Clase 34

Hilos

¿Qué es un hilo?

- Imagine un programa de Java que lee archivos grandes en Internet en distintos servidores. Algunos de ellos estarán sometidos a grandes cargas o tendrán conexiones lentas a Internet. Otros tal vez devuelvan datos con rapidez.
- La mayor parte del tiempo, nuestro programa estará esperando datos de la red. Un enfoque de programación parece bastante claro:

  Leer archivo 1 del servidor A
  Leer archivo 2 del servidor B
  Leer archivo 3 del servidor C
  . . . .

¿Qué es un hilo?, 2

- Realizar estas lecturas de forma secuencial no resulta eficaz, ya que la carga del archivo 2 desde el servidor B no comenzará hasta que se haya cargado el archivo 1 por completo.
- Un enfoque mucho más rápido sería leer desde cada archivo al mismo tiempo y gestionar los archivos parciales a medida que lleguen.
- Esto requiere la capacidad de tener varias tareas procesándose en paralelo (como si cada una estuviese asignada a un procesador independiente).

¿Qué es un hilo?, 3

- La mayoría de los ordenadores sólo tienen un procesador, así que lo que realmente necesitamos es que el programa pueda conmutar a medida que lleguen los orígenes de datos.
- De forma más general, deberemos poder escribir programas en los que el "flujo de control" se ramifique y donde estas ramificaciones se procesen en paralelo.
- El procesador puede conseguir esto conmutando entre las distintas ramas del programa en pequeños incrementos de tiempo.
- Ésta es la estrategia de los *hilos*.

Hilos y procesos

- **La mayoría de sistemas operativos permiten la ejecución de varios *procesos* en paralelo.**
- **Los *procesos* resultan costosos, pero son seguros. Están tan bien aislados los unos de los otros que a veces resulta complicado y costoso incluso que se comuniquen entre sí.**
- **Los *hilos* son más asequibles, pero no están bien aislados entre sí cuando ejecutan el mismo proceso.**

5

Compatibilidad de Java para hilos

- **Java es el único lenguaje de uso general en el que la compatibilidad con los hilos forma parte del lenguaje.**
- **Ada, un lenguaje desarrollado por el Ministerio de Defensa, también integra compatibilidad con hilos, pero se trata de un lenguaje escasamente utilizado fuera del ámbito de defensa.**
- **En otros lenguajes como C y C++, hay bibliotecas para implementar hilos que están más o menos normalizadas.**

6

Java es, inherentemente, un lenguaje multihilo

- En Java, la colección de basura de objetos no referenciados se realiza mediante el sistema de tiempo de ejecución de Java en un hilo independiente.
- Java también utiliza un hilo independiente para entregar eventos de interfaz del usuario. Con ello se consigue que un programa siga respondiendo, incluso si está inmerso en un cálculo de ejecución largo o en una operación E/S.
- Piense en cómo podría implementar una función "Cancel" si no pudiese utilizar hilos.
- Esto significa que Java es inherentemente un lenguaje multihilo. El entorno del tiempo de ejecución de Java utiliza hilos simples incluso si el programa del usuario no lo hace.
- Pero los programadores también pueden utilizar hilos en su propio código. Nuestra estrategia de descarga de varios archivos requiere el uso de hilos.

7

Simplifiquemos

- La clase `Thread` proporciona la compatibilidad de Java para los hilos.
- Menos es siempre más cuando hablamos de hilos.
- Siempre debe simplificar el uso de los hilos tanto como sea posible (simplificar, no hacerlo más fácil).

8

¿Cómo decirle a un hilo lo que debe hacer?

Existen dos enfoques:

1. **Puede crear una subclase de la clase Thread e ignorar el método `public void run()`.**

```
public class MyThread extends Thread  {
  public void run() {
  // el código ejecutado en el hilo va aquí
  }
}
```

Puede crear una instancia de este hilo de la siguiente forma:

```
Thread t = new MyThread();
```

9

¿Cómo decirle a un hilo lo que debe hacer?, 2

2. **Puede escribir una clase por separado que implemente la interfaz `Runnable`, que contiene un método único:**

```
public interface Runnable {
  public void run();
}
```

Se crea el hilo con `Runnable` como argumento del constructor.

Una razón para utilizar este enfoque es que las clases de Java solamente pueden heredar de una clase única. Si desea definir el método `run()` de un hilo en una clase que ya herede de otra, no puede recurrir a la primera estrategia.

