CSCI 340, Fall 2021
Instructor: Simina Fluture, PhD

340 Project 1 – Fall 2021
Due Date: November 30
Cutoff: December 3
(7 points off / for December 1,
additional 8 points off for December 2,
additional 10 points off for December 3)

The project must be done individually - no exceptions. Plagiarism is not accepted. Receiving and/or giving answers and code from/to other students enrolled in this or a previous semester, or a third party source including the Internet is academic dishonesty subject to standard University policies.
The objective of this project is to familiarize you with the creation and execution of threads using the Thread class methods. You should use, when necessary, the following methods to synchronize all threads: **run(), start(), currentThread(), getName(), join(), yield(), sleep(time), isAlive(), getPriority(), setPriority(), interrupt(), isInterrupted().**

The use of semaphores to synchronize threads is strictly DISALLOWED. Additionally, you are NOT PERMITTED to use any of the following: wait(), notify(), notifyAll(), the synchronized keyword (for methods or blocks), and any synchronized collections or synchronization tools that were not discussed in class.

You CAN, however, use the modifier **volatile** and the **AtomicInteger** and **AtomicBoolean** classes if you choose to.

Directions: Synchronize the voter, ID_checkers, kiosk_helper, scanning_helper threads in the context of the problem described below. The number of voters can be entered as a command line argument. Please refer to and read carefully the **project notes, tips and guidelines** before starting the project.

Today is Election Day

Voters are anxious to vote. Once arrived at the designated voting place ((simulated by sleep of random time) they will have to **busy wait** on line for their ID to be checked (you might use a boolean array/vector to keep track of their order). They will busy wait until one of the available ***ID_checkers*** lets them move on.

Once done with the ID check, voters can enter the voting kiosk line. The next step is to fill out a ballot form with their information. There are *num_k* kiosks. The voter will pick the kiosk with the shortest line and **wait** until they get in front so they can enter their information (simulate the wait using **sleep for a long time)**. Due to COVID a lot of regulations are in place. A ***kiosk_helper*** will control all this movement. He will inform the next voter when a kiosk becomes available and that he/she can move on (have the kiosk_helper interrupt the voter). When a voter is done completing his/her ballot (and usually it takes some time – simulate by sleep of random time), he/she will let the helper know and the helper will **interrupt** the next voter on that specific line so that he/she can move on (Use **interrupt()** and **isInterrupted( )**).

Finally, the last step. Now, the voters will rush to the room with scanning machines. (simulate this by increasing their priority. Use **getPriority()**, **setPriority()**, sleep(random time). After the voter has increased its priority, (s)he will sleep a random time and as soon as (s)he wakes up make sure you reset his/her priority back to the normal value. Next voters will **busy_wait** for the **scanning helper** to allow them to use the scanning machines. There are *num_sm* scanning machines. Once all scanners are available, the **scanning_helper** will let *num_sm* to move on. However, the voters get nervous and slowdown a bit. Each of them will yield (implement this by using **yield()** two times). Finally, they will scan their ballot (simulate by sleep of random time) and leave the scanning room. They are ready to leave the building with a very patriotic feeling that they exercised they right to vote and maybe tomorrow will be a better day. Leaving will be done in a specific order in groups. Voters that were at the same time in the scanning_room will join the voter with the highest id (((use **join()** and **isAlive()**).

For example if at a given time in the scanning room they were v0, v2, v12, v17, then v0,v2,v12 will join v17. V17 will leave first.

After all the voters leave the helpers will be able to leave as well.

Your program should implement the types of threads:

The number of voters should be read as a command line argument: *e.g.* –v <int> with a default value of 20.

Default values:
num_voters = 20
num_ID_checkers = 3
num_k = 3
num_sm = 4

TO REITERATE, you are not permitted to use monitors, semaphores, collections or any other synchronization tool if they were not discussed in class. You can, however, use the **volatile** modifier and the classes **AtomicInteger** and **AtomicBoolean**.

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 method is contained in the main thread. All other thread classes must be manually created 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). DO NOT create packages.**

4. The project asks that you create different types of threads. **No manual specification of each thread's activity is allowed (e.g. no Student5.goThroughTheDoor())**

5. **Add the following lines to all the threads you instantiate:**

```
public static long time = System.currentTimeMillis();


public void msg(String m) {
    System.out.println("["+(System.currentTimeMillis()-time)+"] "+getName()+": "+m);
}
```

It is recommended that you initialize the time at the beginning of the main method, so that it is unique to all threads.

6. There should be output messages that describe how the threads are executing. Whenever you want to print something from a thread use: msg("some message about what action is simulated");

7. NAME YOUR THREADS. 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. thread.sleep() is not busy wait. while (expr) {..} is busy wait.
Interrupting a thread should be done ONLY on the simulation mentioned in the project and not as a way of substituting busy waiting.

10. FCFS should be implemented in a queue or other data structure.

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

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

15. 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 2 minutes to run and complete. Set the time in such way that you don't create only exceptional situations.

CSCI 340, Fall 2021
Instructor: Simina Fluture, PhD

A helpful tip:
-If you run into some synchronization issues, and don't know which thread or threads are causing it, press F11 (in Eclipse) which will run the program in debug mode. You will clearly see the thread names in the debug perspective.

Setting up project/Submission:
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
Archive all your source files (NOT CLASS FILES) in a .zip format (NOT .rar)

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.