== VERSIONE 1 ==

Main.java

```
// Soluzione #1 al problema dei cinque filosofi (deadlock)

public class Main {
     public static void main(String[] args) {
          int numFilosofi = 5; //Quanti filosofi
          Filosofo filosofi[] = new Filosofo[numFilosofi];
          Bacchetta bacchette[] = new Bacchetta[numFilosofi];

          // Inizializzo tutte le bacchette
          for (int i = 0; i < 5; i++) {
               bacchette[i] = new Bacchetta(i);
          }

          // Inizializzo tutti i filosofi
          for (int i = 0; i < 5; i++) {
               int sinistra, destra; //bacchette il filosofo può prendere

               // Calcoliamo le bacchette per questo filosofo
               sinistra = i - 1;
               if (sinistra < 0)
                    sinistra = numFilosofi - 1;
               destra = i;

               // Creo il filosofo e lancio il thread
               filosofi[i] = new Filosofo(i, bacchette[sinistra],
bacchette[destra]);
               filosofi[i].start();
          }
     }
}
```

Filosofo.java

```
// La classe che implementa il filosofo
public class Filosofo extends Thread {
     private int IDF; // L'ID di questo filosofo

     // Quanto tempo ha (al max) per pensare o mangiare
     private final int NSEC = 2;
     private Bacchetta bacchettaSin; // Bacchetta sinistra
     private Bacchetta bacchettaDes; // Bacchetta destra


     // Costruttore
     public Filosofo(int ID, Bacchetta sin, Bacchetta des) {
          IDF = ID; //Quale filosofo
          this.setName("Filosofo #" + IDF); // Diamogli un nome
     bacchettaSin = sin; // Oggetto condiviso: la bacchetta sinistra
          bacchettaDes = des; // Oggetto condiviso: la bacchetta destra
     }


     // Il filosofo pensa per un tempo random tra 0 e NSEC secondi
     private void pensa() {
          System.out.println(this.getName() + " sta pensando...");
          try {
               Thread.sleep(Math.round(Math.random() * NSEC *
1000));
          } catch (InterruptedException e) {
               e.printStackTrace();
          }
     }


     // Il filosofo ha fame, e vuole prendere le bacchette
     // Prima a sinistra, poi a destra
     // Possibilità di DEADLOCK!
     // Se volete simulare il deadlock ponete NSEC a 0, ricompilate ed eseguite
     private void prendiBacchette() {
          System.out.println(this.getName() + " ha fame...");

          System.out.println(this.getName() + " sta cercando di
prendere la bacchetta " + bacchettaSin.bacchettaNum + " alla
sinistra");
          // Prendi la bacchetta sinistra (o aspetta se è occupata)
          bacchettaSin.prendi();
          // Ho avuto la bacchetta
          System.out.println(this.getName() + " ha preso la
bacchetta " + bacchettaSin.bacchettaNum);

          System.out.println(this.getName() + " sta cercando di
prendere la bacchetta " + bacchettaDes.bacchettaNum + " alla
destra");
          // Prendi la bacchetta destra (o aspetta se è occupata)
          bacchettaDes.prendi();
```

```
          // Ho avuto la bacchetta
          System.out.println(this.getName() + " ha preso la
bacchetta " + bacchettaDes.bacchettaNum);
     }


     // Il filosofo mangia per un tempo random tra 0 e NSEC secondi
     private void mangia() {
          System.out.println(this.getName() + " sta mangiando...");
          try {
               Thread.sleep(Math.round(Math.random() * NSEC *
1000));
          } catch (InterruptedException e) {
               e.printStackTrace();
          }
     }


     // Il filosofo rilascia le bacchette in suo possesso
     private void rilasciaBacchette() {
          System.out.println(this.getName() + " ha finito di
mangiare.");
          bacchettaSin.rilascia();
          bacchettaDes.rilascia();
     }

     // Metodo principale del thread...
     public void run() {
          for (;;) {
               // Creiamo un loop infinito che modella la vita di un filosofo
               this.pensa();
               this.prendiBacchette();
               this.mangia();
               this.rilasciaBacchette();
          }
     }
}
```

