CSCI 344-715 SP19 PROJECT 1: Due: April 16

Tourists go to Ellis Island to learn about the beginnings of America, and sometimes try to find out about their ancestors who immigrated to this country.

Every half hour a documentary movie is presented. If the movie is in session, visitors **wait** in the lobby of the movie theatre (use a **different object** for each visitor, similar to the **rwcv.java** implementation). When the previous session ends (**signaled** by a clock), all waiting visitors enter the room in order, and take one of the available seats.

If there are no free seats, visitors leave the room, and will **wait** to see the next presentation. Once the movie starts, no visitor can enter and disturb it. He/she will **wait** for the time of the next presentation.

When the movie ends, a speaker will give a short talk to the present audience. Once he is done he will announce to the audience that they are free to leave (use **notifyAll**). However, as a reward, he would like to give sets of party_tickets for a future movie. The audience will gather into groups of party_tickets size (use one **object for each group**). The speaker will call each group at once and distribute the tickets. After that, the speaker will **wait** for the end of the next movie when (s)he will have the chance to give another talk.

Next the visitors will browse around for a while and eventually leave the theater. They leave in no specific order, however the traffic in and out through the theatre door has to be **synchronized**.

In order to keep track of the time, we need an additional thread, named *clock*. The *clock* will **signal** the start time of the next movie and the end time of the previous movie, (between sessions the clock will sleep for a fixed time).

Some of the visitors are so impressed with the presentation that they will want to see it one more time, (randomly with a probability of 75% decide if a visitor is to see the movie one more time).

Develop a monitor synchronization of the three types of threads: speaker, visitors, *clock* in the context of the problem. More detailed implementation requirements will be posted in a separate file.

Throughout the day, they are four movie presentations.

Initial values: numVisitors = 17
theaterCapacity = 8
party_ticket =3

The numVisitors should be an argument to the program.

Closely follow the story and the given implementation details.

Choose the 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 semesters' projects, a project should take somewhere between 50 seconds and at most 1 minute and ¾ 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 to make sure that mutual exclusion over shared variables is satisfied.
Do NOT use busy waiting. If a thread needs to wait, it must wait on an object (class object or notification object).
DO NOT use other synchronization tolls beside basic monitors created thru synchronized blocks, methods and the use of wait, notify and notifyAll.

The 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);
```

}

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 program flow, self-explanatory variable names, correct whitespace and indentations is 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.