code URJC

Programación Concurrente

Tema 5

Programación Concurrente en Java

Micael Gallego
micael.gallego@urjc.es
@micael_gallego

Universidad Rey Juan Carlos

code URJC

Programación Concurrente

Tema 5.3

Gestión de Hilos

Micael Gallego
micael.gallego@urjc.es
@micael_gallego

Universidad Rey Juan Carlos

Gestión de hilos

code URJC

- **Creación de hilos**
- Ciclo de vida de un programa con hilos
- Finalización de hilos
- Hilos demonios (*daemon*)
- Espera por la finalización de un hilo
- Otras cuestiones sobre los hilos en Java
- Conclusiones

Creación de hilos

code URJC

- Un hilo de ejecución es un objeto de la clase **java.lang.Thread**
- Como cualquier otro objeto, se puede almacenar en una variable, tiene constructor, métodos, etc…
- El código que se ejecutará en el nuevo hilo de ejecución se indica en el constructor con un objeto **java.lang.Runnable**
- Por motivos de depuración, se le puede asignar un nombre a un hilo al construirlo

Creación de hilos

```
package sample.hilo;

public class Programa {

  public static void main(String[] args) {

    Thread t = new Thread(()->
        System.out.println("Soy un hilo"));

    t.start();
  }
}
```

Inicio de la ejecución del hilo

Código que se ejecutará en un nuevo hilo

Creación de hilos

- **Estructura básica** de un programa con hilos

```
public class ProgramaCompleto {

  public void metodo(){
    System.out.println("Metodo ejecutado en un nuevo hilo");
  }

  private void exec() {
    new Thread(()-> metodo()).start();
  }

  public static void main(String[] args) {
    new ProgramaCompleto().exec();
  }
}
```

Creación de hilos

code URJC

- **Documentación**
  - JavaDoc de la clase Thread
    - http://download.oracle.com/javase/7/docs/api/java/lang/Thread.html
  - JavaDoc de la clase Runnable
    - http://download.oracle.com/javase/7/docs/api/java/lang/Runnable.html
  - Tutorial oficial de concurrencia en Java
    - http://download.oracle.com/javase/tutorial/essential/concurrency/

Ejercicio 1

code URJC

- Convertir este programa que usa **SimpleConcurrent** en un programa **Java estándar** que haga uso de los mecanismos de creación de **Threads**

```
import static
    simpleconcurrent.SimpleConcurrent.*;

public class ProdCons {

  static volatile boolean producido = false;
  static volatile double producto;

  public static void productor() {
    producto = Math.random();
    producido = true;
  }

  public static void consumidor() {
    while (!producido);
    print("Producto: "+producto);
  }

  public static void main(String[] args) {
    createThread("productor");
    createThread("consumidor");
    startThreadsAndWait();
  }
}
```

Gestión de hilos

- Creación de hilos
- **Ciclo de vida de un programa con hilos**
- Finalización de hilos
- Hilos demonios (*daemon*)
- Espera por la finalización de un hilo
- Otras cuestiones sobre los hilos en Java
- Conclusiones

Ciclo de vida de un programa con hilos

- Toda **aplicación** Java tiene al menos un hilo "main" que ejecuta el método estático **main**
- Existen otros hilos gestionados por la JVM (como el **recolector de basura**)
- Pueden existir otros hilos si se usan librerías o *frameworks* (GUI, Servidores web, …)
- Un **programa** finaliza su ejecución cuando
  - Todos sus hilos (no demonios) han finalizado su ejecución o
  - Se ejecuta el método **System.exit(…)**

Ciclo de vida de un programa con hilos

- **Ciclo de vida de un hilo**
  - Los hilos comienzan su ejecución cuando se ejecuta el método **Thread.start()**
  - Un hilo finaliza su ejecución
    - Cuando se han ejecutado todas sus sentencias o
    - Cuando se eleva una excepción no chequeada (**RuntimeException**) en el método **run()**

Gestión de hilos

- Creación de hilos
- Ciclo de vida de un programa con hilos
- **Finalización de hilos**
- Hilos demonios (*daemon*)
- Espera por la finalización de un hilo
- Otras cuestiones sobre los hilos en Java
- Conclusiones

Finalización de hilos

- Es muy **sencillo** iniciar la ejecución de un **nuevo hilo**
- En la mayoría de las ocasiones, un hilo se **ejecuta** hasta que ha **terminado** de ejecutar todas sus **instrucciones**
- Algunas veces, es necesario **detener la ejecución** de un hilo **antes** de que acabe por si mismo

