SAPIENZA Università di Roma
Facoltà di Ingegneria dell'Informazione

Esercitazioni di
PROGETTAZIONE DEL SOFTWARE
(Corso di Laurea in Ingegneria Informatica)
A.A. 2009-10

Esercitazione sulla programmazione concorrente e sull'uso dei thread in Java

Esercizio 1

Realizzare in Java un'applicazione che simuli una gara di corsa tra un numero n (fornito in input) di corridori, ciascuno caratterizzato dal proprio nome. I corridori sono modellati da thread Java. Ognuno di essi, dopo l'avvio, incrementa iterativamente un contatore, stampando su schermo, ad ogni iterazione, il proprio nome ed il numero di iterazioni eseguite (cioè i metri percorsi). Dopo aver eseguito 100 iterazioni, il thread stampa su schermo un messaggio di notifica del proprio arrivo.

1. Dopo aver eseguito il programma Java, analizzarne l'output e stabilire se l'esecuzione dei vari thread sia avvenuta in maniera sequenziale (ovvero viene completato il primo thread avviato, poi il secondo, e così via) o meno;

2. Se dall'output emerge un comportamento sequenziale, individuarne le cause e proporre una modifica al programma che le rimuova;

3. Dopo aver completato i punti precedenti, analizzare la porzione di codice in cui i thread vengono avviati e verificare se si possa verificare la condizione in cui *alcuni thread sono all'interno del ciclo di conteggio mentre altri devono essere ancora avviati (o istanziati)*. In caso di risposta affermativa, proporre una modifica al programma che escluda tale eventualità.

Esercizio 2

Si consideri l'esercizio precedente. Di esso si vuole creare una versione modificata, in cui, oltre ai corridori, è presente un giudice, anch'esso rappresentato da un thread. La gara si svolge come segue:

- i corridori vengono avviati (tenere presenti le osservazioni dell'esercizio precedente);
- il giudice viene avviato, dopodiché rimane in attesa che il primo corridore raggiunga il traguardo;
- il primo corridore che raggiunge il traguardo memorizza un riferimento all'oggetto che lo rappresenta in un'opportuna struttura ordinata (che rappresenterà la classifica);
- quando il primo corridore giunge al traguardo, il giudice ne annuncia la vittoria stampando un messaggio su schermo, quindi termina;
- ogni altro corridore, quando arriva al traguardo, memorizza il proprio arrivo, accodando un riferimento all'oggetto che lo rappresenta nella struttura ordinata;
- dopo che *tutti* i corridori hanno terminato la propria esecuzione, la classifica di arrivo viene stampata su schermo.

Soluzioni

Esercizio 1 -versione base

La classe Java Application

```
package esercizio1a;

import java.lang.Thread;

public class Application
{
        /**
         * @param args
         */
        public static void main(String[] args)
        {
                for(int i=0;i<30;i++){
                        Thread t = new Thread(new Corridore("corridore " + i));
                        t.start();
                }
        }
}
```

Esercizio 1 - versione base (cont.)

La classe Java Corridore

```
package esercizio1a;
public class Corridore implements Runnable
{
        private final String nome;
        private final static int METRI=100;

        public Corridore(String nome)
        {
                this.nome=nome;
        }

        public void run()
        {
                int spazio=0;
                try
                {
                while(spazio<METRI)
                {
                        spazio++;
                        Thread.sleep(10);
                        System.out.println(nome+" "+spazio+" metri");
                }
                }
                catch(InterruptedException err)
                {
```

```
                        err.printStackTrace();
                }
                System.out.println("Il corridore "+nome+" è arrivato");
        }


        public String getNome() {
                return nome;
        }

}
```

Esercizio 1 - versione estesa

La classe Java Application

```
package esercizio1bang;

import java.lang.Thread;

public class Application{

        /**
         * @param args
         */

        public static void main(String[] args){
                final int N = 30;
                ContatoreSincronizzato contatore = new ContatoreSincronizzato(N);// memorizza il numero di cor
                Bang bang = new Bang();
                try{
                        for(int i=0;i<N;i++){// fa partire tutti i corridori
                                Thread t = new Thread(new Corridore("corridore " + i,contatore, bang));
                                t.start();
                        }

                        // Attende che tutti i corridori siano pronti (cioe' ciascuno e' all'inizio della prop
                        contatore.attendiCompletamento();

                        System.out.println("\n----- Corridori pronti a partire -----\n");

                        // Notifica a tutti i corridori che possono partire
```

```
                        System.out.println("\n----- BANG! -----\n");
                        Thread.sleep(300);// Solo per dare il tempo di leggere l'output
                        bang.spara();
                }
                catch (InterruptedException e){
                        e.printStackTrace();
                }

        }
}
```

Esercizio 1 - versione estesa (cont.)

La classe Java Bang

```
package esercizio1bang;

public class Bang {
        private boolean sparato = false;

        public synchronized void spara(){
                sparato = true;
                notifyAll();
        }

        public synchronized void attendiBang(){
                // ATTENZIONE: se sparato = true, non devo eseguire la wait (la corsa e' gia' iniziata)
                while(!sparato){
                        try{
                                wait();
                        }
                        catch(InterruptedException e){
                                e.printStackTrace();
                        }
                }
        }
}
```

Esercizio 1 - versione estesa (cont.)

