






Declaration of Original Work for SC/CE/CZ2002 Assignment

We hereby declare that the attached group assignment has been researched, undertaken, completed and submitted as a collective effort by the group members listed below.

We have honored the principles of academic integrity and have upheld Student Code of Academic Conduct in the completion of this work.

We understand that if plagiarism is found in the assignment, then lower marks or no marks will be awarded for the assessed work. In addition, disciplinary actions may be taken.

Name	Course (CE2002 or CZ2002)	Lab Group	Signature /Date
BERTRANNE LEE ZI - WEN	SC2002	4	
CHAN ONN SANG, ANSON	SC2002	4	
KOH YUE ZHONG	SC2002	4	
LOH SIJING ALOYSIUS	SC2002	4	
SHEARES TOH XUE WEN	CZ2002	4	

Important notes:

1. Name must **EXACTLY MATCH** the one printed on your Matriculation Card.

SC2002 Object-Oriented Design & Programming



MOvie Booking and LIsting Management Application (MOBLIMA)

Demo Video Link:

<https://youtu.be/neRm5whnvdU>

GitHub Repository:

https://github.com/bertrainn/SC2002_MOBLIMA

Lab Group SS1 - Group 4

Lab Supervisor: QIN CHENGWEI

Group Members	
Name	Matriculation No.
CHAN ONN SANG, ANSON	U2120079C
BERTRANNE LEE ZI - WEN	U2120209L
KOH YUE ZHONG	U2120918H
LOH SIJING ALOYSIUS	U2122662A
SHEARES TOH XUE WEN	U2022585K

1.0 Introduction

MOBLIMA is a non-GUI application which handles online booking and ticketing purchase of movie screenings. The two stakeholders are moviegoers and cinema staff. This application is catered for use by a single operator with multiple multiplexes across the island.

The report will showcase design considerations utilised in the code, UML class diagrams and test results from sample test cases. For a higher resolution image of the UML class diagram or test case outputs, please refer to the GitHub repository linked in the cover page or the file in the submission zip file.

2.0 Design Considerations

We seek to generate clear and maintainable code in our MOBLIMA project. To do so, we employ the 5 SOLID software design principles to ensure high cohesion and low coupling. The following sections will contain snippets of code that show the principles at work and any underlying assumptions will be mentioned as well.

2.1 SOLID Principles

Single Responsibility Principle

The idea behind this principle is each method, function or class is to serve only a single or primary objective in the program. Our classes are categorised into entity, boundary and control folders. Within the 'Entity' portion, each class targets a specific task in the movie booking operation. For example, ***SeatLayout*** is responsible for the seating capacity and arrangement of seats in a cinema theatre.

Open-Closed Principle

Classes, modules or functions are to be available for extension but restricted for modification. We show this principle in 'Boundary', where menus are free to add on from the abstract class ***BaseMenu*** without modifying the base code.

Liskov Substitution Principle

The Liskov Substitution Principle highlights the importance of good inheritance hierarchies. This is well implemented in our application. In our 'Entity' portion, we have the ***MovieGoer*** class and ***Admin*** class that is extended from the base ***User*** class. Both extended classes are substitutable for the base ***User*** Class. This means that the inherited classes are able to achieve the same objectives as the base class and do much more.

Interface Segregation Principle

Interface Segregation principle is implemented by using client specific interfaces instead of using general purpose interfaces. Plenty of our classes in the 'Entity' portion implements the ***Serializable*** interface from ***java.io***. This allows the classes to be a better-fit customised interface, which also serves as a tagging of our entity classes.

Dependency Injection Principle

One of the criteria to achieve loosely coupled programs is by ensuring that base modules do not depend on inherited ones. For example, in the 'Boundary' portion, we have plenty of menus that follow the abstract class, ***BaseMenu***. The ***BaseMenu*** class contains an abstract method **load()** that is to be implemented by the lower level classes for various purposes.

2.2 Object-Oriented Concepts

Polymorphism

Polymorphism allows methods in a class to work according to the type of object passed. One such example is the login page where the *User* class can be upcasted to *Admin* or *MovieGoer* so that the user can be navigated to the appropriate main menu with the associated functions.

Inheritance

The hierarchical arrangement of classes allows the classes to inherit from superclasses. 'Boundary', which contains classes responsible for the interface between the user and system, makes use of this principle. For instance, the abstract class, *BaseMenu* contains the abstract method *load()* which can be used by all the inherited menu classes. There is therefore no need to declare the methods again for each subclass.

Encapsulation

It is critical to disallow outsider access to retrieve and modify state values within the entity classes. By isolating the details of the codes to clients, the behaviour of the class is not modified and can only be implemented using the 'get' and 'set' methods. This ensures good coupling between the objects.

