Halloween Parade and Puppet Show

Students from two different schools will participate in a Halloween Parade. The students from one school wear orange uniforms, and the other students wear green. Students arrive at the meeting place (simulated by sleep(random)) and group in formations of two green and one orange uniform. Once a group is formed, it will align at the starting place of the parade. (**for each group use one notification object**). Only students in groups may march in the parade. Once all groups are formed, they will by notified in the order in which they formed the groups. The students will start marching around the park's stadium. The marching takes a random time, however keep it close to the order of 20 minutes.

After each circle around the stadium, students separate from their groups and take a snack break (simulated by sleep(random)). Next, they will gather in front of the circus tent entrance for a Puppet show. The tent used for the show has a capacity of **numSeats**. Students are not in groups when they take a snack break or attend a Puppet show. If a show is already in session, students will wait (**have each student block on a different object – use a similar approach as in rwcv.java**). When the previous show session ends, the students that just watched the show will leave. Once the room is empty, students that are waiting to see the next show will then **all** be released (in FCFS order) by the staff member, enter the room and take one of the available seats. If there are no free seats, the remaining students will leave the tent and wander around the park for a while and return later on, attempting to see the next show.

Traffic through the entry has to be **synchronized**. When the show is about to start, the staff member will close the tent entrance, and no student can enter and disturb it. Note: the staff member is notified by the clock that it is time for the show to start or end. However, the students are notified by the staff member when they can enter/exit the tent.

There will be a total of four shows and students are anxious to see at least three of them. However, throughout the entire day there will be a total of six parades (one stadium circle) one hour apart, and students must participate in at least three parades; they must alternate between the parade participations and the Puppet shows. Depending on how long they take to circle the stadium, how long the breaks take, and the availability of seats in the tent, not all students will have the chance to see at least three shows.

Parades and show times

Parades times:	11:00AM, 12:00PM, 1:00PM, 2:00PM, 3:00PM, 4:00PM
Show times:	11:15AM, 12:45AM, 2:15PM, 3:45PM

Develop a monitor synchronization of the three types of threads*: green-uniform students*, orange-*uniform students*, and *staff member*.
Do NOT use busy waiting. If a thread needs to wait, it must wait on an object (class object or notification object).

Further you will need to add a new thread *clock* that will keep track of the parade times and the show times. This specific thread doesn't need to block using wait().
After the day ends give a report of what show each student attended, at what time and how many times he/she participated in the parade.
Use only basic monitors as the ones discussed in class (synchronized methods, synchronized blocks, wait(), notify and notifyAll) and NO other synchronization tools like locks, concurrent collections…..or even volatile variables.

The command line/arguments passed to the program are: numGreen, numOrange.

Default Values:

numGreen = 14
numOrange = 7
numSeat = 6 (do not hard code this value)

Use the age( ) method and appropriate println statements to show how synchronization works. It is important to have print statements before, inside, and after critical sections. State clearly what the current thread executing the print statement is doing. Also, be sure to include the name of the thread and a timestamp relative to the start time of the program.

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.

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_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.