Issued date : *October 25 2016* **Due Date :** *November 22 2016*
Professor : *Kerly Titus*

1. Objective

The objective of this programming assignment is to 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. If you need help, consult the references at the bottom.

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()*, 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-free 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.

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 a variable number of philosophers. Submit your output from *"make regression".*

5 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.

6 Grading Scheme

Grading Scheme:
===============

T#	MX	MK
1	/2	
2	/6	
3	/2	

Total: /10
(T# - task number, MX - max (out of), MK - your mark)

7 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