swing

Los elementos que viene con las JFC podemos clasificarlos en los siguientes grandes grupos:

- 1.- Contenedores de más alto nivel: son los que están en lo más alto de la jerarquía (JFrame, JDialog, JApplet)
- 2.- Contenedores de propósito general: contenedores intermedios utilizados en diversas circunstancias (JPanel, JScrollPane, JSplitPane, etc. ...).
- 3.- Contenedores de propósito específico: contenedores intermedios que tienen papeles específicos en el diseño de interfaces (JInternalFrame, JLayeredPane, etc. ...).
- 4.- Controles básicos: componentes atómicos que sirven para obtener la entrada del usuario (JButton, JComboBox, JList, JTextField, etc. ...).
- 5.- Controles de información: componentes atómicos que sólo existen para dar información al usuario (JLabel, JToolTip, JSlider, etc. ...).
- 6.- Controles interactivos: componentes atómicos que muestran información altamente formateada y que puede ser modificada por el usuario (JTable, JTree, JColorChooser, etc. ...)

Echad un vistazo a la siguiente dirección Web: <http://docs.oracle.com/javase/tutorial/uiswing/TOC.html>

1.4. Interfaces de usuario con NetBeans y Matisse

La construcción de interfaces gráficas de usuario ha evolucionado mucho desde la primera versión de Java (Java 1.0). En las primeras versiones se utilizaba la librería AWT (Abstract Windows Toolkit), la cual no era muy potente. Posteriormente, se creó una librería estándar mucho más completa, llamada Swing. Por motivos de compatibilidad y reutilización, la librería Swing sigue utilizando, para su funcionamiento, parte de la librería AWT.

Matisse es el nuevo constructor de interfaces gráficas que se ha añadido en la versión de Netbeans 5.0 y siguientes. Con este nuevo módulo se simplifica todavía más la tarea de construcción del GUI de una aplicación.

En lugar de estar basado en la selección y ajuste por parte del desarrollador de un layout manager Swing, Matisse permite la creación en un modo libre de la interface y el propio entorno infiere el layout manager y el resto de características del mismo, como reglas de redimensionado, espaciado, etc.

Para echar un vistazo a la página de matisse podemos acceder a la siguiente dirección: <http://netbeans.org/features/java/swing.html>

1.5. Tipos de aplicaciones con GUIs

La tecnología Java está muy extendida a día de hoy y permite crear muchos tipos de aplicaciones, con y sin interfaz gráfica. Por ejemplo, aplicaciones de consola, Servlets, Applets, etc.

Las aplicaciones de consola, como ya se ha visto, utilizan la entrada y salida estándar como medio para interactuar con el usuario mediante texto.

Los Servlets son aplicaciones que se ejecutan en un Servidor Web e interactúan con el usuario mediante tecnologías Web (JavaScript, HTML, HTTP, etcétera). Este tipo de aplicaciones no hacen uso de la librería Swing.

Las aplicaciones con interfaz gráfica de usuario se pueden construir tanto para ordenadores personales como para otro tipo de dispositivos, por ejemplo para los teléfonos móviles.

A continuación se presentan, brevemente, los diferentes tipos de aplicaciones que se pueden construir en Java usando Swing: Aplicaciones Autónomas, Applets y Aplicaciones Java Web Start.

1.5.1. Aplicaciones autónomas

Una aplicación autónoma es aquella cuyo funcionamiento es igual a las aplicaciones típicas que se ejecutan en cualquier sistema operativo. Su ejecución se inicia tras pulsar sobre un icono o invocar un comando en la línea de comandos. Habitualmente, cuando se está ejecutando una aplicación de este tipo, aparece en la barra de tareas.

1.5.2. Aplicaciones Java Web Start

Las aplicaciones Java Web Start son aplicaciones muy similares a las aplicaciones autónomas, en lo que se refiere a su funcionamiento durante la ejecución. La diferencia principal es que se pueden cargar desde un servidor Web e instalarse de forma muy cómoda simplemente pulsando un enlace en una página Web.

1.5.3. Applets

Un Applet es una pequeña aplicación Java que se ejecuta dentro de una página Web que está siendo visualizada en un navegador de Internet.

2. CONTENEDORES DE ALTO NIVEL

Los programas que usan componentes Swing ponen los componentes en árboles de contenidos, y cada árbol de contenidos tiene un contenedor de alto nivel en su raíz. Los contenedores de alto nivel son:

1.- JFrame

2.- JApplet

3.- JDialog/JOptionPane.

En general, cada aplicación tiene al menos un árbol de contenidos encabezado por un objeto JFrame. Cada Applet debe tener un árbol de contenido encabezado por un objeto JApplet. Cada ventana adicional de una aplicación o un Applet tiene su propio árbol de contenido encabezado por un frame o dialogo (JDialog/JOptionPane).

La diferencia que hay entre cada uno de los componentes es que un JDialog es una ventana modal que hasta que no se cierra permanece siempre visible. Un JFrame es una ventana por defecto no modal, es decir, que no necesariamente siempre será visible y un JApplet se utiliza en ambientes web empotrado dentro de una página html.

Para crear un contenedor de alto nivel, seleccionaremos Archivo - Nuevo (Ctrl. + N) dentro del proyecto en cuestión. Dentro del panel de categorías seleccionaremos Formularios de interfaz gráfica y dentro del panel de tipo de archivos seleccionaremos uno de los tres contenedores de formularios (JApplet, JDialog JFrame,).

Ver la figura 1 de la página 10

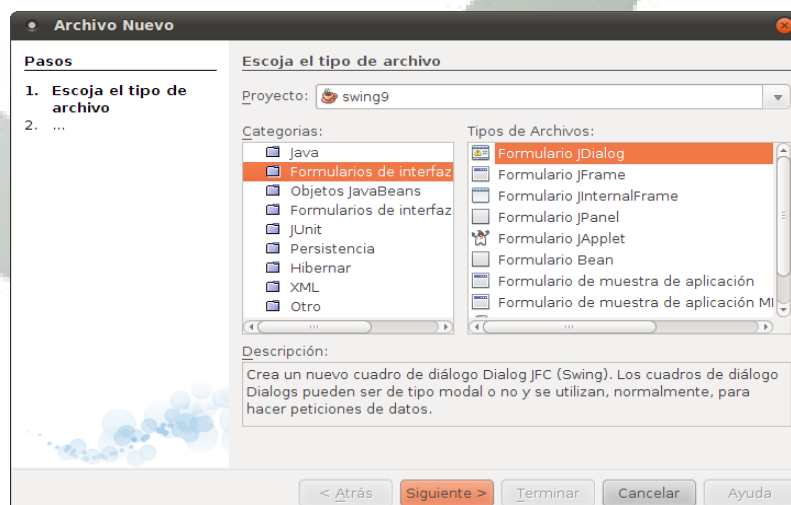


Figura 1: Contenedores de alto nivel

Una vez indicado el nombre de la clase Class Name y el package donde va a estar, el IDE de NetBeans mostrará en modo diseño el contenedor de alto nivel para ir creando el árbol de contenidos, es decir, ir añadiendo cada uno de los componentes que hemos visto anteriormente.

3. SWING.

Este apartado se acompañará con un conjunto de proyectos de ejemplo para cada uno de los componentes que se van a tratar. En cada componente habrá que abrir el proyecto y realizar algunos cambios que se pedirán para ese componente.

3.1. Los componentes de swing

En este apartado se presentan los diferentes elementos gráficos que tiene la librería Swing. Se presentan en diferentes tablas, agrupados por tipos. Se indica, en cada tabla, la clase que representa cada elemento acompañada de una breve descripción. Todas las clases de los diferentes elementos se encuentran en el paquete `javax.swing`.

En Netbeans a los componentes les llama **controles** y aparecen dentro de la paleta.

3.1.1. Componentes simples

Clase	Descripción
JLabel	Etiqueta. Muestra imágenes y texto. El texto se puede formatear usando HTML
JButton	Botón. Puede mostrar imágenes y texto
JCheckBox	Casilla de verificación
JRadioButton	Botón de radio. Se usa para seleccionar una opción entre varias
JToggleButton	Botón que se queda presionado al ser pulsado
JComboBox	Lista desplegable
JScrollBar	Barra de desplazamiento. Se usa en los contenedores que permiten que su contenido sea más grande que ellos. El desarrollador lo usará indirectamente cuando utilice el panel JScrollPane
JSeparator	Se usa en los menús y barras de herramientas para separar opciones
JSlider	Deslizador
JSpinner	Campo de texto con botones para elegir el elemento siguiente o anterior. Se puede usar para números, fechas o elementos propios
JProgressBar	Barra de progreso
JList	Lista de elementos

3.1.2. Componentes complejos

Clase	Descripción
JTable	Tabla
JTree	Arbol
JFileChooser	Selector de ficheros
JColorChooser	Selector de color
JOptionPane	Cuadro de dialogo personalizable
JComboBox	Lista desplegable

3.1.3. Componentes de texto

Clase	Descripción
JTextField	Campo de texto simple. Sólo puede contener una línea de texto
JFormattedTextField	Campo de texto que acepta textos que siguen un patrón
JPasswordField	Campo de texto para contraseñas
JTextArea	Área de texto simple. Puede contener varias líneas de texto
JEditorPane	Área de texto que puede mostrar estilos, tipos de letra, imágenes, etc. Muestra documentos en formato RTF y HTML 3.2
JTextPane	Área de texto que puede mostrar estilos, tipos de letra, imágenes, etc. Permite un mayor control sobre los estilos de texto que JEditorPane

3.1.4. Contenedores

Clase	Descripción
JPanel	Contenedor de propósito general
JScrollPane	Contenedor con barras de desplazamiento
JSplitPane	Contenedor dividido en dos partes
JTabbedPane	Contenedor con pestañas
JDesktopPane	Contenedor para incluir ventanas dentro
JToolBar	Barra de herramientas

3.1.5. Contenedores de Alto Nivel. Ventanas

Clase	Descripción
JFrame	Ventana de aplicación
JDialog	Cuadro de dialogo
JWindow	Ventana sin marco
JInternalFrame	Ventana interna

3.2. Los contenedores de swing

Para hacer una aplicación java con swing necesitamos crear un contenedor de nivel superior como JFrame, y además debemos poner un contenedor de nivel intermedio como JPanel, si queremos poner varias controles como JButton o JLabel. Si sólo vamos a tener un componente no hace falta el contenedor intermedio JPanel.

Un JFrame puede contener más de un JPanel. Ejemplo swing15.

Ver la figura 2 de la página 13

Relación entre componentes y contenedores

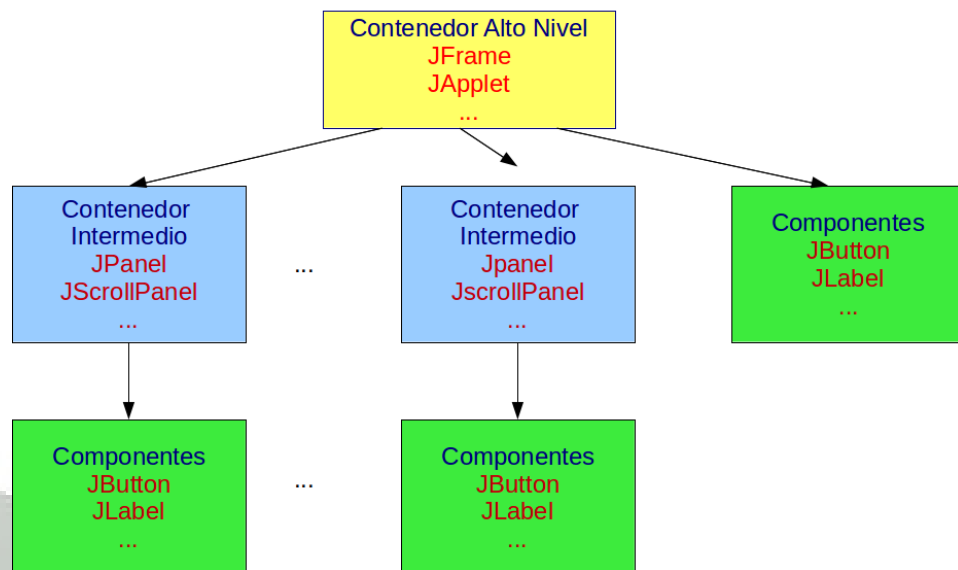


Figura 2: Contenedor Componentes

El panel (Jpanel), permite indicar como situar los controles con el método setLayout, o incluso contener otros paneles. Ejemplo que coloca los controles en una fila de izquierda a derecha:

```
frame.setLayout( new FlowLayout() );
```

3.3. Ejemplo: Contenedores

En en siguiente ejemplo se colocan varios componentes a través de un contenedor intermedio Jpanel.

Ver la figura 3 de la página 14

../prgcodigo/prgcod09/Componentes.java

```

i>_/*
Programa en Java que muestra
todos los componentes de swing
*/
  
```

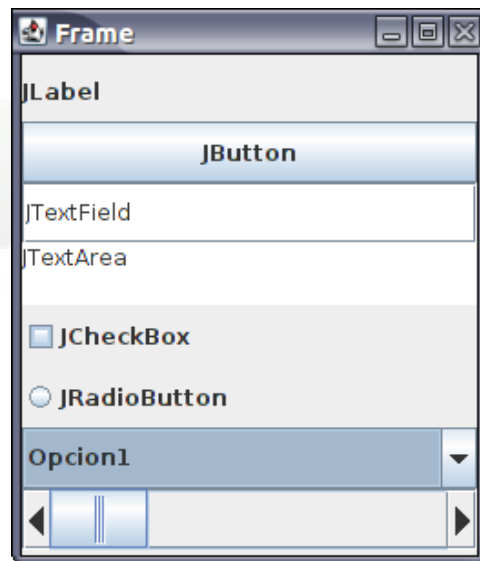


Figura 3: Componentes de Swing

```
import javax.swing.*;
import java.awt.*; // JScrollBar, setLayout

public class Componentes {

    public static void main(String[] args) {
        JLabel jlabel = new JLabel("JLabel");
        JButton jbutton = new JButton("JButton");
        JTextField jtextfield = new JTextField("JTextField");
        JTextArea jtextarea = new JTextArea("JTextArea");
        JCheckBox jcheckbox = new JCheckBox("JCheckBox");
        JRadioButton jradiobutton = new JRadioButton("JRadioButton");
        String[] opciones = { "Opcion1", "Opcion2", "Opcion3"};
        JComboBox jcombobox = new JComboBox(opciones);
        JScrollBar jscrollbar = new JScrollBar(Scrollbar.HORIZONTAL, 0, 64, 0, 255);
        try {
            JFrame frame = new JFrame("Frame");
            frame.add(jlabel);
            frame.add(jbutton);
            frame.add(jtextfield);
            frame.add(jtextarea);
            frame.add(jcheckbox);
            frame.add(jradiobutton);
            frame.add(jcombobox);
            frame.add(jscrollbar);
            frame.setLayout( new GridLayout(0,1) );
            frame.pack();
            frame.setVisible(true);
        } catch (Exception e) {}
    }
}
```

4. JFRAME

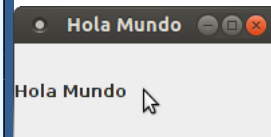
4.1. Hola Mundo con Swing

El siguiente ejemplo abre una nueva ventana que nos muestre el texto "Hola Mundo" en su interior.

Ejemplo donde vemos el código.

../prgcodigo/prgcod09/HolaMundo.java

```
package prgt9e10aldarias;
import javax.swing.JFrame;
import javax.swing.JLabel;
public class HolaMundoSwing extends JFrame {
    public HolaMundoSwing() {
        super("Hola Mundo");
        JLabel label = new JLabel("Hola Mundo");
        this.getContentPane().add(label);
        setSize(200, 100);
        setLocationRelativeTo(null);
        setVisible(true);
    }
    public static void main(String[] args) {
        new HolaMundoSwing();
    }
}
```



Explicación:

- 1.- Lo primero que tenemos que saber es que la clase que representa la ventana (o frame) es JFrame. Y que nuestra clase concreta heredará todo de dicha clase.

```
public class HolaMundoSwing extends JFrame {...}
```

De esta forma conseguimos que nuestra clase HolaMundoSwing ya represente una ventana.

- 2.- Lo siguiente será empezar con la codificación del constructor de la clase. En dicho constructor definiremos la etiqueta de texto, la añadiremos al contenedor y haremos que el contenedor sea visible.

```
public HolaMundoSwing() {...}
```

- 3.- En el constructor invocaremos al constructor del padre. Para ello nos ayudamos del método super. Como el constructor de JFrame, al menos uno de ellos, espera el título de la ventana como parámetro, ejecutaremos el método super() de la siguiente forma:

```
super("Hola Mundo");
```

- 4.- Ahora pasaremos a crear la etiqueta. Las etiquetas en Java Swing las podemos utilizar mediante la clase JLabel. El constructor de la clase JLabel espera el texto de la etiqueta como parámetro. En este caso, al ser "Hola Mundo" realizaremos la creación de la etiqueta de la siguiente forma:

```
JLabel label = new JLabel("Hola Mundo");
```


- 5.- Una vez creada la etiqueta la añadimos a la ventana mediante el método `.add()` del contenedor

```
JLabel label = new JLabel("Hola Mundo");
getContentPane().add(label);
```

- 6.- Ya solo nos quedará darle un tamaño a la ventana, centrarla y hacerla visible. Esto lo hacemos mediante los métodos `setVisible()` y `setSize()`.

```
setSize(200,100);
setLocationRelativeTo(null);
setVisible(true);
}
```

- 7.- El código completo del constructor sería el siguiente:

```
public HolaMundoSwing() {
    super("Hola Mundo");

    JLabel label = new JLabel("Hola Mundo");
    getContentPane().add(label);

    setSize(200,100);
    setLocationRelativeTo(null);
    setVisible(true);
}
```

- 8.- Ya solo nos quedará invocar a la clase `HolaMundoSwing`. Para ello creamos un método `main` y la instanciamos.

```
public static void main(String[] args) {
    new HolaMundoSwing();
}
```

4.2. Cerrar una Ventana con Java Swing

Vamos a ver como podemos cerrar una ventana con Java Swing. De esta manera controlaremos las acciones a realizar una vez que el usuario cierre la ventana.

La idea principal consiste en escuchar el evento `windowClosing`. El evento `windowClosing` es un evento del adaptador de ventana o `WindowAdapter`.

Para la gestión de eventos tenemos Interfaces y Adaptadores. Los interfaces nos obligan a codificar todos los eventos a gestionar, mientras que en el Adaptador solo tenemos que gestionar el evento que necesitemos.

Pero lo primero es utilizar el método `.addWindowListener` para suscribirnos a los eventos que se produzcan en la ventana. Así, en el constructor de nuestro programa utilizaremos dicho método:

```
public CerrarVentana(){
    addWindowListener(new WindowAdapter() {...});
}
```

Como podemos comprobar en el código, al método `.addWindowListener` le estamos pasando una clase `WindowAdapter`.

El método asociado al cierre de la ventana es `windowClosing`. En él solo vamos a realizar un `exit` del sistema con `System.exit`.

```
public void windowClosing(WindowEvent e) {  
    System.exit(0);  
}
```

Así nuestro `WindowAdapter` quedaría de la siguiente forma para gestionar el cierre de una ventana con Java Swing.

```
public CerrarVentana(){  
    addWindowListener(new WindowAdapter() {  
        public void windowClosing(WindowEvent e) {  
            System.exit(0);  
        }  
    });  
}
```

Aunque nosotros solo hemos forzado un `System.exit`, en el método `windowClosing` podemos hacer lo que queramos. Por ejemplo, preguntar al usuario si está seguro de cerrar.

5. JLABEL.

Las etiquetas, junto con los botones y las cajas de selección, son uno de los componentes más básicos de toda interfaz de usuario. Y el más simple de todos ellos es la etiqueta, que se limita a presentar textos en pantalla. Swing introduce la clase `JLabel` (`javax.swing.JLabel`) para presentar estos textos en pantalla; sin embargo, es mucho más versátil que la clase correspondiente del AWT. En Swing, al derivar de `JComponent`, la clase `JLabel` implementa todas las características inherentes a los componentes Swing, como pueden ser los aceleradores de teclado, bordes, y demás.

Tal y como mostramos en la figura algunos ejemplos de etiquetas los podemos ver en el ejemplo del proyecto `swing3`.

Ver la figura 4 de la página 17



Figura 4: swing3

Algunas de las propiedades interesantes son: font, foreground, icon, horizontalAlignment, text, tooltipText. Para utilizar el componente JLabel accedemos al panel Paleta:

En concreto la propiedad icon permite añadir una imagen.

5.1. Jerarquía de clases

Jerarquía de clases para crear un JLabel:

Jerarquía de clases:

```
└─ javax.swing.JFrame
   └─ javax.swing.JPanel
      └─ javax.swing.JLabel
```

5.2. API

La siguiente tabla lista los métodos más utilizados de JLabel:

Método	Propósito
getText() / setText()	Selecciona u obtiene el texto mostrado por la etiqueta.

Para mas información del API completa:

<http://download.oracle.com/javase/6/docs/api/javax/swing/JLabel.html>

5.3. Práctica

Abrir el proyecto swing3 y cambiar algunas de las propiedades que se han comentado anteriormente.

5.3.1. Ejemplo: Label

Ejemplo donde vemos el código.

../prgcodigo/prgcod09/MiJLabel.java

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

public class MiJLabel {

    MiJLabel() {
        // Creamos el Frame
        JFrame f = new JFrame();
        f.setTitle("Ejemplo JLabel");
        f.setLayout( new GridLayout() );
        f.setSize(230, 100);

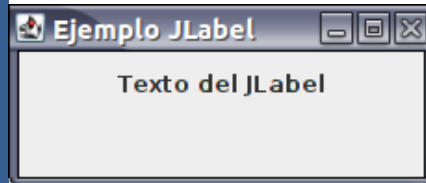
        // Anadimos el JPanel al JFrame
        JPanel p = new JPanel();
        f.add(p);

        // Creamos el JLabel y lo anadimos
        JLabel l = new JLabel();
        l.setText("Texto del JLabel");
        p.add(l);

        // Visualizamos el JFrame
        f.setVisible(true);

    }

    public static void main
    (String args[]){
        new MiJLabel();
    }
}
```



6. JTEXTFIELD

6.1. JTextField

Veamos un ejemplo básico de JTextField.

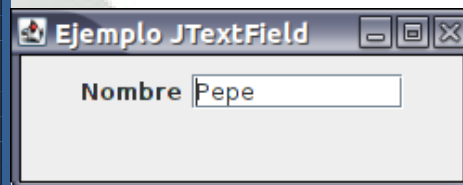
```
../prgcodigo/prgcod09/MiJTextField.java
```

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

public class MiJTextField {

    public static void muestra() {
        JFrame f = new JFrame();
        JPanel p = new JPanel();
        JTextField t = new JTextField();
        JLabel l = new JLabel();
        f.setTitle("Mi JFrame");
        l.setText("Nombre");
        t.setText("Pepe");
        t.setColumns(10);
        f.setLayout(new FlowLayout());
        f.add(p);
        p.add(l);
        p.add(t);
        f.setSize(200, 100);
        f.setVisible(true);
    }

    public static void main
        (String args[]) {
        muestra();
    }
}
```



6.2. Ejemplo: swing6

Los componentes que Swing presenta para la manipulación de textos son la clase `JTextArea`, la clase `JTextField`, las clases `JPasswordField` que viene a ser equivalente al uso de `JTextField` junto con la propiedad `echoChar` que indica cuál será el carácter de reemplazo en vez de mostrar el carácter real (por defecto *), la clase `JEditorPane` que permite cargar páginas con texto formateado (URLs), `JTextPane` que permite que se presente el texto con diferentes fuentes de caracteres, colores, tamaños, etc. y la clase `JFormattedTextField` que permite la introducción de texto formateado.

Ver la figura 5 de la página 21

Las propiedades más comunes que podemos ver en la pestaña de Propiedades son: `font`, `editable`, `toolTipText`, `page`, `text`, `foreground`, `echoChar`, `enabled`.

Para poder añadir mascarar a los campos `JFormattedTextField` desde el IDE de Netbeans hay que realizar el siguiente proceso:

- 1.- Seleccionamos el componente `JFormattedTextField`
- 2.- Accedemos al panel de Código
- 3.- Accedemos al evento Pre-Creation Código.
- 4.- Introducimos el código de creación de mascara:

```
MaskFormatter maskDNI = null;
try {
```



Figura 5: swing6

```
maskDNI = new MaskFormatter("###.###.###-?");
} catch (Exception ex) {
ex.printStackTrace();
}
```

Aclaración: Indicamos los caracteres que va a contener con:

- Número: #
- Carácter: ?

