Un ejemplo práctico de GUI

3.6 Ejemplo práctico de GUI y gráficos: uso de cuadros de diálogo

Cómo mostrar texto en un cuadro de diálogo

Muchas aplicaciones utilizan ventanas, o cuadros de diálogo para mostrar la salida. Por ejemplo, los navegadores Web como Chrome, Firefox, Internet Explorer, Safari y Opera muestran las páginas Web en sus propias ventanas. Los programas de correo electrónico le permiten escribir y leer mensajes en una ventana. Por lo general, los cuadros de diálogo son ventanas en las que los programas muestran mensajes importantes a los usuarios. La clase JOptionPane cuenta con cuadros de diálogo prefabricados, los cuales permiten a los programas mostrar ventanas que contengan mensajes; a dichas ventanas se les conoce como diálogos de mensaje. La figura 3.12 muestra el objeto String “Bienvenido a Java” en un diálogo de mensaje.

```
1  // Fig. 3.12: Dialogo1.java
2  // Uso de JOptionPane para mostrar varias líneas en un cuadro de diálogo.
3  import javax.swing.JOptionPane;
4
5  public class Dialogo1
6  {
7     public static void main(String[] args)
8     {
9        // muestra un diálogo con un mensaje
10       JOptionPane.showMessageDialog(null, “Bienvenido a Java”);
11    }
12 } // fin de la clase Dialogo1
```

Fig. 3.12 Uso de `JOptionPane` para mostrar varias líneas en un cuadro de diálogo

El método static `showMessageDialog` de la clase `JOptionPane`

La línea 3 indica que el programa utiliza la clase ***JOptionPane*** del paquete `javax.swing`. Este paquete contiene muchas clases que le ayudan a crear interfaces gráficas de usuario (GUI). Los componentes de la GUI facilitan la entrada de datos al usuario del programa, además de la presentación de los datos de salida. La línea 10 llama al método `showMessageDialog` de `JOptionPane` para mostrar un cuadro de diálogo que contiene un mensaje. El método requiere dos argumentos. El primero ayuda a la aplicación de Java a determinar en dónde colocar el cuadro de diálogo. Por lo general, un diálogo se muestra desde una aplicación de GUI con su propia ventana. El primer argumento hace referencia a esa ventana (conocida como ventana **padre**) y hace que el diálogo aparezca centrado sobre la ventana de la aplicación. Si el primer argumento es `null`, el cuadro de diálogo aparece en el centro de la pantalla de la computadora. El segundo argumento es el objeto `String` a mostrar en el cuadro de diálogo.

Introducción de los métodos `static`

El método `showMessageDialog` de la clase `JOptionPane` es lo que llamamos un método `static`. A menudo, dichos métodos definen las tareas que se utilizan con frecuencia. Por ejemplo, muchos programas muestran cuadros de diálogo, y el código para hacer esto es el mismo siempre. En vez de “reinventar la rueda” y crear código para realizar esta tarea, los diseñadores de la clase `JOptionPane` declararon un método `static` que realiza esta tarea por usted. La llamada a un método `static` se realiza mediante el uso del nombre de su clase, seguido de un punto (.) y del nombre del método, como en

```
NombreClase.nombreMétodo(argumentos)
```

Observe que no tiene que crear un objeto de la clase `JOptionPane` para usar su método `static` llamado `showMessageDialog`. En el capítulo 6 analizaremos los métodos `static` con más detalle.

Introducción de texto en un cuadro de diálogo

La figura 3.13 utiliza otro cuadro de diálogo `JOptionPane` predefinido, conocido como diálogo de entrada, el cual permite al usuario introducir datoss en un programa. El programa pide el nombre del usuario y responde con un diálogo de mensaje que contiene un saludo y el nombre introducido por el usuario.

```
1  // Fig. 3.13: DialogoNombre.java
2  // Entrada básica con un cuadro de diálogo.
3  import javax.swing.JOptionPane;
4
5  public class DialogoNombre
6  {
7     public static void main(String[] args)
8     {
9        // pide al usuario que escriba su nombre
10       String nombre =JOptionPane.showInputDialog("Cual es su nombre?");
11
12       // crea el mensaje
13       String mensaje =
14          String.format("Bienvenido, %s, a la programacion en Java!", nombre);
15
16       // muestra el mensaje para dar la bienvenida al usuario por su nombre
17       JOptionPane.showMessageDialog(null, mensaje);
18    } // fin de main
19 } // fin de la clase DialogoNombre
```

Fig. 3.13 Cómo obtener la entrada del usuario mediante un cuadro de diálogo

El método static `showInputDialog` de la clase `JOptionPane`

La línea 10 utiliza el método `showInputDialog` de `JOptionPane` para mostrar un diálogo de entrada que contiene un indicador (**prompt**) y un campo (conocido como campo de texto), en el cual el usuario puede escribir texto. El argumento del método `showInputDialog` es el indicador que muestra lo que el usuario debe escribir. El usuario escribe caracteres en el campo de texto, y después hace clic en el botón **Aceptar** u oprime la tecla **Intro** para devolver el objeto `String` al programa. El método `showInputDialog` devuelve un objeto `String` que contiene los caracteres escritos por el usuario. Almacenamos el objeto `String` en la variable nombre. Si oprime el botón **Cancelar** en el cuadro de diálogo u oprime **Esc**, el método devuelve `null` y el programa muestra la palabra "**null**" como el nombre.

El método `static` format de la clase `String`

Las líneas 13 y 14 utilizan el método `static` `String` llamado `format` para devolver un objeto `String` que contiene un saludo con el nombre del usuario. El método `format` es similar al método `System.out.printf`, excepto que `format` devuelve el objeto `String` con formato, en vez de mostrarlo en una ventana de comandos. La línea 17 muestra el saludo en un cuadro de diálogo de mensaje, como hicimos en la figura 3.12.

Ejercicio del ejemplo práctico de GUI y gráficos

3.1 Modifique el programa de suma que aparece en la figura 2.7 para usar la entrada y salida en un cuadro de diálogo con los métodos de la clase `JOptionPane`. Como el método `showInputDialog` devuelve un objeto `String`, debe convertir el objeto `String` que introduce el usuario a un int para usarlo en los cálculos. El método `parseInt` es un método `static` de la clase `Integer` (del paquete `java.lang`) que recibe un argumento `String` que representa a un entero y devuelve el valor como `int`. Si el objeto `String` no contiene un entero válido, el programa terminará con un error.

4.15 Ejemplo práctico de GUI y gráficos: creación de dibujos simples

Una característica atractiva de Java es su soporte para gráficos, el cual permite a los programadores mejorar sus aplicaciones en forma visual. Ahora le presentaremos una de las capacidades gráficas de Java: dibujar líneas. También cubriremos los aspectos básicos sobre cómo crear una ventana para mostrar un dibujo en la pantalla de la computadora.

El sistema de coordenadas de Java

Para dibujar en Java, primero debe comprender su sistema de coordenadas (figura 4.17), un esquema para identificar puntos en la pantalla. De manera predeterminada, la esquina superior izquierda de un componente de la GUI tiene las coordenadas (0, 0). Un par de coordenadas está compuesto por una coordenada x (la coordenada horizontal) y una coordenada y (la coordenada vertical). La coordenada x es la ubicación horizontal que se desplaza de izquierda a derecha. La coordenada y es la ubicación vertical que se desplaza de arriba hacia abajo. El eje x indica cada una de las coordenadas horizontales, y el eje y cada una de las coordenadas verticales. Las coordenadas indican en dónde deben mostrarse los gráficos en una pantalla. Las unidades de las coordenadas se miden en píxeles. El término **píxel** significa “elemento de imagen”. Un píxel es la unidad de resolución más pequeña de una pantalla.