Finalización de hilos

- Esto se puede deber a:
  - **Cancelación del usuario:** A través de GUI o por red
  - **Actividades limitadas en el tiempo:** Por ejemplo algoritmos que buscan una solución y devuelven la mejor solución encontrada cuando se cumple el tiempo de búsqueda
  - **Eventos que ocurren en otros hilos de la aplicación:** Por ejemplo algoritmos implementados en varios hilos
  - **Errores en la aplicación:** Un hilo puede detectar un error que obligue a los demás hilos a finalizar

Finalización de hilos

- **Thread.stop()**
  - Aparentemente la mejor forma de finalizar un hilo sería disponer de un método **stop()** en la clase **Thread**
  - Cuando se ejecutase el método **stop()** en el objeto de la clase **Thread** que representa el hilo, el hilo debería finalizar su ejecución
  - Esta solución sería muy **peligrosa**
  - La tarea que realiza el hilo que finaliza se quedaría **a medias** y los objetos que se estuvieran manipulando se quedarían en un estado **inconsistente**
  - Esto produciría errores imprevisibles en la aplicación

Finalización de hilos

- **Thread.stop()**
  - Al diseñar Java se pensó que el método **stop()** era una forma razonable de finalizar hilos
  - El método stop() **existe** en la clase **Thread**
  - Pero desde Java 1.2 (1998) se **desaconseja** encarecidamente su uso (el método está **obsoleto**, ***deprecated***) por los problemas mencionados
  - No han eliminado el método en las nuevas versiones de Java por **compatibilidad** hacia atrás

http://docs.oracle.com/javase/7/docs/technotes/guides/concurrency/threadPrimitiveDeprecation.html

Finalización de hilos

- **Interrupción de un hilo**
  - Para indicar a un hilo que debe finalizar su ejecución, se le envía una **interrupción**
  - Esta técnica se asemeja a las interrupciones de los procesos en los **sistemas operativos**
  - Para interrumpir un hilo, se invoca el método **interrupt()** en el objeto de la clase **Thread** que representa el hilo

Finalización de hilos

- **Interrupción de un hilo**
  - El hilo **no puede finalizar** su ejecución en un punto arbitrario (por los problemas indicados)
  - El hilo tiene que **preguntar periódicamente** si otro hilo le ha interrumpido
  - Si es así, debería **liberar los recursos** apropiadamente, **cerrar las conexiones** y dejar los objetos en un **estado estable**
  - Un hilo puede determinar si ha sido interrumpido invocando el método estático **Thread.interrupted()**

Finalización de hilos

El programa espera a que el usuario pulse ENTER

Se interrumpe al proceso de generación de números

```
public void exec() {

  Thread t = new Thread(()-> generateNumbers());
  t.start();

  Scanner teclado = new Scanner(System.in);
  System.out.print("Pulse ENTER para finalizar...");
  teclado.nextLine();

  t.interrupt();

  System.out.println("Hilo interrumpido.");
}
```

Finalización de hilos

```
public void generateNumbers() {
  try {
    FileWriter writer = new FileWriter("output.txt");
    while (true) {
      BigInteger prime =
         BigInteger.probablePrime(1024,new Random());
      writer.append(prime.toString());
      writer.append("\r\n");
      if(Thread.interrupted()){
        writer.append("Fin fichero");
        writer.close();
        return;
      }
    }
  } catch (IOException e) {
    System.err.println("Exception using file");
    e.printStackTrace();
    System.exit(1);
  }
}
```

El proceso genera números primos y los guarda en un fichero

Si se interrumpe, se cierra el fichero y finaliza la ejecución

Finalización de hilos

- Un hilo se puede **bloquear** a la espera de que se cumpla una condición, a la espera de datos de un **socket**, esperando que pase cierto tiempo (***sleep***)…
- Es posible que otros hilos quieran **interrumpir** al **hilo** que está **bloqueado**
- Pero si está bloqueado no puede consultar el método **Thread.interrupted()**

Finalización de hilos

- Para solucionar este problema, la mayoría de los métodos que se **bloquean** a la espera de una determinada condición elevan la excepción **InterruptedException**
- Esta excepción se eleva cuando se interrumpe el hilo mientras está **bloqueado** en dicho **método**
- La excepción **no debería ignorarse**, se debería finalizar la ejecución del hilo

