CSCI 340, FL22 – p2
Instructor: Simina Fluture, PhD

340 Project 2 – FALL 2022
The project must be done individually. No exceptions.

You have the following choices:
1. You can submit a pseudo-code implementation:
Due Date: Thursday, Dec. 8; end of the day (no late submission allowed)

OR

2. You can submit a java-code implementation;
Due Date: Sunday, Dec. 11; end of the day (no late submission allowed)

DO NOT SUBMIT BOTH TYPES OF IMPLEMENTATION: EITHER YOU SUBMIT THE PSEUDO-CODE, OR YOU SUBMIT THE JAVA-CODE.

Directions: You are asked to synchronize the threads of the following story (teacher, student) using semaphores and operations on semaphores. Do NOT use busy waiting or any other synchronization tools besides the methods (specified below) of the semaphore class. Please refer to and read the project notes carefully, tips and guidelines before starting the project.

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 source including the Internet is academic dishonesty subject to standard University policies.

Halloween

Students commute to school very excited in the anticipation of Halloween fun (*simulated by sleep of random time*). The teacher **waits** for them in the classroom. Once **all** students are in (use a counter), the teacher will give a brief lecture. Students will **wait** for the lecture to end.

As soon as the lecture ends, they will rush to line up and **wait** for the candies. The teacher will give each student a small bag of candies. When done, the teacher terminates.

Next it is time for trick-or-treating in the neighborhood. Based on their themes, there will be numGroup groups created. Each student will generate an integer between 1 and numGroup to determine the group he is in (e.g, whom s/he will group with). The students will also display the group they belong to.

The group of students will **select** a house to trick-or-treat (use semaphores). There are *numHouse* houses (this number might be less, equal or greater than the number of groups)

Each student will pick a random number of candies (number between 1 and 10). In the end, students of each group will compute and display its average number of candies (total of candies/number of students in the group). The group with the largest average of candies will be in 1st place.

Students will terminate in the order of their ranking. Students in first place will terminate first, next the ones in second place and so on. (make sure you use semaphores).

CSCI 340, FL22 – p2
Instructor: Simina Fluture, PhD

Default values

numStudents = 20
numGroups = 3

1. Pseudo-code implementation

You are asked to synchronize the threads of the story using semaphores and operations on semaphores. Do NOT use busy waiting. Any **wait** must be implemented using P(semaphores). Any shared variable must be protected by a mutex semaphore such that Mutual Exclusion is implemented.

Use pseudo-code similar to the one used in class (NOT java pseudo-code).
Mention the operations that can be simulated by a fixed or random sleep_time. Mention possible data structures that you might use in your implementation.
Your documentation should be clear and extensive.
Give the type and initial value for each semaphore. Explain the reason for each semaphore that you used in your implementation (who will do P and who will do V on that semaphore).

Discuss the possible flows of your implementation. Deadlock is not allowed.

As submission you can have everything written in one (docx, pdf or txt file) or you can have different files (in the mentioned types) for each class. The files' names should contain your full name.

2. Java-code implementation

Using Java programming, synchronize the threads, in the context of the problem. Closely follow the implementation requirements. The synchronization should be implemented through Java semaphores and operations on semaphores (acquire and release)

Keep in mind that semaphores are shared variables (should be declared as static)

For semaphore constructors, use ONLY:

Semaphore(int permits, boolean fair)
Creates a Semaphore with the given number of permits and the given fairness setting.

In methods use ONLY: acquire(), release();

You can also use:

getQueueLength()

Returns an estimate of the number of threads waiting to acquire.

DO NOT USE ANY OF THE OTHER METHODS of the semaphore's class, besides the ones mentioned above.
Any **wait** must be implemented using P(semaphores) (acquire).
Any shared variable must be protected by a mutex semaphore such that Mutual Exclusion is implemented.
Document your project and explain the purpose and the initialization of each semaphore.

DO NOT use synchronized methods or blocks, Do NOT use wait(), notify() or notifyAll() as monitor methods. Whenever a synchronization issue can be resolved use semaphores and not a different type of implementation (no collections, no more atomic classes)

CSCI 340, FL22 – p2
Instructor: Simina Fluture, PhD

Use appropriate System.out.println() statements to reflect the time of each particular action done by a specific thread. This is necessary for us to observe how the synchronization is working.
Submission similar to project1. Name your project: YourLastname_Firstname_CS340_p2
Upload it on Blackboard.

- 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.

- 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.

- The project asks that you create different types of threads. There is more than one instance of a thread. No manual specification of each thread's activity is allowed (e.g. no Passenger5.goThroughTheDoor())

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

  NAME YOUR THREADS. 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 the program, self-explanatory variable names, correct whitespace and indentations are required.

Setting up (java) 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 P is the current project number.

PLEASE ZIP ONLY THE JAVA FILES (SOURCE FILES). PLEASE UPLOAD YOUR FILE ON BLACKBOARD IN THE CORRESPONDING COLUMN.