PROJECT 1 – SUMMER 2025
DUE DATE: THURSDAY, July 17
NO LATE SUBMISSION WILL BE CONSIDERED.

The project must be done individually - no exceptions. Receiving and/or giving answers and code from/to other people enrolled in this or a previous semester, or a third-party source including the Internet is academic dishonesty subject to standard University policies. Just changing the name of variables and the methods will not make the project yours.

Questions regarding your implementation will be given in the midterm and/or final exam. Based on the correctness of your answers, I will assign or not a grade to your project.

If I judge that your performance on projects does not seem consistent with your previous work or there is some aspect of your work that is suspect, I may require you to meet with me or I will ask you to write similar code to the one in the project or to answer to questions related to your own implementation. Your performance in that session will determine your examination grade. This is a necessary process to maintain the integrity of the course grades.

Directions: Synchronize the students, professor and time type of threads in the context of the problem described below. Please refer to and read carefully the project notes, tips and guidelines before starting the project.

Summer Classes

Students take summer classes. The students commute to college (*simulated by sleep of random time*) in different ways. One professor teaches two classes CSCI240 followed by CSCI344 let's say at 10:00AM and 1:00PM. Each student will attempt to take these two classes. Classrooms have a capacity of *num_people.*

Once arrived at the college, if the morning class didn't start yet and there are still available seats, students will take a seat, have a chat (sleep of random time) and after that **wait** for the professor to get in the class and start teaching. If a student couldn't attend the class for one of the mentioned reasons (either s(he) came too late or there are no available seats), s(he) will go to the cafeteria take a seat at a table (how is explained later on). After the break, they will attempt to sit in the second class.

The professor waits for the starting time of the class (the timer will notify her/him when that time occurs). Next the professor enters the classroom, sets up the computer and projector (*sleep of random time)* and signals students that (s)he will start lecturing. Students will get ready (sleep of random time) and **wait** for the ending time of the class. The professor will start teaching (sleep of random time) but since all students now are on their phones, (s)he will also **wait** for the class to end. The timer will let the professor know when the class is about to end. When the class ends, the professor will notify all students. Next the professor will **wait** for the starting of the next class.

Next students will rush to the cafeteria to take a break. At the cafeteria there are *numTables* with *numSeats*. As they arrive the students will take a seat at a table. The tables will be taken one by one. (Students who took a seat at the same table should block on the same object).

Keep in mind that at the cafeteria there might already be some students who missed the first class.

Ten minutes before the starting time of the second class the timer will notify students, table by table in the order in which these tables have been taken by the students in the cafeteria. Next it will signal the starting time of the second class.
Everything repeats.

Note: if some students cannot enter the second classroom due to the limit capacity, they will leave and terminate.

After the second class ends it is time for office hours. Students will decide in a random way if they want to stay for the office hours. (generate a random number between 1 and 10. If < 4 consider waiting for the office hours). A student who stays for office hours will get on line. (it will block on its own object). The professor will **wait** until all the decisions are taken and students are on line (you might want to use a counter). Next the professor will answer students' questions in a FCFS (implement something similar to rwcv).

Once a student is done with his/her questions she/he will just **wait** for the entire office hours session to end. After the professor answers to each (sleep of random time) and every student, (s)he will end the session and leave. Students will leave as well (use a platoon policy).

Note: the timer should keep track of time:

1st class start time
1st class end time
10 minutes before 2nd class
2nd class start time
2nd class end time

The interval of time can be simulated by sleep of fixed time.

Default values:
numPeople = 15
numStudents = 20
numTables = 4
numSeats =3

All types of threads must be started by the main thread at about the same time, not in a specific order. All that the main thread should do is create and start the threads. Otherwise, what a specific type of thread must do needs to be implemented or called from inside of its run method.

Thread types: student
professor
timer

Develop a monitor(s) implementation that will synchronize the threads in the context of the story described above. Closely follow the details of the story and the synchronization requirements. Besides the synchronization details/requirements 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. Even if you prefer, synchronized blocks, make sure you have at least two synchronized methods.

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 ran by 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.

Do NOT use busy waiting. If a thread needs to wait, it must wait on an object (class object or notification object).

Use only basic monitors as the ones discussed in class (synchronized methods, synchronized blocks, wait(), notify() and notifyAll()). Use NO other synchronization tools like locks, concurrent collections, semaphores, or even volatile variables.

Between major events make sure you insert a sleep of random time. Also make sure you give the information necessary to understand what that event is. For example one detailed display should show what students are seated at the same table. You should NOT have sleep inside a monitor (the thread will not release the lock and the concurrency is cut).

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 8 seconds and 15 seconds to run and complete. Set the time in such a way so you don't create only exceptional situations.

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);
}
```

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.

Design an OOP program. The run method should contain a number of methods. 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 are required. **Headers** with information about class name, last modified time, author… is mandatory for each java file.

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

Zip only the source files (NOT .rar) .Upload the project on Brightspace by its due time.