Clase adicional 6

Temas

- o Gestión de eventos en Swing
  - o Eventos Swing
  - o Oyente de eventos
  - o Registrar un gestor de eventos en un objeto
  - o Implementar la gestión de eventos
- o SwingApplication
  - o GUI
  - o Gestión de eventos
  - o Eventos múltiples
  - o Adaptador
- o Java en 2D y editor de dibujo
  - o Formas básicas
  - o Trazos y relleno
- o Problemas de la clase adicional
- o Problemas de diseño

Gestión de eventos en Swing

La gestión de eventos es de vital importancia en los programas que contienen una interfaz gráfica de usuario. Aunque puede intimidar a los programadores principiantes, la gestión de eventos es MUCHO más sencilla de lo que cree. A continuación encontrará un resumen de los tres componentes clave de un proceso de gestión de eventos:

- o Evento (hacer clic en un botón, pulsar una tecla, etc.)
- o Interfaz Listener (ActionListener, WindowListener, etc.)
- o Objeto (botón, marco, campo de texto, etc.) que "escucha" el evento

Evento Swing

Cada vez que el usuario escribe un carácter o hace clic en un botón, la máquina virtual de Java (JVM) genera un evento. A continuación se incluyen algunos ejemplos de eventos Swing:

Acción que desemboca en el evento	Tipo de oyente
El usuario pulsa un botón o Enter mientras escribe en un campo de texto	ActionListener
El usuario cierra un marco (ventana principal)	WindowListener
El usuario pulsa un botón del ratón	MouseListener
El usuario mueve el ratón sobre un componente	MouseMotionListener
El componente se hace visible	ComponentListener
El componente obtiene el foco del teclado	FocusListener
La selección de la tabla o la lista cambia	ListSelectionListener

Oyente de eventos

Cada evento puede desencadenar uno o más oyentes de dicho evento. Como ya dijimos la semana pasada, un oyente de eventos es una interfaz de Java que contiene una colección de declaraciones de métodos. Las clases que implementan la interfaz deben definir estos métodos. A continuación se incluye una lista de eventos, oyentes y métodos:

Evento	Interfaz oyente	Métodos de oyente
WindowEvent	WindowListener	windowActivated(WindowEvent e) windowDeactivated(WindowEvent e) windowClosed(WindowEvent e) windowClosing(WindowEvent e) windowOpened(WindowEvent e) windowDeiconified(WindowEvent e) windowIconified(WindowEvent e)
ActionEvent	ActionListener	actionPerformed(ActionEvent e)
ItemEvent	ItemListener	itemStateChanged(ItemEvent e)
TextEvent	TextListener	textValueChanged(TextEvent e)
FocusEvent	FocusListener	focusGained(FocusEvent e) focusLost(FocusEvent e)
KeyEvent	KeyListener	keyPressed(KeyEvent e) keyReleased(KeyEvent e) keyTyped(KeyEvent e)

Por ejemplo, cuando el usuario hace clic en un botón, el ActionListener será notificada y, entonces, desencadenará el método actionPerformed a ejecutar.

Registrar un gestor de eventos para el objeto

Para escuchar un evento, un objeto debe registrarse para ser el oyente de dicho evento. El siguiente código crea un JButton y registra el ActionListener correspondiente.

```
JButton button = new JButton("¡Soy un botón Swing!");
button.addActionListener(this);
```
3 pasos para implementar un oyente de eventos

Para implementar un gestor de eventos, debe seguir estos tres pasos:

1. Implementar una interfaz de escucha:

   ```
   public class MyClass implements ActionListener
   ```

2. Agregar el oyente a un objeto:

   ```
   button.addActionListener(this)
   ```

3. Definir los métodos de la interfaz de escucha:

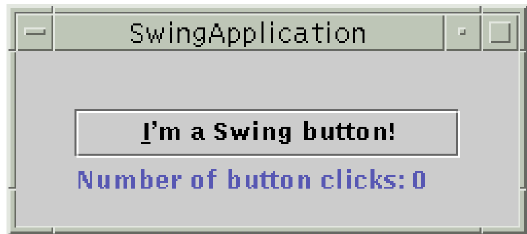
   ```
   public void actionPerformed(ActionEvent e){
   ```

```
...//código que reacciona ante la acción...
}
```

Ahora demostraremos el funcionamiento con el siguiente ejemplo SwingApplication.

SwingApplication

SwingApplication es un ejemplo extraído de la clase adicional de Java. Se trata de un contador de "clics en botones": cada vez que el usuario hace clic en el botón, la etiqueta se actualiza



La semana pasada hablamos de los componentes gráficos de esta aplicación. En esta sección analizaremos el modo en que interactúa con el usuario. Dicho de otro modo, cuando el usuario hace clic en un botón, ¿qué hará la aplicación?

