CZ2002: Object-Oriented Design & Programming

Movie Booking Project (SS7 – Group 5)

Christofer Himawan

Sebastian

Terence Andre

Padhi Abhinandan

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	Lab Group	Signature/Date
Christofer Himawan	CZ2002	SS7	[illegible] 17/11/19
Padhi Abhinandan	CZ2002	SS7	Abhinandan Padhi 17/11/19
Sebastian	CZ2002	SS7	[illegible] 17/11/19
Terence Andre	CZ2002	SS7	[illegible] 17/11/19

Content

1. Introduction

The task is to code an application to book a movie in a cinema. This application will be used by a user or staff. For a user, there will be multiple functions such as booking a movie and choosing the seats, giving review and rating, and showing their own history. For staff, the functionality will be to update, add, and delete movie, price, user-age, and status. The implementation of these functionalities will be based on object-oriented principles.

2. Principle and OO design

2.1 Encapsulation

We differentiate between database, logic, and interface to make sure the data is secure and encapsulated from the user. The interface will be the one that directly interacts with this application's user. Logic classes will handle the processing and make sure that the link to the database must pass through the logic to be processed. This way, any changes made to the logic class may not cascade to changes in both entity class and interface class, achieving the loose coupling.

2.2 Polymorphism

The implementation of polymorphism is in the class handling all the price modifiers (*IPrice*), the cinema type classes (*CinemaType*), and the status of the movie (*MovieStatus*).

The *IPrice* class is an interface that will be inherited by *Day*, *UserAge*, and *Types*. The *Day* class will be inherited by modifiers related to the movie date, *UserAge* class will be inherited by price modifiers due to the customer's age group, and the *Types* class will be inherited by price modifiers due to the cinema's quality. Polymorphism creates efficiency for these classes because we want to preserve the reusability where every booking will have a different combination of user age, day, and cinema type. It is possible to add new classes to each of them easily without changing the parent class, and there is no change needed to the logic and interface class.

The *CinemaType* and *MovieStatus* class are abstract classes which are another way to achieve polymorphism. The *CinemaType* class will be inherited by different possible cinema's qualities while the *MovieStatus* class will be inherited by the possible status of the movies. The assumption is that the cinema's quality will determine the dimensions of the cinema room. It is possible to make an enum or map to store the corresponding dimension for a cinema type, but by creating classes for each type, it enables further expansion without breaking other classes. The same applies to the *MovieStatus* class.

2.3 SOLID Implementation

2.3.1 Single Responsibility

We differentiate between the user logic, movie logic, staff logic, ticket logic, holiday logic, and system update logic. Each of them will have a single specific function. User logic will control the review and history load. Movie logic will take care of the booking. Staff logic will take care of the login authentication for staff. Ticket logic will take care of the purchase-related data. Holiday logic will take care of the dates of a holiday which can be adjusted by the staff. System update logic will take care of the pricing problem. Additionally, entity classes will only be responsible to be the database and interface classes will be responsible to communicate to the application's user.

2.3.2 Open-Closed Principle

The system of inheritance in the code such as the price modifier classes achieves this. For instance, the higher subclasses, such as the *Day* class, will be a more general blueprint on how a certain day, cinema type, or user age group modify the price. Their subclasses will implement the specifics on how exactly they are modifying the price. Therefore, the higher subclasses are closed for modification, but the lower subclasses are allowed to extend the implementation further.

2.3.3 Liskov Substitution Principle

For every class and their subclasses, it is ensured that the superclass can be substituted by their subclasses since the subclasses do not demand more, returns no less. For example, *IPrice* can be changed by its subclasses and the program will still run. In fact, the class is used to achieve polymorphism in price modifiers.

2.3.3 Interface Segregation Principle

The use of multiple interfaces for cinema type, movie status, and price achieves interface segregation principle. We do so because each movie will have a different cinema with different movie status and price, so to make sure each movie will have different functionality. Since there are no common properties between cinema type, movie status, and price, it is better to split them in this way rather than combining them into a single interface.

2.3.4 Dependency Injection Principle

This is implemented by using interface classes as an abstraction layer between the logic classes and the entity classes. For logic, there will be many classes with specific work while the entity classes hold the data and handle how to receive and update their data. This will make sure that the logic will not be dependent on the entity class and vice versa. Both will depend on abstraction.

2.4 Singleton

Since we use the text file to store objects which are the database, we need data consistency so that the classes using the data will be the correct data. Thus, for every call to the database, singleton makes sure there will no more than one class of its type in runtime, allowing every class that requires the data refer to the same database.

