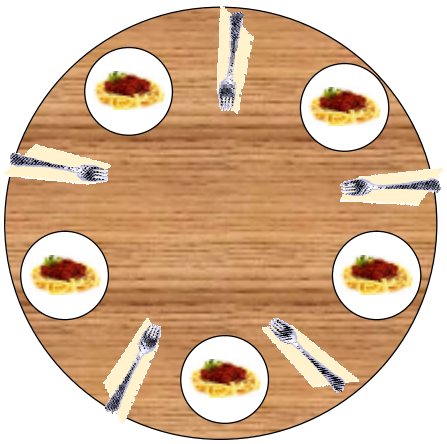
Esercizio

I filosofi a cena

Il problema

- 5 filosofi sono seduti attorno a un tavolo circolare; ogni filosofo ha un piatto di spaghetti tanto scivolosi che necessitano di 2 forchette per poter essere mangiati; sul tavolo vi sono in totale 5 forchette.
- Ogni filosofo ha un comportamento ripetitivo, che alterna due fasi:
  - una fase in cui pensa,
  - una fase in cui mangia.

Rappresentando ogni filosofo con un thread, realizzare una politica di sincronizzazione che eviti situazioni di deadlock.



Osservazioni

- i filosofi non possono mangiare tutti insieme: ci sono solo 5 forchette, mentre ne servirebbero 10;
- 2 filosofi vicini non possono mangiare contemporaneamente perche` condividono una forchetta e pertanto quando uno mangia, l'altro e` costretto ad attendere

Soluzione n.1

Quando un filosofo ha fame:

1. prende la forchetta a sinistra del piatto
2. poi prende quella che a destra del suo piatto
3. mangia per un po'
4. poi mette sul tavolo le due forchette.

→ **Possibilita` di deadlock**: se tutti i filosofi afferrassero contemporaneamente la forchetta di sinistra, tutti rimarrebbero in attesa di un evento che non si potra` mai verificare.

Soluzione n.2

Ogni filosofo verifica se entrambe le forchette sono disponibili:

- in caso affermativo, acquisisce le due forchette (in modo atomico);
- in caso negativo, aspetta.

→ in questo modo non si puo` verificare deadlock (non c'e` possesso e attesa)

Realizzazione soluzione 2

Quali thread?

- filosofo

Risorsa condivisa?

la tavola apparecchiata

-> definiamo la classe tavola, che rappresenta il monitor allocatore delle forchette

Struttura Filosofo$_i$

```
public class filosofo extends Thread
{  tavola m;
   int i;
   public  filosofo(tavola M, int id){this.m =M;this.i=id;}

   public void run()
   {    try{
        while(true)
        { System.out.print("Filosofo "+ i+" pensa....\n");
          m.prendiForchette(i);
          System.out.print("Filosofo "+ i+" mangia....\n");
          sleep(8);
          m.rilasciaForchette(i);
          sleep(100);
        }
        }catch(InterruptedException e){}

   }
}
```

Monitor

```
public class tavola
{ //Costanti:
   private final int NF=5;    // costante: num forchette/filosofi
   //Dati:
   private int []forchette=new int[NF]; //num forchette disponibili
   per ogni filosofo i
   private Lock lock= new ReentrantLock();
   private Condition []codaF= new Condition[NF];//1 coda per ogni
   filosofo i
//Costruttore:
public tavola( ) {
int i;
for(i=0; i<NF; i++)
   codaF[i]=lock.newCondition();
for(i=0; i<NF; i++)
   forchette[i]=2;
}
// metodi public e metodi privati:...}
```

Metodi public

```
public void prendiForchette(int i)throws
  InterruptedException
{ lock.lock();
  try
  {
          while (forchette[i]!=2)
               codaF[i].await();

          forchette[sinistra(i)]--;
          forchette[destra(i)]--;

  } finally{ lock.unlock();}
  return;
}
```

```
public void rilasciaForchette(int
  i)throws InterruptedException
{ lock.lock();
  try
  {
          forchette[sinistra(i)]++;
       forchette[destra(i)]++;
     if (forchette[sinistra(i)]==2)
          codaF[sinistra(i)].signal();
       if (forchette[destra(i)]==2)
          codaF[destra(i)].signal();

  } finally{ lock.unlock();}
  return;
}
```

Metodi privati

```
int destra(int i)
{ int ret;
  ret=(i==0? NF-1:(i-1));
  return ret;
}

int sinistra(int i)
{ int ret;
  ret=(i+1)%NF;
  return ret;
}
```

Programma di test

```
import java.util.concurrent.*;

public class Filosofi {
    public static void main(String[] args) {
      int i;
       tavola M=new tavola();
      filosofo []F=new filosofo[5];
      for(i=0;i<5;i++)
            F[i]=new filosofo(M, i);
      for(i=0;i<5;i++)
            F[i].start();

    }
}
```