Esercizio del 20 Settembre 2004

Lorenzo Gallucci

Esercizio del 20 Settembre 2004

- ***Esercizio 1***
- Un automobile deve attraversare prima una strada a senso unico alternato, quindi un ponte pericolante.
- Il traffico sul tratto di strada a senso unico alternato è regolato in modo che:
  - Definiamo la direzione *d* con 0 che indica una direzione, 1 la direzione opposta.
  - Un veicolo **V** che deve transitare nella direzione 1 può iniziare il transito solo se non ci sono veicoli in attesa nella direzione 0 e viceversa.
- Per quanto riguarda il ponte, invece, un veicolo può attraversarlo solo se sul ponte ci sono meno di 4 altri veicoli.

Esercizio del 20 Settembre 2004

- Ogni veicolo, pertanto, effettuerà le seguenti operazioni:
  - attende un tempo casuale fra 10 e 30 secondi;
  - cerca di transitare nella direzione 0; se ci sono veicoli che transitano nella direzione opposta (1) aspetta che non transitino più veicoli nella direzione 1;
  - attende un tempo casuale fra 50 e 100 secondi;
  - cerca di attraversare il ponte; se ci sono almeno 4 veicoli che transitano sul ponte aspetta che il numero di veicoli sia minore di 4;
  - aspetta 5 minuti e termina.
- Implementare, utilizzando i thread di Java, la classe *Veicolo*, e le classi necessarie per implementare le operazioni di sincronizzazione, utilizzando i monitor di Java secondo lo schema sopra definito. Scrivere infine una classe con un metodo main che testi il sistema.

Strategia di risoluzione (1/2)

- Un thread si occuperà di gestire del semaforo
  - Ogni minuto dovrà invocare un metodo di inversione del senso di marcia
  - È importante controllare il numero dei veicoli in transito!
- Per ogni veicolo, un thread gestirà l'accesso al senso unico ed il transito

Strategia di risoluzione (2/2)

- Incapsuliamo lo stato del semaforo in una classe "Semaforo" che offra:
  - al gestore del semaforo, un metodo per cambiare il senso di marcia consentito;
  - ai veicoli, un metodo per richiedere il passaggio, nonché un metodo per segnalare l'uscita dal senso unico

- N.B. Non consideriamo *ancora* la seconda parte dell'esercizio (ponte pericolante) !!!

Gestore del semaforo (1/3)

```
package semaforoconlock;

/**
 * Thread che gestisce i periodici cambi
 * di direzione di un semaforo, usato
 * per regolare una strada a senso unico
 * alternato.
 */
public class GestoreSemaforo extends Thread
{
    /**
     * Semaforo da gestire
     */
    Semaforo semaforo;
```

Gestore del semaforo (2/3)

```
/**
    * Costruttore con il semaforo da gestire.
    * @param semaforo semaforo da
  gestire
    */
   public GestoreSemaforo(Semaforo
  semaforo)
   {
       super("Gestore semaforo");
       this.semaforo = semaforo;
   }
```

Gestore del semaforo (3/3)

```
    /**
     * Ogni minuto cerca di cambiare stato al
    semaforo, indefinitamente.
     */
    public void run()
    {
        while (true)
        {
            Utility.attendi("Gestore", 60);
            semaforo.cambiaStato();
        }
    }
}
```

Veicolo (1/3)

```
package semaforoconlock;

public class Veicolo extends Thread
{
    /***
     * la direzione di percorrenza, può valere 0 o 1
     */
    int                 direzione;

    /**
     * il semaforo che controlla la circolazione di questo veicolo
     */
    Semaforo            semaforo;

    /**
     * Nome di questo veicolo
     */
    private final String nome;
```

Veicolo (2/3)

```
/**
     * Costruttore che accetta una direzione ed un semaforo.
     * @param nome nome del veicolo
     * @param direzione direzione di percorrenza
     * @param semaforo semaforo che controlla l'accesso al
    tratto di strada
     * su cui vige il senso unico alternato
     */
    public Veicolo(String nome, int direzione, Semaforo
    semaforo)
    {
        super(nome);
        this.nome = nome;
        this.direzione = direzione;
        this.semaforo = semaforo;
    }
```