Finalización de hilos

```
public void generateNumbers() {
  try {
    FileWriter writer = new FileWriter("output.txt");
    while (true) {
      BigInteger prime =
        BigInteger.probablePrime(1024,new Random());
      writer.append(prime.toString());
      writer.append("\r\n");
      try {
        Thread.sleep(1000);
      } catch(InterruptedException e){
        writer.append("Fin fichero");
        writer.close();
        return;
      }
    }
  } catch (IOException e) {
    System.err.println("Exception using file");
    e.printStackTrace();
    System.exit(1);
  }
}
```

El proceso genera números primos y los guarda en un fichero

Para dormir un proceso durante un tiempo se usa el método sleep*

El método se desbloquea el método en cuanto se interrumpe el hilo

* El método sleep se estudia más adelante

Finalización de hilos

code URJC

- Para **finalizar un hilo** el propio hilo tiene que estar preparado para ello cerrando recursos y finalizando su ejecución
- Tiene que revisar periódicamente si le han interrumpido (**Thread.interrupted()**)
- Si está bloqueado, se eleva una excepción cuando le interrumpen (**InterruptedException**)
- Se pueden usar ambas técnicas conjuntamente
- Aunque es **habitual** usar las interrupciones para la finalización, se podría usar para otro tipo de notificación

Gestión de hilos

- Creación de hilos
- Ciclo de vida de un programa con hilos
- Finalización de hilos
- **Hilos demonios (*daemon*)**
- Espera por la finalización de un hilo
- Otras cuestiones sobre los hilos en Java
- Conclusiones

Hilos demonios (*daemon*)

- Un hilo se puede marcar como **demonio** (*daemon*)
- Los hilos demonios finalizan **automáticamente** cuando han finalizado todos los hilos que no son demonios de un programa
- La palabra **demonio** es el nombre que reciben los servicios en máquinas Unix

Hilos demonios (*daemon*)

- Un hilo es demonio si el hilo que lo crea también lo es
- Los métodos **isDaemon()** y **setDaemon(…)** permiten cambiar esta propiedad de los hilos
- **No se suelen usar mucho** estos hilos porque es bastante habitual que los hilos tengan que realizar tareas de limpieza (cierre de ficheros, conexiones, …) antes de finalizar

Gestión de hilos

- Creación de hilos
- Ciclo de vida de un programa con hilos
- Finalización de hilos
- Hilos demonios (*daemon*)
- **Espera por la finalización de un hilo**
- Otras cuestiones sobre los hilos en Java
- Conclusiones

Espera por la finalización de un hilo

code URJC

- Si un hilo quiere **esperar a que otro hilo** termine su ejecución podría usar algún tipo de **sincronización condicional**
- En Java existe una forma directa de hacerlo invocando el método **join()** en el objeto de la clase **Thread** que representa a ese hilo
- El método eleva la excepción **InterruptedExcepcion** si otro hilo le interrumpe mientras espera
- Existen versiones de **join()** para indicar el tiempo máximo de espera

Espera por la finalización de un hilo

code URJC

```
public class ProgramaJoin {

  public void metodo(){
    for(int i=0; i<3; i++){
      System.out.println("Iteración "+i);
    }
  }

  public void exec() throws InterruptedException {

    Thread t = new Thread(()-> metodo());
    t.start();
    t.join();

    System.out.println("Programa finalizado");
  }

  public static void main(String[] args)
     throws InterruptedException {
    new ProgramaJoin().exec();
  }
}
```

Ignoramos la InterruptedException porque ningún hilo va a interrumpir al hilo principal

Se bloquea hasta que el hilo termina su ejecución

Gestión de hilos

- Creación de hilos
- Ciclo de vida de un programa con hilos
- Finalización de hilos
- Hilos demonios (*daemon*)
- Espera por la finalización de un hilo
- **Otras cuestiones sobre los hilos en Java**
- Conclusiones

Otras cuestiones sobre hilos en Java

- Por completitud, veremos otras cuestiones más accesorias sobre los hilos en Java
  - **Prioridad** de ejecución
  - Grupos de hilos con **ThreadGroup**
  - Bloquear un hilo por un tiempo (**sleep**)
  - Otros métodos de la clase **Thread**

Prioridad de ejecución

- Los hilos en Java tienen un prioridad de ejecución
- La prioridad de un nuevo hilo es la misma que la del hilo que lo creó
- La clase **Thread** tiene los métodos:
  - **setPriority(int p):** Establece la prioridad. Su valor tiene que estar comprendido entre **Thread.MIN_PRIORITY** (1) y **Thread.MAX_PRIORITY** (10)
  - **int getPriotiry():** Devuelve la prioridad del hilo

