ASSIGNMENT REPORT

SC/CE/CZ2002: Object-Oriented Design & Programming

By:

Pavanraj Selvaraju (U2122616H)

Tan Jin Wei Daniel (U2121315A)

Yan Renyu (U2121734E)

Zheng Rongtao (U2122337D)

Zhou Xuhang (U2120264L)

<u>CONTENTS</u>

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 (CE2002 or CZ2002)	Lab Group	Signature / Date
Pavanraj Selvaraju	SC2002	SE3	[illegible] 13/11/2022
Tan Jin Wei Daniel	SC2002	SE3	[illegible] 13/11/2022
Yan Renyu	SC2002	SE3	yan 13/11/2022
Zheng Rongtao	SC2002	SE3	[illegible] 13/11/2022
Zhou Xuhang	SC2002	SE3	Zhou Xuhang 13/11/2022

Design Considerations

1) **Encapsulation & Information Hiding**

   Ensures that the private data of an object is protected by building a barrier to prevent other classes from directly accessing the data.

   - Public/Protected/Private: Encapsulation was executed by ensuring that the attributes are declared as private within its specific classes when necessary. By doing this, users are only allowed to know what the class does and how methods are called to perform the required tasks. We can only access the attributes of other classes through the implementation of public get() and set() methods.

2) **Loose Coupling & High cohesion**

   One of our main goals when designing our program is to achieve loose(low) coupling and high cohesion.

   - Loose(low) Coupling: We designed our program in a way where classes are only linked to a few necessary classes. Coupling between classes was minimized to ensure that if there was a need to make changes to a class, the number of related classes affected was limited. In our UML class diagram, there is no bidirectional association as this would mean that objects are dependent on one another to work. This would then suggest a case of tight coupling.
   - High cohesion: Our program was designed to ensure that each class created had a single and well-focused purpose. A more focused class would indicate a greater cohesiveness of the class. This relates to designing with maintainability and reusability in mind.

3) **Reusability**

   This refers to the practice of using existing code for a new function. Reusability is a great advantage of inheritance.

- Booking/Seat/Ticket: These implementations possess common variables and methods that could be reused in other future implementations. For instance, in order to create a program related to booking seats of an event in the future, the existing code could be reused in that program successfully.

Additional Enhancement Features:

Proposed feature 1) Movie Recommendation

A possible feature we could include is a feature that is able to recommend users new movies based on the movies they have watched previously. We can do this by sorting the movie list based on the genre of movie that the user has most watched. By implementing this feature, we can incorporate the following design principles:

- Extensibility: We can implement this feature by adding a printMoviesHistory() in the MovieController class. As our code is extensible, this allows the list of movies to be printed out based on the booking history of the users. This method would only be called if users are logged into the system and have made at least a booking before.

- Reusability: A new class (eg: SortingController) can reuse the existing methods that were used to view the booking history of users, which are in the BookingController class. Hence, we are able to prevent any repetitiveness of our code.