10

Ejemplo de `Runnable`

Por ejemplo, considere la clase `FrameInThread` definida como

```
public class FrameInThread
  extends Frame implements Runnable {
    // los constructores y otros métodos van aquí
    public void run() {
      // el código ejecutado en el hilo va aquí
    }
}
```

Si quisiéramos que una instancia de la clase `FrameInThread` se ejecutase en su propio hilo, podríamos utilizar la sentencia

```
Thread t = new Thread(new FrameInThread() );
```


Iniciar y detener hilos

- ***¿Cómo se inicia un hilo?:*** **Llame a `start()` en la instancia del hilo.**

```
Thread t = new MyThread();
t.start();
```

- ***¿Cómo se detiene un hilo y se destruye?:*** **Deje que termine el método `run()` y que la referencia del hilo finalice o establezca la referencia en *null*. El recolector de basura reclamará el almacenamiento del hilo.**
- `t.stop()` **se despreciará.**

Cómo saber si un hilo se sigue ejecutando

- **Puede preguntarlo:**

```
Thread t = new MyThread();
t.start();
. . .
if ( t.isAlive() ) // se sigue ejecutando
else               // no se ejecuta
```

- **O puede esperarlo:**

```
t.join();  // bloquea hasta que t finaliza
```

13

Ejemplo de hilo simple

- **En este ejemplo implementaremos la estrategia de descarga multihilo descrita anteriormente.**
- **El programa utiliza un `Thread` independiente para leer cada URL de un servidor web en Internet y copia el contenido de dicha URL a un archivo local.**
- **Llamamos a la clase que realiza la tarea y ampliamos la clase `URLCopyThread` del hilo.**
- **`URLCopyThreadMain` crea una nueva instancia de `URLCopyThread` para cada operación de copiado.**

14

URLCopyThreadMain

```
public class URLCopyThreadMain {
  public static void main(String argv[]) {
    String[][] fileList = {
{"http://web.mit.edu/1.00/www/Lectures/Lecture28/
  Lecture28.pdf", "Lecture28.pdf"},
{"http://microscopy.fsu.edu/micro/gallery/dinosau
  r/dino1.jpg" , "dino1.jpg"},
{"http://www.boston.com/","globe.html"},
{"http://java.sun.com/docs/books/tutorial/index.h
  tml", "tutorial.index.html"},
    };
```

15

URLCopyThreadMain, 2

```
    for (int i=0; i<fileList.length; i++) {
      Thread th;
      String threadName = new String( "T" + i );
      th = new URLCopyThread( threadName,
                              fileList[i][0],
                              fileList[i][1] );
      th.start();
      System.out.println("Hilo " + th.getName() +
       " para copiar desde " + fileList[i][0] + " en " +
       fileList[i][1] + " iniciado" );
    }
  }
}
```

16

URLCopyThread

```
import java.io.*;
import java.net.*;

public class URLCopyThread extends Thread {
  private URL fromURL;
  private BufferedInputStream input;
  private BufferedOutputStream output;
  private String from, to;
```

17

URLCopyThread, 2

```
public URLCopyThread(String n, String f,
                     String t) {
  super( n );
  from = f; to = t;
  try {
    fromURL = new URL(from);
    input = new BufferedInputStream(
                     fromURL.openStream());
    output = new BufferedOutputStream(
                new FileOutputStream(to));
  }
```

18

URLCopyThread, 3

```
catch(MalformedURLException m) {
      System.err.println(
      "MalformedURLException creando URL "
         + from);
}
catch(IOException io) {
     System.err.println("IOException " +
       io.toString() );
}
}
```

19

URLCopyThread, 4

```
public void run() {
  byte [] buf = new byte[ 512 ];
  int nread;
  try {
    while((nread=input.read(buf,0,512)) > 0) {
      output.write(buf, 0, nread);
      System.out.println( getName() + ": " +
        nread + " bytes" );
  }
```

20

URLCopyThread, 5

```
      input.close();
      output.close();
      System.out.println("Hilo " + getName() +
          " copiando " + from + " en " + to +
          "finished");
      }
      catch(IOException ioe) {
        System.out.println("IOException:" +
          ioe.toString());
      }
    } // fin del método run()
  } // fin de la clase URLCopyThread
```

21

Sincronización de hilos

Cuando los programas utilizan hilos, a menudo se deben solucionar los conflictos y las incoherencias que éstos provocan.

Los dos problemas más significativos son la *sincronización* y el *interbloqueo*.

