SC2002 Object-Oriented Design & Programming

# MOBLIMA Report / Software Requirement Specification(SRS)

Qian Cheng (U2122085K)
Jeremy Ong (U2122774L)
Rhys Wong (U2121809G)
Kwek Ee Chern (U2122172C)
Yu Yifei Jason (U2120010H)

## APPENDIX B:
## 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 |
|---|---|---|---|
| Qian Cheng | SC2002 | SS11 | CHENG / 11-11-22 |
| Ong Jun Wei Jeremy | SC2002 | SS11 | Jeremy / 11-11 -22 |
| Rhys Wong | SC2002 | SS11 | Wong / 13-11-22 |
| Kwek Ee Chern | SC2002 | SS11 | Kwek / 13-11-22 |
| Jason | SC2002 | SS11 | Jason /13-11-22 |

## 1. Introduction

### 1.1 Purpose

This document describes the specifications and software requirements of MOBILIMA. MOBILIMA is a console-Based application to computerize the process of making online booking and purchase of movie tickets, listing of movies and sales reporting. It will be used by the moviegoers and cinema staff alike. The application serves as a main platform for making bookings throughout all the different cineplexes.

In this specifications report, we will highlight our design consideration, principles and use of OOP concepts. This report also includes a detailed UML class diagram, test cases for the programmes in the form of white box and black box testing, as well as a write up of future implementations.

### 1.2 Links

- Our video can be found at Youtube: https://www.youtube.com/watch?v=yUlwvRwmYz4

- Our code base can be found at Github: https://github.com/Worsl/OODP_SS11_GP3

## 2. Design Considerations

### 2.0.1 Encapsulation

Encapsulation refers to restricting the direct access to some components of an object. In our case, we practiced data hiding such that private attributes are only accessible via its own get and set methods. For our MOBILMA, we practiced encapsulation for all entity classes.

### 2.0.2 Abstraction

Abstraction is defined as the process of hiding implementation details and only showing the functionality to the user. In our design, the user will only interact with the relevant admin or moviegoer entrypoint interface. A user only knows and uses the functionalities of our MOBLIMA program and not the implementation details.

***SOLID design Principles implemented***

### 2.1.1 Single Responsibility Principle(SRP)

The SRP highlights the importance of any single object in OOP should be made for one specific function. This achieves loose coupling in the system thus changes in one component least affect existence or performance of another component.

For OUR MOBLIMA application, our classes are separated into entity, controller or boundary classes, the distinction in roles ensures that each class only has a single responsibility and the responsibilities are not coupled.

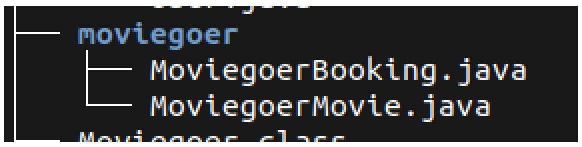
### 2.1.2 Open-Closed Principle (OCP)

OCP states that a class should be open for extension, but closed for extension. For instance in our preliminary design consideration. The **movieGoer** and **Admin** class both extend from the abstract class *User*. This allows specialization which the movieGoer and Admin have different methods

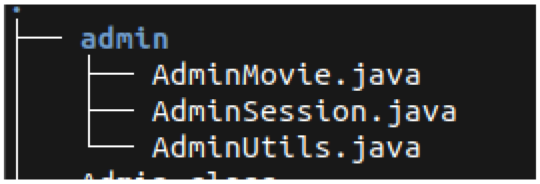
### 2.1.3 Segregation of Admin and MovieGoer

To prevent the user from accessing admin functions, that moviegoer and Admin have different entry points .

