

**ASSIGNMENT CE/CZ2002: OBJECT-ORIENTED DESIGN & PROGRAMMING**

**BUILDING AN OO APPLICATION (MOBLIMA)**

**LAB: SS5**

**GROUP: 3**

**MEMBERS:**

**BALAMURUGAN DARSHINI**

**CALLISTA ROSSARY CHANG**

**CHEW YEE HIAN**

**PENELOPE CHUA HUI ENG**

Contents

[Report Overview 3](#_Toc25000193)

[Design Considerations 3](#_Toc25000194)

[Design Principles 4](#_Toc25000195)

[Assumptions 5](#_Toc25000196)

[Use of OODP concepts 6](#_Toc25000197)

[Future Considerations 7](#_Toc25000198)

[UML Class Diagram 7](#_Toc25000199)

[UML Sequence Diagram 9](#_Toc25000200)

[Test Cases (not shown in video) 10](#_Toc25000201)

[Declaration 13](#_Toc25000202)

# Report Overview

This report details the design process of building the MOBLIMA application. By embedding knowledge of Design Principles and the features of Object-Oriented Programming in our considerations, we have created a Cinema Operator Application using Java on Eclipse IDE, which handles the user and database needs of a fictional Cinema Chain, MOBLIMA.

# Design Considerations

**Boundaries**

The Boundaries Package contains all the applications that manage the flow of information between Managers. MoblimaApp is the main user interface for our application. (handles access privileges for the users?)

After identification, there are 4 main Apps:

UserApp, which is accessed by guest users without an account. This allows limited access privileges – guests can only view movie information, such as seat availabilities and movie reviews. To book a ticket, users require an account.

CinemaStaffApp, which is handled by the cinema staff. This allows cinema staff to edit information related to the entities, such movie listings, showtimes, and prices.

MovieGoerApp, which is accessed by Customers. This allows customers to book movie tickets, view their booking history and leave more reviews.

InputBoundary, which ensures the validity of user inputs. This ensures that the application is resistant to crashing due to invalid inputs or out of range inputs by users.

These apps execute the primary functions expected of our application, getting user input and passing the information on to the respective entities’ Managers.

**Managers**

Managers act as controllers, keeping the ‘Main Package’ and the ‘Models Package’ separate and handling all the main logic of the application. Managers act as controllers for their respective entities, directing the data traffic between entities and ensuring the correct information is stored.

**Models**

Models are the entities which store the information needed in our application. This information by the Managers via accessors and mutators, hence all the objects created are up to date with the changes made by the user during the running of the application. As our data is stored in csv files, Models pass on the temporary information during runtime to Serializers, which handle permanent data storage.

**Serializers**

Serializers aid in the process of managing the databases. With serializers, changes to the database will be permanent by adding them to external CSV files. 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 vice versa during deserialization (when reading from the database).

# Design Principles

**Single Responsibility Principle:** **Only one reason for a class to change**

Each class is created for a single purpose, without mixing responsibilities. For example, in the booking of tickets, price must be calculated per transaction. Instead of adding the functionality to BookingManager, a separate class and entity for the pricing scheme was devised to handle this responsibility, with the logic for the calculation executed by PricingCalculator, and the pricing scheme stored within Pricing.

**Open-Close Principle: Open for extension, close for modification**

For the Cinema abstract class, cinema classes such as PlatinumCinema and GoldCinema implement Cinema as a base class. This allows for easy extension if the cineplex were to decide to add new cinema classes, with the new cinema class implementing Cinema’s main functions, instead of directly modifying one general Cinema class, which would result in a large ripple effect.

**Liskov-Substitution Principle: Subtypes must be substitutable for base types**

Upon starting up the application, users can enter their mode of access. For resultant CinemaStaffApp and MovieGoerApp menus, they are extended from base type UserApp. Hence, they take in the same parameters as UserApp and give the necessary outputs in the code. This follows the basic notion of LSP, with the subtypes expecting no more than the base type and providing no less.

**Interface Segregation Principle: Client specific interfaces are better than one general purpose interface**

While our application does not use many interfaces, abstraction was used extensively to similar effect. For Serializable Package, all classes are abstractions extending from AbstractSerializer. Entity-specific serializers were thus created to import and export data between the entities in Models and the csv databases.

