Group Members	**Johnathan Chow Zheng Feng**	**U2121835D**
	Joel Tan (Chen Enjing)	**U2122877C**
	Sim Guan Yu	**U2120328B**
	Sim Oi Liang	**U2123863L**
Lab group	**SS9 (Group 4)**	
Date of Submission/ Deadline	**13/11/2022 (Week 13 Sunday), 11.59pm**	

Github link: https://github.com/john14759/SC2002-MOBLIMA-assignment

Declaration of Original Work:

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
Johnathan Chow Zheng Feng	**SC2002**	**SS9**	
Sim Oi Liang	**SC2002**	**SS9**	
Joel Tan (Chen Enjing)	**SC2002**	**SS9**	
Sim Guan Yu	**SC2002**	**SS9**	

Important notes:

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

Table of contents

Report outline:

This report was generated by Group 4 of Lab Class SS9. The details in this report include the considerations that our group has made in the design process during the building of our MOBLIMA app. We incorporated the use of the features in Object-Oriented Programming, such as Polymorphism and also design principles such as SOLID.

All in all, we have created a mobile cinema booking system using Java on Eclipse IDE, which has both the frontend UI and backend database to give the user a pleasant experience on our console application.

Design considerations:

The ***UI*** contains all the applications that integrate all the different Managers together.

Firstly, *MoblimaApp* is the main user interface that the user can interact with, when he/she opens the console application.

Then, depending on the user inputs, our 4 other main applications will be activated.

MovieApp is the next application interface that customers will be able to access, if they have logged into an account. This allows customers to view movie listings by sale, overall rating and also book movie tickets.

UserAccountApp is an application interface where the user prompts User Login/Signup in the main interface, and users will have two options in this interface – which is to either to sign up for a new account or to log into their existing account.

CommentRateApp is an application that allows customers to add and update their comments and ratings for a particular movie.

SystemConfig is an application that allows staff members who are logged in using their staff account to edit information related to the entities, such as editing movie listing, editing ticket pricing and showtimes.

Models are the entities which store the information needed in our application. The information is managed via accessors and mutators, hence all the objects created are up to date with the changes made by the user/staff during the running of the application.

Serializers aid in the process of overseeing the databases. This allows for efficient and secure storage as the information will not be erased upon exiting the application. Object arrays are converted into string format during serialization (when making changes to the database) and likewise during deserialization (when reading from the database).

Design principles used:

Single responsibility Principle:

The Single-Responsibility Principle (SRP) states that each class should only hold one responsibility. In other words, it should be programmed to only do one thing.

Hence, we have split all the classes that are used in this application into UIs, models, managers and databases to make sure each class has its own specific role to play in the execution of the application.

For example, we have BookingMgr and inside this booking manager, the pricing of the tickets are not included. Instead we have created a separate class called PriceCalculator, which stores the pricing scheme of all the tickets.

Open-Close Principle:

Open-Closed Principle (OCP) indicated that the entities created, including classes, functions and modules, should be unmodifiable but extendable. The code should be written in a way that allows new functions to be added but in the process, the existing code should not be changed.

In our system, we have used the database to allow for the extension of the existing methods and variables of the class. If there were to be an extension of the system (cinema management comes up with a loyalty program to reward users with points every time a booking is made), then there would simply be another database that would be created to fit this new loyalty system.

OOP concepts used:

Abstraction:

Abstracting all the details that the user does not need to interact with, provides a clean and simple to understand interface.

It is not important for the user to know how the program books the movie tickets and updates the database. However, it is important that the database is updated every time a booking is made.

It is also not important for users to know how the PaymentMethod works in our code, the users will just have to choose their payment option when booking a ticket.

Encapsulation:

Encapsulation ensures the protection of certain fields in a class. Public methods such as get and set can be used to access these private data. In our app, all attributes under the package “Models” are private.