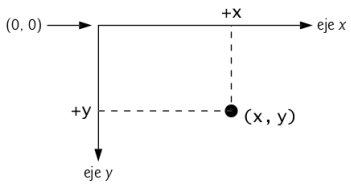

Fig. 4.17 Sistema de coordenadas de Java. Las unidades están en pixeles

Primera aplicación de dibujo

Nuestra primera aplicación de dibujo simplemente dibuja dos líneas. La clase `PanelDibujo` (figura 4.18) realiza el dibujo en sí, mientras que la clase `PruebaPanelDibujo` (figura 4.19) crea una ventana para mostrar el dibujo. En la clase `PanelDibujo`, las instrucciones import de las líneas 3 y 4 nos permiten utilizar la clase `Graphics` (del paquete `java.awt`), que proporciona varios métodos para dibujar texto y figuras en la pantalla, y la clase `JPanel` (del paquete `javax.swing`), que proporciona un área en la que podemos dibujar.

```
1  // Fig. 4.18: PanelDibujo.java
2  // Uso de drawLine para conectar las esquinas de un panel.
3  import java.awt.Graphics;
4  import javax.swing.JPanel;
5
6  public class PanelDibujo extends JPanel
7  {
8     // dibuja una x desde las esquinas del panel
9     public void paintComponent(Graphics g)
10    {
11       // llama a paintComponent para asegurar que el panel se muestre correctamente
12       super.paintComponent(g);
13
14       int anchura = getWidth(); // anchura total
15       int altura = getHeight(); // altura total
16
17       // dibuja una línea de la esquina superior izquierda a la esquina inferior
            derecha
18       g.drawLine(0, 0, anchura, altura);
19
20       // dibuja una línea de la esquina inferior izquierda a la esquina superior
            derecha
21       g.drawLine(0, altura, anchura, 0);
22    }
23 } // fin de la clase PanelDibujo
```

Fig. 4.18 Uso de `drawLine` para conectar las esquinas de un panel

```
1  // Fig. 4.19: PruebaPanelDibujo.java
2  // Crear un objeto JFrame para mostrar un objeto PanelDibujo.
3  import javax.swing.JFrame;
4
5  public class PruebaPanelDibujo
6  {
7     public static void main(String[] args)
8     {
9        // crea un panel que contiene nuestro dibujo
10       PanelDibujo panel = new PanelDibujo();
11
12       // crea un nuevo marco para contener el panel
13       JFrame aplicacion = new JFrame();
14
15       // establece el marco para salir cuando se cierre
16       aplicacion.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
17
18       aplicacion.add(panel); // agrega el panel al marco
19       aplicacion.setSize(250, 250); // establece el tamaño del marco
20       aplicacion.setVisible(true); // hace que el marco sea visible
21    }
22 } // fin de la clase PruebaPanelDibujo
```

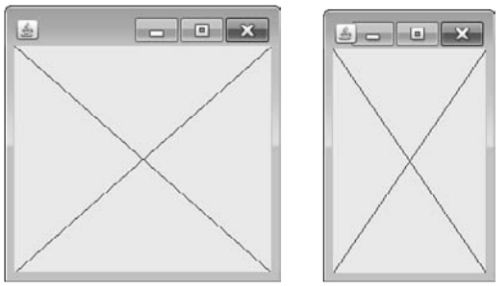

Fig. 4.19 Creación de un objeto `JFrame` para mostrar un objeto `PanelDibujo`

La línea 6 utiliza la palabra clave extends para indicar que la clase `PanelDibujo` es un tipo mejorado de `JPanel`. La palabra clave `extends` representa algo que se denomina relación de herencia, en la cual nuestra nueva clase `PanelDibujo` empieza con los miembros existentes (datos y métodos) de la clase `JPanel`.

La clase de la cual `PanelDibujo` hereda, `JPanel`, aparece a la derecha de la palabra clave `extends`. En esta relación de herencia, a `JPanel` se le conoce como la superclase y `PanelDibujo` es la subclase. Esto produce una clase `PanelDibujo` que tiene los atributos (datos) y comportamientos (métodos) de la clase `JPanel`, así como las nuevas características que agregaremos en nuestra declaración de la clase `PanelDibujo`. En particular, tiene la habilidad de dibujar dos líneas a lo largo de las diagonales del panel. En el capítulo 9 explicaremos con detalle el concepto de herencia. Por ahora, sólo tiene que imitar nuestra clase `PanelDibujo` cuando cree sus propios programas de gráficos.

El método `paintComponent`

Todo `JPanel`, incluyendo nuestro `PanelDibujo`, tiene un método `paintComponent` (líneas 9 a 22), que el sistema llama de manera automática cada vez que necesita mostrar el objeto `PanelDibujo`. El método `paintComponent` debe declararse como se muestra en la línea 9; de no ser así, el sistema no llamará al método. Este método es llamado cuando se muestra un objeto `JPanel` por primera vez en la pantalla, cuando una ventana en la pantalla lo cubree y después lo descubre, y cuando la ventana en la que aparece cambia su tamaño. El método `paintComponent` requiere un argumento, un objeto `Graphics`, que el sistema proporciona por usted cuando llama a `paintComponent`. Este objeto `Graphics` se usa para dibujar líneas, rectángulos, óvalos y otros gráficos.

La primera instrucción en cualquier método `paintComponent` que cree debe ser siempre:

```
super.paintComponent(g);
```

la cual asegura que el panel se despliegue de manera apropiada en la pantalla, antes de empezar a dibujar en él. A continuación, las líneas 14 y 15 llaman a los métodos que la clase `PanelDibujo` hereda de la clase `JPanel`. Como `PanelDibujo` extiende a `JPanel`, éste puede usar cualquier método public

de `JPanel`. Los métodos `getWidth` y `getHeight` devuelven la anchura y la altura del objeto `JPanel`, respectivamente. Las líneas 14 y 15 almacenan estos valores en las variables locales anchura y altura. Por último, las líneas 18 y 21 utilizan la variable `g` de la clase `Graphics` para llamar al método `drawLine`, para que dibuje las dos líneas. El método `drawLine` dibuja una línea entre dos puntos representados por sus cuatro argumentos. Los primeros dos son las coordenadas x y y para uno de los puntos finales de la línea, y los últimos dos son las coordenadas para el otro punto final. Si cambia de tamañoo la ventana, las líneas se escalarán en concordancia, ya que los argumen tos se basan en la anchura y la altura del panel. Al cambiar el tamaño de la ventana en esta aplicación, el sistema llama a `paintComponent` para volver a dibujarr el contenido de `PanelDibujo`.

La clase `PruebaPanelDibujo`

Para mostrar el `PanelDibujo` en la pantalla, debemos colocarlo en una ventana. Usted debe crear una ventana con un objeto de la clase `JFrame`. En `PruebaPanelDibujo.java` (figura 4.19), la línea 3 importa la clase `JFrame` del paquete `javax.swing`. La línea 10 en main crea un objeto `PanelDibujo`, el cual contiene nuestro dibujo, y la línea 13 crea un nuevo objeto JFrame que puede contener y mostrar nuestro panel. La línea 16 llama al método `setDefaultCloseOperation` del método `JFrame` con el argumento `JFrame.EXIT_ON_CLOSE`, para indicar que la aplicación debe terminar cuando el usuario cierre la ventana. La línea 18 utiliza el método `add` de `JFrame` para adjuntarr el objeto `PanelDibujo` al objeto `JFrame`. La línea 19 establece el tamaño del objeto `JFrame`. El método `setSize` recibe dos parámetros que representan la anchura y altura del objeto `JFrame`, respectivamente. Por último, la línea 20 muestra el objeto `JFrame` mediante una llamada a su método `setVisible` con el argumento `true`. Cuando se muestra el objeto `JFrame`, se hace una llamada implícita al método `paintComponent` de `PanelDibujo` (líneas 9 a 22 de la figura 4.18) y se dibujan las dos líneas (vea los resultados de ejemplo en la figura 4.19). Pruebe a cambiar el tamaño de la ventana, y podrá ver que las líneas siempre se dibujan con base en la anchura y altura actuales de la ventana.

Ejercicios del caso de estudio de GUI y gráficos

4.1 Al utilizar ciclos e instrucciones de control para dibujar líneas se pueden lograr muchos diseños interesantes.

a) Cree el diseño que se muestra en la captura de pantalla izquierda de la figura 4.20. Este diseño dibuja líneas que parten desde la esquina superior izquierda, y se despliegan hasta cubrir la mitad superior izquierda del panel. Un método es dividir la anchura y la altura en un número equivalente de pasos (nosotros descubrimos que 15 pasos es una buena cantidad). El primer punto final de una línea siempre estará en la esquina superior izquierda (0,0). El segundo punto final puede encontrarse partiendo desde la esquina inferior izquierda, y avanzando un paso vertical hacia arriba, y uno horizontal hacia la derecha. Dibuje una línea entre los dos puntos finales. Continúe avanzando un paso hacia arriba y a la derecha, para encontrar cada punto final sucesivo. La figura deberá escalarse de manera apropiada conforme usted cambie el tamaño de la ventana.

