Concordia University
Comp 346 – Summer 2024

Operating Systems - Programming Assignment 3

Deadline:	By 11:59 PM - Thursday August 15, 2024.
Late Submission:	No late submission.
Teams:	The assignment can be done individually or in teams of 2. Submit only one assignment per team.
Purpose:	The purpose of this assignment is to apply in practice the multi-threading features of the Java programming language.

- **Objective:**

  Implement the dining philosopher's problem using a monitor for synchronization.

- **Background:**

  This assignment is a slight extension of the classical problem of synchronization – the Dining Philosophers problem. You are going to solve it using the Monitor synchronization construct built on top of Java's synchronization primitives. The extension refers to the fact that sometimes philosophers would like to talk, but only one (any) philosopher can be talking at a time while they are not eating. If you need help, consult the references at the bottom.

- **Source Code Check List:**

  There are five files that come with the assignment. A soft copy of the code is available to download from the course web site. This time the source code is barely implemented (though compiles and runs). You are to complete its implementation. Files distributed with the assignment requirements:
  - common/BaseThread.java – unchanged
  - DiningPhilosophers.java - the main()
  - Philosopher.java - extends from BaseThread
  - Monitor.java - the monitor for the system
  - Makefile - take a look

- **Tasks:**

Make sure you put comments for every task that involves coding to the changes that you've made. This will be considered in the grading process.

✦ Task 1:

Complete the implementation of the Philosopher class, that is all its methods according to the comments in the code. Specifically, eat(), think(), talk(), and run() methods have to be implemented entirely.
Non-mandatory hints are provided within the code.

✦ Task 2 - The Monitor:

Implement the Monitor class for the problem. Make sure it is correct, deadlock- and starvationfree implementation that uses Java's synchronization primitives, such as wait() and notifyAll(); no use of Semaphore objects is allowed. Implement the four methods of the Monitor class; specifically, pickUp(), putDown(), requestTalk(), and endTalk(). Add as many member variables and methods to monitor the conditions outlined below as needed:

1. A philosopher is allowed to pickup the chopsticks if they are both available. That implies having states of each philosopher as presented in your book. You might want to consider the order in which to pick the chopsticks up.
2. If a given philosopher has decided to make a statement, they can only do so if no one else is talking at the moment. The philosopher wishing to make the statement has to wait in that case.

✦ Task 3 - Variable Number of Philosophers:

Make the application to accept a positive integer number from the command line, and spawn exactly that number of philosophers instead of the default one. If there are no command line arguments, the given default should be used. If the argument is not a positive integer, report this fact to the user, print the usage information as in the example below:

```
% java DiningPhilosophers -7.a
"-7.a" is not a positive decimal integer

Usage: java DiningPhilosophers [NUMBER_OF_PHILOSOPHERS]
%
```

Use Integer.parseInt() method to extract an int value from a character string. Test your implementation with the varied number of philosophers. Submit your output from "make regression".

- **Evaluation:**

  You will be evaluated essentially on the implemented tasks. Evaluation criteria

Task	Marks
Task 1.	20%
Task 2.	60%
Task 3.	20%
Total:	100%

- **Submission:**

  - Create one **.**zip file, containing the necessary files (.java, .txt and test cases). If the assignment is done individually, your file should be called p*a3_studentID*, where pa3 is the number of the assignment and *studentID* is your student ID number. If the work is done in a team of 2 people, the zip file should be called p*a3_studentID1_studentID2* where *studentID1* and *studentID2* are the student ID numbers of each student.

  - Upload your .zip file on moodle (or on your section web page) as *Programming Assignment 3* before midnight on the due date.

IMPORTANT (Please read very carefully):

A demo will take place with the markers afterwards. Markers will inform you about the details of demo time and how to book a time slot for your demo. If working in a group, both members must be present during demo time. Different marks may be assigned to teammates based on this demo.

- If you fail to demo, a zero mark is assigned regardless of your submission.
- If you book a demo time, and do not show up, for whatever reason, you will be allowed to reschedule a second demo but a **penalty of 50% will be applied**.
- Failing to demo at the second appointment will result in zero marks and no more chances will be given under any conditions.

- **Notice:**
    - Note that this code has been tested on Windows using Eclipse. You may need to make changes if you would like to run on other OS.
    - You must not modify the original Java code provided nor change the size of the arrays but simply implement the required elements.