Abstraction

Abstraction defines the main characteristics of the class while omitting the details. For example, *BaseMenu* as an abstract class allows specific menus including *BookingMenu* and *AdminMainMenu* to be implemented simply by extending the defined properties or behaviours. It is important to note that the implementation details within the abstract class are hidden and only its essential attributes are visible for extension.

2.3 Assumptions

In addition to the assumptions set in the assignment specifications, our group is also assuming the following statements.

1. All movies are shown for 30 days.
2. Price of tickets are in SGD and are to the nearest cent.
3. Only one user can use the system for browsing and purchasing tickets at any one time.

3.0 Future developments

The two proposed features to be included in future would be:

1. Allow users to cancel booking but with a small penalty

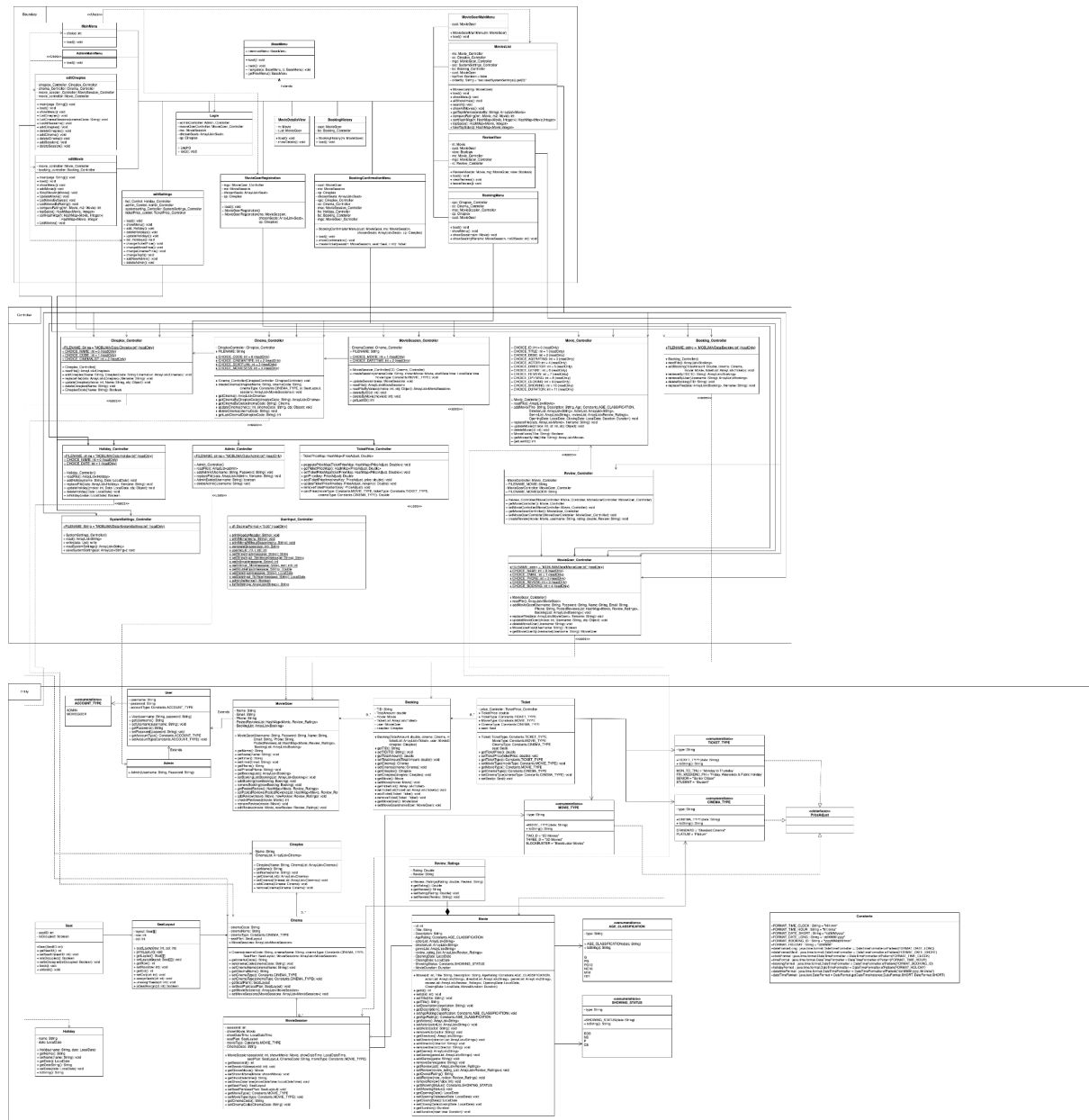
In a scenario where a moviegoer is unable to attend their movie session, they will be able to cancel their booking while incurring a small penalty to prevent abuse of this function, such as fining them an amount proportional to the amount they spent on the ticket.

