Práctica 3. Interfaces gráficas de usuario con Java Swing

Entornos de Usuario

10 de octubre de 2018

Objetivos

El objetivo principal de esta práctica introducir los elementos de una Interfaz Gráfica de Usuario (en adelante, IGU) a través de la biblioteca de clases Swing. Esta tarea se realizará a través del desarrollo de una aplicación sencilla, pero que sin embargo incorpora muchos de los elementos que es posible encontrar en una IGU. La aplicación a desarrollar es una calculadora conversora de monedas, que permite introducir la cantidad a convertir y la tasa de cambio.
A través de esta práctica, el estudiante puede adquirir las siguientes habilidades y conocimientos:

- Adquirir un conocimiento profundo de la API Swing.
- Implementar una aplicación gráfica con Java Swing.
- Entender cómo Swing gestiona los eventos de usuario y cómo se asocian y gestionan a través de una aplicación.
- Entender el Modelo Vista Controlador (MVC).
- Diseñar e implementar una aplicación utilizando el Modelo Vista Controlador (MVC).

Índice

Índice de figuras

Índice de listados

1 Introducción

En esta práctica se van a uilizar varios de los elementos habituales de una Interfaz Gráfica de Usuario utilizando la biblioteca de clases Swing. En concreto, la interfaz gráfica de Usuario utilizará:

- Los componentes y contenedores gráficos que nos sirven para crear las ventanas, botones, etiquetas, etc. típicos de una aplicación gráfica; estos componentes son clases englobadas en el paquete `javax.swing`.
- Los componentes de menú: JMenuBar, JMenu y JMenuItem, clases englobadas también en el paquete `javax.swing`.
- Los eventos y la gestión de eventos.
- Los Gestores de Disposición (*Layout Managers*).

En los subapartados siguientes revisaremos con detalle estos elementos.

1.1 Componentes y Contenedores

La interfaz gráfica de usuario (IGU) está construida en base a elementos gráficos básicos denominados **Componentes**. Típicos ejemplos de estos Componentes son los botones, barras de desplazamiento, etiquetas, listas, cajas de selección o campos de texto.

Los Componentes permiten al usuario interactuar con la aplicación. En la biblioteca Swing, todos los Componentes de la interfaz gráfica de usuario son instancias de la clase `JComponent` o de una clase descendiente de ella.

Los Componentes no se encuentran aislados, sino agrupados dentro de **Contenedores**. Los Contenedores contienen y organizan la situación de los Componentes. Los Contenedores son en sí mismos Componentes y como tales pueden ser situados dentro de otros Contenedores. En la biblioteca Swing, todos los Contenedores son instancias de la clase `JContainer` o una clase descendiente de ella.

1.2 Eventos

Con el término evento se pretende dar a entender la acción que se desencadena cuando el usuario interacciona con una aplicación gráfica: pinchar con el ratón sobre un botón, seleccionar una opción de menú, etc. Un componente envía un mensajes de aviso cuando el usuario interacciona con él. Es responsabilidad del programador escribir código que recoja ese aviso y actúe en consecuencia (es decir, que gestione de forma adecuada el evento de acuerdo con las necesidades de la aplicación).

Por ejemplo, si el usuario pincha con el raton sobre un botón **Salir** o **Cerrar** , es necesario escribir código que recoja el aviso que envía el ratón y cerrar la ventana (o la aplicación). Cada manejador de eventos consta de tres piezas de código:

- En la declaración de la clase manejadora del evento, una línea de código especifica que la clase o bien implementa una interfaz escuchadora o bien extiende una clase que implementa una interfaz escuchadora. Por ejemplo:

```
1  public class MyClass implements ActionListener
2  {
3    // Codigo de la clase ...
4  }
```

- Otra línea de código registra en uno o más componentes un objeto de la clase que implementa la interfaz escuchadora. Por ejemplo:

```
1  someComponent.addActionListener(objectOfMyClass);
```

- La clase manejadora del evento tiene código que implementa los métodos de la interfaz escuchadora. Por ejemplo:

```
1  public void actionPerformed(ActionEvent e)
2  {
3    // Codigo que ejecuta las acciones apropiadas en respuesta al evento
4  }
```

1.3 Gestores de Disposición

La manera de construir interfaces gráficas en Java es creando contenedores que contienen componentes. La manera en que los componentes se distribuyen en el interior del contenedor se controla mediante estos Layaouts o Gestores de Disposición. Java dispone de varios, en la actual versión, tal como se muestra en la imagen.

