Esercizi sui Thread

Esercitazione di Laboratorio di Programmazione di Rete A

Daniele Sgandurra

Università di Pisa

24/09/2008

Creare Thread: l'Interfaccia Runnable

Per eseguire un'attività in un thread parallelo:

1. Si inserisce il codice dell'attività nel metodo run di una classe che implementa l'interfaccia runnable, ad es.:

```
class MyRunnable implements Runnable
{
    public void run()
    {//codice dell'attivita'
    }
}
```

2. Si costruisce un oggetto della classe:

```
Runnable r = new MyRunnable();
```

3. Si costruisce un oggetto Thread della classe:

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

4. Si avvia il thread

```
t.start();
```

Approccio Alternativo: Estendere la Classe Thread

- Si realizza una sottoclasse della classe Thread, ad es.:

```
class MyThread extends Thread
{
    public void run()
    {//codice dell'attivita'
    }
}
```

- Poi si costruisce un oggetto della sottoclasse `MyThread` e si invoca il metodo `start()`.
- Tuttavia questo approccio è sconsigliato: è auspicabile disaccoppiare l'attività da eseguire in parallelo ad altre rispetto al meccanismo che la esegue.

Attenzione!

- Non si deve invocare direttamente il metodo `run()` della classe `Thread` o dell'oggetto `Runnable`.
- Eseguendo il metodo `run()` si esegue l'attività nello stesso thread senza avviarne uno nuovo.
- Bisogna sempre invocare il metodo `start()` della classe `Thread`, in modo da creare un nuovo thread che, una volta avviato, eseguirà il metodo `run()`.

Interrompere i Thread

- Un thread termina quando il suo metodo run() ritorna.
- Non è possibile forzare l'interruzione di un thread.
- Si può utilizzare il metodo interrupt() per richiedere l'interruzione di un thread.
- Questo metodo imposta lo stato di interruzione del thread.
- Si usa il metodo statico isInterrupted() per verificare se lo stato di interruzione è stato impostato.

Controllare lo Stato di Interruzione

```
while(!Thread.currentThread().isInterrupted() && “altro lavoro”)
{
   //continua il lavoro
}
```

- Il metodo statico Thread.currentThread() ritorna il thread corrente:
  - non si può scrivere `this.isInterrupted()` nel `run`, perché this è un riferimento ad un `Runnable` non al thread (a meno che non si estenda la classe `Thread`).
- Se un thread è bloccato (es. in sleep o wait) e riceve un’interruzione, non potendo verificare lo stato d’interruzione, viene lanciata una InterruptedException.
- Java non richiede che un thread interrotto debba essere terminato: l’interruzione serve solo ad avvertire il thread. È il thread stesso a decidere come reagire all’interruzione.

Controllare lo Stato di Interruzione

```
public void run()
{
    try
    {
        ...
        while(...)
        {
            ...
            Thread.sleep(time);
        }
     }
    catch(InterruptedException ex) { ... }
    finally{ ... }
 }
```

Attenzione!

- Ci sono due metodi simili in Java: interrupted() e isInterrupted().
- Il metodo Thread.interrupted() è un metodo statico che verifica se il thread che lo invoca è stato interrotto. La chiamata a questo metodo pulisce lo stato di interruzione.
- Il metodo isInterrupted(), invece, è un metodo di istanza usato per verificare se un thread (di cui si ha il riferimento) è stato interrotto, e non pulisce lo stato di interruzione.
- Quando il metodo sleep genera una InterruptedException, ripulisce anche lo stato di interruzione.

Pool di Thread

- La creazione di un nuovo thread può essere molto dispendiosa.
- Invece, un pool di thread contiene molti thread inattivi, pronti per essere avviati.
- Quando al pool viene assegnato un runnable, uno dei thread esegue il metodo `run()`.
- Quando il metodo `run()` ritorna, il thread non termina ma rimane attivo per ricevere altri task.
- La classe Executors prevede alcuni metodi statici per costruire pool di thread.

Pool di Thread

Metodi di fabbrica di Executors:

- newCachedThreadPool(): si creano nuovi thread se necessario; i thread inattivi rimangono vivi per 60 secondi;
- newFixedThreadPool(): il pool contiene un insieme fisso di thread;
- newSingleThreadPool(): un pool degenere con un solo thread che esegue le attività sottomesse sequenzialmente.

Es.:

```
ExecutorService threadExecutor = Executors.newCachedThreadPool();
for(int i = 0; i < nTasks; i++)
   threadExecutor.execute(new myRunnable());
```

Esercizio 1

- Scrivere un programma che avvia un thread che va in sleep per 10 secondi.
- Il programma principale interrompe il thread dopo 5 secondi.
- Il thread deve catturare l'eccezione e stampare il tempo trascorso in sleep.
- *Nota*: per ottenere l'ora corrente usare il metodo `System.currentTimeMillis()`, che rappresentata il numero di millisecondi (long) trascorsi dal 1/1/1970.

Esercizio 2

- Scrivere un programma che attiva un thread T che effettua il calcolo approssimato di π.
- Il programma principale prende come input da riga di comando un parametro che indica il grado di accuratezza (accuracy) per il calcolo di π e il tempo massimo di attesa dopo cui il main interrompe il thread T.
- Il thread T effettua un ciclo infinito per il calcolo di π usando la serie di *Gregory-Leibniz*: $\pi = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} \ldots$.
- Il thread esce dal ciclo quando una delle due condizioni seguenti risulta verificata:
  1. Il thread è stato interrotto dal main.
  2. La differenza tra il valore stimato di π e il valore `Math.PI` è minore di accuracy.

Esercizio 3

- Creare un pool di thread ognuno dei quali va in sleep per un periodo casuale di millisecondi compreso tra 0 e 5000.
- Ogni thread, prima di effettuare la sleep, stampa a video il proprio nome (passato al momento della creazione) e il numero di millisecondi di durata della sleep e, al termine della sleep, stampa l'avvenuta conclusione.
- Il main, dopo aver passato al pool i job da eseguire, stampa a video una stringa di avvenuta terminazione.
- Il numero di job creati è un parametro di riga di comando.
- Per generare numeri casuali, usare la classe java.util.Random e il metodo nextInt(int).

Soluzioni

Inviare la soluzione degli esercizi (solo i file .java) a :

`ricci@di.unipi.it`
`sgandurra@di.unipi.it`

Tra due settimane saranno disponibili le soluzioni.