DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING
CONCORDIA UNIVERSITY
COMP 346: Operating Systems
Fall 2023
Programming Assignment 3
Due date: Wednesday November 22nd before 11:59 pm

Total Marks: 50

Objectives: This programming assignment is an extension to the classical problem of synchronization – the Dining Philosophers problem. You are going to solve it using the Monitor construct built using Java's atomicity and synchronization primitives. The extensions here are as follows: (1) the solution has to be both deadlock- and starvation-free; (2) sometimes philosophers would like to talk, but only one (any) philosopher can be talking at a time while the philosopher is not eating.

The source code for this assignment is supplied separately in a zip file. You will need to fill in the missing code. You have to complete the following **4** tasks as part of this assignment:

Note: All code written by you should be well commented. This may be part of the grading scheme.

Task 1. The Philosopher Class

Complete implementation of the Philosopher class, i.e., all its methods, according to the comments in the code. Specifically, *eat()*, *think()*, *talk(),* and *run()* methods have to be fully implemented. Some hints are provided within the code. Your added code must be well commented.

Task 2. The Monitor

Implement the Monitor class for the problem. Make sure that it is correct; **both deadlock- and starvation-free (**Note: this is an important criterion for the grading scheme**)**; uses either Java's synchronization primitives such as wait()/notify()/notifyAll() or uses Java utility *lock* and condition variables; and **does not use** any Semaphore objects. Implement the four methods of the Monitor class; specifically, *pickUp()*, *putDown()*, *requestTalk()*, and *endTalk()*. Add as many member variables and methods to meet the following specifications:

1. A philosopher is allowed to pick up the chopsticks if they are both available. It has to be **atomic** so that no deadlock is possible. Refer to the related discussion in your textbook.
2. **Starvation is to be handled**. (Note: Starvation will not be possible if a philosopher's wait is guaranteed to be upper bounded).
3. If a given philosopher has decided to make a statement, the philosopher can do so only if no one else is talking at the moment. The philosopher wishing to make the statement first makes a request to talk; and has to wait if someone else is talking. When talking is finished then others are notified by *endTalk*.

Task 3. Variable Number of Philosophers

Make the program accept the number of philosophers as a command line argument, and spawn exactly that many numbers of philosophers instead of the default specified in code. If there is no command line argument, the given default should be used. If the argument is not valid, report an error to the user and 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] %
```

You can use Integer.parseInt() method to extract an int value from a character string. Test your implementation with a varied number of philosophers. Submit your output from "*make regression*" (refer to the included Makefile for a Linux environment); for any other environment you are using, consult the user's manual for equivalence.

Task 4. Starvation

Briefly explain in a few sentences how your implementation handles starvation. This will facilitate marking.

Deliverables

Submit your assignment electronically via Moodle. Working copy of the program(s) and sample outputs should be submitted for the tasks that require them. A text or pdf file with written answers, if any, to tasks 1 to 4 should be provided. Archive all files with an archiving and compressing utility, such as zip, into a single file.