**Dependency Injection Principle: High-level and Low-level modules should depend on abstractions rather than on each other; Details should depend upon abstractions, but abstractions should not depend on details**

The Managers Package links the Main Package and the Models Package. Thus, Main and Models need not be aware of each other, and only deal with Manager, preventing direct interaction between them. This allows both Main and Model to be designed independently without information of the other.

# Assumptions

1. Only customers with MOBLIMA account can book movie tickets.
2. Once booking is completed, the tickets cannot be cancelled, and no refunds are provided (i.e. Booking of movie tickets is irrevocable).
3. Student tickets can be purchased online without validation. The validation will be done upon entering the cinema.
4. The filtering process was designed on the preference of the user. It is assumed that customers prioritize movies and staff prioritize cineplexes where they work at.
5. The booking process is designed with the assumption that customers are aware of the pricing scheme (ex. The base price, discounts and premiums for different cinema class).

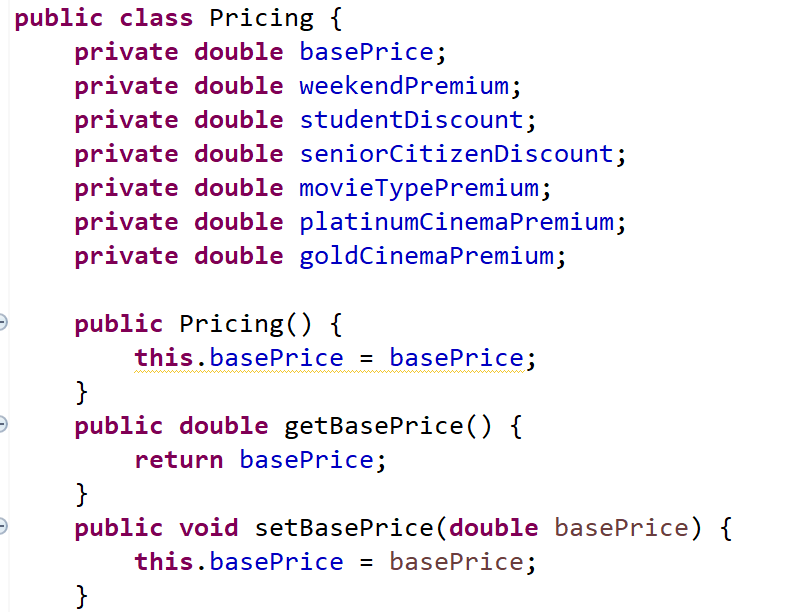
# Use of OODP concepts

**Abstraction**

Abstraction allows simplification and creates more useful and reusable tools. In MOBLIMA, various classes and methods are created using abstraction. For example, cinema is created as an abstract class and is extended to three sub-classes: GoldCinema, PlatinumCinema and RegularCinema. Similarly, AbstractSerializer is created as an abstract class and is extended into each of the entity-specific serializer class. This allows the general information present in the superclass to be used in the sub-classes and reduces the complexity and repetitiveness of the code.

**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 MOBLIMA, all attributes under the package “Models” are private. This ensures that the data is safe within each class. An example of encapsulation is as below:



As shown, all attributes are declared as private and public methods such as get and set are used to access these data from other classes.

**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. This is shown in UserApp, which stores all the common functions for any user type. Inheritance allows MovieGoerApp and CinemaStaffApp to use all these functions and simply add the necessary specific functions for the user, reducing duplicates in the code.

**Polymorphism**

Polymorphism refers to the ability to take more than one form and an operation may exhibit different behaviours under different instances. Hence, polymorphism is used to augment the use of inheritance and interface is a necessary tool to implement polymorphism. The interface “ISerializable” contains a single unimplemented method that is used as the base to independently implement in each of the class in the “Models” package.