¿Por qué Java proporciona estos esquemas predefinidos de disposición de componentes? La razón es simple: imaginemos que deseamos agrupar objetos de distinto tamaño en celdas de una rejilla virtual: si confiados en nuestro conocimiento de un sistema gráfico determinado, y codificamos a mano tal disposición, deberemos prever el redimensionamiento de la ventana, su repintado cuando sea cubierto por otra ventana, etc., además de todas las cuestiones relacionadas con un posible cambio de plataforma (uno nunca sabe a donde van a ir a parar nuestras ventanas). Sigamos imaginando, ahora, que un hábil equipo de desarrollo ha previsto las disposiciones gráficas más usadas y ha creado un gestor para cada una de tales configuraciones, que se ocupará, de forma transparente para nosotros, de todas esas cuitas de formatos. Bien, pues estos gestores son instancias de las distintas clases derivadas de Layout Manager y que se utilizan en los contenedores de nuestras aplicaciones.

Los Layouts liberan al programador de tener que preocuparse de dónde ubicar cada uno de los componentes cuando una ventana es redimensionada o cuando una ventana es refrescada o cuando una ventana es llevada a una plataforma que maneja un sistema de coordenadas diferente.

Mostraremos a continuación con cierto detalle los *Layout Managers* conocidos, algunos de los cuales que podremos usar a lo largo de esta práctica.

1.3.1 FlowLayout

Es el más simple y el que se utiliza por defecto en todos los Paneles si no se fuerza el uso de alguno de los otros. Los Componentes añadidos a un Panel con FlowLayout se encadenan en forma de lista. La cadena es horizontal, de izquierda a derecha, y se puede seleccionar el espaciado entre cada Componente.

1.3.2 BorderLayout

La composición BorderLayout (de borde) proporciona un esquema más complejo de colocación de los Componentes en un panel. La composición utiliza cinco zonas para colocar los Componentes sobre ellas: Norte, Sur, Este, Oeste y Centro. Es el layout o composición que se utilizan por defecto JFrame y JDialog. El Norte ocupa la parte superior del panel, el Este ocupa el lado derecho, Sur la zona inferior y Oeste el lado izquierdo. Centro representa el resto que queda, una vez que se hayan rellenado las otras cuatro partes.

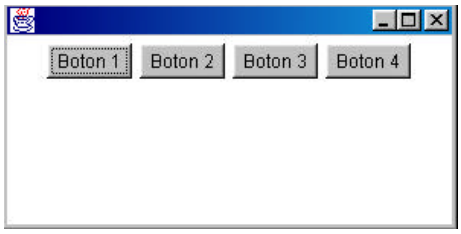

Figura 1: Ejemplo Flowlayout

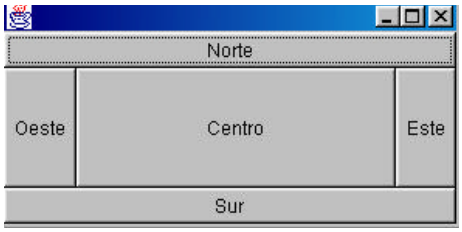

Figura 2: Ejemplo Borderlayout

1.3.3 GridLayout

La composición GridLayout proporciona gran flexibilidad para situar Componentes. El layout se crea con un número de filas y columnas y los Componentes van dentro de las celdas de la tabla así definida. En la figura siguiente se muestra un panel que usa este tipo de composición para posicionar seis botones en su interior, con tres filas y dos columnas que crearán las seis celdas necesarias para albergar los botones:

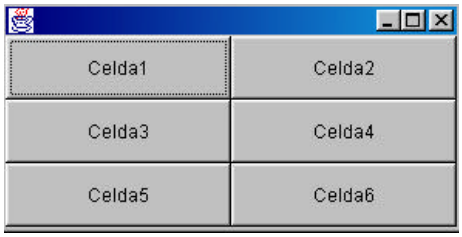

Figura 3: Ejemplo Gridlayout

1.3.4 Cardlayout

Este es el tipo de composición que se utiliza cuando se necesita una zona de la ventana que permita colocar distintos Componentes en cada momento. Este layout suele ir asociado con botones de lista (Choice), de tal modo que cada selección determina el panel (grupo de componentes) que se presentarán. En la figura siguiente mostramos el efecto de la selección sobre la apariencia de la ventana que contiene el panel con la composición CardLayout:

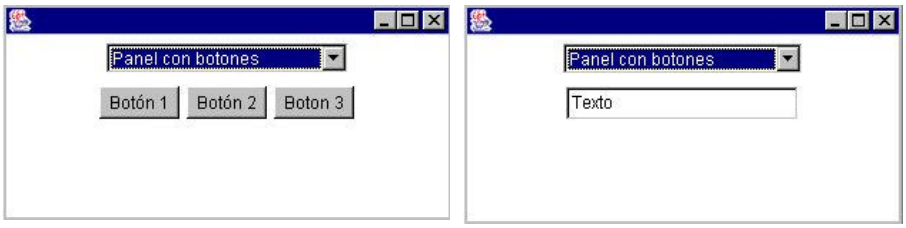

Figura 4: Ejemplo Cardlayout (CardBotones y CardTexto)

1.3.5 BoxLayout

El BoxLayout es un gestor que permite distibuir varios componentes vertical u horizontalmente. Los componentes siempre mantienen esa distribución, de modo que una distribución vertical siempre se mantendrá vertical aunque se cambie el tamaño del JFrame. El constructor requiere un parámetro para especificar el tipo de organización del gestor. Hay varias opciones, pero nos interesan fundamentalmente dos:

- **X_AXIS**. Los componentes se distribuyen horizontalmente de izquierda a derecha.
- **Y_AXIS**. Los componentes se distribuyen verticalmente de arriba a abajo.

Sea cual sea la dirección elegida, los componentes se organizan en el mismo orden en el que son añadidos al contenedor.

2 La calculadora conversora

El trabajo de esta práctica consiste en implementar la interfaz gráfica de usuario de una calculadora conversora cuyo ejecutable se ha proporcionado junto con el guión. Dicha calculadora, haciendo uso del interfaz gráfico, permite introducir la cantidad a convertir y la tasa de cambio tal como se muestra en las figuras 6 y 7. Para ello utilizaremos el modelo vista controlador (MVC) introducido en la segunda práctica.

2.1 La arquitectura Modelo Vista Controlador (MVC)

Esta arquitectura es muy útil para el diseño de programas interactivos con interfaz gráfica. La principal característica es que introduce una separación de la lógica de negocio de la interfaz de usuario. Esto proporciona dos características muy importantes: 1) facilita la evolución por separarado de ambos aspectos y 2) incrementa la reutilización y flexibilidad de los programas.

El flujo de control del MVC es:

1. El usuario realiza una acción en la interfaz
2. El controlador trata el evento de entrada (Previamente se ha registrado)
3. El controlador notifica al modelo la acción del usuario, lo que puede implicar un cambio del estado del modelo (si no es una mera consulta)
4. Se genera una nueva vista. La vista toma los datos del modelo
5. El modelo no tiene conocimiento directo de la vista
6. La interfaz de usuario espera otra interacción del usuario, que comenzará otro nuevo ciclo

¿Cómo vamos a aplicar el modelo vista controlador (MVC) a nuestra calculadora conversora? Para aplicar el MVC a nuestra calculadora es recomendable utilizar tres clases:

- La clase `EuroConversorView`, que describe la interfaz de usuario: la pantalla de visualización (que implementará la clase `DisplayPanel`), el teclado (`NumberPanel`), el botón para convertir (`OperationPanel`), el botón para resetear la pantalla (`ClearPanel`) y el menú (que se implementa en `EuroconversorMenu`).
- La clase `EuroConversorModel`, es la que modela el comportamiento del objeto real, la conversión de la cantidad introducida. Esta clase se suministra completa, pero deberás leer y comprender el código para aprender a usarla.
- La clase `EuroConversorController`, que será la intermediaria entre la vista y el modelo. Se encargará de tratar todo los eventos que se produzcan en la vista (responder a los *clicks* sobre los botones, Salir de un menú, etc), mantener consistente nuestra calculadora con respecto a las modificaciones que produce el usuario sobre ella a través del interfaz. También se encargará de notificar al modelo la cantidad a convertir solicitada por el usuario y de mostrar y actualizar en la vista las salidas producidas por éste. El controlador no aporta funcionalidad, sólo aporta el código necesario para que el modelo y la vista se comuniquen.

