جامعة حلوان HELWAN UNIVERSITY	Operating Systems 2 (Fall 2023) Project Discussion	كلية الحاسبات والذكاء الإصطناعي Faculty of Computers & Artificial Intelligence

Project Number: (1)

Project Name: Sleeping Teaching Assistant

. Description:
The Sleeping Teaching Assistant project addresses the management of interactions between a Teaching Assistant (TA) and students within a university computer science department. Developed in Java, utilizing threads, mutex locks, and semaphores, the system ensures synchronized coordination. The TA aids students with programming assignments during regular office hours, and specific protocols guide student interactions based on the TA's current state.

```
Input:329
Output: Student 1 took a Seat. Waiting Students: 0, Left Students: 0
Student 1 is Asking TA. Waiting Students: 1, Left Students: 0
The TA is Answering a student. Waiting Students: 1, Left Students: 0
Student 2 took a Seat. Waiting Students: 1, Left Students: 0
Student 2 is Asking TA. Waiting Students: 2, Left Students: 0
Student 3 took a Seat. Waiting Students: 2, Left Students: 0
Student 3 is Asking TA. Waiting Students: 3, Left Students: 0
The TA is Answering a student. Waiting Students: 2, Left Students: 0
Student 4 took a Seat. Waiting Students: 2, Left Students: 0
Student 4 is Asking TA. Waiting Students: 3, Left Students: 0
Student 5 took a Seat. Waiting Students: 2, Left Students: 0
Student 5 is Asking TA. Waiting Students: 3, Left Students: 0
There are no free seats for student 6. Waiting Students: 2, Left Students: 0
The TA is Answering a student. Waiting Students: 2, Left Students: 1
Student 7 took a Seat. Waiting Students: 1, Left Students: 1
Student 7 is Asking TA. Waiting Students: 2, Left Students: 1
There are no free seats for student 8. Waiting Students: 1, Left Students: 1
The TA is Answering a student. Waiting Students: 1, Left Students: 2
Student 9 took a Seat. Waiting Students: 1, Left Students: 2
Student 9 is Asking TA. Waiting Students: 2, Left Students: 2
```

```
The TA is Answering a student. Waiting Students: 1, Left Students: 2
The TA is Answering a student. Waiting Students: 0, Left Students: 2
The TA is Answering a student. Waiting Students: 0, Left Students: 2
```

. What We Have Actually Done:
- Implemented Java threads to model the TA and students.
- Utilized mutex locks to control access to shared resources and prevent race conditions.
- Employed semaphores for synchronization, managing shared resources like chairs in the hallway.
- Developed logic to handle TA states (sleeping, helping) and student interactions based on the TA's availability.

How to solve:
To address the potential for deadlock in this code, you can rearrange the order in which the semaphores are acquired and released to avoid circular dependencies. Specifically, you should ensure that semaphores are released in the reverse order in which they are acquired.

In your `Student` class, you acquire `Access` before `Student`, and in your `Teacher` class, you acquire `Student` before releasing `Access`. This can lead to a deadlock if a `Student` holds the `Access` semaphore and is waiting for the `TA` semaphore, while the `Teacher` is holding the `Student` semaphore and waiting for the `Access` semaphore.

To avoid this, consider the following changes:

1. In the `Student` class, move the `Student.release();` statement outside the `Access` semaphore block:

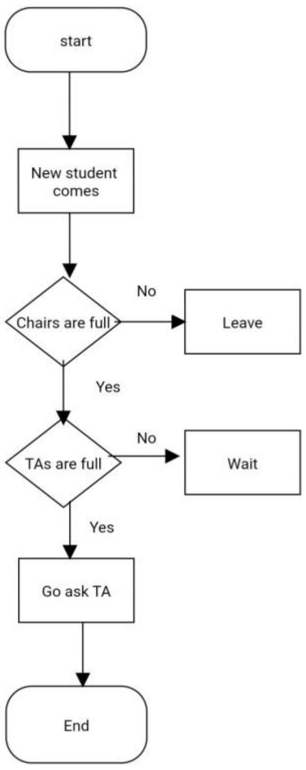
```java
if (NumOfEmptyChairs > 0) {
   System.out.println("Student " + this.A + " took a Seat. Waiting Students: " +
waitingStudentsCount + ", Left Students: " + leftStudentsCount);
   NumOfEmptyChairs--;
   Access.release();
   waitingStudentsCount++; // Increment the waiting students count
   try {
      TA.acquire();
      notAsk = false;
      this.Get_AskTA();
```

```java
    } catch (InterruptedException ex) {
    }
    Student.release(); // Move this line outside the Access semaphore block
} else {
    // ...
}
```

2. In the `Teacher` class, move the `Access.release();` statement outside the `TA` semaphore block:

```java
try {
    Student.acquire();
} catch (InterruptedException ex) {
    // Handle InterruptedException
}
NumOfEmptyChairs++;
TA.release();
Student.release(); // Move this line outside the TA semaphore block
Access.release();
AskTA();
```

This adjustment ensures that semaphores are released in a consistent order, reducing the likelihood of a deadlock scenario. Keep in mind that deadlock situations can be complex, and it's essential to carefully review the synchronization logic in your code to prevent such issues.
3.Flow chart diagram:



. Code Documentation

Pieces:

- `TAThread.java`:
   - Represents the TA as a thread.
   - Manages TA activities, state transitions, and synchronization using mutex locks and semaphores.