- 5.- Seleccionamos en el panel Código el evento Custom Creation Código e introducimos el siguiente código:

```
new JFormattedTextField(maskDNI);
```

6.3. Práctica

Abrir el proyecto **swing6** y cambiar la mascara DNI del JFormattedTextField por una mascara de entrada de matriculas formato: ####-???

Ver la figura 34 de la página 69

6.4. API

La siguiente tabla muestra alguno de los métodos mas utilizados:



Figura 6: swing6

Método	Propósito
String getText()	Devuelve el texto del componente
char[] getPassword()	Devuelve la contraseña en una vector de caracteres.
setPage(URL)	Carga un editor pane (o text pane) con el texto de la URL especificada.
setPage(String) (en JEditorPane)	
URL getPage() (en JEditorPane)	Obtiene la URL de la página actual de editor pane (o text pane).
void setEditable(boolean)	Selecciona u obtiene si el usuario puede editar el texto del componente de texto.
boolean isEditable()	

Para mas información del API completa:

- <http://download.oracle.com/javase/6/docs/api/javaw/swing/JTextField.html>
- <http://download.oracle.com/javase/6/docs/api/javaw/swing/JPasswordField.html>
- <http://download.oracle.com/javase/6/docs/api/javaw/swing/JFormattedTextField.html>
- <http://download.oracle.com/javase/6/docs/api/javaw/swing/JTextArea.html>
- <http://download.oracle.com/javase/6/docs/api/javaw/swing/JEditorPane.html>
- <http://download.oracle.com/javase/6/docs/api/javaw/swing/JTextPane.html>
- <http://docs.oracle.com/javase/1.4.2/docs/api/javaw/swing/text/MaskFormatter.html>

6.5. Jerarquía de clases

Jerarquía de clases para crear un JLabel:

Jerarquía de clases:

- └ javax.swing.JFrame
 - └ javax.swing.JPanel
 - └ javax.swing.JTextField

6.6. Eventos

Los eventos que podemos capturar a partir de estos componentes son los que muestra la pestaña de Eventos cuando seleccionamos alguno de ellos. El evento más común es el que detecta cuando hay un cambio en el componente del `JTextPane` está en `Eventos caretUpdate`.

Ver la figura 7 de la página 23

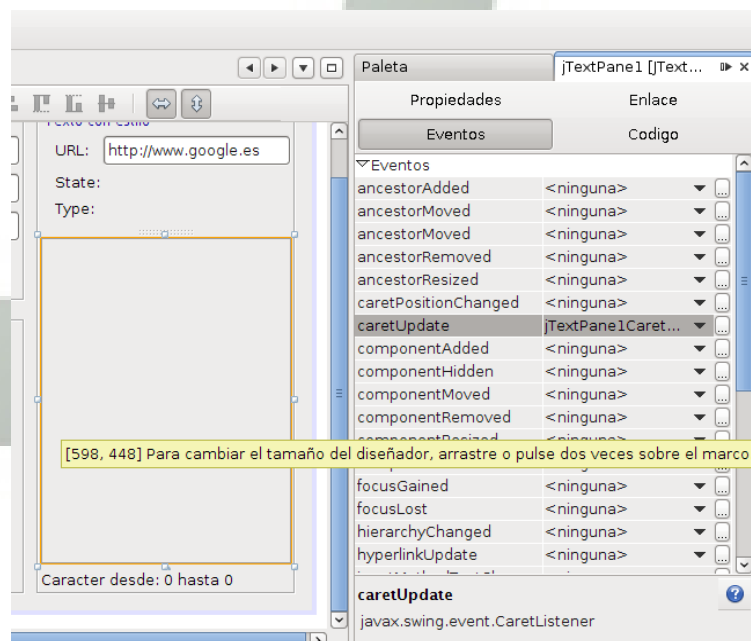


Figura 7: swing06textoformateado

6.6.1. Práctica

Siguiendo con el proyecto `swing6`, ver los métodos `getText()` y `getPassword()` en los componentes de texto a partir del evento del botón "Ver datos" (seleccionando el componentes botón derecha ratón eventos `actionPerformed`). Ver también el evento `caretUpdate` en el componente `jTextPane1`.

6.7. Verificar entradas.

Java dispone de la clase `InputVerifier` verificar entradas de tal forma que, hasta no se introduzca bien el dato correctamente, no cambie a otra componente.

Ejemplo:

El siguiente programa verifica que el valor introducido sea numérico.

`../prgcodigo/prgcod09/VerificarEntradas.java`

```
i»¿package prgt9e10aldarias;
import java.awt.Frame;
import java.awt.GridLayout;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import javax.swing.InputVerifier;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextField;

/*****/
public class VerificarEntradas extends JFrame {

    public VerificarEntradas() {
        JLabel jLabel1 = new JLabel("Numero:");
        JTextField jTextField1 = new JTextField("");
        JLabel jLabel2 = new JLabel("Texto:");
        JTextField jTextField2 = new JTextField("");

        add(jLabel1);
        add(jTextField1);
        add(jLabel2);
        add(jTextField2);
        setLayout(new GridLayout(2, 2));

        // Verificar la entrada.
        jTextField1.setInputVerifier(new Verificador());

    }

    public static void main(String[] args) {
        Frame f = new VerificarEntradas();
        f.pack();
        f.setSize(200, 100);
        f.setLocationRelativeTo(null);
        f.setVisible(true);
    }
}

/*****/
class Verificador extends InputVerifier {

    public boolean verify(JComponent input) {

        if (input instanceof JTextField) {
            String texto = ((JTextField) input).getText();
            try {
                // Si se puede convertir en entero, está bien
                Integer.parseInt(texto);
                return true;
            } catch (Exception e) {
                // Si no se ha podido convertir a entero, mostramos
                // una ventana de error y devolvemos false
                JOptionPane.showMessageDialog(input, "No es un número");
            }
        }
    }
}
```

```
        return false;
    }
    }
    return true;
}
}
```

Mediante la función `setInputVerifier`, podemos verificar el contenido.

```
jtextfield1.setInputVerifier(new Verificador());
```

7. LAYOUT

En las JFrames o los JPanel, podemos indicar como distribuir los botones usando la funcion `setLayout`.

Vamos a ver los siguiente:

- `FlowLayout`. Ajusta el flujo de los botones a la ventana.
- `GridLayout`. Permite indicar filas y columnas.
- `BorderLayout`. Permite indicar el punto cardinal.

7.1. FlowLayout

`FlowLayout` ajusta los elementos a la Ventana o Panel.

`../prgcodigo/prgcod09/MiFlowLayout.java`

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

public class MiFlowLayout extends JFrame {

    public MiFlowLayout() {

        Button button1 = new Button("Ok");
        Button button2 = new Button("Open");
        Button button3 = new Button("Close");
        add(button1);
        add(button2);
        add(button3);

        // Frame
        setLayout(new FlowLayout());
        setSize(100, 100);
        setLocationRelativeTo(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //pack();

    }

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

Ver la figura 8 de la página 26



Figura 8: MiFlowLayout

7.2. GridLayout

FlowLayout ajusta los elementos en filas (row) o columnas (columns).

../prgcodigo/prgcod09/MiGridLayout.java

```
public class GridLayoutDemo extends JFrame {

    private JLabel l1;
    private JLabel l2;
    private JLabel l3;
    private JLabel l4;
    private GridLayout gl;

    public GridLayoutDemo() {

        l1 = new JLabel("Dato1");
        l2 = new JLabel("Dato2");
        l3 = new JLabel("Dato3");
        l4 = new JLabel("Dato4");

        add(l1);
        add(l2);
        add(l3);
        add(l4);

        gl = new GridLayout();
        gl.setRows(2); // Filas
        gl.setHgap(10);
        gl.setColumns(2); // Columnas
        gl.setVgap(10);

        setLayout(gl);

        //algunas configuraciones de la ventana
        //pack();
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setLocationRelativeTo(null); // Centrar
        setSize(100, 100);
        setVisible(true);

    }

    public static void main(String[] argc) {
        new GridLayoutDemo();
    }

}
```

Ver la figura 9 de la página 27

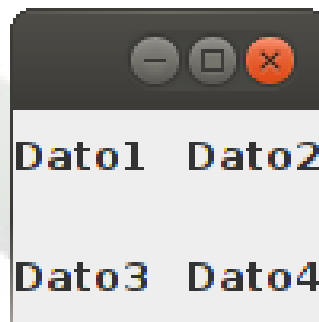


Figura 9: MiGridLayout

7.3. BorderLayout

Ajusta los elementos a los puntos cardinales de la Ventana o Panel.

../prgcodigo/prgcod09/MiBorderLayout.java

```

i»;public class BorderLayoutDemo extends JFrame {

    Button btnOeste = new Button();
    Button btnEste = new Button();
    Button btnNorte = new Button();
    Button btnSur = new Button();
    Button btnCentro = new Button();
    BorderLayout borderLayout1 = new BorderLayout();

    public BorderLayoutDemo() {

        this.setSize(new Dimension(336, 253));
        this.setLayout(borderLayout1);

        btnOeste.setLabel("Oeste");
        btnEste.setLabel("Este");
        btnNorte.setLabel("Norte");
        btnSur.setLabel("Sur");
        btnCentro.setLabel("Centro");

        //borderLayout1.setVgap(20);
        borderLayout1.setHgap(20);

        this.add(btnOeste, BorderLayout.WEST);
        this.add(btnEste, BorderLayout.EAST);
        this.add(btnNorte, BorderLayout.NORTH);
        this.add(btnSur, BorderLayout.SOUTH);
        this.add(btnCentro, BorderLayout.CENTER);

        //algunas configuraciones de la ventana
        this.pack();
        this.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        this.setLocationRelativeTo(null); // Centrar
        this.setVisible(true);

    }

    public static void main(String[] argc) {
        new BorderLayoutDemo();
    }

```

}

Ver la figura 10 de la página 28

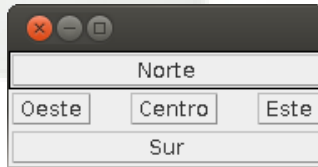


Figura 10: BorderLayout

7.4. Combinando

En el siguiente ejemplo vemos la combinación de FlowLayout y GridLayout.

../prgcodigo/prgcod09/GridLayoutDemo1.java

```
import java.awt.FlowLayout;
import java.awt.GridLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JPasswordField;
import javax.swing.JTextField;
import javax.swing.WindowConstants;

public class GridLayoutDemo1 extends JFrame {

    private JPanel p1;
    private JLabel l1;
    private JTextField t1;

    private JPanel p2;
    private JLabel l2;
    private JPasswordField t2;

    public GridLayoutDemo1() {

        GridLayout gl = new GridLayout();
        FlowLayout fl = new FlowLayout();

        p1 = new JPanel();
        p2 = new JPanel();

        gl.setRows(2);
        gl.setHgap(10);
        gl.setColumns(1);
        gl.setVgap(10);

        setLayout(gl);
        p1.setLayout(fl);
        p2.setLayout(fl);

        l1 = new JLabel("User");
        l2 = new JLabel("Pass");

        t1 = new JTextField("Paco");
        t1.setColumns(15);
```

```
t2 = new JPasswordField();
t2.setColumns(15);

add(p1);
add(p2);

p1.add(l1);
p1.add(t1);

p2.add(l2);
p2.add(t2);

// algunas configuraciones de la ventana
// pack();
setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
setLocationRelativeTo(null); // Centrar
setSize(300, 100);
setVisible(true);

}

public static void main(String[] argc) {
    new GridLayoutDemo1();
}

}
```

Ver la figura 11 de la página 29

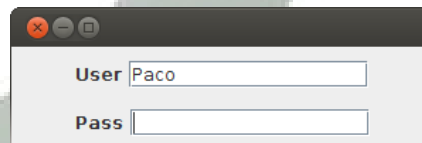


Figura 11: Combinando

8. ACTIONLISTENER

Los componentes java JButton, JTextField, permiten que nos suscribamos a eventos que pasan en ellos, de forma que cuando ocurre este evento, el componente nos avisa.

Por ejemplo, podemos estar interesados en cuándo se pulsa un botón, cuando un componente gana el foco, cuando pasa el ratón por encima, cuándo se cierra una ventana, etc, etc.

Para enterarnos de todos estos eventos, los componentes tienen métodos del estilo addXListener() donde la X, de alguna forma, representan el nombre del tipo de evento. Así, por ejemplo, los componentes pueden tener métodos addActionListener(), addMouseListener(), addWindowListener(), etc.

Lo que se comenta aquí valdrá para todos ellos, pero nos centraremos en el ActionListener.

Cuando usamos el `addActionListener()` de un componente, nos estamos suscribiendo a la "acción típica" o que java considera más importante para ese componente. Por ejemplo, la acción más típica de un `JButton` es pulsarlo. Para un `JTextField`, java considera que es pulsar `¡INTRO!` indicando que hemos terminado de escribir el texto, para un `JComboBox` es seleccionar una opción, etc.

Cuando llamamos al `addActionListener()` de cualquiera de estos componentes, nos estamos suscribiendo a la acción más "típica" de ese componente.

Como parámetro, debemos pasar una clase que implemente `ActionListener`. Por tanto, debemos hacer una clase que implemente la interface `ActionListener`. Hay muchas formas de hacer esto y vamos a verlas.

8.1. Una clase normalita

Una forma es hacer una clase en un fichero java que implemente `ActionListener`.

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

class ActionListener1a implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.exit(0); // Cierra la aplicación
    }
}
```

Luego, simplemente tenemos que añadir esta clase al componente java usando su método `addActionListener()`.

```
import javax.swing.JButton;
import javax.swing.JFrame;
public class ActionListener1 extends JFrame {
    private JButton b;
    public ActionListener1() {
        super("ActionListener1");
        b = new JButton("Pulsame");
        ActionListener1a elListener = new ActionListener1a();
        b.addActionListener(elListener);
        add(b);
        pack();
        setLocationRelativeTo(null); // Centrar
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Cierra Aplicacion
    }
    public static void main(String[] argc) {
        new ActionListener1().setVisible(true);
    }
}
```

Ya está, cuando se pulse el botón, java llamará a nuestro método `actionPerformed()` de nuestra clase `ActionListener1a`.

Este método no se suele usar casi nunca, puesto que es bastante engorroso hacerse una clase para cada uno de los componentes java que tengamos en la ventana que hagan algo.

Otra pega es que la clase es totalmente independiente, por lo que no tenemos acceso a los métodos de ninguna otra clase, salvo que hagamos algo para pasárselas y hacerselas accesibles.

8.2. Una clase interna

Otra forma es hacer una clase interna dentro de otra clase. Por ejemplo, si nuestra clase es una clase que hereda de `JFrame` y tiene botones, podemos hacerlo así

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;

public class ActionListener2 extends JFrame {
    private JButton b;
    public ActionListener2() {
        super("ActionListener2");
        b = new JButton("Pulsame");
        ActionListener2a elListener = new ActionListener2a();
        b.addActionListener(elListener);
        add(b);

        pack();
        setLocationRelativeTo(null); // Centrar
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Cierra Aplicacion
    }
    private void pulsado() {
        b.setText("Pulsado");
    }
    public static void main(String[] argc) {
        new ActionListener2().setVisible(true);
    }
    class ActionListener2a implements ActionListener {

        @Override
        public void actionPerformed(ActionEvent e) {
            pulsado();
        }
    }
}
```

```
}  
}
```

Este método sigue presentando el inconveniente de que hay que hacer una clase completa para cada componente java que tengamos en la ventana que queramos que haga algo.

La ventaja es que desde esta clase interna sí podemos acceder a métodos de la clase externa a la que pertenece. Como vemos en el ejemplo, desde ActionListener2a podemos llamar al método pulsado() de la clase ActionListener2.

8.3. Hacer que la clase principal implemente ActionListener

Otra forma que sí se usa a veces es hacer que la clase principal implemente ActionListener así

```
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import javax.swing.JButton;  
import javax.swing.JFrame;  
  
public class ActionListener3 extends JFrame implements ActionListener {  
    private JButton b;  
    public ActionListener3() {  
        super("ActionListener2");  
        b = new JButton("Pulsame");  
        b.addActionListener(this);  
        add(b);  
        pack();  
        setLocationRelativeTo(null); // Centrar  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Cierra Aplicacion  
    }  
    private void pulsado() {  
        b.setText("Pulsado");  
    }  
    public void actionPerformed(ActionEvent e) {  
        pulsado();  
    }  
    public static void main(String[] argc) {  
        new ActionListener2().setVisible(true);  
    }  
}}
```

Aquí ahorramos hacer una clase nueva y tenemos accesibles todos los métodos de la clase principal. Esta opción es muy cómoda si tenemos pocos componentes a los que añadir el listener.

8.4. Clase anónima

La opción que más se usa es la de hacer la clase "sobre la marcha". Es el sistema utilizado por matisse.

El código es este

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;

public class ActionListener4 extends JFrame {

    private JButton b;
    private int contador = 1;

    public ActionListener4() {
        super("ActionListener 4");
        b = new JButton("Pulsame 0");

        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                b.setText("Pulsado " + contador);
                contador++;
            }
        });

        add(b);

        pack();
        setLocationRelativeTo(null); // Centrar
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Cierra Aplicacion
    }

    public static void main(String[] argc) {
        new ActionListener4().setVisible(true);
    }
}
```

De esta forma, cuando compilemos se creará algo parecido a un fichero

ActionListener4\$1.class

Ese es el fichero de la clase anónima. Esa clase existe, pero no tenemos el fuente separado.

Aquí también tenemos acceso a todo lo de la clase ActionListener4. Incluso tenemos acceso a las variables locales y parámetros del método de la clase

principal donde hayamos añadido el listener. En este caso, a las variables locales y parámetros del constructor.

8.5. getSource

Podemos distinguir qué botón causa la acción con getSource.

Cuando nos decidimos a hacer una clase, suele ser habitual querer aprovechar la clase para varios botones.

En el método actionPerformed() de esa clase se nos pasa un ActionEvent. Con este ActionEvent podemos obtener información sobre quién es el que ha provocado el evento.

```
Object fuente = event.getSource();
```

Este método nos devuelve el componente java (JButton, JTextField, etc) que ha provocado el evento. Si la clase está añadida, por ejemplo a boton1, boton2 y boton3, en el método actionPerformed() podemos hacer algo como esto

```
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;

public class ActionListener5 extends JFrame implements ActionListener {

    private JButton b1;
    private JButton b2;

    public ActionListener5() {
        super("ActionListener5");

        setLayout(new FlowLayout());

        b1 = new JButton("Pulsame");
        b1.addActionListener(this);
        add(b1);

        b2 = new JButton("Salir");
        b2.addActionListener(this);
        add(b2);

        pack();
        setLocationRelativeTo(null); // Centrar
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Cierra Aplicacion
    }

    private void metodo1() {
```

```

        b1.setText("Pulsado");
    }

    private void metodo2() {
        System.exit(0); // Cierra la aplicación
    }

    public void actionPerformed(ActionEvent e) {
        Object fuente = e.getSource();

        if (fuente == b1) {
            metodo1();
        } else if (fuente == b2) {
            metodo2();
        }
    }

    public static void main(String[] argc) {
        new ActionListener5().setVisible(true);
    }
}

```

8.6. getActionCommand

Otra opción interesante, es decirle al botón cual es su "comando". Se hace así `boton.setActionCommand("Borra");` // Borra es un texto cualquiera, que NO se ve en el botón

Luego, en el `ActionEvent`, podemos obtener el "comando" del componente que ha provocado el evento.

```

import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;

public class ActionListener6 extends JFrame implements ActionListener {

    private JButton b1;
    private JButton b2;

    public ActionListener6() {
        super("ActionListener6");
    }
}

```

```
        setLayout(new FlowLayout());

        b1 = new JButton("Pulsame");
        b1.addActionListener(this);
        b1.setActionCommand("Pulsame");
        add(b1);

        b2 = new JButton("Salir");
        b2.addActionListener(this);
        b2.setActionCommand("Salir");
        add(b2);

        pack();
        setLocationRelativeTo(null); // Centrar
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Cierra Aplicacion
    }

    private void metodo1() {
        b1.setText("Pulsado");
    }

    private void metodo2() {
        System.exit(0); // Cierra la aplicación
    }

    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();

        if (command.equals("Pulsame")) {
            metodo1();
        } else if (command.equals("Salir")) {
            metodo2();
        }
    }

    public static void main(String[] argc) {
        new ActionListener6().setVisible(true);
    }
}
```

Este método tiene la ventaja, además de ser más claro, que permite que añadamos el mismo comando a varios botones o componentes y todos ellos harán lo mismo. Por ejemplo, imagina que tenemos un menú en el que una de las opciones es "Borra" y una barra de herramientas rápida en la que tenemos un

botón que también "Borra". Añadiendo el mismo ActionListener a ambos, menú y botón, la acción que se ejecutará es la misma.

Si hicieramos esto con el `e.getSource()`, deberíamos poner un `if` con un `OR`.

