Esercizio 6.3

Impostazione:

1. Quali processi?
   PersonaInEst
   PersonaInOvest

2. Quale struttura per i processi
   PersonaInEst:
```
public void run()
{
     people.incrementDaEst();  /* il thread PersonaInEst entra da est */
     people.decrementDaEst();   /* il thread PersonaInEst esce da est */
}
```
   PersonaInOvest:
```
public void run()
{
     people.incrementDaOvest();  /* il thread PersonaInOvest entra da ovest */
     people.decrementDaEst();       /* il thread PersonaInOvest esce da est */
}
```

3. Definizione della classe monitor
   Dati:
   Contatore
   Oppure
   ContatoreEntraEsci
   Oppure
   ContatoreEquoEntraEsci

   Operazioni:
   `synchronized void` **`incrementDaEst`**`()` /* metodo eseguito dal thread persona per entrare da est nel giardino*/
   `synchronized void` **`incrementDaOvest`**`()` /* metodo eseguito dal thread persona per entrare da ovest nel giardino*/
   `synchronized void` **`decrementDaEst`**`()` /* metodo eseguito dal thread persona per uscire dal giardino*/

Soluzione:

```
/*Argomenti del main:
  args[0] = "equo"  se si vuole implementare la soluzione equa per  gli ingressi e per le uscite
  args[0] = "variante"  se si vuole implementare la soluzione che privilegia le entrate
  args[0] =  qualsiasi altra stringa se si vuole implementare la soluzione di sola sincronizzazione.
 */

public class GiardinoMain {

/* Il main è la classe di lancio del programma */
   public static void main (String args[]){
       Contatore cont;
       String ifVariante = args[0];
       if (ifVariante.equals("variante"))
```

```
      {
/* Si sceglie ContatoreEntraEsci che privilegia le entrate rispetto alle uscite */
        cont= new ContatoreEntraEsci();
      }
      else
      {
        if (ifVariante.equals("equo"))
        {
/* Si sceglie ContatoreEquoEntraEsci che implementa una soluzione equa per gli ingressi e per le uscite */
            cont= new ContatoreEquoEntraEsci();
        }
        else
        {
            System.out.println("Contatore");
            cont= new Contatore();
        }
      }
      int numInCoda = 6;
      PersonaInEst[] est = new PersonaInEst[numInCoda];
      PersonaInOvest[] ovest = new PersonaInOvest[numInCoda];

      for (int i=0; i<numInCoda; i++)
      {
        est[i] = new PersonaInEst(cont);
        ovest[i] = new PersonaInOvest(cont);
      }

      for (int i=0; i<numInCoda; i++)
      {
        est[i].start();
        ovest[i].start();
      }
  }
}

class PersonaInEst extends Thread {

  Contatore people;
  PersonaInEst(Contatore c)
  {
      people = c;
   } /* costruttore*/

  public void run()
  {
      try
      {
        while (true)
        {
          Thread.sleep(500);
          people.incrementDaEst();
/* Poiché realisticamente quando una persona entra, prima o poi esce, è possibile una soluzione che preveda
   l'invocazione del metodo decrementDaEst dopo l'invocazione del metodo incrementDaEst, all'interno di
   questo stesso thread.
   Si ipotizza che le persone possano uscire solo dal cancello Est */
```

```
          people.decrementDaEst();
        }
      }
      catch (InterruptedException e) {}
      }
}

class PersonaInOvest extends Thread {

  Contatore people;
  PersonaInOvest(Contatore c) {

      people = c;
  } //costruttore

  public void run()
  {
      try
      {
        while (true)
        {
           Thread.sleep(500);
           people.incrementDaOvest();
/* Poiché realisticamente quando una persona entra, prima o poi esce, è possibile una soluzione che preveda
   l'invocazione del metodo decrementDaEst dopo l'invocazione del metodo incrementDaEst.
   Si ipotizza che le persone possano uscire solo dal cancello Est */
          people.decrementDaEst();
  }
      }
      catch (InterruptedException e) {}
  }
}

/* Contatore è la classe monitor, che rappresenta la risorsa condivisa da sincronizzare.
```

La versione seguente tiene conto solamente del numero di utenti entrati nel giardino. La classe Contatore non opera alcuna sincronizzazione (come richiesto dal testo dell'esercizio) */

```
class Contatore {

  private int numVisitatori = 0;
  private int numEst = 0;
  private int numOvest = 0;

/* Il metodo è dichiarato synchronized perché richiede l'accesso mutuamente esclusivo alle variabili condivise
   del monitor: il contatore. */
   synchronized void incrementDaEst() throws InterruptedException
   {
      int temp = numVisitatori;
      numVisitatori=temp+1; /* modifica del valore del contatore condiviso che misura gli utenti
              nel parco; */
   }

  /* Il metodo è dichiarato synchronized perché richiede l'accesso mutuamente esclusivo alle variabili condivise
     del monitor: il contatore. */
   synchronized void incrementDaOvest() throws InterruptedException
```

```
  {
      System.out.println("Sono entrato da ovest.");
      int temp = numVisitatori;
      numVisitatori=temp+1; /* modifica del valore del contatore condiviso che misura gli utenti
            nel parco; */
  }

  synchronized void decrementDaEst() throws InterruptedException{}
}
```