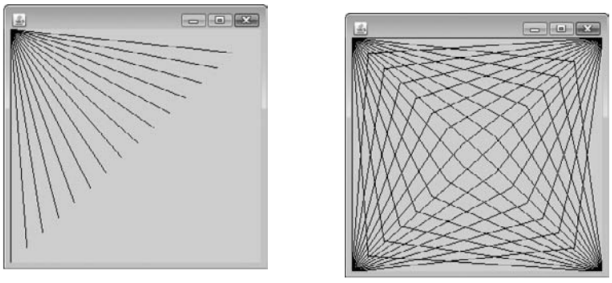

Fig. 4.20 Despliegue de líneas desde una esquina

b) Modifique su respuesta en la parte (a) para hacer que las líneas se desplieguen a partir de las cuatro esquinas, como se muestra en la captura de pantalla derecha de la figura 4.20. Las líneas de esquinas opuestas deberán intersecarse a lo largo de la parte media.

4.2 La figura 4.21 muestra dos diseños adicionales, creados mediante el uso de ciclos `while` y de `drawLine`.

a) Cree el diseño de la captura de pantalla izquierda de la figura 4.21. Empiece por dividir cada borde en un número equivalente de incrementos (elegimos 15 de nuevo). La primera línea empieza en

la esquina superior izquierda y termina un paso a la derecha, en el borde inferior. Para cada línea sucesiva, avance hacia abajo un incremento en el borde izquierdo, y un incremento a la derecha en el borde inferior. Continúe dibujando líneas hasta llegar a la esquina inferior derecha. La figura deberá escalarse a medida que usted cambie el tamaño de la ventana, de manera que los puntos finales siempre toquen los bordes.

b) Modifique su respuesta en la parte (a) para reflejar el diseño en las cuatro esquinas, como se muestra en la captura de pantalla derecha de la figura 4.21.

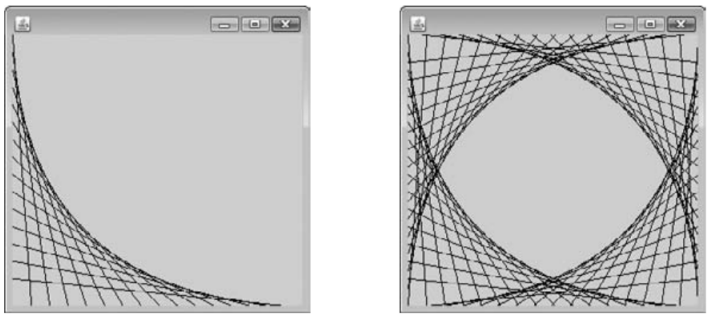

Fig. 4.21 Arte lineal con ciclos y `drawLine`

5.11 Ejemplo práctico de GUI y gráficos: dibujo de rectángulos y óvalos

Esta sección demuestra cómo dibujar rectángulos y óvalos, mediante los métodos `drawRect` y `drawOval` de `Graphics`, respectivamente. Estos métodos se demuestran en la figura 5.27.

```
1  // Fig. 5.27: Figuras.java
2  /// Cómo dibujar una cascada de figuras con base en la elección del usuario.
3  import java.awt.Graphics;
4  import javax.swing.JPanel;
5
6  public class Figuras extends JPanel
7  {
8     private int opcion; // opción del usuario acerca de cuál figura dibujar
9
10    // el constructor establece la opción del usuario
11    public Figuras(int opcionUsuario)
12    {
13       opcion = opcionUsuario;
14    }
15
16    // dibuja una cascada de figuras, empezando desde la esquina superior
         izquierda
17    public void paintComponent(Graphics g)
18    {
19       super.paintComponent(g);
20
21       for (int i = 0; i < 10; i++)
22       {
23          // elige la figura con base en la opción del usuario
24          switch (opcion)
25          {
26             case 1: // dibuja rectángulos
27                g.drawRect(10 + i * 10, 10 + i * 10,
28                   50 + i * 10, 50 + i * 10);
29                break;
30             case 2: // dibuja óvalos
31                g.drawOval(10 + i * 10, 10 + i * 10,
32                   50 + i * 10, 50 + i * 10);
33                break;
34          }
35       }
36    }
37 } // fin de la clase Figuras
```

Fig. 5.27 Cómo dibujar una cascada de figuras con base en la elección del usuario

La línea 6 empieza la declaración de la clase para `Figuras`, que extiende a `JPanel`. La variable de instancia opcion determina si `paintComponent` debe dibujar rectángulos u óvalos. El constructor de `Figuras` inicializa opcion con el valor que se pasa en el parámetro `opcionUsuario`.

El método `paintComponent` realiza en sí el dibujo. Recuerde que la primera instrucción en todo método `paintComponent` debe ser una llamada a `super.paintComponent`, como en la línea 19. Las líneas 21 a 35 iteran 10 veces para dibujar 10 figuras. La instrucción `switch` anidada (líneas 24 a 34) elige entre dibujar rectángulos y dibujar óvalos.

Si opcion es 1, entonces el programa dibuja rectángulos. Las líneas 27 y 28 llaman al método `drawRect` de `Graphics`. El método `drawRect` requiere cuatro argumentos. Los primeros dos representan las coordenadas x y y de la esquina superior izquierda del rectángulo; los siguientes dos simbolizan la anchura y la altura del rectángulo. En este ejemplo, empezamos en la posición 10 píxeles hacia abajo y 10 píxeles a la derecha de la esquina superior izquierda, y cada iteración del ciclo avanza la esquina superior izquierda otros 10 píxeles hacia abajo y a la derecha. La anchura y la altura del rectángulo empiezan en 50 píxeles, y se incrementan por 10 píxeles en cada iteración.

Si opcion es 2, el programa dibuja óvalos. Crea un rectángulo imaginario llamado rectángulo delimitador, y dentro de éste crea un óvalo que toca los puntos medios de todos los cuatro lados. El método `drawOval` (líneas 31 y 32) requiere los mismos cuatro argumentos que el método `drawRect`. Los argumentos especifican la posición y el tamaño del rectángulo delimitador para el óvalo. Los valores que se pasan a `drawOval` en este ejemplo son exactamente los mismos valores que se pasan a `drawRect` en las líneas 27 y 28. Como la anchura y la altura del rectángulo delimitador son idénticas en este ejemplo, las líneas 27 y 28 dibujan un círculo. Como ejercicio, modifique el programa que dibuja rectángulos y óvalos, para ver cómo se relacionan `drawOval` y `drawRect`.

La clase `PruebaFiguras`

La figura 5.28 es responsable de manejar la entrada del usuario y crear una ventana para mostrar el dibujo apropiado, con base en la respuesta del usuario. La línea 3 importa a `JFrame` para manejar la pantalla, y la línea 4 importa a JOptionPane para manejar la entrada. Las líneas 11 a 13 muestran un cuadro de diálogo al usuario y almacenan la respuesta de éste en la variable entrada. Cabe mencionar que al mostrar varias líneas de texto en un `JOptionPane`, hay que usar \n para comenzar una nueva línea, en vez de %n. La línea 15 utiliza el método `parseInt` de `Integer` para convertir el objeto `String` introducido por el usuario en un `int`, y almacena el resultado en la variable opcion. En la línea 18 se crea un objeto `Figuras` y se pasa la opción del usuario al constructor. Las líneas 20 a 25 realizan las operaciones estándar para crear y establecer una ventana en este ejemplo práctico (crear un marco, configurarlo para que la aplicación termine cuando se cierre, agregar el dibujo al marco, establecer su tamaño y hacerlo visible).

```
1  // Fig. 5.28: PruebaFiguras.java
2  // Obtener la entrada del usuario y crear un JFrame para mostrar Figuras.
3  import javax.swing.JFrame; // maneja la visualización
4  import javax.swing.JOptionPane;
5
6  public class PruebaFiguras
7  {
8     public static void main(String[] args)
9     {
10       // obtiene la opción del usuario
11       String entrada = JOptionPane.showInputDialog(
12          "Escriba 1 para dibujar rectangulos\n" +
13          "Escriba 2 para dibujar ovalos");
14
15       int opcion = Integer.parseInt(entrada); // convierte entrada en int
16
17       // crea el panel con la entrada del usuario
18       Figuras panel = new Figuras(opcion);
19
20       JFrame aplicacion = new JFrame(); // crea un nuevo objeto JFrame
21
22       aplicacion.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
23       aplicacion.add(panel);
24       aplicacion.setSize(300, 300);
25       aplicacion.setVisible(true);
26    }
27 } // fin de la clase PruebaFiguras
```

