Esercitazione 11
23 maggio 2011

Monitor in Java

Esercizio 1

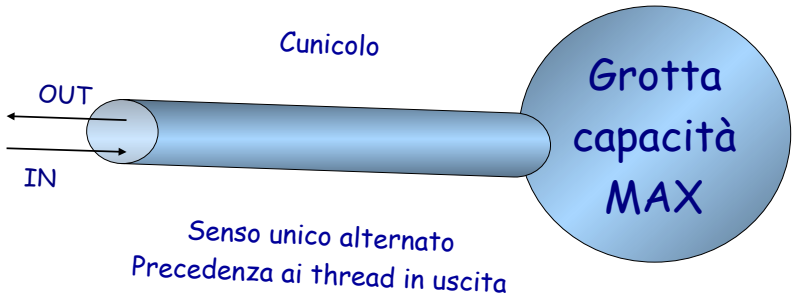
In un sito di interesse geologico e` possibile visitare una grotta sotterranea, raggiungibile soltanto attraverso uno stretto cunicolo.

Per motivi di sicurezza, la grotta non puo` ospitare piu` di MAX visitatori contemporaneamente.

Il cunicolo e` cosi` stretto, da non consentire il passaggio contemporaneo di 2 persone in direzioni opposte: il cunicolo deve quindi essere percorso a senso unico alternato.

Per evitare situazioni di deadlock, nell'accesso al cunicolo i processi in uscita dalla grotta devono essere favoriti rispetto a quelli in ingresso.

Utilizzando il linguaggio Java, realizzare una politica di controllo degli accessi alla grotta che tenga conto delle specifiche indicate.



Impostazione

Quali thread?

- n visitatori, ognuno con la medesima struttura:
  1. **<entrata tunnel in direzione IN>**
  2. <attraversamento tunnel in direzione IN>
  3. **<uscita tunnel in direzione IN - ingresso grotta>**
  4. <visita grotta>
  5. **<uscita grotta - entrata tunnel in direzione OUT>**
  6. <attraversamento tunnel in direzione OUT>
  7. **<uscita tunnel in direzione OUT>**

Quale risorsa comune?

- **Grotta:** associamo alla Grotta un **"monitor"**, che controlla gli accessi in base alla politica data. La sincronizzazione viene realizzata mediante variabili condizione.
- **Metodi public**: entrata e uscita dal tunnel (1,3,5,7)

Struttura Thread

```
public class visitatore extends Thread{
   private final int IN=0;
   private final int OUT=1;

   monitor g;
   int id;

   public visitatore(monitor G, int ID){
         g=G;
         id=ID;
   }

   public void run() {
         try {    g.EntraTunnel(IN, id);
                  sleep((int)(Math.random()*1000));// attrav. Tunnel ingresso
                  g.EsciTunnel(IN, id);
                  sleep((int)(Math.random()*5000));
                  g.EntraTunnel(OUT, id);
                  sleep((int)(Math.random()*1000)); // attrav. Tunnel uscita
                  g.EsciTunnel(OUT, id);
         } catch (Exception e) {}
   }
}
```


Monitor grotta: variabili

- Stato del Tunnel:
  - Numero visitatori nella grotta:
    `private int dentro;`
  - Numero visitatori nel tunnel per ogni direzione:
    `private int []tunnel=new int[2];`
- Sincronizzazione:
  - lock:
    `private Lock lock = new ReentrantLock();`
  - Per la sospensione dei thread che vogliono accedere al tunnel, 1 condition per ogni direzione:
    `private Condition []coda=new Condition[2];`
  - Contatori dei thread sospesi per ogni direzione:
    `private int []sosp=new int[2];`

Monitor

```
public class monitor {
   private final int MAX=7;       //capacita` della grotta
   //costanti di direzione:
   private final int IN=0;
   private final int OUT=1;
   private Lock lock = new ReentrantLock();
   private Condition []coda=new Condition[2];
   private int dentro; //persone nella grotta
   private int []tunnel=new int[2]; //persone nel tunnel per ogni direzione
   private int []sosp=new int[2]; //persone sospese per ogni direzione

   public monitor()
   {    int i;
        for (i=0; i<2; i++)
        {       tunnel[i]=0;
                sosp[i]=0;
                coda[i]=lock.newCondition();
        }
        dentro=0;
   }
```


Monitor

```
public void EntraTunnel(int d, int id) throws InterruptedException
{       lock.lock();
        try {
        if (d==IN)
        {       while ((tunnel[OUT]>0) || (dentro+tunnel[IN]==MAX) ||
                                sosp[OUT]>0)
                {       sosp[d]++;
                        coda[d].await();
                        sosp[d]--;
                }
                tunnel[d]++;
        }
        else //uscita dalla grotta
        {       while (tunnel[IN]>0)
                {       sosp[d]++;
                        coda[d].await();
                        sosp[d]--;
                }
                tunnel[d]++; dentro--;
        }
        System.out.println("visitatore "+id+" entrato in dir."+d+"\n");

        } finally {lock.unlock();}
}
```

Monitor

```
public void EsciTunnel(int d, int id) throws InterruptedException
{       lock.lock();
        try {  tunnel[d]--;
               if (d==IN)
                      dentro++;
               if (tunnel[d]==0) //corridoio vuoto
                      coda[altra(d)].signalAll();
               System.out.println("vis. "+id+
               " uscito in direzione"+d+"\n");
        } finally {lock.unlock();}
}
```


