

**NANYANG
TECHNOLOGICAL
UNIVERSITY**
SINGAPORE

SC/CE/CZ2002: Object Oriented Design & Programming

Report: Building an OO Application

SS7 Group 5

Submitted by:

Jarel Tan Zhi Wen (U2121582H)

Lin Da Wei (U2120027J)

Lui Shi Ying (U2121282F)

Solomon Duke Tneo Yruan Rui (U2122205J)

Wong Yi Xian (U2121401C)

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY**






Declaration of Original Work for SC/CE/CZ2002 Assignment	3
Design Considerations	4
Approach	4
Object-Oriented Design Principles Used	4
1. Implementation of SOLID Principles	4
a. Single Responsibility Principle (SRP)	4
b. Open-Closed Principle (OCP)	5
c. Interface Segregation Principle (ISP)	5
2. Design Patterns	6
a. Facade Design Pattern (FDP)	6
3. Object-Oriented Concepts	7
a. Encapsulation	7
b. Abstraction	7
Feature Enhancement	
1. Diversification of Cinema Seat Layouts	8
2. Exclusive movie lists for Premium Members	8
Assumptions	9
Pricing Information	10
Test Cases	11
UML Class Diagram	13

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	Lab Group	Signature/Date
Jarel Tan Zhi Wen	SC2002	SS7	 12/11/2022
Lin Da Wei	SC2002	SS7	 12/11/2022
Lui Shi Ying	SC2002	SS7	 12/11/2022
Solomon Duke Tneo Yruan Rui	SC2002	SS7	 12/11/2022
Wong Yi Xian	SC2002	SS7	 12/11/2022

Important notes:

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

Important Links

1. Group Video Demonstration: <https://youtu.be/IV-vGRfmPg>
2. Containing UML Class Diagram and source code files:
<https://github.com/Yixian17/SC2002>

Design Considerations

Approach

Using Object-Oriented Principles, we designed our movie booking application to facilitate the online booking of movie tickets. Upon logging in to a legitimate staff account, a cineplex staff is entitled the right to update and amend movie listings, showtimes and ticket prices. Furthermore, the application provides a fast and convenient movie ticket booking process for movie go-ers. Movie go-ers are also granted viewer access to their current movie ticket, booking history and the top 5 movies listed at any time. Both staff and users alike would be able to

Object-Oriented Design Principles Used

1. Implementation of SOLID Principles

a. Single Responsibility Principle (SRP)

We incorporated the Single Responsibility Principle (SRP) by establishing that the number of responsibilities borne by each class ought not to exceed one. If classes embody more than one responsibility, amendments and updates to the numerous responsibilities will translate into the excessively frequent need to change the class as a whole. Besides, these changes may consequently cause changes in other classes that are dependent on the class (Janssen, 2021).

For example,

- The class **Price.java** is only accountable for the calculating the price of the movie ticket (calculatePrice()) and nothing else
- The class **Seats.java** is also doing nothing but enabling the users to select their preferred seat for booking (assignSeat())

To enable convenience in applying this concept, our project embraces the various Class Stereotypes of “Boundary”, “Control” and “Entity”. For example, the **PriceConfigure.java**, **PriceConfigureController.java** and **Prices.java** classes in our code follow the “Boundary-Control-Entity (BCE)” framework. Focusing on these 3 classes, the **Prices.java** class calls functions from the **PriceConfigureController.java** class in order to obtain the values of price adjusters that can be used by **Prices.java**’s calculatePrice() function to determine the movie ticket price. **PriceConfigureController.java** writes and reads from a **PriceConfiguration** Serialised text file to obtain the latest value of price adjusters updated by the Staff – and updates the attributes (price adjusters) entity class of **PriceConfigure.java**.

b. Open-Closed Principle (OCP)

Open-Closed Principle is applied in some of the classes, allowing them to be open for extension but closed for modification. For example, the **StaffUI.java** and **UserUI.java** classes extend the use of **MainUIInterface.java** and its abstract displayInterface() method but do not modify the actual method in the **MainUIInterface.java** class. This allows the changing of UI class responsibility without changing the source code of the classes.

c. Interface Segregation Principle (ISP)

By dividing the program into various, independent components, ISP seeks to minimise the negative impacts and the frequency of necessary changes. We use our **MainUIInterface.java** interface that aids us in simplifying the creation of separate user interfaces for staff and movie go-ers respectively. We avoid writing all the code for both groups of users (staff and movie go-ers) in one single user interface class and over-complicating as a result. The act of breaking down into smaller classes enables

the **StaffUI.java** class to focus on fulfilling the needs of the cineplex staff and the **UserUI.java** class to concentrate on the movie go-ers.

