Capítulo 4
Aplicaciones gráficas

1. Introducción

Hasta ahora, todos los ejemplos de código que hemos realizado funcionan exclusivamente en modo texto. La información se visualiza en una consola y también se informa desde dicha consola. La sencillez de este modo de funcionamiento supone una ventaja innegable para el aprendizaje de un lenguaje. Sin embargo, la mayoría de los usuarios de las futuras aplicaciones seguramente esperan disponer de una interfaz un poco menos austera que una pantalla en modo texto. En este capítulo vamos a estudiar cómo funcionan las interfaces gráficas con Java. Se dará cuenta enseguida de que el diseño de interfaces gráficas en Java no es tan sencillo y requiere escribir muchas líneas de código. En la práctica, contará con varias herramientas de desarrollo, capaces de encargarse de la generación de una gran parte de este código según el diseño gráfico de la aplicación que esté dibujando. Sin embargo, es importante entender correctamente los principios de funcionamiento de este código para intervenir en él y eventualmente optimizarlo. En este capítulo, no emplearemos ninguna herramienta específica, sólo conservaremos nuestro propio editor de texto, un compilador y la máquina virtual.

1.1 Las bibliotecas gráficas

El lenguaje Java propone dos bibliotecas dedicadas al diseño de interfaces gráficas: la biblioteca AWT y la biblioteca SWING. Los fundamentos de uso son casi idénticos en ambas bibliotecas. El uso simultáneo de las dos bibliotecas en una misma aplicación puede provocar problemas de funcionamiento y por ello debería evitarse.

1.1.1 La biblioteca AWT

Esta biblioteca es la primera disponible para el desarrollo de interfaces gráficas. Contiene una multitud de clases e interfaces que permiten la definición y la gestión de interfaces gráficas. En realidad, esta biblioteca utiliza las funcionalidades gráficas del sistema operativo. Por lo tanto, no es el código presente en esta biblioteca el que asegura el resultado gráfico de los diferentes componentes. Este código hace de intermediario con el sistema operativo. Su uso ahorra bastante recursos, pero presenta varios inconvenientes.

– Al estar relacionado el aspecto visual de cada componente con la representación que el sistema operativo hace de él, puede resultar delicado desarrollar una aplicación que tenga una apariencia coherente en todos los sistemas. El tamaño y la posición de los diferentes componentes son los dos elementos que se ven principalmente afectados por este problema.
– Para que esta biblioteca sea compatible con todos los sistemas operativos, los componentes que contiene están limitados a los más corrientes (botones, zonas de texto, listas…).

1.1.2 La biblioteca Swing

Esta biblioteca se diseñó para resolver las principales carencias de la biblioteca AWT. Se obtuvo esta mejora al escribir completamente la biblioteca en Java sin apenas recurrir a los servicios del sistema operativo. Únicamente algunos elementos gráficos (ventanas y cuadros de diálogo) siguen relacionados con el sistema operativo. Para los demás componentes, es el código de la biblioteca Swing el encargado de determinar completamente su aspecto y su comportamiento.

La biblioteca Swing contiene por lo tanto una cantidad impresionante de clases que sirven para redefinir los componentes gráficos. Sin embargo, no debemos pensar que la biblioteca Swing convierte la biblioteca AWT en algo completamente obsoleto. De hecho, Swing recupera muchos de los elementos de la biblioteca AWT. En el resto del capítulo, emplearemos esencialmente esta biblioteca.

1.2 Construcción de la interfaz gráfica de una aplicación

El diseño de la interfaz gráfica de una aplicación se fundamenta ante todo en crear instancias de las clases que representan los diferentes elementos necesarios, modificar las características de estas instancias de clase, agruparlas y prever el código de gestión de los diferentes eventos que pueden intervenir durante el funcionamiento de la aplicación. Así, una aplicación gráfica está compuesta por una multitud de elementos superpuestos o anidados. Entre estos elementos, uno toma un papel preponderante en la aplicación. Se le suele llamar contenedor de primer nivel. Es el encargado de interactuar con el sistema operativo y agrupar a los demás elementos. Este contenedor de primer nivel no suele contener directamente los componentes gráficos sino otros contenedores en los cuales están ubicados los componentes gráficos. Para facilitar la disposición de estos elementos entre sí, vamos a utilizar la ayuda de un renderizador. Esta superposición de elementos podría asimilarse a un árbol, en su raíz principal tenemos el contenedor de primer nivel y cuyas diferentes ramas están constituídas por los demás contenedores. Las hojas del árbol corresponden a los componentes gráficos.