```
if (( e.getSource()==boton ) || (e.getSource()==opcionMenu ))  
...
```

9. EVENTOS

9.1. Pasos básicos para la creación de una IGU

En este apartado se muestran los pasos a seguir en la construcción de una aplicación con interfaz gráfica de usuario. Cada uno de los elementos de una interfaz gráfica de usuario en Java se representa por una instancia de una clase determinada. Por ejemplo, si se quiere que la interfaz de la aplicación tenga un botón, será necesario utilizar de la paleta de componentes el componente asociado, en este caso: `JButton (javax.swing.JButton)`.

Para configurar los aspectos gráficos de los componentes se utilizarán las propiedades del componente concreto. Por ejemplo, se podrá cambiar el texto del botón se utilizará la propiedad `text` (`void setText(String texto)`) del componente `JButton`.

Todos los componentes de Swing heredan de la clase `javax.swing.JComponent`, la cual hereda de `java.awt.Container`, y esta, a su vez, de `java.awt.Component`. Por ahora, saber cuales son las clases padre de todos los componentes permitirá ir conociendo algunas de sus características comunes. En la siguiente figura se muestra la Jerarquía de herencia del componente `JLabel`, que representa una etiqueta.

Ver la figura 12 de la página 38

Como puede verse, tanto el paquete `java.awt` como el paquete `javax.swing` contienen clases relativas a la interfaz de usuario, lo que hace patente la relación entre Swing y AWT. de la clase `javax.swing`.

La interfaz gráfica que se va a construir estará formada por una ventana ; dentro de ésta aparecerán: un botón, una etiqueta y un cuadro de texto.

Los pasos a seguir son los siguientes:

- 1.- Crear una ventana de aplicación.
- 2.- Añadir un contenedor.
- 3.- Añadir los componentes que se van en dicho contenedor.
- 4.- Cambiar las propiedades de visualización de los componentes.

La jerarquía de componentes de esta sencilla interface gráfica se puede ver la siguiente figura.

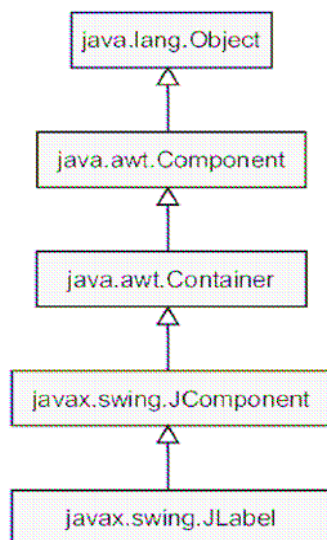


Figura 12: Herencias de JLabel

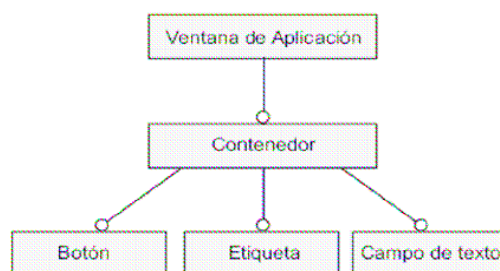


Figura 13: Elementos de la IGU a construir

Ver la figura 13 de la página 38

En ella se indica que un componente va a ser pintado dentro de un determinado contenedor con una línea terminada en círculo. También se muestra igualmente el contenedor que se pinta dentro de la ventana.

9.2. Creación de un proyecto con IGU

Iniciaremos NetBeans y crearemos un nuevo proyecto:

- 1.- **Archivo - Proyecto Nuevo.** También podemos acceder por la barra de herramientas en el icono.
- 2.- En el panel de Proyectos seleccionamos **Aplicación Java** y Botón Siguiente.
- 3.- Introducimos como nombre del proyecto **Swing1** y especificamos la ruta.

4.- Nos aseguraremos de que tenemos marcada la pestaña de **Hacer como Proyecto Principal** y *desmarcaremos* la pestaña de **Crear Clase Principal** si está seleccionada.

5.- Haremos clic en Terminar.

Ver la figura 14 de la página 39

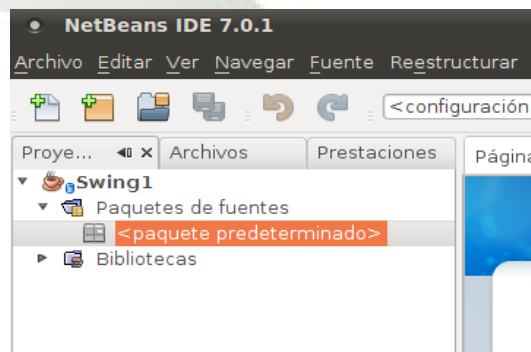


Figura 14: Proyecto IGU Creado

9.2.1. Creación de una ventana de aplicación

Para crear una ventana de aplicación hay que crear un contenedor JFrame (javax.swing.JFrame):

- Seleccionamos el proyecto Swing1 que hemos creado y con el Botón de la derecha del ratón seleccionamos el nodo Nuevo - Formulario JFrame.
- Añadimos InterfazSimple como nombre de la clase
- Añadimos swing1 como nombre del package (Un paquete es un contenedor de clases)
- Hacemos clic en Terminar.

Ahora tendremos una clase InterfazSimple.java en modo diseño para construir nuestra aplicación como se muestra en la siguiente figura.

Ver la figura 15 de la página 40

En este punto tenemos nuestro proyecto creado para iniciar nuestra aplicación, en el entorno podemos distinguir:

- Area de Diseño: : La principal ventana para crear y modificar la GUI.
- Inspector: Muestra la jerarquía de componentes de nuestra aplicación.
- Paleta: Lista de todos los componentes que podemos utilizar JFC/Swing, AWT, and JavaBeans
- Ventana de Propiedades: Muestra las propiedades del componente que hemos seleccionado

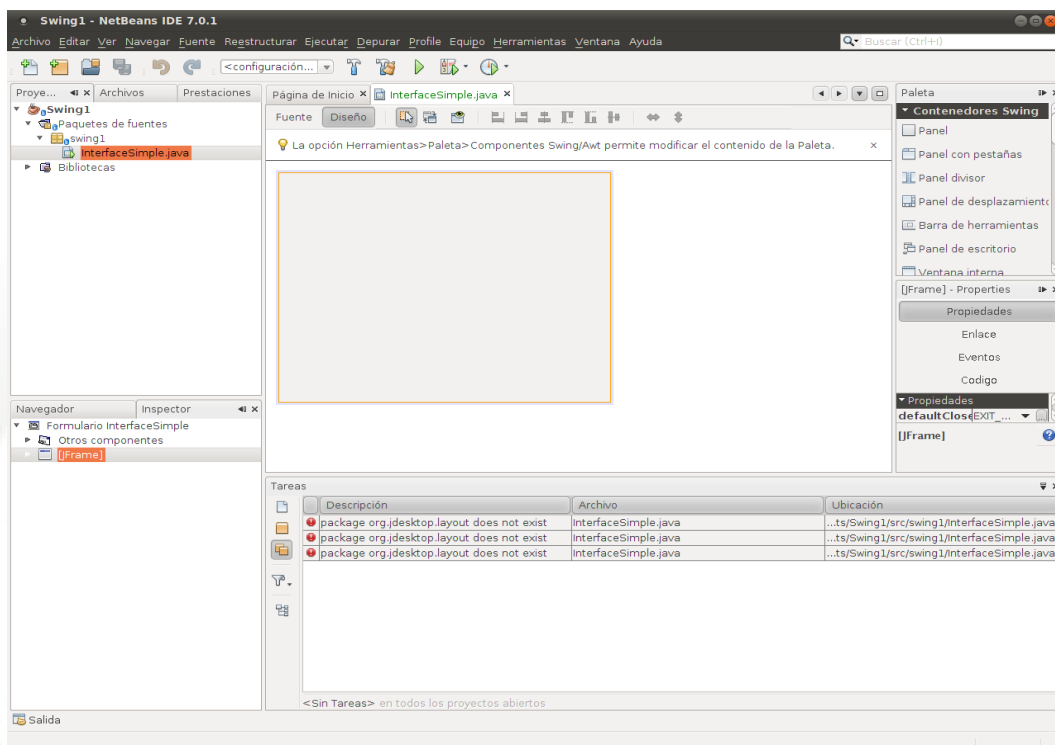


Figura 15: Crear JFrame

9.2.2. Creación de componentes una ventana de aplicación

Para utilizar un componente basta con ir al panel Paleta y hacer clic en el componente deseado.

9.2.3. Creación de un contenedor

En Swing existen muchos tipos de contenedores; sus diferencias radican en la forma en la que manejan los componentes que tienen dentro. Por ejemplo, el `JTabbedPane` (`javax.swing.JTabbedPane`) es un contenedor con pestañas donde cada una está asociada a un componente. También existe otro contenedor dividido en dos partes, para dos componentes, donde la separación puede cambiar de posición, es el `JSplitPane` (`javax.swing.JSplitPane`).

En este caso, para construir la interfaz propuesta se va a hacer uso del contenedor de propósito general `JPanel` (`javax.swing.JPanel`), que es el más sencillo de todos. Puede contener cualquier número de componentes en su interior, los cuales serán mostrados a la vez. La posición y tamaño de los componentes es configurable.

En el panel Paleta seleccionamos el componente `Panel` (`JPanel`).

Situamos el `Panel` dentro del contenedor `JFrame` en el área de Diseño y lo ajustamos a este

En el panel Inspector seleccionamos jPanel1 botón de la derecha Cambiar Nombre de Variable: jPanelGeneral

Ver la figura 16 de la página 41

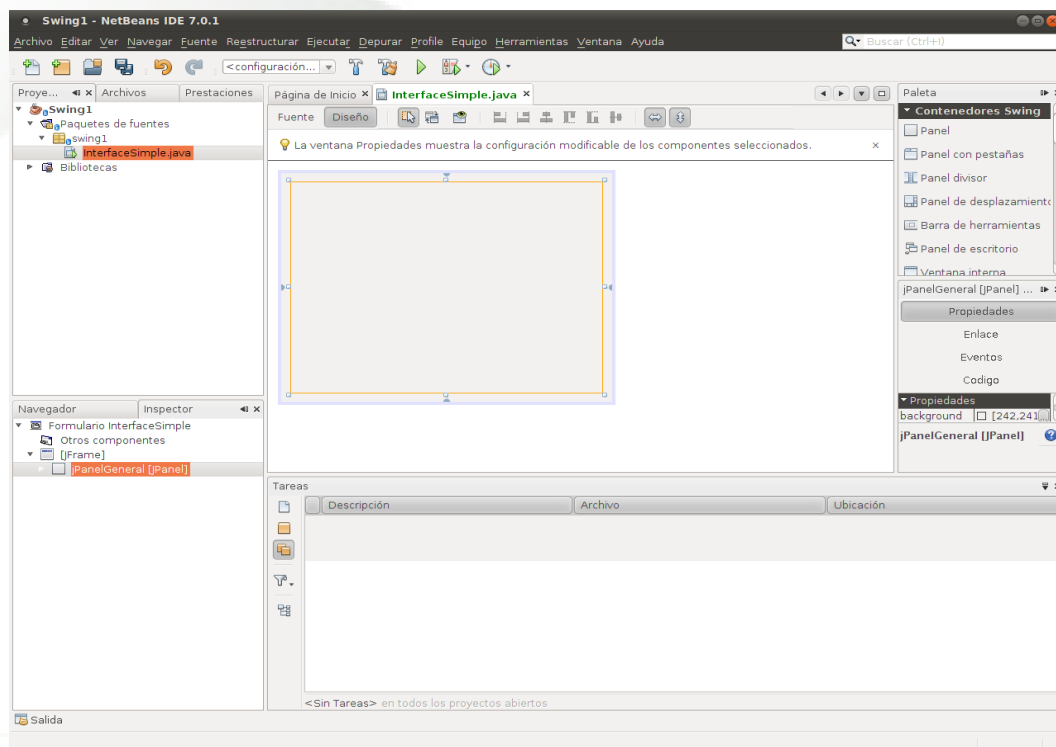


Figura 16: Crear JPanel

9.2.4. Asociación de componentes a un contenedor

Ahora sólo nos queda añadir los componentes botón (JButton), texto (JTextField) y etiqueta (JLabel), tal y como aparece en la siguiente figura:

Ver la figura 17 de la página 42

1.- Seleccionamos el componente **Etiqueta** o JLabel (javax.swing.JLabel) del panel Paleta, en el bloque de Controles Swing.

- Lo añadimos dentro del jPanelGeneral que hemos creado previamente
- Una vez añadido y posicionado en el área deseada, seleccionamos del panel inspector el nuevo componente creado JLabel1
- Botón derecha ratón Editar Texto Etiqueta
- Botón derecha ratón Cambiar Nombre de la Variable... - JLabelEtiqueta

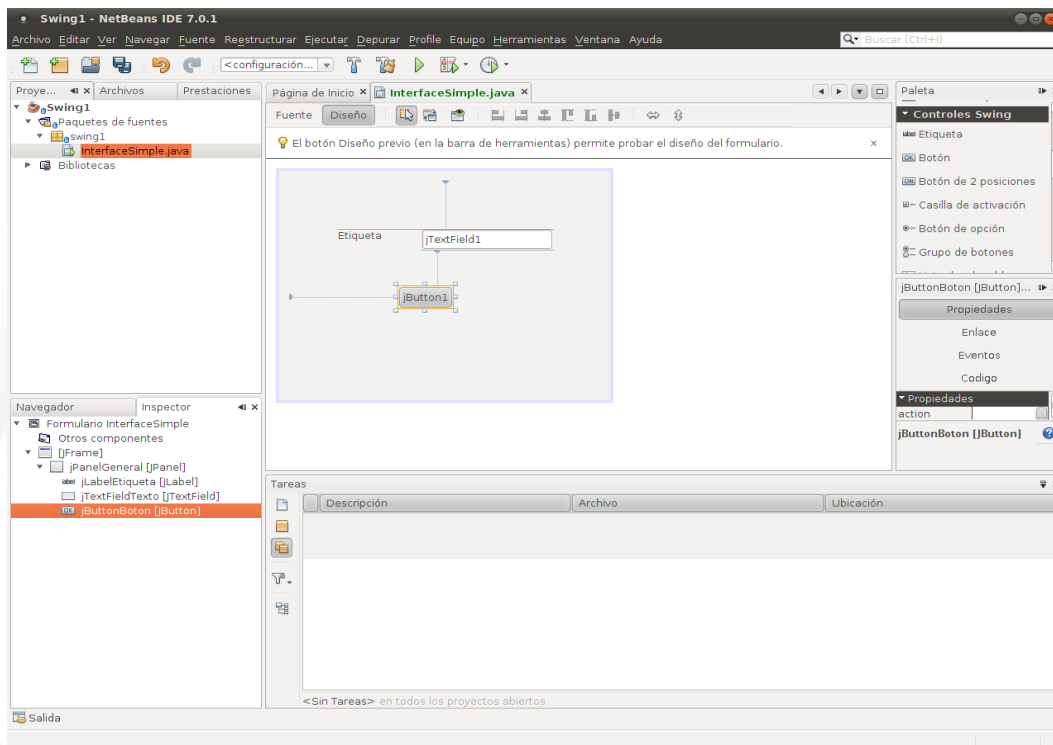


Figura 17: Asociar componentes

2.- Seleccionamos el componente **Campo de Texto** o JTextField (javax.swing.JTextField) del panel Paleta

- Lo añadimos dentro del jPanelGeneral que hemos creado previamente
- Una vez añadido y posicionado en el área deseada, seleccionamos del panel inspector el nuevo componente creado jTextField1
- Botón derecho ratón Cambiar Nombre Variable . . . : jTextFieldTexto

3.- Por último Seleccionamos el componente **Botón** o JButton (javax.swing.JButton) del panel Paleta

- Lo añadimos dentro del jPanelGeneral que hemos creado previamente
- Una vez añadido y posicionado en el área deseada, seleccionamos del panel inspector el nuevo componente creado jButton1
- Botón derecho ratón Editar Texto Botón
- Botón derecho ratón Cambiar Nombre Variable. . . jButtonBoton

Ya podemos ver como queda el aspecto de nuestra primera aplicación con swing. Ejecutamos F6 o icono Ejecutar

9.3. Eventos

Cuando se construye una aplicación con interfaz gráfica de usuario la forma de programar varía con respecto a una aplicación de consola. En una aplicación con interfaz gráfica se asocia código a diferentes eventos que puede producir el usuario cuando interactúa con la aplicación. De manera que, cuando se genera un evento (por ejemplo, pulsar un botón, pulsar una tecla del teclado, pasar el ratón sobre una imagen, etcétera), se ejecuta el código asociado y, habitualmente, se modifica la interfaz gráfica de usuario para mostrar el resultado de esa ejecución. Cuando finaliza la ejecución de ese código asociado, la aplicación espera nuevos eventos por parte del usuario.

Cada componente de la interfaz gráfica de usuario puede generar varios tipos de eventos. Por ejemplo, un componente de árbol genera eventos de un tipo cuando se selecciona alguno de sus nodos, eventos de otro tipo diferente cuando entra o sale el ratón en él, etc.

Se puede asociar un código a cada uno de los eventos que genera un componente. Es importante tener en cuenta que Swing permite asociar varios códigos distintos al mismo tipo de evento del mismo componente. En la siguiente figura se muestra un esquema indicando esta posibilidad; para ello, se representa mediante una flecha punteada la relación entre un componente que genera eventos y el código asociado.

Ver la figura 18 de la página 43

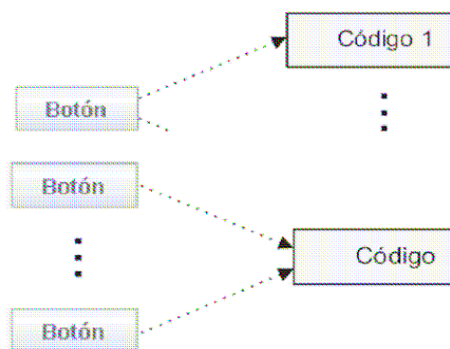


Figura 18: Asociar eventos

Por otra parte, también está permitido asociar un mismo código al mismo tipo de evento de varios componentes distintos, de forma que, un mismo código será ejecutado cuando se genere ese tipo de evento en cualquiera de los componentes a los que está asociado.

Por cada tipo de evento que puede ser generado en cualquier componente, existe una interfaz denominada `XXListener`, siendo `XX` el tipo de evento. Por ejemplo, cuando se interactúa de alguna forma con una ventana, se genera un evento de

tipo Window. Por tanto, existe una interfaz llamada `java.awt.event.WindowListener`. Todas estas interfaces heredan de la interfaz padre `java.util.EventListener`.

Estas interfaces tienen un método por cada una de las acciones concretas que provocan la generación del evento. Por ejemplo, la interfaz `WindowListener` tiene los métodos:

- `windowActivated(...)`
- `windowClosed(...)`
- `windowClosing(...)`
- `windowDeactivated(...)`
- `windowDeiconified(...)`
- `windowIconified(...)`
- `windowOpened(...)`

Estos eventos los podemos ver en NetBeans si seleccionamos el componente `JFrame` en el panel Inspector y la pestaña de Eventos.

Cada uno de estos métodos será ejecutado cuando se produzca dicha acción. Por ejemplo, cuando la ventana se abra, se ejecutará el método `windowOpened(...)`. Cada uno de los métodos de una interfaz `XXListener` tiene un único parámetro llamado `event` de la clase `XXEvent`. De esta forma, los métodos de la interfaz `WindowListener` tienen un único parámetro de la clase `java.awt.event.WindowEvent`. Todas las clases `XXEvent` disponen de métodos que ofrecen información acerca del evento producido. Por ejemplo, la clase `WindowEvent`, tiene el método `int getNewState()` que informa del estado en el que ha quedado la ventana después de producirse el evento. Las clases `XXEvent` heredan de `java.util.EventObject`. Esta clase tiene el método `Object getSource()` que devuelve una referencia al objeto donde se generó el evento.

Por último, cada componente que genere eventos dispone del método `void addXXListener(XXListener listener)`. Este método se utiliza para asociar el código de gestión de eventos del objeto `listener` al componente. Cuando en el componente se genere un evento de ese tipo, se ejecutará el método correspondiente en el `listener`. Para saber los eventos que se pueden generar en un componente determinado, basta con ver en el panel Events el conjunto de eventos.

En nuestro caso vamos a asociar al evento `actionPerformed` de componente `JButton` (Botón) que muestre un mensaje con el valor que contiene la caja de texto `.JTextFieldTexto`. Para ello realizaremos los siguientes pasos:

- 1.- Seleccionaremos del panel inspector el componente `JButton`
- 2.- Hacemos doble clic con el componente seleccionado o bien nos vamos al panel de events marcamos con el ratón el evento `ActionPerformed` y después con el ratón nos vamos al área de diseño y hacemos clic.
- 3.- Rellenamos el evento con el siguiente código:


```
JOptionPane.showMessageDialog(this, jTextFieldTexto.getText());  
//Mostramos el valor de la caja de texto
```

Ya podemos ver como queda el aspecto de nuestra primera aplicación con swing. Ejecutamos F6 o icono Ejecutar.

Ver la figura 19 de la página 45

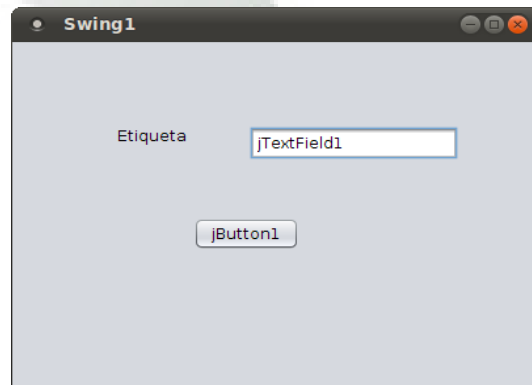


Figura 19: Swing1

Si hay algún error de que no encuentra la clase hacemos un Fix Import (Control + Mayúsculas + I) para que importe el paquete correspondiente.

9.4. Práctica

Realizar un proyecto llamado Conversor, que muestre una interfaz de usuario que nos permita realizar la conversión de Euros a Pesetas, tal y como se muestra en la figura adjunta.

Ver la figura 33 de la página 68



Figura 20: Conversor

Realizarlo con Netbeans y Matisse.

El código del evento asociado al botón es el siguiente (El nombre de los componentes JLabel y JTextField ha de ser el mismo):

```
try {
    jLabelImporte.setForeground(Color.BLACK);
    jLabelImporte.setText(Integer.toString(((int)
        (Double.parseDouble(jTextFieldImporte.getText())*166.386)))) ;
} catch (Exception e) {
    jLabelImporte.setText("-Error-");
    jLabelImporte.setForeground(Color.RED);
}
```

10. JBUTTON

Swing añade varios tipos de botones, todos los botones (javax.swing.JButton), cajas de selección (javax.swing.JCheckBox), botones de selección (javax.swing.JRadioButton) y cualquier opción de un menú deben derivar de AbstractButton. En la siguiente figura se muestran ejemplos de distintos tipos de botones.

Ver la figura 21 de la página 46

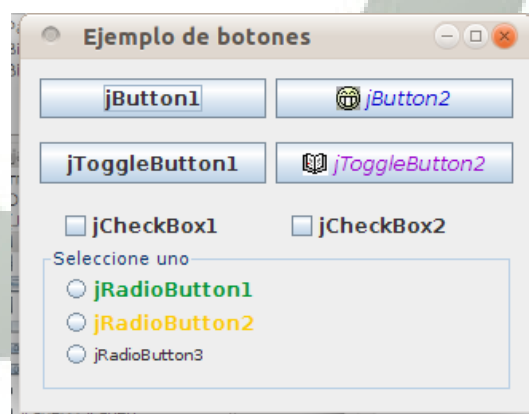


Figura 21: swing4

Algunas de las propiedades interesantes que podemos modificar desde el panel de propiedades para cada uno de estos componentes son: font, foreground, icon, horizontalAlignment, text, tooltipText.

Para añadir un grupo de botones hay que añadir el componente JButtonGroup y para cada componente que queramos que forme parte del grupo hay que modificar la propiedad buttonGroup y asociarla al grupo.

Para utilizar los componentes JButton, JCheckBox, JRadioButton, JButtonGroup accederemos al panel Paleta.

10.1. API

La siguiente tabla lista los métodos más utilizados:

Método	Propósito
String getText()	obtiene el texto mostrado por la etiqueta.
void setText(String)	Selecciona el texto mostrado por la etiqueta.
void setSelected(boolean)	Selección botón.
boolean isSelected()	Devuelve true si el botón está seleccionado. Tiene sentido en botones que tienen un estado on/off, como checkbox.

Para más información del API completa:

- <http://download.oracle.com/javase/6/docs/api/javax/swing/JButton.html>
- <http://download.oracle.com/javase/6/docs/api/javax/swing/JToggleButton.html>
- <http://download.oracle.com/javase/6/docs/api/javax/swing/JCheckBox.html>
- <http://download.oracle.com/javase/6/docs/api/javax/swing/JRadioButton.html>

10.2. Práctica

Abrir el proyecto swing4 y cambiar algunas de las propiedades que se han comentado anteriormente, ver el uso y manejo del evento actionPerformed.

10.3. Ejemplo: JButton con Eventos

Programa en Java que crea un texto, y dos botones. Al pulsar el botón Escribe escribe un texto y al pulsar el botón Borra borra el texto existente.

Los eventos que podemos capturar a partir de estos componentes son los que muestra la pestaña de Eventos cuando seleccionamos alguno de ellos. El evento más común es el que genera la acción de pulsado, ya sea con el ratón o bien con el teclado, Eventos Action actionPerformed.

Ver la figura 22 de la página 47

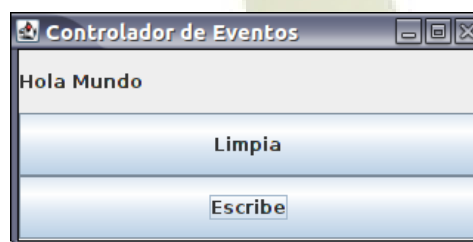


Figura 22: Eventos.java

../prgcodigo/prgcod09/Eventos.java

```
/** Controlador de evento
 * Crea dos botones uno que borra
 * y otro que escribe texto
 */

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

public class Eventos {

    private static JLabel label = new JLabel("---");
    private static JButton botonLimpia = new JButton("Limpia");
    private static JButton botonEscribe = new JButton("Escribe");

    public static void acciones (ActionEvent e) {
        Object obj = e.getSource();
        if ( obj == botonLimpia) {
            label.setText("");
        }
        if ( obj == botonEscribe) {
            label.setText("Hola Mundo");
        }
    }

    public static void main ( String[] args ) {
        try {
            UIManager.setLookAndFeel(
                UIManager.getCrossPlatformLookAndFeelClassName() );
        } catch (Exception e) {}
        JFrame frame = new JFrame ("Controlador de Eventos");
        botonLimpia.addActionListener ( new ActionListener() {
            public void actionPerformed (ActionEvent e) {
                acciones(e);
            }
        });
        botonEscribe.addActionListener ( new ActionListener() {
            public void actionPerformed (ActionEvent e) {
                acciones(e);
            }
        });
        frame.getContentPane().add(label);
        frame.getContentPane().add(botonLimpia);
        frame.getContentPane().add(botonEscribe);
        frame.addWindowListener ( new WindowAdapter() {
            public void windowClosing ( WindowEvent e) {
                System.exit(0);
            }
        });
        frame.setLayout(new GridLayout(0,1));
        frame.pack();
        frame.setVisible(true);
    }
}
```

Comentarios:

- Con UIManager indicamos el aspecto de la aplicación como multiplataforma.
- Con la función acciones indicamos que acciones se van a realizar según el botón que se pulse.

- Con `label.setText` ponemos el texto en el label.
- Con `BotonLimpia.addActionListener` definimos el listener para que realice la función acciones.
- Con `addWindowListener` definimos el listener del JFrame para que cuando se cierre la ventana, se termine el programa con `System.exit(0)`.

11. MVC. FORMULARIO

Vamos a ver como tener un formulario donde tenemos aplicado el MVC, Los datos mostrados se encuentran en un fichero cvs que esta en la carpeta principal del proyecto.

Consulta el ejemplo: `swing06bFormulario`

Ver la figura 23 de la página 49

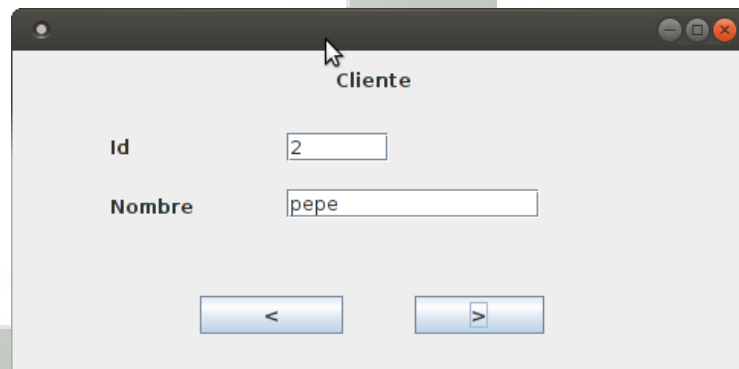


Figura 23: `swing06bFormulario`

1.- Clase Main.java

Podemos tener el main en una clase donde partimos el MVC

```
public class Main {
    public static void main(String args[]) {
        ModeloCliente m = new ModeloCliente();
        VistaPantalla v = new VistaPantalla();
        Controlador c = new Controlador(v, m);
    }
}
```

2.- Clase ModeloCliente.java

Definimos los datos del modelo de datos cliente.