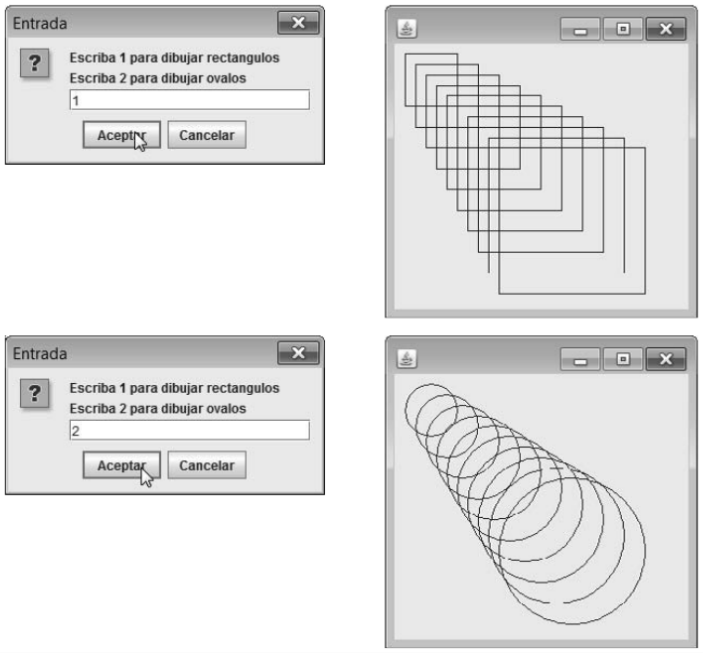

Fig. 5.28 Cómo obtener datos de entrada del usuario y crear un objeto `JFrame` para mostrar `Figuras`

Ejercicios del ejemplo práctico de GUI y gráficos

5.1 Dibuje 12 círculos concéntricos en el centro de un objeto `JPanel` (figura 5.29). El círculo más interno debe tener un radio de 10 píxeles, y cada círculo sucesivo debe contar con un radio 10 píxeles mayor que el anterior. Empiece por buscar el centro del objeto `JPanel`. Para obtener la esquina superior izquierda de un círculo, avance un radio hacia arriba y un radio a la izquierda, partiendo del centro. La anchura y la altura del rectángulo delimitador es el diámetro del círculo (el doble del radio).

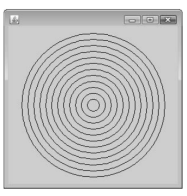

Fig. 5.29 Cómo dibujar círculos concéntricos

5.2 Modifique el ejercicio 5.16 de los ejercicios de fin de capítulo para leer la entrada usando cuadros de diálogo, y mostrar el gráfico de barras usando rectángulos de longitudes variables.

6.13 Ejemplo práctico de GUI y gráficos: colores y figuras rellenas

Aunque podemos crear muchos diseños interesantes sólo con líneas y figuras básicas, la clase Graphics ofrece muchas posibilidades más. Las siguientes dos herramientas que presentaremos son los colores y las figuras rellenas. Al agregar color se enriquecen los dibujos que ve un usuario en la pantalla de la computadora. Las figuras se pueden rellenar con colores sólidos.

Los colores que se muestran en las pantallas de las computadoras se definen con base en sus componentes rojo, verdee y azull (conocidos como valores RGB), los cuales tienen valores enteros de 0 a 255. Cuanto más alto sea el valor de un componente específico, más intensidad de color tendrá esa figura. Java utiliza la clase `Color` (paquete `java.awt`) para representar colores mediante sus valores RGB. Por conveniencia, la clase `Color` contiene varios objetos `static Color` predefinidos: BLACK, BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE y YELLOW. Para acceder a estos objetos predefinidos, se utiliza el nombre de la clase y un punto (.), como en Color.RED. Puede crear colores personalizados pasando los valores de los componentes rojo, verde y azul al constructor de la clase `Color`:

```
public Color(int r, int g, int b)
```

Los métodos `fillRect` y `fillOval` de `Graphics` dibujan rectángulos y óvalos rellenos, respectivamente. Tienen los mismos parámetros que `drawRect` y `drawOval`; los primeros dos parámetros son las coordenadas para la esquina superior izquierda de la figura, mientras que los otros dos determinan su anchura y su altura. El ejemplo de las figuras 6.11 y 6.12 demuestra los colores y las figuras rellenas, al dibujar y mostrar una cara sonriente amarilla en la pantalla.

```
1  // Fig. 6.11: DibujarCaraSonriente.java
2  // Dibuja una cara sonriente usando colores y figuras rellenas.
3  import java.awt.Color;
4  import java.awt.Graphics;
5  import javax.swing.JPanel;
6
7  public class DibujarCaraSonriente extends JPanel
8  {
9     public void paintComponent(Graphics g)
10    {
11       super.paintComponent(g);
12
13       // dibuja la cara
14       g.setColor(Color.YELLOW);
15       g.fillOval(10, 10, 200, 200);
16
17       // dibuja los ojos
18       g.setColor(Color.BLACK);
19       g.fillOval(55, 65, 30, 30);
20       g.fillOval(135, 65, 30, 30);
21
22       // dibuja la boca
23       g.fillOval(50, 110, 120, 60);
24
25       // convierte la boca en una sonrisa
26       g.setColor(Color.YELLOW);
27       g.fillRect(50, 110, 120, 30);
28       g.fillOval(50, 120, 120, 40);
29    }
30 } // fin de la clase DibujarCaraSonriente
```

Fig. 6.11 Cómo dibujar una cara sonriente, con colores y figuras rellenas

Las instrucciones import en las líneas 3 a 5 de la figura 6.11 importan las clases `Color`, `Graphics` y `JPanel`. La clase `DibujarCaraSonriente` (líneas 7 a 30) utiliza la clase `Color` para especificar los colores, y utiliza la clase `Graphics` para dibujar.

La clase `JPanel` proporciona de nuevo el área en la que vamos a dibujar. La línea 14 en el método `paintComponent` utiliza el método `setColor` de `Graphics` para establecer el color actual para dibujar en `Color.YELLOW`. El método `setColor` requiere un argumento, que es el `Color` a establecer como el color para dibujar. En este caso, utilizamos el objeto predefinido `Color.YELLOW`.

La línea 15 dibuja un círculo con un diámetro de 200 para representar la cara; cuando los argumentos anchura y altura son idénticos, el método `fillOval` dibuja un círculo. A continuación, la línea 18 establece el color en `Color.BLACK`, y las líneas 19 y 20 dibujan los ojos. La línea 23 dibuja la boca como un óvalo, pero esto no es exactamente lo que queremos.

Para crear una cara feliz, vamos a retocar la boca. La línea 26 establece el color en `Color.YELLOW`, de manera que cualquier figura que dibujemos se mezclará con la cara. La línea 27 dibuja un rectángulo con la mitad de altura que la boca. Esto borra la mitad superior de la boca, dejando sólo la mitad inferior. Para crear una mejor sonrisa, la línea 28 dibuja otro óvalo para cubrir ligeramente la porción superior de la boca. La clase `PruebaDibujarCaraSonriente` (figura 6.12) crea y muestra un objeto `JFrame` que contiene el dibujo. Cuando se muestra el objeto `JFrame`, el sistema llama al método `paintComponent` para dibujar la cara sonriente.

```
1  // Fig. 6.12: PruebaDibujarCaraSonriente.java
2  // Aplicación de prueba que muestra una cara sonriente.
3  import javax.swing.JFrame;
4
5  public class PruebaDibujarCaraSonriente
6  {
7     public static void main(String[] args)
8     {
9        DibujarCaraSonriente panel = new DibujarCaraSonriente();
10       JFrame aplicacion = new JFrame();
11
12       aplicacion.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13       aplicacion.add(panel);
14       aplicacion.setSize(230, 250);
15       aplicacion.setVisible(true);
16    }
17 } // fin de la clase PruebaDibujarCaraSonriente
```

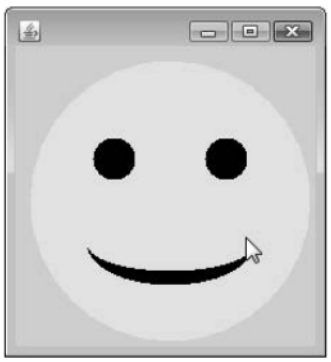

Fig. 6.12 Aplicación de prueba que muestra una cara sonriente

Ejercicios del ejemplo de GUI y gráficos

6.1 Con el método `fillOval`, dibuje un tiro al blanco que alterne entre dos colores aleatorios, como en la figura 6.13. Use el constructor `Color(int r, int g, int b)` con argumentos aleatorios para generar colores aleatorios.

