Programación Orientada a Objetos
Sesión 9: Aplicaciones Visuales mediante Interfaces Gráficos de Usuario en Java
Curso 2016-17

Luis Daniel Hernández Molinero, Santiago Paredes Moreno y Gregorio Martínez Pérez
{ldaniel, chapu, gregorio}@um.es

Índice

Parte I
Teoría

1. Introducción

La mayoría de programas modernos son aplicaciones visuales, es decir, presentan al usuario botones, campos de texto, listas desplegables y otros componentes típicos agrupados en paneles dentro de lo que se conoce como ventanas. La parte del programa que se encarga de manejar todo el apartado visual se denomina **Interface Gráfico de Usuario** (o **GUI** por las siglas en inglés **Graphical User Interface**) y su labor fundamental es permitir al usuario interactuar con el ordenador utilizando el ratón, el teclado y la pantalla.

Java incluye dos bibliotecas de clases para hacer aplicaciones visuales llamadas **JFC (Java Fundation Classes)** y **Swing**. Cada componente de una aplicación visual (ventana, botón, etc) será un objeto de alguna clase perteneciente a JFC o a Swing. En todos los casos el funcionamiento es siempre el mismo: por un lado, se determina el aspecto visual de los distintos componentes dándole valores a sus propiedades; y por otro lado, si llamamos **evento** a cualquier interacción que el usuario es capaz de hacer usando el ratón o el teclado sobre los distintos componentes, será necesario programar el comportamiento que el programa deberá tener al producirse los distintos eventos.

Evidentemente, el aspecto visual o GUI de una aplicación es sólo la mitad del programa; la otra mitad es la encargada de llevar a cabo la tarea fundamental del mismo y la denominamos núcleo de la aplicación. Los resultados obtenidos por el núcleo se mostrarán a través de componentes del GUI y el comportamiento del mismo podrá ser modificado a través de los eventos generados por el usuario.

La ventaja de las aplicaciones visuales es que permiten sacar el máximo partido a una aplicación con muy poco esfuerzo para el usuario. Es decir, podemos escribir programas que realicen tareas generales y permitir al usuario que introduzca los datos que desee y configure el funcionamiento concreto de la tarea de forma fácil usando el ratón y el teclado. Por ejemplo, un programa que calcula cuántos dólares son 100 euros es muy poco útil. Sin embargo, un conversor de monedas que dada una cantidad, una moneda origen, y otra moneda como destino es capaz de hacer la conversión es mucho más útil. Para que el usuario introduzca las monedas origen y destino y la cantidad a convertir conviene tener una aplicación visual con un GUI sencillo.

2. Diseño de aplicaciones visuales

Una aplicación visual tiene un modelo de funcionamiento ligeramente diferente a las aplicaciones no visuales. En general, el método principal de la clase principal sólo hará una cosa, crear el objeto que represente el GUI. Durante su creación, el GUI creará las ventanas que sean necesarias (normalmente una), colocará dentro los distintos componentes y los hará visibles a través de la pantalla. A partir de ese momento, quedará a la espera de que el usuario realice alguna acción con el teclado o el ratón sobre los distintos componentes del GUI.

Es importante plantear primero las distintas clases que necesitaremos para resolver el problema y que formarán el núcleo del programa. Después diseñaremos la parte visual decidiendo qué componentes y en qué parte de la ventana deberán colocarse; y, por último, escribiremos tanto las clases del núcleo de la aplicación como las que representen a nuestro GUI. Para hacer esto último podemos utilizar la herramienta de diseño automático que incluye NetBeans o podemos hacerlo a mano. Más adelante veremos cómo hacer ambas cosas.

En lo que sigue vamos a ver cómo escribir la parte del programa que se encarga del GUI pero asumiendo que ya hemos escrito previamente las clases que resuelven el problema. No en vano, es así como está planteada esta asignatura de Programación Orientada a Objetos.

