Dining Philosophers Problem
Solution based on Monitor

```
package diningphilisopher;

/**
Dining Philosopher Problem
Monitor.java
@author G Jung
@version March 27, 2017
 */
public class Monitor {

          /**
           instance variable philosopherState[]: array size 5,  elementValues: 0 = thinking, 1=Hungry,
2=eating
           instance variable chopStickStates[]: array size 5, elementValues: false = chop stick is being
used, true = available
           */
int philospherStates[] = new int[5];
boolean chopStickStates[] = new boolean[5];  // false = in use, true = free

/**
default (no-argument) constructor
Precondition: no argument
Postcondition: 5 philosophers states are initialized 0 (i.e., thinking)
and 5 chop sticks states are initialized true (i.e., available)
 */
public Monitor() {
   for(int i=0;i<5;i++) {
            philospherStates[i]=0;
            chopStickStates[i]=true;
   }
}

/**
Precondition:
Postcondition: print out the states of the 5 philosophers: Thinking, Hungry, Eating
 */
public synchronized void printPhilosopherStates() {

          System.out.println();
                    for(int i=0;i<5;i++)
                              switch(philospherStates[i]){
                                    case 0: System.out.print(" " + "...Thinking");
                                            break;
                                    case 1: System.out.print(" " + "....Hungry");
                                            break;
                                    case 2: System.out.print(" " + ".....Eating");
                                            break;
        }
}

/**
Precondition: philosopherID must be in [0..4]
Postcondition: If both chop sticks (left, right) are not available philosopherID must wait (call
wait())
and set phisosopher's state 1 (Hungry). If both chop sticks are available set phisosopher's state 2
(Eating)
and update states of both chop sticks in use (i.e., false)
@param philosopherID
 */
public synchronized void pickUpChopStickToEat(int philosopherID)
{
  while(!chopStickStates[philosopherID] || !chopStickStates[(philosopherID+1)%4])
  {   // while it can't have both forks, wait
          philospherStates[philosopherID] = 1;
    try{
    wait();
    }
    catch(InterruptedException e){}
  }
  philospherStates[philosopherID] = 2;  // eating
  chopStickStates[philosopherID] = false;  // in use
  chopStickStates[(philosopherID + 1) % 4] = false;
}

/**
Precondition: philosopherID must be in [0..4]
Postcondition: Update both (left, right) chop sticks available (i.e., true)
and call notify() to wake up other philosopher
 @param philosopherID
 */
public synchronized void PutDownChopStickAfterEating(int philosopherID)
{
        chopStickStates[philosopherID] = true; // available
        chopStickStates[(philosopherID + 1) % 4] = true;
        philospherStates[philosopherID] = 0;  // thinking
   notify();
```

```
}
}
```

Philosopher.java

```
package diningphilisopher;

/**
Dining Philosopher Problem
Philosopher.java
@author G Jung
@version March 27, 2017
*/


import java.util.Random;

public class Philosopher implements Runnable {

/**
 instance variables
 random r is initialized by new Random()
 */
        private Monitor monitorObject;
        private WaiterServingPhilosophers waiterObject;
        private Random r = new Random();  // Random number generator object
        private int philosopherID;
        private double time;

/**
 @param philosopherID
 @param monitorObject
 @param waiterObject
 Precondition: philosopherID must in [0..4], m: Monitor object, w: WaiterServingPhilosophers object
 Postcondition: initialize the instance variables philosopherID, monitorObject, waiterObject
 */
        public Philosopher(int philosopherID, Monitor monitorObject, WaiterServingPhilosophers
waiterObject) { // constructor

                this.philosopherID = philosopherID;
                this.monitorObject = monitorObject;
                this.waiterObject = waiterObject;
        }

/**
Precondition:
Postcondition: Each philosopher iterates 10 times executing the following tasks:
pickupChopStickToEat and then sleep random milliseconds
and putDownChopStickAfterEating. Then sleep random milliseconds. random sleep time is calculated
by (int)(1000 * r.nextDouble()), where r is random object.
After the philosopher completes the tasks (10 iterations), the waiterObject must report the status
of the philosopher (i.e., waiterObject.reportFinishedDining(philosopherID).
*/
   public void run() {

         for(int i=0; i<10; i++) {

                monitorObject.pickUpChopStickToEat(philosopherID);
                time = 1000 * r.nextDouble();
                try {Thread.sleep((int)time);} catch(Exception e){}
                monitorObject.PutDownChopStickAfterEating(philosopherID);
                time = 1000 * r.nextDouble();
                try {Thread.sleep((int)time);} catch(Exception e){}
        }//of for


        waiterObject.reportFinishedDining(philosopherID);   // tell the timer this one is done
   }
 }




package diningphilisopher;

/**
 Dining Philosopher Problem
 WaiterServingPhilosophers.java
 @author G Jung
 @version March 27, 2017
 */

public class WaiterServingPhilosophers implements Runnable {

/**
instance variables
```

```
 */
          private Monitor monitorObject;
          private int noOfPhilosophersFinishedDining;
/**
 Constructor
 @param m
 Precondition: Monitor object m
 Postcondition: Printout "Waiter thread Serving 5 Philosophers Started.......". Initialize
monitorObject = m
 and noOfPhilosophersFinishedDining = 0
 */

public WaiterServingPhilosophers(Monitor m) {   // constructor
  System.out.println("Waiter thread Serving 5 Philosophers Started.......");
  monitorObject = m;
  noOfPhilosophersFinishedDining = 0;
  new Thread(this, "Timer").start(); // make a new thread and start it
 }

/**
 @param philosopherID
 Precondition: philosopherID must be in [0..4]
 Postcondition: increase noOfPhilosophersFinishedDining by one, and printout philosopher
philosopherID
 finishes eating..
 */
public void reportFinishedDining(int philosopherID) {
          noOfPhilosophersFinishedDining++;
  System.out.println("\n!!!!!! " + "philosopher " + philosopherID + " is now terminating !!!"
          + "\n........Number of philosopher threads finished by now: " +
noOfPhilosophersFinishedDining );
}

/**
 Precondition:
 Postcondition: as long as noOfPhilosophersFinishedDining is not 5, wait until all of them finish
eating
 Waiting is done by Thread.sleep(500). If all of them finished eating waiter thread terminates.
 */
public void run() {
  while(noOfPhilosophersFinishedDining!=5) {
            try {Thread.sleep(500);
            monitorObject.printPhilosopherStates();
            }
      catch(Exception e){}
  }
}
}



package diningphilisopher;

public class TestDiningPhisopher {

        public static void main(String args[]) {
                  Monitor monitorObject = new Monitor(); // thing begins here
                  WaiterServingPhilosophers waiterObject = new
WaiterServingPhilosophers(monitorObject);
                  Philosopher [] philosophers = new Philosopher[5];
                  Thread philosopherThread;
                  for(int i=0; i<5; i++){
                    philosophers[i] = new Philosopher(i,monitorObject,waiterObject);
                    philosopherThread = new Thread(philosophers[i], "Philosopher" + i);
                    System.out.println("Philosopher " + i + " starts...... ");
                    philosopherThread.start();
                  }
                  System.out.println("____________________________________________________");
        }

}
```