Team Members

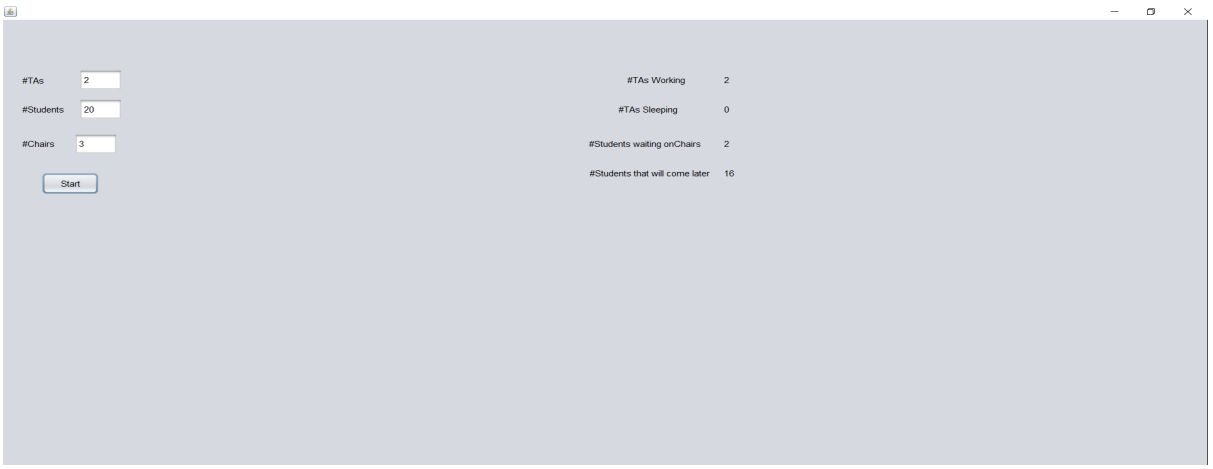
Nama	ID	JOP
هاجر عبدالله محمود محمد	202001016	Multithreading Correct usage of threads, and synchronization mechanisms
يوسف محمد سيد محمد	202001105	GUI with correct I/O and Thread communication or realtime update
هايدي خالد محمد عبدالحميد	202001026	GUI with correct I/O and Thread communication or realtime update
محمد ايمن محمد باهي	201901003	Documentation (1)
يحيى نبيل حسن سعيد الابراشى	202001063	Multithreading Correct usage of threads, and synchronization mechanisms
أسماء رياض علي النجار	202000133	GUI with correct I/O and Thread communication or realtime update
يوسف رضا محمد أبوسريع	202001089	Documentation (1)

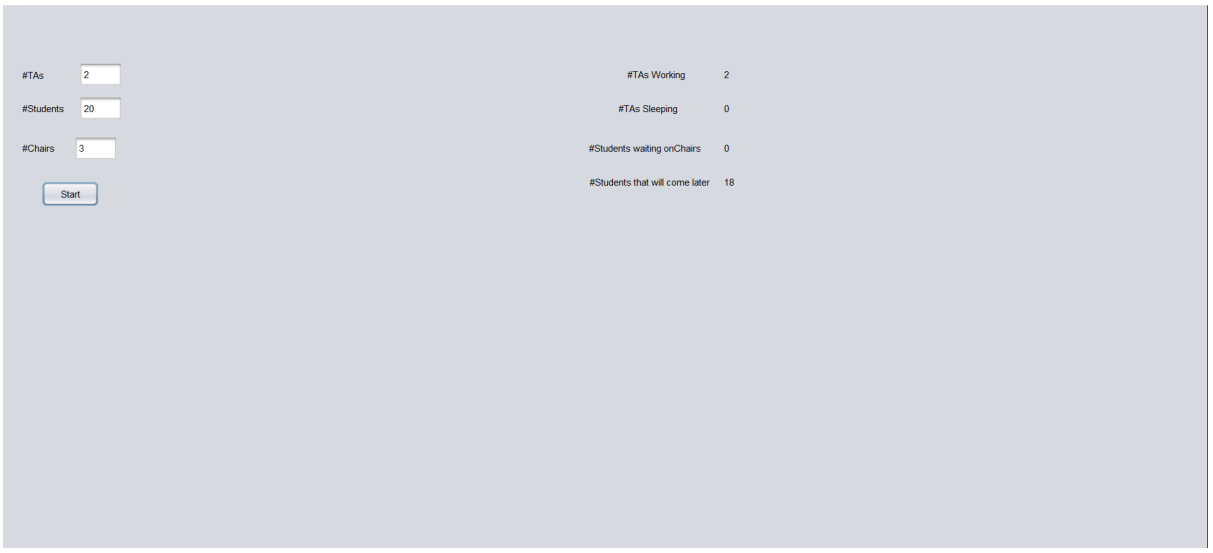
Project Description

Project 1: The Sleeping Teaching Assistant:

A university computer science department has a teaching assistant (TA) who helps undergraduate students with their programming assignments during regular office hours. The TA ’s office is rather small and has room for only one desk with a chair and computer. There are three chairs in the hallway outside the office where students can sit and wait if the TA is currently helping another student. When there are no students who need help during office hours, the TA

sits at the desk and takes a nap. If a student arrives during office hours and finds the TA sleeping, the student must awaken the TA to ask for help. If a student arrives and finds theTA currently helping another student, the student sits on one of the chairs in the hallway and waits. If no chairs are available, the student will come back at a later time.Using JAVA threads, mutex locks, and semaphores, implement a solution that coordinates the activities of the TA and the students. The number of disks in the TA's room (aka no. of TAs), number of chair for waiting students and number of students that have questions should be provided as an input to your program.





1-Test Case

IF no avaliable students come to visit the TA and the TA will check the hallway outside his office, none of the students are seated and waiting for him.

The TA will sleep in his office.

2-Test Case

When a student arrives at the TA’s office and finds the TA sleeping,then the student will awaken the TA and ask for help. When the TA assists the student,when the TA finishes helping one student, he will check if there is any other student waiting in the hallway. If yes, he will help the next student and if not, TA goes back to sleeping and TA's semaphore becomes 1 and awaits student's semaphore.

3-Test Case

When a student arrives while the TA.
IF the TA is busy,the student will have to wait seated outside in the hallway until the TA is done with his session. When the TA completes his session, the student seated outside will be called in by the TA for a review session.Once all students have finished their sessions and left the TA’s office, the TA will go back to sleep after making sure no students are waiting.

4-Test Case

When a student arrives while the TA is busy in a review session, and all the seats in the hallway are occupied. Then, students will have to leave the hallway and come back later. When the student comes back, eventually, and there is a seat available, he will take a seat and wait for his turn with the TA

Total Threads.

Main Thread:

- The `main` method (not provided in the code snippet) is likely the entry point for the program.
- The `main` method would create instances of the `TA` class, start threads, and possibly perform other tasks.
- The `main` method would run in the main thread.

TA Thread:

- Each instance of the `TA` class is associated with a thread because the class implements the `Runnable` interface.
- The `run` method of the `TA` class contains the logic executed by the thread associated with an instance of `TA`.
- Multiple instances of the `TA` class can be created, each running in its own thread

`Student` class, there is a single thread associated with each instance of the class. The class implements the `Runnable` interface, and its behavior is defined in the `run` methodTotal Number of Threads:

- The total number of threads used depends on how many instances of the `TA`,`Student` class are created and started in the `main` method.

Code Documentation

```
class Student {
    private Thread t;

    static class Student implements Runnable {
        private int id;
        public Student(int id) {
            this.id = id;
        }
        public void run() {
            try {
                // Check if there is an available chair in the hallway
                if (chairMutex.tryAcquire()) {
                    // If yes, student waits
                    waitingStudents++;
                    // Notify TA
                    student.release();
                    // Acquire TA's attention
                    ta.acquire();
                    // Ask questions
                    // Release chair
                    chairMutex.release();
                    // Simulate time taken to ask questions
                    Thread.sleep(1000);
```

```
                // Simulate TA answering questions
                studentsWithQuestions--;
                // Notify TA that student is done
                student.release();
            } else {
                // If no chair available, student comes back later
            }
        }
    }
}
```

```
static class TA implements Runnable {
        private Thread t;
        public void run() {
            try {
                while (studentsWithQuestions > 0) {
                    // Wait for student to arrive
                    student.acquire();
                    // Wake up TA
                    taMutex.acquire();
                    // Check if there are waiting students
                    if (waitingStudents > 0) {
                        // TA helps the student
                        waitingStudents--;
                        // Notify student
                        ta.release();
                        Thread.sleep(2000);
                    } else {
                        // No students waiting, TA can take a nap
                        // Release TA's room
                        taMutex.release();
                        // Notify student
                        ta.release();
```

```
                    // Wait for next student
                    student.acquire();
                    // Wake up TA
                    taMutex.acquire();
                    // TA helps the student
                    waitingStudents--;
                    // Notify student
                    ta.release();
                    // Simulate time taken to help the student
                    Thread.sleep(2000);
                }
                // Release TA's room
                taMutex.release();
            }
        }
    }
}
```