Introducción a Hilos, Concurrencia y Paralelismo en Java

Cosijopii García

December 5, 2024

¿Qué son los Hilos?

- Un hilo es la unidad más pequeña de procesamiento que puede realizar un programa.
- Permiten realizar varias tareas simultáneamente dentro de un programa.
- En Java, los hilos son representados por la clase `Thread` o implementando la interfaz `Runnable`.

Concurrencia vs Paralelismo

- **Concurrencia**: Capacidad de un programa para manejar múltiples tareas a la vez, incluso si no se ejecutan simultáneamente.
- **Paralelismo**: Ejecución simultánea de múltiples tareas en varios procesadores o núcleos.

Ejemplo

- Descargar varios archivos (**Concurrencia**).
- Procesar cada archivo en un núcleo diferente (**Paralelismo**).

Crear Hilos en Java - Extender Thread

```
1  class MiHilo extends Thread {
2      @Override
3      public void run() {
4          System.out.println("Hilo␣ejecut ndose:␣" +
               Thread.currentThread().getName());
5      }
6  }
7
8  public class Main {
9      public static void main(String[] args) {
10         MiHilo hilo = new MiHilo();
11         hilo.start();
12     }
13 }
```

Crear Hilos en Java - Implementar Runnable

```
1  class MiRunnable implements Runnable {
2      @Override
3      public void run() {
4          System.out.println("Hilo ejecut ndose: " +
               Thread.currentThread().getName());
5      }
6  }
7
8  public class Main {
9      public static void main(String[] args) {
10         Thread hilo = new Thread(new MiRunnable());
11         hilo.start();
12     }
13 }
```

Métodos de la Clase `Thread`

- `start()`: Inicia la ejecución de un hilo.
- `run()`: Contiene el código que se ejecutará en el hilo.
- `sleep(milliseconds)`: Suspende el hilo por un período específico.
- `join()`: Espera a que un hilo termine su ejecución.
- `isAlive()`: Comprueba si el hilo está activo.

Sincronización de Hilos

- Cuando varios hilos acceden a un recurso compartido, puede haber problemas de consistencia.
- La sincronización garantiza el acceso exclusivo a los recursos compartidos.

```
1  class Contador {
2      private int cuenta = 0;
3
4      public synchronized void incrementar() {
5          cuenta++;
6      }
7
8      public int obtenerCuenta() {
9          return cuenta;
10     }
11 }
```

Ejemplo Completo de Sincronización I

```
1  public class Main {
2      public static void main(String[] args) {
3          Contador contador = new Contador();
4
5          Thread hilo1 = new Thread(() -> {
6              for (int i = 0; i < 1000; i++) contador.
                   incrementar();
7          });
8
9          Thread hilo2 = new Thread(() -> {
10             for (int i = 0; i < 1000; i++) contador.
                   incrementar();
11         });
12
13         hilo1.start();
14         hilo2.start();
15
16         try {
17             hilo1.join();
18             hilo2.join();
```

Ejemplo Completo de Sincronización II

```
19              } catch (InterruptedException e) { }
20
21              System.out.println("Cuenta final: " + contador
                    .obtenerCuenta());
22          }
23      }
```

Ejercicio 1: Hilo Simple

Enunciado

Crea un programa que:

- Inicie tres hilos diferentes.
- Cada hilo debe imprimir su nombre 5 veces.
- Usa `Thread.sleep()` para pausar cada hilo entre impresiones.

Ejercicio 2: Sincronización

Enunciado

Escribe un programa donde:

- Dos hilos incrementen un contador compartido.
- Implementa la sincronización para evitar inconsistencias.
- Imprime el valor final del contador.

Usando ExecutorService

```
1  import java.util.concurrent.ExecutorService;
2  import java.util.concurrent.Executors;
3
4  public class Main {
5      public static void main(String[] args) {
6          ExecutorService executor = Executors.
               newFixedThreadPool(2);
7
8          Runnable tarea1 = () -> System.out.println("
               Tarea␣1␣ejecutada");
9          Runnable tarea2 = () -> System.out.println("
               Tarea␣2␣ejecutada");
10
11         executor.execute(tarea1);
12         executor.execute(tarea2);
13
14         executor.shutdown();
15     }
16 }
```

Consideraciones Finales

- Usa hilos cuando necesites realizar tareas concurrentes o paralelas.
- Siempre sincroniza recursos compartidos para evitar errores.
- Usa `ExecutorService` para manejar hilos de forma más sencilla.