6.2 Cree un programa para dibujar 10 figuras rellenas al azar en colores, posiciones y tamaños aleatorios (figura 6.14). El método `paintComponent` debe contener un ciclo que itere 10 veces. En cada iteración, el ciclo debe determinar si se dibujará un rectángulo o un óvalo relleno, crear un color aleatorio y elegir tanto las coordenadas como las medidas al azar. Las coordenadas deben elegirse con base en la anchura y la altura del panel. Las longitudes de los lados deben limitarse a la mitad de la anchura o altura de la ventana.

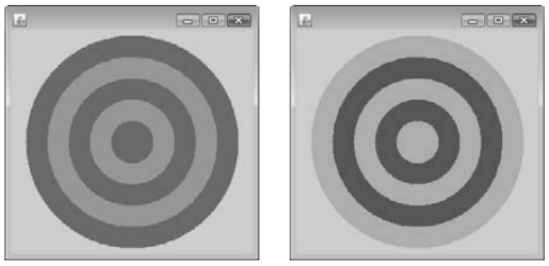

Fig. 6.13 Un tiro al blanco con dos colores alternantes al azar

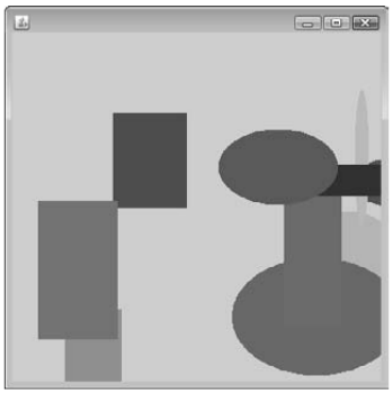

Fig. 6.14 Figuras generadas al azar

7.17 Ejemplo práctico de GUI y gráficos: cómo dibujar arcos

Mediante el uso de las herramientas para gráficos de Java, podemos crear dibujos complejos que, si los codificáramos línea por línea, sería un proceso más tedioso. En las figuras 7.25 y 7.26 utilizamos arreglos e instrucciones de repetición para dibujar un arco iris, mediante el uso del método `fillArc` de `Graphics`. El proceso de dibujar arcos en Java es similar a dibujar óvalos; un arco es simplemente una sección de un óvalo.

En la figura 7.25, las líneas 10 y 11 declaran y crean dos nuevas constantes de colores: VIOLETA e INDIGO. Como tal vez lo sepa, los colores de un arco iris son rojo, naranja, amarillo, verde, azul, índigo y violeta. Java tiene constantes predefinidas sólo para los primeros cinco colores. Las líneas 15 a 17 inicializan un arreglo con los colores del arco iris, empezando con los arcos más interiores primero. El arreglo empieza con dos elementos `Color.WHITE`, que como veremos pronto, son para dibujar los arcos vacíos en el centro del arco iris. Las variables de instancia se pueden inicializar al momento de declararse, como se muestra en las líneas 10 a 17. El constructor (líneas 20 a 23) contiene una sola instrucción que llama al método `setBackground` (heredado de la clase `JPanel`) con el parámetro `Color.WHITE`. El método `setBackground` recibe un solo argumento `Color` y establece el color de fondo del componente a ese color.

```
1  // Fig. 7.25: DibujoArcoIris.java
2  // Dibuja un arcoiris mediante el uso de arcos y un arreglo de colores.
3  import java.awt.Color;
4  import java.awt.Graphics;
5  import javax.swing.JPanel;
6
7  public class DibujoArcoIris extends JPanel
8  {
9     // Define los colores índigo y violeta
10    private final static Color VIOLETA = new Color(128, 0, 128);
11    private final static Color INDIGO = new Color(75, 0, 130);
12
13    // los colores a usar en el arco iris, empezando desde los más interiores
14    // Las dos entradas de color blanco producen un arco vacío en el centro
15    private Color[] colores =
16       { Color.WHITE, Color.WHITE, VIOLETA, INDIGO, Color.BLUE,
17         Color.GREEN, Color.YELLOW, Color.ORANGE, Color.RED };
18
19    // constructor
20    public DibujoArcoIris()
21    {
22       setBackground(Color.WHITE); // establece el fondo al color blanco
23    }
24
25    // dibuja un arco iris, usando arcos concéntricos
26    public void paintComponent(Graphics g)
27    {
28       super.paintComponent(g);
29
30       int radio = 20; // el radio de un arco
31
32       // dibuja el arco iris cerca de la parte central inferior
33       int centroX = getWidth() / 2;
34       int centroY = getHeight() - 10;
35
36       // dibuja arcos rellenos, empezando con el más exterior
37       for (int contador = colores.length; contador > 0; contador--)
38       {
39          // establece el color para el arco actual
40          g.setColor(colores[contador - 1]);
41
42          // rellena el arco desde 0 hasta 180 grados
43          g.fillArc(centroX - contador * radio,
44             centroY - contador * radio,
45             contador * radio * 2, contador * radio * 2, 0, 180);
46       }
47    }
48 } // fin de la clase DibujoArcoIris
```

Fig. 7.25 Dibujo de un arco iris mediante el uso arcos y un arreglo de colores

La línea 30 en `paintComponent` declara la variable local radio, que determina el radio de cada arco. Las variables locales `centroX` y `centroY` (líneas 33 y 34) determinan la ubicación del punto medio en la base del arco iris. El ciclo en las líneas 37 a 46 utiliza la variable de control contador para contar en forma regresiva, partiendo del final del arreglo, dibujando los arcos más grandes primero y colocando cada arco más pequeño encima del anterior. La línea 40 establece el color para dibujar el arco actual del arreglo. La razón por la que tenemos entradas `Color.WHITE` al principio del arreglo es para crear el arco vacío en el centro. De no ser así, el centro del arco iris sería tan solo un semicírculo sólido color violeta. Puede cambiar los colores individuales y el número de entradas en el arreglo para crear nuevos diseños.

La llamada al método `fillArc` en las líneas 43 a 45 dibuja un semicírculo relleno. El método `fillArc` requiere seis parámetros. Los primeros cuatro representan el rectángulo delimitador en el cual se dibujará el arco. Los primeros dos de estos cuatro especifican las coordenadas para la esquina superior izquierda del rectángulo delimitador, y los siguientes dos especifican su anchura y su altura. El quinto parámetro es el ángulo inicial en el óvalo, y el sexto especifica el barrido, o la cantidad de arco que se

cubrirá. El ángulo inicial y el barrido se miden en grados, en donde los cero grados apuntan a la derecha. Un barrido positivo dibuja el arco en sentido contrario a las manecillas del reloj, en tanto que un barrido negativo dibuja el arco en sentido de las manecillas del reloj. Un método similar a `fillArc` es `drawArc`; requiere los mismos parámetros que `fillArc`, pero dibuja el borde del arco, en vez de rellenarlo.

La clase `PruebaDibujoArcoIris` (figura 7.26) crea y establece un objeto `JFrame` para mostrar el arco iris en la pantalla. Una vez que el programa hace visible el objeto `JFrame`, el sistema llama al método `paintComponent` en la clase `DibujoArcoIris` para dibujar el arco iris en la pantalla.