2. Design Patterns

a. Facade Design Pattern (FDP)

FDP is a software design pattern that is frequently used in Object-Oriented Design. A facade is an object that acts as a front-facing interface while concealing more complicated underlying or structural code. A facade class will enable us to enhance our library's readability and usability by hiding interaction with more sophisticated components behind a single (and frequently simplified) API. It also empowers the usage of more generic functions a context-specific interface (complete with context-specific input validation)

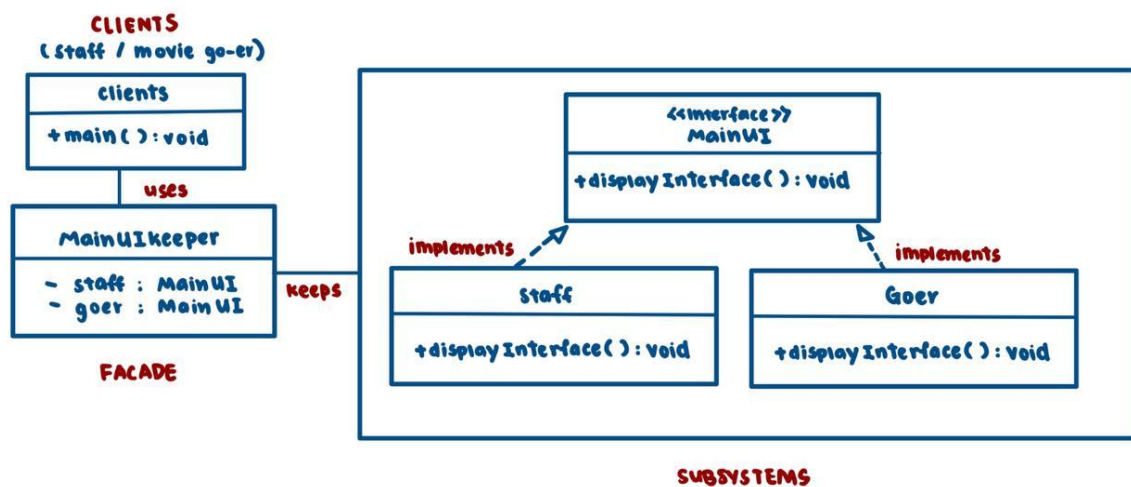


Figure 1: Application of Facade Design Pattern in the User Interfaces

As illustrated in Figure 1, we applied this concept in our code by using the **MainUIInterface.java** interface to hide complexities in the main user interface of our MOBLIMA application. Our concrete classes **StaffUI.java** and **UserUI.java** implement the interface **MainUIInterface.java** by implementing its `displayInterface()` method. The facade class used in this case would be **MainUIKeeper.java** – which allows us to maintain a simple main interface, that

hides the large code in StaffUI.java and UserUI.java where all the additional and more specific functions are contained in.

3. Object-Oriented Concepts

a. Encapsulation

Encapsulation is the concept of using public getters and setters to access private attributes of a class. The advantage of this lies in concealing, from users, the complexities regarding data reading and storing within a class. As seen in Figure 2a and 2b, the **Review.java** and **TicketBuyer.java** classes use getters (e.g., getReview()) or getAge()) and setters (e.g. setReview(), setAge()) to empower the accessibility of private data attributes (and methods) by other classes or users, without them actually being aware of the minute details regarding their implementations.

```
public class Review implements Serializable{
    private static final long serialVersionUID = 1L;
    private String review;
    private int rating;

    public Review() {
    }

    Review(int r, String rev){
        this.review = rev;
        this.rating = r;
    }

    public String getReview() {
        return this.review;
    }

    public int getRating() {
        return this.rating;
    }

    public void setRating(int r)
    {
        this.rating = r;
    }

    public void setReview(String review)
    {
        this.review = review;
    }
}
```