Private attributes:

```
private static final long serialVersionUID = 1L;
private int cineplexID;
private String name;
```

```
private static final long serialVersionUID = 1L;
private int movieRankID;
private int movieID;
private int numRaters;
private double overallRating;
private double sales;
```

Get & set methods:

```
public int getMovieRankID() {
    return this.movieRankID;
}

public void setMovieRankID(int id) {
    this.movieRankID =id;
}
public int getMovieID() {
    return this.movieID;
}

public void setMovieID(int id) {
    this.movieID = id;
}

public int getNumRaters() {
    return this.numRaters;
}

public void setNumRaters(int num) {
    this.numRaters = num;
}

public double getOverallRating() {
    return this.overallRating;
}

public void setOverallRating(double rating) {
    this.overallRating = rating;
}

public double getSales() {
    return this.sales;
}

public void setSales(double sales) {
    this.sales = sales;
}
```

```
public MovieStates getMovieState() {
    return this.state;
}

public void setMovieState(MovieStates movieState) {
    this.state = movieState;
}

public int getMovieID(){
    return this.movieID;
}

public void setMovieID(int id){
    this.movieID = id;
}

public String getTitle(){
    return this.title;
}

public void setTitle(String title){
    this.title = title;
}

public String getDirector(){
    return this.director;
}

public void setDirector(String director){
    this.director = director;
}

public ArrayList<String> getCasts(){
    return this.casts;
}

public void setCasts(ArrayList<String> casts){
    this.casts = casts;
}

public String getMovieContent(){
    return this.movieContent;
}
```

Inheritance:

With inheritance, new classes can be created that contain the base class' methods. These subclasses can easily extend from base classes, affording great reusability and increasing efficiency in the code.

Our PayByQRCode and PayByCreditCard inherits from the superclass PaymentMethod, where the common functions are stored. This inheritance allows PayByQRCode and PayByCreditCard

to use all these functions and simply add the specific functions for the user to pay with each method, reducing duplicates in the code.

Polymorphism:

For our database, every classes' read and save functions are overloaded in order to obtain the accurate fields from the database.

Assumptions made in our app:

1. Customers that are logged in are allowed to purchase and book movie tickets. Those that are not logged in, will only be allowed to view the movie listing.

2. Once booking is completed, the user cannot change the existing booking that is made which means that the tickets are non-refundable by the cinema.

