University of Westminster
Department of Computer Science

7SENG007C Concurrent and Distributed Systems 2022/2023	
Module leader	Mr. Guhanathan Poravi.
Unit	Coursework
Weighting:	50%
Qualifying mark	40%
Description	Report
Learning Outcomes Covered in this Assignment:	LO1, LO2
Handed Out:	12th Feb 2024
Due Date	08th Apr 2024 by 1:00pm
Expected deliverables	Source code and any other resources
Method of Submission:	Electronic submission on BB via a provided link close to the submission time. The file you upload should have the following naming format: wNNNNNNNN.zip (where wNNNNNNN is your university ID login name)
Type of Feedback and Due Date:	Written feedback and marks 15 working days (3 weeks) after the submission deadline, the week starting Monday 1st of June 2023. Oral feedback will also be offered upon appointment with the module leader. All marks will remain provisional until formally agreed upon by an Assessment Board.

Copying and plagiarism

Any external sources utilized should be correctly referenced using a common referencing technique (e.g., the Harvard technique). For more details on referencing please visit https://www.westminster.ac.uk/current-students/studies/study-skills-and-training/research-skills/referencing-your-work .

Copying and plagiarism carry severe penalties. Please note that the University offers an online learning tutorial designed to help students understand and avoid plagiarism. This can be accessed by any student under My Organisation on Blackboard. The tab is labelled 'Avoiding Plagiarism'.

Penalty for Late Submission

If you submit your coursework late but within 24 hours or one working day of the specified deadline, 10 marks will be deducted from the final mark, to minimum of the pass mark (50%), as a penalty for late submission,. If you submit your coursework more than 24 hours or more than one working day after the specified deadline you will be given a mark of zero for the work in question unless a claim of Mitigating Circumstances has been submitted and accepted as valid.

It is recognised that on occasion, illness or a personal crisis can mean that you fail to submit a piece of work on time. In such cases you must inform the Faculty Registry Office in writing on a mitigating circumstances form, giving the reason for your late or non-submission. You must provide relevant documentary evidence with the form. This information will be reported to the relevant Assessment Board that will decide whether the mark of zero shall stand. For more detailed information regarding University Assessment Regulations, please refer to the following website: http://www.westminster.ac.uk/study/current-students/resources/academic-regulations.

Assignment Title: Multi-threaded Bank System Simulation

Objective:

To design and implement a simulated bank system using a programming language of your choice that demonstrates the use of multiple threads, synchronization mechanisms, monitor pattern for resource management, thread lifecycle management, thread groups, and ensures thread safety across various banking operations. (The assignment specification should be designed with Java in mind as it is the preferred programming language by the module leader. However, using other programming languages will not be a disadvantage).

Background:

In a bank system, multiple clients may attempt to perform various operations like deposits, withdrawals, and balance checks simultaneously. Implementing such a system using multi-threading can significantly increase efficiency but also requires careful management of shared resources to prevent data inconsistencies and ensure thread safety.

Requirements:

1. **System Design:**
    - Define a **BankAccount** class with attributes such as account number, account holder's name, and balance. Include methods for deposit, withdrawal, and balance check.
    - Implement a **Bank** class that manages multiple **BankAccount.** It should allow adding new accounts and fetching existing accounts by account number.
2. **Multi-threading Implementation:**
    - Use threads to simulate multiple clients performing operations (deposits, withdrawals, balance checks) on their bank accounts concurrently.
    - Implement a **Client** class that represents a banking client, which will perform operations on their bank account.
3. **Synchronization and Monitor:**

- Ensure that access to the **BankAccount** methods is synchronized to prevent data inconsistency. Use the monitor pattern to manage access to bank account resources.

4. **Thread Lifecycle Management:**
    - Demonstrate the use of thread lifecycle methods (start, run, wait, notify/notifyAll, join) within your simulation to handle various operational states of banking operations.
5. **Thread Group:**
    - Organize clients into thread groups based on criteria like VIP and regular clients. Manage these groups separately, demonstrating the use of thread group methods and properties.
6. **Thread Safety:**
    - Implement thread safety mechanisms to ensure that all banking operations (deposit, withdrawal, balance check) are safely executed without causing data corruption or inconsistency.

Tasks:

1. **Design and Implementation:**
    - Design the **BankAccount**, **Bank**, and **Client** classes as per the requirements.
    - Implement the bank system simulation with at least 10 concurrent clients performing various operations.
2. **Documentation:**
    - Document your design choices, particularly how you implemented synchronization, the monitor pattern, and ensured thread safety.
    - Explain how your implementation manages the thread lifecycle and utilizes thread groups.
3. **Testing:**
    - Write test cases to ensure your system works as expected under concurrent access. Demonstrate how your system handles synchronization, thread safety, and correct management of thread lifecycle and groups.
4. **Reflection:**
    - Reflect on the challenges faced during the implementation and how you overcame them.
    - Discuss any potential improvements or alternative approaches that could be taken.

Submission:

- Source code files for the Bank system simulation.
- A detailed report documenting your design, implementation choices, testing procedures, challenges faced, and reflections on the assignment.
- Test cases and results demonstrating the successful execution of your system under concurrent operations.

Marking Rubrics

1. Design and Implementation (40 marks)

- **BankAccount and Bank Class Design (10 marks):** Well-structured and encapsulated classes with appropriate attributes and methods.
- **Client and Multithreading Implementation (15 marks):** Effective use of threads to simulate client operations. Creativity in simulating realistic banking operations.
- **Synchronization and Monitor Usage (10 marks):** Correct implementation of synchronization and the monitor pattern to manage access to shared resources.
- **Thread Lifecycle and Group Management (5 marks):** Demonstrated understanding and application of thread lifecycle methods and effective use of thread groups.

2. Functionality and Correctness (25 marks)

- **Correct Execution of Banking Operations (10 marks):** All banking operations (deposit, withdrawal, balance check) work correctly under concurrent access.
- **Data Integrity and Thread Safety (10 marks):** No data inconsistencies or race conditions. Effective thread safety mechanisms in place.
- **Error Handling and Edge Cases (5 marks):** Proper handling of exceptional situations and edge cases (e.g., insufficient balance for withdrawal).

3. Testing (15 marks)

- **Test Case Design and Coverage (10 marks):** Comprehensive test cases covering various scenarios, including concurrent operations and edge cases.
- **Test Execution and Results (5 marks):** Clear presentation of test cases, execution, and results demonstrating the system's reliability and correctness.

4. Documentation and Reflection (20 marks)

- **System Design and Implementation Choices (5 marks):** Clear explanation of design choices, including class designs and concurrency mechanisms.
- **Challenges and Solutions (5 marks):** Insightful discussion on challenges faced during implementation and solutions applied.
- **Potential Improvements (5 marks):** Reflection on potential improvements or alternative approaches for the system.
- **Clarity and Organization (5 marks):** Well-organized documentation with clear headings, sections, and coherent explanations.

Tips for Success:

- Start early to have ample time for testing and refining your implementation.
- Pay close attention to the synchronization and thread safety aspects, as these are critical for a multi-threaded application.
- Make sure your documentation not only explains what your code does but also why certain design choices were made.
- Test thoroughly, not just for correctness but also for concurrency issues, which can be subtle and hard to detect.