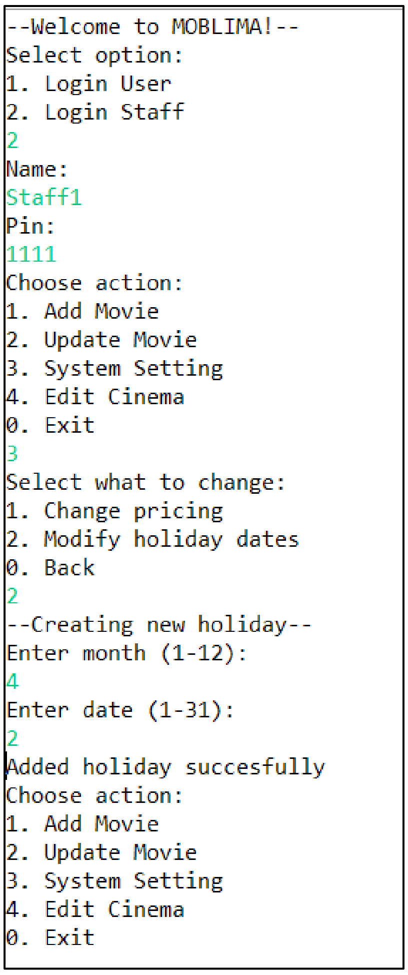
3. Additional Feature

3.1 Price in Percentage

In terms of price, we implement the pricing according to the type of cinema, day of the week, and the user-age. For standardization where different combinations will provide different prices, we use percentage to calculate. For instance, the gold cinema will be 20% more expensive then the silver cinema, so gold cinema will have a variable of 1.2 which will be multiple to the base price.

3.2 Update Holiday date

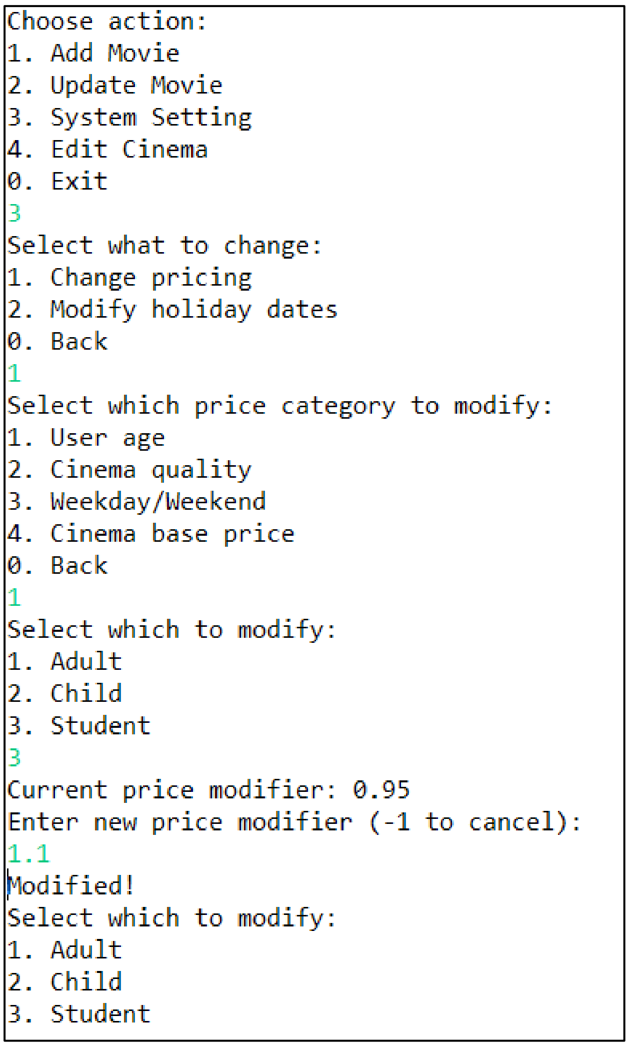
Staff can add a new holiday date. A possible real application of this is the ability to create a discount for certain internal events, such as celebrating their company’s birthday.