3. Ventanas

Como hemos dicho, hay distintos componentes que podemos utilizar para construir un GUI. El principal es la ventana. Una ventana sirve como contenedor de otros elementos. En Java, la clase cuyos objetos representan ventanas se llama `JFrame` y pertenece al paquete `javax.swing`. Una ventana se crea como cualquier otro objeto, se rellena de otros componentes usando el `método add(Component c)`, y se hace visible usando el método `setVisible(boolean b)` pasándole el valor `true`. A partir de ese momento, el usuario puede usar el ratón para mover la ventana, cambiarla de tamaño o hacer click sobre los componentes de la misma, si los tiene.

En el listado 1 vemos un ejemplo de programa en el que se crea una ventana conteniendo un botón con el texto Ok en su interior. Si se ejecuta es posible mover la ventana, cambiar su tamaño y pulsar el botón aunque no se produce ninguna reacción. La línea 6 se encarga de hacer que al cerrarse la ventana se termine la aplicación. Si no se hace esto, al cerrar la ventana la aplicación seguirá funcionando.

En una ventana podemos añadir componentes usando el método `add`, como se puede ver en la línea 7, pero de este modo no tendremos control sobre la posición donde aparecerán visualmente dichos componentes; de hecho, sólo podremos añadir un componente y si intentamos añadir varios sólo será visible el último.

```
1  import javax.swing.*;
2
3  class Main {
4     public static void main(String args[]) {
5        JFrame ventana = new JFrame("Titulo de la ventana");
6        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
7        ventana.add(new JButton("Ok"));
8        ventana.pack();
9        ventana.setVisible(true);
10    }
11 }
```

Listado 1: Código para crear una ventana con un botón en su interior

Para poder añadir varios componentes, y determinar con precisión la posición y tamaño de los mismos, debemos utilizar un tipo especial de componente, denominado `JPanel`, que actúa como contenedor de otros componentes incluidos otros objetos de tipo `JPanel`. Los componentes añadidos a un `JPanel` se distribuyen siguiendo las reglas marcadas por su **layout**. Un layout es una descripción de cómo se deben distribuir los componentes.

Los layout se representan mediante objetos de clases derivadas de la clase `Layout` (del paquete `java.awt`) y, una vez creados, se pasa un `JPanel` en su constructor antes de añadir componentes al mismo. Hay varios tipos de `Layout` diferentes siendo los siguientes los más comúnmente utilizados:

- `BorderLayout`. Contiene hasta 5 componentes repartidos en las posiciones central, este, oeste, norte y sur. Si el tamaño del contenedor cambia, los tamaños de sus cinco componentes cambian intentando que el central sea el mayor de todos pero tratando de respetar los tamaños mínimos de los que lo rodean.

- `FlowLayout`. Distribuye un número variable e ilimitado de componentes de forma lineal, uno detrás de otro en horizontal o en vertical, según se decida, mientras que haya sitio disponible.

- `GridLayout`. Distribuye los objetos en forma de cuadrícula de tantas filas y columnas como se haya especificado en el constructor de esta clase. Sin embargo, si se indican 0 columnas y un número de filas mayor que cero, el número de columnas dependerá de la cantidad de componentes añadidos. Si sólo se especifica el número de columnas, dejando a cero el de filas, quedará fijado el número de columnas creciendo el de filas tanto como sea necesario para acomodar a todos los componentes añadidos.

Para añadir componentes a un objeto de tipo `JPanel` se puede hacer usando el método `add` que recibe como único parámetro el componente. El método está sobrecargado admitiendo también indicar en qué parte del layout queremos que se añada el componente con un segundo parámetro. Estas indicaciones se hacen utilizando constantes definidas en las clases que representan los layouts.