We do not want the users to have access to admin features, users should only be able to access the moviegoer program. Therefore we chose to separate them into different applications for security purposes. We have also separated the utility functions for moviegoer and admin into two different directories for organization.



```
├── moviegoer
│   ├── MoviegoerBooking.java
│   └── MoviegoerMovie.java
```



```
├── admin
│   ├── AdminMovie.java
│   ├── AdminSession.java
│   └── AdminUtils.java
```

*Advanced Design Patterns*

### 2.2.1 Facade structural Pattern

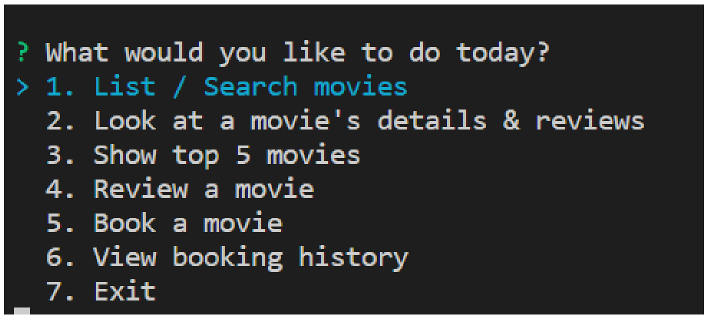
The Facade pattern provides a unified interface to a set of interfaces in a subsystem.

For a complex system like MOBLIMA, our starting interface provides an unified access common point, which branches out to other methods, modules and interfaces. Hence there is access to all **functionality through one single interface**.

*Non-functional Design requirements*

### 2.3.1 UI enhancements for selecting options

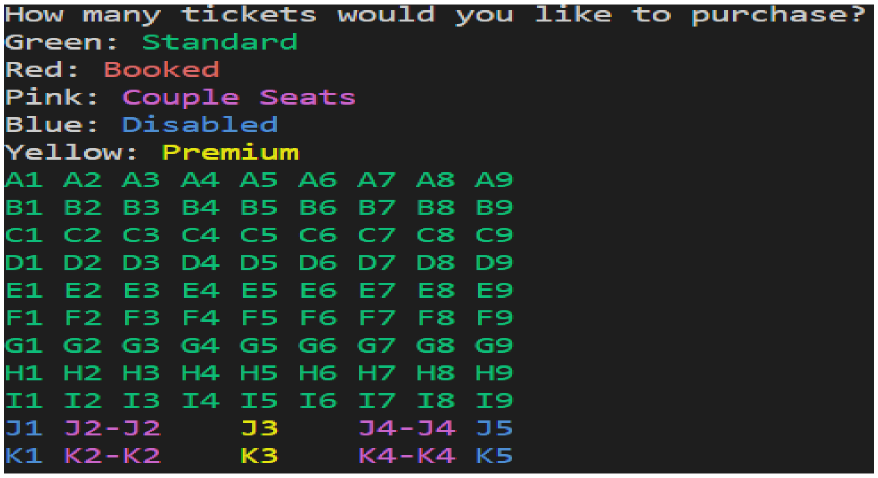
We have implemented a UI design interface that allows the user to select his desired options by simply scrolling the up and down arrow keys instead of keying in a number inside the terminal. Not only does this provide good user experience, it also prevents any of system misuse or potential bugs by providing invalid user inputs



```
? What would you like to do today?
> 1. List / Search movies
  2. Look at a movie's details & reviews
  3. Show top 5 movies
  4. Review a movie
  5. Book a movie
  6. View booking history
  7. Exit
```

### 2.3.2 Ticket selection screen

For a better user experience, we have implemented coloring coding to represent the different types of seats, as well as availability of seats.

