

## Practical Task 2

**Aim:** Practical experience with synchronization concepts

**Begin Date:** November 21, 2017

**End Date:** December 3, 2017

**Submission:** an archive file (zip) that includes all files relevant to execute and test your program should be submitted through MyMoodle; if the submitted program does not run (execute) using the instructions provided in the ReadMe.txt file, your solution is not accepted.

**Instructions:** you are allowed and encouraged to use the literature recommended in the course; use of other literature is also encouraged; assignment must be completed independently and without the help of other colleagues or the teacher.

### Problem definition

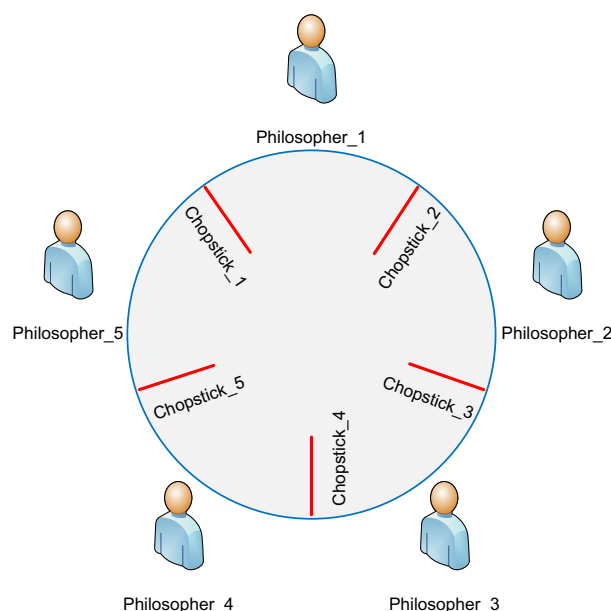


Figure 1. Dining Philosophers

Develop a Java application that simulates the Dining Philosophers problem (see Chapter 6 of the book<sup>1</sup>). Five philosophers sit around a table (see Figure 1) and in front of each philosopher there is a bowl of food. A philosopher can be in three states: EATING, THINKING, or HUNGRY. There are five chopsticks available. A hungry philosopher needs two chopsticks (his left and right chopstick) for eating. If a chopstick is used by a neighbor, then the philosopher must wait until the chopstick is released. The philosopher first picks up the left chopstick, and then the right chopstick. Initially all five philosophers are in THINKING state. Time for THINKING and EATING is generated pseudo-randomly from a discrete uniform distribution [0, 1000]. Basically, a philosopher is in THINKING or EATING state for 0-1000 time units (milliseconds); one of these values is selected in pseudo-random<sup>2</sup> manner to simulate thinking or eating time.

We will provide four classes for you: *DiningPhilosopher*, *Philosopher*, *Chopstick*, and *DiningPhilosopherTest*. Details for each class are listed below:

1. The *DiningPhilosopher* class contains:
  - a. Variables: *philosophers* (list), *chopsticks* (list), *DEBUG* (boolean), *NUMBER\_OF\_PHILOSOPHERS* (int), *SIMULATION\_TIME* (int), *SEED* (int), *executorService*
  - b. Methods: *printTable* (that prints the average eating/thinking/hungry times for each philosopher); *initialize* (used to set the simulation time, which indicates for how many milliseconds the simulation will run; the seed for the random generator); *start* (that

<sup>1</sup> Operating System Concepts, 9th Edition, by Abraham Silberschatz, Peter B. Galvin, Greg Gagne

<sup>2</sup> <http://docs.oracle.com/javase/7/docs/api/java/util/Random.html>

starts the simulation process).

2. The *Philosopher* class implements the runnable interface, which means that each philosopher will be running in a separate thread. This class contains:
  - a. Variables: Left and right chopstick objects.
  - b. Methods: *getAverageThinkingTime*, *getAverageEatingTime*, *getAverageHungryTime* (to retrieve the average thinking, eating, and hungry times); *getTotalThinkingTime*, *getTotalEatingTime*, *getTotalHungryTime* (to retrieve the total thinking, eating, and hungry times); *getNumberOfThinkingTurns*, *getNumberOfEatingTurns*, *getNumberOfHungryTurns* (to retrieve the number of thinking, eating, and hungry turns); *run()* (that basically simulates the think, eat, and hungry process for the corresponding philosopher).
3. The *Chopstick* class is used by the *Philosopher* class. It contains:
  - a. Variables: *id* (the chopstick id)
  - b. Methods:
4. The *DiningPhilosopherTest* is a class that test the correctness of your implementation
  - a. Methods: *test1*, *test2*, and *test3* (test different scenarios)

You should:

1. Import these four classes.
2. Complete the *start()* method in the *DiningPhilosopher* class. You need to provide means to interrupt the philosophers when the simulation time is over.
3. Complete the *initialize(...)* method in the *DiningPhilosopher* class. You should create the chopsticks, and philosophers, and assign chopsticks to philosophers.
4. Implement the *run()* method in the *Philosopher* class.
5. Calculate the average<sup>3</sup> EATING, THINKING, and HUNGRY times. When the *start()* method has finished the average eating, thinking, and hungry times for each philosopher should be known.
5. If DEBUG is set to true, your implementation should print in the console the major events that occur during the simulation (for instance, *Philosopher\_1 is THINKING*, *Philosopher\_4 is EATING*, *Philosopher\_3 is HUNGRY*, *Philosopher\_2 released Chopstick\_3*, *Philosopher\_1 picked-up Chopstick\_1*, *Deadlock detected ...*)
6. Detect and report the deadlock (that is a state where all philosophers hold the left chopstick and wait for the right chopstick). Alternatively, prevent deadlocks from happening and explain how you prevented it from happening.
7. Add comprehensive comments (in the code) to explain your implementation.

### Folder structure

---

- Create a new project and name it 1dv512.userid (e.g. 1dv512.sm222bt)
- Put all your files in one package (default)
- Pack whole project directory (src) into zip archive. Your zip should contain some root directory (e.g. sm222bt), not just list of files.

Example of the folder structure:

- sm222bt
  - src
    - Chopstick.java,

---

<sup>3</sup> <http://mathworld.wolfram.com/ArithmeticMean.html>

- DiningPhilosopherTest.java,
- DiningPhilosopher.java
- Main.java
- Philosopher.java,
- ReadMe.txt,

### Instructions for running and testing the application

---

Provide any information that is relevant for running your application in the ReadMe.txt file. Expect that we test your solution using a method like this:

```
public static void main(String args[]) {  
    DiningPhilosopher dp = new DiningPhilosopher();  
    dp.DEBUG = true;  
    int simulationTime = 10000;  
    int seed = 100;  
    dp.initialize(simulationTime, seed);  
    dp.start();  
    dp.printTable();  
}
```