En el listado 2, al igual que en el 1, justo antes de hacer visible la ventana se invoca al método `pack()`. Este método es el encargado de colocar los componentes en su sitio aplicando las reglas de los layout por lo que es necesario ejecutarlo siempre antes de hacer visible las ventanas. Por otro lado, las líneas 8 a 10 muestran cómo añadir tres botones distintos en tres zonas del layout de tipo `BorderLayout` que se ha utilizado para crear el panel que, finalmente, es añadido a la ventana. La figura 1 muestra el aspecto que tendría el GUI del programa del listado 2

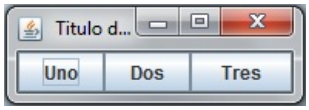

Figura 1: GUI del programa del listado 2

En ocasiones nos interesa que una ventana no pueda ser cambiada de tamaño por el usuario. Para eso podemos utilizar el método `void setResizable(boolean b)` pasándole como parámetro el valor `false`.

```
1  import javax.swing.*;
2  import java.awt.BorderLayout;

4  class Main {
5     public static void main(String args[]) {
6        JFrame ventana = new JFrame("Titulo de la ventana");
7        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
8        JPanel panel = new JPanel(new BorderLayout());
9        panel.add(new JButton("Uno"), BorderLayout.WEST);
10       panel.add(new JButton("Dos"), BorderLayout.CENTER);
11       panel.add(new JButton("Tres"), BorderLayout.EAST);
12       ventana.add(panel);
13       ventana.pack();
14       ventana.setVisible(true);
15    }
16 }
```

Listado 2: Código para crear una ventana con varios botones agrupados en un panel

Del mismo modo, aunque el título de la ventana no suela cambiar, es posible hacerlo usando el método `void setTitle(String newTitle)`.

4. Componentes de una ventana

Hay muchos componentes que podemos insertar en una ventana para hacer nuestro GUI siendo todos ellos descendientes de la clase `JComponent`; por lo tanto, todos tienen en común los métodos heredados de la misma, de los cuales, destacamos los siguientes:

- `void setEnabled(boolean b)`. Permite activar o desactivar los componentes. Desactivados se verán en un color gris claro y no reaccionarán ante las acciones del ratón o teclado.
- `void setVisible(boolean b)`. Permite mostrar u ocultar un componente.
- `void setPreferredSize(Dimension d)`. Permite especificar el tamaño deseado para el componente. El contenedor al que se añada un componente no siempre será capaz de cumplir el deseo, esto dependerá del layout utilizado, del tamaño del contenedor y del resto de componentes.
- `int getWidth()`. Devuelve el ancho real del componente.
- `int getHeight()`. Devuelve el alto real del componente.

Se indican a continuación algunos de los componentes de los que se suele hacer mayor uso:

- `JLabel`. Este componente sirve para mostrar texto, normalmente el nombre de algún campo de entrada cercano. Al crear objetos de este tipo podemos pasar un String como argumento con el texto que queremos que se muestre. También podemos usar el método `void setText(String texto)` para modificar el texto que se mostrará, y el método `String getText()` para obtener dicho valor.
- `JButton`. Este componente sirve para mostrar un botón que, al ser pulsado, generará un evento al que podremos asociar la ejecución de código como veremos en la siguiente sección. Se pueden utilizar los métodos `void setText(String texto)` y `String getText()` para modificar y obtener el texto que aparece en el botón, respectivamente. Otro método muy útil es el denominado `void setActionCommand(String command)` que permite asignar una cadena de texto (la indicada en el parámetro `command`) al botón de forma que, como veremos más adelante, podamos identificar un botón por su comando.
- `JComboBox`. Este componente sirve para mostrar listas de elementos. En principio sólo se muestra el elemento seleccionado pero permite, pulsando su botón asociado, mostrar la lista en forma de desplegable. Tenemos disponible el método `void addItem(Object o)` para añadir elementos que

serán mostrados en la lista utilizando como cadena el resultado de aplicarles el método `String toString()` de la clase `Object`. Esto significa que en la clase de cualquier objeto que vayamos a usar en un `JComboBox` habrá que sobreescribir dicho método. El método `int getSelectedIndex()` devuelve el índice del elemento seleccionado (empezando por 0) o un -1 si no hay ninguno seleccionado. El método `Object getSelectedItem()` lo que devuelve es el objeto seleccionado o `null` si no hay ninguno. Y el método `void setSelectedIndex(int index)` nos permite seleccionar uno de los elementos.