```
class ModeloCliente {

    private String Id;
    private String Nombre;

    public ModeloCliente(String i, String n) {
```

```
        Id = i;
        Nombre = n;
    }
    ...
```

3.- Clase VistaPantalla.java

Utilizaremos Matisse para diseñarlo y crearemos métodos set y get que están al final de la clase para acceder a los objetos de la clase.

```
public class VistaPantalla extends javax.swing.JFrame {

    /**
     * Creates new form Vista
     */
    public VistaPantalla() {
        initComponents();
    }
    ...
    public JButton getjButtonAnterior() {
        return jButtonAnterior;
    }

    public JButton getjButtonSiguiente() {
        return jButtonSiguiente;
    }

    public void setjTextFieldId(String s) {
        jTextFieldId.setText(s);
    }

    public void setjTextFieldNombre(String s) {
        jTextFieldNombre.setText(s);
    }
}
```

4.- Clase Controlador.java

Implementará ActionListener para capturar los evento de la VistaPantalla

```
class Controlador implements ActionListener {

    VistaPantalla vista = null;
    ModeloCliente cliente = null;
    VistaFichero f = new VistaFichero();

    Controlador(VistaPantalla v, ModeloCliente m) {
        vista = v;
        cliente = m;

        vista.setVisible(true);
        primero();
        rellenavista(cliente);
        programaBotones();
    }

    private void programaBotones() {
        //Se añade las acciones a los controles del formulario vista
        vista.getjButtonAnterior().setActionCommand("Anterior");
        vista.getjButtonSiguiente().setActionCommand("Siguiente");
        //Se pone a escuchar las acciones del usuario
        vista.getjButtonAnterior().addActionListener(this);
        vista.getjButtonSiguiente().addActionListener(this);
    }
}
```

```
private void rellenavista(ModeloCliente cliente) {
    vista.setjTextFieldId(cliente.getId());
    vista.setjTextFieldNombre(cliente.getNombre());
}

@Override
public void actionPerformed(ActionEvent ae) {

    String comando = ae.getActionCommand();

    /* Acciones del formulario padre */
    switch (comando) {
        case "Anterior":
            anterior();
            rellenavista(cliente);
            break;
        case "Siguiente":
            siguiente();
            rellenavista(cliente);
            break;
    }
}

private void primero() {
    cliente = f.primer();
}

private void anterior() {
    int id;

    if (cliente == null) {
        return;
    }

    id = Integer.parseInt(cliente.getId());
    id--;

    if (id == 0) {
        return;
    }

    cliente = f.getClientes(String.valueOf(id));
}

private void siguiente() {
    int id;
    if (cliente == null) {
        return;
    }

    id = Integer.parseInt(cliente.getId());
    id++;
    cliente = f.getClientes(String.valueOf(id));
}
}
```

5.- Clase VistaFichero.java

Se encarga de leer el fichero csv

```
class VistaFichero {

    final String FICHERO = "swing06bFormulario.csv";

    ModeloCliente primero() {
```

```
File fs = new File(FICHERO);
ModeloCliente cliente = null;

if (fs.exists()) {
    try {
        StringTokenizer stringTokenizer;
        FileReader fr = new FileReader(fs);
        BufferedReader br = new BufferedReader(fr);
        String linea;

        linea = br.readLine();
        if (linea != null) {
            stringTokenizer = new StringTokenizer(linea, ";");

            String id;
            String nombre;

            id = stringTokenizer.nextToken();
            nombre = stringTokenizer.nextToken();

            cliente = new ModeloCliente(id, nombre);
        }

        if (fr != null) {
            fr.close();
        }

    } catch (IOException e) {
    }
} // if
return cliente;
}

ModeloCliente getCliente(String id_) {

    File fs = new File(FICHERO);
    ModeloCliente cliente = null;

    if (fs.exists()) {
        try {
            StringTokenizer stringTokenizer;
            FileReader fr = new FileReader(fs);
            BufferedReader br = new BufferedReader(fr);
            String linea;

            while ((linea = br.readLine()) != null) {
                stringTokenizer = new StringTokenizer(linea, ";");

                String id;
                String nombre;

                id = stringTokenizer.nextToken();
                nombre = stringTokenizer.nextToken();

                cliente = new ModeloCliente(id, nombre);
                if (id.equals(id_)) {
                    return cliente;
                }
            } // while

            if (fr != null) {
                fr.close();
            }
        }
    }
}
```



```
    } catch (IOException e) {  
    }  
    } // if  
    return cliente;  
  }  
}
```

12. JDIALOG

Muchas clases Swing soportan diálogos (ventanas que son más limitadas que los frames). Para crear un diálogo, simple y estándar se utiliza JOptionPane. Para crear diálogos personalizados, se utiliza directamente la clase JDialog.

El código para diálogos simples puede ser mínimo. Por ejemplo, aquí tenemos un diálogo informativo:

Ver la figura 24 de la página 53

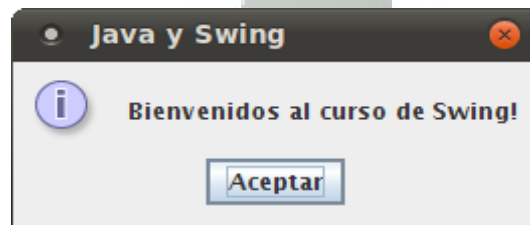


Figura 24: Swing10

El código para mostrar este cuadro de dialogo es el siguiente:

```
JOptionPane.showMessageDialog(this,  
"Bienvenidos al curso de Swing!", "Java y Swing",  
JOptionPane.INFORMATION_MESSAGE);
```

Hay diferentes tipos de diálogos:

1.- showMessageDialog

Muestra un diálogo modal con un botón, etiquetado "OK". Se puede especificar fácilmente el mensaje, el icono y el título que mostrará el diálogo.

Ejemplo:

```
JOptionPane.showMessageDialog(null, "El texto introducido es : "  
+ str, "Titulo Ventana", 1);
```

2.- showConfirmDialog

Muestra un diálogo modal con dos botones, etiquetados "Yes" y "No". Estas etiquetas no son siempre terriblemente descriptivas con las acciones específicas del programa que causan.

3.- showInputDialog

Muestra un diálogo modal que obtiene una cadena del usuario. Un diálogo de entrada muestra un campo de texto para que el usuario teclee en él, o un ComboBox no editable, desde el que el usuario puede elegir una de entre varias cadenas.

Ejemplo:

```
String str = JOptionPane.showInputDialog(null,
    "Introducir texto : ", "Titulo Ventana", 1);
```

4.- showOptionDialog

Muestra un diálogo modal con los botones, los iconos, el mensaje y el título especificado, etc. Con este método, podemos cambiar el texto que aparece en los botones de los diálogos estándar. También podemos realizar cualquier tipo de personalización.

Hay veces que nuestro dialogo debe de aparecer dentro de un JFrame, en este caso se añade la palabra Internal después de show: showInternalMessageDialog, showInternalConfirmDialog, showInternalInputDialog y showInternalOptionDialog.

12.1. API

El API detallada con los parámetros de cada uno de las opciones las podemos ver:

<http://download.oracle.com/javase/6/docs/api/javax/swing/JOptionPane.html>

12.2. Ejemplo:

El siguiente ejemplo crea un botón que al pulsarlo nos muestra un showInputDialog que nos pide un texto el cual luego nos los muestra con un showMessageDialog.

Utilizaremos el proyecto swing10a.

Ver la figura 25 de la página 55

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

public class ShowInputDialog{
    public static void main(String[] args){
        JFrame frame = new JFrame("swing10a");
        JButton button = new JButton("Show Input Dialog Box");
        button.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent ae){

                String str = JOptionPane.showInputDialog(null, "Enter some text : ",
                    "Titulo Ventana", 1);

                JOptionPane.showMessageDialog(null, "You entered the text : " + str,
                    "Titulo Ventana", 1);
            }
        });
    }
}
```

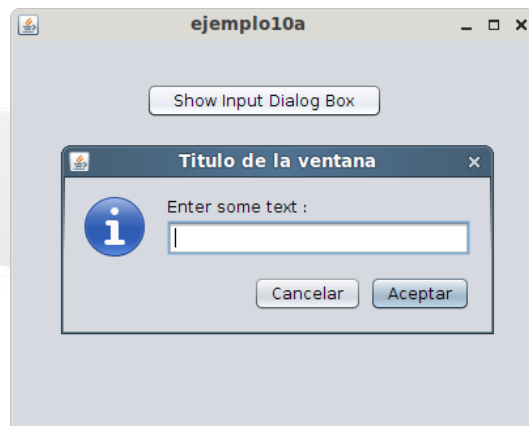


Figura 25: swing10a

```
}  
});  
  
JPanel panel = new JPanel();  
panel.add(button);  
frame.add(panel);  
frame.setSize(400, 400);  
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
frame.setVisible(true);  
}  
}
```

12.3. Práctica

Utilizando el proyecto swing10 crear un Dialogo que permita obtener una entrada en una caja de texto al usuario mediante `showInputDialog`.

Ver la figura 26 de la página 56

13. JFRAME Y JDIALOG

Usando Netbeans7.x se agregan en el nuevo proyecto dos ventanas. Referencia proyecto swing06aDosVentanas. La ventana principal tipo `JFrame` con un botón dentro y la ventana secundaria de tipo `JDialog`.

En este ejemplo la ventana principal queda inhabilitada hasta que no se cierre la ventana secundaria (`JDialog`). Las ventanas de tipo `JDialog` tienen la peculiaridad de que al abrirse bloquean el resto (por seguridad).

En netbeans7.x (por defecto) no aparece en la lista el paquete `JDialog Form`. Para ello, clic con botón izquierdo sobre icono paquete del proyecto y en el menú que aparece seguir el siguiente camino: - New - Other - Swing GUI FORMS - `JDialog form`.

El código del botón sería:

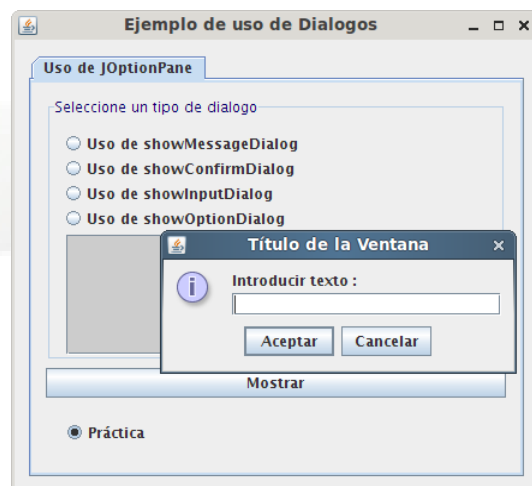


Figura 26: ejemplo10s

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
    new JDialogVentana2(this, true).setVisible(true);  
}
```

Ver la figura 27 de la página 56

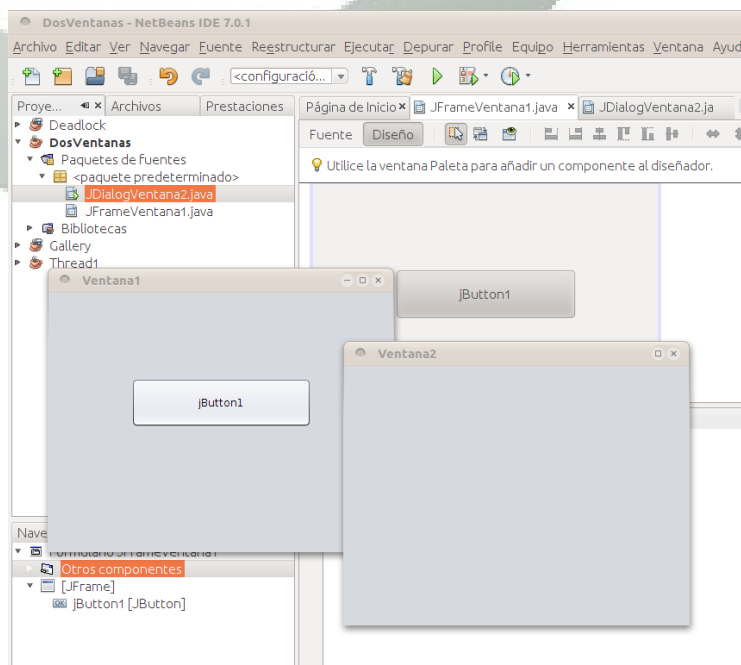


Figura 27: swing06aDosVentanas

13.0.1. Práctica

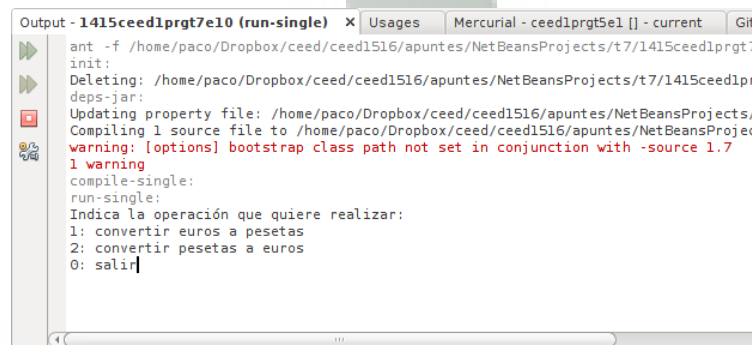
Tomando el ejemplo swing06aDosVentanas, añadir un botón nuevo y hacer que se abra una ventana nueva (ventana3).

14. MVC. APLICACION EUROS-PESETAS

El siguiente ejemplo explica como independizar una aplicación MVC las vistas. Es decir, podemos usar vistas diferentes pero el resto de la aplicación funcionará igual.

Basado en: Patrones de diseño orientado a objetos. Conversor Euro-Pesetas. <https://www.fdi.ucm.es/profesor/jpavon/poo/>

Tendremos la aplicación con vista en terminal. Ver la figura 28 de la página 57



```
Output - 1415ceed1prgt7e10 (run-single) x Usages Mercurial - ceed1prgt5e1 [] - current Git
ant -f /home/paco/Dropbox/ceed/ceed1516/apuntes/NetBeansProjects/t7/1415ceed1prgt;
init:
Deleting: /home/paco/Dropbox/ceed/ceed1516/apuntes/NetBeansProjects/t7/1415ceed1pr
deps-jar:
Updating property file: /home/paco/Dropbox/ceed/ceed1516/apuntes/NetBeansProjects/
Compiling 1 source file to /home/paco/Dropbox/ceed/ceed1516/apuntes/NetBeansProje
warning: [options] bootstrap class path not set in conjunction with -source 1.7
1 warning
compile-single:
run-single:
Indica la operación que quiere realizar:
1: convertir euros a pesetas
2: convertir pesetas a euros
0: salir
2
```

Figura 28: europesetasterminal

Tendremos la aplicación con vista igu. Ver la figura 29 de la página 57

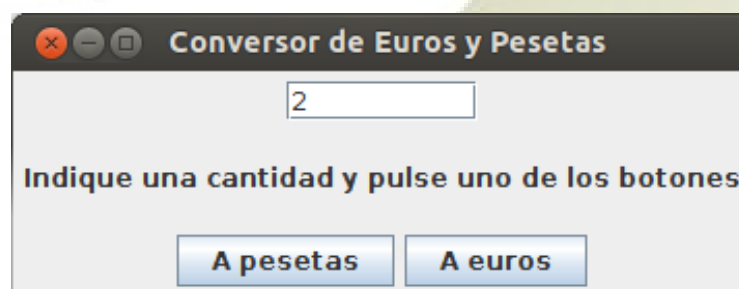


Figura 29: europesetasigu

14.1. Controlador

Main.java

```
package pkg1415ceed1prgt7e10.mvc;
/**
```

```
* Fichero: Main.java
* @date 16-feb-2014
* @author Paco Aldarias <paco.aldarias@ceedcv.es>
*/
```

```
public class Main {

    public static void main(String[] args) {
        // el modelo:
        ModeloEurosPesetas modelo = new ModeloEurosPesetas();
        // la vista:
        //Vista vista = new VistaTexto();
        Vista vista = new VistaGrafica();
        // y el control:
        Controlador control = new Controlador(vista,
            modelo);
        // configura la vista
        vista.setControlador(control);
        // y arranca la interfaz (vista):
        vista.arranca();
    }
}
```

Controlador.java

```
package pkg1415ceed1prgt7e10.mvc;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Controlador implements ActionListener {

    private Vista vista;
    private ModeloEurosPesetas modelo;

    public Controlador(Vista vista, ModeloEurosPesetas modelo) {
        this.vista = vista;
        this.modelo = modelo;
    }

    public void actionPerformed(ActionEvent evento) {

        double cantidad = vista.getCantidad();

        if (evento.getActionCommand().equals(Vista.AEUROS)) {
            vista.escribeCambio(cantidad + " pesetas son: "
                + modelo.pesetasAeuros(cantidad) + " euros");
        } else if (evento.getActionCommand().equals(Vista.APESETAS)) {
            vista.escribeCambio(cantidad
                + " euros son: " + modelo.eurosApesetas(cantidad) + " pesetas");
        } else {
            vista.escribeCambio("ERROR");
        }
    }
}
```

14.2. Modelo

ModeloEuros.java

```
package pkg1415ceed1prgt7e10.mvc;

class ModeloEuros {
```

```
private double cambio;

public ModeloEuros(double valorCambio) {
    // valor en la moneda de 1 euro
    cambio = valorCambio;
}

public double eurosAmoneda(double cantidad) {
    return cantidad * cambio;
}

public double monedaAeuros(double cantidad) {
    return cantidad / cambio;
}
}
```

ModeloEurosPesetas.java

```
package pkg1415ceed1prgt7e10.mvc;

public class ModeloEurosPesetas extends ModeloEuros { // Adaptador de clase

    public ModeloEurosPesetas() {
        super(166.386D);
    }

    public double eurosApesetas(double cantidad) {
        return eurosAmoneda(cantidad);
    }

    public double pesetasAeuros(double cantidad) {
        return monedaAeuros(cantidad);
    }
}
```

14.3. Vista

Vista.java

```
package pkg1415ceed1prgt7e10.mvc;

public interface Vista {

    void setControlador(Controlador c);

    void arranca();
    // comienza la visualización

    double getCantidad();
    // cantidad a convertir

    void escribeCambio(String s); //texto con la conversión
    // Constantes que definen las posibles operaciones:
    static final String AEUROS = "Pesetas a Euros";
    static final String APESETAS = "Euros a Pesetas";
}
```

VistaTexto.java

```
package pkg1415ceed1prgt7e10.mvc;
```

```
import java.awt.event.ActionEvent;
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class VistaTexto implements Vista {

    private Controlador controlador;
    // Gestión de la entrada por teclado
    private BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

    private int leeOpción() {
        String s = null;
        try {
            s = in.readLine();
            return Integer.parseInt(s);
        } catch (Exception e) {
            operaciónIncorrecta();
            return 0;
        }
    }

    private double leeCantidad() {
        String s = null;
        try {
            s = in.readLine();
            return Double.parseDouble(s);
        } catch (Exception e) {
            System.out.println("Error en formato del número, tiene que ser 99.99: ");
            return leeCantidad();
        }
    }

    private void solicitaOperación() {
        System.out.println("Indica la operación que quiere realizar:");
        System.out.println("1: convertir euros a pesetas");
        System.out.println("2: convertir pesetas a euros");
        System.out.println("0: salir");
    }

    private void procesaNuevaOperacion() {
        int operacion;
        solicitaOperación();
        operacion = leeOpción();
        if (operacion == 0) {
            System.out.println("Adiós.");
            System.exit(0);
        }
        if (operacion == 1) {
            controlador.actionPerformed(new ActionEvent(this, operacion, AEUROS));
        }
        if (operacion == 2) {
            controlador.actionPerformed(new ActionEvent(this, operacion, APESETAS));
        }
        operaciónIncorrecta();
    }

    private void operaciónIncorrecta() {
        System.out.print("Operación incorrecta. ");
        procesaNuevaOperacion();
    }
}

// Métodos de la interfaz de la Vista:

public void setControlador(Controlador c) {
    controlador = c;
}

public void arranca() {
    procesaNuevaOperacion();
}
```



```

    }

    public void escribeCambio(String s) {
// escribe el resultado:
        System.out.println(s);
// y vuelve a solicitar al usuario una operación:
        procesaNuevaOperacion();
    }

    public double getCantidad() {
        System.out.print("Cantidad a convertir (formato 99.99): ");
        return leeCantidad();
    }
}

```

VistaGrafica.java

```

package pkg1415ceed1prgt7e10.mvc;

import java.awt.BorderLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class VistaGrafica extends JFrame implements Vista {

    private JButton convertirApesetas;
    private JButton convertirAeuros;
    private JTextField cantidad;
    private JLabel resultado;

    public VistaGrafica() {
        /* Se crear un panel principal panelPrincipal
        * donde se añaden otros paneles:
        * panelaux al Norte
        * panelaux2 al centro
        * botonera al sur
        */

        super("Conversor de Euros y Pesetas");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JPanel panelPrincipal = new JPanel();
        panelPrincipal.setLayout(new BorderLayout(10, 10));
        cantidad = new JTextField(8);
        JPanel panelaux = new JPanel();
        panelaux.add(cantidad);
        panelPrincipal.add(panelaux, BorderLayout.NORTH);
        resultado = new JLabel("Indique una cantidad y pulse uno de los botones");
        JPanel panelaux2 = new JPanel();
        panelaux2.add(resultado);
        panelPrincipal.add(panelaux2, BorderLayout.CENTER);
        convertirApesetas = new JButton("A pesetas");
        convertirApesetas.setActionCommand(APESETAS);
        convertirAeuros = new JButton("A euros");
        convertirAeuros.setActionCommand(AEUROS);
        JPanel botonera = new JPanel();
        botonera.add(convertirApesetas);
        botonera.add(convertirAeuros);
        panelPrincipal.add(botonera, BorderLayout.SOUTH);
        getContentPane().add(panelPrincipal);
    }

    // Métodos de la interfaz Vista:
    public void escribeCambio(String s) {
        resultado.setText(s);
    }
}

```

```
}

public double getCantidad() {
    try {
        return Double.parseDouble(cantidad.getText());
    } catch (NumberFormatException e) {
        return 0.0D;
    }
}

public void setControlador(Controlador c) {
    convertirApesetas.addActionListener(c);
    convertirAeueros.addActionListener(c);
}

public void arranca() {
    pack();// coloca los componentes
    setLocationRelativeTo(null);// centra la ventana en la pantalla
    setVisible(true);// visualiza la ventana
}
}
```

Discusión final

- ¿Qué hay que cambiar en el modelo y el control para utilizar la vista textual en vez de la gráfica?

Respuesta: Nada

- ¿Qué hay que cambiar en el programa principal? Respuesta:

En el main() hay que cambiar Vista vista = new VistaGrafica(); por Vista vista = new InterfazTextualConversor();

15. JLIST Y JCOMBOBOX.

Un componente JComboBox (javax.swing.JComboBox) o JList (javax.swing.JList) le permite al usuario seleccionar una opción o un conjunto de opciones de una lista. Para el JComboBox la opción por defecto es no editable.

Ver la figura 30 de la página 63

Si tuviéramos un JComboBox editable, el comportamiento sería análogo a una caja de texto mas el propio del JComboBox.

Las propiedades más comunes que podemos ver en la pestaña de Propiedades son: font, editable, maximumRowCount, toolTipText, selectionMode, layoutOrientation.

JCombobox tiene las siguientes funciones:

- JComboBox(). Constructor.
- void addItem(E item). Añade un item.
- Object getSelectedItem(). Devuelve el item seleccionado.
- void setSelectedIndex(int anIndex). Selecciona un item.

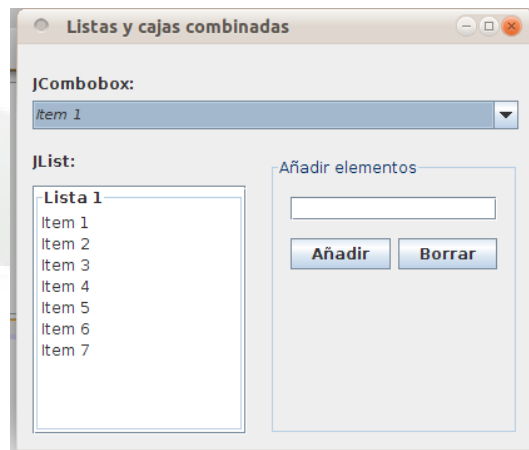


Figura 30: swing5

- void removeAllItems(). Borra todos los items.
- Object[] getSelectedObjects(). Devuelve un vector con todos los items.
- void setEnabled(boolean b). Activa el jcombobox

15.1. JComboBoxEjemplo1

En el siguiente ejemplo vamos a ver como se puede crear un JComboBox y gestionar los datos.

Ver la figura 31 de la página 64

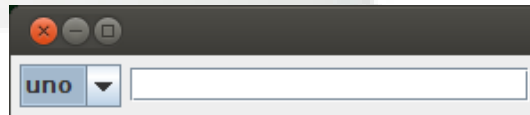


Figura 31: JComboBoxEjemplo1