Monitor

```
public void EsciTunnel(int d, int id) throws InterruptedException
{       lock.lock();
        try {   tunnel[d]--;
               if (d==IN)
                      dentro++;
               if (tunnel[d]==0) //corridoio vuoto
                      coda[altra(d)].signalAll();
               System.out.println("vis. "+id+
               " uscito in direzione"+d+"\n");
        } finally {lock.unlock();}
}

private int altra(int dir)
{  int ret;
   if (dir==IN) ret=OUT;
   else ret=IN;
   return ret;
}

}  // fine monitor
```

Test

```
import java.util.concurrent.*;

public class Grotta {
   public static void main(String[] args) {
        int N=10;
        int i;
        int NV;

        visitatore [] V = new visitatore[N];
        monitor g=new monitor();
        for (i=0;i<N;i++) //creazione uomini e donne
               V[i] = new visitatore(g,i+1);

        for (i=0;i<N;i++) //attivazione
               V[i].start();
        }
}
```


Esercizio 2

Si consideri un sito web dedicato ai collezionisti di figurine dell'album "***Campionato di calcio 2010-2011***".

L'album è composto da 100 diverse figurine, ognuna individuata univocamente da un intero; tra di esse, 30 sono classificate come **figurine rare**, e le rimanenti 70 come **figurine normali**.

Il sito offre un servizio che permette ad ogni utente collezionista di effettuare scambi di figurine.

A questo scopo il sistema gestisce un deposito di figurine, nel quale, per ogni diversa figurina vi può essere più di un esemplare.

Il meccanismo di scambio, è regolamentato come segue:

- Si può scambiare solo **una figurina alla volta**;
- **Richiesta di scambio:** ogni utente U che desidera una figurina A può ottenerla, se a sua volta offre un'altra figurina B; in seguito a una richiesta di scambio, il sistema aggiunge la figurina B all'insieme delle figurine disponibili e successivamente verifica se esiste almeno una istanza di A disponibile:
  - se A è disponibile, essa viene assegnata all'utente, che può così continuare la propria attività;
  - se A non è disponibile, l'utente U viene messo in attesa.

Si progetti la politica di gestione del servizio di scambio che tenga conto delle specifiche date e che inoltre soddisfi il seguente vincolo:

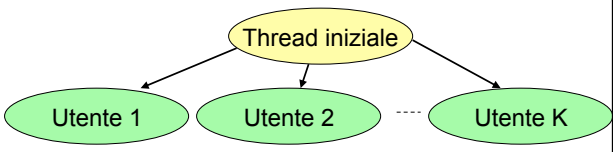
- le richieste di **utenti che offrono figurine rare abbiano la precedenza sulle richieste di utenti che offrono figurine normali**;

Si realizzi un'applicazione nel linguaggio Java che, utilizzando il concetto di monitor, implementi la politica di gestione data.

Spunti e suggerimenti (1)

Quali thread?

- thread iniziale
- K utenti del servizio



Quale risorsa comune?

- **Deposito delle Figurine**

- associamo al Deposito un "*monitor*", che controlla gli accessi in base alla specifica politica di accesso. La sincronizzazione viene realizzata mediante variabili condizione

Spunti e suggerimenti (2)

Struttura dei thread:

```
public class collezionista extends Thread{
   private monitor M;
   private int offerta, richiesta, max;

public collezionista(monitor m, int NF){
       this.M=m;
       this.max=NF;
   }

   public void run(){
   try {       while (true)
               {       <definizione di offerta e richiesta>
                       M.scambio(offerta, richiesta);
                       Thread.sleep(250);
               }
       } catch (InterruptedException e) {       }
   }
}
```


Monitor Deposito Figurine

Stato del Deposito:

- **Figurine disponibili:** vettore di 100 interi (uno per figurina della collezione)

  `private int[] FIGURINE;`

  Dove:

  `FIGURINE[i]` **è il numero di esemplari disponibili della figurina i.**

  Convenzione adottata:

  - **Se i<30**, si tratta di una **figurina rara;**
  - **Se i>= 30**, si tratta di una **figurina comune.**

Monitor Deposito Figurine

- Sincronizzazione:
  - **Lock per la mutua esclusione:**

```
private Lock lock = new ReentrantLock();
```

  - **Condition**. Per la sospensione dei thread in attesa di una figurina, definiamo 2 condition (una per ogni livello di priorità):

```
private Condition rare= …; //coda thread che hanno
offerto figurine rare

private Condition normali= …; //coda thread hanno
offerto figurine normali
```

  - **Contatori dei thread sospesi** in ogni coda:

```
private int[] sospRare;
private int[] sospNormali;
```


Definizione Monitor

```
public class monitor
{ //Dati:
private final int N=100; //numero totale di figurine
private final int maxrare=30;
private int[] FIGURINE; //figurine disponibili
private Lock lock= new ReentrantLock();
private Condition rare= lock.newCondition();
private Condition normali= lock.newCondition();
private int[] sospRare;
private int[] sospNormali;

//Costruttore:
public monitor(int N ) {…}

//metodi "entry":
public void scambio(int off, int rich) throws
   InterruptedException
{…}
}
```