CS340 – Project 1 – SP19

Due date: April, Friday 19 with no submission penalty until Tuesday, April 23

The project must be done individually with no other source(s). No exceptions.

Read the Academic Integrity link one more time.

Through its implementation, this project will familiarize you with the creation and execution of threads, and with the use of the Thread's 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( ), and maybe synchronized methods.

In synchronizing threads, DO NOT use any semaphores. DO NOT use wait( ), notify( ) or notifyAll();

Tourists go to Ellis Island to learn about the beginnings of America, and sometimes to 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 ***busy wait*** in the lobby of the movie theatre. When the previous session ends (announced by a clock thread), visitors hurry up to enter the theater and take one of the available seats (simulate this by increasing the priority of the visitor; after a sleep of random time, set its priority to its default value; use **getPriority()** and **setPriority()** methods)

If there are no free seats, visitors leave the room, and will wait to see the next presentation (you can implement the mutual exclusion over the shared variable free seats by having it access through a synchronized method, or by declaring it as volatile. Make sure that if you use synchronized methods you never have a thread sleep inside of them). On their way out, they want to be polite at the door, so visitors yield to each other (use **yield()** twice). Once the movie starts, no visitor can enter and disturb it. While the movie is on visitors sleep for a long time (the only way to get out of sleep will be by an interrupt).

Your display should clearly show who got in the theater, who left, and who was able to see the current presentation.

When the movie ends (announced by a clock), a speaker will wake up the visitors by sending an interrupt to each visitor in the theater (use the **interrupt()** method. To check that it works, have a message part of the catch block).

Next, as a reward, the speaker gives out sets of party_tickets for a future movie. The audience will gather into groups of party_tickets size (maybe you want to use a Boolean

array/vector) and *busy wait* to be called. The speaker will call each group at once and distribute the tickets. After that, the speaker will *busy wait* for the end of the next movie.

Next the visitors will browse around for a while and eventually leave the theater. They leave in the order of their ID (let's say in the theater you have T3, T5, T2 and T1. The order in which they leave is T1, T2, T3, T5. Use the **isAlive()** and **join()** methods.

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).

Throughout the day, they are four movie presentations. The speaker will terminate after the visitors of the last presentation leave the theater. The visitors terminate after they leave the theater or if they didn't have the chance to see a presentation, when the museum closes.

Shortly after the four presentations end, the clock announces the closing of the museum, and terminates.

Develop a monitor synchronization of the three types of threads: speaker, visitors, *clock* in the context of the problem.

Initial values: numVisitors = 23

theaterCapacity = 5

party_ticket =3

The numVisitors should be an argument to the program.

Using the synchronization tools and techniques learned in class, synchronize the visitor threads, speaker and clock threads – in the context of the problem described above.

In order to simulate different actions you must pick reasonable intervals of random time. Make sure that the execution of the entire program is somewhere between 40 seconds and 90 seconds.

Guidelines

1. 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.

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

3. 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. Don't create packages.

4. The program asks you to create different types of threads. When there is more than one instance of the thread, no manual specification of each thread's activity is allowed (e.g. no Clerk1.checkCustomer()).

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.

11-12. DO NOT USE System.exit(0); the threads are supposed to terminate naturally by running to the end of their run methods.

13. The program should allow easy change to the default values. Do not hard-code your program.

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 (.zip)**

-Press Finish

PLEASE UPLOAD YOUR FILE ON BLACKBOARD ON THE CORRESPONDING COLUMN.

The project must be done individually with no use of other sources including Internet. No plagiarism, No cheating.