Using our price controller, we can set a new key value pair in our price map, labelled “penalty” with the value being the percentage value they have to pay if they cancelled their booking as well as a new method to calculate this penalty value. Extending the systems that are currently in place. Regarding removing the booking from our systems, in our booking_controller class, we already have a method called deleteBooking() which removes a booking in our system based on its TID.

2. Book cinema for events.

Another possible feature we could implement is the private booking of cinemas for company or wealthy individual’s use. Which could provide potential business opportunities for people to use the MOBLIMA application.

For this function we could restructure the current moviesession class, converting it into a super class which will be inherited by 2 new classes: public_moviesessions and private_moviesessions. Leveraging LSP we will still be able to use our current movie session controller to manage these 2 new classes rather than making 2 new dedicated controllers for these classes.



For a clearer image of the UML please refer to the project folder. (2002 Movie Class Diagrams-Page-2.drawio.svg)

5.0 Testing sample cases

In this section we will be going over the test cases for our application and how the user is to interact with our MOBLIMA application.

Firstly, here is a summarised list of the test cases which were covered in the demonstration video. Please refer to the video for a showcase of some test cases. The link is on the cover page of this report.

The test cases in the video are summarised in the table below:

User type	Scenario
Candice (moviegoer)	Browse system for top-grossing movies
	Book tickets at preferred time and location
	Leave a review after watching
Cineplex admin personnel (admin)	Add, modify and remove movies shown in theatres
Head operator for cineplexes (admin)	Modify calendar holidays
	Change ticket pricing
	Toggle how top 5 movies are selected
	Manage admin user accounts

Additional Test Cases

Secondly, these are a selected few test cases which we did not have time to cover during the demonstration video but we felt were important to the user experience when using MOBLIMA.

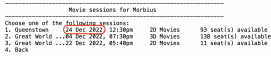
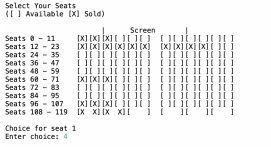
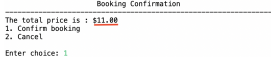
Purchasing couple seats

If a moviegoer intends to purchase a pair of couple seats, the seating position of one of the two seats only has to be selected. The system automatically selects the other seat and charges the user the price of two seats.

	Test Case	Expected Outcome	Actual Results
1.	The user selects seat 131 (left side of the couple seat)	MOBLIMA to automatically select the other side of the couple seat 132.	<pre> How many tickets would you like to purchase? (Enter 1 to buy 1 couple seat) Enter choice: 1 Select Your Seats ([] Available [X] Sold) Screen Seats 0 - 12 [] [] [] [] [] [] [] [] [] [] [] [] Seats 13 - 25 [X] [] [] [] [] [] [] [] [] [] [] [] Seats 26 - 38 [] [] [] [] [] [] [] [] [] [] [] [] Seats 39 - 51 [] [] [] [] [] [] [] [] [] [] [] [] Seats 52 - 64 [] [] [] [] [] [] [] [] [] [] [] [X] Seats 65 - 77 [X] [] [] [] [] [] [] [] [] [] [] [] Seats 78 - 90 [X] [] [] [] [] [] [] [] [] [] [] [] Seats 91 - 103 [] [] [] [] [] [] [] [] [] [] [] [] Seats 104 - 116 [] [] [] [] [] [] [] [] [] [] [] [] Seats 117 - 129 [X] [X] [] [] [] [] [] [] [] [] [] [] Seats 130 - 142 [] [X] [] [] [] [] [] [] [] [] [] [] Choice for seat 1 Enter choice: 131 </pre>
2.	User confirms their booking	System to charge them for 2 seats.	<p>The total price is : \$19.00 1. Confirm booking 2. Cancel</p> <p>The user pays the price of two seats (\$9.50 each)</p>
3.	System to update the seating plan after the booking has taken place	System to show an X over both couple seats	<pre> Select Your Seats ([] Available [X] Sold) Screen Seats 0 - 12 [] [] [] [] [] [] [] [] [] [] [] [] Seats 13 - 25 [X] [] [] [] [] [] [] [] [] [] [] [] Seats 26 - 38 [] [] [] [] [] [] [] [] [] [] [] [] Seats 39 - 51 [] [] [] [] [] [] [] [] [] [] [] [] Seats 52 - 64 [] [] [] [] [] [] [] [] [] [] [] [X] Seats 65 - 77 [X] [] [] [] [] [] [] [] [] [] [] [] Seats 78 - 90 [X] [] [] [] [] [] [] [] [] [] [] [] Seats 91 - 103 [] [] [] [] [] [] [] [] [] [] [] [] Seats 104 - 116 [] [] [] [] [] [] [] [] [] [] [] [] Seats 117 - 129 [X] [X] [] [] [] [] [] [] [] [] [] [] Seats 130 - 142 [] [X] [X] [] [] [] [] [] [] [] [] [] </pre>