- `JTextField` y `JTextArea`. Son componentes que permiten que el usuario introduzca una cadena (o varias cadenas) de texto, respectivamente. Los métodos más usados son `String getText()` y `void setText(String text)` que se comportan de modo similar a los métodos del mismo nombre de los componentes anteriores. A los objetos de tipo `JTextField` también es posible asignarles un comando con `void setActionCommand(String command)`. Sin embargo, dado que `JTextArea` permite introducir y mostrar varias cadenas de texto en distintas líneas, tiene disponibles algunos métodos particulares. Por ejemplo, los métodos `void setRows(int rows)` y `void setColumns(int columns)` sirven para definir el tamaño del componente en función del tamaño de letra que se utilice; y el método `void append(String str)` permite añadir una cadena de texto al final.

- `JPanel`. Como ya hemos comentado, `JPanel` es el componente que sirve de contenedor de otros componentes. Podemos usar el método `void setBorder(Border b)` que, sin ser propio de esta clase, si no heredado de `JComponent`, es aquí donde más se suele usar para dar a los paneles un aspecto más elegante. Hay varios tipos de bordes pero el más común es el que lleva un título incrustado y este se puede crear con el método estático `void createTitledBorder(String título)` de la clase `BorderFactory` del paquete `javax.swing`.

5. Eventos

Hasta el momento nos hemos dedicado a diseñar el aspecto del GUI rellenando la ventana con componentes. Hemos visto como utilizar distintos layouts y qué métodos podemos aplicar a los componentes más comunes. Sin embargo, si tras hacer todo esto ejecutamos nuestra aplicación ni al pulsar los botones ni al escribir texto en los campos de texto pasará nada. Esto es así porque, de momento, no hemos escrito el código que debe ejecutarse cuando se producen estas acciones que llamamos **eventos**.

Cada componente genera un tipo de evento y cuando éstos se producen se invoca a un método concreto sobre el objeto que se encargue de manejar los eventos. Estos objetos se denominan `EventListener` o manejadores de eventos y se caracterizan por implementar ciertos interfaces predefinidos que especifican los métodos que el componente en cuestión invocará cuando el usuario interaccione con él.

En esta sección sólo vamos a estudiar los eventos que producen los botones al ser pulsados, los campos de texto al pulsar la tecla de retorno de línea, y los desplegables al seleccionar un elemento de la lista. Para procesar el evento generado por cualquiera de estos componentes es necesario tener un objeto cuya clase implemente el interface `ActionListener`. La forma más sencilla para conseguir esto es que la misma clase que se encarga de crear el GUI sea la que implemente dicho interface y, por lo tanto, un objeto de la misma se ocupe de gestionar todos los eventos de los componentes que incluya el GUI.

En el listado 3 tenemos el primer ejemplo de programa que crea un GUI que responde a las acciones del usuario. En concreto, hay dos botones que al pulsarlos muestran un mensaje que dice "Se ha pulsado un botón". Para conseguirlo hemos escrito una clase auxiliar llamada `Controlador` que será la encargada de crear el GUI y de responder a los eventos que se produzcan.

Como se puede ver, el GUI se crea a través de la invocación al método `void setup()` desde el constructor de la clase. El método `void setup()` es privado a la clase `Controlador`. Además, esta clase implementa el interface `ActionListener` (del paquete `java.awt.event`) por lo que incluye el método `void actionPerformed(ActionEvent ae)`. De este modo, los objetos de tipo `Controlador` servirán como manejadores de eventos. De hecho, el método `actionPerformed` es el que se ejecutará cuando se pulse cualquiera de los botones por lo que es en él donde está el código necesario para mostrar el mensaje que hemos comentado anteriormente.