Con objeto de simplificar el desarrollo de la práctica, junto con el enunciado se suministra el fichero `EuroConversor.zip` con el esqueleto básico del proyecto. Descarge el fichero y descomprímalo dentro de la carpeta en la que NetBeans almacena sus proyectos (generalmente `NetBeansProjects` en el directorio personal del usuario). Después acceda al proyecto desde NetBeans con **File** → **Open Project...** . En la figura 5 se muestra la estructura completa de paquetes y ficheros del proyecto que describiremos con detalle en los siguientes apartados.

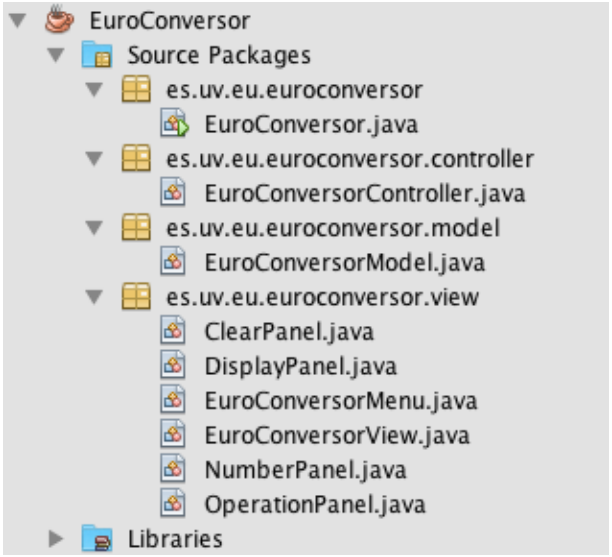

Figura 5: Estructura del proyecto `EuroConversor` tal y como se suministra para la realización de la práctica

2.2 La clase `EuroConversor.java`

Esta es la clase principal de nuestro proyecto. A través de su método `main` se encarga de instanciar el modelo, la vista y el controlador y hacer visible la interfaz de usuario. El código de la clase se suministra completo (no es necesario modificarlo) y tiene una estructura muy similar a la que ya vimos para este tipo de aplicaciones en la segunda práctica. Esta clase se encuentran en el paquete `es.uv.es.euroconversor`

2.3 El modelo: la clase `EuroConversorModel.java`

Esta es la clase principal del modelo con el que vamos a trabajar en esta práctica. El código de la clase se suministra completo (no es necesario modificarlo). Es necesario entender el código suministrado para aprender a usarla. Esta clase se encuentran en el paquete `es.uv.es.euroconversor.model`

2.4 La vista: clase `EuroConversorView.java` y clases asociadas

Esta clase junto con el resto de clases que se encuentran en el paquete `es.uv.es.calculadora.view`, describen la IGU de la aplicación.

Para entender más fácilmente lo que se espera de esta clase, en la figura 6 se muestra una captura de pantalla de la interface gráfica generada por la clase. En la imagen derecha de la figura 6 se han remarcado en color rojo las distintas áreas que componen la interfaz.

La clase `EuroConversorView` es la que implementa la ventana principal. Esta clase deriva de la clase base de Swing `JFrame`. En la barra de ventana aparece el nombre de la aplicación. Dentro de esta clase se crearán el resto de los elementos que componen la interface:

- La barra de menu de la aplicacion. Este menú sólo cuenta con:
  - La opción **Calculadora**
  - La subopción **Calculadora** → **Salir**

  - La subopción **Calculadora** → **Change Rate**

    Cuando se selecciona la subopción **Change Rate** se abre una ventana de diálogo tal como se muestra en la figura 7 que permite al usuario, introducir la tasa de cambio.

    La clase `CalculadoraMenu` se encarga de crear este menú.

- La clase `DisplayPanel` define la zona de la pantalla en donde se muestra los datos de la calculadora. Está formada por dos zonas:

  - La parte superior muestra el resultado de la última conversión o la cantidad que está introduciendo el usuario y desea convertir. Para ello se utiliza un tamaño de letra grande, fácil de leer.
  - Una línea inferior que muestra la tasa de cambio que está utilizando la calculadora de modo que el usuario puede apreciar en todo momento dicha tasa.

