Corso sul linguaggio Java

Modulo JAVA9

B1 – I thread

Prerequisiti

- Programmazione base in Java
- Programmazione *multithreading*
- Concetti di base sui sistemi operativi
- Sezione critica

Introduzione

Normalmente un programma viene eseguito in modo sequenziale: la **CPU** esegue, una dopo l'altra, le istruzioni del programma, dall'inizio alla fine.

Come si può fare in modo che più parti indipendenti di un stesso programma, una volta lanciato, **possano essere eseguite in parallelo**, **ossia contemporaneamente**?

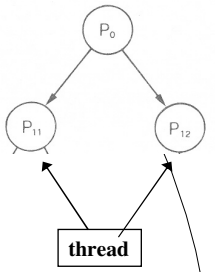
In Java la tecnica della programmazione concorrente si applica mediante l'uso dei **thread** dei quali ci occupiamo in questa Unità.

Al giorno d'oggi, ogni buon programmatore deve padroneggiare la programmazione *multithreading*, che tratta l'esecuzione concorrente di più *thread*, dello stesso processo.

Java e il multiprocessing

Java consente di scrivere applicazioni in grado di generare più processi che evolvono in *multiprocessing* e che sono detti **thread**.

```
class P0
{ P11;          // P0 genera processo P11
  P12;          // P0 genera processo P12
}
class P11       // codice di P11
{ … }
class P12       // codice di P12
{ … }
```



Esempi di *thread*

Alcuni esempi di *thread* sono:

- scrollare una pagina Web, mentre si scarica un'applet o un'immagine
- vedere un'animazione mentre si ascolta musica
- stampare una pagina mentre se ne sta scaricando un'altra.

Il *thread* generato dal main()

```
public class ThreadExists
{public static void main (String args[])
 {Thread t = Thread.currentThread();
  t.setName ("Thread principale");
  t.setPriority (10);
  System.out.println("Thread in esecuzione: " + t);
  try {for (int n = 5; n > 0; n--)  { System.out.println("" + n);
                                      t.sleep(1000);      }
      }
  catch (InterruptedException e)  {System.out.println("Thread interrotto");  }
 }
}
```

In ogni applicazione un *thread* esiste sempre ed è quello prodotto dall'esecuzione del **main**().

L'applicazione ha un'esecuzione di circa 5 secondi, in quanto le stampe di *i* vengono intervallate dalla chiamata al metodo **sleep** (1000), che effettua la pausa di un secondo.

Il *thread* generato dal main()

Quando viene mandato in esecuzione un qualsiasi programma Java, viene automaticamente eseguito il codice all'interno del metodo **main**().

In effetti il sistema operativo (o meglio la *Java Virtual Machine*) crea un processo a cui è associato un solo *thread*, che provvede ad eseguire il codice nel **main**().

Tutti i thread aggiuntivi che noi creiamo vengono associati allo stesso processo.

Metodi della classe *Thread*

Nell'esempio possiamo notare:

- il *thread t* è il processo corrente, ossia il **main**(), per cui non necessita dell'operatore **new** in quanto già esiste;
- il metodo **setName**() consente di assegnare un nome al *thread*;
- il metodo **setPriority**() consente di assegnare al *thread* un valore di priorità (sono ammessi valori da 1 a 10, indicanti priorità crescenti);
- il metodo **sleep**() esegue sul *thread* un'operazione di blocco temporaneo, per un dato numero di millisecondi;

Il blocco **try-catch** è superfluo in questo esempio, ma serve quando ci sono più *thread* che possono sollevare eccezioni.

Creazione di un *thread*

- Esempio 1

Per creare un *thread* occorre:

- estendere la classe **Thread**
- ridefinire il metodo **run**() (ciò che il *thread* deve fare)
- avviare il *thread* con **start**()

```
class Contatore extends Thread
 { public void run()
   { int n = 0;
     while(true)   { System.out.println("“ + n); ++n; }
   }
  public static void main (String[] args)
 { new Contatore().start(); }
}
```


Creazione di un *thread*

- Esempio 2

run() contiene ciò che il *thread* deve fare

```
public class MyThread extends Thread
{   public void run()
    {     try  {   Thread.sleep (3000);
                   System.out.println(“… e io sono ciop!");
                 }
          catch (InterruptedException e)
               {System.out.println("Thread interrotto");  }
    }
    public MyThread()
    {     // azioni per il costruttore    }
    public static void main (String args[])
    {     Thread t = new MyThread();
          t.start();.
          System.out.println(“Io sono Cip…”);
    }
}
```

Il **main**() termina con la stampa, ma *MyThread* resta in esecuzione fino a che non siano trascorsi 3 secondi

Creazione di un *thread*
- Esempio 2

Se creiamo un solo *thread*, come nell'esempio, avremo quindi realmente in esecuzione due *thread*:

- uno da noi creato (nel nostro caso il *thread* t di tipo *MyThread*)
- uno creato automaticamente all'avvio del programma, cioè il *thread* che esegue il **main**().

Questi due *thread* vengono eseguiti in modo parallelo a tutti gli effetti, come se fossero due processi distinti.

