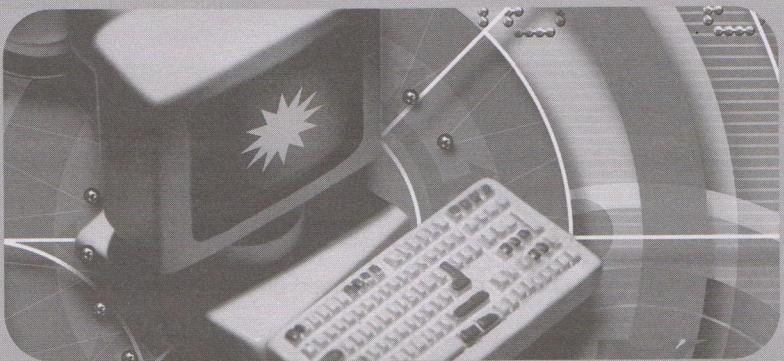


# CAPÍTULO 3



## Uso de métodos gráficos

En este capítulo conoceremos:

- La naturaleza de los eventos.
- Cómo dibujar figuras con métodos gráficos.
- El uso de los parámetros.
- Cómo comentar programas.
- Cómo usar colores.
- El concepto de secuencia.

### ● Introducción

El término *gráficos de computadora* evoca una variedad de posibilidades. Podríamos estar hablando de una película de Hollywood generada por computadora, de una fotografía estática o de una imagen más simple, construida a partir de líneas. En este capítulo nos limitaremos a las imágenes estáticas elaboradas a partir de formas simples, y nos enfocaremos en el uso de métodos de biblioteca para crearlas. También introduciremos el uso de un botón para permitir la interacción del usuario.

### ● Eventos

Muchos programas se crean de cierta forma para permitir la interacción del usuario mediante una GUI (Interfaz Gráfica de Usuario). Dichos programas proveen botones, campos de texto, barras de desplazamiento, etc. En términos de Java, cuando el usuario manipula el ratón y el teclado *crea* "eventos" a los que el programa responde. Algunos eventos comunes son:

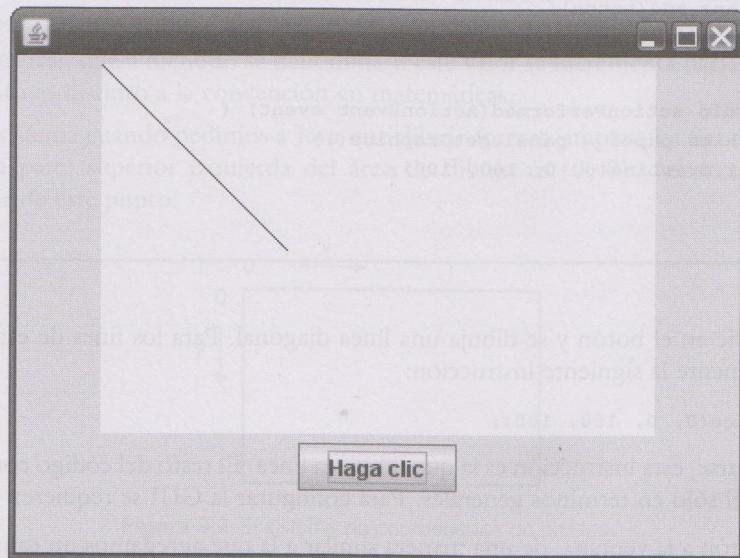
- El clic de del ratón.
- Oprimir una tecla.
- El uso de un control deslizable para desplazarnos por ciertos valores.

El sistema de Java clasifica los eventos en varias categorías. Por ejemplo, desplazarse por una página se considera un evento de “cambio”, mientras que hacer clic en un botón se considera como un evento de “acción”.

Al escribir un programa de Java, debe asegurarse de que el programa detecte los eventos; de lo contrario no ocurrirá nada. La transmisión de un evento (como un clic del ratón) a un programa no se lleva a cabo en forma automática; debemos configurar el programa para que “detecte” los tipos de eventos. Por fortuna, el código para ello es estándar y lo podemos reutilizar en todos los programas, en vez de tener que crearlo de nuevo para cada programa.

Al proceso de responder a un evento se le conoce como “manejar” el evento.

El siguiente programa provee un botón. La figura 3.1 muestra la pantalla.