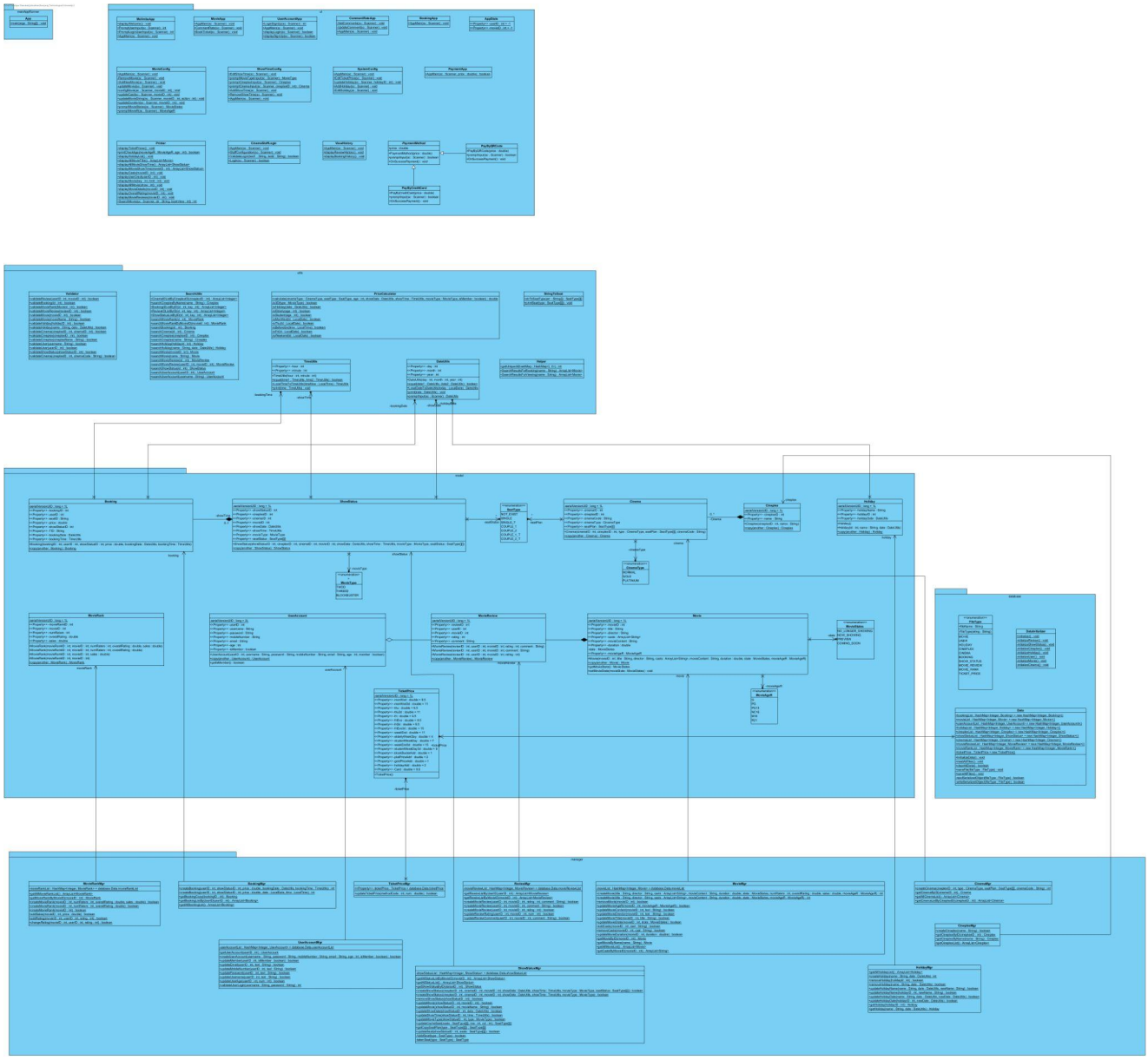
3. The system assumes that users already know the price of the various add-ons (such as premiums for different classes, 3D, IMAX etc.)

4. The currency will be in Singapore Dollar (SGD) and inclusive of Good and Services Tax (GST).

5. Payment will always be successful.

6. THREE cineplexes will be created for the demonstration.

UML Class diagram:



(Please refer to the file named “Class Diagram final version” for a clearer and higher resolution image)

Future considerations in our app:

Movie Recommendation System:

The movie recommendation system application will give movie recommendations based on the user's age, gender, favourite genre, and past movies watched. Then, according to the user's input, each movie will be given points. For example, if the user puts 10 years old as his/her age, genres like animation and family will get higher points, and those like horror or thriller will get a lower score. The tallied score will be calculated for all the movies in the database and the movie recommendation system will display movies with the highest score to the lowest score.

Our code provides the extensibility through the open-closed principle. For example, under our MovieConfig class, we can set and update title, director's name and casts. We can further extend this class to a MovieRec class where we can set movie genres and age ratings (G, PG13, M18 etc).

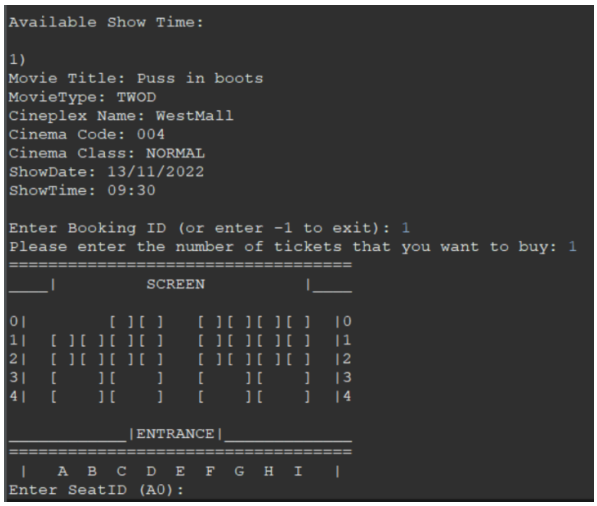
Enhanced Profile System:

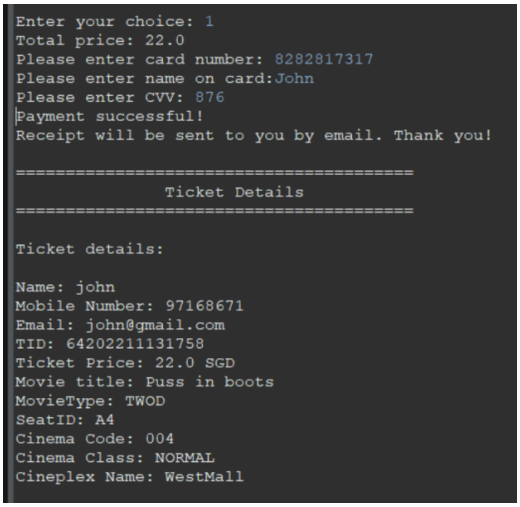
Besides creating an account for moviegoers, we can enhance this account login page with extra features like a rewards system, a deals redemption page and so on. When the moviegoers make a purchase, points will be credited according to the amount spent. After accumulating points, moviegoers can enter the deals redemption page to redeem deals with the points they have.

Test cases not shown in the video:

1. View all movie listing without filtering based on sales/ratings:

2. Booking of couple seats:



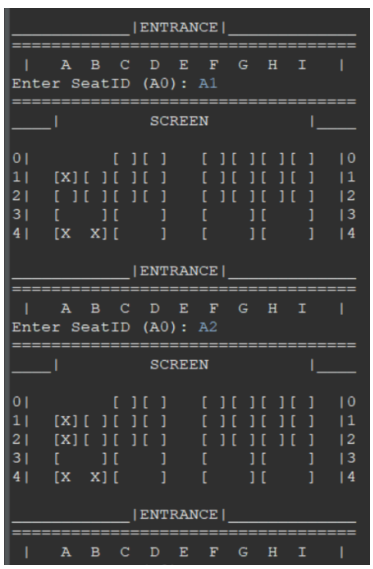


Booking shows the price of couple seats correctly, which is $22, and this also means that the customer will only have to book once for the system to register that it is a couple seat for 2 people.

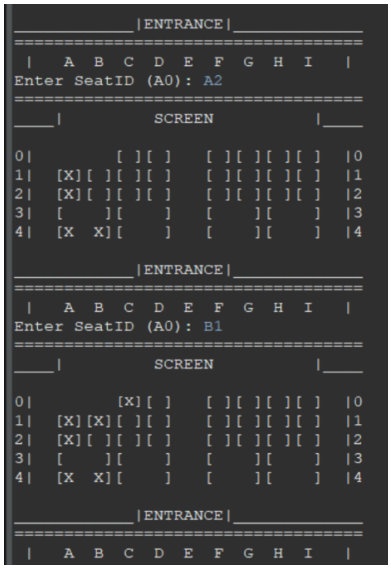
3. Multiple bookings at once:

```
Enter Booking ID (or enter -1 to exit): 1
Please enter the number of tickets that you want to buy: 4
```

Customer chooses to buy 4 tickets at once, and gets prompted to choose 4 seats.