Figure 2a: Encapsulation in **Review.java** Class

```
public class TicketBuyer implements Serializable{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    //instance variables
    private String name;
    private int mobilenum;
    private String emailaddress;
    private int age;
    private BookingHistory history;
    //booking history

    //constructors
    public TicketBuyer(String name, int mobilenum, String emailaddress, int age) throws Exception {
        this.name = name;
        this.mobilenum = mobilenum;
        this.emailaddress = emailaddress;
        this.age = age;
        //create a booking for the ticket buyer
        this.history = new BookingHistory();
    }

    //getters
    public String getName () { return this.name; }
    public int getMobileNumber() { return this.mobilenum; }
    public String getEmailAddress() { return this.emailaddress; }
    public int getAge() { return this.age; }
    public BookingHistory getBookingHistory() { return this.history; }

    //setters
    public void setName (String name) {this.name = name;}
    public void setMobileNumber(int mobilenum) {this.mobilenum = mobilenum; }
    public void setEmailAddress(String emailaddress) { this.emailaddress = emailaddress;}
    public void setAge(int age) {this.age = age;}
    public void setBookingHistory(BookingHistory history) {this.history = history;}
}
```

Figure 2b: Encapsulation in **TicketBuyer.java** Class

b. Abstraction

Abstraction concerns the hiding of unnecessary details, for example in method implementation. When using the MOBLIMA application, cineplex staff or users that interact directly with the User Interface (UI) would not wish to acquire excessively detailed information regarding (for example) the detailed working mechanisms behind how the application lists movies. This example is illustrated in Figure 3a and 3b, with

snippets of code extracted from **MovieTiming2.java** and **StaffUI.java** or **UserUI.java**.

```
public class MovieListing2 {
    //public static final String SEPARATOR = "|";
    private static ArrayList<Movie2> movies = new ArrayList<Movie2>();
    private static ArrayList<Movie2> usermovies = new ArrayList<Movie2>();

    // (1) Create/Update/Remove Movie Listing
    //See Movie List Function //Using File Input Stream and Object Input Stream
    public static void listMovie() throws Exception {
        FileInputStream fis = new FileInputStream("MovieDetails6.txt");
        try {ObjectInputStream ois= new ObjectInputStream(fis);
            int num = ois.readInt();
            System.out.println("-----");
            System.out.println("List of movie:");
            movies.clear();
            for (int i = 0; i < num; i++) {
                movies.add((Movie2) ois.readObject());
                System.out.println("Movie " + (i + 1) + " : " + movies.get(i).getTitle());
            }
            ois.close();
        } catch (Exception EOFException) {return;}
    }
}
```

Figure 3a: listMovie() function abstracts the detailed implementations required to list movies

```
case 1: try {MovieListing2.listMovie();}
        catch (Exception EOFException) {System.out.println("No movie found");}
        break;
```

Figure 3b: listMovie() function called in the StaffUI and UserUI

Feature Enhancement

1. Diversification of Cinema Seat Layouts

Currently, we have a standard movie seat layout across all cinema halls. Our current assumption is that Platinum and Gold Class cinemas have bigger screens with higher quality, more leg room and reclining functions built on every seat type (Normal, Couple, Elite and Ultima). However, in order for the cineplex to further entice movie go-ers into selecting the Platinum and Gold Class halls, we ought to further enhance the differentiating factor between the Normal, Platinum and Gold Class cinemas in the future. This warrants restructuring the seating layout for every cinema hall, such that seats are strategically placed at angles that optimise comfortability in movie viewing. Potential for alterations in seat layout comes with coding our MOBLIMA application such that seat layouts are stored into modifiable text files.

2. Exclusive movie lists for Premium Members

Currently, all movie go-ers alike view the same list of movies available for booking. No individual is granted exclusive viewership to any particular film. In view of amplifying the appeal of implementing a premium membership scheme, Premium Members should be granted access to an exclusive and distinguished list of movies ahead of other users. For example, they should enjoy exclusive viewing rights to a movie preview for a long-anticipated blockbuster movie ahead of other movie go-ers. Realisation of this functionality can be permitted through future implementation of method overloading on `MovieListing2.listMovie()` in our **MovieListing2.java** class.

Data Storage

For this assignment, we created many objects like Staff, Movie (saved as Movie2), Cineplex and many more. We used the concept of object serialisation to store our objects. When we want to list the objects created, we initialise `ObjectInputStream` objects with `FileInputStream` objects, then we use `readObject` method to return references of the objects from the input stream. As for when we want to update an object in the serialised file, first we use the method `writeObject`, then we initialise `ObjectOutputStream` with `FileInputStream` objects. All classes that are used to create new objects implements `Serializable` so that the object can be serialised.

