Nona esercitazione

Java Thread

Stefano Monti
stefano.monti6@unibo.it

Esercizio precedente

Si consideri un lavaggio automatico di veicoli a cui possono accedere auto e moto. Ciascun veicolo può entrare nel lavaggio auto e richiedere il lavaggio specifico per il tipo di veicolo.

Facendo riferimento al meccanismo di creazione dei thread in Java si realizzi un'applicazione che crei i thread, rappresentativi dei veicoli.

In particolare, l'applicazione dovrà creare

- thread rappresentativi delle moto e
- thread rappresentativi di auto
  - auto piccole
  - auto grandi

  che ereditino le proprietà dalla classe auto!

L'applicazione permetterà a ciascun veicolo di scrivere a video il tipo di lavaggio richiesto.

Impostazione

Come creare i thread?

Automobili: auto grandi e piccole come sottoclassi di auto:
- È necessario usare l'interfaccia **Runnable**

Moto: non ci sono vincoli
- È possibile usare entrambi i metodi

Quante classi?

- Lavaggio: contiene il main
- Auto: definisce le caratteristiche delle auto in generale (marca, modello, targa, cilindrata,….)
- Autograndi: eredita da Auto ed esibisce un comportamento specifico (messaggio da stampare)
- Autopiccole: eredita da Auto ed esibisce un comportamento specifico (messaggio da stampare)
- Moto

Moto

```
public class Moto extends Thread{
    private String modello;
    public Moto(String modello){
        this.modello=modello;
    }
    public String getModello(){
        return modello;
    }
    public void run(){
        System.out.println("Veicolo "+getModello()+" ha
          richiesto il lavaggio \"moto\"");
    }
}
```

Auto

```
public class Auto {
    private String modello;
    public Auto(String modello)
      {this.modello=modello;}
    public String getModello(){return modello;}
}
```

```
public class AutoGrande
    extends Auto implements
      Runnable{
    public AutoGrande(String
      modello){
        super(modello);
    }
    public void run(){
        System.out.println("Veicolo
          "+getModello()+" ha
          richiesto il
          lavaggio \"auto grande\"");
}}
```

```
public class AutoPiccola
    extends Auto implements
      Runnable{
    public AutoPiccola(String
      modello){
        super(modello);
    }
    public void run(){
        System.out.println("Veicolo
          "+getModello()+" ha
          richiesto il
          lavaggio \"auto
          piccola\"");
}}
```

Lavaggio

```
public class Lavaggio{
    public static void main (String[] args){
        AutoGrande ag1 = new AutoGrande("Auto grande 1");
        Thread t_ag1 = new Thread(ag1);

        AutoPiccola ap1 = new AutoPiccola("Auto piccola 1");
        Thread t_ap1 = new Thread(ap1);

        AutoPiccola ap2 = new AutoPiccola("Auto piccola 2");
        Thread t_ap2 = new Thread(ap2);

        Moto m1 = new Moto("Moto 1");
        Moto m2 = new Moto("Moto 2");

        m1.start(); t_ag1.start();
        t_ap2.start(); m2.start();
        t_ap1.start(); } }
```

Esercizio 1

Si realizzi un programma Java per la gestione e la elaborazione di stringhe.

In particolare, il programma dovrà

- ricevere e memorizzare stringhe in un buffer di **dim. finita N** da un **thread produttore**
- applicare **in sequenza** 3 operazioni (delegate ad altrettanti thread *operatori*) al **primo elemento del buffer**
  - troncamento della seconda metà della stringa
  - *append* di una stringa relativa alla data corrente
  - stampa a video e prelievo dal buffer

Note alla soluzione

Vincoli di sincronizzazione

- accesso al buffer in mutua esclusione
  - sia per aggiunta stringhe
  - sia per operazioni di troncamento/append/prelievo
- problema di sincronizzazione:
  - produttore non deve scrivere se il buffer è pieno
  - operatori non possono agire su buffer vuoto
- vincolo temporale
  - necessità di ordinare le operazioni dei tre operatori