La classe Java ContatoreSincronizzato

```
package esercizio1bang;

public class ContatoreSincronizzato {
        private final int max;
        private int valoreCorrente = 0;

        public ContatoreSincronizzato(int max){
                this.max = max;
        }

        public synchronized void incrementa(){
                if (valoreCorrente < max){
                        valoreCorrente++;
                }
                notifyAll();
        }

        public synchronized void attendiCompletamento(){
                while(valoreCorrente < max){
                        // Aspetta che il valore corrente raggiunga il massimo
                        try{
                                wait();
                        }
                        catch(InterruptedException e){
                                e.printStackTrace();
```

```
                        }
                    }
                }
            }
```

Esercizio 1 - versione estesa (cont.)

La classe Java Corridore

```
package esercizio1bang;

public class Corridore implements Runnable{

        private final String nome;
        private final static int METRI=100;
        private ContatoreSincronizzato pronti;
        private final Bang bang;

        public Corridore(String nome, ContatoreSincronizzato pronti, Bang bang){
                this.nome = nome;
                this.bang = bang;
                this.pronti = pronti;
        }

        public void run(){
                int spazio=0;
                try
                {
                        while(spazio<METRI){

                                if(spazio == 0){// E' la prima iterazione
                                        System.out.println(nome+" pronto a partire");
                                        pronti.incrementa();
                                        bang.attendiBang();
```

```
                        }

                        spazio++;
                        Thread.sleep(1);
                        System.out.println(nome+" "+spazio+" metri");
                }
                System.out.println("Il corridore " + nome + " e' arrivato");
        }
        catch(InterruptedException err){
                err.printStackTrace();
        }

}


public String getNome() {
        return nome;
}

}
```

Esercizio 2

La classe Java Application

```
package esercizio2;

public class Application{
        /**
         * @param args
         */

        public static void main(String[] args){
                int numero=30;
                Thread[] threadCorridori = new Thread[numero];
                Corsa corsa=new Corsa();

                for(int i=0;i<numero;i++){
                        // crea un thread per ciascun corridore e lo avvia
                        // i thread vengono memorizzati in un vettore per la sincronizzazione futura
                        threadCorridori[i] = new Thread(new Corridore(corsa,i+""));
                        threadCorridori[i].start();
                }

                // Crea un nuovo thread per il giudice e lo fa partire.
                // Il thread "main" si ferma in attesa che il giudice abbia terminato l'esecuzione.
                Thread giudice = new Thread(new Giudice(corsa));
                giudice.start();
                try{
                        // main attende che il giudice abbia dichiarato il vincitore
                        giudice.join();
```

```
                // main attende che tutti i corridori abbiano terminato la corsa
                for (int i = 0; i< numero; i++){
                        threadCorridori[i].join();
                }
            }
            catch(InterruptedException e){
                e.printStackTrace();
            }

            System.out.println(corsa);// Stampa la classifica
        }
}
```

Esercizio 2

La classe Java Corridore

```
package esercizio2;

public class Corridore implements Runnable
{
        private final static int METRI=100;
        private final String nome;
        private final Corsa corsa;
        private int spazio;

        public Corridore(Corsa unaCorsa,String nome)
        {
                this.corsa=unaCorsa;
                this.nome=nome;
                this.spazio=0;
        }

        public void run()
        {
                try
                {
                        while(spazio<METRI)
                        {
                                spazio++;
                                Thread.sleep(5);
                                System.out.println(nome+ " ha percorso " + spazio + " metri");
                        }
```

```
                }
                catch(InterruptedException err)
                {
                        err.printStackTrace();
                }
                corsa.setArrivato(this);
        }


        public String getNome() {
                return nome;
        }

}
```

Esercizio 2

La classe Java Corsa

```
package esercizio2;

import java.util.*;

public class Corsa
{
        Vector<Corridore> classifica = new Vector<Corridore>();

        public synchronized String getVincitore()
        {
                if(classifica.isEmpty())
                        return null;
                return classifica.get(0).getNome();
        }

        public synchronized void setArrivato(Corridore corridore){
                classifica.add(corridore);
                notifyAll();
        }

        public synchronized String toString(){
                String result ="";
                for(int i = 0; i < classifica.size(); i++){
                        Corridore c = classifica.get(i);
                        result += (i+1) + ": "+c.getNome()+"\n";
                }
```

```
                return result;
        }
}
```

Esercizio 2

La classe Java Giudice

```
package esercizio2;

public class Giudice implements Runnable
{
        private Corsa corsa;

        public Giudice(Corsa unaCorsa)
        {
                this.corsa=unaCorsa;
        }

        public void run()
        {
                synchronized (corsa) {
                        try {
                                while(corsa.getVincitore() == null){
                                        corsa.wait();// rimane in attesa finche' non c'e' un vincitore
                                }
                                // ora c'e' un vincitore
                                System.out.println("_____________________ Ha vinto "+corsa.getVincitore() + " -
                        } catch (InterruptedException e) {
                                e.printStackTrace();
                                return;
                        }
                }
        }
}
```