Proposed feature 2) Ordering Food

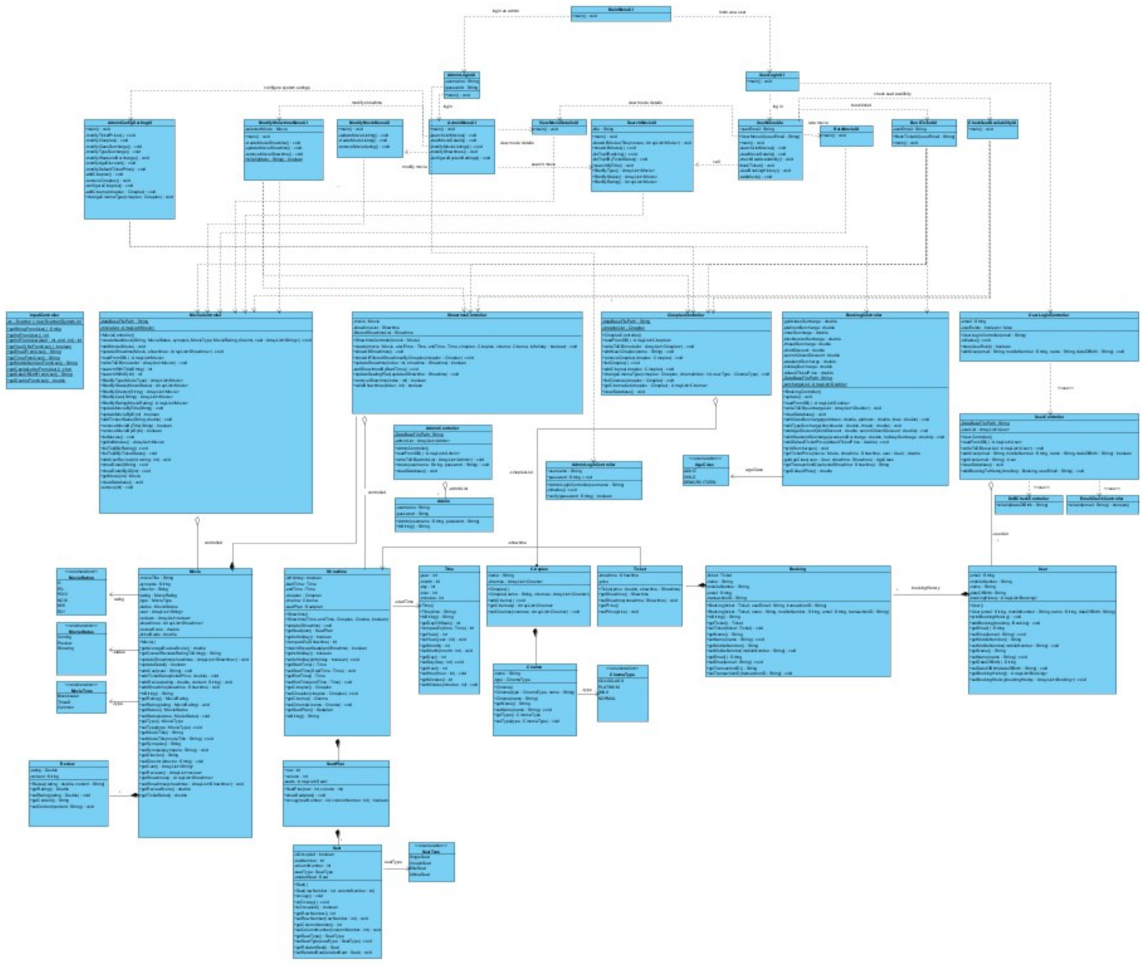
A second feature we could include is to allow users to have the option of purchasing snacks when booking their movie tickets. We can do this by creating a new interface called ViewFoodOrder. The implementation of this feature incorporates the following design principles:

- Single Responsibility Principle (SRP): Creation of an interface class that only has one specific responsibility of giving the users a choice to purchase movie snacks when buying their tickets. By doing so, we are also able to achieve high cohesion through the use of the Single Responsibility Principle (SRP).