22

Sincronización: el problema

- **En muchos casos, un segmento de código se puede ejecutar como "todo o nada" antes de poder ejecutar otro hilo.**
- **Por ejemplo, suponga que está insertando un nuevo objeto en un `Vector` y que el nuevo elemento supera la capacidad actual. El método `addElement()` del vector deberá copiar el contenido de `Vector` en una nueva ubicación de la memoria con mayor capacidad y, entonces, agregar el nuevo elemento.**
- **Si esta operación la ejecuta un hilo y ha finalizado parcialmente cuando otro hilo toma el control e intenta obtener un elemento del mismo `Vector`, aparece el problema: el primer hilo interrumpido deberá abandonar el vector parcialmente copiado en un estado incoherente.**

23

Métodos `synchronized`

- **Java permite declarar un método como `synchronized` para evitar este tipo de problemas.**
- **Una definición de método como ésta**

```
public synchronized void foo() {
  // cuerpo del método
}
```

  significa que `foo()` no puede ser interrumpido por otro método `synchronized` que actúe sobre el mismo objeto.
- **Si otro hilo intentase ejecutar otro método `synchronized` en el mismo objeto, este hilo debería esperar a que el primer método `synchronized` finalizase.**

24

Precauciones con el método `synchronized`

- Tenga en cuenta que los métodos `synchronized` solamente esperan a otros método `synchronized`.
- Los métodos comunes no sincronizados invocados en el mismo objeto sí se ejecutarán.
- Y cualquier otro hilo puede ejecutar otro método `synchronized` en otra instancia de la misma clase.

25

Funcionamiento de la sincronización

- Java implementa métodos `synchronized` a través de un bloque especial llamado monitor que forma parte de cada instancia de cada clase que hereda de `Object`.
- Cuando un hilo necesita entrar en un método `synchronized`, intenta adquirir el bloqueo en el objeto actual.
- Si ningún otro método `synchronized` llamado en este objeto se encuentra activo en el algún hilo, el bloqueo está libre y el hilo puede continuar. Pero si otro hilo está ejecutando un método `synchronized` en el objeto, el bloque no estará libre y el primer método deberá esperar.
- Si un método estático está sincronizado, el bloque forma parte del objeto que representa la clase (una instancia de la clase `Class`).

26

Sincronización en el JDK

- **El truco reside en saber si un método necesita ser sincronizado. Muchos métodos de las clases predefinidas de Java ya están sincronizados.**
- **Por ejemplo, la mayoría de los métodos de la clase `Vector` están sincronizados por el motivo descrito anteriormente.**
- **Otro ejemplo: el método de la clase `Component` de Java AWT que agrega un objeto `MouseListener` a un `Component` (para que `MouseEvents` se registren en el `MouseListener`) también está sincronizado. Si comprueba el código fuente de AWT y Swing, encontrará que la firma de este método es**

```
public synchronized void
   addMouseListener(MouseListener l)
```

27

Valores predeterminados de sincronización en Java

- **De forma predeterminada (es decir, a no ser que se declare lo contrario), los métodos NO están sincronizados.**
- **Declarar un método como sincronizado ralentizará la ejecución del programa, ya que la adquisición y liberación de bloqueos genera una sobrecarga.**
- **También introduce la posibilidad de un nuevo tipo de fallo llamado interbloqueo.**
- **Sin embargo, en muchos casos resulta esencial sincronizar los métodos para que el programa se ejecute correctamente.**

28

Interbloqueo

- Cuando dos hilos distintos requieren acceso exclusivo a los mismos recursos, pueden darse situaciones en las que uno de ellos obtenga acceso al recurso que el otro hilo necesita. En ese caso ninguno de los dos podrá continuar.
- Por ejemplo, suponga que cada hilo necesita privilegios exclusivos de escritura en dos archivos distintos. El hilo 1 podría abrir el archivo A de forma exclusiva y el hilo 2 hacer lo mismo con el archivo B.
- Ahora el hilo 1 necesita acceso exclusivo al archivo B y el hilo 2 necesita acceso exclusivo al archivo A. Ambos hilos se obstaculizan entre sí. El origen más común de este problema tiene lugar cuando dos hilos intentan ejecutar métodos `synchronized` en el mismo conjunto de objetos.

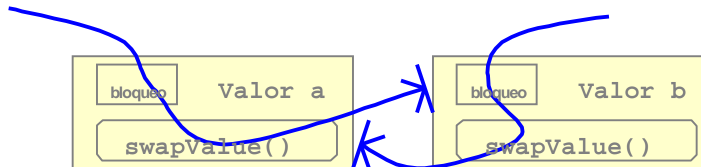
29

Ejemplo de interbloqueo

```
public class Value
{
  private long value;
  public Value( long v ) { value=v; }
  synchronized long getValue() { return value; }
  synchronized void setValue( long v) { value=v; }
  synchronized void swapValue( Value other ) {
    long t = getValue();
    long v = other.getValue();
    setValue( v );
    other.setValue(t);
  }
}
```

30

Diagrama de interbloqueo



Extraído de Doug Lea, *Concurrent Programming in Java* (2000). Excelente referencia para usuarios avanzados