-`StudentThread.java`:
   - Represents a student as a thread.
   - Handles student behavior, including waiting, awakening the TA, and managing chair access using semaphores.

- `SemaphoreManager.java`:
   - Manages semaphores for synchronization.

  - Controls access to shared resources, such as chairs, ensuring coordinated interactions among threads.

- `MutexLocks.java`:
  - Implements mutex locks to prevent race conditions.
  - Ensures exclusive access to critical sections of the code, like checking the TA's state.

Threads:

- TA Thread:
  - Manages the TA's activities, transitions between states, and synchronization with students.

- Student Threads:
  - Represent individual student interactions, including waiting, awakening the TA, and managing access to chairs.

Input Parameters:
- Number of TAs (disks)
- Number of chairs for waiting students
- Number of students with questions

Note:
  - Thread synchronization mechanisms prevent conflicts and ensure orderly interactions.
  - The program is customizable based on input parameters, accommodating various scenarios.

This documentation provides a thorough overview of the Sleeping Teaching Assistant project, including its purpose, the implemented solution, and detailed code documentation using Java. The individual pieces and threads are explained for clarity and understanding.

key points:

1. Semaphores:
  - `TA`, `Student`, and `Access` are semaphores used for synchronization between threads.
  - `TA` controls access to the Teaching Assistant.
  - `Student` is used for signaling when a student is waiting.
  - `Access` controls access to shared resources, specifically the number of empty chairs.

2. Counters:
   - `NumOfEmptyChairs`: Represents the number of empty chairs available for students to sit.
   - `waitingStudentsCount`: Counts the number of students currently waiting for their turn.
   - `leftStudentsCount`: Keeps track of the number of students who have left without getting assistance.

3. Student Class:
   - Each student is represented as a thread (`Student` class).
   - Students continuously check for available chairs. If a chair is available, they sit, release the `Access` semaphore, signal the TA using the `Student` semaphore, and wait for TA assistance.
   - If no chairs are available, the student leaves and increments `leftStudentsCount`.

4. Get_AskTA Method:
   - Represents the process of a student asking the TA for assistance.
   - After acquiring the TA semaphore, the student waits for 5 seconds (simulating TA assistance) and then releases the TA semaphore.

5. Teacher Class:
   - Represents the Teaching Assistant (TA).
   - The TA waits for a signal from a student (`Student` semaphore) indicating that a student is waiting.
   - Upon receiving the signal, the TA increments `NumOfEmptyChairs`, releases the `TA` semaphore, releases the `Access` semaphore, and then proceeds to assist the student for 5 seconds.

6. Main Method:
   - Initializes the number of chairs, TAs, and students based on user input.
   - Starts the TA thread and a number of student threads.
   - Each student is given a unique ID, starts, and then sleeps for 2000 milliseconds (2 seconds) before the next student starts.

7. **Run Method:**
   - The `run` method is overridden from the `Thread` class and contains the main logic of the `TASleeping` thread.
   - It takes user input for the number of chairs, TAs, and students.
   - Initializes the number of empty chairs and starts the TA thread.

- Creates and starts student threads, introducing a 2-second delay between the start of each student.

8. Sleeping Periods:
   - Throughout the code, there are sleep periods to simulate certain processes (e.g., waiting for assistance, TA answering a student).

This code simulates a scenario where students arrive, take seats if available, and interact with the TA for assistance. The TA assists students in a first-come, first-served manner. The semaphores and counters help manage access to shared resources and keep track of the state of the system.

Team Members[1]:

	Team Member ID	Team member name (**in Arabic**)	Grade
1	202000654	فيولا عريان يوسف بارح	
2	202000681	كيرلس فايز حلمي دكسيس	
3	202000234	جانو طارق محفوظ ميخائيل	
4	202000501	عبد الرحمن اسامة عبد البديع شعبان	
5	201900595	مارك ايليا لبيب شنودة	
6			
7			

Evaluation Criteria

General Criteria

Criteria		Grade
Multithreading (5)	No multithreading (2 out of 5)	……………………………………………………………………….
	Threads in serial (3 out of 5) Correct usage of and threads, synchronization mechanisms	
	Multithreading (4 or 5 out of 5) Correct usage of and threads, synchronization mechanisms	
GUI (2)	No GUI (0 out of 2)	……………………………………………………………………….
	GUI without thread communication or realtime update (1 out of 2)	
	GUI with correct I/O and Thread communication or realtime update (2 out of 2)	
Documentation (1)		

Understanding (2)		