

Clase 35

Aprendizaje activo: Uso de hilos para crear una animación

Hilos y Swing

Todos los programas de Java ejecutan, al menos, tres hilos:

1. el hilo `main()`; es decir, el hilo que comienza con el método `main`;
 2. el hilo de eventos, con el que el sistema de ventanas notifica sobre los eventos en los que se ha registrado y
 3. el hilo del recolector de basura.
- El hilo del recolector de basura se ejecuta en segundo plano (con prioridad menor) y uno puede olvidarse incluso de que está ahí.
 - Pero, tan pronto como coloquemos una interfaz gráfica de usuario, deberemos tener en cuenta el hilo de eventos.

JFileViewer

- Si el programa crea una GUI, pero luego únicamente reacciona ante los eventos de entrada del usuario, realmente intercambiará un hilo por otro.
- Pero si crea una GUI y la actualiza desde el hilo de main, deberá tener más cuidado.
- Pongamos un ejemplo sencillo. En la clase 30, donde hablábamos de los flujos, analizamos un ejemplo llamado JFileViewer que leía archivos de texto y los mostraba en una clase llamada JTextViewer.
- Tal como lo hicimos, se leía todo el archivo de texto y, después, el texto resultante pasaba a ser el contenido de JTextViewer.

3

JFileViewer (revisado)

- Un enfoque más interesante habría supuesto colocar la interfaz y comenzar a leer el archivo, mostrando el nuevo texto a medida que se leía en el disco.
- El problema con este enfoque era que implicaba la modificación (la llamada a los métodos) de objetos en la GUI desde un hilo distinto al de eventos.

4

Hilos y el AWT

- El paquete de GUI inicial de Java, el AWT, sincronizaba varios métodos en las clases de programación. Pero hacía que las clases del AWT estuviesen expuestas a interbloqueos.
- Cuando los programadores de Java se plantearon implementar capacidades mucho más complejas de Swing, de hecho, abandonaron.
- El AWT intenta ser multihilo, esto es, permitir la llamada a clases desde varios hilos.

5

Hilos y Swing

- Dejando a un lado poquísimas excepciones, las clases de Swing esperan que sus métodos se llamen únicamente desde el hilo de eventos. Tal como describen los desarrolladores de Java:

"Una vez que un componente de Swing se detecta, todo el código que afecte al estado de dicho componente o dependa de él debe ejecutarse en el hilo de entrega de eventos."

6

Hilos y Swing, 2

- Un componente se *detecta* cuando el sistema de ventanas lo asocia a una ventana que realmente lo muestra en pantalla.
- Esto suele ocurrir cuando el componente se hace visible por primera vez o cuando se le otorga un tamaño preciso por primera vez (mediante la llamada a `pack()`, por ejemplo).
- Hasta ese momento, se puede modificar desde cualquier otro hilo como el hilo principal, ya que no hay posibilidad de que se pueda acceder a él desde el hilo de eventos hasta que el sistema de ventanas no lo detecte.
- Así, puede agregar componentes (`add()`) al contenedor desde el hilo principal o agregar texto a un `JTextArea`, siempre y cuando no sea detectado.

7

Hilos y Swing, 3

- Ahora bien, una vez que se hace visible, puede aceptar clics del ratón o pulsaciones de teclas, o cualquier otro tipo de evento y pueden utilizarse los métodos de llamada correspondientes.
- Swing NO sincroniza estos métodos ni los métodos a los que pueden llamar, como `setText()` o `add()`.
- Si quiere llamar a `setText()` o a métodos similares desde cualquier otro hilo que no sea el de eventos, deberá utilizar una técnica especial.

8

Modificar una GUI desde otro hilo

- Básicamente, se crea un objeto que describa la tarea que se debe realizar en el hilo de eventos a una hora determinada.
- A continuación, se pasa dicha tarea al hilo de eventos mediante un método sincronizador que la pone en cola con el resto de eventos en la cola de eventos del hilo de eventos.
- Swing ejecutará la tarea cuando quiera hacerlo, ya que Swing sólo procesa un evento cada vez, incluidas estas tareas especiales que pueden llamar a métodos no sincronizados de las clases de la GUI.

9

Uso de `invokeLater()`

- ¿Cómo creamos esta tarea?

```
Runnable update = new Runnable() {
    public void run() {
        component.doSomething();
    }
};
SwingUtilities.invokeLater( update );
```
- `invokeLater()` es un método estático sincronizado de la clase `SwingUtilities` en el paquete `javax.swing`. Inserta la tarea en la cola de eventos.