31

Síntomas del interbloqueo

- **Los síntomas del interbloqueo son simplemente el bloqueo del programa (deja de ejecutarse) o la interrupción infinita de un programa regido por un hilo específico.**
- **Los problemas de sincronización y de interbloqueo son realmente complicados de depurar, ya que un programa con este tipo de errores puede ejecutarse a la perfección muchas veces hasta que, un día, falla.**
- **Esto ocurre porque el orden y la secuencia de ejecución de los distintos hilos no es completamente predecible.**
- **Los programas deben funcionar correctamente sin tener en cuenta el orden o la secuencia en que se ejecutan los distintos hilos.**
- **En el momento en que se sincroniza para prevenir posibles interferencias perjudiciales entre hilos, el riesgo del interbloqueo aparece.**

32

Reglas de sentido común para hilos

Éstas son algunas reglas que pueden ser útiles:

1. **Utilice sólo varios hilos cuando sea necesario o cuando las ventajas sean evidentes.**
2. **Siempre que utilice varios hilos, piense detenidamente si los métodos que ha escrito deben sincronizarse.**
3. **Si tiene dudas, declárelos como `synchronized`.**
4. **Si las distintas ejecuciones del mismo programa con más de un hilo se realizan de forma muy distinta, incluso con las mismos datos de entrada, puede sospechar de la presencia de un problema de sincronización.**
5. **Si utiliza varios hilos, intente asegurarse de que se destruyen cuando ya no son necesarios.**

33

Ejemplo de cronómetro

- **Como ejemplo más complicado para ilustrar una interacción elegante con Swing consideraremos la clase `Clock` que implementa un cronómetro.**
- `Clock` **implementa** `Runnable` **y, por tanto, puede utilizarse para crear su propio hilo.**
- `Time` **es una clase interna que proporciona la pantalla del tiempo del cronómetro.**

34

Clock, main()

```
public class Clock
  extends JFrame implements Runnable {
    private Thread clockThread = null;
    private Time time;
    private long accumTime = 0L;
    private long startTime = -1L;

    public static void main( String[] args ) {
        Clock clock = new Clock();
        clock.show();
    }
```

35

Clock, constructor

```
public Clock() {
  super( "Reloj" ); setDefaultCloseOperation(
  EXIT_ON_CLOSE ); JPanel buttons = new
  JPanel();
  JButton bStart = new JButton( "iniciar" );
  bStart.addActionListener( new ActionListener()
    { public void actionPerformed(ActionEvent e)
      { start(); }
    } );
  // crea los botones bStop y bReset de la misma forma
```

36

`Clock`, constructor, 2

```
  buttons.add( bStart );
  buttons.add( bStop );
  buttons.add( bReset );
  Container content = getContentPane();
  content.add( buttons, BorderLayout.NORTH );
  time = new Time();
  content.add( time, BorderLayout.CENTER );
  setSize( 240, 120 );
  startTime = System.currentTimeMillis();
}
```

37

`Clock, start()`

```
private void start() {
  if (clockThread == null) {
    clockThread = new Thread(this, "Reloj");
    startTime = System.currentTimeMillis();
    clockThread.start();
  }
}
```

38

Clock, stop(), reset()

```
private void stop() {
  clockThread = null;
  accumTime +=
    System.currentTimeMillis() - startTime;
  time.repaint();
}

private void reset() {
  accumTime = 0L;
  startTime = System.currentTimeMillis();
  time.repaint();
}
```


Clock, run()

```
public void run() {
  Thread myThread = Thread.currentThread();
  while (clockThread == myThread) {
    time.repaint();
    try {
      Thread.sleep(100);
    } catch (InterruptedException e){}
  }
}
```

Clock, clase interna Time

```
   private class Time extends JPanel
   {
Font timeFont = new Font( "SansSerif", Font.BOLD,
                                      32 );
     private static final int timeX = 60;
     private static final int timeY = 40;

     public void paintComponent(Graphics g) {
       super.paintComponent( g );
       long ticks;
```


Time, paintComponent()

```
if ( clockThread == null )
  ticks = accumTime;
else
  ticks = System.currentTimeMillis() - startTime
            + accumTime;
long tenths = ticks/100L;
long seconds = tenths/10L; tenths %= 10;
long minutes = seconds/60L; seconds %= 60;
StringBuffer sb = new StringBuffer();
 if ( minutes < 10 )
   sb.append( '0' );
 sb.append( minutes );
```

Time, paintComponent(), 2

```
      sb.append( ':' );
      if ( seconds < 10 )
        sb.append( '0' );
      sb.append( seconds );
      sb.append( '.' );
      sb.append( tenths );
      g.setFont( timeFont );
      g.drawString(sb.toString(), timeX, timeY);
    }  // fin de paintComponent
  }    // fin de Time
}      // fin de Clock
```

43

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.**

44

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.

45

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.

46

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.

47

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.*"

48

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.

49

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.

50

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.**

51

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.**

52

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 add`***EventType***`Listener(Listener l)`
- `public void remove`***EventType***`Listener(`
  `Listener l)`

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();
}
```

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 );
    }
}
```