# Future Considerations

1. **More access types**

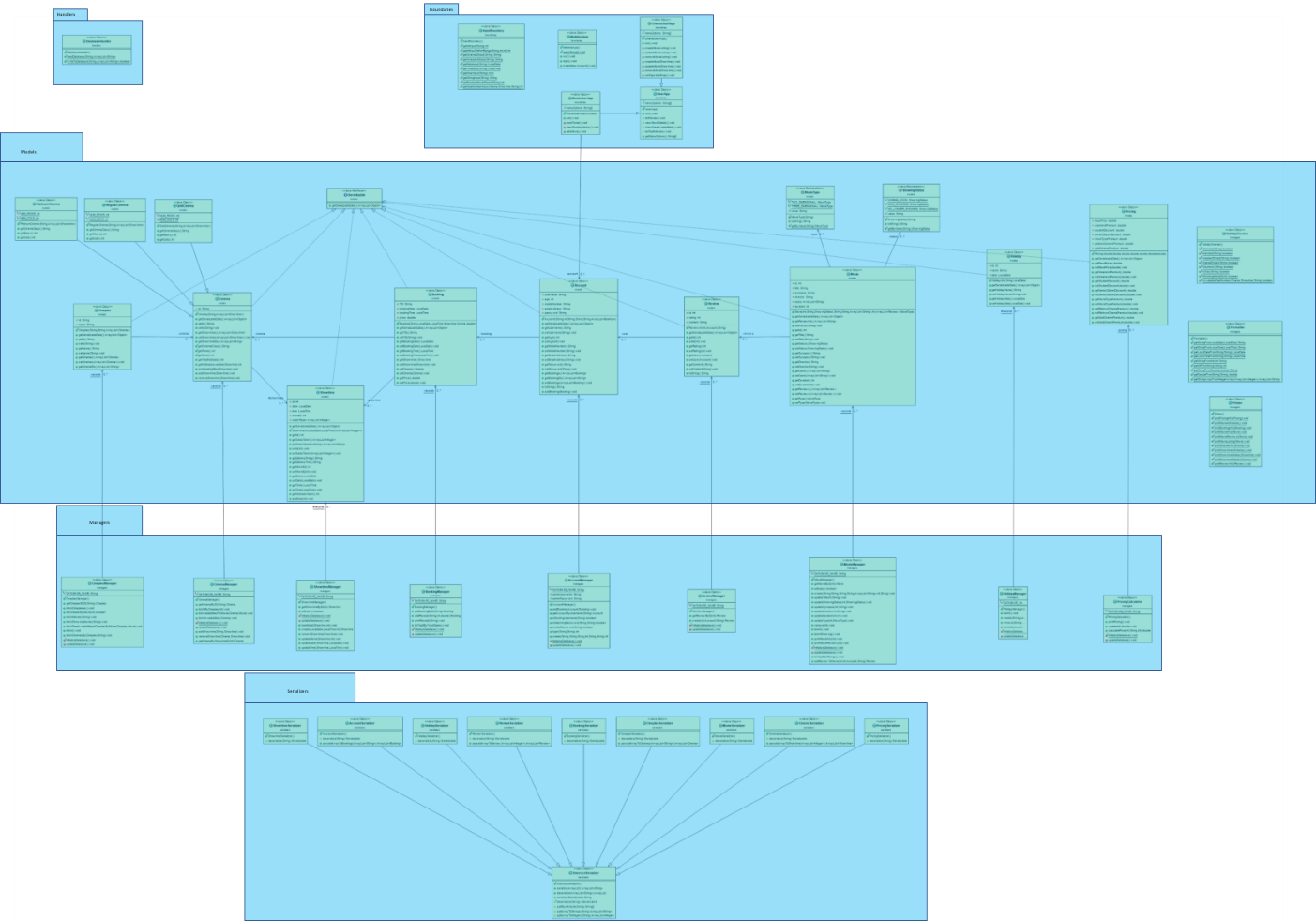
If more user types were necessary, apart from Customer and CinemaStaff, they can be created via extending the UserApp, through the Liskov-Substitution Principle. This would allow management to tier levels of access privileges within the staff. For example, a JuniorCinemaStaffApp could be created from the UserApp, highlighting reusability. This app would afford CinemaStaff privileges but with restricted access to editing the information, for lower-ranking staff.

1. **Adding a new attribute to Movies**

Apart from the current movie details contained in the Movie entity, adding a new field or attribute to Movie would be simple, showing its extensibility. If MOBLIMA wanted to add an age rating to Movies, for example, this can easily be achieved by editing the Movie entity and MovieManager, without many severe changes. This is due to the MovieManager handling the logic of the Movie Entity, without the Movie Entity directly interacting with the Main apps via the Dependency Injection Principle.