```
../NetBeansProjects/t7/1415ceed1prgt7e10/src/JComboBoxEjemplo1.  
java
```

```
import java.awt.FlowLayout;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import javax.swing.JComboBox;  
import javax.swing.JFrame;  
import javax.swing.JTextField;  
import javax.swing.WindowConstants;  
  
public class JComboBoxEjemplo1 {  
  
    private JTextField tf;  
    private JComboBox combo;  
    private JFrame v;  
  
    public JComboBoxEjemplo1() {  
        // Creacion del JTextField  
        tf = new JTextField(20);  
  
        // Creacion del JComboBox y anyadir los items.  
        combo = new JComboBox();  
        combo.addItem("uno");  
        combo.addItem("dos");  
        combo.addItem("tres");  
  
        // Accion a realizar cuando el JComboBox cambia de item seleccionado.  
        combo.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent e) {  
                tf.setText(combo.getSelectedItem().toString());  
            }  
        });  
  
        // Creacion de la ventana con los componentes  
        v = new JFrame();  
        v.getContentPane().setLayout(new FlowLayout());  
        v.getContentPane().add(combo);  
        v.getContentPane().add(tf);  
        v.pack();  
        v.setLocationRelativeTo(null); // Centrar  
        v.setVisible(true);  
        v.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);  
    }  
  
    public static void main(String[] args) {  
        new JComboBoxEjemplo1();  
    }  
}
```

}

15.2. JComboboxEjemplo2

En el siguiente ejemplo vamos a ver como se puede crear un JComobox usando un arraylist, además veremos con hacer que un elemento sea seleccionado.

Ver la figura 32 de la página 65

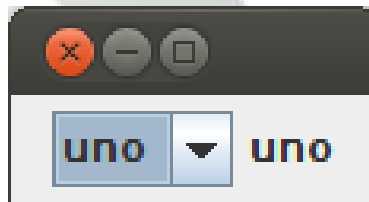


Figura 32: JComboboxEjemplo2

```
../NetBeansProjects/t7/1415ceed1prgt7e10/src/JComBoXEjemplo2.  
java
```

```
import java.awt.FlowLayout;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.util.ArrayList;  
import javax.swing.JComboBox;  
import javax.swing.JFrame;  
import javax.swing.JLabel;  
import javax.swing.WindowConstants;  
  
public class JComBoXEjemplo2 {  
  
    private JLabel jl;  
    private JComboBox jc;  
    private JFrame jf;  
  
    public JComBoXEjemplo2() {  
        // Creacion del JTextField  
        jl = new JLabel(" ");  
        ArrayList items = new ArrayList();  
        items.add("uno");  
        items.add("dos");  
        items.add("tres");  
  
        // Creacion del JComboBox y anyadir los items mediante arraylist  
        jc = new JComboBox();  
        for (int i = 0; i < items.size(); i++) {  
            jc.addItem(items.get(i));  
            if (items.get(i).equals("dos")) {  
                jc.setSelectedItem(items.get(i));  
            }  
        }  
  
        // Accion a realizar cuando el JComboBox cambia de item seleccionado.  
        jc.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent e) {  
                jl.setText(jc.getSelectedItem().toString());  
            }  
        })  
    }  
}
```

```

    });

    // Creacion de la ventana con los componentes
    jf = new JFrame();
    jf.setLayout(new FlowLayout());
    jf.add(jc);
    jf.add(jl);
    jf.pack();
    jf.setLocationRelativeTo(null); // Centrar
    jf.setVisible(true);
    jf.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
}

public static void main(String[] args) {
    new JComboBoxEjemplo2();
}
}

```

15.3. Práctica

Sobre el proyecto swing5, probad a modificar las propiedades `maximunRowCount` (en el `JComboBox`), `selectionMode` y `LayoutOrientation` (en el `JList`) y ver qué efectos tiene.

15.4. API

La siguiente tabla muestra alguno de los métodos mas utilizados:

Método	Propósito
<code>void addItem(Object)</code>	Añade o inserta un ítem en la lista.
<code>void insertItemAt(Object, int)</code>	Añade o inserta un ítem en la lista.
<code>Object getItemAt(int)</code>	Obtiene un ítem de la lista.
<code>Object getSelectedItem()</code>	Devuelve el ítem seleccionado
<code>void removeAllItems()</code>	Borra todos los ítems de la lista
<code>void removeItemAt(int)</code>	Elimina uno ítems de la lista.
<code>void removeItem(Object)</code>	Elimina uno o más ítems de la lista.
<code>int getItemCount()</code>	Obtiene el número de ítems de la lista.

Para más información del API completa:

- <http://download.oracle.com/javase/6/docs/api/javaw/swing/JComboBox.html>
- <http://download.oracle.com/javase/6/docs/api/javaw/swing/JList.html>

15.5. Eventos

Los eventos que podemos capturar a partir de estos componentes son los que muestra la pestaña de Eventos cuando seleccionamos alguno de ellos.

El evento más común en:

- JComboBox: Menú: Events - Action - **ActionPerformed**.
- JList: Menú: Events - ListSelection - **ValueChanged**.

15.5.1. Práctica

Siguiendo con el proyecto swing5, ver el uso de los métodos `getSelectedIndex()`, `removeItemAt(int index)`, `addItem(Object)`, que se encuentran en los eventos `actionPerformed` de cada uno de los botones (seleccionando el componente en el área de diseño botón derecha ratón events `actionPerformed`). Ver los eventos que se producen también en los componentes `JList` (Events `ListSelection valueChanged`) y `JComboBox` (Events `Action actionPerformed`) y el uso de los métodos `getSelectedValues()` y `addItem()` respectivamente.

15.5.2. Eventos

Los eventos que podemos capturar a partir de estos componentes, son los que muestra la pestaña de eventos cuando seleccionamos alguno de ellos. El evento más común es el que detecta cuando hay un cambio en el componente `Eventos-Acciones- actionPerformed` o bien mediante doble clic en el panel Inspector sobre el componente seleccionado.

15.5.3. Práctica

Añadir eventos para las opciones de Añadir, Listar, Matricular y Salir. Para las opciones Añadir, Matricular y Listar rellenar el evento con el siguiente código:

```
JOptionPane.showMessageDialog(this,"clic en "+  
((JMenuItem)evt.getSource()).getText());
```

Notar que a partir del evento `actionPerformed` sobre un componente podemos acceder al objeto que provocó ese evento mediante `evt.getSource()`, que en este caso los componentes son `JMenuItems`.

Para la opción salir añadimos:

```
System.exit(0);
```

16. EJERCICIOS PROPUESTOS

Realiza las actividades y entrega todos los archivos de las actividades obligatorias en el cvs. Subiendo al aula virtual la dirección para clonar, respetando el nombre de la práctica.

Realizar los siguientes ejercicios propuestos con Netbeans y entregar los proyectos, y una captura de pantalla del escritorio.

16.1. Ejercicio0901. Conversor

Realizar un proyecto llamado Conversor, que muestre una interfaz de usuario que nos permita realizar la conversión de Euros a Pesetas, tal y como se muestra en la figura adjunta.

Ver la figura 33 de la página 68



Figura 33: Conversor

Realizarlo con Netbeans y Matisse.

El código del evento asociado al botón es el siguiente (El nombre de los componentes JLabel y JTextField ha de ser el mismo):

```
try {
    jLabelImporte.setForeground(Color.BLACK);
    jLabelImporte.setText(Integer.toString(((int)
        (Double.parseDouble(jTextFieldImporte.getText())*166.386)))) ;
} catch (Exception e) {
    jLabelImporte.setText("-Error-");
    jLabelImporte.setForeground(Color.RED);
}
```


16.2. Ejercicio0902 DNI

Abrir el proyecto **swing6** y cambiar la mascara DNI del JFormattedTextField por una mascara de entrada de matriculas formato: ####-???

Ver la figura 34 de la página 69

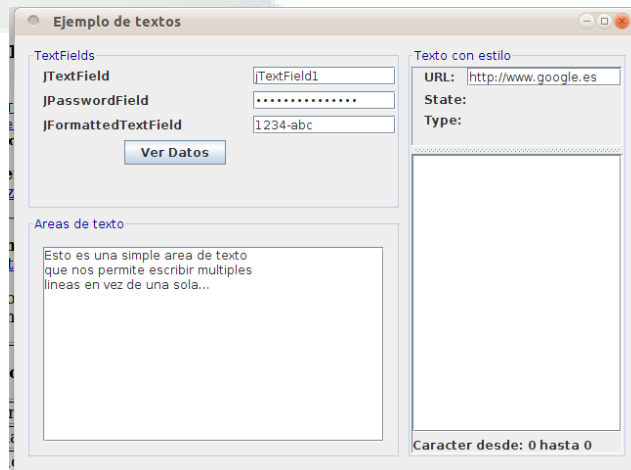


Figura 34: swing6

16.3. Ejercicio0903 Modelo Compartido

Utilizando la base del proyecto **swing15**, y utilizando el modelo SimpleTable-Model, crear dos JTables que compartan información, de manera que, cuando en una de ellas se modifique algo automáticamente cambie en la otra. Tal y como se muestra en la figura adjunta. Implementar los eventos Añadir y Borrar fila.

Ver la figura 61 de la página 98



Figura 35: swing15

16.4. Ejercicio0904 Calculadora

Realiza una calculadora básica con netbeans. Consultar videos de youtube.

Hay varios vídeos sobre esto. Yo recomiendo este:

Calculadora en java 7 con Netbeans - Diseño y programación

<http://www.youtube.com/watch?v=PkHqUY82qU8>

16.5. Ejercicio0905 InputVerifier

Realizar un programa en java que tenga como interface gráfica de usuario un JFrame que llamaremos Ejercicio0905InputVerifier con un campo para introducir un número y otro para introducir un email.

Debe comprobar con InputVerifier que se introduce obligatoriamente un valor y además sea correcto. Si el dato no es correcto no deberá permitir cambiar de campo.

Crear una clase que llamaremos VerificadorEntrada que tenga un función que devuelve cierta o falsa según se cumpla la condición del dato analizado.

Ver la figura 36 de la página 70

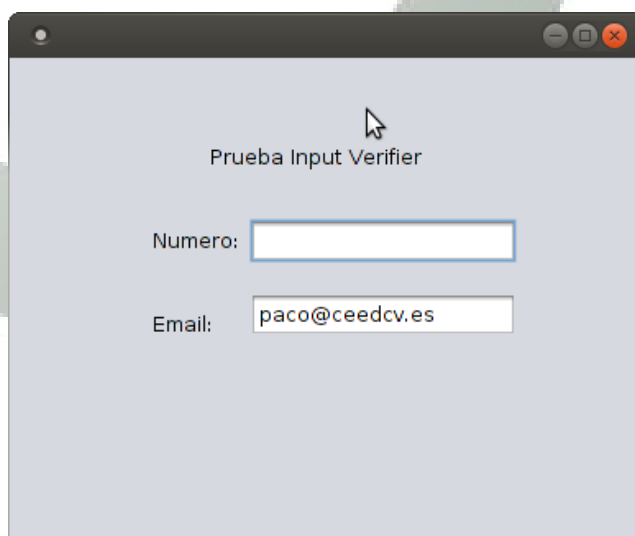


Figura 36: Ejercicio0905 InputVerifier

17. JPANEL

17.1. Definición

JPanel es un contenedor de propósito general para componentes de peso ligero. Como todos los contenedores, utiliza un Controlador de Distribución

para posicionar y dimensionar sus componentes (en nuestro caso el control de distribución que utilizaremos es el de Matisse FreeDesign).

Como todos los componentes Swing, JPanel permite añadirle bordes y determinar si utiliza el doble buffer para aumentar el rendimiento.

1.- JPanel.

Es sólo una de las varias clases de contenedores que se pueden utilizar. Existen algunos contenedores de propósito especial que podríamos utilizar en lugar de un JPanel.

2.- JLayeredPane

Proporciona una tercera dimensión, profundidad, para posicionar componentes. Los paneles con capas no tienen controladores de distribución pero pueden ser utilizados para colocar los componentes en capas en un JPanel. Un tipo de layeredpane, JDesktopPane, está diseñado específicamente para manejar frames internos.

3.- JScrollPane.

Proporciona una vista desplazable de un componente grande.

4.- JSplitPane

Muestra dos componentes cuyos tamaños relativos pueden ser modificados por el usuario.

5.- JTabbedPane

Permite a varios componentes, normalmente objetos JPanel, compartir el mismo espacio.

17.2. Creado varios tipos de paneles

La siguiente figura podemos ver un de JSplitPane dentro un JTabbedPane:
Ver la figura 37 de la página 72

17.3. API

Para mas información del API completa:

- <http://download.oracle.com/javase/6/docs/api/javaw/swing/JPanel.html>
- <http://download.oracle.com/javase/6/docs/api/javaw/swing/JTabbedPane.html>
- <http://download.oracle.com/javase/6/docs/api/javaw/swing/JDesktopPane.html>
- <http://download.oracle.com/javase/6/docs/api/javaw/swing/JSplitPane.html>

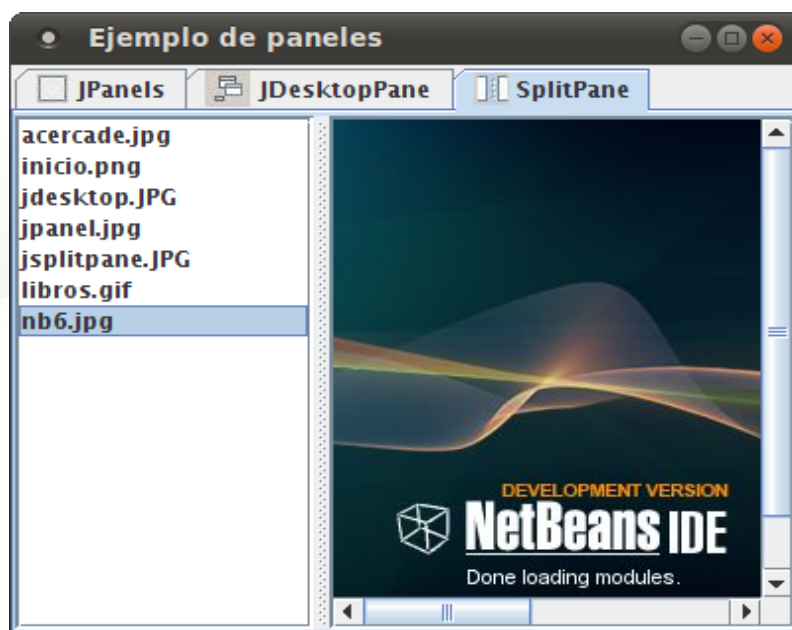


Figura 37: JPannel01

17.4. Práctica: jpanel01

Abrir el proyecto JPannel01 y ver el uso de los componentes.

17.5. Identificar el panel

La clase JTabbedPane permite contener varios JPanel (Tabs) siempre es interesanet identificar el tab en el que estamos, para ello usaremos la función `JTabbedPane1.getSelectedIndex()`.

- Captura de programa.
Ver la figura 38 de la página 72

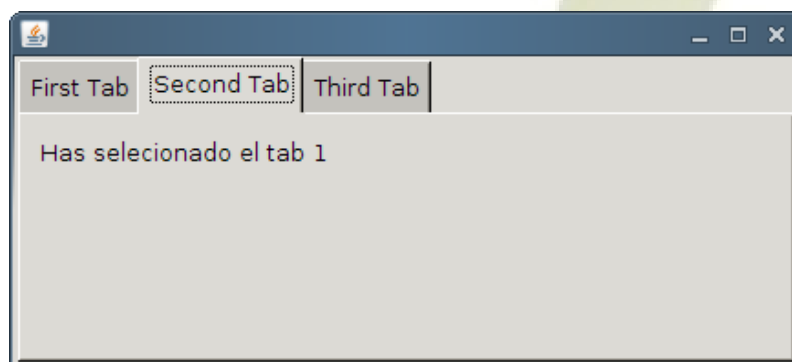


Figura 38: jpanel02

- Diagrama de paquetes
Ver la figura 39 de la página 73

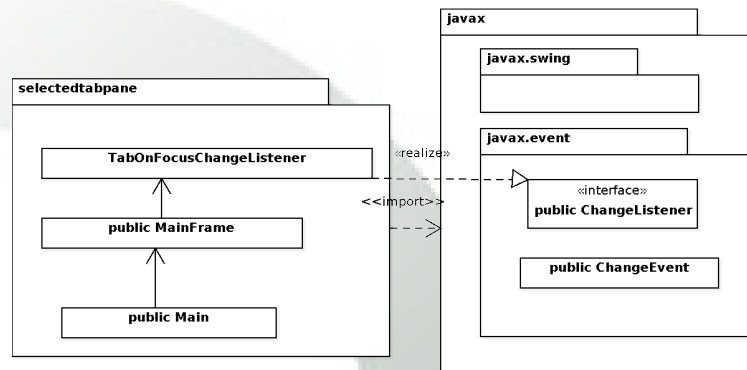


Figura 39: jpanel02

En la clase MainFrame, cuando crea las componentes crea un objeto TabOnFocusChangeListener, esto permitirá avisarnos de cuando cambiamos de Panel.

```
jTabbedPane1.addChangeListener(new TabOnFocusChangeListener(this));
```

La clase TabOnFocusChangeListener, llama al método tabsOnFocus, sobre escribiendo el método stateChanged de la clase ChangeListener la cual herada el método.

```
public void stateChanged(ChangeEvent e) {
    jframe.tabsOnFocus(e);
}
```

La clase MainFrame codifica el método tabsOnFocus, que permite obtener el JPanel seleccionado mediante el método getSelectedIndex.

```
public void tabsOnFocus(ChangeEvent e) {
    int num = jTabbedPane1.getSelectedIndex();
    String texto = "Has seleccionado el tab " + String.valueOf(num);
    tabOneLabel.setText(texto);
    tabTwoLabel.setText(texto);
    tabThreeLabel.setText(texto);
}
```

Es importante destacar que cada tab en el JTabbedPane es independiente. Es decir, que si tenemos abierto el tab1, las operaciones del tab2 no se harán hasta que entremos en el tab2.

17.6. Práctica: jpanel02

Abrir el proyecto jpanel02 y ver el uso de los componentes y hacer que muestre el título del tab.

Ver la figura 40 de la página 74

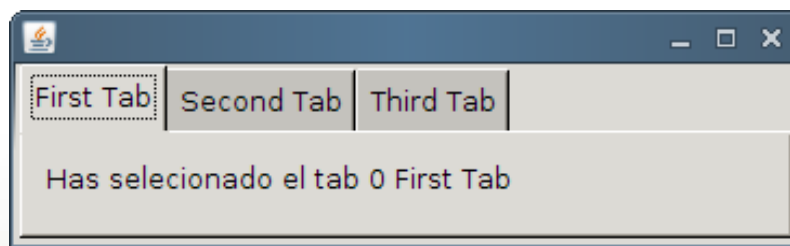


Figura 40: jpanel02s

18. MENÚS

Los menús son únicos en que, por convención, no se sitúan con los otros componentes en el UI. En su lugar, aparecen en una barra de menú o en un menú desplegable.

Una barra de menú contiene uno o más menús, y tiene una posición dependiente de la plataforma - normalmente debajo de la parte superior de la ventana.

Un menú desplegable es un menú que es invisible hasta que el usuario hace una acción del ratón específica de la plataforma, como pulsar el botón derecho del ratón sobre un componente. Entonces el menú desplegable aparece bajo el cursor.

La siguiente figura muestra los componentes Swing que implementan cada parte de un sistema de menús

Ver la figura 41 de la página 74

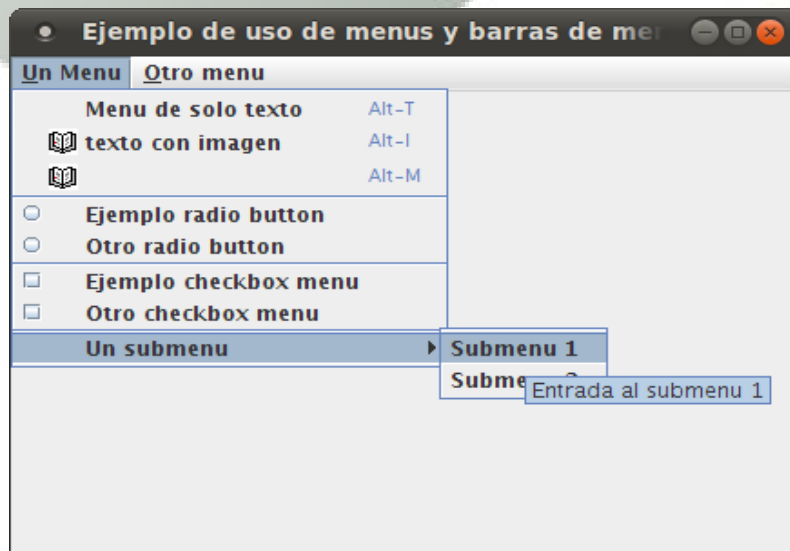


Figura 41: swing7

Para poder crear una estructura de menús desde el IDE accederemos al panel Paleta y seleccionaremos el componente JMenuBar:

- 1.- Añadimos el componente al área de diseño
- 2.- Seleccionamos en el panel Inspector el nuevo componente `JMenu1`:
 - 1.- Botón de la derecha de ratón Cambiar Nombre de la Variable: `JMenuMenu1`
 - 2.- Botón de la derecha del ratón Editar Texto: Un menú
 - 3.- Botón de la derecha del ratón Añadir de Paleta.
 - Menu - Un submenú
 - Menu Item - Una entrada simple de menú
 - Menu Item /CheckBox - Un menú de check
 - Menu Item / RadioButton - Un menú radio
 - Separator - Una línea de separación
- 3.- Si seleccionamos el componente `JMenuBar1` creado:
 - 1.- Botón derecha ratón Añadir Desde Paleta
 - Menu Otra entrada de menú.

Por ejemplo si creamos un elemento de tipo `JMenuItem`, es decir, seleccionando el componente `JMenu` y con el botón de la derecha del ratón - Añadir de Paleta y seleccionamos el Menu Item, podemos acceder al panel de Propiedades y cambiar las siguientes propiedades:

- `accelerator`. Tecla de aceleración (Por ejemplo: `Alt + M`)
- `text`. el texto del menú (Por ejemplo: Añadir)
- `icon`. Classpath y la ruta de la imagen (Conviene tener todas las imágenes en un package)
- `toolTipText`. Texto flotante de descripción (Por ejemplo: Añadir alumnos)

18.0.1. Práctica

Realizar un proyecto que muestre un menú con 4 entradas: Añadir, Listar, Matricular y Salir tal y como se muestra en la figura 42 de la página 76.

Si queremos añadir una barra de herramientas accederemos al panel Palette y seleccionaremos el componente `JToolBar`:

Lo añadimos al área de diseño y para añadir los botones seleccionamos el componente `JButton`, podemos añadir botones con texto, con imagen o texto e imagen.



Figura 42: Menú

18.0.2. API

La siguiente tabla muestra alguno de los métodos mas utilizados:

Método	Propósito
<code>void setEnabled(boolean)</code>	Si el argumento es true, activa el ítem de menú, si es false lo desactiva.

Para mas información del API completa:

- <http://download.oracle.com/javase/6/docs/api/javaw/swing/JMenuBar.html>
- <http://download.oracle.com/javase/6/docs/api/javaw/swing/JMenuItem.html>
- <http://download.oracle.com/javase/6/docs/api/javaw/swing/JMenu.html>
- <http://download.oracle.com/javase/6/docs/api/javaw/swing/JCheckBoxMenuItem.html>
- <http://download.oracle.com/javase/6/docs/api/javaw/swing/JRadioButtonMenuItem.html>
- <http://download.oracle.com/javase/6/docs/api/javaw/swing/JToolBar.html>

19. JTABLE

19.1. Conceptos básicos

JTable sirve para crear tablas, con sus filas y columnas.

Ejemplo: Ver la figura 43 de la página 81

19.1.1. JTable y DefaultTableModel

La forma más sencilla de usar un JTable y tener toda su funcionalidad es instanciar un DefaultTableModel y meterlo en el JTable, en el constructor.

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

Podemos añadir columnas directamente en el modelo