Bacchette.java

```
// Classe che modella le bacchette presenti sul tavolo
// Questa è la risorsa condivisa - bisogna sincronizzare gli accessi

public class Bacchetta{
     // Un bool di int per monitorare se la bacchetta è libera o meno
     private boolean bacchettaLibera;

     // Quale bacchetta?
     public int bacchettaNum;


     // Costruttore
     public Bacchetta(int quale){
          bacchettaNum = quale;
          bacchettaLibera = true;
     }


     // Metodo sincronizzato per prendere la bacchetta
     public synchronized void prendi(){
          while (!bacchettaLibera){
               // La bacchetta è occupata, dobbiamo aspettare
               try{

     System.out.println(Thread.currentThread().getName() + " sta
aspettando che si liberi la bacchetta " + bacchettaNum);
                    wait();
               }
               catch(InterruptedException e){
                    e.printStackTrace();
               }
          }
          // Rendi questa bacchetta non disponibile
          bacchettaLibera = false;
     }


     // Metodo sincronizzato per rilasciare le bacchetta
     public synchronized void rilascia(){
          // Rendi questa bacchetta disponibile
          bacchettaLibera = true;
          // Notifico gli altri filosofi
          notifyAll();
     }
}
```

== VERSIONE 2 ==

Main.java

```
// Soluzione #2 al problema dei cinque filosofi (no deadlock, possibile
posticipazione indefinita)

public class Main {
     public static void main(String[] args) {
          int numFilosofi = 5; //Quanti filosofi
          Filosofo filosofi[] = new Filosofo[numFilosofi];

          //Oggetto condiviso: le bacchette
          Bacchette bacchette = new Bacchette(numFilosofi);

          for (int i=0;i<5;i++){
               //Creiamo i filosofi
               filosofi[i] = new Filosofo(i, bacchette);
               filosofi[i].start();
          }
     }
}
```

Filosofo.java

```
public class Filosofo extends Thread {
     private int IDF; // L'ID di questo filosofo
     private final int NSEC_THINK = 0; // tempo max per pensare
     private final int NSEC_EAT = 5; // tempo max per mangiare
     private Bacchette bacchette; // Oggetto condiviso: le bacchette


     // Costruttore
     public Filosofo(int ID, Bacchette b){
          IDF = ID; // Quale filosofo
          this.setName("Filosofo #" + IDF); // Diamogli un nome
          bacchette = b; // Oggetto condiviso: le bacchette
     }


     // Il filosofo pensa per un tempo random tra 0 e NSEC_THINK secondi
     private void pensa(){
          System.out.println(Thread.currentThread().getName() + "
sta pensando...");
          try{

     Thread.sleep(Math.round(Math.random()*NSEC_THINK*1000));
          }
          catch (InterruptedException e){
               e.printStackTrace();
          }
     }
```

```
     // Il filosofo ha fame, e vuole prendere le bacchette
     // Le prende contemporaneamente!
     // Non è possibile il deadlock, ma lo starvation sì

     private void prendiBacchette(){
          System.out.println(Thread.currentThread().getName() + "
ha fame...");
          bacchette.prendiBacchette(IDF);
     }


     // Il filosofo mangia per un tempo random tra 0 e NSEC_EAT secondi
     private void mangia(){
          System.out.println(Thread.currentThread().getName() + "
sta mangiando...");
          try{

     Thread.sleep(Math.round(Math.random()*NSEC_EAT*1000));
          }
          catch (InterruptedException e){
               e.printStackTrace();
          }
     }


     // Il filosofo rilascia le bacchette in suo possesso
     private void rilasciaBacchette(){
          System.out.println(Thread.currentThread().getName() + "
ha finito di mangiare.");
          bacchette.rilasciaBacchette(IDF);
     }


     // Metodo principale del thread...
     public void run(){
          for(;;){
               // loop infinito che modella la vita di un filosofo
               this.pensa();
               this.prendiBacchette();
               this.mangia();
               this.rilasciaBacchette();
          }
     }
}
```