# UML Class Diagram

Two class diagrams were created – one without dependency and one with dependency. The diagram below shows the class diagram that does not contain dependency. Dependency was not included for the following diagram to improve readability. The MOBLIMA folder can be referred to view the other class diagram.

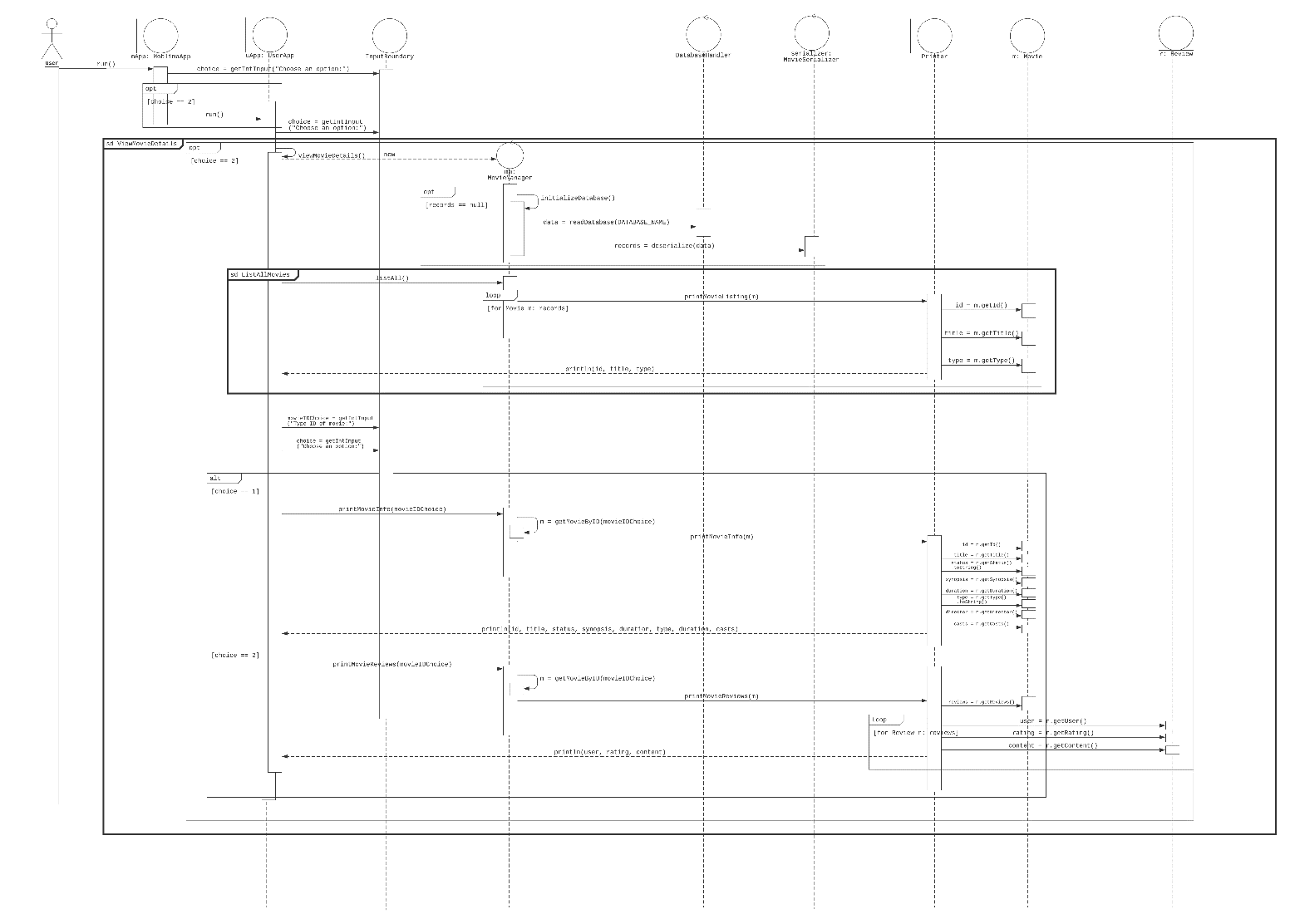


(Please refer to the file labelled “class\_diagram.pptx” in the zip folder for a higher resolution image).