```
modelo.addColumn("etiqueta columna 1");  
modelo.addColumn("etiqueta columna 2");
```

Podemos añadir datos directamente en el modelo, así como borrarlos o modificarlos

```
Object [] fila = new Object[2];  
fila[0] = "dato columna 1";  
fila[1] = "dato columna 3";  
  
// Añade una fila al final  
modelo.addRow ( fila );  
  
// Cambia el valor de la fila 1, columna 2.  
modelo.setValueAt ("nuevo valor", 0, 1);  
  
// Borra la primera fila  
modelo.removeRow (0);
```

Todo lo que hagamos se reflejará de inmediato en el JTable.

19.1.2. Obtener fila y columna del JTable en la que se hace click

A veces nos interesa seleccionar una fila del JTable para hacer algo con ella (sacar un menú, recoger datos para mostrarlos en otro sitio, etc).

Una forma de hacerlo es añadiendo un MouseListener al JTable, de esta manera

```
tabla.addMouseListener(new MouseAdapter()  
{  
    public void mouseClicked(MouseEvent e)  
    {  
        int fila = tabla.rowAtPoint(e.getPoint());  
        int columna = tabla.columnAtPoint(e.getPoint());  
        if ((fila > -1) && (columna > -1))  
            System.out.println(modelo.getValueAt(fila,columna));  
    }  
});
```

Hemos añadido un MouseAdapter para no tener que implementar todos los métodos del MouseListener.

Con el método tabla.rowAtPoint() es posible enterarnos en qué fila de del JTable ha ocurrido el evento del ratón (el click en este caso). Para ello basta llamar a este método pasándole las coordenadas x,y del evento de ratón, que se obtienen con el método e.getPoint().

Una vez que sabemos la fila, debemos comprobar si es mayor que -1. El método `rowAtPoint()` nos devuelve -1 si pinchamos en el `JTable`, pero fuera de cualquier fila. Es el caso de que el `JTable` tenga un tamaño en pixels superior al que le corresponde según su número de filas.

Lo mismo vale para `columnAtPoint()`.

Una vez que tenemos la fila y sabemos que no es -1, es fácil a través del modelo obtener los datos correspondientes. En este caso se escribe por pantalla con un `System.out.println()` el valor de la fila y columna que se ha seleccionado.

19.1.3. Hacer que una celda del `JTable` no sea editable

Si usamos `DefaultTableModel` las celdas del `JTable` son editables por defecto. A veces esto no nos interesa y `JTable` no tiene métodos para impedirlo. La forma de decidir qué celdas son o no editables es hacer nuestro propio modelo de datos, nuestro `TableModel`. La forma sencilla de hacerlo es heredar de `DefaultTableModel` y redefinir el método `isCellEditable()` para que sirva a nuestros propósitos

```
public class MiModelo extends DefaultTableModel
{
    public boolean isCellEditable (int row, int column)
    {
        // Aquí devolvemos true o false según queramos que una celda
        // identificada por fila,columna (row,column), sea o no editable
        if (column == 3)
            return true;
        return false;
    }
}
```

En este ejemplo, hemos creado nuestro propio modelo de datos que hace que la columna 4 (los índices empiezan en cero) de la tabla sea editable y el resto no. Ahora simplemente instanciamos el `JTable` usando este modelo y rellenamos los datos igual que antes

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

19.1.4. Cambiar el tipo de dato con `DefaultTableModel`

`DefaultTableModel` por defecto le dice al `JTable` que todos los datos que tiene son `Object`. A veces, porque queramos cambiar el `TableCellRenderer` o cualquier otro motivo, nos interesa que determinadas columnas se consideren como `Boolean`, como `Integer` o cualquier otro tipo de dato.

Para modificar esto, tenemos que crearnos nuestro propio modelo de Datos. La forma más sencilla es heredar de `DefaultTableModel` y redefinir el método `getColumnClass()`.

```
public class MiModelo extends DefaultTableModel
{
    /** Primera columna Boolean, segunda Integer y el resto Object */
    public Class getColumnClass(int columna)
    {

```

```
        if (columna == 0) return Boolean.class;
        if (columna == 1) return Integer.class;
        return Object.class;
    }
}
```

En el ejemplo se ha hecho que la primera columna sea de Boolean, la segunda de Integer y el resto de Object.

Una cosa curiosa de los Boolean, es que el JTable al pintarlos los pone como JCheckBox.

19.1.5. Hacer visible una fila concreta del JTable dentro de un JScrollPane

Para que un JTable tenga barras de scroll y tenga una cabecera con las etiquetas de las columnas, es necesario meterla en un JScrollPane. Esto se puede hacer de dos formas.

```
JTable tabla = new JTable();
JScrollPane scroll = new JScrollPane(tabla);
```

o bien

```
JTable tabla = new JTable();
JScrollPane scroll = new JScrollPane();
scroll.setViewportView(tabla);
```

es bastante habitual al principio equivocarse y usar el método add(), que NO funcionará correctamente

```
JTable tabla = new JTable();
JScrollPane scroll = new JScrollPane();

// Esto NO funciona.
scroll.add(tabla);
```

Cuando tenemos un JTable metido dentro de un JScrollPane, a veces queremos que las barras de scroll se desplacen para que una determinada celda sea visible.

Una posible solución es:.

```
...
// Nos devuelve la posición en pixels de una celda en fila, columna
Rectangle r = tabla.getCellRect( fila, columna, true);

// Mueve el scroll para que el rectangulo sea visible
scrollPane.getViewPort().scrollRectToVisible (r);
```

19.1.6. Clases para trabajar con tablas

Hay cuatro posibilidades: Trabajar directamente con JTable, usar TableModel, usar AbstractTableModel o usar DefaultTableModel.

La primera está bien si la tabla es fija y no vas a requerir cambio en los datos. Fíjate que JTable apenas tiene métodos para borrar filas, añadir columnas, etc, etc.

La de TableModel es la más costosa de programar, pero la que da más versatilidad. TableModel es una interface y tienes que escribir tú todos los métodos para añadir, borrar y modificar datos, además de implementar un mecanismo de suscripción a cambios en los datos <http://www.chuidiang.com/java/tablas/tablamodelo/tablamodelo.html>. Puedes hacerlo como quieras y de la forma que mejor te venga, pero tienes que hacerlo tú todo.

La de DefaultTableModel es la más sencilla. No tienes que escribir nada de código. DefaultTableModel tiene todos los métodos necesarios para añadir, modificar y borrar celdas.

AbstractTableModel es una intermedia entre las dos anteriores. Tiene implementado y por tanto no tienes que hacer tú todos los mecanismos de suscripción a cambios de datos. Únicamente tienes que codificar los añadir, borrar y modificar datos.

19.2. JTable y Netbeans

Con la clase JTable, se puede mostrar tablas de datos, y opcionalmente permitir que el usuario los edite. JTable no contiene ni almacena datos; simplemente es una vista de nuestros datos (Mas adelante veremos el patrón MVC).

Para crear un componente JTable desde el IDE de NetBeans seleccionaremos del panel Palette el componente JTable.

Un componente JTable debe de estar ubicado dentro de un JScrollPane, de esta forma si nuestra tabla contiene mas datos que proporciona el área de la vista podremos desplazarnos por medio del scroll. Si en el IDE de NetBeans seleccionamos JTable automáticamente nos añadirá un JScrollPane.

Toda tabla contiene un modelo que es el que representa los datos de forma visual (más adelante veremos el uso del patrón MVC), si queremos modificar el conjunto de datos que contiene la tabla seleccionaremos la propiedad model. Por defecto el IDE de Netbeans crea un DefaultTableModel para poder trabajar con los elementos de una JTable.

Para acceder a la propiedad model del componente:

- Seleccionaremos el componente JTable del panel inspector.
- Accederemos al panel Propiedades propiedad model
- Modificar las columnas y valores.

Este es un ejemplo de un JTable básico editado desde el IDE de NetBeans:

Ver la figura 43 de la página 81

Código autogenerado cuando se define la propiedad model:

```
jTableBasico.setModel(new javax.swing.table.DefaultTableModel(
```



Figura 43: Swing11

```
new Object [][] {
{"ALBERTO", "APARICIO VILA", new Integer(28), "BASKET", null},
{"JUAN", "LOPEZ APARICIO", new Integer(34), "FUTBOL", new Boolean(true)},
{"MARIA", "GARCIA PALOP", new Integer(36), "AJEDREZ", new Boolean(true)},
{"DOLORES ", "FUERTES DE BARRIGA", new Integer(28), "FUTBOL", null}
},
new String [] {"Nombre", "Apellidos", "Edad", "Deporte", "Vegetariano"}
) {Class[] types = new Class [] {
java.lang.String.class, java.lang.String.class, java.lang.Integer.class,
java.lang.String.class, java.lang.Boolean.class
};
public Class getColumnClass(int columnIndex) {
return types [columnIndex];
}
});
```

Como se ve crea de forma automática una clase DefaultTableModel con los datos y las columnas que hemos definido mediante la propiedad model.

19.3. Crear un modelo de tabla

En ocasiones será interesante crear nuestro propio modelo de tabla para tener más control sobre el modelo. El uso de la clase por defecto DefaultTableModel no está mal, pero tiene un problema de rendimiento bastante gordo, y es que utiliza la clase Vector, la clase Vector tiene todos sus métodos sincronizados y en la mayor parte de los casos innecesarios. Sería mas conveniente usar un ArrayList para almacenar los datos, es equivalente a Vector, pero no tiene todos sus métodos sincronizados con lo cual el rendimiento con relación a DefaultTableModel es bastante mejor.

En el siguiente fragmento de código mostramos la clase SimpleTableModel que hereda de AbstractTableModel e implementa un modelo de tabla basado en ArrayList. Código referencia ejemplo proyecto **swing11**. SimpleTableModel.java:


```
public class SimpleTableModel extends AbstractTableModel {
    ArrayList datos = new ArrayList();
    String [] columnas = {"Nombre", "Apellidos", "Edad", "Deporte",
    "Vegetariano"};
    Class[] types = new Class [] {
    java.lang.String.class, java.lang.String.class, java.lang.Integer.class,
    java.lang.String.class, java.lang.Boolean.class};
    /** Creates a new instance of SimpleTableModel */
    public SimpleTableModel() {
        //Creamos los objetos de una fila
        Object [] fila = new Object[5];
        fila[0] = "Alberto";
        fila[1] = "Aparicio vila";
        fila[2] = new Integer(29);
        fila[3] = "BASKET";
        fila[4] = new Boolean(false);
        //El array list contendra un array de Object[] en cada fila
        datos.add(fila);
    }
    public String getColumnName(int col) {
        return columnas[col].toString();
    }
    public int getRowCount() { return datos.size(); }
    public int getColumnCount() { return columnas.length; }
    public Object getValueAt(int row, int col) {
        Object[] fila = (Object[]) datos.get(row);
        return fila[col];
    }
    public Class getColumnClass(int columnIndex) {
        return types[columnIndex];
    }
    public boolean isCellEditable(int row, int col) { return true; }
    public void setValueAt(Object value, int row, int col) {
        Object [] fila = (Object []) datos.get(row);
        fila[col] = value;
        fireTableCellUpdated(row, col);
    }
    public void addRow(Object [] fila) {
        datos.add(fila);
        fireTableDataChanged();
    }
    public void removeRow(int fila) {
        datos.remove(fila);
        fireTableDataChanged();
    }
}
```


Para utilizar un modelo personalizado con el IDE de NetBeans necesitamos:

- 1.- Seleccionamos el componente JTable en el panel inspector
- 2.- Accedemos al panel de Contenido de la Tabla:
 - En Modelo de la Tabla
 - En el combo seleccionamos Modelo Personalizado
 - Añadimos el siguiente código:

```
new SimpleTableModel()
```

Ejemplo código referencia proyecto **swing11**:

Ver la figura 44 de la página 83

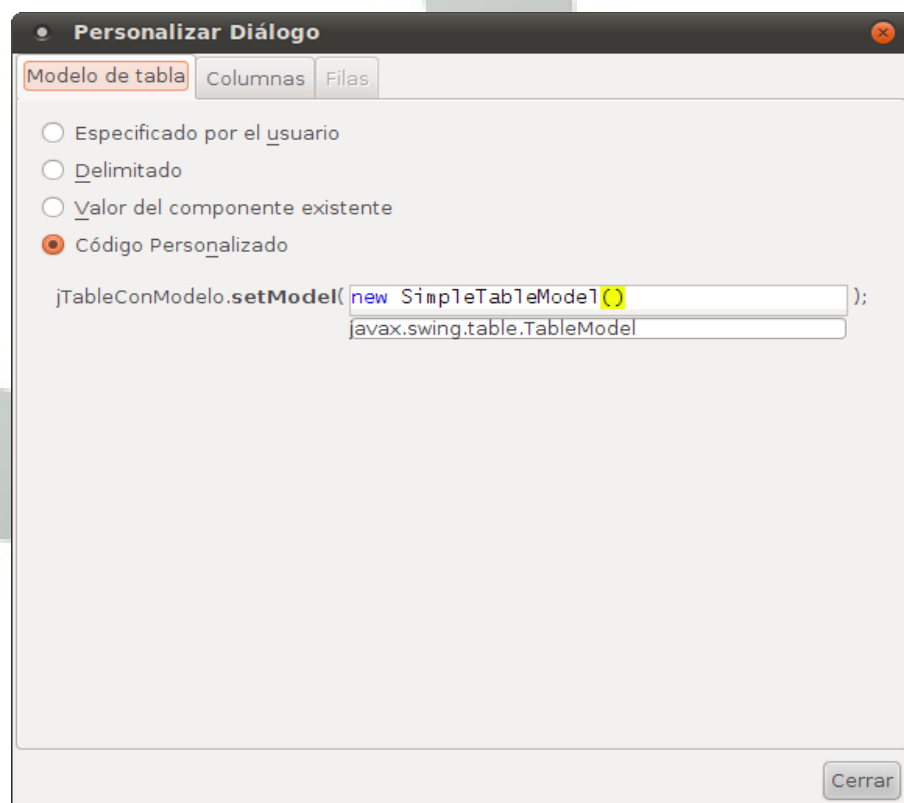


Figura 44: Swing11. Modelo de Tabla Simple

19.4. Práctica

Ampliar el modelo anterior SimpleTableModel.java, para que permita una nueva columna que se llame Localidad de tipo String que añada la localidad del alumno, tal y como muestra la siguiente figura:

Ver la figura 45 de la página 84



Figura 45: Swing11b. Modelo de Tabla. Práctica

19.5. Utilizando un combobox como editor de celda

Si queremos que alguna de nuestras columnas de la tabla permita seleccionar un valor a partir de un JComboBox tal y como muestra la figura, necesitamos especificar para la columna que dicho valor se ha de seleccionar a partir de un JComboBox.

Ver la figura 46 de la página 84



Figura 46: Swing12

Para añadir un editor de celda de tipo ComboBox en el IDE de NetBeans:

- Seleccionamos el componente JTable en el panel inspector
- Accedemos al panel de Código y seleccionamos Código pos-inicio. (Es decir añadimos el siguiente código después de las inicializaciones del componente JTable)

```
//Columna que mostrará los datos a partir de un JComboBox
TableColumn columnaDeporte = jTableCell.getColumnModel().getColumn(3);
//Creamos el ComboBox y añadimos los elementos que formarán parte
JComboBox jTableCellDeporte = new JComboBox();
jTableCellDeporte.addItem("ESQUI");
jTableCellDeporte.addItem("BASKET");
jTableCellDeporte.addItem("TENIS");
jTableCellDeporte.addItem("AJEDREZ");
jTableCellDeporte.addItem("NADA");
//Añadimos el JComboBox como editor de la columna deporte
columnaDeporte.setCellEditor(new DefaultCellEditor(jTableCellDeporte));
```

Ver la figura 47 de la página 85

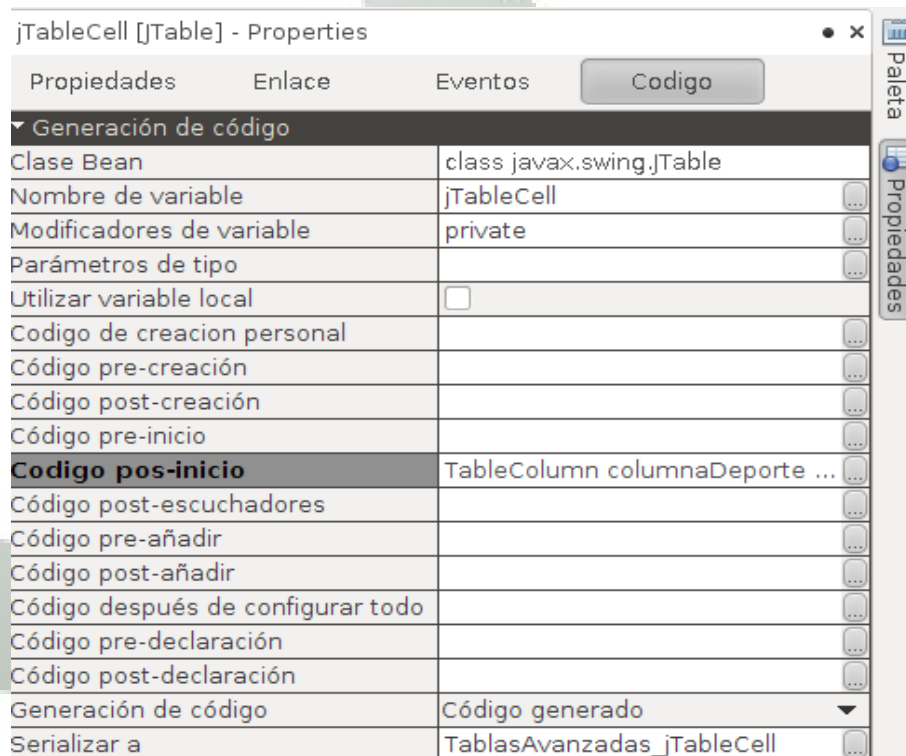


Figura 47: Swing12. Código

Seleccionamos la columna de Deporte a partir del método `getColumnModel()` en el objeto `jTableCell`, este método devuelve un objeto que representa el modelo de columnas `TableColumnModel` y a partir del método `getColumn()` obtenemos la columna correspondiente, en nuestro caso la columna con índice 3 (deporte).

19.6. Personalizando la edición de celdas

Si queremos que nuestras celdas tengan un editor personalizado necesitamos crear nuestra propia clase para tener este tipo de funcionalidad. Para ello necesitamos utilizar la interfaz `TableCellRenderer` que permite definir el método `getTableCellRendererComponent()` que devuelve un objeto de tipo `Component`, es decir, cualquier tipo de componente swing.

La siguiente figura muestra como quedaría la columna Nombre cuando definimos un Renderer especial:

Ver la figura 48 de la página 86



Figura 48: Swing12

Y el código que nos permite realizar esta personalización lo podemos ver en la clase ColorRenderer que implementa el interfaz TableCellRenderer.

Ejemplo código referencia proyecto **swing12**:

```
public class ColorRenderer extends JLabel
    implements TableCellRenderer{
    /** Creates a new instance of ColorRenderer */
    public ColorRenderer() {
    }
    /**
    * Devuelve un componente para personalizar la celda
    * @param table Componente JTable padre
    * @param value Valor de la celda
    * @param isSelected True si la celda está seleccionada
    * @param hasFocus Si tiene el foco
    * @param row Indice de la fila
    * @param column Indice de la columna
    * @return Componente para ser visualizado
    */
    public Component getTableCellRendererComponent(JTable table, Object value,
        boolean isSelected, boolean hasFocus, int row, int column) {
        if (!isSelected) {
            this.setBackground(Color.YELLOW);
            this.setForeground(Color.PINK);
        } else {
            this.setBackground(Color.LIGHT_GRAY);
            this.setForeground(Color.RED);
        }
        this.setText((String)value);
        this.setToolTipText("El valor de esta celda es: "+value);
        this.setOpaque(true);
        return this;
    }
}
```

Una vez definida la clase para asociarla a la columna de Nombre seguiremos los siguientes pasos:

- 1.- Seleccionamos en el panel Inspector el componente JTable, en nuestro caso JTableRenderer
- 2.- Accedemos al panel Código y seleccionamos Código Post-Inicio.
- 3.- Añadimos el siguiente fragmento de código:

```
TableColumn nombre = jTableRenderer.getColumnModel().getColumn(0);
nombre.setCellRenderer( new ColorRenderer() );
```

19.7. Práctica

Con los pasos anteriores usar la plantilla del proyecto swing12b, y añadir un Renderer a la columna edad (ColorRenderer) y que realice lo siguiente:

- Si la *Edad* ≥ 0 y ≤ 15 entonces el color debe ser Rojo
- Si la *Edad* > 15 y ≤ 30 entonces el color debe ser Verde
- Si la *Edad* > 30 entonces el color debe ser Azul

Ver la figura 49 de la página 87

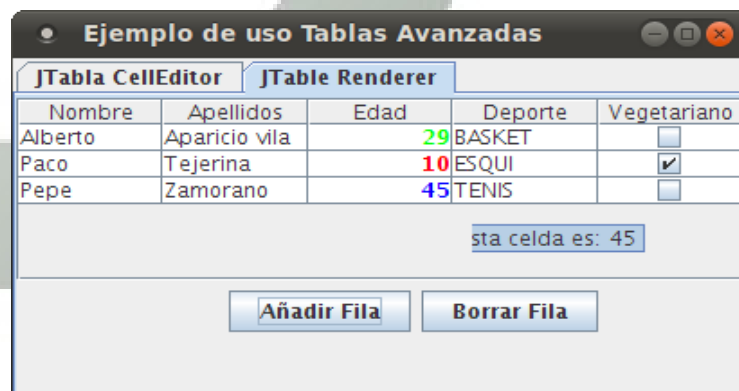


Figura 49: Swing12b

19.8. Ordenación en tablas

Para poder realizar una ordenación en las tablas, de manera que haciendo clic en cada uno de los encabezados, pudiera realizar una ordenación ascendente o descendente, necesitaremos de uno o mas modelos especializados, además del modelo de datos.

Ver la figura 50 de la página 88

En nuestro caso vamos a utilizar una clase llamada TableSorter.java que realizará la ordenación del modelo que le pasemos en su constructor, que en nuestro caso le pasaremos nuestro modelo de referencia SimpleTableModel.java.

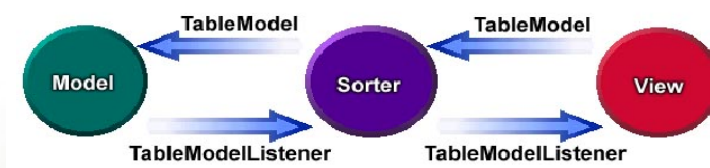


Figura 50: Modelo de datos de ordenación

Para realizar la ordenación a través del IDE de NetBeans realizaremos lo siguiente:

- 1.- Seleccionaremos del panel Inspector la tabla a ordenar
- 2.- Seleccionaremos en el panel Código de el evento Código Pre-Creación y añadiremos el siguiente fragmento de código:

```
SimpleTableModel miModelo = new SimpleTableModel();
TableSorter modeloOrdenado = new TableSorter(miModelo);
```

- 3.- Seleccionaremos en el panel Código el evento Código Post-Creación y añadiremos el siguiente fragmento de código:

```
JTableHeader header = jTableOrdenada.getTableHeader();
modeloOrdenado.setTableHeader(header);
```

- 4.- Seleccionaremos en el Panel Propiedades la propiedad model.
 - En el combo seleccionamos "Código del Personal"
 - Añadimos el siguiente código:

