CSCI 344-715, SM18
Lecturer: Dr. Simina Fluture

Project 1: DUE FRIDAY 5PM, JULY 20 (the very latest)

THE COMMUTE

Every weekday people commute to work. While some people are close to their work place others are not so fortunate. Let's consider the unlucky ones.

The commute consists of several steps:
First the commuter drives from home to New York City (sleep of random time)

At the George Washington Bridge the commuter has to pay the toll. There is one EZPass lane and one Cash lane. Depending on whether the commuter has an EZPass subscription or not, the commuter will choose the appropriate lane. When the commuter gets to the booth he will pay and will be able to continue (have each commuter block on a different object; use vectors to ensure a FCFS order)

Once in the city, the commuter will park the car in a Parking Garage. There are two attendants responsible for serving the incoming customers. If there is no attendant available, the customers will wait until one of the attendants becomes free (any access to shared variables or wait must be done using monitors)

Once the car is parked, the commuter gets into the subway station to take one of the subway trains that will bring him closest to the office. The subways' trains come periodically and stop for a fixed amount of time. There are two different lines, A and B. (for each line use a different notification object). When the expected train arrives, the commuter attempts to board. In the subway there are, already, a random number of commuters. Commuters will step in until the capacity of the subway is reached or the subway standing time elapses. (any shared variable or wait should be done using monitors).

When the subway train arrives at the destination (consider it the end of the line) all commuters get off (notifyAll) and walk the last blocks to their work place.

Develop a monitor that will synchronize the three types of threads, commuter, garrageAttendant, subwayTrain in the context of the problem.

The number of commuters should be read as a command line argument.

Default values:
numCommuters: 15
trainCapacity: 10

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.

CSCI 344-715, SM18
Lecturer: Dr. Simina Fluture

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);
}
```

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.

CSCI 344-715, SM18
Lecturer: Dr. Simina Fluture

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. I will create a column for it.