Esercizi sugli Oggetti Monitor

Esercitazione di Laboratorio di Programmazione di Rete A

Daniele Sgandurra

Università di Pisa

29/10/2008

`Wait` e `Notify`

- Ogni oggetto in Java ha un lock implicito.
- Il lock implicito è associato ad una sola condizione.
- Il metodo `wait()` aggiunge un thread al set di attesa della condizione.
- I metodi `notify()/notifyAll()` sbloccano i thread in attesa.

`Wait` e `Notify`

- Si utilizza `wait()` per attendere il cambiamento di una condizione che non può essere controllata dal metodo corrente:
  - di solito questa condizione è modificata da un altro thread.
- Quando un thread invoca la `notify()` o `notifyAll()`, il thread bloccato viene svegliato:
  - un thread invoca questi metodi quando una condizione è cambiata.

`Wait` e `Notify`

- `void wait()`: fa sì che un thread aspetti fino a quando non viene notificato.
- `void notify()`: sblocca un thread scelto a caso tra i thread che hanno chiamato la `wait()` su questo oggetto.
- `void notifyAll()`: sblocca i thread che hanno chiamato la `wait()` su questo oggetto.

Da notare che:

1. tutti questi metodi possono essere invocati solo dentro un metodo o blocco sincronizzato;
2. se il thread corrente non è proprietario del lock sull'oggetto, viene lanciata una `IllegalMonitorStateException`.

Importante!

- I metodi `sleep()` e `yield()` non rilasciano il lock associato all'oggetto.
- Se un thread invoca la `wait()`, l'esecuzione del thread è sospesa e il lock dell'oggetto è rilasciato:
  - quindi, il lock può essere acquisito da altri thread.
- Altri metodi `synchronized` dell'oggetto possono essere invocati durante la `wait()`:
  - questo aspetto è essenziale: infatti, questi metodi potrebbero provocare il cambiamento della condizione e quindi il risveglio del thread sospeso.

Esempio della Banca Rivisto

Esercizio sulla banca rivisto: la condizione sono i fondi sufficienti.

- Versione precedente:

```
public synchronized void transfer(int from, int to, int amount)
                    throws InterruptedException
{
   if(accounts[from] < amount)
      return;
   accounts[from] -= amount;
   accounts[to] += amount;
}
```

- Versione con `wait()` e `notifyAll()`:

```
public synchronized void transfer(int from, int to, int amount)
                    throws InterruptedException
{
   while(accounts[from] < amount)
      wait();
   accounts[from] -= amount;
   accounts[to] += amount;
   notifyAll();
}
```

Blocchi Sincronizzati

Esempio di utilizzo nei blocchi sincronizzati:

- `wait()`:

```
synchronized(x)
{
    while(someCondition)
        x.wait();
}
```

- `notifyAll()`:

```
synchronized(x)
{
    ...
    //la condizione cambia
    x.notifyAll();
}
```

Monitor

Concetto introdotto da Per Brinch Hansen e Tony Hoare negli anni'70. Un monitor deve avere le seguenti proprietà:

- Un monitor è una classe che prevede solo campi `private`.
- Ogni oggetto di questa classe ha un lock associato.
- Tutti i metodi della classe sono bloccati da questo lock:
  - dato che tutti i campi sono privati, questa condizione garantisce che nessun thread possa accedere ai campi mentre un altro li sta elaborando.
- Il lock può avere un qualsiasi numero di condizioni associate.

In realtà, in Java non è richiesto che tutti i campi siano `private` e tutti i metodi siano `synchronized`...

Esercizio 1

Il laboratorio di Informatica del Polo Marzotto è utilizzato da tre tipi di utenti: studenti, tesisti e professori. Ogni utente deve fare una richiesta al tutor per accedere al laboratorio. I computer del laboratorio sono numerati da 1 a 20. Le richieste di accesso sono diverse a seconda del tipo dell'utente:

- i professori accedono in modo esclusivo a tutto il laboratorio, poiché hanno necessità di utilizzare tutti i computer per effettuare prove in rete;
- i tesisti richiedono l'uso esclusivo di un solo computer, identificato dall'indice `i`, poiché su quel computer è installato un particolare software necessario per lo sviluppo della tesi;
- gli studenti richiedono l'uso esclusivo di un qualsiasi computer;
- i professori hanno priorità su tutti nell'accesso al laboratorio, i tesisti hanno priorità sugli studenti.

Esercizio 1

- Scrivere un programma JAVA che simuli il comportamento degli utenti e del tutor. Il programma riceve in ingresso il numero di studenti, tesisti e professori che utilizzano il laboratorio ed attiva un thread per ogni utente.
- Ogni utente accede `k` volte al laboratorio, con `k` generato casualmente. Simulare l'intervallo di tempo che intercorre tra un accesso ed il successivo e l'intervallo di permanenza in laboratorio mediante il metodo `sleep`.
- Il tutor deve coordinare gli accessi al laboratorio. Il programma deve terminare quando tutti gli utenti hanno completato i loro accessi al laboratorio.

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.