```
modeloOrdenado
```

Ver la figura 51 de la página 88

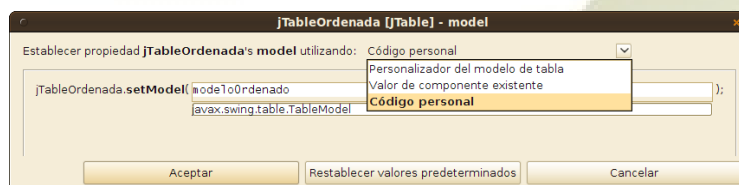


Figura 51: Selección del Modelo

19.9. Práctica

Utilizando los pasos anteriores usar la plantilla del proyecto swing13, añadir un modelo ordenado siguiendo los pasos anteriores. Una vez realizados los pasos si hacemos clic en el título de las columnas el modelo hará una ordenación, tal y como muestra la figura siguiente:

Ver la figura 52 de la página 89

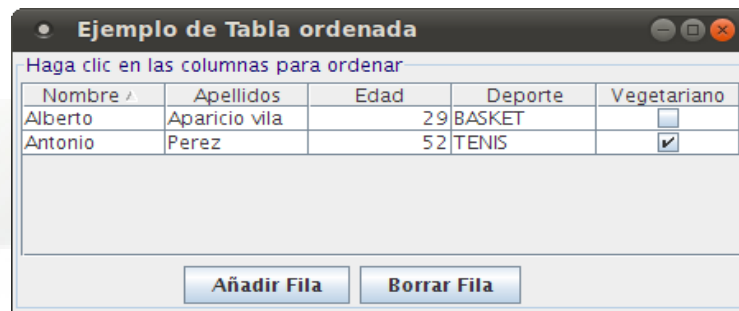


Figura 52: Swing13

19.10. API

El API para hacer uso de JTables es compleja y las podemos dividir en las siguientes categorías:

Clase/Interface	Propósito
JTable	El componente que presenta la tabla al usuario. http://download.oracle.com/javase/6/docs/api/javax/swing/JTable.html
JTableHeader	El componente que presenta los nombres de columnas al usuario. Por defecto, la tabla genera este componente automáticamente. http://download.oracle.com/javase/6/docs/api/javax/swing/table/JTableHeader.html
TableModel, AbstractTableModel	Respectivamente, el interface que un modelo de tabla debe implementar y la superclase usual para implementaciones de modelos de tabla. http://download.oracle.com/javase/6/docs/api/javax/swing/table/TableModel.html
TableCellRenderer, DefaultTableCellRenderer	Respectivamente, el interface que un intérprete de celda debe implementar y la implementación más usual. http://java.sun.com/javase/6/docs/api/javax/swing/TableCellRenderer.html
TableCellEditor, DefaultCellEditor	Respectivamente, el interface que debe implementar un editor de celda, y la implementación más usual. http://download.oracle.com/javase/6/docs/api/javax/swing/table/TableCellRenderer.html
TableColumnModel, DefaultTableColumnModel	Respectivamente, el interface que debe implementar un modelo de columna, y su implementación usual. Normalmente no tenemos que tratar directamente con el modelo de columna a menos que necesitemos obtener el modelo de selección de columna u obtener un índice de columna o un objeto. http://download.oracle.com/javase/6/docs/api/javax/swing/table/TableColumnModel.html
TableColumn	Controla todos los atributos de una columna de la tabla, incluyendo, redimensionado, anchuras mínima, máxima, preferida y actual; y editor/intérprete opcional específico de la columna. http://download.oracle.com/javase/6/docs/api/javax/swing/table/TableColumn.html
DefaultTableModel	Un modelo de tabla basado en Vector utilizado por JTable cuando construimos una tabla sin especificar modelo de datos ni datos. http://download.oracle.com/javase/6/docs/api/javax/swing/table/DefaultTableModel.html

19.11. Eventos

El conjunto de eventos que podemos capturar para una tabla los podemos ver si seleccionamos en el panel Inspector el componente JTable y con el botón derecha de ratón accedemos a Eventos.

Para modificar el código pinchamos en el combo y seleccionamos el elemento que aparece. En el panel de código habrá una nueva función donde podremos escribir nuestro código de manejo del evento.

19.12. Práctica

Probad a capturar el Mouse en su evento `MouseClicked`, y mostrar la fila seleccionada cuando se haga clic en las filas.

20. JCALENDAR

JCalendar es una librería externa (que no viene en el jdk de java) que permite gestionar las fechas visualizando el calendario.

Ver la figura 53 de la página 90

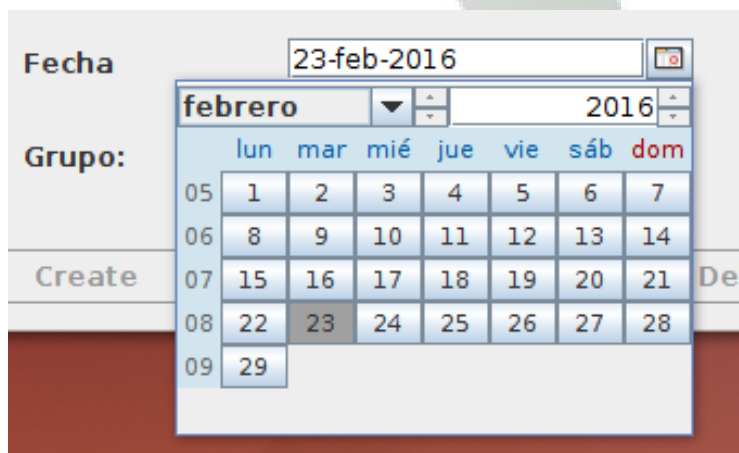


Figura 53: JCalendar

Los objeto `JDateChooser` permite gestionar fechas. Utilizaremos las funciones `getDate()` y `setDate`. En concreto si `vista.getjdcFecha()` nos devuelve un objeto de tipo `JDateChooser`, tendremos:

Podremos obtener el objeto de tipo fecha con:

```
Date fecha = vista.getjdcFecha().getDate();
```

Podremos modificar el objeto de tipo fecha con:

```
vista.getjdcFecha().setDate(alumno.getFecha());
```

21. MVC. SWING

Los componentes y los eventos son los pilares básicos en la construcción de una interfaz gráfica de usuario en Swing. En este punto se verá cómo están contruidos internamente los componentes, lo que permitirá añadir más funcionalidad a los programas que se desarrollen. Además, se mostrará cómo el principio básico de construcción de los componentes se puede aplicar en cualquier aplicación con interfaz gráfica a cualquier escala.

Se verán los modelos que permitirán utilizarse en más de una componente, y la ventajas que representan, siendo la más importante que la inserción de datos en el modelo, afectan a las componentes que lo utilizan.

21.1. Introducción

Los componentes de Swing basan su construcción en una arquitectura denominada model-view-controller (MVC) que proviene de SmallTalk.

Esta arquitectura establece que en la programación de aplicaciones con interfaz gráfica de usuario deben aparecer tres partes fundamentales:

- 1.- **Modelo (model).** Parte que albergará los datos y cuya construcción será totalmente ajena a la forma en que se representen esos datos en la interfaz gráfica.
- 2.- **Vista (view).** Parte encargada de la representación gráfica de los datos.
- 3.- **Controlador (controller).** Parte que se ocupará de la interpretación de las entradas del usuario, las cuales permiten modificar el modelo (datos) de forma adecuada.

La arquitectura MVC puede aplicarse a muchos niveles en la programación de una aplicación. Se puede aplicar, por ejemplo, para la creación de un simple botón de la interfaz gráfica de usuario y se puede aplicar también, por ejemplo, en la estructuración de una aplicación distribuida en Internet, en la cual se observan las tres partes fundamentales.

En nuestro caso vamos a ver la arquitectura MVC aplicada en dos niveles:

- **Construcción interna de componentes.** Internamente los componentes de Swing utilizan la arquitectura MVC. Se verá con más detalle cómo se realiza esta implementación y qué ventajas ofrece.
- **Construcción de una aplicación con interfaz gráfica.** Se guiará el desarrollo de las aplicaciones con interfaz gráfica de usuario aplicando la arquitectura MVC.

21.2. Construcción Interna de Componentes.

En este apartado se verá cómo implementar los conceptos generales de la arquitectura MVC para construir internamente los componentes de la interfaz gráfica de usuario, y las ventajas que esto conlleva.

Hasta ahora, se ha visto que los componentes de la interfaz gráfica de usuario son los únicos objetos capaces de generar o elevar eventos. Se puede asociar código a estos eventos, el cual se ejecutará cuando el evento se produzca.

Pero, la generación de eventos no es exclusiva de los componentes de la interfaz gráfica de usuario. Por ejemplo, un temporizador puede configurarse para generar un evento cada 20 minutos. En realidad, cualquier objeto puede generar

eventos cuando su estado cambia si se implementa su clase para que tenga esta capacidad.

En una primera aproximación, la implementación interna de cada componente se realiza delegando la funcionalidad en tres objetos: un objeto será el encargado de albergar los datos; otro de representar gráficamente esos datos y el tercero de interpretar las acciones del usuario. Estos objetos se relacionan entre sí de la forma que se muestra en la siguiente figura:

Ver la figura 54 de la página 92

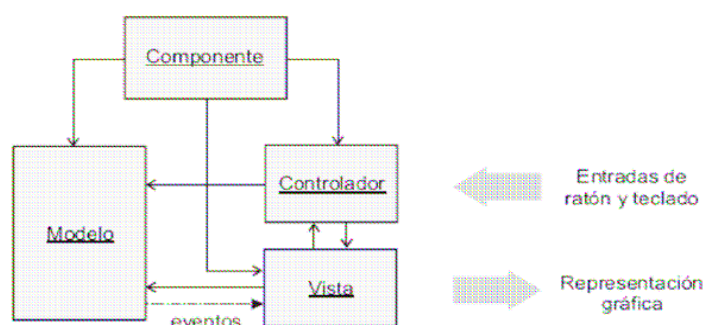


Figura 54: MVC

La responsabilidad de cada uno de los objetos anteriores es la siguiente:

- 1.- **Modelo.** Objeto encargado de almacenar los datos. Como puede observarse en la figura, este objeto eleva eventos cuando su estado cambia. Por ejemplo, si el componente es un JCheckbox, el modelo es el objeto que guarda la información sobre si el botón de chequeo está seleccionado o no. Y cada vez que su estado cambie el objeto elevará un evento.
- 2.- **Vista.** Objeto encargado de representar visualmente los datos que están almacenados en el modelo. Este objeto cambia la representación visual cada vez que el modelo eleva un evento de cambio. Si el evento es ligero, la vista debe preguntar al modelo sobre su estado para poder visualizarle; si el evento es informativo, con la información que viene en el propio evento es suficiente para poder representar los datos.
- 3.- **Controlador.** Es el responsable de interpretar las entradas del usuario. En el ejemplo del botón de chequeo este objeto interpreta las teclas que pulsa el usuario, y si es la tecla espacio cambia el estado del botón. En los componentes más complejos, la vista y el controlador tienen que intercambiar mucha información.

Los desarrolladores de Swing se basaron en esta idea para construir los componentes, pero realizaron algunos cambios importantes: fusionaron en un solo objeto la responsabilidad de visualización y control, debido al fuerte acoplamiento que tienen y a la complejidad de la interacción entre ambos. A esta nueva clase se la denomina delegado UI (UI delegate). De esta forma, los componentes de

Swing tienen lo que se conoce como arquitectura modelo- delegado o arquitectura de modelo separable.

Los componentes tienen principalmente 2 atributos: el modelo y el delegado, tal y como se muestra en la siguiente figura:

Ver la figura 55 de la página 93

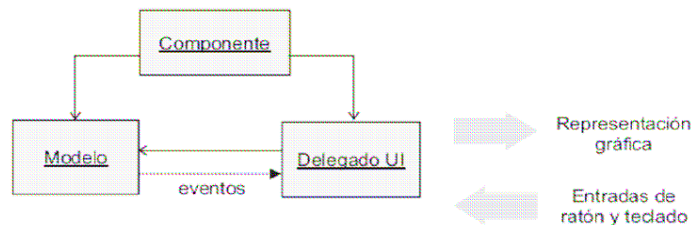


Figura 55: Estructura de los componentes swing

Para cada uno de los componentes existe una interfaz para su modelo y una clase para su delegado UI. Algunos componentes comparten la misma interfaz para definir su modelo. En la siguiente tabla se muestra las interfaces para cada uno de los modelos de los componentes de la interfaz de usuario:

COMPONENTE	MODEL INTERFACE
javax.swing.AbstractButton	ButtonModel
javax.swing.JButton	ButtonModel
javax.swing.JMenuItem	ButtonModel
javax.swing.JCheckBoxMenuItem	ButtonModel
javax.swing.JMenu	ButtonModel
javax.swing.JRadioButtonMenuItem	ButtonModel
javax.swing.JToggleButton	ButtonModel
javax.swing.JCheckBox	ButtonModel
javax.swing.JRadioButton	ButtonModel
javax.swing.JScrollBar	BoundedRangeModel
javax.swing.JSlider	BoundedRangeModel
javax.swing.JSpinner	SpinnerModel
javax.swing.JList	ListModel, ListSelectionModel
javax.swing.JComboBox	ComboBoxModel
javax.swing.JProgressBar	BoundedRangeModel
javax.swing.JTabbedPane	SingleSelectionModel
javax.swing.JTable	TableModel, TableColumnModel
javax.swing.JTree	TreeModel, TreeSelectionModel
javax.swing.text.JTextComponent	Document
javax.swing.EditorPane	Document
javax.swing.JTextPane	Document
javax.swing.JTextArea	Document
javax.swing.JTextField	Document
javax.swing.JFormattedTextField	Document
javax.swing.JPasswordField	Document

La mayoría de los componentes tienen un modelo y un delegado como atributo, y los asocian de la forma vista anteriormente. Ahora vamos a ver los métodos y las clases relacionadas con los modelos.

Todos los componentes que definen modelos tienen las siguientes características:

- 1.- Métodos de acceso en la clase de cada uno de los componentes para el modelo (siendo XXModel la interfaz del modelo que use cada clase):

- `public XXModel getModel()`
- `public void setModel(XXModel modelo)`

En los componentes que tienen el modelo `Document`, estos métodos se sustituyen por `public Document getDocument()` y `public void setDocument(Document d)`.

En aquellos componentes que tienen más de un modelo, los métodos `getModel()` y `setModel(...)`, son usados para acceder al modelo de datos principal. Para los otros modelos existen los métodos siguientes:

- `public XXModel getXXModel()`
- `public void setXXModel(XXModel modelo)`

- 2.- La mayoría de los componentes tienen un constructor al que se le puede pasar el modelo o los modelos del componente (siendo JXX la clase del componente):

- `public JXX(XXModel modelo, ...)`

Si se usa un constructor en el que no se especifica ningún modelo, se instancia un modelo por defecto de la clase `DefaultXXModel` (siendo `XXModel` la interfaz del modelo).

21.3. Varios componentes representan los mismos datos

Para poder implementar el hecho de que varios componentes representan los mismos datos con `NetBeans`, necesitamos utilizar la propiedad `model` de cada uno de los componentes que vayan a compartir los modelos.

Ver la figura 56 de la página 95

En este caso se ha utilizado el modelo por defecto en el componente `JComboBox`, `DefaultComboBoxModel`, y se ha puesto en el componente `JList` el modelo que usa el componente `JComboBox`.

Para compartir modelos:

- 1.- Seleccionamos el componente `JComboBox` en el panel inspector
- 2.- Accedemos a la propiedad `model` en el panel Propiedades
- 3.- Añadimos los elementos por defecto del `JComboBox`
Ver la figura 57 de la página 95
- 4.- Seleccionamos el componente `JList` en el panel inspector
- 5.- Accedemos a la propiedad `model` en el panel Propiedades
- 6.- En el combo seleccionamos `Value from existing component`

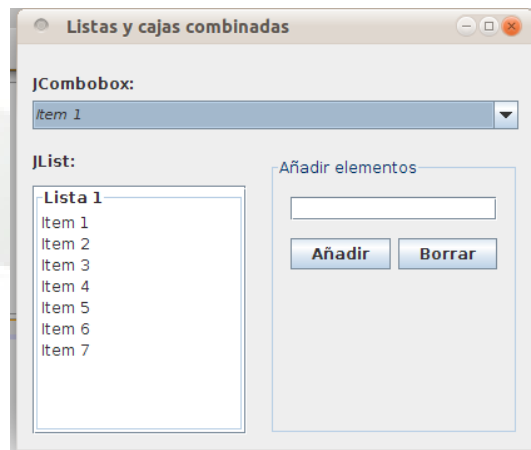


Figura 56: swing5

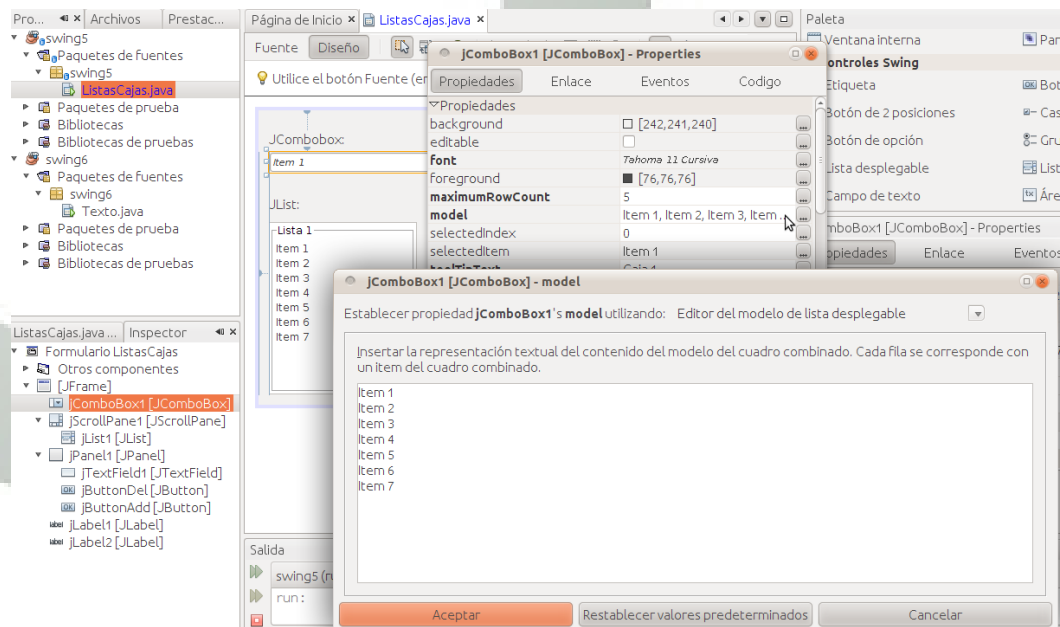


Figura 57: Elementos del modelo por defecto del JComboBox

- 7.- Buscamos el componente JComboBox1 y seleccionamos el modelo.
Ver la figura 58 de la página 96

Si accedemos a la pestaña Fuente vemos que el código que genera (desplegar el Generate Code), es:

Ejemplo código referencia proyecto swing5:

```
private void initComponents() {
    JComboBox1 = new javax.swing.JComboBox();
    ...
}
```

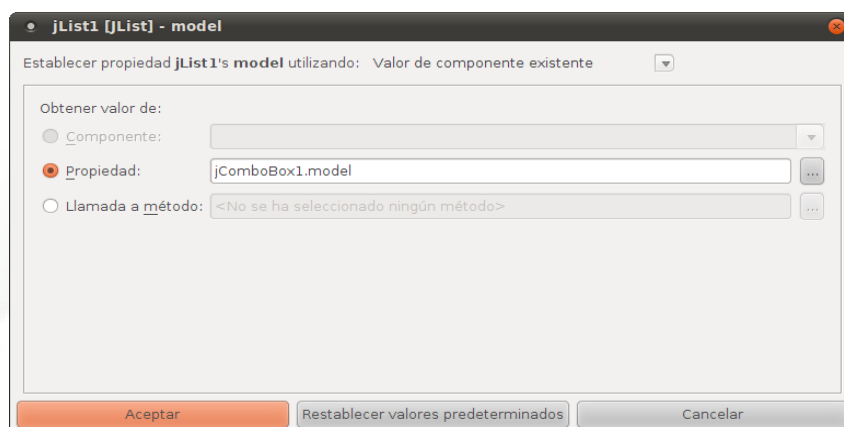



Figura 58: swing5.

