**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.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Course | Lab Group | Signature / Date |
| LEE JUIN | SC2002 | SSP2 |  |
| OI YEEK SHENG | SC2002 | SSP2 |  |
| NG HONG JIN, JONATHAN | SC2002 | SSP2 |  |
| JERICK LIM KAI ZHENG | SC2002 | SSP2 |  |
| LEE CI HUI | SC2002 | SSP2 |  |

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.

Documentation Report

for

MOBLIMA

Version 1.0 approved

Prepared by

LEE JUIN,

JONATHAN NG HONG JIN,

OI YEEK SHENG,

JERICK LIM KAI ZHENG,

LEE CI HUI

SSP2 Group 1, Nanyang Technological University

2022-11-08

Table of Contents

Preface i

1. Table of Contents ii
2. Revision History iii
3. Introduction 1
   1. Document Conventions 1
   2. Class Model 1
   3. Testing 1
   4. Assumptions & Dependencies
4. Design Standards 4
   1. Software Engineering Principles 4
   2. Object-oriented Design Principles 5
   3. Proposed Features

Appendix A: Data Dictionary

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| LEE JUIN | 2022-11-08 | Initial write-up. | 1.0 |

# Introduction

## Document Conventions

This section covers the conventional standards used throughout this document.

**Font**: Times New Roman

**Code Font:** Courier New

**Heading:** Bold, Size 18

**Sub-heading:** Bold, Size 14

**Content:** Bold, Size 12

**Technical Standards:** IEEE 830-1998

## Class Model

The MOBLIMA Application is developed using Object-oriented Design Strategies. To begin development, the team started by constructing a concise and comprehensive class model, which comprises of two sub-models —— Entity Class model and Control Boundary Class model. The former helped the team in developing persistent data relationships while the latter depicted a general overview of the program flow.

The development team proceeds to integrate the constructed models to form the over-arching system architecture. Further details regarding the considerations on the system architecture design can be found under section *2. Design Standards*.

The Class Models are presented as follows. A supplementary copy of each class model diagram is provided independent of this document for clarity.

### Diagram Description automatically generatedEntity Class Model

### Control Boundary Class Model

## Testing

After development, the MOBLIMA Application has undergone several Unit Testing and Integration Testing. This section covers the techniques used during the testing phase. Supplementary copies of the testing results in the form of pictorial screenshots are provided.

### White-box Testing

### Grey-box Testing

## Assumptions & Dependencies

This section covers all the assumptions made by the development team during the development phase. This section also serves as a repository for all dependencies assumed throughout the development of the project.

### Assumptions

* The MOBLIMA Application is designed for no concurrent user. Hence, exceptions such as race condition from multiple users booking the same seat are not handled by the application.
* The MOBLIMA Application is designed with a pseudo-manual notion. No external server is hosting the MOBLIMA Application. Hence, actions such as removing a showing time can only be done manually upon start-up of the application by the administrator.
* The MOBLIMA Application is designed with a simple login feature. No help is provided for lost account retrieval.
* The MOBLIMA Application does not verify the legitimacy of Movie Goers who self-proclaimed as either Student, Child, or Senior Citizen. The verification shall be done by the vendor instead.
* The MOBLIMA Application does not verify the legitimacy of Movie Goers’ payment information.
* The MOBLIMA Application assumes that a movie may have multiple showing types, i.e., a movie can be showed as a 3D movie, or a standard movie.
* The MOBLIMA Application assumes that a movie’s review is independent of its showing type. Hence, all reviews under the same movie title are displayed to the user upon query, regardless of the showing types available.
* The MOBLIMA Application assumes that a movie’s title is always unique. Hence, no exception handling regarding movie of the same name is done.

### Dependencies

|  |  |
| --- | --- |
| Java support | The MOBLIMA Application is developed under Java SE 7. |
| Command-line Interface (CLI) support | The MOBLIMA Application requires a terminal window to operate.  The Command Prompt from Windows or Terminal on MacOS will suffice for the application. |

# Design Standards

## Software Engineering Principles

This section covers all the good Software Engineering principles practiced and applied throughout the development process.

### Model-View-Controller (MVC) Design Pattern

The MOBLIMA Application adopts a Model-View-Controller (MVC) system architectural design pattern. The Model component holds all persistent data of the application, while the Controller component represents all the business logic behind the application. Finally, the View component is the CLI presented to the Movie Goers.

The development team decides to adopt the MVC architectural pattern as it allows segregation of code between components. This allowed each developer to work independently and simultaneously on different components, significantly speeding up the development process.

Text

Description automatically generated

**MVC**

### Version Control

The development team uses GitHub for version control. YADA YADA

## Object-oriented Design Principles

### Liskov Substitution Principle

In the View component, the development team applied the Liskov Substitution Principle. Each View class inherits from the MainView class. The children View classes implements the printMenu() and appContent() methods without reducing any functionalities, and thus behaving the same way as MainView class. Hence, the MainView class is replaceable by its children classes.

Text

Description automatically generatedA code snippet of the MainView class is shown in Figure.

### Diagram Description automatically generatedOpen-Closed Principle

In the Model component, the development team adopted the Open-Closed Principle when building the model for Movie, Cinema and Movie Goer. A section of the modified Entity Class Diagram for Movie is shown in Figure.

Suppose we wish to support a new Movie type, Dolby Atmos movie, in our application. To do so, we only need to create a new class called DolbyAtmosMovie and inherit from the Movie class. Since methods such as getMoviePrice and setMoviePrice are abstract methods to be realized, we do not need to modify the Movie class to include the price for Dolby Atmos movies. On top of that, suppose that Dolby Atmos movies have additional attributes or methods, we can directly implement those attributes and methods in the DolbyAtmosMovie class without modifying the Movie class.

Hence, the Movie class is effectively *closed* for modification, but *open* for extension to support more types of movies.

### Text Description automatically generatedDependency Inversion Principle

*Figure*

Graphical user interface

Description automatically generated with medium confidence

Building on top of the concept of Open-Closed Principle, the development team also applied the Dependency Inversion Principle such that the business logic found in all the Controller classes depend on the abstract and interface classes, instead of the concrete classes.

An example is shown in Figure and Figure. As mentioned, the development team applied the Open-Closed Principle when building models. As such, the model MovieGoer is abstracted and extended by concrete classes such as Adult, Child, Student and SeniorCitizen.

However, all Controller and View classes uses and depends on the abstract class MovieGoer instead, as shown in Figure. This allows looser coupling between the Controller classes and the Model classes, as instead of depending on the concrete classes like Adult or Student, the Controller class only interacts with the abstract MovieGoer class. Hence, when there are changes to be made in the concrete classes such as Adult, we do not have to change anything in the Controller class.

*Figure*

## Proposed Features

1. Food Payment (Segregation\_
2. Add new LoyaltyPoints concrete class Payment, add new attribute to MovieGoer (loyalty points)