COMP 346 - Winter 2019
Programming Assignment 3

Due Date: 11:59 PM – April 5, 2019

Dining Philosophers Problem

1) Objective

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

2) Source Code

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.

2.1 File Checklist

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

3) 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. Additionally, a philosopher can be talking for a limited time (please see below for details). While one philosopher is talking, none of the others philosophers can sleep; however, they can be eating or thinking.

能够有一个人讲话，其他人不能睡觉，另外的人必须要eating or thinking

4) **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: The Philosopher Class

Complete the implementation of the Philosopher class, that is all its methods according to the comments in the code. Specifically, eat(), think(), talk(), sleep(), 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 starvation-free 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.

3. Each professor has a limited time to talk, make sure you implement them accordingly.

Task 3: The Monitor; Alternative Solution

Implement alternative solution to the Monitor class for the problem if philosophers are assigned priority numbers/values (that is, a philosopher with a higher priority should be allowed to eat first). You can assume that the priority for each philosopher is unique and indicated by a number between 1 and N, where N is the number of philosophers on the table; where 1 indicates highest priority and N indicates lowest priority. Make sure the solution is correct; that is, it is deadlock- and starvation-free that uses Java's synchronization primitives, such as wait() and notifyAll();

Task 4: 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".

Task 5: Dynamic Modification of the Number of Philosophers

In that task, you are required to show whether or not Task 2 above can be modified so that the number of philosophers can be changed randomly in the middle of the task; that is philosophers should be allowed to leave in the middle or new philosophers can join the table. If this is feasible, implement the changes. If not, you should comment clearly why this is not possible.

5) Bonus Task

Task 6: Additional Condition

In that task, two pepper shakers are placed on the table for the philosophers to use while eating. You are required to add another mutual exclusion for the sharing of these two pepper shakers between only eating philosophers.

6) Deliverables

Submit the complete source code after the last task you managed to complete as well as the final output you are getting.
Archive it into say pa3.zip and submit it electronically.

7) Grading Scheme

Grading Scheme:

T#	MX	MK
1	/1	
2	/3	
3	/2	
4	/2	
5	/2	
6	/2 (Bonus)	
Total:	/10	

(T# - task number, MX - max (out of), MK - your mark)

8) References

1. Java API: http://java.sun.com/j2se/1.3/docs/api/
2. http://java.sun.com/docs/books/tutorial/essential/TOC.html#threads