# UML Sequence Diagram

The UML sequence diagram illustrates the process of the following:

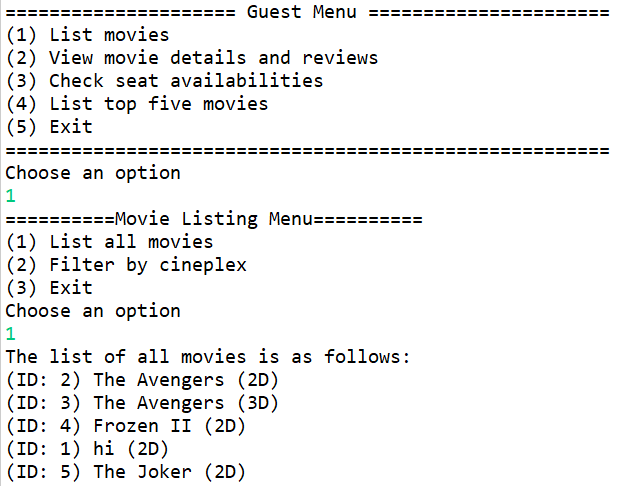
1. Search/List movie
2. View movie details – Including reviews and ratings



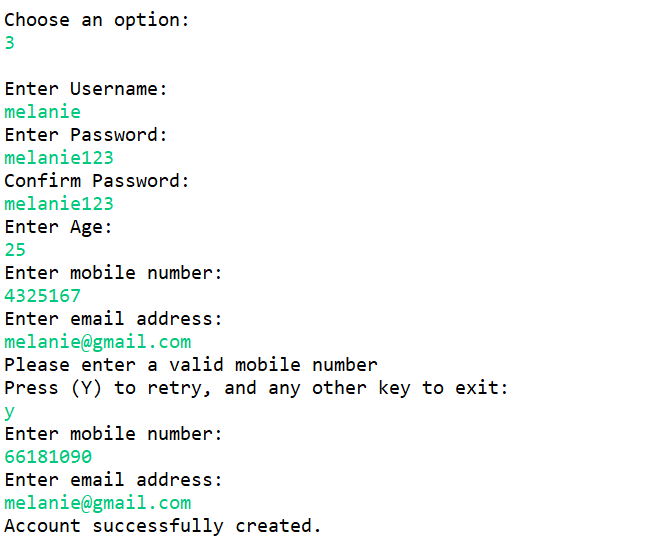
(Please refer to the file labelled “sequence\_diagram.png” in the zip folder for a higher resolution image).

Test Cases (not shown in video)

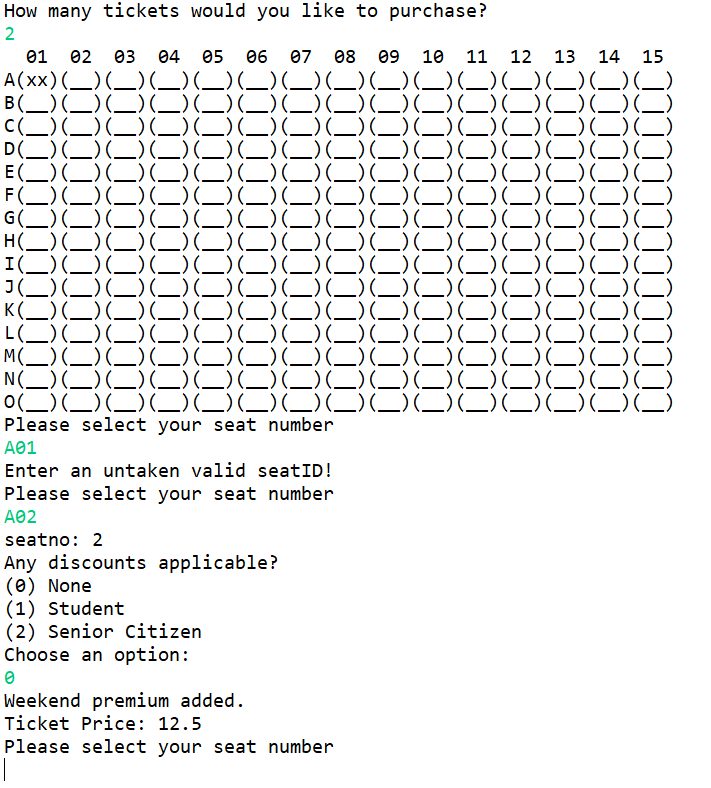
1. List all movies, with no conditions stated:



