Instructor: Dr. Simina Fluture

CSCI 344-715 Spring 2017

Project 1 – Due Date: April 7th (with no penalty until April 12th)

Note: the projects must be done individually. No exceptions.

Exam Day

Students come to school (simulated by sleep of random time) and ***wait*** in front of the classroom for the instructor to arrive. Each student should take three exams. The exams are scheduled every 2 hours and each exam takes 1 hour. Four exams are administrated throughout the day.

The instructor allows students to enter the classroom 15 minutes before the exam starts (use notifyAll). Students enter the classroom up to the classroom's *capacity*. 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 wait outside the classroom to take the next exam at the next available time.

Once in the classroom, students take a seat at a table (students that sit at the same table should ***wait*** on the same object). Each table has *numSeats* seats. All the tables, but the very last one must have all seats occupied.

When it's time to start the exam, the instructor hands out the exams (by signaling the students). Then, the students and the instructor **wait** for the exam to end.

When the exam ends, each student takes a brief time to check their notes (simulated by sleep(random time) about 5 units). Next, the student will return the exam by signaling the instructor and will ***wait*** for the instructor to grade it (each student blocks on a different object; use an implementation similar to the one in **rwcv.java**). It will take the instructor a very brief (random) time (up to 2 units) to check an exam. After each exam is checked, the instructor will assign a grade and notify the student (FCFS order).

Next, the instructor and students will get ready for the next exam.

At the end of the day, the exam grades for each student should be displayed. If a student missed an exam, his grade will be 0.

If a student took three exam can terminate. The instructor will terminate after all four exams are administrated.

Develop a monitor that will synchronize the three types of threads: student, teacher, timer, in the context of the problem. Besides the specified notification objects, you should have at least one synchronized method.

The number of students *numStudents*, *capacity*, *numTables* should be consider arguments.

numStudents 16

Instructor: Dr. Simina Fluture

capacity	**12**
numSeats	**3**

Closely follow the story and the given implementation details.

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.

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

Follow the story closely and cover the requirements of the project's description. Besides the synchronization details provided there are other synchronization aspects that need to be covered. You can use synchronized methods or additional synchronized blocks. Do NOT use busy waiting. If a thread needs to wait, it must wait on an object (class object or notification object).

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.

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);
    }
```

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

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);
}
```

Instructor: Dr. Simina Fluture

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

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

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

Tips:

-If you run into some synchronization issue(s), 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: Fluture_Simina_CS344-715_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.