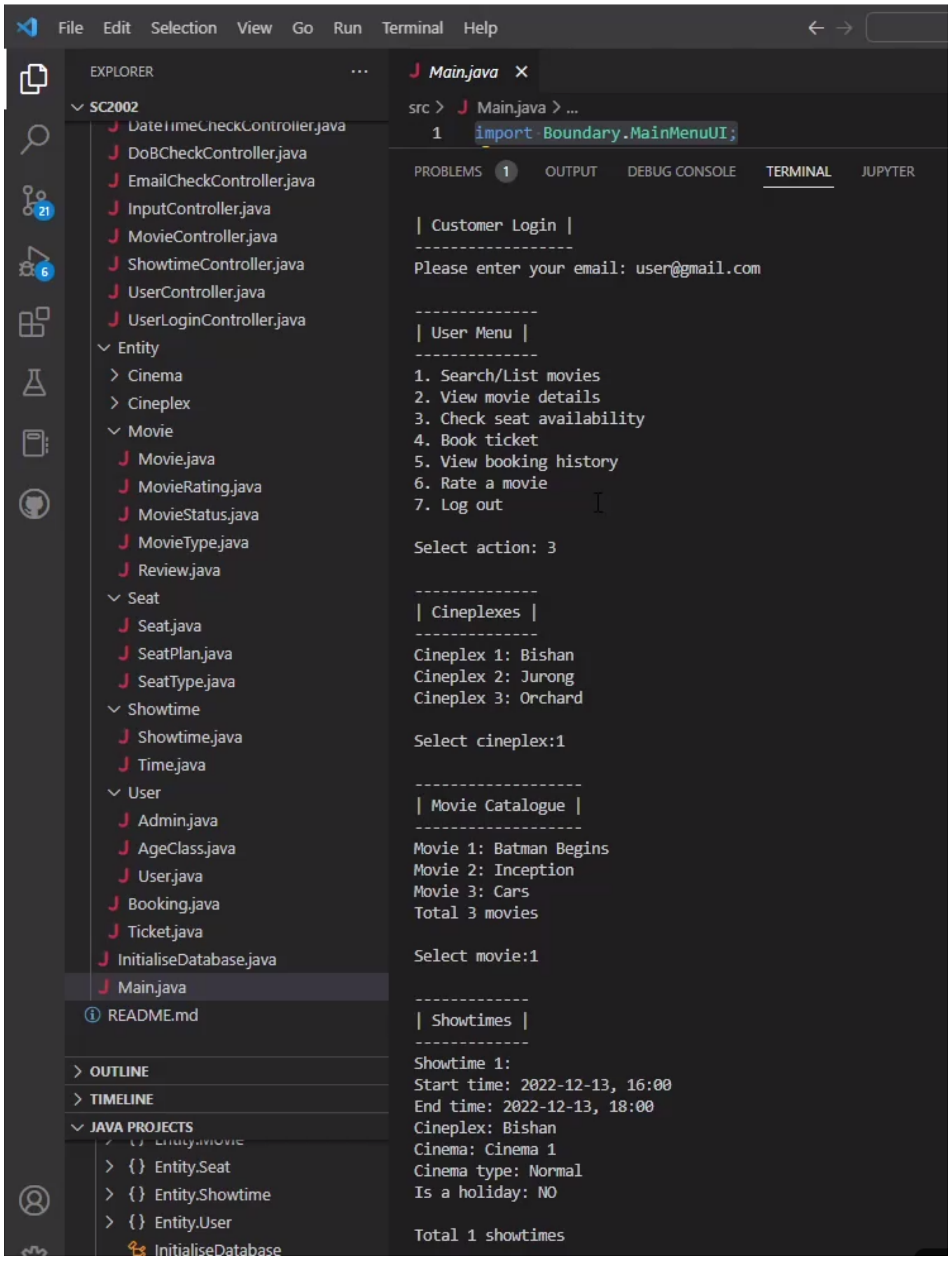
- Open-closed principle/Inheritance: The feature is achievable by extending our current interface class to the new ViewFoodOrder class which can display movie snack options that users could purchase.

UML Class Diagram

Refer to the class diagram attached separately.



Testing



```java
import Boundary.MainMenuUI;
```

```
| Customer Login |
------------------
Please enter your email: user@gmail.com

-------------
| User Menu |
-------------
1. Search/List movies
2. View movie details
3. Check seat availability
4. Book ticket
5. View booking history
6. Rate a movie
7. Log out

Select action: 3

-------------
| Cineplexes |
-------------
Cineplex 1: Bishan
Cineplex 2: Jurong
Cineplex 3: Orchard

Select cineplex:1

------------------
| Movie Catalogue |
------------------
Movie 1: Batman Begins
Movie 2: Inception
Movie 3: Cars
Total 3 movies

Select movie:1

------------
| Showtimes |
------------
Showtime 1:
Start time: 2022-12-13, 16:00
End time: 2022-12-13, 18:00
Cineplex: Bishan
Cinema: Cinema 1
Cinema type: Normal
Is a holiday: NO

Total 1 showtimes
```