Bacchette.java

```
public class Bacchette{

     private int numBacchette;
     // Un array di int per monitorare se una bacchetta è libera o meno
     // Se statoBacchette[i]==-1 -> la bacchetta i-esima è libera
     // Se statoBacchette[i]==(fil>=0) -> la bacchetta i-esima è tenuta dal
filosofo fil

     int [] statoBacchette;

     public Bacchette(int n){
          numBacchette = n;
          //Crea l'array di bacchette
          statoBacchette = new int[numBacchette];
          for (int i=0;i<5;i++)
               statoBacchette[i] = -1; //Inizialmente sono tutte libere
     }

     // Metodo sincronizzato per prendere entrambe le bacchette
     public synchronized void prendiBacchette(int numFilosofo){
          int bacchettaSinistra, bacchettaDestra;

          // Calcola la bacchetta sinistra per il filosofo che la chiede
          // Abbiamo imposto un ordinamento tra le bacchette
          bacchettaSinistra = numFilosofo-1;
          if (bacchettaSinistra<0)
               bacchettaSinistra = numBacchette-1;

          // Calcola la bacchetta destra per il filosofo che la chiede
          bacchettaDestra = numFilosofo;

          System.out.println(Thread.currentThread().getName() + "
sta cercando di prendere le bacchette " + bacchettaSinistra + " e
" + bacchettaDestra  + " req " + i);
          while (statoBacchette[bacchettaSinistra]!=-1 ||
statoBacchette[bacchettaDestra]!=-1){

               // La bacchetta è occupata, dobbiamo aspettare
               try{

                    // Stampiamo le info su quali filosofi tengono le
                    // bacchette che mi servono
                    if (statoBacchette[bacchettaSinistra]!=-1)

     System.out.println(Thread.currentThread().getName() + " sta
aspettando il filosofo " + statoBacchette[bacchettaSinistra]);
                    if (statoBacchette[bacchettaDestra]!=-1)

     System.out.println(Thread.currentThread().getName() + " sta
aspettando il filosofo " + statoBacchette[bacchettaDestra]);

                    wait(); // Il thread esce dal monitor.
                            // Altri thread possono entrare!
               }
```

```
                catch(InterruptedException e){
                    e.printStackTrace();
                }
            }

            // A questo punto entrambe le bacchette sono libere.. prendiamole
            // Ho avuto la bacchetta sinistra, rendila non disponibile
            System.out.println(Thread.currentThread().getName() + "
ha preso la bacchetta "+bacchettaSinistra+ ");
            statoBacchette[bacchettaSinistra] = numFilosofo;

            // Ho avuto la bacchetta destra, rendila non disponibile
            System.out.println(Thread.currentThread().getName() + "
ha preso la bacchetta "+bacchettaDestra+ ");
            statoBacchette[bacchettaDestra] = numFilosofo;
        }


        // Metodo sincronizzato per rilasciare le bacchette
        public synchronized void rilasciaBacchette(int numFilosofo){
            int bacchettaSinistra, bacchettaDestra;
            // Calcola la bacchetta sinistra e destra per il filosofo
            // Abbiamo imposto un'odinamento tra le bacchette
            bacchettaSinistra = numFilosofo-1;
            if (bacchettaSinistra<0)
                bacchettaSinistra = numBacchette-1;
            bacchettaDestra = numFilosofo;

            System.out.println(Thread.currentThread().getName() + "
sta rilasciando le bacchette " +bacchettaSinistra + " e " +
bacchettaDestra);
            statoBacchette[bacchettaSinistra] = -1;
            statoBacchette[bacchettaDestra] = -1;

            // Notifico gli altri thread
            notifyAll();
        }
}
```

== VERSIONE 3 ==

garantisce che tutti i filosofi possano mangiare mantenendo un contatore del numero di volte in cui questi vi riescono

Main.java

```
class Main{
     public static void main(String[] args) {
          int nFilosofi=5;
          Filosofo filosofi_array[] = new Filosofo[nFilosofi];
          Tavolo tavolo = new Tavolo(nFilosofi);

          for(int i=0; i<nFilosofi; i++){
               filosofi_array[i] = new Filosofo(i,tavolo);
               filosofi_array[i].start();
          }
     }
}
```

