Practical Task 2

Aim: *Practical experience with synchronization concepts*

Begin Date: *November 14, 2019*

End Date: *November 22, 2019, midnight*

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

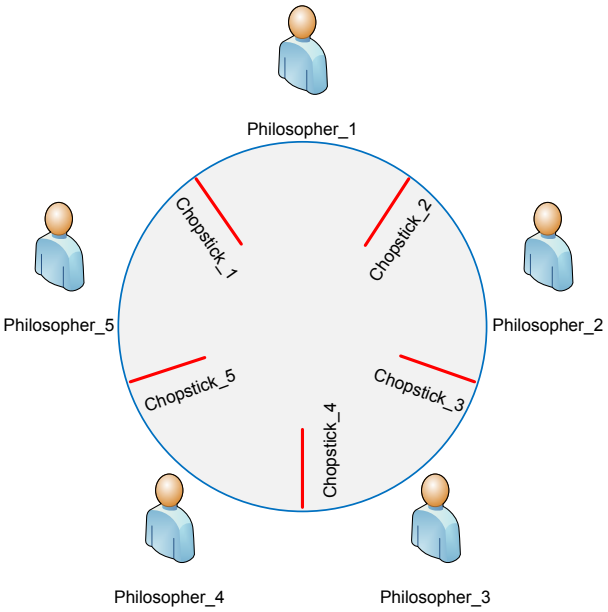

Figure 1. Dining Philosophers

Develop a Java application that simulates the Dining Philosophers problem (see Chapter 6 of the book[1]). Five philosophers sit around a table (see Figure 1) and in front of each philosopher there is a bowl of food. There are five chopsticks available. A philosopher can be in three states: EATING, THINKING, or HUNGRY. 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 0–999 time units (milliseconds); these values are selected in a pseudo-random[2] manner to simulate thinking or eating time.

We will provide five classes for you: *DiningPhilosopher, Philosopher*, *Chopstick, DiningPhilosopherTest*, and *Main*. 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

[1] Operating System Concepts, 9th Edition, by Abraham Silberschatz, Peter B. Galvin, Greg Gagne
[2] http://docs.oracle.com/javase/7/docs/api/java/util/Random.html

milliseconds the simulation will run; the seed for the random generator); *start* (that 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), *myLock* (the lock associated with the chopstick)
    b. Methods: *getId()*, *getLock()*
4. The *DiningPhilosopherTest* is a class that tests the correctness of your implementation.
    a. Methods: *test1, test2, and test3* (test different scenarios)
5. *Main* is a simple driver class that launches a single run of your implementation.

You should:

1. Import these five classes.
2. Complete the *initialize(...)* method in the *DiningPhilosopher* class. You should create the chopsticks, and philosophers, and assign chopsticks to philosophers.
3. Complete the *start()* method in the *DiningPhilosopher* class. You need to provide the means to interrupt the philosophers when the simulation time is over. Make sure all of the threads actually terminate!
4. Implement the *run()* method in the *Philosopher* class.
5. Calculate the average[3] EATING, THINKING, and HUNGRY times. When the *start()* method has finished the average eating, thinking, and hungry times for each philosopher should be known. Look at the code and the comments in the *DiningPhilosopherTest* class for further information.
6. 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, Philosoper_2 released Chopstick_3, Philosopher_1 picked-up Chopstick_1, Deadlock detected* …)
7. Prevent **deadlocks** (a state where all philosophers hold the left chopstick and wait for the right chopstick) in your implementation and explain in the comments how you prevented it from happening. Run the unit tests with *DiningPhilosopherTest* multiple times to ensure that you have indeed prevented occasional deadlocks.
8. Add further 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 the whole project directory (src) into zip archive. Your zip should contain some root

[3] http://mathworld.wolfram.com/ArithmeticMean.html

directory (e.g. sm222bt), not just a list of files.

Example of the folder structure:

- sm222bt
  - src
    - Chopstick.java
    - 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 (see the *Main* class):

```
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();
}
```