Veicolo (3/3)

```
    public void run()
    {
        //percorre un tratto di strada in un tempo casuale
    compreso
        //tra 10 e 30 secondi
        Utility.attendi(nome, 10, 30);
        //arrivato al semaforo deve accedere al senso unico
        semaforo.accediSensoUnico(nome, direzione);
        //percorre il senso unico in un tempo casuale compreso
    tra i 50 e i 100 secondi
        Utility.attendi(nome, 50, 100);
        //esce dal senso unico
        semaforo.esciSensoUnico(nome);
        //percorre un altro tratto di strada in un'ora
        Utility.attendi(nome, 5 * 60);
    }
}
```

Utility (1/3)

```
package semaforoconlock;

/**
 * Classe di utilità contenente vari metodi.
 */
public class Utility
{
    /**
     * Attendi un tempo casuale compreso in un dato intervallo.
     * @param nome nome del componente che richiede l'attesa
     * @param secondiMin tempo minimo (in secondi)
     * @param secondiMax tempo massimo (in secondi)
     */
    public static void attendi(String nome, int secondiMin, int secondiMax)
    {
        int intervalloSecondi = secondiMax - secondiMin;
        long milliSecondi = secondiMin * 1000 + (int) (Math.random() *
    intervalloSecondi * 1000);
        attendiMilliSecondi(nome, milliSecondi);
    }
```

Utility (2/3)

```
/**
 * Attendi un tempo dato, specificato in secondi.
 * @param nome nome del componente che richiede l'attesa
 * @param secondi tempo di attesa (in secondi)
 */
public static void attendi(String nome, int secondi)
{
    attendiMilliSecondi(nome, secondi * 1000);
}
```

Utility (3/3)

```
/**
    * Attendi un tempo dato, specificato in millisecondi.
    * @param nome nome del componente che richiede l'attesa
    * @param milliSecondi tempo di attesa (in millisecondi)
    */
   public static void attendiMilliSecondi(String nome, long milliSecondi)
   {
      try
      {
         System.out.println(nome + " attende " + (milliSecondi / 1000) + " secondi");
         Thread.sleep(milliSecondi / 10);
         System.out.println(nome + " ha terminato un'attesa di " + (milliSecondi / 1000)
    + " secondi");
      }
      catch (InterruptedException e)
      {
         System.out.println("Impossibile attendere per " + (milliSecondi / 1000) + "
    secondi");
         e.printStackTrace();
      }
   }
}
```

Semaforo (1/9)

```
package semaforoconlock;

import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

/**
 * Classe di controllo del semaforo che regola l'accesso al senso unico.
 */
public class Semaforo
{
   /**
    * stato del semaforo: può valere:
    * <ul>
    * <li><code>0</code> se il semaforo è verde dalla prima direzione, rosso
    dall'altra</li>
    * <li><code>1</code> se il semaforo è verde dalla secondo direzione, rosso
    dall'altra</li>
    * <li><code>2</code> se il semaforo è rosso da ambo i lati</li>
    * </ul>
    */
   int      statoCorrente;
```

Semaforo (2/9)

```
/**
     * il numero di auto in transito sul senso
   unico
     */
    int       veicoliInTransito;

    /**
     * Lock che controlla l'accesso alle
     * strutture dati del semaforo
     */
    Lock      lock;
```

Semaforo (3/9)

```
/**
     * Condizione che indica il disimpegno
     * del tratto di strada da parte di uno
     * o più dei veicoli che la occupavano.
     */
    Condition stradaLibera;

    /**
     * Condizione che indica il passaggio da
     * rosso a verde del semaforo che regola
     * l'accesso da uno dei due lati.
     */
    Condition semaforoVerde;
```