Assumptions (On top of assignment requirements)

- Time is in 24-hour clock
- For prices:
 - Normal movies on Monday to Wednesday follows the base price set
 - Blockbuster prices are always \$1 higher than prices for Normal movies
 - For seniors and students, prices for Normal movies and Blockbuster movies are the same
- For reviews:
 - Reviews and Ratings will only show up when there is only more than one review
- For holidays:
 - Staff users should be wary to key in both the date of the public holiday and the eve of the public holiday when they attempt to create a new holiday list

Pricing Details

Pricing Formula	<ol style="list-style-type: none"> Age-Related Discounts <ul style="list-style-type: none"> Seniors and Students Price (Normal Movies) = Base Price - Discount Student Price (3D Movies) = Student Price (Normal Movies) + Student 3D Adder Holiday, Day-Related & Movie Type-Related <ul style="list-style-type: none"> Price = Base Price + Adder Seat-Related Price Adjustments <ul style="list-style-type: none"> Price = Derived Price * Multiplier Cinema-Related Price Adjustments <ul style="list-style-type: none"> Price = Overall Price + Adder <p>**Note: Seat & Cinema-Related adjustments are always done second last and last respectively</p>
Default Price Settings	<p><u>Age-Related Discounts:</u> Seniors Discount = \$4.50 Student Discount (Normal) = \$1.50 Student (3D Movies) Adder = \$2.00</p> <p><u>Price Adders for Normal Adults</u> Normal Movies: <ul style="list-style-type: none"> Thur-Fri before 6pm = \$1.00 Fri after 6pm - Sun & Holidays = \$2.50 Blockbuster Movies: <ul style="list-style-type: none"> Mon-Wed = \$1.00 Thurs-Fri before 6pm = \$2.00 Fri after 6pm-Sun & Holidays = \$3.50 <u>3D Movies:</u> <ul style="list-style-type: none"> Mon-Thurs = \$2.50 Fri-Sun & Holidays = \$6.50 <u>Seat Type Price Multiplier</u> <ul style="list-style-type: none"> Normal = 1 Couple = 2 Elite = 2.5 Ultima = 3 <u>Cinema Class Price Adders</u> <ul style="list-style-type: none"> Classic = \$0 Platinum = \$1.50 Gold = \$3.00 </p>

Test Cases

1. Configure Holidays

Test Case	Output Expected
<p>User configures “20221113” as public holiday</p> <p>Create movie timing on “20221113” for movie Minions</p> <p>Price reflected upon booking is \$11, holiday pricing</p>	<pre> ----- Select the following action: (1) Create/Update/Remove Movie Listing (2) Create/Update/Remove Cinema Showtimes and Movies (3) Configure System Setting ----- 3 ----- Select the following action: (1) Display List of Cineplex (2) Create New Cineplex (3) Remove Cineplex (4) Add New Cinema (5) Create Holiday (6) Update Pricing (7) Toggle Top 5 for user (8) Back ----- 5 ----- Enter the number of holidays: 1 Enter holiday date (YYYYMMDD) 1: 20221113 ----- </pre> <p style="text-align: center;">➡</p> <pre> ----- Select a movie: Movie (1): Up Movie (2): Black Panther Movie (3): SpiderMan Movie (4): Black Adam Movie (5): Minions Movie (6): Ajoomma ----- 5 ----- Enter Date (YYYYMMDD): 20231113 Enter Time (HHMM): 1000 ----- </pre> <p style="text-align: center;">➡</p> <pre> Please enter your phone number: 9000000 Please enter your email address: bob@gmail.com Please enter your age: 25 ----- Your Booking Has been Confirmed! Date: 20231113 Show Time: 1000 Name: bob Mobile Number: 9000000 Email Address: bob@gmail.com Seat Number: R1, C1 Price: 11.0 Transaction ID: TBPHALLIC1202211130130 ----- Type of Seats: Normal Seats(Single): [] Couple Seats(Double): [] Elite Seats(Double): [] Ultima Seats(Double): [] ----- [SCREEN] ----- 1 2 3 4 5 6 7 8 9 10 11 12 1 [XX][][][][][][][][][][][] 2 [][][][][][][][][][][][] 3 [][][][][][][][][][][][] 4 [][][][][][][][][][][][] 5 [][][][][][][][][][][][] 6 [][][][][][][][][][][][] </pre>

