**Assignment 2: Cinema Online Seat Reservation System Report**

**Student Name: BONG HONG JUN**

**Student Id: 22018519**

**1.0 Introduction**

This report provides a comprehensive explanation of the development and implementation of a Cinema Online Seat Reservation System. The system is designed to handle concurrent seat reservations for a cinema with three theatres, ensuring that no seat is double-booked. The report addresses three critical aspects: how the system blocks other customers from selecting the same seat(s), how to interpret the results to prove the system fulfills the requirements, and the discussion on whether careful design is required to avoid deadlock.

**2.0 Code Explanation and Implementation**

The Cinema Online Seat Reservation System is implemented in Java, leveraging concurrency features to handle up to 100 simultaneous customer reservation attempts. The system comprises several classes, including CinemaReservationSystem, Theatre, and Customer, each serving a specific role in the functionality.

**2.1 CinemaReservationSystem class**

The CinemaReservationSystem class initializes the theatres and manages the execution of customer reservation attempts using a thread pool. Three Theatre objects are created, each representing a theatre with 20 seats. The system utilizes an ExecutorService with a fixed thread pool of 10 threads to simulate concurrent customer reservations. Customers are generated with random theatre selections and seat choices, ranging from 1 to 3 seats.

A screenshot of a computer screen

Description automatically generatedA computer screen shot of a program

Description automatically generated

The generateRandomSeats method ensures that the seat selection is random, contributing to the realistic simulation of the system's operation.

A computer screen shot of a program code

Description automatically generated

**2.2 Theatre class**

The Theatre class is central to managing seat reservations. Each theatre uses a ReentrantLock to ensure that seat reservations are thread safe. When a customer attempts to reserve seats, the system locks the theatre to prevent other customers from making reservations simultaneously. The reserveSeats method checks if the requested seats are available. If they are, the seats are marked as reserved, and the reservation is confirmed; otherwise, the reservation attempt fails.

A screen shot of a computer

Description automatically generated

A computer screen shot of text

Description automatically generated

**2.3 Customer class**

The Customer class encapsulates the details of each customer, including their ID, chosen theatre, and selected seats. This class is used to represent individual reservation attempts in the system.

A screen shot of a computer program

Description automatically generated

**3.0 Ensuring Seat Exclusivity**

To ensure that no two customers can reserve the same seat simultaneously, the system employs a ‘ReentrantLock’ in the ‘Theatre’ class. This lock is integral to maintaining thread safety within the critical section of the code where seat reservations are processed.

The process begins when a customer attempts to reserve seats. At this point, the system locks the theatre, ensuring exclusive access to the seat reservation functionality. The system then verifies the availability of the requested seats If there are still seats available, the reservation is confirmed, and the seats are noted as reserved. Conversely, if the seats are already reserved, the reservation attempt fails, and an appropriate message is logged. Finally, the lock is released, allowing other threads to attempt reservations. This locking mechanism guarantees that while one customer is in the process of reserving seats, no other customer can access the reservation process for the same theatre.



A screen shot of a computer code

Description automatically generated A black screen with text

Description automatically generated

**4.0 Result Interpretation**

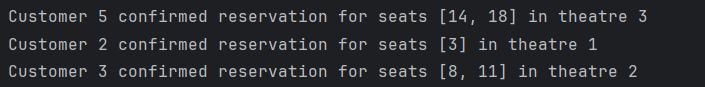
The output of the Cinema Online Seat Reservation System demonstrates the system's ability to handle concurrent seat reservations across multiple theatres effectively. Each customer's attempt to reserve specific seats is recorded, indicating whether the reservation was successful or failed. This section elaborates on how to interpret the results to prove that the system fulfills the requirements.

A screen shot of a computer

Description automatically generated

**4.1 Successful Reservations**

A successful reservation is indicated by a message stating that a customer has "confirmed reservation for seats" followed by the list of seat numbers and the theatre number. For example:



These messages confirm that the system can handle multiple customers reserving seats simultaneously without errors. The system correctly updates the status of the seats, ensuring they are marked as reserved and are no longer available for other customers. Ensuring that no two clients can reserve the same seat at the same time and upholding the integrity of the seat reservation process depend heavily on this feature.

**4.2 Failed Reservations**

A failed reservation is indicated by a message stating that a customer "failed to reserve seats" followed by the list of seat numbers and the theatre number. For example:







These messages occur when the requested seats are already reserved by another customer. The reservation is prevented, and the consumer is notified of the failure by the system, which verifies the availability of the seats. This mechanism ensures that the system maintains accurate records of seat reservations and avoids double-booking.

**4.3 Concurrent Reservation Handling**

**4.3.1 Seat Availability Consistency**

The mixed sequence of successful and failed reservations shows that multiple threads (representing different customers) are operating concurrently. The absence of any incorrect double-booking in the output validates the concurrency control mechanism’s effectiveness.





