|  |  |
| --- | --- |
| **6SENG006W Concurrent Programming**  **Coursework (2024/25)** | |
| **Module leader** | Guhanathan Poravi (P.Guganathan@westminster.ac.uk) |
| **Unit** | Coursework |
| **Weighting:** | 50% |
| **Qualifying mark** | 30% |
| **Description** | To develop:  A concurrent (multi-threaded) Java program to implement based off 3 given scenarios. |
| **Learning**  **Outcomes Covered in this Assignment:** | The coursework assesses learning outcomes:  LO1, LO2, LO3 & LO4. |
| **Handed Out:** | Week 08 |
| **Due Date** | **13:00, Thursday, 9th January, 2025** |
| **Expected deliverables** | Electronic files:   1. The Java source code. 2. Vodcast – demonstrating the programs, explaining, and justifying your chosen solution.   Source code files should be compressed into a single ZIP archive. The ZIP archive should be named using your surname & "CW", e.g.  "howells\_CW.zip" Vodcast submitted separately on Blackboard. |
| **Method of** | Online via Blackboard |
|  |  |
| **Submission:** |  |
| **Type of Feedback and Due Date:** | Verbal feedback in tutorials as the assessment progresses.  Electronic via module's Blackboard bulletin board.  Written feedback and marks 15 working days (3 weeks) after the submission deadline.  **All marks will remain provisional until formally agreed by an Assessment Board.** |
|  |  |

**Assessment regulations**

Refer to the “How you study” guide for postgraduate students for a clarification of how you are assessed, penalties and late submissions, what constitutes plagiarism etc.

**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, as a penalty for late submission, except for work which obtains a mark in the range 40 – 49%, in which case the mark will be capped at the pass mark (40%). 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 FST Registry in writing on a Mitigating Circumstances

(MC) 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**](http://www.westminster.ac.uk/study/current-students/resources/academic-regulations)

# Coursework Description

**Scenario 1: Total [20 Marks]**

Scenario: Coffee Shop Ordering System

Consider a coffee shop where customers place orders and baristas prepare coffee. In this scenario, we have multiple customers who can place orders simultaneously, and multiple baristas who prepare the coffee based on those orders. The shop has a limited number of orders it can handle at a time, ensuring that it does not become overwhelmed with too many orders.

Requirements and marks break down:

Shared Resource: A limited order queue that can only hold a certain number of orders at a time. [**5 Marks]**

Customer [**1 Mark]**: These place orders in the queue. If the queue is full, they must wait until there is space available to place their order. [implement / extend appropriate interfaces/class [**2 Marks**], implement appropriate procedures / functions [**2 Marks**]]

Barista [**1 Mark]**: These take orders from the queue to prepare the coffee. If the queue is empty, they must wait until there are orders available. **[implement / extend appropriate interfaces/class** [**2 Marks**], implement appropriate procedures / functions [**2 Marks**]]

Mutual: Ensure that only one customer or barista can access the order queue at a time to avoid race conditions and ensure consistency. **[5 Marks]**

**Identify and use an appropriate concurrency mechanism to ensure the code executes in a safe manner.**

A screenshot of a computer

Description automatically generated

**Scenario 2: Total [30 Marks]**

You are tasked with implementing a concurrent banking transaction system that handles money transfers between multiple bank accounts. Each account can only be accessed by one thread at a time, but the complication is that transfers between two accounts must lock both accounts in a safe order to prevent deadlock.

Requirements and marks break down:

1. **Transaction Safety [5 Marks]:** A transaction from Account A to Account B must ensure that both accounts are locked before transferring money. If two threads attempt to transfer money between the same pair of accounts (in either direction), one must wait until the other completes.

2. **Avoiding Deadlock [5 Marks]**: When multiple threads try to transfer money between different pairs of accounts, ensure that no deadlock occurs. For example, if Thread 1 is transferring money from Account A to Account B and Thread 2 is transferring from Account B to Account C, a potential deadlock could occur if a third thread is transferring money from Account C to Account A.

3. **Fair Access [5 Marks]**: If multiple threads are waiting to access the same account, they should be granted access in the order they arrived (first-come-first-served).

4. **Read-Write Operations [5 marks]**: Some operations, such as getBalance() or getTransactionHistory(), should not require locking the account as long as there are no ongoing transactions. Concurrent reads are allowed, but no read should interfere with an ongoing transaction.

5. **Multiple Transactions [5 Marks]:** The system should allow multiple transactions to occur simultaneously if they do not involve the same set of accounts.

6. **Transaction Reversal [5 Marks]:** The system should be able to safely reverse a transaction if an error occurs during the transfer, ensuring that the states of all involved accounts are consistent after the operation.

Example Scenario:

Consider the following threads:

Thread 1: Transfers $100 from Account 1 to Account 2.

Thread 2: Transfers $200 from Account 2 to Account 3.

Thread 3: Transfers $50 from Account 3 to Account 1.

Thread 4: Reads the balance of Account 1 and Account 3 concurrently.

In this situation:

Thread 1 should complete its transaction before Thread 2 can access Account 2.

Thread 3 should not start its transaction until both Thread 1 and Thread 2 are done.

Thread 4 can read the balances without blocking any ongoing transactions.

If a failure occurs, the system should be able to safely roll back any partial transfers and ensure the integrity of all accounts.

Implement a system that satisfies the above requirements, ensuring safe concurrent access, avoiding deadlocks, and providing consistent results for all transactions and queries.

**Identify and use an appropriate concurrency mechanism to ensure the code executes in a safe manner.**

A screenshot of a computer

Description automatically generated

**Scenario 3: Total [20 Marks]**

University has a shared bathroom stall. Multiple employees/Students need to use them throughout the day, but:

Requirements and marks break down:

* Only a fixed number of people can use it at once (limited stalls)
* We need to keep track of available stalls **[5 Marks]**
* Employees / Students need to wait if all stalls are occupied **[5 Marks]**
* As people finish using a stall they should leave, and other users should be able to enter and use. No stall should have more than one user at a time. [**5 Marks]**
* Simulate for a single floor to keep things simpler.
* Program should have **6-bathroom stalls** and **100 (Employees and Students)**

**[5 Marks]**

No, The Program should have **0-bathroom stalls** and -**10 (Employees and Students) Throw and handle the exception.**

A screenshot of a computer

Description automatically generated

**Vodcast: Total [30 Marks]**

Justify your choice of mechanism on why it is most suitable for this scenario elaborating why it is better than other similar mechanisms. Justification will be done in a short Vodcast where you will:

1. Execute your solutions demonstrating your work with appropriate outputs clearly displayed [**10 Marks]**
2. Explain your solutions **[5 Marks]** with justificationsfor your chosen concurrency control mechanisms to manage the concurrency issue of the requirements. Discussions may among others centre on concurrency, resource sharing, safety properties, liveness properties, **[15 Marks].** It is important your explanations are technical and much beyond surface explanations as marks will be reduced in each section if explanations use language not at the level expected for Level 6 (Year 3 students).

**Vodcast should be approximately 10 mins long.**

### Components to Submit

1. The Java source code for each scenario **Note** submit a zip folder with all 3 projects scenarios in an easily runnable state. That is your marker should not have to make changes to your code to make it run. Submit a full project structure from Netbeans IDE or an agreed IDE with your module leader.

**[70 MARKS]**

1. A vodcast demonstrating your applications and the output produced. Walk through the process giving detailed appropriately technical explanation and justification of your chosen concurrent mechanism to control access to the resource. Offer some justifications as to why the chosen concurrency mechanism

**[30 MARKS]**