/* Nella soluzione seguente, la classe ContatoreEntraEsci, sincronizza le entrate e le uscite. In questa soluzione, però, i thread in uscita potrebbero andare in starvation. Infatti, finché esiste un thread che in entrata, quelli in uscita rimangono sospesi */

```
class ContatoreEntraEsci extends Contatore
{
  public final static int capacityPark = 40;

  private int numVisitatori = 0;
  private int numWaitIn = 0;
```

/* Il metodo è dichiarato synchronized perché richiede l'accesso mutuamente esclusivo alle variabili condivise del monitor: il contatore. Inoltre contiene sia l'istruzione wait sia notifyAll che devono essere sempre inserite all'interno di una sezione synchronized */

```
  synchronized void incrementDaEst() throws InterruptedException
  {
    ++numWaitIn;
```

/* La seguente istruzione verifica le condizioni di sospensione delle persone in entrata al giardino(thread) */

```
    while (numVisitatori == capacityPark)
      {System.out.println("Aspetto a est");
```

/* wait è l'istruzione che sospende il thread (persona) che la invoca e libera il lock associato all'istanza del monitor. */

```
       wait();}
    --numWaitIn;
    int temp = numVisitatori;
    numVisitatori=temp+1;
    System.out.println("Sono entrato da est");
    System.out.println("Numero di visitatori nel parco: "+
                                                    numVisitatori);
```

/* notifyAll è l'istruzione che risveglia tutte le altre persone (thread) in attesa di poter entrare o uscire dal giardino */

```
    notifyAll();
  }

  synchronized void incrementDaOvest() throws InterruptedException
  {
    ++numWaitIn;
    while (numVisitatori == capacityPark)
      {System.out.println("Aspetto a ovest");wait();}
    --numWaitIn;
    int temp = numVisitatori;
    numVisitatori=temp+1;
    System.out.println("Sono entrato da ovest");
    System.out.println("Numero di visitatori nel parco: "+
                                                  numVisitatori);
```

```
    notifyAll();
  }

  synchronized void decrementDaEst() throws InterruptedException
  {
    while ((numVisitatori == 0) || (numWaitIn>0))
      {System.out.println("Aspetto di uscire a est");wait();}
    --numVisitatori;
    System.out.println("Sono uscito da est");
    System.out.println("Numero di visitatori nel parco: "+
                                                    numVisitatori);
    notifyAll();
  }
}
```

/* Nella soluzione seguente, la classe ContatoreEquoEntraEsci, sincronizza le entrate e le uscite. Questa soluzione la soluzione equa per gli ingressi e per le uscite */

```
class ContatoreEquoEntraEsci extends Contatore
{
  public final static int capacityPark = 40;
  private int numVisitatori = 0;
  private int numWaitIn = 0;
  private int numWaitOut = 0;
  boolean exitTurn = false;

  synchronized void incrementDaEst() throws InterruptedException
  {
    ++numWaitIn;
    while ((numVisitatori == capacityPark) || (numWaitOut>0 &&
exitTurn)){System.out.println("Aspetto di entrare a est");wait();}
    --numWaitIn;
    int temp = numVisitatori;
    numVisitatori=temp+1;
    System.out.println("Sono entrato da est");
    System.out.println("Numero di visitatori nel parco: "+
                                                    numVisitatori);
    exitTurn = true;
    notifyAll();
  }

  synchronized void incrementDaOvest() throws InterruptedException
  {
    ++numWaitIn;
    while ((numVisitatori==capacityPark)||(numWaitOut>0&& exitTurn))
      {System.out.println("Aspetto di entrare a ovest");wait();}
    --numWaitIn;
    int temp = numVisitatori;
    numVisitatori=temp+1;
    System.out.println("Sono entrato da ovest");
    System.out.println("Numero di visitatori nel parco: "+
                                                    numVisitatori);
    exitTurn = true;
    notifyAll();
   }
```

```
   synchronized void decrementDaEst() throws InterruptedException
   {
     ++numWaitOut;
     while ((numVisitatori == 0) || (numWaitIn>0 && !exitTurn))
      {System.out.println("Aspetto di uscire a est");wait();}
     --numWaitOut;
     --numVisitatori;
     exitTurn = false;
     System.out.println("Sono uscito da est");
     System.out.println("Numero di visitatori nel parco: "+
                                                    numVisitatori);
     notifyAll();
  }
}
```