Dado que el contenedor de primer nivel es el elemento indispensable de cualquier aplicación gráfica, empezaremos estudiando con detalle sus características y utilización, para luego pasar a comprender los principales componentes gráficos.

2. Diseño de una interfaz gráfica

Hemos visto un poco más arriba que cualquier aplicación gráfica se compone de, al menos, un contenedor de primer nivel. La biblioteca Swing dispone de tres clases que permiten llevar a cabo este papel:

`JApplet`: representa una ventana gráfica incluida en una página html para que un navegador se haga cargo de ella. Se estudia este elemento en detalle en el capítulo correspondiente.

`JWindow`: representa la ventana gráfica más rudimentaria que pueda existir. No dispone de ninguna barra de título, ningún menú de sistema, ningún borde: en realidad es un mero rectángulo. Esta clase se utiliza rara vez excepto para la visualización de una pantalla de inicio en el momento del arranque de una aplicación (*splash screen*).

`JFrame`: representa una ventana gráfica completa y plenamente funcional. Dispone de una barra de título, de un menú de sistema y de un borde. Puede fácilmente contener un menú y, por supuesto, es el elemento que vamos a emplear en la mayoría de los casos.

2.1 Las ventanas

La clase `JFrame` es un elemento indispensable en cualquier aplicación gráfica. Como en el caso de una clase normal, debemos crear una instancia, modificar eventualmente las propiedades y utilizar los métodos. A continuación se muestra el código de la primera aplicación gráfica.

```
package es.eni;
import javax.swing.JFrame;
public class Main {
     public static void main(String[] args)
     {
          JFrame ventana;
          // creación de la instancia de la clase JFrame
          ventana=new JFrame();
          // modificación de la posición y del
          // tamaño de la ventana
          ventana.setBounds(0,0,300,400);
          // modificación del título de la ventana
```

```
        ventana.setTitle("primera ventana en JAVA");
        // visualización de la ventana
        ventana.setVisible(true);
    }
```

y el resultado de su ejecución:

Es fácil de usar y muy eficaz. De hecho, es tan eficaz que no se puede parar la aplicación. En efecto, incluso si el usuario cierra la ventana, este cierre no provoca la supresión de la instancia de `JFrame` de la memoria. La única solución para detener la aplicación es apagar la máquina virtual Java con la combinación de teclas [Ctrl] **C**. Ante esto, se recomienda proporcionar otra solución para detener más fácilmente la ejecución de la aplicación, es decir, junto con el cierre de la ventana.

Una primera solución consiste en gestionar los eventos que se producen en el momento del cierre de la ventana y, en uno de ellos, provocar la detención de la aplicación. Se estudiará esta solución en el párrafo dedicado a la gestión de los eventos.

La segunda solución utiliza comportamientos predefinidos para el cierre de la ventana. Estos comportamientos están determinados por el método `setDefaultCloseOperation`. Se definen varias constantes para determinar la acción emprendida al cierre de la ventana.

`DISPOSE_ON_CLOSE`: esta opción provoca la detención de la aplicación en el momento del cierre de la última ventana controlada por la máquina virtual.

`DO_NOTHING_ON_CLOSE`: con esta opción, no ocurre nada cuando el usuario pide el cierre de la ventana. En este caso, es obligatorio gestionar los eventos para que la acción del usuario tenga algún efecto sobre la ventana o la aplicación.

`EXIT_ON_CLOSE`: esta opción provoca la detención de la aplicación incluso si otras ventanas siguen visibles.

`HIDE_ON_CLOSE`: con esta opción la ventana simplemente queda oculta como consecuencia de una llamada a su método `setVisible(false)`.

La clase `JFrame` se encuentra al final de una jerarquía de clases bastante importante e implementa numerosas interfaces. Por este motivo, dispone de varios métodos y atributos.

Class JFrame

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Frame
                  └javax.swing.JFrame
```

All Implemented Interfaces:
ImageObserver, MenuContainer, Serializable, Accessible, RootPaneContainer, WindowConstants

La meta de este libro no es retomar toda la documentación del JDK, de modo que no recorreremos todos los métodos disponibles sino sencillamente los más utilizados según las necesidades. Sin embargo puede resultar interesante revisar la documentación antes de realizar el diseño de un método para determinar si lo que queremos diseñar no ha sido ya previsto por los diseñadores de Java.

Ahora que somos capaces de visualizar una ventana, el grueso del trabajo va a consistir en añadir un contenido a la ventana. Antes de poder añadir algo a una ventana, conviene entender bien su estructura, que resulta relativamente compleja. Un objeto `JFrame` se compone de varios elementos superpuestos, cada uno con un papel muy específico en la gestión de la ventana.