Esercizi sulla Sincronizzazione

Esercitazione di Laboratorio di Programmazione di Rete A

Daniele Sgandurra

Università di Pisa

08/10/2008

La Classe `ReentrantLock`

- Un lock protegge determinate sezioni di codice critiche, permettendo ad un solo thread alla volta di eseguire quel codice.
- JDK 5.0 ha introdotto la classe `ReentrantLock`.

```
private Lock myLock = new ReentrantLock();
...
myLock.lock();
try
{
   //sezione critica
}
finally
{
   myLock.unlock();
}
```

La Classe `ReentrantLock`

- Non appena un thread acquisisce un lock sull'oggetto `myLock`, nessun altro thread può superare l'istruzione `lock()`.
- Se altri thread invocano `lock()`, rimangono bloccati fino a che il primo thread non sblocca il lock tramite il metodo `unlock()`.
- Il lock è rientrante perché un thread *può acquisire più volte un lock che già possiede*.
- Il lock possiede un contatore che tiene traccia delle chiamate del metodo `lock()`.
- Il thread deve invocare il metodo `unlock()` per ogni chiamata di `lock()`, per rilasciare il lock.
- Perciò, un blocco protetto da lock può invocare un altro metodo che contiene un blocco protetto dallo stesso lock.

La Classe `ReentrantLock`

- Tramite il metodo `tryLock()` è possibile tentare di acquisire un lock, senza essere sospesi se il lock è già stato acquisito.

```
private ReentrantLock myLock = new ReentrantLock();

public void criticalMethod()
{
    if(myLock.tryLock());
    {
        try
        {
            //sezione critica
        }
        finally{ myLock.unlock(); }
    }
    else
        ...
}
```

La Classe `ReentrantLock`

```
private ReentrantLock myLock = new ReentrantLock();

public void criticalMethod()
{
   try
   {//come prima, ma prova ad acquisire il lock per 2 sec
      if(myLock.tryLock(2, TimeUnit.SECONDS));
      {
         try
         {
            //sezione critica
         }
         finally{ myLock.unlock(); }
      }
      else
         ...
   }catch(InterruptedException e)...
}
```

La Parola Chiave `synchronized`

- A partire dalla versione 1.0, ogni oggetto in Java ha un lock implicito.
- La parola chiave `synchronized` mette a disposizione automaticamente un lock associato a una "condizione".
- Se un metodo è dichiarato con la parola chiave `synchronized`, il lock implicito dell'oggetto protegge l'intero metodo.
- Si possono dichiarare anche metodi statici sincronizzati: si acquisisce il lock implicito dell'oggetto `class` associato.

La Parola Chiave `synchronized`

```
public synchronized void criticalMethod()
{
   //regione critica
}
```

è equivalente a:

```
public void criticalMethod()
{
   lockimplicito.lock();
   try
   {
      //regione critica
   }
   finally
   {
      lockimplicito.unlock();
   }
}
```

Blocchi Sincronizzati

- Sintassi per un blocco sincronizzato:

```
private Object obj = new Object();
...
public void method()
{
    ...
    synchronized(obj)
    {
        //regione critica
    }
    ...
}
```

- obj è un oggetto creato solo per sincronizzare il blocco.
- Se si vogliono sincronizzare diversi blocchi, bisogna sincronizzarci sullo stesso lock. Ad es., all'interno di metodi di istanza di una stessa classe il lock può essere `this`.

Esercizio 1

- Simulare il comportamento di una banca che gestisce un certo numero di conti correnti. In particolare interessa simulare lo spostamento di denaro tra due conti correnti.
- Ad ogni conto è associato un thread T che implementa un metodo che consente di trasferire una quantità casuale di denaro tra il conto servito da T ed un altro conto il cui identificatore è generato casualmente.
- Sviluppare una versione *non thread-safe* del programma in modo da produrre un comportamento scorretto del programma.
- Definire 3 versioni *thread-safe* del programma che utilizzino, rispettivamente:
  - Lock esplicite;
  - Blocchi sincronizzati;
  - Metodi sincronizzati.

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.