Ejercicio del caso de estudio de GUI y gráficos

7.1 (**Dibujo de espirales**) En este ejercicio, dibujará espirales con los métodos `drawLine` y `drawArc`.

a) Dibuje una espiral con forma cuadrada (como en la captura de pantalla izquierda de la figura 7.27), centrada en el panel, con el método `drawLine`. Una técnica es utilizar un ciclo que incremente la longitud de la línea después de dibujar cada segunda línea. La dirección en la cual se dibujará la siguiente línea debe ir después de un patrón distinto, como abajo, izquierda, arriba, derecha.

b) Dibuje una espiral circular (como en la captura de pantalla derecha de la figura 7.27); use el método `drawArc` para dibujar un semicírculo a la vez. Cada semicírculo sucesivo deberá tener un radio más grande (según lo especificado mediante la anchura del rectángulo delimitador) y debe seguir dibujando en donde terminó el semicírculo anterior.

```
1  // Fig. 7.26: PruebaDibujoArcoIris.java
2  // Aplicación de prueba para mostrar un arco iris.
3  import javax.swing.JFrame;
4
5  public class PruebaDibujoArcoIris
6  {
7     public static void main(String[] args)
8     {
9        DibujoArcoIris panel = new DibujoArcoIris();
10       JFrame aplicacion = new JFrame();
11
12       aplicacion.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13       aplicacion.add(panel);
14       aplicacion.setSize(400, 250);
15       aplicacion.setVisible(true);
16 }
17 } // fin de la clase PruebaDibujoArcoIris
```

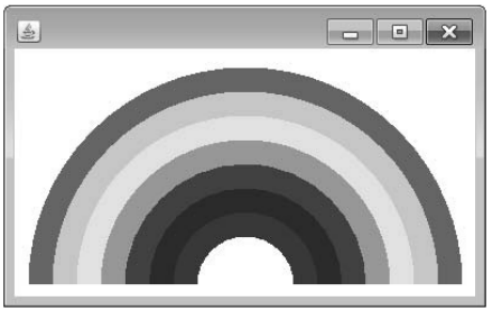

Fig. 7.27 Dibujo de una espiral con `drawLine` (izquierda) y `drawArc` (derecha)

8.16 Ejemplo práctico de GUI y gráficos: uso de objetos con gráficos

La mayoría de los gráficos que ha visto hasta este punto no varían cada vez que se ejecuta el programa. El ejercicio 6.2 en la sección 6.13 le pidió que creara un programa para generar figuras y colores al azar. En ese ejercicio, el dibujo cambiaba cada vez que el sistema llamaba a `paintComponent`. Para crear un dibujo más consistente que permanezca sin cambios cada vez que se dibuja, debemos almacenar información acerca de las figuras mostradas, para que podamos reproducirlas cada vez

que el sistema llame a paintComponent. Para ello, crearemos un conjunto de clases de figuras que almacenan información sobre cada figura. Haremos a estas clases “inteligentes”, al permitir que sus objetos se dibujen a sí mismos mediante el uso de un objeto Graphics.

La clase MiLinea

La figura 8.17 declara la clase `MiLinea`, que tiene todas estas capacidades. La clase `MiLinea` importa a `Color` y a `Graphics` (líneas 3 y 4). Las líneas 8 a 11 declaran variables de instancia para las coordenadas de los puntos finales necesarios para dibujar una línea, y la línea 12 declara la variable de instancia que almacena el color de la línea. El constructor en las líneas 15 a 22 recibe cinco parámetros, uno para cada variable de instancia que inicializa. El método dibujar en las líneas 25 a 29 requiere un objeto `Graphics` y lo utiliza para dibujar la línea en el color apropiado y entre los puntos finales.

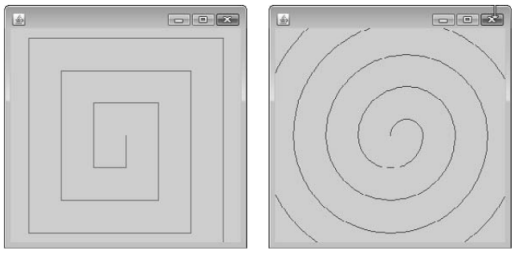

Fig. 8.17 La clase `MiLinea` representa a una línea

La clase `PanelDibujo`

En la figura 8.18, declaramos la clase `PanelDibujo`, que generará objetos aleatorios de la clase `MiLinea`. La línea 12 declara un arreglo `MiLinea` llamado lineas para almacenar las líneas a dibujar. Dentro del constructor (líneas 15 a 37), la línea 17 establece el color de fondo a `Color.WHITE`. La línea 19 crea el arreglo con una longitud aleatoria entre 5 y 9. El ciclo en las líneas 22 a 36 crea un nuevo objeto MiLinea para cada elemento en el arreglo. Las líneas 25 a 28 generan coordenadas aleatorias para los puntos finales de cada línea, y las líneas 31 y 32 generan un color aleatorio para la línea. La línea 35 crea un nuevo objeto `MiLinea` con los valores generados al azar, y lo almacena en el arreglo. El método `paintComponent` itera a través de los objetos `MiLinea` en el arreglo lineas usando una instrucción for mejorada (líneas 45 y 46). Cada iteración llama al método dibujar del objeto `MiLinea` actual, y le pasa el objeto `Graphics` para dibujar en el panel.

```
1  // Fig. 8.17: MiLinea.java
2  // La clase MiLinea representa a una línea.
3  import java.awt.Color;
4  import java.awt.Graphics;
5
6  public class MiLinea
7  {
8     private int x1; // coordenada x del primer punto final
9     private int y1; // coordenada y del primer punto final
10    private int x2; // coordenada x del segundo punto final
11    private int y2; // coordenada y del segundo punto final
12    private Color color; // el color de esta figura
13
14    // constructor con valores de entrada
15    public MiLinea(int x1, int y1, int x2, int y2, Color color)
16    {
17       this.x1 = x1;
18       this.y1 = y1;
19       this.x2 = x2;
20       this.y2 = y2;
21       this.color = color;
22    }
23
24    // Dibuja la línea en el color específico
25    public void dibujar(Graphics g)
26    {
27       g.setColor(color);
28       g.drawLine(x1, y1, x2, y2);
29    }
30 } // fin de la clase MiLinea
```

Fig. 8.18 Programa que utiliza la clase `MiLinea` para dibujar líneas al azar

La clase `PruebaDibujo`

La clase `PruebaDibujo` en la figura 8.19 establece una nueva ventana para mostrar nuestro dibujo. Como sólo estableceremos una vez las coordenadas para las líneas en el constructor, el dibujo no cambia si se hace una llamada a `paintComponent` para actualizar el dibujo en la pantalla.

```
1  // Fig. 8.18: PanelDibujo.java
2  // Programa que utiliza la clase MiLinea
3  // para dibujar líneas al azar.
4  import java.awt.Color;
5  import java.awt.Graphics;
6  import java.security.SecureRandom;
7  import javax.swing.JPanel;
8
9  public class PanelDibujo extends JPanel
10 {
11    private SecureRandom numerosAleatorios = new SecureRandom();
12    private MiLinea[] lineas; // arreglo de lineas
13
14    // constructor, crea un panel con figuras al azar
15    public PanelDibujo()
16    {
17       setBackground(Color.WHITE);
18
19       lineas = new MiLinea[5 + numerosAleatorios.nextInt(5)];
20
21       // crea lineas
22       for (int cuenta = 0; cuenta < lineas.length; cuenta++)
23       {
24          // genera coordenadas aleatorias
25          int x1 = numerosAleatorios.nextInt(300);
26          int y1 = numerosAleatorios.nextInt(300);
27          int x2 = numerosAleatorios.nextInt(300);
28          int y2 = numerosAleatorios.nextInt(300);
29
30          // genera un color aleatorio
31          Color color = new Color(numerosAleatorios.nextInt(256),
32             numerosAleatorios.nextInt(256), numerosAleatorios.nextInt(256));
33
34          // agrega la línea a la lista de líneas a mostrar en pantalla
35          lineas[cuenta] = new MiLinea(x1, y1, x2, y2, color);
36       }
37    }
38
39    // para cada arreglo de figuras, dibuja las figuras individuales
40    public void paintComponent(Graphics g)
41    {
42       super.paintComponent(g);
43
44       // dibuja las líneas
45       for (MiLinea linea : lineas)
46          linea.dibujar(g);
47    }
48 } // fin de la clase PanelDibujo
```