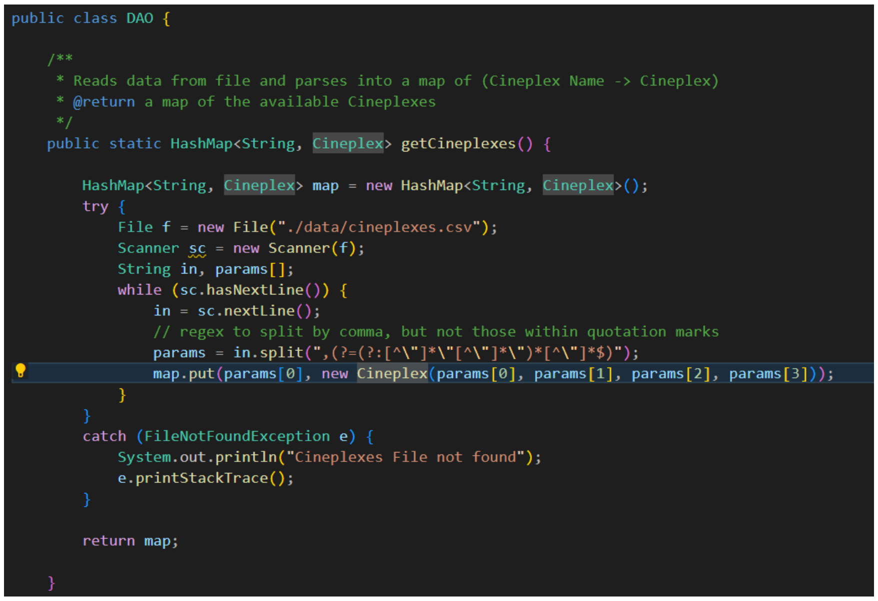


```
How many tickets would you like to purchase?
Green: Standard
Red: Booked
Pink: Couple Seats
Blue: Disabled
Yellow: Premium
A1 A2 A3 A4 A5 A6 A7 A8 A9
B1 B2 B3 B4 B5 B6 B7 B8 B9
C1 C2 C3 C4 C5 C6 C7 C8 C9
D1 D2 D3 D4 D5 D6 D7 D8 D9
E1 E2 E3 E4 E5 E6 E7 E8 E9
F1 F2 F3 F4 F5 F6 F7 F8 F9
G1 G2 G3 G4 G5 G6 G7 G8 G9
H1 H2 H3 H4 H5 H6 H7 H8 H9
I1 I2 I3 I4 I5 I6 I7 I8 I9
J1 J2-J2    J3    J4-J4 J5
K1 K2-K2    K3    K4-K4 K5
```

## 3. Data Storage

As the assignment mentions no database application or JSON , XML is to be used.  We have decided to comma separated value (**CSV) files** to store our data for the purpose of this application.

To further facilitate the management of data, we have a **DAO.java** class file that interacts with our data files. In this class, Data is read from a file and parsed into a HashMap structure. **Regex** (regular expression) is used to extract information from the text files to match our specific modules and method calls. Subsequently, additional information stored will be parsed as valid CSV records into our data files.



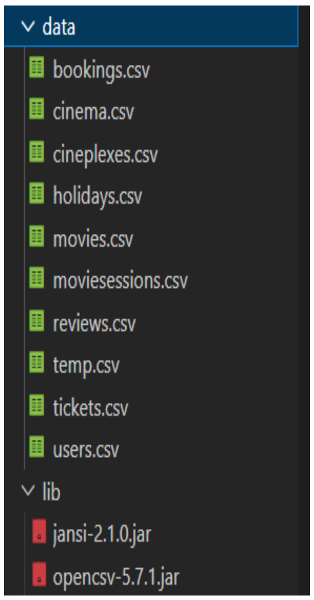
```java
public class DAO {

    /**
     * Reads data from file and parses into a map of (Cineplex Name -> Cineplex)
     * @return a map of the available Cineplexes
     */
    public static HashMap<String, Cineplex> getCineplexes() {

        HashMap<String, Cineplex> map = new HashMap<String, Cineplex>();
        try {
            File f = new File("./data/cineplexes.csv");
            Scanner sc = new Scanner(f);
            String in, params[];
            while (sc.hasNextLine()) {
                in = sc.nextLine();
                // regex to split by comma, but not those within quotation marks
                params = in.split(",(?=(?:[^\"]*\"[^\"]*\")*[^\"]*$)");
                map.put(params[0], new Cineplex(params[0], params[1], params[2], params[3]));
            }
        }
        catch (FileNotFoundException e) {
            System.out.println("Cineplexes File not found");
            e.printStackTrace();
        }

        return map;

    }
```



```
data
    bookings.csv
    cinema.csv
    cineplexes.csv
    holidays.csv
    movies.csv
    moviesessions.csv
    reviews.csv
    temp.csv
    tickets.csv
    users.csv
lib
    jansi-2.1.0.jar
    opencsv-5.7.1.jar
```