For instance, Customer 11 confirmed reservation for seats [3, 8, 2] in theatre 3, followed by Customer 21 failed to reserve seats [3] in theatre 3, proves that once Customer 11 reserves seat 3, it is no longer available for Customer 21.

**4.3.2 Distribution Across Theatres**

The output also demonstrates seat reservations across multiple theatres, further verifying that each theatre operates independently with its own set of seats.

A black background with white text

Description automatically generated

For instance,

Theatre 1: Customer 2 confirmed reservation for seats [3]

Theatre 2: Customer 3 confirmed reservation for seats [8, 11]

Theatre 3: Customer 9 confirmed reservation for seats [3]

This independence ensures that the locking mechanism does not interfere across theatres, which is crucial for the scalability and efficiency of the system.

**4.3.3 Distribution Across Theatres**

Additionally, the pattern of reservation attempts, and their corresponding success or failure reflects optimal resource utilization. Customers make reservations almost simultaneously, and the system manages these concurrent requests without deadlock, ensuring all resources (seats) are utilized efficiently without conflicts. The empirical validation of the system’s effectiveness is demonstrated through the high number of successful reservations juxtaposed with appropriate handling of failed attempts. This balance indicates that the system not only prevents double-booking but also maximizes the probability of successful reservations by handling conflicts efficiently.

**4.3.4 Summary**

In conclusion, the detailed interpretation of the output confirms that the Cinema Online Seat Reservation System fulfils the requirement of preventing multiple customers from reserving the same seat(s) concurrently. The resilience and dependability of the system in a multi-threaded context is demonstrated by its capacity to manage concurrent requests, guarantee data consistency, and preserve operational efficiency without deadlock.

**5.0 Deadlock Avoidance**

Deadlock is very big problem in concurrent system. It can bring down many potential threads which stops executing and keep on waiting for resources so that no more processing happens. These are some of the strategies used in designing this Cinema Online Seat Reservation System to prevent deadlocks and provide uninterrupted operation.

First, using a ‘ReentrantLock’ in this method is less prone to causing complex lock interactions as simply each theatre has its own ‘ReentrantLock’. Disabling each theatre lock therefor localizes concurrency control on a per-theatre scope. This design choice reduces the possibility of circular wait conditions, where multiple threads each hold and are waiting for locks on resources that another thread holds therefore eliminating one necessary condition for deadlock.

Secondly, the duration for which the lock is held is minimized. The lock is acquired only during the critical section that encompasses the seat reservation check and update process. So, holding for this small duration of the lock will reduce the time for which a resource can be held, so there are very less chances to get into deadlock. It is held only for the time it takes to test availability of requested seats and (if available) save a set by marking them as reserved. So, the lock should be released as soon as possible within successful or failed seat reservation operation. This allows for other threads to race ahead and make the reservation with as little delay.

The system also avoids nest locks, reducing the possible of lock contention with multiple lock scenarios. A thread holding one lock tries to acquire another that is already held by some other waiting yet done thread, resulting in circular wait and possibility of deadlocks. By doing this, we have ensured that no 2 theatres can run jitter free and simultaneously locked by a thread. This solution avoids threads blocking one another in a cycle.

Additionally, lock ordering and timed locking are inbuilt properties of the system's architecture. Lock ordering reduces the potential for deadlocks because locks are always acquired and released in a consistent global order. While this isn’t explicitly necessary in this design with the independent nature of theatre locks, it highlights that lock management should be structured. While not explicitly used in this system, a timed lock could be implemented to let threads try and obtain the lock for some time before failing gracefully- preventing infinite blocking of course.

One of the main design goals for the system was to make the important parts as small and effective as feasible. We may enhance throughput and minimize lock contention by limiting the operations within these locked regions to only the essential checks and updates. This strategy adheres to best practices in concurrent programming, where maintaining high performance and avoiding deadlocks requires reducing the scope and length of locks.

Together, these design principles guarantee that the system runs smoothly and quickly, allowing it to manage several seat bookings at once. The careful application of concurrency control techniques and deadlock prevention tactics highlights the system's resilience in effectively and dependably handling numerous customer interactions.

**6.0 Conclusion**

The Cinema Online Seat Reservation System successfully uses a ‘ReentrantLock’ to prevent double-booking of seats, managing 100 concurrent customer threads, and ensuring thread safety. The system’s output provides clear evidence of its functionality, confirming reservations and highlighting failed attempts due to prior reservations. Additionally, the design principles applied ensure the system operates without the risk of deadlock. This report outlines the approach taken, justifies the design decisions and provides a robust solution for concurrent seat reservations in a cinema setting.