Una vez que tenemos una clase cuyos objetos sirven como manejadores de eventos lo único que falta es indicar a los componentes que queramos (en este caso los dos botones) cuál será el objeto sobre el que deben invocar el método `actionPerformed` cuando el usuario haga click con el ratón sobre ellos.

```
1  import java.awt.BorderLayout;
2  import java.awt.event.ActionEvent;
3  import java.awt.event.ActionListener;
4  import javax.swing.*;

6  public class Controlador implements ActionListener {
7    public Controlador() {
8      setup();
9    }

11   private void setup() {
12     JFrame ventana = new JFrame("Titulo de la ventana");
13     ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14     JPanel panel = new JPanel(new BorderLayout());
15     JButton jButtonUno = new JButton("Uno");
16     jButtonUno.addActionListener(this);
17     JButton jButtonDos = new JButton("Dos");
18     jButtonDos.addActionListener(this);
19     panel.add(jButtonUno, BorderLayout.WEST);
20     panel.add(jButtonDos, BorderLayout.EAST);
21     ventana.add(panel);
22     ventana.pack();
23     ventana.setVisible(true);
24   }

26   public void actionPerformed(ActionEvent ae) {
27     System.out.println("Se ha pulsado un boton\n");
28   }
29 }
```

Listado 3: Ejemplo simple de gestión de eventos producidos por botones

Para hacer esto, en las líneas 16 y 18 se invoca el método `void addActionListener(ActionListener actionListener)` sobre las variables que se refieren a los objetos de tipo `JButton` del GUI pasándoles como parámetro `this`. Dado que `this` es un objeto de tipo `Controlador`, y esta clase implementa el interface `ActionListener`, la unión entre componente y controlador queda hecha.

Como hemos podido comprobar, el texto mostrado al pulsar un botón aparece dentro de NetBeans en la ventana de salida en lugar de aparecer directamente en alguna parte visual de nuestra propia aplicación. Además, al tener un único objeto controlador todos los eventos se tratan igual. Sin embargo, éste no suele ser el comportamiento deseado. Normalmente, necesitaremos que cada botón tenga un efecto distinto y que el resultado aparezca en la propia ventana gráfica no en NetBeans.

Para solucionar el primer problema vamos a añadir un nuevo componente que permita mostrar texto, por ejemplo un `JTextArea` que usaremos invocando sobre él el método `void append(String texto)` con el mensaje deseado. Para que desde el método `void actionPerformed(ActionEvent actionEvent)` de la clase `Controlador` se pueda acceder al componente de tipo `JTextArea` de nuestro GUI debemos tener acceso a la variable de dicho tipo que se refiera al mismo. Para conseguir esto, y como regla general, todos los componentes que vayamos a usar desde nuestra aplicación los declararemos como miembros privados de la clase controlador como muestra el listado 4.

Por otro lado, necesitamos que cada botón tenga un efecto distinto incluso cuando estemos utilizando un único controlador. Para conseguirlo, podemos utilizar el parámetro del método `void actionPerformed (ActionEvent actionEvent)` ejecutado por el componente que genera el evento. A través de ese parámetro de tipo `ActionEvent` podemos pasar información. Uno de los valores que incluye dicho objeto es el comando que tuviera asignado el componente. Dicho comando, que consiste en una cadena de texto, podemos asignárselo a cada componente con el método `void setActionCommand(String command)`. De este modo, como muestra el listado 5 en las líneas 19 y 22, podemos etiquetar cada componente (en nuestro caso cada botón) con un "comando" distinto que usaremos para distinguir la procedencia del evento. Concretamente, dentro del método `void actionPerformed(ActionEvent actionEvent)` de la clase `Controlador`, podremos consultar el "comando" asociado al componente que ha generado el evento usando el método `String getActionCommand()` de la clase `ActionEvent` sobre el parámetro `actionEvent`.

```
1  import java.awt.BorderLayout;
2  import java.awt.event.*;
3  import javax.swing.*;

5  public class Controlador implements ActionListener {
6      private JFrame ventana;
7      private JButton jButtonUno, jButtonDos;
8      private JTextArea jTextAreaSalida;

10     public Controlador() {
11         setup();
12     }

14     private void setup() {
15         ventana = new JFrame("Titulo de la ventana");
16         ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17         JPanel panel = new JPanel(new BorderLayout());
18         jButtonUno = new JButton("Uno");
19         jButtonUno.addActionListener(this);
20         jButtonDos = new JButton("Dos");
21         jButtonDos.addActionListener(this);
22         jTextAreaSalida = new JTextArea();
23         panel.add(jButtonUno, BorderLayout.WEST);
24         panel.add(jTextAreaSalida, BorderLayout.CENTER);
25         panel.add(jButtonDos, BorderLayout.EAST);
26         ventana.add(panel);
27         ventana.pack();
28         ventana.setVisible(true);
29     }

31     public void actionPerformed(ActionEvent actionEvent) {
32         jTextAreaSalida.append("Se ha pulsado un boton\n");
33     }
34 }
```