## 4. Test Cases

### 4.1 Testing on Basic Functionalities

We have designed test cases for the basic functionalities of the application. These test cases have been elaborated further in the video demonstration. The following are the test cases we have designed:

#### 4.1.1 Moviegoer Module

| **Test Case** | **Expected Outcome** |
|---|---|
| List Movies | The list of movies in the system are printed out |
| View Movie Details and reviews | The movie details and the list of reviews are printed |
| Show top 5 Movies | The list of top 5 movies are shown |
| Review a Movie | A review is added to a movie based on the user’s input |

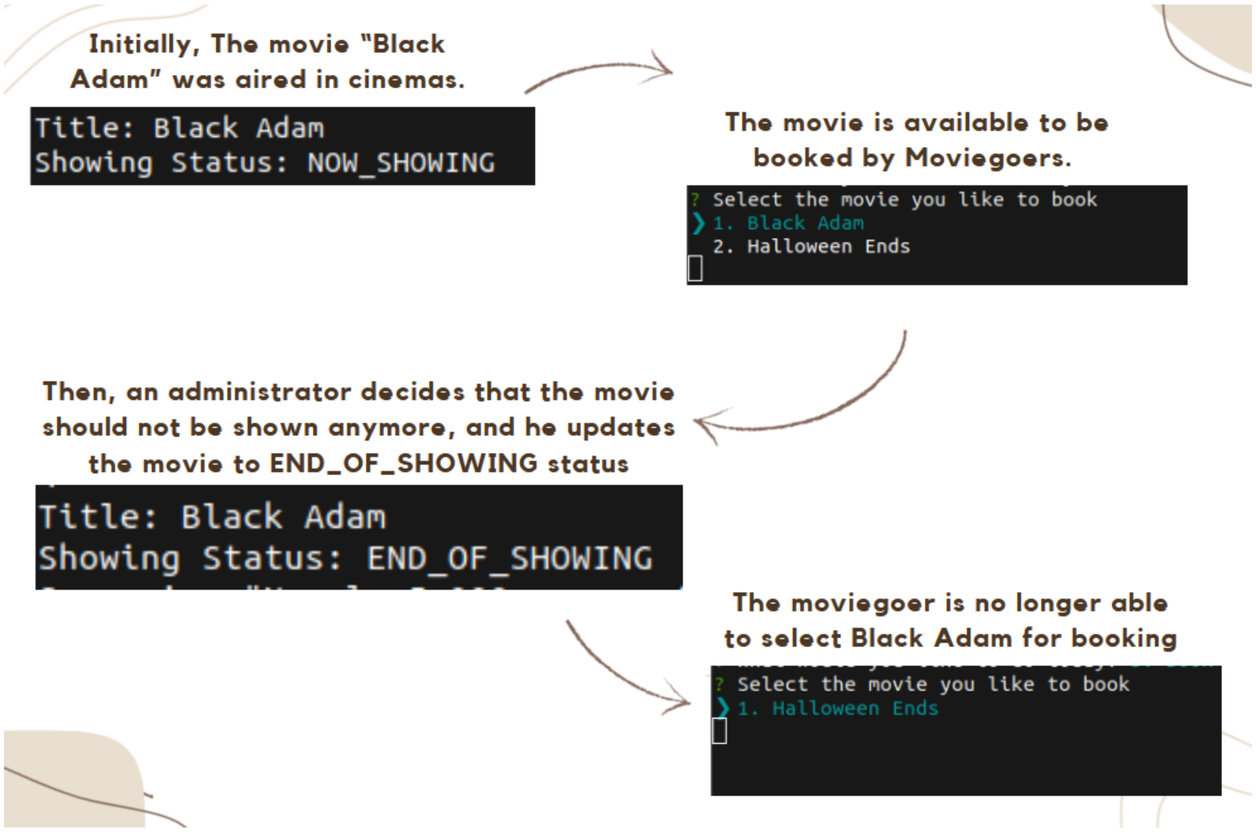
| | |
|---|---|
| Book a movie | A booking record is created with his tickets saved and the seats become marked as unavailable |
| View Booking History | The records of the current login user's bookings are printed out |