GUI

Para refrescar la memoria, aquí tiene el fragmento GUI del código:

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

public class SwingApplication extends JFrame{   int numClicks = 0;
  JPanel pane = new JPanel();
  JLabel label = new JLabel("Número de clics de botones: " + numClicks);
  JButton button = new JButton("¡Soy un botón Swing!");

//Constructor

SwingApplicatin (){
    super ("SwingApplication");   //define el título del marco
    pane.setLayout(new GridLayout(0, 1)); pane.add(button);
    pane.add(label);
    this.getContentPane().add(pane, BorderLayout.CENTER);
  }
  public static void main(String[] args){
    SwingApplication app = new SwingApplication();
    app.pack();
    app.setVisible(true);
  }
```

}

Gestión de eventos

Sigamos los pasos descritos en la última sección

- o Evento: ActionEvent
- o Objeto: botón
- o Interfaz: ActionListener con el método actionPerformed

Éste es el código:

1. Implementar una interfaz de escucha:

   ```
   public class SwingApplication extends JFrame implements ActionListener
   ```

2. Agregar el oyente al botón (una vez creado)

   ```
   button.addActionListener(this);
   ```

3. Definir los métodos de la interfaz de escucha:

   ```
   Public void actionPerformed (ActionEvent e)
   {
       numClicks ++;
       label.setText ("Número de clics de botones: " + numClicks);
    }
   ```

Escucha de objetos múltiples

¿Qué ocurre si hay varios objetos escuchando al mismo evento? ¿Cómo decide Java qué método actionPerformed ejecutar? Por ejemplo, tenemos dos botones, b1 y b2. Los dos están registrados como un ActionListener. Si el usuario hace clic en b1, el mensaje "se ha hecho clic en b1" se imprimirá en la etiqueta. Si el usuario hace clic en b2, el mensaje "se ha hecho clic en b2" se imprimirá. Por tanto, necesitamos implementar el método actionPerformed del siguiente modo:

```
JButton b1 = new JButton ("B1");

b1.addActionListener (this);

JButton b2 = new JButton ("B2");

b2.addActionListener (this);

        ...........................    Public void actionPerformed (ActionEvent e)
```

```
{
if (e.getSource() == b1) //¿Qué objeto es el origen del evento?
  label.setText ("se imprime b1");
if (e.getSource() == b2)
  label.setText ("se imprime b2");
 }
```

Clases adaptadoras

Ahora volvamos a nuestra SwingApplication. Necesitamos implementar otro evento; queremos que la aplicación se cierre cuando el usuario cierre la ventana. De nuevo, seguiremos estos pasos:

- Evento: WindowEvent
- Objeto: marco
- Interfaz: WindowListener con el método windowClosing ...

Un momento; hay 7 métodos en la interfaz WindowListener. Como ya mencionamos anteriormente, la clase que implementa las interfaces debería definir TODOS los métodos de dicha interfaz. Esta regla es válida para interfaces como ActionListener, que sólo tienen un método. Pero WindowListener tiene 7 (sí, 7) métodos. ¿Tendremos que implementarlos todos, incluso si sólo necesitamos uno?

La respuesta es sí y no. Si implementa WindowListener, SÍ, deberá implementar TODOS los 7 métodos. Ya lo sé, escribir el código para seis métodos que no vamos a utilizar es el tipo de trabajo tedioso que a nadie le gusta hacer. Para simplificar esta tarea, Java ha desarrollado una clase adaptadora para cada interfaz de escucha que tenga más de un método. Una clase de este tipo implementa todos los métodos en la interfaz pero no hace nada con ellos. A continuación se indica cómo utilizar un adaptador.

```
 frame.addWindowListener(new WindowAdapter(){
   public void windowClosing(WindowEvent e)
   {
       System.exit(0);
   }
 }
);
```

Aquí, hemos definido una clase interna de tipo WindowAdapter como el adaptador de addWindowListener (). Para saber más acerca de las clases internas puede consultar el libro de texto (pág. 281). En este caso, simplemente necesita saber cómo se utiliza.

Java en 2D y editor de dibujo

La API de Java en 2D proporciona gráficos mejorados en dos dimensiones, texto y funcionalidad de imágenes para programas de Java. Admite representaciones con líneas, texto e imágenes en un entorno de trabajo integral y flexible para desarrollar interfaces de usuario más completas, programas de dibujo más sofisticados y editores de imágenes. Esta sección se basará en el ejemplo de clase con algunas funciones adicionales.

Formas básicas

	g2.draw(new Line2D.Double(x, y+rectHeight, x+rectWidth, y));
	g2.draw(new Rectangle2D.Double(x, y, rectWidth, rectHeight));
	g2.draw(new Ellipse2D.Double(x, y, rectWidth, rectHeight));
	g2.draw(new RoundRectangle2D.Double(x, y, rectWidth, rectHeight, 10, 10));
	g2.draw(new Arc2D.Double(x, y, rectWidth, rectHeight, 90, 135, Arc2D.OPEN));

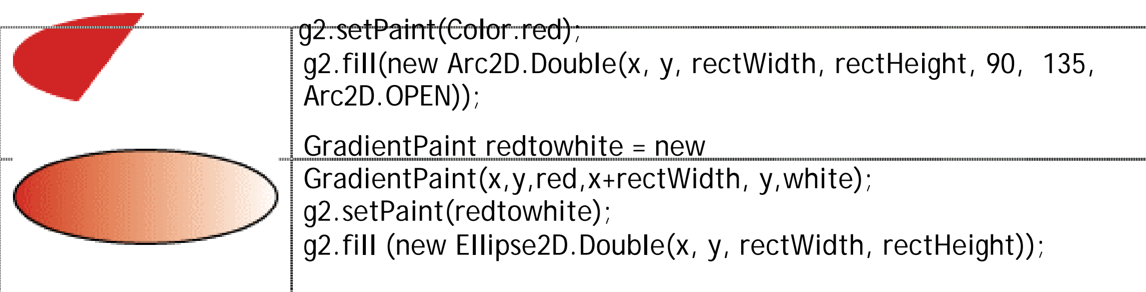
Trazos y gráficos de relleno

Podemos aplicar fácilmente estilos de línea y patrones de relleno a los gráficos cambiando los atributos de dibujo y de trazo en el contexto de gráficos en 2D.

El trazo permite especificar distintos grosores y patrones de puntos para líneas y curvas. Por ejemplo, el siguiente código crea un arco de línea gruesa (consulte la tabla arriba):