Semaforo (4/9)

```
/**
 * Costruttore che inizializza il semaforo in un
 * dato stato, pone il numero di veicoli in transito
 * pari a zero ed inizializza un lock.
 * @param statoIniziale stato iniziale (può valere 0, 1 o 2).
 */
public Semaforo(int statoIniziale)
{
    statoCorrente = statoIniziale;
    veicoliInTransito = 0;
    lock = new ReentrantLock(true);
    stradaLibera = lock.newCondition();
    semaforoVerde = lock.newCondition();
}
```

Semaforo (5/9)

```
/**
     * il medoto seguente è invocato dal gestore del semaforo
     * quando lo statoCorrente vale 0 o 1
     */
    public void cambiaStato()
    {
        //se lo stato corrente vale 0, il prossimo stato sarà 1
        //altrimenti sarà 0
        int prossimoStato = 1 - statoCorrente;

        lock.lock();
        try
        {
            while (veicoliInTransito > 0)
            {
                //non posso fare passare le auto in senso opposto
                //quindi il semaforo diventa rosso per tutti
                statoCorrente = 2;
                //e il gestore aspetta finché non escono tutte le auto in transito
```

Semaforo (6/9)

```
            try
             {
                System.out.println("Il gestore attende che il tratto di strada si liberi");
                stradaLibera.await();
             }
             catch (InterruptedException e)
             {
                System.out.println("Impossibile attendere");
                e.printStackTrace();
                throw new RuntimeException(e);
             }
          }
          System.out.println("Il semaforo diventa verde dalla direzione " +
    prossimoStato);
          statoCorrente = prossimoStato;
          semaforoVerde.signalAll();
       }
       finally
       {
          lock.unlock();
       }
    }
```

Semaforo (7/9)

```
/**
 * metodo invocato da un veicolo che deve attraversare
 * il senso unico in direzione d, che può valere 0 o 1
 * @param nome nome del componente che vuole accedere al
senso unico
 * @param d direzione di accesso desiderata
 */
public void accediSensoUnico(String nome, int d)
{
    lock.lock();
    try
    {
        while (d != statoCorrente)
        {
            System.out.println(nome + " attende che il semaforo
diventi verde nella direzione " + d);
```

Semaforo (8/9)

```
            try
            {
                semaforoVerde.await();
            }
            catch (InterruptedException e)
            {
                System.out.println("Impossibile attendere");
                e.printStackTrace();
                throw new RuntimeException(e);
            }
        }
        System.out.println(nome + " accede alla strada dalla direzione " + d);
        veicoliInTransito++;
    }
    finally
    {
        lock.unlock();
    }
}
```

Semaforo (9/9)

```
    /**
     *    metodo invocato da un
     *    veicolo che deve uscire dal senso unico
     * @param nome
     */
    public void esciSensoUnico(String nome)
    {
        lock.lock();
        try
        {
            System.out.println(nome + " libera la strada");
            veicoliInTransito--;
            stradaLibera.signalAll();
        }
        finally
        {
            lock.unlock();
        }
    }
}
```

EsempioSemaforo (1/2)

```
package semaforoconlock;

public class EsempioSemaforo
{
    public static void main(String[] args)
    {
        Semaforo semaforo = new Semaforo(0);
        GestoreSemaforo gestoreSemaforo = new
     GestoreSemaforo(semaforo);
        gestoreSemaforo.setDaemon(true);
        gestoreSemaforo.start();
        int numVeicoli = 10;
        Veicolo[] veicoli = new Veicolo[numVeicoli];
        // Inizializza veicoli
        for (int i = 0; i < numVeicoli; i++)
        {
            veicoli[i] = new Veicolo("Veicolo #" + (i + 1), i % 2, semaforo);
            veicoli[i].start();
        }
```

EsempioSemaforo (2/2)

```
    // Attendi la conclusione del ciclo di tutti i veicoli
        try
        {
            for (int i = 0; i < numVeicoli; i++)
            {
                veicoli[i].join();
            }
        }
        catch (InterruptedException e)
        {
            System.out.println("Impossibile attendere");
            e.printStackTrace();
            throw new RuntimeException(e);
        }
    }

}
```