**4.1.2 Admin Module**

| **Test Case** | **Expected Outcome** |
|---|---|
| Add a new Movie | A new movie is saved into the system and the moviegoers are able to see the new movie |
| Update a Movie | An existing movie details is updated which is reflected to moviegoers |
| Delete an Existing Movie | The movie's showing status gets set to END_OF_SHOWING |
| Add a new Movie Session | A new session is saved into the system and the moviegoers will be able to select that session |
| Update a Movie Session | An existing movie session details is updated which is reflected to moviegoers |
| Delete a Movie Session | The session gets deleted and moviegoers will no longer be able to book that session |
| Add a holiday | Adds a new date to be a "Holiday", such that the price of booking is increased |

## 4.2 Movie Showing Status

This test case is to show that the showing status of a movie affects whether it should be shown when a Moviegoer is booking.


Initially, The movie "Black Adam" was aired in cinemas.
The movie is available to be booked by Moviegoers.
Then, an administrator decides that the movie should not be shown anymore, and he updates the movie to END_OF_SHOWING status
The moviegoer is no longer able to select Black Adam for booking

```
Title: Black Adam
Showing Status: NOW_SHOWING
```

```
? Select the movie you like to book
❯ 1. Black Adam
  2. Halloween Ends
```

```
Title: Black Adam
Showing Status: END_OF_SHOWING
```

```
? Select the movie you like to book
❯ 1. Halloween Ends
```

**<u>4.3 Price Difference depending on Day of Movie Session</u>**

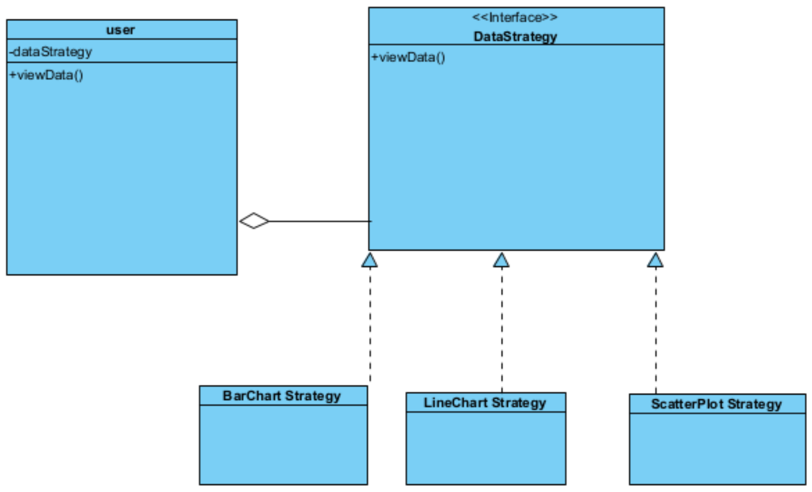
This test case shows that the price can be different depending on the day of the movie session, even though he may make the same choice for other options. This test case is clearly illustrated in the last portion of the demo video.