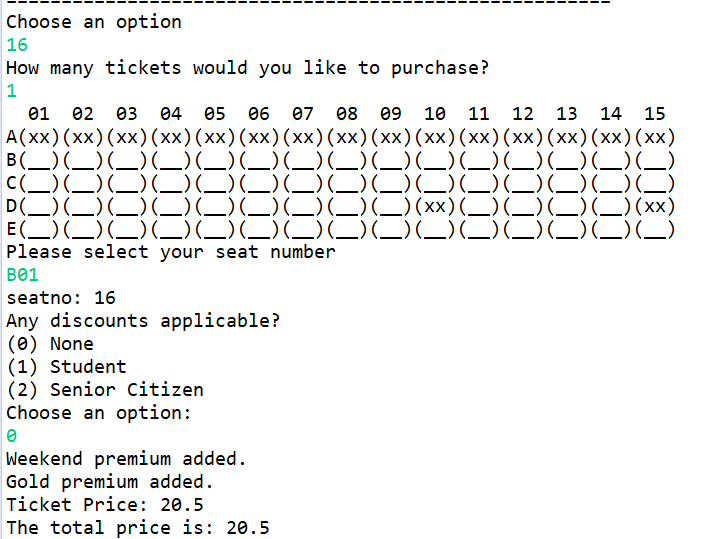
1. Error when mobile number does not start with 6,8 or 9.



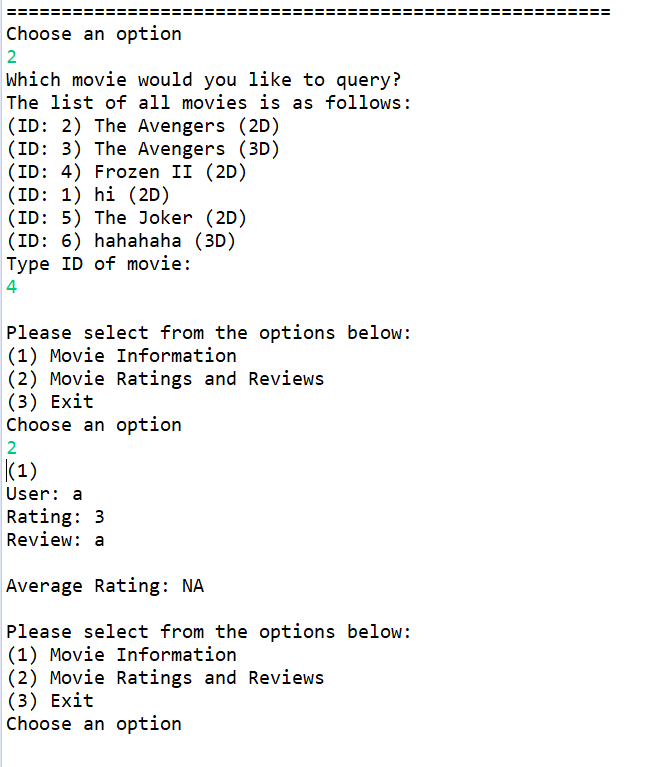
1. Selecting an already taken seat:



1. Weekend and movie class premium:



1. No overall rating shown when there is only one review:



1. Different statuses of movies (Coming Soon, Now Showing, No Longer Showing):

|  |
| --- |
|  |
|  |
|  |

# Declaration

**Declaration of Original Work for 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 |
| Balamurugan Darshini | CZ2002 | SS5 |  |
| Callista Rossary Chang | CZ2002 | SS5 |  |
| Chew Yee Hian | CZ2002 | SS5 |  |
| Penelope Chua Hui Eng | CZ2002 | SS5 |  |

Important notes:

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