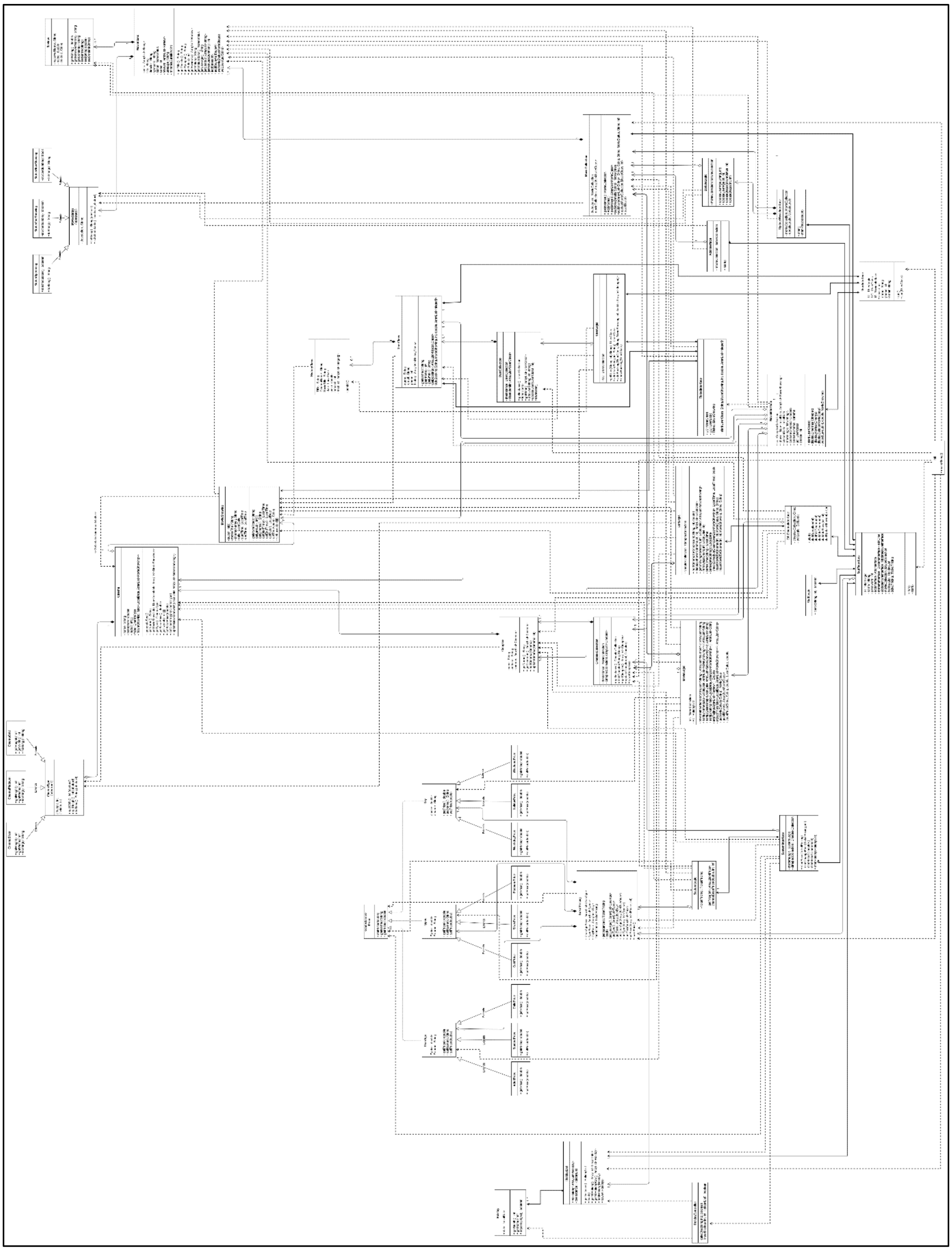
```
--Welcome to MOBLIMA!--
Select option:
1. Login User
2. Login Staff
2
Name:
Staff1
Pin:
1111
Choose action:
1. Add Movie
2. Update Movie
3. System Setting
4. Edit Cinema
0. Exit
3
Select what to change:
1. Change pricing
2. Modify holiday dates
0. Back
2
--Creating new holiday--
Enter month (1-12):
4
Enter date (1-31):
2
Added holiday succesfully
Choose action:
1. Add Movie
2. Update Movie
3. System Setting
4. Edit Cinema
0. Exit
```

3.3 Update Price Percentage

Staff also can update the price percentage for each factor.


```
Choose action:
1. Add Movie
2. Update Movie
3. System Setting
4. Edit Cinema
0. Exit
3
Select what to change:
1. Change pricing
2. Modify holiday dates
0. Back
1
Select which price category to modify:
1. User age
2. Cinema quality
3. Weekday/Weekend
4. Cinema base price
0. Back
1
Select which to modify:
1. Adult
2. Child
3. Student
3
Current price modifier: 0.95
Enter new price modifier (-1 to cancel):
1.1
Modified!
Select which to modify:
1. Adult
2. Child
3. Student
```

4. UML Class Diagram



For a clearer image, please refer to the file in the same directory.

5. UML Sequential Diagram

Please refer to the zip file in the same directory as this report.

6. Testing and Result

Please see the video demonstration for realtime testing and result.