```
BasicStroke wideStroke = new BasicStroke(8.0f);
g2.setStroke(wideStroke);
g2.draw(new Arc2D.Double(x, y, rectWidth, rectHeight, 90, 135, Arc2D.OPEN));
```

Los patrones de relleno vienen definidos por la interfaz Paint. La API de Java en 2D proporciona tres implementaciones de Paint: Color, TexturePaint y GradientPaint. TexturePaint define un patrón de relleno con un fragmento simple de imagen que se repite uniformemente. GradientPaint define un patrón de relleno como gradiente entre dos colores. A continuación se incluyen dos ejemplos, uno con Color y otro con GradientPaint



	g2.setPaint(Color.red); g2.fill(new Arc2D.Double(x, y, rectWidth, rectHeight, 90, 135, Arc2D.OPEN));

	GradientPaint redtowhite = new GradientPaint(x,y,red,x+rectWidth, y,white);
	g2.setPaint(redtowhite);
	g2.fill (new Ellipse2D.Double(x, y, rectWidth, rectHeight));

Problemas de la clase adicional

Ejercicio 1: Conteste la siguientes preguntas

Pregunta 1: ¿Qué es un objeto de eventos?

a) Un objeto de eventos representa un evento concreto, por ejemplo, un clic de ratón.
b) Un objeto de eventos es un objeto que escucha clics de ratón.
c) Un objeto de eventos es una lista de todos los eventos que ocurren en el sistema.
d) Un objeto de eventos es lo que el programa hace como respuesta a un evento.

Pregunta 2: Si hay varios botones en un marco, ¿generará cada uno de ellos sus propios eventos cuando el usuario haga clic sobre él?

a) Sí; cada evento es único y está generado por un botón específico.
b) No; los eventos no se pueden distinguir unos de otros
c) Sí; cada botón puede generar un evento. Después, permanece inactivo.
d) No; sólo los oyentes de eventos son únicos.

Pregunta 3: ¿Puede un objeto de escucha único responder a varios tipos de eventos que provengan del mismo componente?

a) Sí; siempre hay un solo oyente en todo el programa, así que debe responder a todos los tipos de eventos.
b) No; debe existir un objeto de escucha específico para cada evento de cada componente en una GUI.
c) Sí; siempre y cuando se cuente con un método adecuado para cada tipo.
d) No; un programa sólo puede responder a un tipo de evento

Ejercicio 2:

La clase LabelCopy tiene 2 etiquetas, sourceLabel y targetLabel, y 1 botón. Éste es el código necesario para crear la GUI.

```
import javax.swing.*;

public class LabelCopy extends JPanel{
JLabel sourceLabel, targetLabel;
JButton copyButton;

public LabelCopy(){
sourceLabel = new JLabel("Etiqueta origen");
targetLabel = new JLabel("Etiqueta destino");
button = new JButton("Copiar");
add(sourceLabel);
add(button);
add(targetLabel);
}

public static void main (String[] args){
   JFrame frame = new JFrame (“Ejercicio 2”);
   LabelCopy pane = new LabelCopy();
   frame.getContentPane().add(pane, BorderLayout.CENTER);
   frame.pack();
   frame.setVisible(true);
  }
```

} Ahora, modifique la definición de la clase LabelCopy para que cuando el usuario haga clic en el botón, el texto de sourceLabel se copie en targetLabel

Problema de diseño

Desarrolle una calculadora sencilla de enteros. Tendrá dos campos de texto, una etiqueta que muestre el resultado y 4 botones de opción (+, -, *, /). Deberá funcionar del siguiente modo:

- Los números se introducen en los campos de texto, p.ej. 10 .. 10
- La operación se elige seleccionado uno de los botones de opción, p.ej. +
- La etiqueta mostrará el resultado basado en la operación seleccionada, p.ej. 20

Nota: lo que obtiene a partir del campo de texto es una cadena. Deberá convertirla a un entero. El código necesario es:

```
int firstInput = Integer.parseInt(string1);
```