**Figura 3.1** Pantalla del programa `EjemploDibujo` después de hacer clic en el botón.

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

public class EjemploDibujo extends JFrame
    implements ActionListener {

    private JButton button;
    private JPanel panel;
```

```
public static void main(String[] args) {
    EjemploDibujo marco = new EjemploDibujo();
    marco.setSize(400, 300);
    marco.createGUI();
    marco.setVisible(true);
}

private void createGUI() {
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    Container ventana = getContentPane();
    ventana.setLayout(new FlowLayout());

    panel = new JPanel();
    panel.setPreferredSize(new Dimension(300, 200));
    panel.setBackground(Color.white);
    ventana.add(panel);

    boton = new JButton("Haz clic");
    ventana.add(boton);
    boton.addActionListener(this);
}

public void actionPerformed(ActionEvent event) {
    Graphics papel = panel.getGraphics();
    papel.drawLine(0, 0, 100, 100);
}
```

El usuario hace clic en el botón y se dibuja una línea diagonal. Para los fines de este capítulo, nos interesa principalmente la siguiente instrucción:

```
papel.drawLine(0, 0, 100, 100);
```

Como es de esperarse, esta instrucción es la que dibuja la línea. El resto del código configura la GUI, y hablaremos de él sólo en términos generales. Para configurar la GUI se requiere:

- Agregar un botón a la ventana, de una manera similar a la que agregamos un campo de texto en el capítulo 2.
- Agregar un “panel” que utilizaremos para dibujar.
- Preparar el programa para que detecte los clic del ratón (clasificados como eventos de acción).

El siguiente punto es muy importante: en capítulos posteriores veremos la creación de interfaces de usuario. Por ahora consideraremos el código anterior de la GUI como estándar.

## ● El evento de clic de botón

El principal evento en este programa se genera cuando el usuario hace clic en el botón “**Haz clic**”. Un clic de botón hace que el programa ejecute esta sección del programa:

```

public void actionPerformed(ActionEvent event) {
    Graphics papel = panel.getGraphics();
    papel.drawLine(0, 0, 100, 100);
}

```

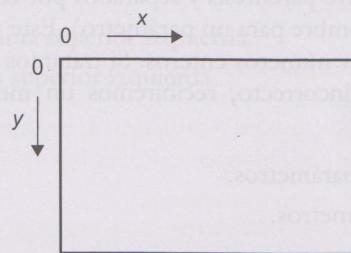
Esta sección del programa es un *método*, conocido como `actionPerformed`. Cuando el usuario hace clic en el botón, el sistema de Java invoca al método y se ejecutan en secuencia las instrucciones entre la llave de apertura (`{`) y la de cierre (`}`). Aquí es en donde colocamos nuestras instrucciones de dibujo. Ahora veremos los detalles sobre el dibujo de figuras.

## ● El sistema de coordenadas de gráficos

Los gráficos de Java se basan en píxeles. Un píxel es un pequeño punto en la pantalla que se puede cambiar a un color específico. Cada píxel se identifica mediante un par de números (sus coordenadas), empezando desde cero:

- La posición horizontal, que a menudo se denomina *x* en las matemáticas (y también en la documentación de Java); este valor se incrementa de izquierda a derecha.
- La posición vertical, que a menudo se denomina *y*: este valor se incrementa hacia abajo. Tome en cuenta que esto es distinto a la convención en matemáticas.

Utilizamos este sistema cuando pedimos a Java que dibuje formas simples. La figura 3.2 muestra la metodología. La parte superior izquierda del área de dibujo es (0, 0) y el trazo de las formas comienza a partir de este punto.



**Figura 3.2** El sistema de coordenadas de píxeles.

## ● Explicación del programa

La única sección que nos concierne es la parte pequeña que hace el dibujo:

```

1 public void actionPerformed(ActionEvent event) {
2     Graphics papel = panel.getGraphics();
3     papel.drawLine(0, 0, 100, 100);
4 }

```

La línea 1 introduce la sección del programa que se ejecuta al hacer clic en el botón. Es un método. Cualquier instrucción que coloquemos entre el carácter `{` de la línea 1 y el carácter `}` de la línea 4 se ejecutará en secuencia, descendiendo por la página.

La línea 2 provee un área de gráficos para dibujar figuras; en este caso la llamamos `papel`. Si recuerda, en el capítulo 2 mencionamos que Java es orientado a objetos. Los objetos nos proporcionan herramientas. En el capítulo 2 vimos un ejemplo de un reproductor de CD, el cual provee un rango de herramientas como la siguiente:

```
cdCocina.seleccionar(4);
```

De hecho, nuestra área de dibujo no es sólo una hoja de papel en blanco; es más como un kit de dibujo que incluye el papel y un conjunto de herramientas, como una regla y un transportador.

En la línea 3 el papel utiliza su método `drawLine` para dibujar una línea sobre sí mismo. Los cuatro números entre paréntesis especifican la posición de la línea.

El método `drawLine` es uno de los diversos métodos que proporciona el sistema Java en una biblioteca. La línea 3 es una llamada (también conocida como invocación) al método, en donde le pide que realice la tarea de mostrar una línea en pantalla.

Al utilizar el método `drawLine` tenemos que suministrárle valores para los puntos inicial y final de la línea; además, necesitamos hacerlo en el orden correcto:

1. El valor horizontal (*x*) del inicio de la línea.
2. El valor vertical (*y*) del inicio de la línea.
3. El valor horizontal del final de la línea.
4. El valor vertical del final de la línea.

En Java estos elementos se denominan parámetros; son entradas para el método `drawLine`. Los parámetros deben ir encerrados entre paréntesis y separados por comas (tal vez se encuentre con el término *argumento*, que es otro nombre para un parámetro). Este método en especial requiere cuatro parámetros, los cuales deben ser números enteros. Si tratamos de usar el número incorrecto de parámetros, o que sean de tipo incorrecto, recibiremos un mensaje de error del compilador. Necesitamos asegurarnos de:

- Proveer el número correcto de parámetros.
- Proveer el tipo correcto de parámetros.
- Ordenarlos de la forma correcta.

Algunos métodos no requieren parámetros. En este caso también debemos utilizar los paréntesis, como en:

```
marco.crearGUI();
```

En nuestro ejemplo hay dos tipos de métodos en acción:

- Los que escribe el programador, como `actionPerformed`. El sistema Java invoca este método cuando el usuario hace clic en el botón.
- Los que se incluyen en las bibliotecas, como `drawLine`. Nuestro programa los invoca.

Una última observación: observe el punto y coma “;” al final de los parámetros de `drawLine`. En Java debe aparecer un punto y coma al final de cada “instrucción”. Pero, ¿qué es una instrucción? ¡La respuesta no es tan sencilla! Como puede ver en el programa anterior, no hay un punto y coma al final de cada línea. En vez de confundirlo con reglas formales complicadas aquí, le aconsejamos

que base sus primeros programas en nuestros ejemplos. Sin embargo, el uso de un método seguido de sus parámetros es de hecho una instrucción, por lo que se requiere un punto y coma.

## ● Métodos para dibujar

Además de líneas, la biblioteca de Java nos proporciona herramientas para dibujar:

- Rectángulos.
- Óvalos (y, por ende, círculos).

A continuación le mostraremos una lista de los parámetros para cada método, junto con un programa de ejemplo en el que se utilizan.

### **drawLine**

- El valor horizontal del inicio de la línea.
- El valor vertical del inicio de la línea.
- El valor horizontal del final de la línea.
- El valor vertical del final de la línea.

### **drawRect**

- El valor horizontal de la esquina superior izquierda.
- El valor vertical de la esquina superior izquierda.
- La anchura del rectángulo.
- La altura del rectángulo.

### **drawOval**

Imagine el óvalo ajustado dentro de un rectángulo. Los parámetros que requerimos son:

- El valor horizontal de la esquina superior izquierda del rectángulo.
- El valor vertical de la esquina superior izquierda del rectángulo.
- La anchura del rectángulo.
- La altura del rectángulo.

También se pueden dibujar las siguientes figuras, pero se requiere un conocimiento adicional sobre Java. Vamos a omitir los detalles sobre sus parámetros, ya que no las usaremos en nuestros programas.

- Arcos (sectores de un círculo).
- Rectángulos elevados (tridimensionales).
- Rectángulos con esquinas redondeadas.
- Polígonos.

Además, podemos dibujar figuras sólidas con `fillRect` y `fillOval`. Sus parámetros son idénticos a los de los métodos `drawRect` y `drawOval` equivalentes.

## Dibujos a color

Es posible establecer el color para dibujar. Hay 13 colores estándar:

black	blue	cyan	darkGray
gray	green	lightGray	magenta
orange	pink	red	white
yellow			

(el `cyan` es verde oscuro/azul y el `magenta` es rojo oscuro/azul).

Tenga cuidado con la ortografía; tome en cuenta el uso de mayúsculas a la mitad de algunos nombres. He aquí cómo podemos usar los colores:

```
papel.setColor(Color.red);
papel.drawLine(0, 0, 100, 50);
papel.setColor(Color.green);
papel.drawOval(100, 100, 50, 50);
```

El código anterior dibuja una línea roja, después un óvalo verde sin relleno. Si no establece el color, Java elige el negro.

## Creación de un nuevo programa

En el programa `EjemploDibujo` anterior nos concentraremos en su método `actionPerformed`, el cual contenía una invocación al método `drawLine`. Nuestro enfoque era aprender sobre la invocación y el paso de parámetros a los métodos de dibujo. Pero, ¿qué hay sobre las demás líneas de código? Están involucradas en tareas tales como:

- Crear el marco exterior (ventana) para el programa.
- Establecer el tamaño del marco.
- Agregar el área de dibujo y el botón a la interfaz de usuario.

Estas tareas se llevan a cabo mediante la invocación a métodos. En capítulos posteriores le explicaremos los detalles.

De hecho, la configuración de la interfaz de usuario es idéntica para cada programa de este capítulo. Todos los programas utilizan un área de dibujo y un solo botón para iniciar el dibujo. Sin embargo, no podemos usar el código idéntico en cada programa ya que el nombre de archivo a elegir debe coincidir con el nombre después de las palabras `public class` dentro del programa. Dé un vistazo al programa `EjemploDibujo`. Está almacenado en un archivo llamado `EjemploDibujo.java` y contiene la siguiente línea:

```
public class EjemploDibujo extends JFrame
```

El nombre de la clase debe empezar con letra mayúscula y sólo puede contener letras y dígitos. No puede contener signos de puntuación, como guiones, comas ni puntos; tampoco puede contener espacios. Al elegir el nombre de la clase se determina el nombre del archivo que debemos usar para guardar el programa.

Hay una línea adicional:

```
EjemploDibujo marco = new EjemploDibujo();
```

Esta línea se encuentra dentro del método `main` del programa. Al ejecutar un programa de Java, lo primero que ocurre es una invocación automática al método `main`. La primera tarea de `main` es crear una nueva instancia (un objeto) de la clase apropiada (en este caso, `EjemploDibujo`). En el capítulo 6 examinaremos el uso de `new` para crear nuevas instancias. Por ahora, tome en cuenta que el programa `EjemploDibujo` contiene tres ocurrencias del nombre `EjemploDibujo`:

- Una después de las palabras `public class`.
- Dos en el método `main`.

Al crear un nuevo programa tiene que cambiar estas ocurrencias.

He aquí un ejemplo. Vamos a crear un nuevo programa llamado `CirculoDibujo`. Los pasos son:

1. Abrir el archivo `EjemploDibujo.java` existente en cualquier editor, seleccionar todo el texto y copiarlo al portapapeles.
2. Abrir el software que utiliza para crear y ejecutar programas de Java (un IDE tal como Eclipse o un editor de texto).
3. Si utiliza un IDE, tendrá que crear un nuevo proyecto en este punto. El nombre del proyecto podría ser `ProyectoCirculoDibujo` (el nombre no necesita estar relacionado con `CirculoDibujo`, pero facilita la identificación de los proyectos).
4. Cree un nuevo archivo de Java en blanco llamado `CirculoDibujo.java`. Si utiliza un IDE, tal vez tenga que eliminar código que el IDE crea de manera automática.
5. Pegue el código que copió y cambie las tres ocurrencias de `EjemploDibujo` por `CirculoDibujo`.

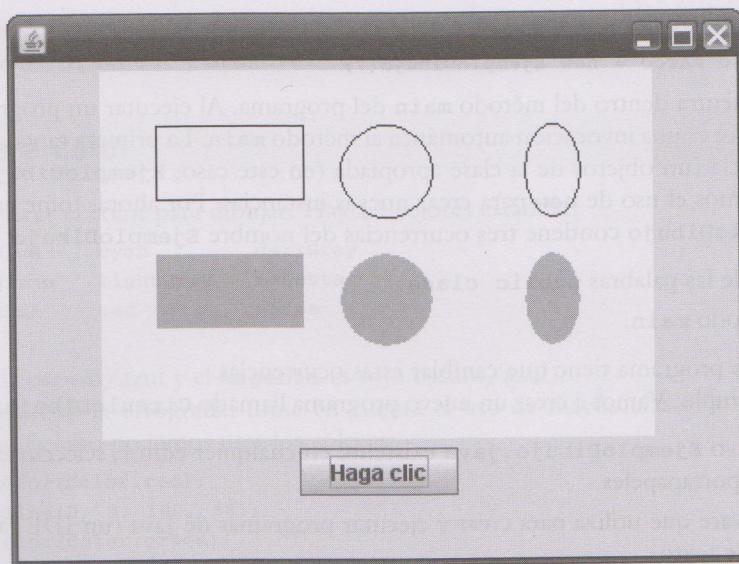
Ahora se puede concentrar en el tema principal de este capítulo y colocar las invocaciones apropiadas a los métodos de dibujo dentro del método `actionPerformed`.

## PRÁCTICA DE AUTOEVALUACIÓN

- 3.1** Cree un nuevo programa llamado `CirculoDibujo`. Al hacer clic en el botón, debe dibujar un círculo de 100 píxeles de diámetro.

### ● El concepto de secuencia

Cuando tenemos varias instrucciones en un programa, éstas se ejecutan de arriba abajo en secuencia (a menos que especifiquemos lo contrario mediante los conceptos de selección y repetición que veremos más adelante). El siguiente programa dibuja diversas figuras. La figura 3.3 muestra la salida resultante. En el listado omitimos el código que crea la interfaz de usuario, ya que esta parte es exactamente igual que en el programa anterior.



**Figura 3.3** Pantalla del programa **AlgunasFiguras**.