Fig. 8.19 Creación de un objeto `JFrame` para mostrar un `PanelDibujo` en pantalla

Ejercicio del ejemplo práctico de GUI y gráficos

8.1 Extienda el programa de las figuras 8.17 a 8.19 para dibujar rectángulos y óvalos al azar. Cree las clases `MiRectangulo` y `MiOvalo`. Ambas deben incluir las coordenadas x1, y1, x2, y2, un color y una bandera `boolean` para determinar si la figura es rellena. Declare un constructor en cada clase con argumentos para inicializar todas las variables de instancia. Para ayudar a dibujar rectángulos y óvalos, cada clase debe proporcionar los métodos `obtenerXSupIzq`, `obtenerYSupIzq`, `obtenerAnchura` y `obtenerAltura`, que calculen la coordenada x superior izquierda, la coordenada y superior izquierda, la anchura y la altura, respectivamente. La coordenada x superior izquierda es el más pequeño de los dos valores de coordenada x, la coordenada y superior izquierda es el más pequeño de los dos valores

de coordenada y, la anchura es el valor absoluto de la diferencia entre los dos valores de coordenada x, y la altura es el valor absoluto de la diferencia entre los dos valores de coordenada y.

La clase `PanelDibujo`, que extiende a `JPanel` y se encarga de la creación de las figuras, debe declarar tres arreglos, uno para cada tipo de figura. La longitud de cada arreglo debe ser un número aleatorio entre 1 y 5. El constructor de la clase `PanelDibujo` debe llenar cada uno de los arreglos con figuras de posición, tamaño, color y relleno aleatorios. Además, modifique las tres clases de figuras para incluir lo siguiente:

a) Un constructor sin argumentos que establezca todas las coordenadas de la figura a 0, el color de la figura a `Color.BLACK` y la propiedad de relleno a false (sólo en `MiRectangulo` y `MiOvalo`).

b) Métodos establecerr para las variables de instancia en cada clase. Los métodos para establecer el valor de una coordenada deben verificar que el argumento sea mayor o igual a cero, antes de establecer la coordenada; si no es así, deben establecer la coordenada a cero. El constructor debe llamar a los métodos establecer, en vez de inicializar las variables locales directamente.

c) Métodos obtener para las variables de instancia en cada clase. El método dibujar debe hacer referencia a las coordenadas mediante los métodos obtener, en vez de acceder a ellas de manera directa.

9.7 Ejemplo práctico de GUI y gráficos: mostrar texto e imágenes usando etiquetas

Los programas usan a menudo etiquetas cuando necesitan mostrar información o instrucciones al usuario en una interfaz gráfica de usuario. Las etiquetas son una forma conveniente de identificar componentes de la GUI en la pantalla, y de mantener al usuario informado sobre el estado actual del programa. En Java, un objeto de la clase JLabel (del paquete javax.swing) puede mostrar texto, una imagen o ambos. El ejemplo de la figura 9.13 demuestra varias características de JLabel, incluyendo una etiqueta de texto simple, una etiqueta de imagen y una con texto e imagen.

```
1  // Fig 9.13: DemoLabel.java
2  // Demuestra el uso de etiquetas.
3  import java.awt.BorderLayout;
4  import javax.swing.ImageIcon;
5  import javax.swing.JLabel;
6  import javax.swing.JFrame;
7
8  public class DemoLabel
9  {
10    public static void main(String[] args)
11    {
12       // Crea una etiqueta con texto solamente
13       JLabel etiquetaNorte = new JLabel("Norte");
14
15       // crea un icono a partir de una imagen, para poder colocarla en un objeto
            JLabel
16       ImageIcon etiquetaIcono = new ImageIcon("GUItip.gif");
17
18       // crea una etiqueta con un icono en vez de texto
19       JLabel etiquetaCentro = new JLabel(etiquetaIcono);
20
21       // crea otra etiqueta con un icono
22       JLabel etiquetaSur = new JLabel(etiquetaIcono);
23
24       // establece la etiqueta para mostrar texto (así como un icono)
25       etiquetaSur.setText("Sur");
26
27       // crea un marco para contener las etiquetas
28       JFrame aplicacion = new JFrame();
29
30       aplicacion.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
31
32       // agrega las etiquetas al marco; el segundo argumento especifica
33       // en qué parte del marco se va a agregar la etiqueta
34       aplicacion.add(etiquetaNorte, BorderLayout.NORTH);
35       aplicacion.add(etiquetaCentro, BorderLayout.CENTER);
36       aplicacion.add(etiquetaSur, BorderLayout.SOUTH);
37
38       aplicacion.setSize(300, 300);
39       aplicacion.setVisible(true);
40    } // fin de main
41 } // fin de la clase DemoLabel
```

Fig. 9.13 JLabel con texto y con imágenes

Las líneas 3 a 6 importan las clases que necesitamos para mostrar los objetos JLabel. BorderLayout del paquete java.awt que contienen constantes que especifican en dónde podemos colocar componentes de GUI en el objeto JFrame. La clase ImageIcon representa una imagen que puede mostrarse en un JLabel, y la clase JFrame representa la ventana que contiene todas las etiquetas.

La línea 13 crea un objeto JLabel que muestra el argumento de su constructor: la cadena Norte. La línea 16 declara la variable local etiquetaIcono y le asigna un nuevo objeto ImageIcon. El constructor para ImageIcon recibe un objeto String que especifica la ruta del archivo de la imagen. Como sólo especificamos un nombre de archivo, Java supone que se encuentra en el mismo directorio que la clase DemoLabel. ImageIcon puede cargar imágenes en los formatos GIF, JPEG y PNG. La línea 19 declara e inicializa la variable local etiquetaCentro con un objeto JLabel que muestra el objeto etiquetaIcono. La línea 22 declara e inicializa la variable local etiquetaSur con un objeto JLabel similar al de la línea 19. Sin embargo, la línea 25 llama al método setText para modificar el texto que muestra la etiqueta. El método setText puede llamarse en cualquier objeto JLabel para modificar su texto. Este objeto JLabel muestra tanto el icono como el texto.

La línea 28 crea el objeto JFrame que muestra a los objetos JLabel, y la línea 30 indica que el programa debe terminar cuando se cierre el objeto JFrame. Para adjuntar las etiquetas al objeto JFrame en las líneas 34 a 36, llamamos a una versión sobrecargada del método add que recibe dos parámetros. El primero es el componente que deseamos adjuntar, y el segundo es la región en la que debe colocarse. Cada objeto JFrame tiene un esquema asociado, que ayuda al JFrame a posicionar los componentes de la GUI que tiene adjuntos. El esquema predeterminado para un objeto JFrame se conoce como BorderLayout, y tiene cinco regiones: NORTH (arriba), SOUTH (abajo), EAST (lado derecho), WEST (lado izquierdo) y CENTER (centro). Cada una de estas regiones se declara como una constante en la clase BorderLayout. Al llamar al método add con un argumento, el objeto JFrame coloca el componente en la región CENTER de manera automática. Si una posición ya contiene un componente, entonces el nuevo toma su lugar. Las líneas 38 y 39 establecen el tamaño del objeto JFrame y lo hacen visible en pantalla.

Ejercicio del ejemplo práctico de GUI y gráficos

9.1 Modifique el ejercicio 8.1 del ejemplo práctico de GUI y gráficos para incluir un objeto JLabel como barra de estado, que muestre las cuentas que representan el número de cada figura mostrada. La clase PanelDibujo debe declarar un método para devolver un objeto String que contenga el texto de estado. En main, primero cree el objeto PanelDibujo, y después el objeto JLabel con el texto de estado como argumento para el constructor de JLabel. Adjunte el objeto JLabel a la región SOUTH del objeto JFrame, como se muestra en la figura 9.14.

