CSCI 340, Fall 2020
Instructor: Simina Fluture, PhD

# Project 1 – FALL 2020

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 students and teacher threads in the context of the problem described below. The number of students can be entered as a command line argument. Please refer to and read carefully the project notes, tips and guidelines before starting the project.

## A SCHOOL DAY DURING COVID

Students at PS1111 are following a hybrid learning regimen. During this COVID time they have to follow strict rules. After a student wakes up and gets ready for a new school day, (s)he will complete a Health Questionnaire (simulate this using a sleep of random time). Next s(he) will commute to school. (sleep of random time).

Once arrived at school, student(s) will **(busy)wait** in the schoolyard to be called by the teacher. Once called by the teacher, before entering the classroom, students must wash their hands. so they will head to the restrooms. There are two restrooms, one for "girls" and one for "boys." The capacity of the bathroom is equal to two. You can decide if a student is a boy or a girl using a random number or you can consider that students with an odd id are boys while the others are girls. If a bathroom is already taken, the student takes a break (use **yield()** three times) and later on (s)he will wait again (use **busy waiting**) for the bathroom to become available. Students will use the bathroom in a **First Come First Serve** basis (you can use a Boolean array/vectors)

Next, some students are very eager to learn new things and they will rush to get to the classroom (simulate this by increasing their priority. Use **getPriority()**, **setPriority(), sleep(random time)**. After the student 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). On the other hand, the other students are not in such a rush.

By the time a student gets to class, if the class is already in session, the student(s) will leave for a while (**sleep of random time**) and walk around the campus and come back later on. If the class is not in session yet, student(s) will **(busy) wait** for the teacher to arrive and enter the auditorium.

Once the class is in session, students will immediately get bored and fall asleep (**simulate this using a sleep of a long time**). The teacher will teach (simulate it using **sleep for a fixed interval of time**) and when done, he will let the students know that the class ended by interrupting them (use **interrupt()** and **isInterrupted()**. Make sure that in the student code, as part of a catch block you have a println that will show that the student has been interrupted). Student(s) will leave the classroom and hurry to have some fun between classes (sleep of random time).

Once having arrived at the school the teacher will let students in. During the day, he will teach four classes during 5 periods. Each class takes a fixed amount of the time period. Between any two classes there is a break. Between the second and third class there is an office hour (the 5th period).

The school closes after the four classes end. At the end of school day, students leave the school. Each student will join another student; they will leave in decreasing order of their name or their ID.

For example: student 1 joins student 2, student 2 joins student 3, … student N-1 joins student N (use **join()** and **isAlive()**). The teacher will join the last student to leave and after that he will terminate as well.

A daily report with information about what classes and when each student attended throughout the day must be displayed. It can be displayed by the Teacher before terminating or in the main method.

Something like:

Student Name   Total Number of attended classes   Class Name (or number)   Period number.

-------------------------------------------------------------------------------------------

Your program should implement two types of threads:

**Students** (there are many students)
**Teacher** (there is only 1 teacher)

The number of students should be read as command line arguments: *e.g.* –s <int> with default values of 13.

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. There is more than one instance of a thread. **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.

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.

12. A command line argument must be implemented to allow changes to the **nStudent** variable.

13. Javadoc is not required. Proper basic commenting explaining the flow of the program, selfexplanatory variable names, correct whitespace and indentations are required.

14. 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.

**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.