10

Métodos Swing sincronizados

Como ya hemos mencionado, y como tal vez ya haya deducido del ejemplo del cronómetro, existen algunos métodos Swing que pueden llamarse desde otro hilo con total seguridad. Éstos incluyen:

- `public void repaint()`
- `public void revalidate()`
- `public void addEventTypeListener(Listener l)`
- `public void removeEventTypeListener(Listener l)`

11

JBetterFileViewer

```
public void load( String path )
    throws IOException {
    FileReader in = new FileReader( path );
    int nread;
    char [] buf = new char[ 512 ];

    while( ( nread = in.read( buf ) ) >= 0 ) {
        Update update = new Update( buf, nread );
        SwingUtilities.invokeLater( update );
    }

    in.close();
}
```

12

JBetterFileViewer, 2

```
private class Update
implements Runnable {
    private final String theString;

    public Update( char [] b, int n ) {
        theString = new String( b, 0, n );
    }

    public void run() {
        theViewer.append( theString );
    }
}
```

13

Aplicación Ticker

- **Nuestro objetivo es utilizar un hilo independiente para crear una animación que desplace una cadena por la pantalla.**
- **En la versión completa, puede detener la animación pulsando con el ratón en la pantalla y reanudarla soltando el ratón.**
- **En este punto inicial, le ofrecemos Ticker0.java. Descárguelo del sitio web de clase y lo analizaremos juntos. No está animado. Los clics del ratón hacen que la cadena se desplace por la pantalla.**

14

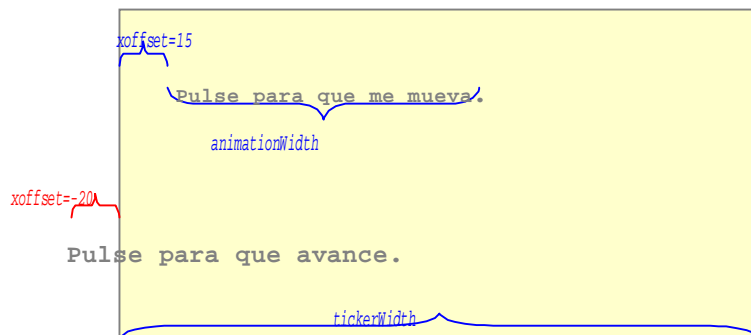
Ticker0, miembros de datos

```
public class Ticker0 extends JPanel
{
    private String animationString;
    private Font animationFont;
    private int animationWidth;
    private int xoffset;
    private int tickerWidth;

    public static final Dimension DEFAULT_SIZE =
        new Dimension( 400, 200 );
}
```

15

Ticker0, Geometría



16

Ticker0, Constructor

```
public Ticker0(String s) {
    animationString = s;
    setOpaque( true );
    setBackground( Color.black );
    setForeground( Color.white );
    animationFont= new Font("SansSerif",Font.BOLD,16);
    animationWidth = getFontMetrics(animationFont).
        stringWidth( animationString );
    setFont( animationFont );
    xoffset = 15;
    tickerWidth = DEFAULT_SIZE.width;
}
```

17

Ticker0, MouseListener

```
addMouseListener( new MouseAdapter() {
    public void mouseClicked( MouseEvent e )
    {
        // move left to x position with mouse cursor
        xoffset -= 5;
        // si se clickeó en el borde izquierdo de la right
        // devuelve a la derecha
        if (xoffset <= -animationWidth)
            xoffset = tickerWidth;
    } repaint();
} );

} // end constructor
```

18

Ticker0, paintComponent()

```
public void paintComponent(Graphics g)
{
    super.paintComponent( g );
    int yoffset = getHeight() / 2;
    tickerWidth = getWidth(); // permite cambiar tamaño
    g.drawString( animationString, xoffset, yoffset );
}
```

19

Ticker, Paso 1

- **Copie Ticker0 y renómbrelo como Ticker1.**
- **Escriba la frase que quiera como argumento constructor de Ticker1 en el método main().**
- **Comente el código de MouseListener hasta el momento.**
- **Ahora modifique Ticker1 para que se desplace continuamente a una velocidad de un píxel por centésima de segundo.**
- **Tal vez le resulte útil el modelo del programa SimpleClock.**

20

SimpleClock, Constructor

```
public class SimpleClock
  extends javax.swing.JPanel implements Runnable {
    private Font clockFont =
        new Font("SansSerif",Font.BOLD,24);
    private Thread clockThread = null;
    private int xPos = 40;
    private int yPos = 50;

    public SimpleClock() {
        . . .
        clockThread = new Thread( this, "Reloj" );
        clockThread.start();
    }
}
```

21

SimpleClock, run()

```
public void run()
{
    while ( true )
    {
        repaint();
        try
        {
            Thread.sleep( 1000 );
        }
        catch (InterruptedException e )
        {}
    }
}
```

22

Ticker, Paso 2

- **Ahora, como segundo paso, copie Ticker1 y renómbrelo como Ticker2. Modifique Ticker2 para que la animación se detenga cuando pulse el ratón y se reanude cuando lo suelte.**
- **¿Qué eventos debe escuchar?**
- **¿Cómo conseguirá detener e iniciar la animación?**