Prioridad de ejecución

- Java se pueda ejecutar en diferentes sistemas operativos y arquitecturas hardware
- Por esto no se puede asegurar una semántica concreta para las prioridades de los hilos
- La prioridad de un hilo es un "deseo" del desarrollador que la JVM y el SO respetarán en la medida de sus posibilidades

Prioridad de ejecución

- Usos habituales de las prioridades

Valores	Uso
10	Gestión de crisis
7-9	Interfaz de usuario
4-6	Entrada / Salida
2-3	Tareas en segundo plano
1	Ejecutar si no hay otra cosa

Grupos de hilos con ThreadGroup

code URJC

- Los objetos de la clase Thread se pueden agrupar en grupos representados por objetos de la clase **ThreadGroup**
- Un **ThreadGroup** puede tener dentro a otros grupos de hilos, creando una estructura de árbol
- Los **ThreadGroup** no se utilizan mucho. Se usan únicamente para:
  - Limitar la prioridad de los hilos que contienen
  - Gestionar ciertas propiedades de los hilos de forma conjunta

http://download.oracle.com/javase/7/docs/api/java/lang/ThreadGroup.html

Bloqueo durmiendo (*sleep*)

- La forma más **sencilla** de **bloqueo** se tiene cuando un hilo **suspende** su ejecución durante un intervalo de tiempo
- Para ello, un hilo puede invocar el método estático **sleep(long millis)** de la clase **Thread**
- El método que ejecuta ese método quedará **bloqueado durante el tiempo** indicado
- Como Java no es sistema de tiempo real, **no garantiza** que se cumpla de forma **precisa** el tiempo indicado

Bloqueo durmiendo (*sleep*)

- El método **sleep(…)** eleva un excepción de tipo **InterruptedException** cuando se interrumpe un hilo mientras está bloqueado

```
public class SleepExample {

  public static void main(String[] args) throws InterruptedException {
    while(true){
      System.out.println("Trabajar");
      System.out.println("Disfrutar");
      Thread.sleep(1000); //Por la noche
    }
  }
}
```

Métodos auxiliares sobre hilos

- La clase Thread proporciona algunos métodos auxiliares
  - **Thread.currentThread():** Método estático que devuelve el objeto de la clase Thread que representa el hilo que llama al método
  - **getName():** Devuelve el nombre del hilo
  - **isAlive():** Indica si el hilo está en ejecución
  - **getState():** Devuelve el estado del hilo (nuevo, ejecutando, esperando…)
  - Revisar el JavaDoc de la clase Thread

http://download.oracle.com/javase/7/docs/api/java/lang/Thread.html

Ejercicio 2

code URJC

- Se desea implementar en Java un programa con dos hilos, el hilo principal Main y un hilo de Mensajes
- El hilo principal:
  - Crea el hilo de mensajes
  - Espera a que el hilo de mensajes finalice
  - Si no lo hace, cada segundo imprime "Todavía esperando..."
  - Cuando ha pasado un tiempo máximo de 5 segundos, si el hilo de mensajes no ha terminado todavía, dice "Cansado de esperar!", le interrumpe y espera a que termine
  - Al finalizar el hilo dice "Por fin!"

Ejercicio 2

- El hilo de mensajes:
  - Dice las siguientes frases cada dos segundos: "La vida es bella", "O no...", "Los pajaritos cantan", "Y molestan..."
  - Si el hilo principal le interrumpe antes de terminar, dice "Se acabó!"
- Cada hilo debe indicar su nombre cada vez que dice algo

Gestión de hilos

- Creación de hilos
- Ciclo de vida de un programa con hilos
- Finalización de hilos
- Hilos demonios (*daemon*)
- Espera por la finalización de un hilo
- Otras cuestiones sobre los hilos en Java
- **Conclusiones**

Conclusiones

- Se ha visto que en Java se puede controlar de forma **precisa el ciclo de vida** de un hilo
  - Un hilo se puede **crear** en cualquier momento de la ejecución de un programa y desde cualquier hilo
  - Un hilo puede **finalizar** de forma natural o se le puede **interrumpir** para que finalice
- Se ha visto una forma compacta de crear objetos **Runnable** con **clases anónimas** para la creación de hilos

Conclusiones

- Se han presentado cuestiones relacionadas con los hilos
  - Hilos demonios (*daemon*)
  - Bloqueo durmiendo (*sleep*)
  - Prioridad de ejecución de los hilos
  - Métodos de la clase Thread para gestión de hilos