## 5. Proposed New Features

### 5.1 Data Visualisation presentator

The current feature that a user has is simply to view the top movies by a set of limited parameters, such as movie sales. We plan to introduce the feature of **data visualization which allows a user to better visualize the data** based on more specific representations (such as Bar graph, Line charts, Pie Charts etc.). With this data visualization, the movieGoers can make more decisions about choosing a movie to watch.

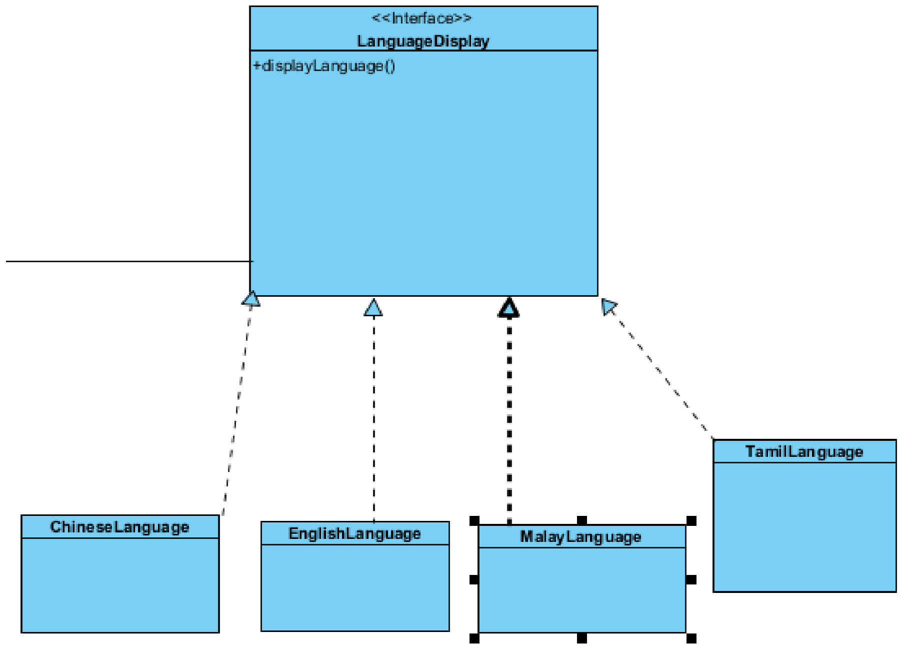
This form of data visualization can be implemented to our current source code easily via a behavioral software design pattern known as the **strategy pattern**. In short, the strategy pattern allows us to define a family of algorithms , and select them only at runtime. The following UML is a simple illustration of how this would be implemented



## 5.2 Multiple Language display Support

The current MOBLIMA only supports the English Language. However, like all other good interface systems, there should be a need to incorporate multiple system languages to cater to a diverse user base , especially in the sense of Singapore where most interface and booking systems support 4 languages.

We implement this by creating a LanguageDisplay interface. We have also adhered to the **Interface Segregation Principle (ISP)** so that the MOBLIMA system only relies on the interface but not the subclasses.  We have also adhered to the **open-closed principle** such that we can extend from the LanguageDisplay class by adding new other language displays without the need to modify the original source code. And upon starting the MOBILMA system, the user can select the language he would like to use and LanguageDisplay will display messages according to the user's preferred language.



## Appendix

### Class Diagram

Please refer to the attached PDF file (found in /UML_Diagrams directory) for class diagrams for clarity. Due to the large size, we are unable to place it in this documentation without having a clear image.

**UML Class Diagram**

RED: CONTROL

YELLOW: BOUNDARY

GREEN: ENTITY

### Login Credentials for Admin

Username: admin

Password: password

**Login Credentials for Moviegoer**

Username: Bob

Password: password

Alternatively, you can look at /src/data/users.csv to see the different user accounts. You can also add your own user account to the data file. More info can be found on the README.