- La clase `NumberPanel` implementa el teclado numérico que contiene los dígitos, el punto decimal y la operación "C" (borrar dígito). Está estructurado como un `GridLAyout` de tamaño 3×4.

- La clase `OperationPanel` contiene botón que convierte la cantidad introducida por el usuario al nuevo cambio.

- Por último, la clase `ClearPanel` únicamente cuenta con el botón **CLEAR** que borra (pone a 0) la pantalla de la calculadora y hace un `reset` del estado interno del modelo.

Todas las clases cuyo nombre finaliza con la palabra `Panel` son clases que derivan de la clase base `JPanel`. La clase `CalculadoraMenu` extiende la clase `JMenuBar`.

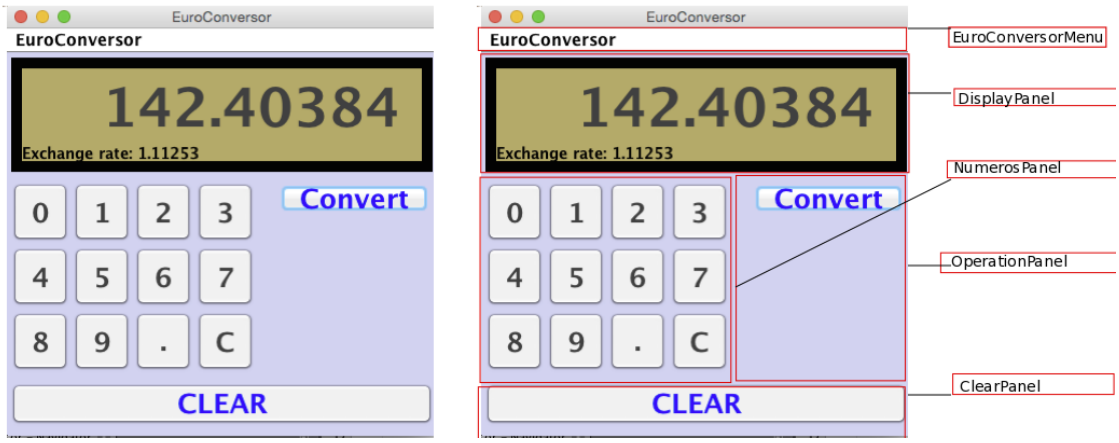

Figura 6: Captura de pantalla de la interface gráfica principal de la calculadora.

Nota: Antes de implementar la vista indica que tipo de gestores de disposición (layouts) y que componentes vas a utilizar para cada una de las clases anteriores que derivan de la clase JPanel y la clase JFrame y que componen la vista.

Figura 7: Captura de pantalla de la ventana de diálogo que permite introducir la tasa de cambio

2.5 El controlador: la clase `CalculadoraControlador`

Para que la aplicación sea capaz de responder a los eventos del usuario, es necesario que todos los componentes de la IGU que hemos descrito en el apartado anterior cuenten con un objeto auditor o "escuchador" de eventos. Los escuchadores deben implementar la interface `ActionListener` y proporcionar el método:

```
1 actionPerformed(ActionEvent evento)
2 {
3    ...
4 }
```

La misión del controlador es proporcionar el código del "escuchador" o de los "escuchadores" (en nuestro caso, como clases empotradas) y asegurarse de que cada clase de la vista tenga registrado (asociado) un objeto auditor de eventos. Esta clase se encuentran en el paquete `es.uv.es.euroconversor.controller`

2.6 Tareas

Esta práctica se desarrollará en dos sesiones de laboratorio, con la siguiente distribución de trabajo:

- **Sesión 1**. Se dedicará a completar la vista de la aplicación (clase `EuroConversorView` y resto de clases asociadas). Al finalizar la primera sesión de laboratorio, la aplicación deberá mostrar la interface gráfica aunque sin implementar ninguna funcionalidad.
- **Sesión 2**. Esta sesión se dedicará al desarrollo del controlador (clase `EuroConversorController`). Al final de esta sesión, la aplicación debería estar completa y ser funcional.

2.7 Entrega

La entrega se realizará en Aula Virtual antes del comienzo de la sesión de la siguiente práctica.