2. Add Ratings for a Movie

Test Case	Output Expected
<p>User enters Rating to Selected Movie - “Black Panther”</p>	<pre> ----- Select the following action: (1) Search / List Movies (2) View Movie Details (3) Review Movie (4) Check Seat Availability and Selection of Seats (5) Book and Purchase Ticket (6) View Booking History (7) List Top 5 (8) Back ----- 3 ----- Select a movie: Movie (1): Up Movie (2): Black Panther Movie (3): SpiderMan Movie (4): Black Adam ----- 2 ----- What is your score (out of 5) for the movie: Black Panther ? 4 ----- What are your comments on the movie? pretty good movie ----- </pre> <p style="text-align: center;">➡</p> <pre> Black Panther Movie Details =====INFORMATION===== (1) Title: Black Panther (2) Showing status: Now Showing (3) Movie type: Blockbuster (4) Movie rating: PG (5) Synopsis: -- A black coloured panther (6) Director: Orson Welles (7) Cast information: -- Jack Nicholson -- Natalie Portman (8) Overall rating: 4.5 (9) Past reviews: -- 5/5: damn good movie -- 4/5: pretty good movie ===== </pre>

3. Movie Go-er User Interface does not show movies that have been put to “End of Showing”

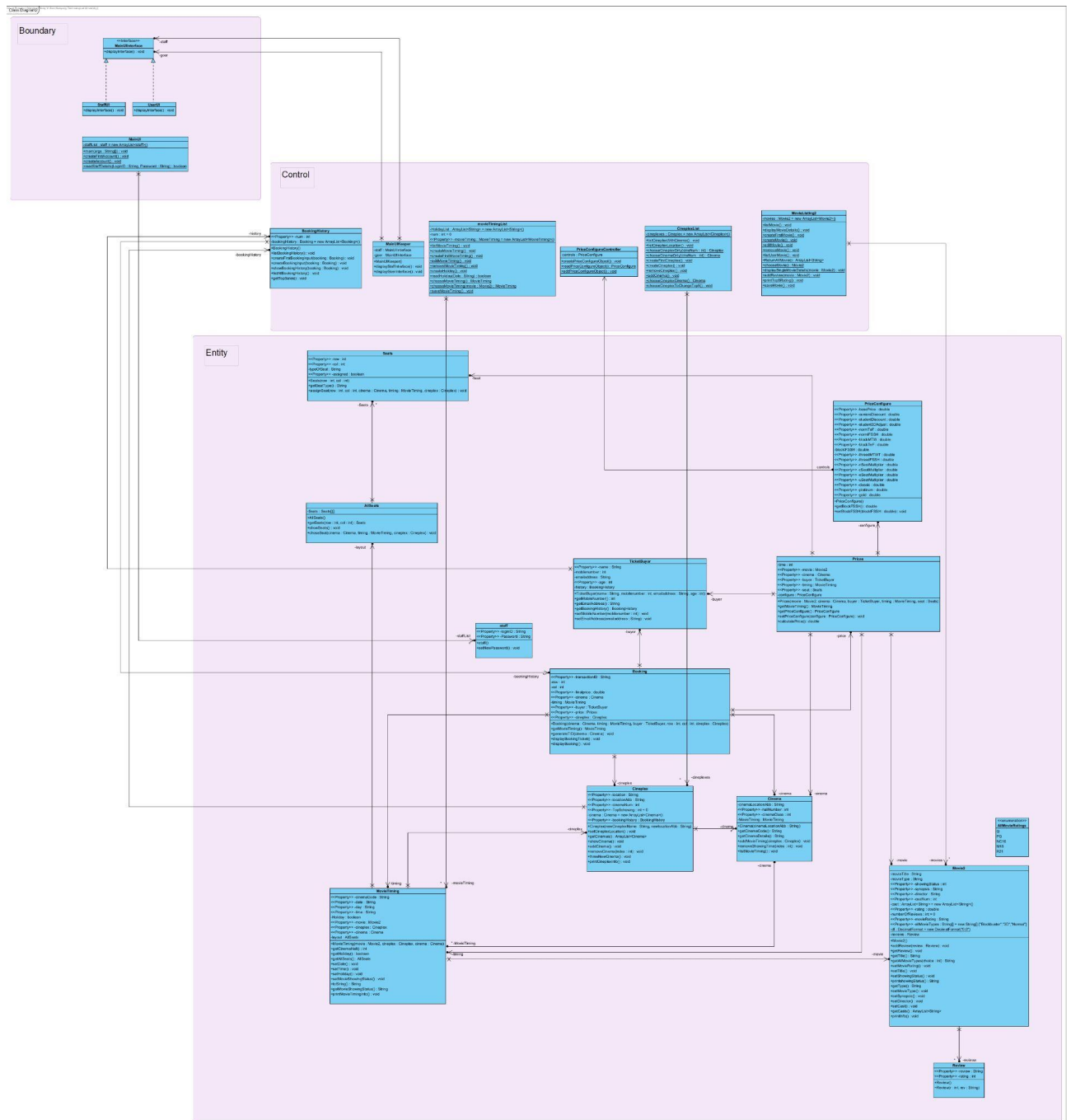
Test Case & Output Expected		
<pre> Select information to edit: =====INFORMATION===== (1) Title: Ajoomma (2) Showing status: Now Showing (3) Movie type: Normal (4) Movie rating: NC16 (5) Synopsis: -- hi i am ajoomma (6) Director: 1 (7) Cast information: -- 1 (8) Overall rating: NA (9) Past reviews: -- NA ===== 2 ----- Enter Showing Status: 1. coming soon 2. preview 3. now showing 4. end of showing ----- 4 </pre>	<pre> Movie 6: Ajoomma =====INFORMATION===== (1) Title: Ajoomma (2) Showing status: End Of Showing (3) Movie type: Normal (4) Movie rating: NC16 (5) Synopsis: -- hi i am ajoomma (6) Director: 1 (7) Cast information: -- 1 (8) Overall rating: NA (9) Past reviews: -- NA ===== </pre>	<pre> ----- Select the following action: (1) Search / List Movies (2) View Movie Details (3) Review Movie (4) Check Seat Availability and Selection of Seats (5) Book and Purchase Ticket (6) View Booking History (7) List Top 5 (8) Back ----- 4 ----- Select a movie: Movie (1): Up Movie (2): Black Panther Movie (3): SpiderMan Movie (4): Black Adam Movie (5): Minions ----- </pre>