Ejemplo Complejo: Reproducción de Música I

- Usar hilos para manejar la reproducción de música sin bloquear la interfaz principal.
- Ejemplo básico usando la clase `Clip` de Java para reproducir un archivo de audio.

```
1  import javax.sound.sampled.*;
2
3  public class Reproductor implements Runnable {
4      private String rutaArchivo;
5
6      public Reproductor(String rutaArchivo) {
7          this.rutaArchivo = rutaArchivo;
8      }
9
10     @Override
11     public void run() {
12         try (AudioInputStream audio = AudioSystem.
               getAudioInputStream(
13                 new File(rutaArchivo))) {
```

Ejemplo Complejo: Reproducción de Música II

```
14                  Clip clip = AudioSystem.getClip();
15                  clip.open(audio);
16                  clip.start();
17                  clip.loop(Clip.LOOP_CONTINUOUSLY); //
                        Reproducir en bucle
18                  Thread.sleep(10000); // Reproducir por 10
                        segundos
19                  clip.stop();
20              } catch (Exception e) {
21                  e.printStackTrace();
22              }
23      }
24
25      public static void main(String[] args) {
26          Thread hiloMusica = new Thread(new Reproductor
                ("cancion.wav"));
27          hiloMusica.start();
28      }
29  }
```

Explicación: Reproducción de Música

- La clase `AudioSystem` de Java proporciona herramientas para manejar audio.
- `Clip`: Permite cargar y reproducir un archivo de audio.
- El hilo asegura que la reproducción de música no bloquee otras operaciones.
- Consideraciones:
  - El archivo de audio debe estar en formato `.wav`.
  - Manejar excepciones para evitar fallos si el archivo no existe.

Ejemplo Complejo: Animación en un Panel I

- Usar hilos para crear animaciones fluidas.
- Ejemplo básico de una animación que mueve un círculo en un panel.

```
1  import javax.swing.*;
2  import java.awt.*;
3
4  public class Animacion extends JPanel implements
       Runnable {
5      private int x = 0;
6
7      @Override
8      protected void paintComponent(Graphics g) {
9          super.paintComponent(g);
10         g.setColor(Color.BLUE);
11         g.fillOval(x, 50, 50, 50); // Dibuja un
               c rculo
12     }
13
14     @Override
```

Ejemplo Complejo: Animación en un Panel II

```
15      public void run() {
16          while (true) {
17              x += 5;
18              if (x > getWidth()) x = 0;
19              repaint();
20              try {
21                  Thread.sleep(50); // Controla la
                        velocidad de la animaci n
22              } catch (InterruptedException e) {
23                  e.printStackTrace();
24              }
25          }
26      }
27
28      public static void main(String[] args) {
29          JFrame ventana = new JFrame("Animaci n");
30          Animacion panel = new Animacion();
31          ventana.add(panel);
32          ventana.setSize(400, 200);
```

Ejemplo Complejo: Animación en un Panel III

```
33            ventana.setDefaultCloseOperation(JFrame.
                  EXIT_ON_CLOSE);
34            ventana.setVisible(true);
35
36            Thread hiloAnimacion = new Thread(panel);
37            hiloAnimacion.start();
38        }
39    }
```

Explicación: Animación en un Panel

- La clase `JPanel` es utilizada para dibujar elementos gráficos.
- El método `paintComponent()` actualiza el dibujo en el panel.
- El hilo ejecuta un bucle infinito que:
  - Actualiza la posición del círculo.
  - Llama a `repaint()` para redibujar el panel.
- La llamada a `Thread.sleep()` controla la velocidad de la animación.
- Consideraciones:
  - Asegúrate de manejar adecuadamente el ciclo infinito.
  - Manejar excepciones para evitar interrupciones.