Filosofo.java

```
class Filosofo extends Thread {
    private int IDF;
    private Tavolo tavolo;
    private int NSECMANGIA = 0;//fisso in alcune versioni
    private int NSECPENSA = 0;//mutevole

     // Costruttore
     public Filosofo(int ID, Tavolo f){
          IDF = ID; // Quale filosofo
          tavolo = f; // Oggetto condiviso: le bacchette
          if(IDF==4){
            NSECPENSA=3;
          }

     }

    private void mangia(){
        try{
          Thread.sleep(Math.round(Math.random()*NSECMANGIA*1000));
        }
        catch (InterruptedException e){
            e.printStackTrace();
        }
    }

    private void pensa(){
        try{
           Thread.sleep(Math.round(Math.random()*NSECPENSA*1000));
        }
        catch (InterruptedException e){
            e.printStackTrace();
        }
    }
```

```
    void prendi(){
        tavolo.prendiForchette(IDF);
    }

    void rilascia(){
        tavolo.rilasciaForchette(IDF);
    }

    public void run(){
        while(true){
            pensa();
            prendi();
            mangia();
            rilascia();
        }
    }
}
```

Tavolo.java

```
class Tavolo{
     private int nForchette;
     private boolean statoForchette[]; //true occupato e false libero
     private int nMangiano;
     private int nvolte[];
     private int nvolteMangiato[];

     public Tavolo(int n){
          nForchette=n;
          statoForchette = new boolean[nForchette];

          // numero volte in cui non sono riuscito a prendere le forchette
          nvolte = new int[nForchette];
          nvolteMangiato = new int[nForchette]; // numero volte in cui
sono riuscito a mangiare
          for(int i=0;i<n;i++){
               statoForchette[i]=false; //tutte libere
               nvolte[i]=0;
               nvolteMangiato[i]=0;
          }
          nMangiano=0;
     }

     public synchronized void prendiForchette(int id){
          //System.out.println("***Stanno mangiando in " +
nMangiano + "***");
          //calcolo forchette dx e sx
          int forchettasx = id;
          int forchettadx = (id+1)%5;
          int delta = 0; //1000000000 differenza consentita

          System.out.println(id + " sta provando a prendere le
forchette");
```

```
          while(statoForchette[forchettasx] ||
               statoForchette[forchettadx] ||
          nvolteMangiato[id]>nvolteMangiato[(id-1+5)%5]+delta ||
          nvolteMangiato[id]>nvolteMangiato[(id+1)%5]+delta) {

               //controllo alternativo
               // while(statoForchette[forchettasx] ||
statoForchette[forchettadx] || nvolte[id]<nvolte[(id-1+5)%5] ||
nvolte[id]<nvolte[(id+1)%5]) {

               try{
                    if(statoForchette[forchettasx]){
                         System.out.println("id + " sta aspettando
la forchetta " + forchettasx);
                    }
                    if(statoForchette[forchettadx]){
                         System.out.println("id + " sta aspettando
la forchetta " + forchettadx);
                    }
                    System.out.println("*** Stanno mangiando in " +
nMangiano + " ***");
                    nvolte[id]++;
                    stampaMangiato(id,"");

                    System.out.println(id + " wait..");
                    wait();
                    System.out.println(id + " risvegliato..");

               } catch(InterruptedException e){
                    e.printStackTrace();
               }
          }

          statoForchette[forchettasx]=true;
          statoForchette[forchettadx]=true;
          System.out.println(id + " ha preso le forchette");
          nvolte[id]=0;
          nMangiano++;
          stampaMangiato(id,"");

     }

     public synchronized void rilasciaForchette(int id){
          //calcolo forchette dx e sx
          int forchettasx = id;
          int forchettadx = (id+1)%5;

          //libero le forchette
          statoForchette[forchettasx]=false;
          statoForchette[forchettadx]=false;
          nMangiano--;
          nvolteMangiato[id]++;
          System.out.println(id + " ha mangiato");
          System.out.println(id + " ha liberato le forchette.");
```

```
            notifyAll();
        }

        private void stampaMangiato(int id){
            for(int i=0; i < nvolteMangiato.length; i++){
                        System.out.print(nvolte[i] + " ");
            }
            System.out.print("*** ");
            for(int i=0; i < nvolteMangiato.length; i++){
                        System.out.print(nvolteMangiato[i] + " ");
            }
            System.out.println();
        }
}
```