Listado 4: Declaración de los componentes principales como miembros privados del controlador para tener acceso a ellos desde los métodos de manejo de eventos

Además del comando, en ocasiones también nos puede interesar obtener el objeto que generó el evento para lo que la clase `ActionEvent` también incluye el método `Object getSource()`. En cualquier caso, aunque no contáramos con este método, como nuestros componentes son miembros privados del objeto controlador, sabiendo el comando podríamos determinar el miembro que lo ha generado y actuar en consecuencia según lo que nos pueda interesar.

6. Cuadros de diálogo diseñados por el programador

En ocasiones necesitamos hacerle al usuario una pregunta concreta, por ejemplo, cuando queremos que confirme que realmente quiere hacer una operación. A estas ventanas se les llama **cuadros de diálogo** y para generarlas tenemos dos opciones. La más fácil consiste en recurrir a la clase `JOptionPane` que incluye varios métodos estáticos útiles para este fin tal y como se vio en la sesión anterior de entrada/salida.

Sin embargo, si deseamos tener nuestro propio cuadro de diálogo tenemos que construirlo nosotros mismos. Para eso, en lugar de utilizar un objeto de tipo `JFrame` usaremos uno de tipo `JDialog`. Tras crearlo lo rellenaremos de componentes del mismo modo que hemos hecho con las ventanas normales, y también asignaremos los manejadores de eventos del mismo modo.

Con todo, existen algunas diferencias con una ventana normal, en particular, no se puede indicar el título; tampoco es necesario usar un `JPanel` para añadir más de un componente porque ya contienen uno al que, además, se le puede asignar un layout a través del método `void setLayout(Layout l)`; y por último, no conviene hacerlos visibles hasta el momento en que sean necesarios con lo que dejaremos para el último momento la invocación a los métodos `pack()` y `setVisible(true)`.

Además, el comportamiento normal de un cuadro de diálogo consiste en bloquear el GUI de la ventana principal hasta que el del cuadro de diálogo haya terminado. Para conseguir este comportamiento

```
1  import java.awt.BorderLayout;
2  import java.awt.event.*;
3  import javax.swing.*;

5  public class Controlador3 implements ActionListener {
6    private JFrame ventana;
7    private JButton jButtonUno, jButtonDos;
8    private JTextArea jTextAreaSalida;

10   public Controlador3() {
11     setup();
12   }

14   private void setup() {
15     ventana = new JFrame("Titulo de la ventana");
16     ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17     JPanel panel = new JPanel(new BorderLayout());
18     jButtonUno = new JButton("Uno");
19     jButtonUno.setActionCommand("ComandoUno");
20     jButtonUno.addActionListener(this);
21     jButtonDos = new JButton("Dos");
22     jButtonDos.setActionCommand("ComandoDos");
23     jButtonDos.addActionListener(this);
24     jTextAreaSalida = new JTextArea();
25     panel.add(jButtonUno, BorderLayout.WEST);
26     panel.add(jTextAreaSalida, BorderLayout.CENTER);
27     panel.add(jButtonDos, BorderLayout.EAST);
28     ventana.add(panel);
29     ventana.pack();
30     ventana.setVisible(true);
31   }

33   public void actionPerformed(ActionEvent actionEvent) {
34     if (actionEvent.getActionCommand().equals("ComandoUno")) {
35       jTextAreaSalida.append("Se ha pulsado un boton uno\n");
36     }
37     if (actionEvent.getActionCommand().equals("ComandoDos")) {
38       jTextAreaSalida.append("Se ha pulsado un boton dos\n");
39     }
40   }
41 }
```

Listado 5: Uso de los "comandos" para distinguir la procedencia de los eventos

hay que configurar el objeto de tipo `JDialog` para que sea **modal**. Para ello, invocaremos el método `void setModal(true)`. Por otro lado, es necesario cerrar el cuadro de diálogo usando el método `void dispose()` por lo que debe ser la última sentencia del código del manejador de los botones que cierran el diálogo.

