Esercizio 6.2

Impostazione:

1. Quali processi?
   FilosofiFree

2. Quale struttura per i processi
   FilosofiFree:

```
 public void run()
   {
/*I filosofi di posto pari prelevano prima la forchetta di sinistra, i filosofi di posto dispari, quella di destra */
  if (identity%2 == 0) {left.get();}
  else {right.get();}
/*I filosofi di posto pari prelevano poi la forchetta di destra, i filosofi di posto dispari, quella di sinistra */
  if (identity%2 == 0) {right.get();}
  else{left.get();}
/* tempo per mangiare*/
/*Tutti i filosofi rilasciano prima la forchetta di destra e poi quella di sinistra */
   right.put();
   left.put();
}
```

3. Definizione della classe monitor
   Dati:
   `Fork(int id){ identity = id;}` /* L'id della forchetta ne identifica la posizione */

   Operazioni:
   `synchronized void get()` /*metodo eseguito dal thread FilosofiFree per richiedere una forchetta */
   `synchronized void put()` /*metodo eseguito dal thread FilosofiFree per rilasciare una forchetta */

Soluzione:

```
public class FilosofiMain{
```



```
/*Il main è la classe di lancio del programma */
  public static void main (String args[]){     int numFil = 5;
      Fork[] forchetta = new Fork[numFil];
      Thread[] filosofo = new Thread[numFil];
      for (int i=0; i<numFil; i++)
      {
        forchetta[i] = new Fork(i);
      }
      for (int k=0; k<numFil; k++)
      {
        filosofo[k] = new FilosofiFree(k,forchetta[(k-
                                        numFil)%numFil],forchetta[k]);

/* Tramite l'invocazione del metodo start, viene messo in esecuzione il thread corrispondente a ciascun filosofo.
*/
        filosofo[k].start();
      }
  }
}
```

```
/* La classe FilosofiFree è il thread che rappresenta il filosofo libero da deadlock */
class FilosofiFree extends Thread {
  int identity;
  Fork left;
  Fork right;

  FilosofiFree(int identity, Fork left, Fork right) {

    this.identity = identity;
    this.left = left;
    this.right = right;
  }

/* Il metodo run contiene il codice eseguito dal thread quando questo viene messo in esecuzione. Il metodo run
   viene invocato automaticamente all'esecuzione del metodo start.      */
  public void run()
  {
    while (true)
    {
      try
      {

/* Per evitare deadlock, è necessario simulare un comportamento differente dei filosofi numerati come "pari" e
   dei filosofi numerati come "dispari".
In particolare: i filosofi "pari" cercheranno di ottenere prima la forchetta di sinistra e poi quella di destra, mentre i filosofi "dispari" cercheranno di ottenere prima la forchetta di destra e poi quella di sinistra.
   Tutti i filosofi rilasceranno prima la forchetta di destra e poi quella di sinistra.*/
        if (identity%2 == 0)
        {
          System.out.println("Sono il filosofo "+identity+" e sto
                              aspettando una forchetta a sinistra.");
          left.get();
        }
        else
        {
          System.out.println("Sono il filosofo "+identity+" e sto
                              aspettando una forchetta a destra.");
          right.get();
        }
        sleep(500);
        if (identity%2 == 0)
        {
          System.out.println("Sono il filosofo "+identity+" e sto
                              aspettando una forchetta a destra.");
          right.get();
        }
        else
        {
          System.out.println("Sono il filosofo "+identity+" e sto
                              aspettando una forchetta a sinistra.");
          left.get();
        }
        System.out.println("Sono il filosofo "+identity+" e sto
                                                    mangiando.");
```

```
        sleep((int)(50*Math.random()));
        System.out.println("Sono il filosofo "+identity+" e rilascio
                                          la forchetta di destra.");
        right.put();
        System.out.println("Sono il filosofo "+identity+" e rilascio
                                        la forchetta di sinistra.");
        left.put();
      } catch (java.lang.InterruptedException e) {}
    }
  }
}
```

/* Fork è la classe monitor, che rappresenta la risorsa condivisa da sincronizzare */

```
public class Fork {
  private boolean taken=false;
  private int identity;
```

/* Costruttore per inizializzare la variabile che identifica l'istanza della forchetta */

```
  Fork(int id){ identity = id;}
```

/* Il metodo è dichiarato synchronized perché richiede l'accesso mutuamente esclusivo alle variabili condivise del monitor. Inoltre contiene l'istruzione wait che deve essere sempre inserita all'interno di una sezione synchronized */

```
  synchronized void get() throws java.lang.InterruptedException
  {
```

/* La seguente istruzione verifica le condizioni di sospensione per i filosofi (thread). Se la forchetta è già stata presa, il filosofo si sospende. */

```
    while (taken) wait();
    taken=true;
  }
```

/* Il metodo è dichiarato synchronized perché richiede l'accesso mutuamente esclusivo alle variabili condivise del monitor. Inoltre contiene l'istruzione notify che deve essere sempre inserita all'interno di una sezione synchronized */

```
  synchronized void put()
  {
```


/* La seguente istruzione risveglia il filosofo accanto a quello corrente, sospeso sulla forchetta corrente. */

```
    taken=false;
    notify();
  }

}
```