```

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

public class AlgunasFiguras extends JFrame
    implements ActionListener {
    // aquí se omitió el código de la GUI...
    public void actionPerformed(ActionEvent event) {
        Graphics papel = panel.getGraphics();
        papel.drawRect(30, 30, 80, 40);
        papel.drawOval(130, 30, 50, 50);
        papel.drawOval(230, 30, 30, 50);
        papel.setColor(Color.lightGray);
        papel.fillRect(30, 100, 80, 40);
        papel.fillOval(130, 100, 50, 50);
        papel.fillOval(230, 100, 30, 50);
    }
}

```

Las instrucciones se obedecen (ejecutan, llevan a cabo, ...) de arriba hacia abajo por la página, aunque esto es imposible de observar debido a la velocidad de la computadora. En capítulos posteriores veremos cómo repetir una secuencia de instrucciones una y otra vez.

## PRÁCTICA DE AUTOEVALUACIÓN

- 3.2 Escriba y ejecute un programa que dibuje una figura de ‘T’ grande en la pantalla.

### Cómo agregar significado mediante comentarios

¿Qué hacen las siguientes líneas de código?

```
papel.drawLine(20, 80, 70, 10);
papel.drawLine(70, 10, 120, 80);
papel.drawLine(20, 80, 120, 80);
```

El significado no es obvio de inmediato, y probablemente usted haya tratado de averiguarlo utilizando lápiz y papel. La respuesta es que dibuja un triángulo con una base horizontal, pero esto no se puede deducir con sólo ver las tres instrucciones. En Java podemos agregar comentarios (un tipo de anotación) a las instrucciones si les anteponemos los caracteres `//`. Por ejemplo, podríamos poner:

```
// dibuja un triángulo
papel.drawLine(20, 80, 70, 10);
papel.drawLine(70, 10, 120, 80);
papel.drawLine(20, 80, 120, 80);
```

Un comentario puede contener cualquier cosa —no hay reglas definidas—. Es responsabilidad de usted utilizarlos para expresar cierto significado.

También podemos colocar comentarios al final de una línea, como en el siguiente ejemplo:

```
// dibuja un triángulo
papel.drawLine(20, 80, 70, 10);
papel.drawLine(70, 10, 120, 80);
papel.drawLine(20, 80, 120, 80); // dibuja la base
```

No es conveniente abusar de los comentarios. No se recomienda comentar cada una de las líneas de un programa, ya que a menudo se corre el riesgo de duplicar información. El siguiente es un ejemplo de un mal comentario:

```
papel.drawRect(0, 0, 100, 100); // dibuja un rectángulo
```

Aquí la instrucción indica con claridad lo que hace, sin necesitar un comentario. Use los comentarios para declarar el tema general de una sección del programa, en vez de recalcar los detalles de cada instrucción. En el capítulo 19 aprenderá acerca de los estilos de comentarios adicionales que ayudan en la documentación de un programa.

## Fundamentos de programación

- Java cuenta con un vasto conjunto de métodos de biblioteca que podemos invocar.
- Los parámetros que pasamos a los métodos tienen el efecto de controlar las figuras que se dibujan.

## Errores comunes de programación

- Tenga cuidado con la puntuación. Las comas, los puntos y comas, los paréntesis y las llaves se deben escribir exactamente como se muestra en los ejemplos.
- El uso de mayúsculas debe ser exacto. Por ejemplo, `drawline` está mal, mientras que `drawLine` es correcto.

## Secretos de codificación

El orden y tipo de los parámetros debe ser correcto para cada método.

## Nuevos elementos del lenguaje

- ( ) para encerrar una lista de parámetros separados por comas.
- Si no se requieren parámetros, es necesario colocar los caracteres () después del nombre del método.
- // para indicar comentarios.

## Resumen

- Los programas pueden detectar eventos.
- Responder a un evento se conoce como “manejar” el evento.
- Las instrucciones se obedecen en secuencia, de arriba abajo (a menos que se solicite lo contrario).
- La biblioteca de Java tiene un conjunto de métodos “draw” que podemos invocar para mostrar gráficos.
- El posicionamiento de los gráficos se basa en coordenadas de píxeles.
- Se pueden pasar valores de parámetros a los métodos.

## Ejercicios

En los siguientes ejercicios le recomendamos que realice bosquejos y cálculos antes de escribir el programa. Seleccione un nombre adecuado para cada programa, que empiece con una letra mayúscula y no contenga espacios ni signos de puntuación. Base su programa en el código de `EjemploDibujo`, e inserte nuevas instrucciones dentro del método `actionPerformed`.

- 3.1 (a) Dibuje un cuadrado con tamaño de 100 píxeles, que comience a 10 píxeles de la esquina superior izquierda del área de dibujo. Use `drawRect`.  
(b) Realice la misma tarea, pero llame a `drawLine` cuatro veces.
- 3.2 Dibuje un triángulo con un lado vertical.
- 3.3 Dibuje un tablero para el juego del "gato" (o tres en línea).
- 3.4 Diseñe una casa sencilla y dibújela.
- 3.5 Dibuje una paleta de colores que conste de 13 pequeños cuadrados, cada uno de los cuales debe contener un color distinto.
- 3.6 He aquí las cifras de precipitaciones pluviales anuales para el país de Xanadú:
- |      |        |
|------|--------|
| 2004 | 150 cm |
| 2005 | 175 cm |
| 2006 | 120 cm |
- (a) Represente los datos mediante una serie de líneas horizontales.  
(b) En vez de líneas utilice rectángulos llenos de distintos colores.
- 3.7 Diseñe una diana con círculos concéntricos. Después agregue distintos colores.

## Respuestas a las prácticas de autoevaluación

- 3.1 Abra el programa `EjemploDibujo` existente con un editor. Guárdelo con el nombre `CirculoDibujo.java`. Cambie las tres ocurrencias de `EjemploDibujo` por `CirculoDibujo`; asegúrese de que el uso de mayúsculas sea el correcto. Por último, llame al método `drawOval` con los parámetros apropiados en el método `actionPerformed`. He aquí el programa completo:

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

public class CirculoDibujo extends JFrame
    implements ActionListener {

    private JButton button;
    private JPanel panel;

    public static void main(String[] args) {
        CirculoDibujo marco = new CirculoDibujo();
        marco.setSize(400, 300);
        marco.crearGUI();
        marco.setVisible(true);
    }
}
```



*Respuestas a las prácticas de autoevaluación (continúa)*

```
private void createGUI() {  
    setDefaultCloseOperation(EXIT_ON_CLOSE);  
    Container ventana = getContentPane();  
    ventana.setLayout(new FlowLayout());  
  
    panel = new JPanel();  
    panel.setPreferredSize(new Dimension(300, 200));  
    panel.setBackground(Color.white);  
    ventana.add(panel);  
  
    boton = new JButton("Haga clic");  
    ventana.add(boton);  
    boton.addActionListener(this);  
}  
  
public void actionPerformed(ActionEvent event) {  
    Graphics papel = panel.getGraphics();  
    papel.drawOval(0, 0, 100, 100);  
}  
}  
  
3.2  papel.drawLine(20, 20, 120, 20);  
      papel.drawLine(70, 20, 70, 120);
```