En el código mostrado en el listado 6 podemos destacar el hecho de que el mismo manejador de eventos, es decir, el código del método `void actionPerformed(ActionEvent actionEvent)`, controla la pulsación del botón con texto "Uno" situado en la ventana principal y la del botón `Ok` del diálogo. Cuando el botón pulsado es el que tiene asignado el comando `ComandoUno` se crea el diálogo y se hace visible y modal. En el segundo caso lo único que hace es cerrar el mismo usando el método `dispose()`.

Por último, es importante observar como sólo la variable de tipo `JDialog` es miembro de la clase `Controlador` ya que es la única que se necesita utilizar desde el manejador.

7. Usando NetBeans para hacer Interfaces Gráficos de Usuario

También es posible utilizar NetBeans para hacer los GUI. Para ello, y teniendo en cuenta todo lo visto anteriormente, seguiremos los siguientes pasos generales:

1. Crear un `JFrame` pulsando con el botón derecho del ratón sobre el paquete y eligiendo nuevo `JFrame Form...` Esto crea un fichero con una clase que heredará de `JFrame` y en la que NetBeans pondrá

el código necesario para crear el GUI. Cada vez que pinchemos dos veces en este fichero, en lugar de código, aparecerá un editor gráfico en el que, de forma visual, podremos añadir componentes arrastrándolos desde la barra de herramientas de la parte derecha. También podremos ver el código generado cambiando entre la vista de diseño (`Design`) y la de código fuente (`Source`) con los botones de la parte superior de la ventana.

2. Para cambiar el título de la ventana usaremos la paleta de propiedades situada en la parte derecha inferior. Esta paleta refleja las propiedades del componente seleccionado por lo que, en primer lugar, habrá que seleccionar el `JFrame`. Podemos hacerlo haciendo click directamente en la ventana que estamos diseñando o en el componente que aparece en la ventana del `Navegador` situada en la parte inferior izquierda. En ella aparecen, de forma jerárquica, todos los componentes que contiene el `JFrame` que estemos diseñando.

3. Precisamente, desde la ventana del `Navegador` es fácil cambiar el layout de los distintos paneles o del propio JFrame. Para ello pincharemos sobre el componente con el botón derecho y seleccionaremos el layout que queramos a través del menú `Set Layout`. También podremos configurarlo en el caso de que se pueda. Por ejemplo, con uno de tipo `GridLayout` podremos indicar las columnas y filas que queramos que tenga.

4. Para añadir componentes a la ventana principal basta con arrastrarlos desde la barra de herramientas. Una vez añadidos es conveniente cambiar el nombre de la variable a la que estarán ligados y el texto que muestran por defecto. Ambas cosas podremos hacerlas con el botón derecho del ratón seleccionando respectivamente `Edit Text` y `Change Variable Name`.

5. Para introducir el código asociado a los eventos que producen botones, campos de texto o desplegables podremos hacerlo seleccionando la opción `Events` y luego un evento de todos los que salen en la lista desplegable que se nos muestra. En ese momento aparecerá el editor de código con el método que se ejecutará al producirse el evento. Ahí podremos escribir lo que deseemos. Todos los eventos que tengan código ya asociado aparecerán en negrita.

6. Finalmente, podemos añadir diálogos sin más que arrastrar un componente de tipo `JDialog` a la parte externa de la ventana que estemos diseñando. A partir de ese momento, en la ventana del inspector aparecerán las dos entidades separadas y si pinchamos en la de tipo `JDialog` podremos diseñar su contenido añadiendo los componentes necesarios. Es importante también ponerle un nombre significativo a la variable que represente el cuadro de diálogo y recordar que para mostrarla será necesario que se comporte de forma modal. También habrá que invocar a `dispose()` al cerrarla.