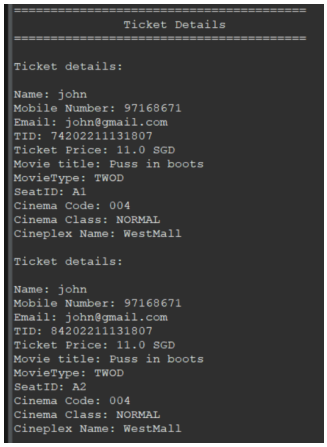
```
            |ENTRANCE|
==================================
  |   A  B  C  D  E  F  G  H  I   |
Enter SeatID (A0): A1
==================================
____|          SCREEN          |____

0|         [ ][ ]    [ ][ ][ ][ ]  |0
1|   [X][ ][ ][ ]    [ ][ ][ ][ ]  |1
2|   [ ][ ][ ][ ]    [ ][ ][ ][ ]  |2
3|   [    ][    ]    [    ][    ]  |3
4|   [X  X][    ]    [    ][    ]  |4

            |ENTRANCE|
==================================
  |   A  B  C  D  E  F  G  H  I   |
Enter SeatID (A0): A2
==================================
____|          SCREEN          |____

0|         [ ][ ]    [ ][ ][ ][ ]  |0
1|   [X][ ][ ][ ]    [ ][ ][ ][ ]  |1
2|   [X][ ][ ][ ]    [ ][ ][ ][ ]  |2
3|   [    ][    ]    [    ][    ]  |3
4|   [X  X][    ]    [    ][    ]  |4

            |ENTRANCE|
==================================
  |   A  B  C  D  E  F  G  H  I   |
```



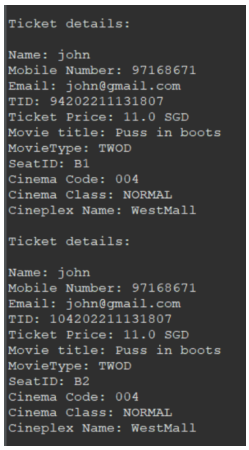
```
            |ENTRANCE|
==================================
  |   A  B  C  D  E  F  G  H  I   |
Enter SeatID (A0): A2
==================================
____|          SCREEN          |____

0|         [ ][ ]    [ ][ ][ ][ ]  |0
1|   [X][ ][ ][ ]    [ ][ ][ ][ ]  |1
2|   [X][ ][ ][ ]    [ ][ ][ ][ ]  |2
3|   [    ][    ]    [    ][    ]  |3
4|   [X  X][    ]    [    ][    ]  |4

            |ENTRANCE|
==================================
  |   A  B  C  D  E  F  G  H  I   |
Enter SeatID (A0): B1
==================================
____|          SCREEN          |____

0|         [X][ ]    [ ][ ][ ][ ]  |0
1|   [X][X][ ][ ]    [ ][ ][ ][ ]  |1
2|   [X][ ][ ][ ]    [ ][ ][ ][ ]  |2
3|   [    ][    ]    [    ][    ]  |3
4|   [X  X][    ]    [    ][    ]  |4

            |ENTRANCE|
==================================
  |   A  B  C  D  E  F  G  H  I   |
```

Customer chooses A1, A2, B1 and C0 respectively.

Customers get 4 tickets for the movie respectively.



```
==================================
            Ticket Details
==================================

Ticket details:

Name: john
Mobile Number: 97168671
Email: john@gmail.com
TID: 74202211131807
Ticket Price: 11.0 SGD
Movie title: Puss in boots
MovieType: TWOD
SeatID: A1
Cinema Code: 004
Cinema Class: NORMAL
Cineplex Name: WestMall

Ticket details:

Name: john
Mobile Number: 97168671
Email: john@gmail.com
TID: 84202211131807
Ticket Price: 11.0 SGD
Movie title: Puss in boots
MovieType: TWOD
SeatID: A2
Cinema Code: 004
Cinema Class: NORMAL
Cineplex Name: WestMall
```



```
Ticket details:

Name: john
Mobile Number: 97168671
Email: john@gmail.com
TID: 94202211131807
Ticket Price: 11.0 SGD
Movie title: Puss in boots
MovieType: TWOD
SeatID: B1
Cinema Code: 004
Cinema Class: NORMAL
Cineplex Name: WestMall

Ticket details:

Name: john
Mobile Number: 97168671
Email: john@gmail.com
TID: 104202211131807
Ticket Price: 11.0 SGD
Movie title: Puss in boots
MovieType: TWOD
SeatID: B2
Cinema Code: 004
Cinema Class: NORMAL
Cineplex Name: WestMall
```