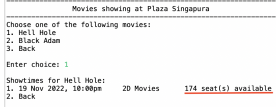
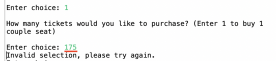
Booking a ticket on the eve of a holiday

The price of movie tickets are raised on the eve of a holiday. To illustrate this, a moviegoer purchases a ticket to watch 'Mobius' at Queenstown cineplex on Christmas Eve, 24 Dec.

	Test Case	Expected Outcome	Actual Results
1.	User selects a movie session for a specific movie and selects the movie session which is on the eve of a public holiday	System list all the sessions for that particular movie	 <p>Movie sessions for Mobius</p> <p>Choose one of the following sessions:</p> <p>1. Opening ... 12:00pm 20 Movies 80 seats available</p> <p>2. Great World ... 2:00pm 20 Movies 100 seats available</p> <p>3. Great World ... 4:00pm 20 Movies 120 seats available</p> <p>4. Back</p>
2.	User selects his seat and confirm his booking	System to display the seating plan to the user	 <p>Select Your Seats</p> <p>([] Available [X] Sold)</p> <p>Screen</p> <p>Seats 0 - 11 [X] [X] [X] [] [] [] [] [] [] [] [] []</p> <p>Seats 12 - 23 [X] [X] [X] [X] [X] [X] [X] [X] [X] [X] [X] [X]</p> <p>Seats 24 - 35 [] [] [] [] [] [] [] [] [] [] [] []</p> <p>Seats 36 - 47 [] [] [] [] [] [] [] [] [] [] [] []</p> <p>Seats 48 - 59 [] [] [] [] [] [] [] [] [] [] [] []</p> <p>Seats 60 - 71 [X] [X] [X] [] [] [] [] [] [] [] [] []</p> <p>Seats 72 - 83 [] [] [] [] [] [] [] [] [] [] [] []</p> <p>Seats 84 - 95 [] [] [] [] [] [] [] [] [] [] [] []</p> <p>Seats 96 - 107 [X] [X] [X] [] [] [] [] [] [] [] [] []</p> <p>Seats 108 - 119 [X] [X] [X] [X] [] [] [] [] [] [] [] []</p> <p>Choice for seat 1</p> <p>Enter choice: 4</p>
3.	User proceeds to pay for his ticket	System to charge them Friday, Weekend & Holiday price for their ticket	 <p>Booking Confirmation</p> <p>The total price is: \$11.00</p> <p>1. Confirm Booking</p> <p>2. Cancel</p> <p>Enter choice: 1</p> <p>The price of a ticket is \$11.00 instead of the usual standard price of \$9.50.</p>

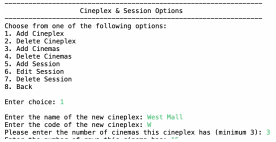
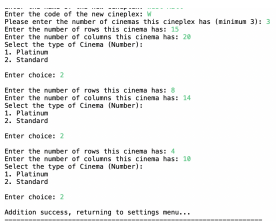
Selecting more seats than available in a movie session.

If the moviegoer purchases tickets exceeding the number of available seats left in a cinema theatre, an error will be shown and the user has to indicate the number of tickets.

	Test Case	Expected Outcome	Actual Results
1.	User selects a movie at a specific cineplex	System list all the sessions and the number of seats available	
2.	User enters his number of seats more than the number of seats available	System to display: "Invalid selection, please try again"	

Creating a new cineplex

An administrator is free to manage the cineplexes and movie sessions. The creation of a new cineplex is demonstrated here. The name, number of cinema theatres, seating layout and type of cinema (standard or platinum) can be defined.

	Test Case	Expected Outcome	Actual Results
1.	Admin selects create new cineplex option in the "Cineplex & Session Options" menu	System prompts the admin for the information relating to the new cineplex.	
2.	System prompts the user for the details about the cinemas in the cineplex. User enters the details	System to return "Addition Success, Returning to main menu" and add the new cineplex and cinemas to the database	
3	User to display all the cineplexes in the system, including the one just added	System to display all the cineplexes in the system with the relevant information.	