Cuando usemos NetBeans para crear un GUI tenemos que tener en cuenta que el GUI se creará en una clase independiente. Por lo tanto, desde nuestro programa principal deberemos crear un objeto de dicha clase si queremos que el GUI comience a funcionar. Además, los objetos que tengamos que usar desde los métodos manejadores de eventos deberán estar declarados dentro de la clase que hace NetBeans. Para conseguir esto abriremos el `JFrame` diseñado por NetBeans en modo código fuente y añadiremos los miembros necesarios a la clase. Si necesitamos inicializarlos lo haremos en el constructor del `JFrame` justo después de la llamada que, automáticamente, aparece al método `initComponents()`. Finalmente, NetBeans añade a la propia clase de cada `JFrame` que crea un método estático principal que sirve para probar el funcionamiento del GUI. Conviene igualmente aclarar que en el código que genera NetBeans aparecen sombreadas en gris las partes que no se deben modificar ya que, en sucesivos usos de la herramienta gráfica, se volverán a generar.

Parte II
Prácticas

8. Ejercicios sobre aplicaciones visuales mediante interfaces gráficos de usuario en Java

En esta sesión de prácticas, para la que debes haber leído y entendido en detalle la parte teórica de esta sesión, vamos a trabajar sobre los conceptos relativos a las aplicaciones visuales que contienen o hacen uso de interfaces gráficos de usuario en el lenguaje de programación Java.

EJERCICIO 1. *Crea un proyecto y copia el código del listado 1. Añade dos botones más del mismo modo que se hace con el botón Ok. Ejecuta el programa y comprueba que sólo se ve el último botón añadido.*

EJERCICIO 2. *Crea un proyecto y añade la clases* `Controlador` *como se indica en el listado 3. Crea también una clase principal en cuyo método* `main()` *simplemente se cree un objeto de tipo Controlador. Prueba el funcionamiento del programa y comprueba que no importa qué botón se pulse siempre sale el mismo mensaje.*

EJERCICIO 3. *Tanto* `JComboBox` *como* `JTextField` *generan el mismo evento que los botones. Añade al programa del listado 5 un componente de cada tipo y amplía el método* `void actionPerformed(ActionEvent actionEvent)` *para procesar los eventos provenientes de dichos componentes de forma adecuada.*

EJERCICIO 4. *Crea un proyecto y añade el código del listado 6 modificando el diseño de la ventana para que incluya una objeto de tipo* `JLabel` *con un valor inicial de "" (texto vacío). Modifica también el diálogo para que incluya un campo de texto cuyo contenido sea mostrado en la etiqueta añadida a la ventana principal.*

```
1  import java.awt.BorderLayout;
2  import java.awt.FlowLayout;
3  import java.awt.event.*;
4  import javax.swing.*;

6  public class Controlador4 implements ActionListener {
7    private JDialog dialogo;
8    private JFrame ventana;
9    private JButton jButtonUno, jButtonOk;


12   public Controlador4() {
13     setup();
14   }

16   private void setup() {
17       ventana = new JFrame("Titulo de la ventana");
18       ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19       JPanel panel = new JPanel(new BorderLayout());
20       jButtonUno = new JButton("Uno");
21       jButtonUno.setActionCommand("ComandoUno");
22       jButtonUno.addActionListener(this);
23       panel.add(jButtonUno, BorderLayout.CENTER);
24       ventana.add(panel);
25       ventana.pack();
26       ventana.setVisible(true);
27   }

29   public void actionPerformed(ActionEvent actionEvent) {
30     if (actionEvent.getActionCommand().equals("ComandoUno")) {
31       dialogo = new JDialog();
32       dialogo.setLayout(new FlowLayout());
33       JLabel texto = new JLabel("Pulsa Ok para continuar...");
34       jButtonOk = new JButton("Ok");
35       jButtonOk.setActionCommand("ok");
36       jButtonOk.addActionListener(this);
37       dialogo.add(texto);
38       dialogo.add(jButtonOk);
39       dialogo.setModal(true);
40       dialogo.pack();
41       dialogo.setVisible(true);
42     }
43     if (actionEvent.getActionCommand().equals("ok")) {
44       dialogo.dispose();
45     }
46   }
47 }
```

Listado 6: Creación de un diálogo personalizado