```
jComboBox1.setModel(new javax.swing.DefaultComboBoxModel(  
new String[] { "Item 1", "Item 2", "Item 3", "Item 4", "Item 5", "Item 6", "Item 7" }));  
...  
jList1.setModel(jComboBox1.getModel());  
...
```

Se puede ver que la componente `jComboBox1` se le asigna el modelo formado por Strings. Y posteriormente se le asigna el modelo de `jComboBox1` a `jList1`.

En este caso hemos realizado la compartición de dos modelos que son de componentes distintos debido a que el `ComboBoxModel` hereda de `ListModel`.

21.4. Eventos en el modelo

Los cambios en los datos de un componente que suceden en respuesta a una acción del usuario no generan eventos en el propio componente. Esto es debido a que estos eventos se generan en el modelo. Por tanto cuando se esté trabajando con un componente, se tendrá que ver cuáles son los eventos generados en el propio componente y cuáles los generados en el modelo o modelos que usa el componente.

21.5. Construcción de una aplicación con GUI

La idea general de MVC es la separación de los datos y de la interfaz gráfica en dos entidades distintas con el mínimo acoplamiento entre ellas. El desarrollo del código de gestión de datos no debe estar influenciado por la forma en que estos datos se van a visualizar y manejar en la interfaz gráfica de usuario. Como consecuencia de esta situación, el código de gestión de datos no debe hacer ninguna referencia al código de la interfaz gráfica.

La figura 59 de la página 97 muestra la estructura general.

figura 59 de la página 97

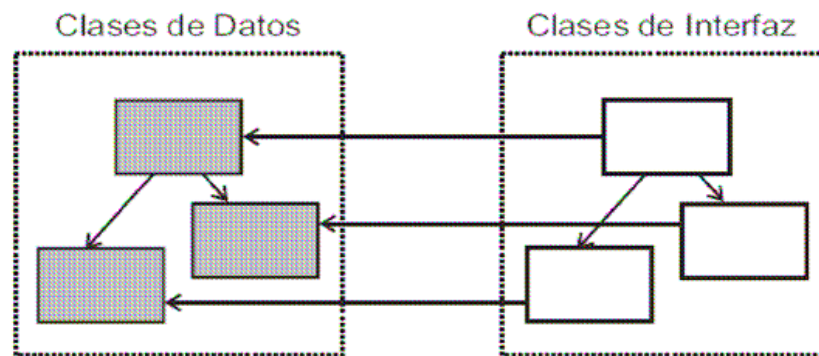


Figura 59: Separación datos - interface en MVC

Con clases de datos se hace referencia a todas aquellas clases que representan los datos de la aplicación y la gestión que se hace de ellos, sin ninguna consideración sobre la interfaz gráfica. Las clases de interfaz serán aquellas que configuran aspectos como botones, ventanas, colores, etc . . .

Para permitir que la interfaz gráfica de usuario cambie cuando los datos cambian; bien porque se han modificado en otro punto de la interfaz o bien de cualquier otro modo, se necesita que las clases de datos generen eventos cada vez que su estado cambia.

Ver la figura 60 de la página 97

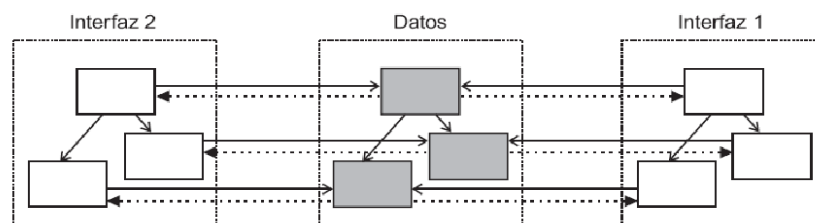


Figura 60: Modelo compartido con dos visualizaciones diferentes

Las clases que se construyan deberán elevar eventos al cambiar de estado si se pretende tener dos visualizaciones consistentes, o bien, si se quiere que ante un cambio en los datos que no provenga de la interfaz gráfica, la visualización cambie.

21.6. Práctica

Utilizando la base del proyecto **swing15**, y utilizando el modelo SimpleTable-Model, crear dos JTables que compartan información, de manera que, cuando en una de ellas se modifique algo automáticamente cambie en la otra. Tal y como se muestra en la figura adjunta. Implementar los eventos Añadir y Borrar fila.

Ver la figura 61 de la página 98

Ejemplos de modelos compartidos

Ejemplo tabla 1

Nombre	Apellidos	Edad	Deporte	Vegetariano
Alberto	Aparicio vila	29	BASKET	<input type="checkbox"/>

Ejemplo tabla 2

Nombre	Apellidos	Edad	Deporte	Vegetariano
Alberto	Aparicio vila	29	BASKET	<input type="checkbox"/>

Añadir Fila Borrar Fila

Figura 61: swing15

22. JNLP

22.1. Java Web Start

Java Web Start es la implementación de referencia de la especificación Java Networking Launching Protocol (JNLP) y está desarrollada por Sun Microsystems, mediante la cual permite arrancar aplicaciones Java que están en un servidor web de aplicaciones comprobando previamente si el cliente tiene la versión actualizada de dicha aplicación. Si no es así descargará la última versión y se ejecutará en local. El arranque de dichas aplicaciones puede ser efectuado mediante enlaces en una página web o bien a través de enlaces en el escritorio cliente. Mediante esta tecnología se asegura que una aplicación es distribuida siempre en su última versión.

Los ficheros que contienen la información sobre donde se encuentra la aplicación, versión, etc. tienen la extensión .jnlp.

Un ejemplo de esta tecnología es la de un servidor web donde se encuentra una página web con enlaces a aplicaciones Java. Cada uno de estos enlaces apuntará a ficheros .jnlp que indicarán la ruta de la aplicación en este u otro servidor. En ese momento arrancará automáticamente Java Web Start y comprobará la seguridad y si el usuario tiene la última versión instalada en su equipo; si no es así, la descargará y ejecutará.

Actualmente Java Web Start viene incluido en el Java Runtime Environment (JRE).

22.2. JNLP

Java Network Launching Protocol (JNLP) es una especificación usada por Java Web Start. Esta especificación, permite tener centralizado en un servidor web un programa, evitando los problemas de distribución e instalación.

La ventaja de JNLP es que no hace falta instalar el programa para ejecutarlo. El inconveniente es que se necesita descargar cada vez para poder ejecutarlo.

22.3. Ejemplo

Ejemplo: La web de las firmas digitales. Pulsar el botón launch.

<https://genera.accv.es/apsc/frontal/enroll.html?action=irPaginaJNLP>

Ver la figura 62 de la página 99

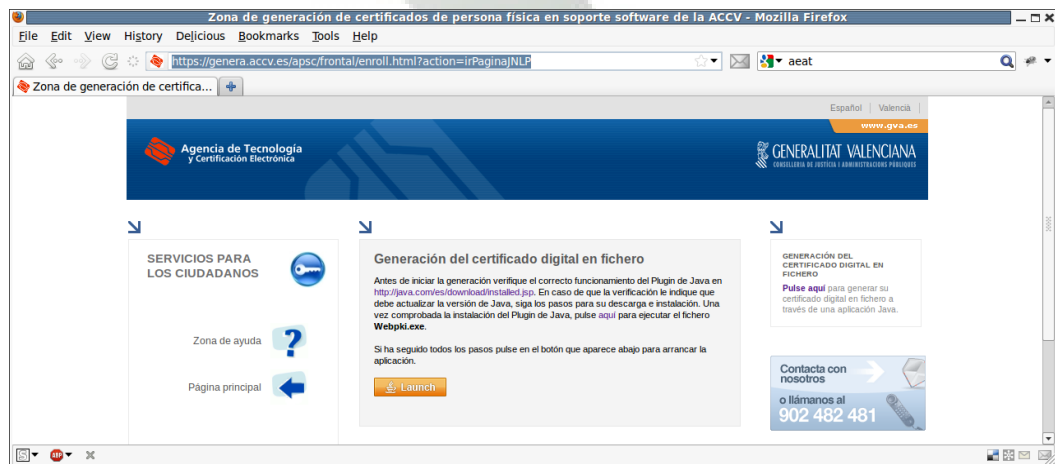


Figura 62: Ejecutar programa desde la web con jnlp

Ver la figura 63 de la página 99

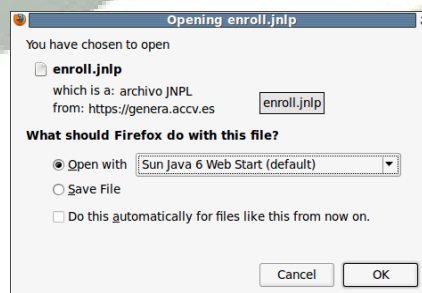


Figura 63: Jnlp pide que aceptemos

22.4. Qué hace

Al instalar cualquier aplicación, normalmente se dan una serie de pasos:

- Se descarga de Internet o se introduce en un medio extraíble (disco compacto, DVD, memoria USB, etc.)
- Se instala.

- Se ejecuta.

El programa javaws permite hacer esto de un modo más fácil y transparente para el usuario, ya que podemos realizar estos pasos simplemente pinchando sobre un enlace mientras estamos utilizando nuestro navegador, de modo que descarga, instalación y ejecución se realizan de modo transparente al usuario.

JavaWS no usa applets, ya que descarga aplicaciones Java normales y necesita, por tanto, de una máquina virtual. Además, viene incluido en el Java Runtime Environment (JRE) de Java desde la versión 1.4.

22.5. Cómo funciona

Cualquier enlace JNLP, al iniciar el proceso de ejecución, pide autorización al usuario. Además, las aplicaciones pueden estar firmadas (firma electrónica) para asegurar el remitente de la aplicación de modo que pueden seguir el modelo de seguridad de la plataforma Java 2 para asegurar la integridad de los datos que obtenemos a través de la red, de forma que no se produzcan ataques de tipo Man in the Middle, DNS cache poisoning, o corrupción de datos.

22.6. Estructura de un archivo JNLP

Un archivo JNLP es un XML especialmente formado compuesto por:

- Una cabecera XML típica:

```
<?xml version="1.0" encoding="conjunto de caracteres"?>
```

Donde conjunto de caracteres" puede ser cualquier conjunto válido: utf-8, ISO-8859-1...

- Una ruta predeterminada para que los archivos puedan ser llamados desde una ruta relativa.

```
<jnlp spec="1.0+" codebase="http://URL/directorio/del/programa"  
  href="NombreDelArchivoJNLP.jnlp">
```

- Una o más etiquetas information en el que se colocan varias informaciones (ver ejemplo).
- Una etiqueta security (con información variada).
- Una etiqueta resources (con información variada).
- Una etiqueta application-desc con la clase predeterminada a ejecutar.

22.7. Ejemplo

El siguiente es un breve ejemplo de un archivo típico JNLP. No incluye todas las posibles opciones.

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+"
codebase="http://es.wikipedia.org/wiki/JNLP:8080/ElPrograma/"
href="jnlp.jnlp">

<information>
  <title>Ejemplo de un JNLP wikipédico</title>
  <vendor>Anónimo</vendor>
  <homepage href="http://es.wikipedia.org/Portada" />
  <description> Ejemplo de un JNLP muy wikipédico</description>
  <description kind="short">
    Esta es una breve información, repito, muy wikipédica.
  </description>
  <icon href="NombreImagen.jpg" />
  <offline-allowed />
</information>

<security>
  <all-permissions />
</security>

<resources>
  <j2se version="1.4+" />
  <jar href="aplicación.jar" />
  <jar href="lib1.jar" />
  <jar href="lib2.jar" />
</resources>

<application-desc main-class="org.wikipedia.es.JNLP" />
</jnlp>
```

22.8. JNLP con Netbeans

Para que netbeans construye el fichero jbnlp tenemos que sacar el menú contextual del proyecto y seleccionamos propiedades. Activaremos Aplicación - Inicio Web como muestra la imagen.

Ver la figura 64 de la página 101

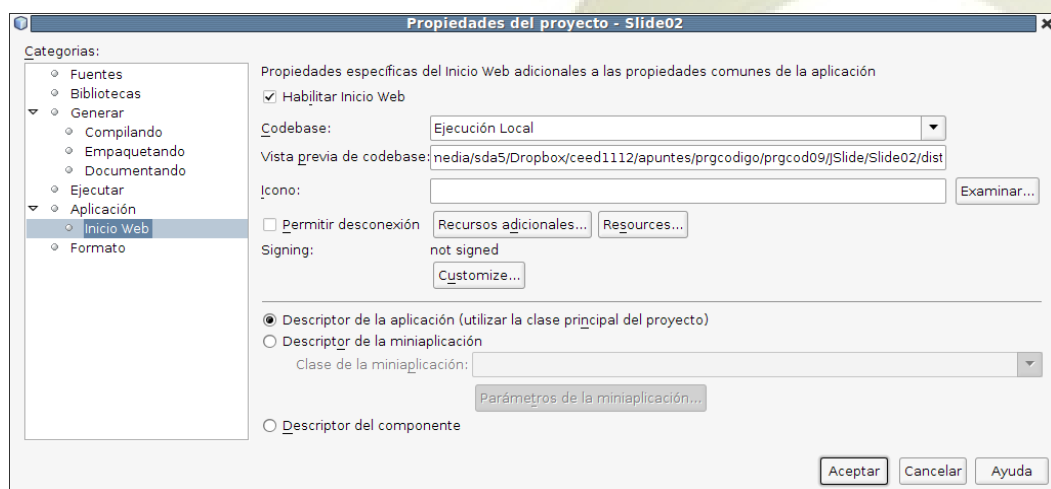


Figura 64: Propiedades del Proyecto

Seguidamente seleccionamos el Menú - Ejecutar - Main Project para que genere el fichero html y jnlp en la carpeta dist del proyecto.

El fichero launch.html deberá verse con un botón para lanzar la aplicación.

Ver la figura 65 de la página 102

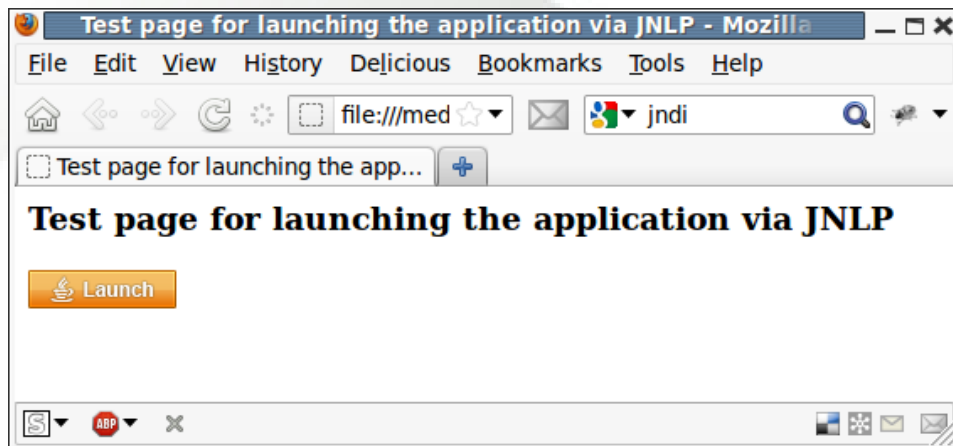


Figura 65: launch.html

Puediéndose ver su ejecución después de pulse el botón launch, No es necesario abrir el fichero html, directamente dándole a ejecutar en Netbeans se verá el mismo resultado.

Ver la figura 66 de la página 102

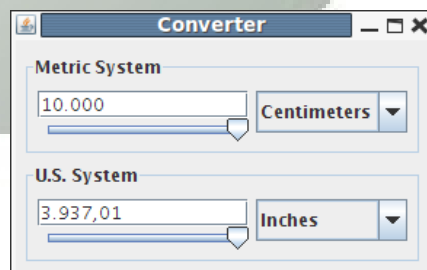


Figura 66: JSlide02 en ejecución

22.9. Práctica

Abrir el proyecto Slide02 y crear el fichero html y jnlp con netbeans. Después abrir con el navegador y ejecutar la aplicación.

23. JAPPLET

El lenguaje Java se puede usar para crear dos tipos de programas: los applets y las aplicaciones. Un applet es un elemento más de una página web, como una

imagen o una porción de texto. Cuando el navegador carga la página web, el applet insertado en dicha página se carga y se ejecuta.

Mientras que un applet java puede transmitirse por la red Internet una aplicación reside en el disco duro local. Una aplicación Java es como cualquier otra que está instalada en el ordenador. La otra diferencia es que un applet java no está autorizado a acceder a archivos o directorios del ordenador cliente si no es un applet completamente fiable.

Es importante no confundir applet java, con javascript, son dos cosas diferentes. Para poder ver los applets java en el navegador hay que tener activada esta opción en el navegador y tener instalado los plugins para java.

23.1. Primer Applet

Para crear un applet tenemos que definir una clase denominada Applet1 derivada de Applet. La primera sentencia import nos proporciona información acerca de las clases del paquete applet. Dicho paquete contiene las clases necesarias para crear applets que se ejecutan en la ventana del navegador, entre las cuales está la clase base Applet.

```
import java.applet.*;
public class Applet1 extends Applet {
}
```

El siguiente paso es dar funcionalidad a la clase, definir nuestras propias funciones miembro o redefinir funciones de la clase base Applet.

Definimos la función init para establecer el color de fondo del applet mediante setBackground. La función init se llama cuando se carga el applet.

```
public class Applet1 extends Applet {
    public void init(){
        setBackground(Color.white);
    }
    //...
}
```

A continuación, vamos a mostrar un mensaje, para ello definimos el método paint. El método paint nos suministra el contexto gráfico g, un objeto de la clase Graphics con el cual podemos dibujar en el área de trabajo del componente llamando desde dicho objeto g a las funciones definidas en la clase Graphics.

Para mostrar un mensaje, llamamos desde el objeto g a la función miembro drawString, el primer argumento es el string que deseamos mostrar, y los dos números indican las coordenadas de la línea base del primer carácter.

```
../prgcodigo/prgcod09/applet1/Applet1.java
```

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

```
import java.applet.*;

public class Applet1 extends Applet {
    public void init(){
        setBackground(Color.white);
    }
    public void paint(Graphics g){
        g.drawString("Primer applet", 10, 25);
    }
}
```

Un applet, no es como una aplicación que tiene un método main. El applet está insertado en una página web que se muestra en la ventana del navegador. El navegador toma el control del applet llamando a algunos de sus métodos, uno de estos es el método paint que se llama cada vez que se necesita mostrar el applet en la ventana del navegador.

Cuando el applet se carga, el navegador llama a su método init. En este método el programador realiza tareas de inicialización, por ejemplo, establecer las propiedades de los controles, disponerlos en el applet, cargar imágenes, etc.

El método init se llama una sola vez. Después, el navegador llama al método paint.

A continuación, se llama al método start. Este método se llama cada vez que se accede a la página que contiene el applet. Esto quiere decir, que cuando dejamos la página web que contiene el applet y regresamos de nuevo pulsando en el botón "hacia atrás" el método start vuelve a llamarse de nuevo, pero no se llama el método init.

Cuando dejamos la página web que contiene el applet, por ejemplo, pulsando en un enlace, se llama al método stop.

Finalmente, cuando salimos del navegador se llama al método destroy.

23.1.1. Insertando un applet en una página web

Las etiquetas HTML como <H1>, <TABLE>, , etc. señalan el tamaño y la disposición del texto y las figuras en la ventana del navegador. Cuando Sun Microsystems desarrolló el lenguaje Java, se añadió la etiqueta que permite insertar applets en las páginas web. Como otras etiquetas tiene un comienzo <APPLET> y un final señalado por </APPLET>.

../prgcodigo/prgcod09/applet1/Applet1.html

```
<HTML>
<HEAD>
<TITLE>
HTML Test Page
</TITLE>
</HEAD>
<BODY>
Applet1 will appear below in a Java enabled browser.<BR>
<APPLET
  CODEBASE = "."
  CODE      = "Applet1.class"
  NAME      = "TestApplet"
  WIDTH     = 400
  HEIGHT    = 300
```



```
HSPACE = 0
VSPACE = 0
ALIGN = middle
>
</APPLET>
</BODY>
</HTML>
```

Ver la figura 67 de la página 105



Figura 67: Applet1

Cuando se compila el applet se producen archivos cuya extensión es .class. Uno de estos archivos es el que resulta de la compilación de la clase que describe el applet, en nuestro caso Applet1.class .

Dentro de la etiqueta applet el parámetro más importante es CODE que señala el nombre del archivo cuya extensión es .class, y cuyo nombre coincide con el de la clase que describe el applet.

Los valores de los parámetros WIDTH y HEIGHT determinan las dimensiones del applet. En este caso el applet tiene una anchura de 400 y una altura de 300.

El nombre del applet, parámetro NAME, es importante cuando se pretende comunicar los applets insertados en una página web.

23.2. Appletviewer

El visualizador de applets (appletviewer) es una aplicación que permite ver en funcionamiento applets, sin necesidad de la utilización de un navegador.

Se ejecuta desde la línea de ordenes de DOS/LINUX y se le pasa como argumento la página HTML que invoca al applet.

En nuestro applet tenemos la siguiente función:

```
appletviewer Applet1.html
```


23.3. Applet con Netbeans

Cuando el proyecto es complejo, al compilarlo se crean varios archivos .class en el mismo subdirectorio. Resulta engorroso trasladarlos desde nuestro ordenador al servidor cuando publicamos las páginas web. Se corre el peligro de mezclar los archivos o perder alguno por el camino. Para facilitar esta tarea, se puede comprimir todos los archivos .class resultantes del proceso de compilación de un proyecto en un único archivo cuya extensión es .jar, mediante un asistente del IDE.

En la siguiente figura se muestra un ejemplo de un JApplet:

Ver la figura 68 de la página 106



Figura 68: Swing9

23.3.1. Ver el applet en el navegador

La clase Visor.java, , está dentro del paquete swing9. Al compilar se creará un fichero llamado Visor.class. Para ver el applet en el navegador deberemos crear el proyecto .jar, el cual creará el fichero swing9.jar,

Deberemos crear la página web, teniendo en la misma carpeta el .jar. El fichero .html en el apartado APPLET deberá tener el siguiente código:

```
CODE      = "swing9.Visor.class"
ARCHIVE   = "swing9.jar"
```

El resto de la página web es igual al ejemplo Applet1.html visto anteriormente.

Ver la figura 69 de la página 107

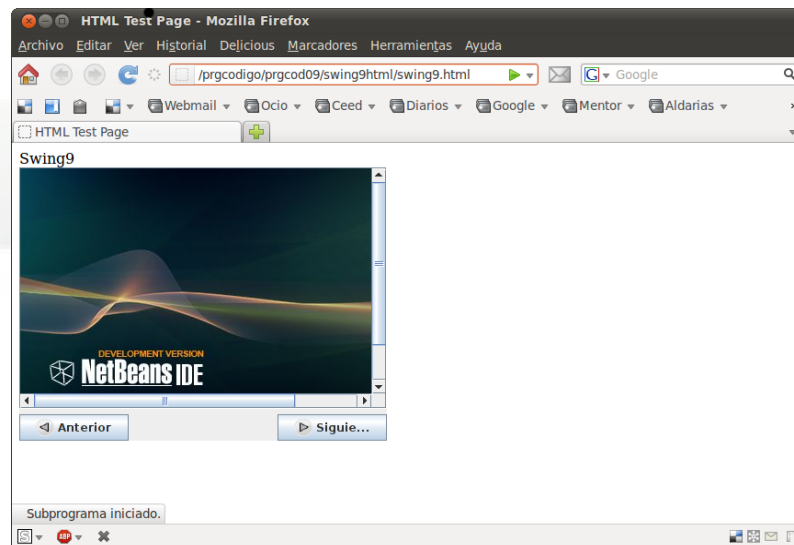


Figura 69: Swing9.html

23.3.2. Práctica

Abrir el proyecto swing9 y sobre la clase Visor.java botón derecha ratón actionPerformed en el botón Siguiente y Anterior. Integrar el applet en una página html. Ejercutar Fihcero (Run File) sobre Visor.java . Ver los eventos.

23.3.3. API

Los métodos más comunes en el trabajo con contenedores de alto nivel eran el tema de posicionamiento, distribución del Layout y adición componentes, que con el uso de Matisse/NetBeans se ha simplificado al entorno gráfico.

Para mas información del API completa:

- <http://download.oracle.com/javase/6/docs/api/javax/swing/JFrame.html>
- <http://download.oracle.com/javase/6/docs/api/javax/swing/JDialog.html>
- <http://download.oracle.com/javase/6/docs/api/javax/swing/JApplet.html>

23.3.4. Eventos

El conjunto de eventos que podemos capturar en este tipo de componentes los podemos ver en la pestaña de events habiendo seleccionado el componente el la pestaña inspector, o bien botón de la derecha ratón pestaña inspector sobre el componente y accedemos a events. Especial hincapié en los eventos de tipo Window WindowActivated, WindowClose, etc, ...

24. ORIENTACIÓN

24.1. Objetivos

Los objetivos son:

- Conocer las diferencias entre el entorno consola e interface gráfica de usuario (IGU).
- Crear una pequeña aplicación gráfica con un componente básico.
- Saber asociar una acción a un botón.
- Saber recoger el texto rellenado en un campo de texto.
- Conocer como se organizan las clases para trabajar las IGU.
- Utilizar herramientas del entorno de desarrollo para crear interface gráficos de usuario simples.
- Saber programar controladores de eventos.
- Saber realizar programas que utilicen interfaces gráficos para la entrada salida de información.

24.2. Prácticas

- 1.- MiJFrame. Ejemplo HolaMundo. JFrame y JLabel.
- 2.- MiJTextField. Ejemplo con JFrame, JPanel, setLayout, JTextField.
- 3.- MiJButton. Ejemplo con JButton, Eventos.
- 4.- MiMvc. Formularios.

24.3. Código Fuente

El código fuente se puede encontrar en control de versiones.

<https://bitbucket.org/1516ceed1/>

Ejemplos:

`git clone https://bitbucket.org/1516ceed1/1415ceed1prgt7e10`

Ejercicios:

`git clone https://bitbucket.org/1516ceed1/1415ceed1prgt7e20`

24.4. Software

Se recomienda el siguiente software:

- Sistema Operativo: Linux
- IDE: Netbeans.

24.5. Datos bibliográficos

Se verá desde el punto 6.5 del libro [MP11] del **Tema 6. Lectura y escritura de Información**. Sólo para consulta ya que el libro no toca Netbeans.

24.6. Recursos en Internet

- Video Tutorial Programación pantalla de Acceso(Login) en JAVA Netbeans.
<http://www.youtube.com/watch?v=ayCeQh8IGgA>
- Calculadora en java 7 con Netbeans - Diseño y programación
<http://www.youtube.com/watch?v=PkHqUY82qU8>
- Creating a GUI With JFC/Swing: Table of Contents (Documentación oficial de oracle)
<http://docs.oracle.com/javase/tutorial/uiswing/TOC.html>
- Hola Mundo con Swing
<http://lineadecodigo.com/tag/hola-mundo-con-swing/>
- Cerrar una Ventana con Java Swing
<http://lineadecodigo.com/tag/addwindowlistener/>
- JOptionPane y diálogos modales
http://chuwiki.chuidiang.org/index.php?title=JOptionPane_y_di%C3%A1logos_modales
- ActionListener
<http://chuwiki.chuidiang.org/index.php?title=ActionListener>
- Patrones de Diseño, MVC. <http://codejavu.blogspot.com.es/2013/06/que-son-los-patrones-de-diseno.html>
- Setbounds. <https://iloveyouinformatica.wordpress.com/2011/05/15/manejo-de-setbounds-en-java/>

Maven:

- Using NetBeans with Apache Maven <http://wiki.netbeans.org/Maven>

- Cómo transformar un proyecto Netbeans en uno Maven <http://magmax.org/blog/netbeans-a-maven/>
- Como crear un proyecto con maven. Hola Mundo con maven. <http://sixservix.com/blog/tech/2012/09/14/como-crear-un-proyecto-en-maven/>

24.7. Temporalización

Esta materia se impartirá a distancia en:

DESCRIPCION	CANTIDAD
Total de horas del tema	19
Total de semanas	2
Total de horas por semana	9,5
Total de horas de tutorías grupales por semana	2
Total de horas del curso	230

BIBLIOGRAFÍA

- [DEI01] H. M. DEITEL. *Advanced Java 2 Platform*. Prentice Hall, 1 edition, 2001. ISBN 0-13-034151-7.
- [FIS05] PAUL FISCHER. *An Introduction to Graphical User Interfaces with Java Swing*. Pearson Education, 1 edition, 2005. ISBN 0321 22070 6.
- [FQ08] AGUSTIN FROUTE QUINTAS. *Java 2. Manual del Usuario y Tutorial*. 5 edition, 2008. ISBN: 978-84-7897-875-5. PRECIO:40EUR. JAVA 2SE.
- [HOL00] ESTEVEN HOLZNER. *La biblia de Java 2*. ANAYA MULTIMEDIA, 1 edition, 2000. ISBN: 8441510377. PRECIO: 52EUR. BASICO, MUY COMPLETO.
- [MP11] JUAN CARLOS MORENO PEREZ. *Programación*. Ra-ma, 1 edition, 2011. ISBN: 978-84-9964-088-4. LIBRO ALUMNO CICLOS. PRECIO: 35 EUR.