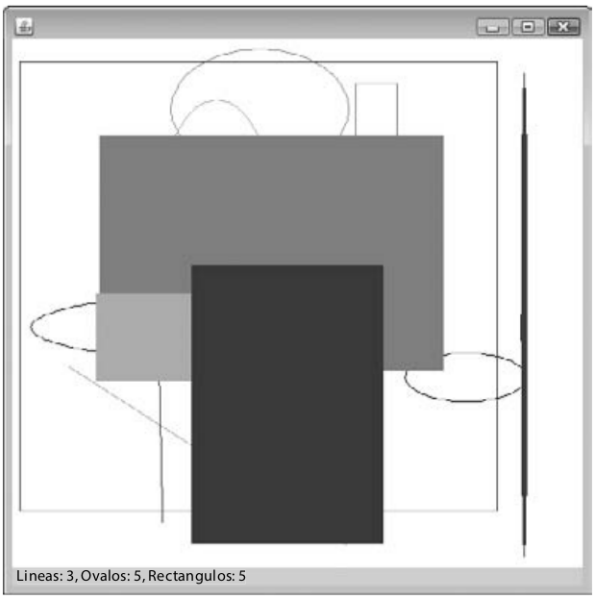

Fig. 9.14 Objeto JLabel que muestra las estadísticas de las figuras

10.11 Ejemplo práctico de GUI y gráficos: realización de dibujos mediante el polimorfismo

Tal vez haya observado en el programa de dibujo que creamos en el ejercicio 8.1 del ejemplo práctico de GUI y gráficos (y que modificamos en el ejercicio 9.1 del ejemplo práctico de GUI y gráficos) que existen muchas similitudes entre las clases de figuras. Mediante la herencia, podemos "factorizar" las características comunes de las tres clases y colocarlas en una sola superclasee de figura. Después, podemos manipular objetos de los tres tipos de figuras en forma polimórficaa usando variables del tipo de la superclase. Al eliminar la redundancia en el código se producirá un programa más pequeño y flexible, que será más fácil de mantener.

Ejercicios del ejemplo práctico de GUI y gráficos

10.1 Modifique las clases `MiLinea`, `MiOvalo` y `MiRectangulo` de los ejercicios 8.1 y 9.1 del ejemplo práctico de GUI y gráficos, para crear la jerarquía de clases de la figura 10.17. Las clases de la jerarquía `MiFigura` deben ser clases de figuras "inteligentes", que sepan cómo dibujarse a sí mismas (si se les proporciona un objeto `Graphics` que les indique en dónde deben dibujarse). Una vez que el programa cree un objeto a partir de esta jerarquía, podrá manipularlo polimórficamentee por el resto de su vida como un objeto `MiFigura`.

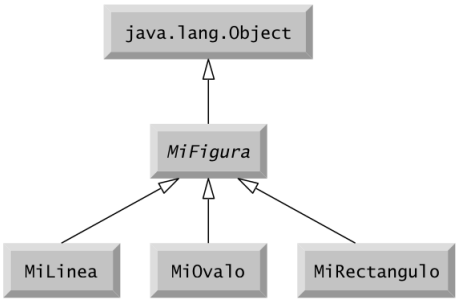

Fig. 10.17 La jerarquía `MiFigura`

En su solución, la clase `MiFigura` de la figura 10.17 debee ser `abstract`. Como `MiFigura` representa a cualquier figura en general, no es posible implementar un método dibujar sin saber exactamente qué figura es. Los datos que representan las coordenadas y el color de las figuras en la jerarquía deben declararse como miembros private de la clase `MiFigura`. Además de los datos comunes, la clase `MiFigura` debe declarar los siguientes métodos:

a) Un constructor sin argumentos que establezca todas las coordenadas de la figura en 0, y el color en `Color.BLACK`

b) Un constructor que inicialice las coordenadas y el color con los valores de los argumentos suministrados.

c) Métodos establecer para las coordenadas individuales y el color, que permitan al programador establecer cualquier pieza de datos de manera independiente, para una figura en la jerarquía.

d) Métodos obtener para las coordenadas individuales y el color, que permitan al programador obtener cualquier pieza de datos de manera independiente, para una figura en la jerarquía.

e) El método `abstract`

```
public abstract void dibujar(Graphics g);
```

que se llamará desde el método `paintComponent` del programa para dibujar una figura en la pantalla.

Para asegurar un correcto encapsulamiento, todos los datos en la clase `MiFigura` deben ser `private`. Para esto se requiere declarar métodos `establecer` y `obtener` apropiados para manipular los datos. La clase `MiLinea` debe proporcionar un constructor sin argumentos y uno con argumentos para las coordenadas y el color. Las clases `MiOvalo` y `MiRectangulo` deben proporcionar un constructor sin argumentos y uno con argumentos para las coordenadas, para el color y para determinar si la figura es rellena. El

constructor sin argumentos debe, además, establecer los valores predeterminados, y la figura como una figura sin relleno.

Puede dibujar líneas, rectángulos y óvalos si conoce dos puntos en el espacio. Las líneas requieren coordenadas x1, y1, x22 y y2. El método `drawLine` de la clase `Graphics` conectará los dos puntos suministrados con una línea. Si tiene los mismos cuatro valores de coordenadas (x1, y1, x22 y y2) para óvalos y rectángulos, puede calcular los cuatro argumentos necesarios para dibujarlos. Cada uno requiere un valor de coordenada x superior izquierda (el menor de los dos valores de coordenada xx), un valor de coordenada y superior izquierda (el menor de los dos valores de coordenada y), una anchura (el valor absoluto de la diferencia entre los dos valores de coordenada xx) y una altura (el valor absoluto de la diferencia entre los dos valores de coordenada y). Los rectángulos y óvalos también deben tener una bandera relleno, que determine si la figura se dibujará con un relleno.

No debe haber variables `MiLinea`, `MiOvalo` o `MiRectangulo` en el programa; sólo variables `MiFigura` que contengan referencias a objetos `MiLinea`, `MiOvalo` y `MiRectangulo`. El programa debe generar figuras aleatorias y almacenarlas en un arreglo de tipo `MiFigura`. El método `paintComponent` debe recorrer el arreglo `MiFigura` y dibujar cada una de las figuras mediante una llamada polimórfica al método dibujar de cada figura.

Permita al usuario que especifique (mediante un diálogo de entrada) el número de figuras a generar. Después, el programa generará y mostrará las figuras en pantalla, junto con una barra de estado para informar al usuario cuántas figuras de cada tipo se crearon.

10.2 ((**Modificación de la aplicación de dibujo**) En el ejercicio anterior, usted creó una jerarquía `MiFigura` en la cual las clases `MiLinea`, `MiOvalo` y `MiRectangulo` extienden directamente a `MiFigura`. Si su jerarquía estuviera diseñada de manera apropiada, debería poder ver las similitudes entre las clases `MiOvalo` y `MiRectangulo`. Rediseñe y vuelva a implementar el código de las clases `MiOvalo` y `MiRectangulo`, para "factorizar" las características comunes en la clase abstracta `MiFiguraDelimitada`, para producir la jerarquía de la figura 10.18.

La clase `MiFiguraDelimitada` debe declarar dos constructores que imiten a los de `MiFigura`, sólo con un parámetro adicional para ver si la figura es rellena. La clase `MiFiguraDelimitada` también debe declarar métodos obtener y establecer para manipular la bandera de relleno y los métodos que calculan la coordenada x superior izquierda, la coordenada y superior izquierda, la anchura y la altura. Recuerde que los valores necesarios para dibujar un óvalo o un rectángulo se pueden calcular a partir de dos coordenadas (x, yy). Si se diseñan de manera apropiada, las nuevas clases `MiOvalo` y `MiRectangulo` deberán tener dos constructores y un método dibujar cada una.

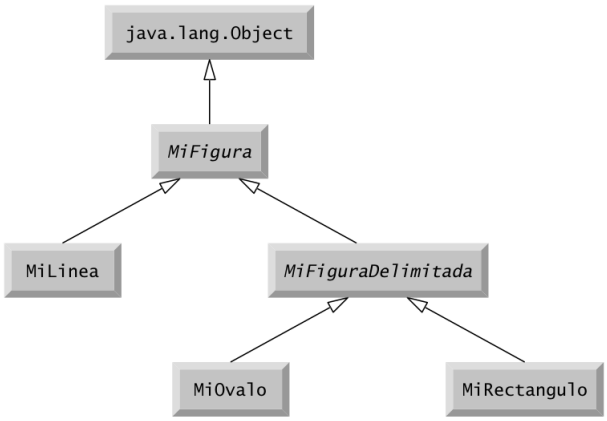

Fig. 10.18 Jerarquía `MiFigura` con `MiFiguraDelimitada`