**cs340 – Spring 2017**
**Project 1**

# Exam Day
## Due Date: April 24th

Through its implementation, this project will familiarize you with the creation and execution of threads, and with the use of the Thread class methods. In order to synchronize the threads you will have to use (when necessary), run( ), start( ), currentThread( ), getName( ), join( ), yield( ), sleep(time), isAlive( ), getPriority( ), setPriority( ), interrupt( ), isInterrupted( ), synchronized methods.
In synchronizing threads, DO NOT use any semaphores. DO NOT use wait( ), notify( ) or notifyAll();

Students come to school (simulated by sleep of random time) and ***busy wait*** (use a boolean variable) in front of the classroom for the instructor to arrive
Each student should take two exams. Three exams are administrated throughout the day.

Once the instructor arrives (sleep of random time), he allows students to enter the classroom and wait until is time for the exam to start (use sleep of exact time). Students are rushing to enter the classroom (simulate this by increasing the student's priority. Next, the student sleeps for a short time, next it resets its priority to the default values. Use the **getPriority** and **setPriority** methods). Students enter the classroom up to the classroom's *capacity* (use a counter and access it from a **synchronized method** in order to implement mutual exclusion). For later implementation you need to keep track of the order in which students entered the room.

If the classroom is full or if the student didn't make it by the time the exam starts, we consider that the student missed the exam. The student will ***yield*** twice and next he will ***busy wait*** outside the classroom to take the next exam.

Students who are in the classroom, will work on the exam (simulated by a sleep of **long time**)
The instructor will sleep throughout the exam (simulated by sleep of exact time, allowing students to cheat as much as they want).
When the exam ends, the instructor will collect the exams, and allow students to leave the classroom in the order in which they entered (implement this by having the instructor interrupt the students one by one in order – use **isInterrupted()** and **interrupt()** methods)

Next, the instructor and students (who took the exam) will take a break (sleep of random time) and get ready for the next exam. NOTE: The students who missed the exam are already busy waiting to enter the classroom for the next exam.

At the end of the day, the exam grades for each student should be displayed (exam1, exam2, exam3). You can assign grades by generating random numbers between 10 and 100. If a student missed an exam, his grade will be 0.

After the grades are posted students will leave by joining each other. Each student will join another student; they will leave in sequential order. Student N joins with student N-1, student N-1 joins with N-2, …, student 2 joins with student 1 (use **join( ), isAlive( )).**

The instructor will terminate after all students leave.
Default values**: number of students:** **14**
**Class capacity:** **10**

Develop a multithreaded Java program that will implement the Instructor and student threads and closely follow the given story.

<u>More about interrupting threads</u>: Method **Thread.interrupt()** does not interrupt a running thread (you will use interrupt directly on your student threads, as in passengerThread1.interrupt()). What the method actually does is throw an interrupt if the thread is blocked, so that it exits the blocked state. More precisely, if the thread is not blocked, calling interrupt() will not hurt; otherwise (if the thread is blocked at one of the methods), the thread will get an exception (the thread must be prepared to handle this condition which in this project just means that you can proceed forward) and escape the blocked state.

*Choose appropriate amount of time(s) that will agree with the content of the story. I haven't written the code for this project yet, but from the experience of grading previous semester's projects, a project should take somewhere between 45 seconds and at most 1 minute and ½, to run and complete.*

## <u>Guidelines</u>

1. Do not submit any code that does not compile and run. If there are parts of the code that contain bugs, <u>comment it out and leave the code in</u>. A program that does not compile nor run will not be graded.

2. Closely follow all the requirements of the Project's description.

3. Main class is run by the main thread. The other threads must be manually specified by either implementing the Runnable interface or extending the Thread class. Separate the classes into separate files. Do not leave all the classes in one file. Create a class for each type of thread. **Don't create packages**.

4. The program asks you to create different types of threads. There is more than one instance of the thread. No manual specification of each thread's activity is allowed (e.g. no Student2.getInClssroom()).

5. Add the following lines to all the threads you make:

```
public static long time = System.currentTimeMillis();

public void msg(String m) {
   System.out.println("["+(System.currentTimeMillis()-time)+"]
"+getName()+": "+m);
   }
```

It is recommended to initialize time at the beginning of the **main method**, so that it will be unique to all threads.

6. There should be printout messages indicating the execution interleaving. Whenever you want to print something from that thread use: msg("some message about what action is simulated");

7. NAME YOUR THREADS or the above lines that were added would mean nothing. Here's how the constructors could look like (you may use any variant of this as long as each thread is unique and distinguishable):

```
// Default constructor
public RandomThread(int id) {
    setName("RandomThread-" + id);
}
```

8. Design an OOP program. All thread-related tasks must be specified in its respective classes, no class body should be empty.

9. **No** implementation of semaphores or use of **wait( ), notify( ) or notifyAll( )** are allowed.
10. thread.sleep() is not busy wait.   while (expr) {..} is busy wait.

11. "Synchronized" is not a FCFS implementation. The “Synchronized” keyword in Java allows a lock on the method, any thread that accesses the lock first will control that block of code; it is used to enforce mutual exclusion on the critical section. **FCFS should be implemented in a queue or other data structure**.

12. DO NOT USE System.exit(0); the threads are supposed to terminate naturally by running to the end of their run methods.

13.  Command line arguments must be implemented to allow changes to the **Nstudents, capacity**.

14.  Javadoc is not required. Proper basic commenting explaining the flow of the program, self-explanatory variable names, correct whitespace and indentations are required.

Tips:
-If you run into some synchronization issues, and don't know which thread or threads are causing it, press F11 which will run the program in debug mode. You will clearly see the thread names in the debug perspective.

## **Setting up project/Submission:**

### **In Eclipse:**

Name your project as follows: LASTNAME_FIRSTNAME_CSXXX_PY
where LASTNAME is your last name, FIRSTNAME is your first name, XXX is your course, and Y is the current project number.
For example: Doe_John_CS340_p1

**To submit:**
-Right click on your project and click export.
-Click on General (expand it)
-Select Archive File
-Select your project (make sure that .classpath and .project are also selected)
-Click Browse, select where you want to save it to and name it as LASTNAME_FIRSTNAME_CSXXX_PY
-Select Save in **zip format**, Create directory structure for files and also Compress the contents of the file should be checked.
-Press Finish

**Upload the project on Blackboard.**

**The project must be done individually, not any other sources. No plagiarism, No cheating. Read the academic integrity list one more time.**