E' importante osservare che un programma *multithreaded* termina solamente quando tutti i *thread* che lo compongono hanno terminato

Il metodo **run**()

Il metodo **run**() costituisce l'entry point del *thread*:

- ogni istruzione inclusa in questo metodo viene eseguita dal *thread* o nei metodi invocati direttamente o indirettamente da **run**()
- un *thread* è considerato "vivo" finchè il metodo **run**() non ritorna. Quando **run**() ritorna il *thread* è considerato "morto"

E' importante osservare che:

- una volta che un *thread* è "morto" non puo essere rieseguito (pena un'eccezione *IllegalThreadStateException*), ma se ne deve creare una nuova istanza.
- non si può far partire lo stesso *thread* (la stessa istanza) più volte.

Il metodo **start**()

Il metodo **start**() costituisce l'avvio del *thread*:

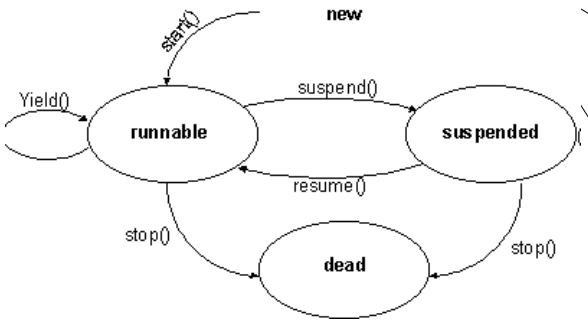
- una chiamata di **start**() ritorna immediatamente al chiamante senza aspettare che l'altro *thread* abbia effettivamente iniziato l'esecuzione.
- **start**() avverte la JVM che l'altro *thread* è pronto per l'esecuzione (quando lo *scheduler* lo riterrà opportuno). Prima o poi verrà invocato il metodo **run**() del nuovo *thread*

I due *thread* saranno eseguiti in modo concorrente ed indipendente.

L'ordine con cui ogni *thread* eseguirà le proprie istruzioni è noto, ma l'ordine in cui le istruzioni dei vari *thread* saranno eseguite effettivamente è indeterminato (**indeterminismo**).

Il ciclo di vita di un *thread*

Il ciclo di vita di un *thread* in Java è rappresentato dal seguente diagramma degli stati.



I passaggi di stato dei *thread*

Gli stati in cui un processo si può trovare sono:

- un *thread* appena creato è nello stato *new*, ma non è attivo. Per attivarlo occorre chiamare **start**(): il *thread* passa nello stato *runnable*.
- un *thread runnable* ottiene di tanto in tanto il processore (o uno dei processori)
- un *thread runnable* può cedere il passo agli altri con **yield**() rimanendo *runnable* (descritto in Unità successive)
- un *thread* può sospendersi con **suspend**()
- un *thread* sospeso torna in esecuzione con **resume**()
- un *thread* può terminare eseguendo **stop**()

Un thread non riparte con **start**()!!!

Operazioni sconsigliate

Occorre osservare che i metodi:

- **stop**()
- **suspend**()
- **resume**()

è bene siano evitati poiché agiscono direttamente su un *thread* senza alcun controllo sul suo stato e sulle risorse in uso.

Può quindi succedere che:

- un *thread* T che abbia acquisito una risorsa R venga stoppato, con la conseguenza che la risorsa resta allocata a T; di conseguenza, i *thread* che necessitano di R restano in attesa indefinita;
- un thread sospeso durante una sezione critica, potrebbe portare a risultati errati.

La classe **Thread**

Sono riportati sinteticamente i principali metodi della classe **Thread**. Alcuni di essi, sono descritti in successive Unità.

METODO	EFFETTO
Thread currentThread()	Restituisce il *thread* del processo corrente
void setName (**String** *tname*)	Assegna al *thread* il nome *tname*
String getName()	Restituisce il nome del *thread*
void setPriority (**int** *priority*)	Assegna al *thread* il valore di priorità *priority*
int getPriority()	Restituisce la priorità corrente
void sleep (**int** *msec*)	Pone in attesa il *thread* per *msec* millisecondi
void start()	Avvia il *thread*
void run()	Specifica l'elaborazione che deve eseguire il thread
void yield()	Cede il controllo ad un altro thread rimettendosi in coda
void join()	Attende la terminazione del *thread*

Argomenti

- Java e il multiprocessing
- Esempi di *thread*
- Il *thread* generato dal **main**()
- Metodi della classe **Thread**
- Creazione di un *thread*
  - Esempio 1
  - Esempio 2
- Il metodo **run**()
- Il metodo **start**()
- Il ciclo di vita di un *thread*
- I passaggi di stato dei *thread*
- Operazioni sconsigliate
- La classe **Thread**

Altre fonti di informazione

- P.Gallo, F.Salerno – Informatica Generale 1, ed. Minerva Italica
- M.Romagnoli, P.Ventura – Linguaggio C/C++, ed. Petrini
- M. Bigatti – Il linguaggio Java, ed. Hoepli