4. Examples of Error Handling in our code

Test Case & Output Expected		
<pre> ----- Select the following action: (1) Search / List Movies (2) View Movie Details (3) Review Movie (4) Check Seat Availability and Selection of Seats (5) Book and Purchase Ticket (6) View Booking History (7) List Top 5 (8) Back ----- 3 ----- Select a movie: Movie (1): Up Movie (2): Black Panther Movie (3): SpiderMan Movie (4): Black Adam ----- 2 ----- What is your score (out of 5) for the movie: Black Panther ? -1 Please input a number between 0 to 5 What is your score (out of 5) for the movie: Black Panther ? </pre>	<pre> ----- Select a movie: Movie (1): Up Movie (2): Black Panther Movie (3): SpiderMan Movie (4): Black Adam ----- 5 ----- Try Again! Select a movie from above ----- </pre>	<pre> Enter row number to book: 1 Enter column number to book: 1 ----- Type of Seats: Normal Seats(Single): [] Couple Seats(Double): [~~~~~] Elite Seats(Double): [~~~~~] Ultima Seats(Double): [*****] ----- [SCREEN] ----- 1 2 3 4 5 6 7 8 9 10 11 12 1 [XX][][][][][] [][][][][][] 2 [][][][][][] [][][][][][] 3 [][][][][][] [][][][][][] 4 [~~~~~][~~~~~][~~~~~][~~~~~][~~~~~] 5 [~~~~~][XXXXXX][~~~~~][~~~~~][XXXXXX][~~~~~] 6 [*****][*****][XXXXXX][*****][XXXXXX][XXXXXX] ----- Please Enter the Valid Seat Enter row number to book: </pre>
User enters a rating < 0	User selects a movie number to book that is not shown	When a movie go-er selects a seat that is already booked

UML Class Diagram

To access a clearer version of our UML Class Diagram, kindly access this link <https://github.com/Yixian17/SC2002> and open “UML Class Diagram.jpg”.



References

Janssen, T. (2021, April 5). *Solid design principles: The Single Responsibility explained*. Stackify. Retrieved November 11, 2022, from <https://stackify.com/solid-design-principles/>