Uso dei semafori

- accesso al buffer in mutua esclusione
  - utilizzo di un semaforo binario
- problema di sincronizzazione:
  - utilizzo di un semaforo di dimensione N
- vincolo temporale
  - utilizzo di 3 semafori
    - uno per ciascuna operazione
    - inizialmente a 0 → il termine della operazione precedente sblocca quella seguente


```
public class Risorsa {
    final int capienzaMaxBufferStringhe = 10; // N
    // --- Semafori ------------
    Semaforo s_aggiuntaStringa,
          s_operazione1, s_operazione2, s_operazione3,
          sM;
    java.util.List bufferStringhe; //Buffer
    int numeroStringhe = 0;// Contatore stringhe presenti nel buffer
    int stringheElaborate = 0;

    public Risorsa (){                                    semaforo Buffer
        s_aggiuntaStringa = new Semaforo(capienzaMaxBufferStringhe);
        s_operazione1 = new Semaforo(0);
        s_operazione2 = new Semaforo(0);          semafori ordinamento
        s_operazione3 = new Semaforo(0);
        sM = new Semaforo(1);              semaforo mutua esclusione
        bufferStringhe= new java.util.Vector();
    }

    public void nuovaStringa(String string){ }
    public void operazione1(){ }
    public void operazione2(){ }
    public void operazione3(){ }
```

```
public class Risorsa {
//...
public void nuovaStringa(String string){
      s_aggiuntaStringa.p();
      sM.p();
      bufferStringhe.add(string);
      numeroStringhe ++;
      if (numeroStringhe == 1)
            s_operazione1.v();
      sM.v();
}

public void operazione1(){
      s_operazione1.p();
      sM.p();
      String daElaborare = (String) bufferStringhe.get(0);
      daElaborare = daElaborare.substring(0,daElaborare.length()/2);
      bufferStringhe.set(0,daElaborare);
      s_operazione2.v();
      sM.v();
}
```

ATTENZIONE: è necessaria la condizione *if*? Cosa succederebbe se la togliessi?


```
public class Risorsa {
...
public void operazione2(){
      s_operazione2.p();
      sM.p();
      String daElaborare = (String) bufferStringhe.get(0);
      daElaborare = daElaborare + (new java.util.Date()).toString();
      bufferStringhe.set(0,daElaborare);
      s_operazione3.v();
      sM.v();
}

public void operazione3(){
      s_operazione3.p();
      sM.p();
      stringheElaborate++;
      bufferStringhe.remove(0);
      numeroStringhe--;
      if (numeroStringhe > 0)
            s_operazione1.v();
      s_aggiuntaStringa.v();
      sM.v();
}
```

Esercizio 2

Si realizzi un programma Java che simuli la gestione di una cucina di un ristorante.

In particolar modo:

- due thread AiutoCuoco si incaricano di portare sul tavolo di preparazione (risorsa condivisa) gli ingredienti necessari (di due tipi diversi)

- un thread Chef sovrintende alla preparazione del piatto,
  - attende che siano disponibili le quantità necessarie di ingredienti
  - quando disponibili, le preleva dal tavolo e prepara il piatto

Esercizio 2- dettagli

- Tavolo ha capacità limitata (diversa) per ingrediente1 e ingrediente2 (risp. M1 e M2)

- Ciascun AiutoCuoco porta sempre una unità di ingrediente

- Per preparare il piatto, Chef ha bisogno di quantità prestabilite per i 2 ingredienti
  - Q1 per ingrediente1
  - Q2 per ingrediente2

Esercizio 2 - Sincronizzazione

- Tavolo è una risorsa condivisa con due buffer, uno per ciascun ingrediente
  - gestione della capacità: non è possibile altre unità di ingredienteX se il tavolo ha già saturato la capacità per X
- necessità di mutua esclusione tra thread che accedono al tavolo
- thread Chef deve attendere che siano disponibili gli ingredienti, e soltanto allora, iniziare la preparazione del piatto (ordinamento)


```
public class semaforo {
    private int value;
    public semaforo (int initial){
        this.value = initial;
    }
    synchronized public void v(){
         ++value;
        notify();
    }
    synchronized public void p() throws InterruptedException
    { while(value == 0) {
        try { wait();
        } catch (InterruptedException e) {}
    }
        value--;
    }}
```