



## ASSIGNMENT

# **CE/CZ2002: Object-Oriented Design & Programming**

*Building an OO Application*

**Lab:** SS4

**Group:** 4

**Members:** Mohamed Fazli Bin Abd Razak, Swa Ju Xiang, Low Yu Benedict, Madibek  
Nurbakyt

2019/2020 SEMESTER 1

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING NANYANG  
TECHNOLOGICAL UNIVERSITY

CZ2002 Object Oriented Programming & Design  
MOBLIMA Project Report

<b>1. Analysis of program</b>	<b>3</b>
Design Considerations	3
Design Principles	3
SOLID	4
YAGNI	5
Composition Over Inheritance	6
Suggested Improvements	6
Invite Friends	6
Sending Confirmation Email Upon Successful Booking	7
Use of OOP Concepts	7
Abstraction	7
Encapsulation	8
Inheritance	8
Polymorphism	8
Assumptions	9
<b>2. Supplementary Diagrams</b>	<b>10</b>
UML Sequence Diagram	10
UML Class Diagram	11
<b>3. Sample Tests</b>	<b>12</b>
Email checking at the end of purchase	12
Staff login verification	13
Date checking when adding a new showtime	14
<b>4. Declaration of Original Work for CE/CZ2002 Assignment</b>	<b>15</b>

**Appendix A****16**

# **1. Analysis of program**

## **Design Considerations**

As this project was given only 6 weeks to complete, we had to design our system to ensure it could be tested and modified easily at any time. In addition, our design had to support parallel development.

As such, we first began by separating the project into two major subsystems for parallel development - Staff and Moviegoer. After which, we decided on using an MVC (Model-View-Controller) architecture as this ensured ready testability, modifiability and also promoted parallel development.

To delve further into our choice of an MVC architecture, we chose an ECB (Entity-Control-Boundary) design pattern. The ECB design pattern allowed for the Boundary to handle user input, the Control to handle business logic, and the Entity to contain information. This resulted in a distinct separation of concerns between the application layers.

All design considerations can be seen reflected in the Class Diagram (Chapter 2).

## **Design Principles**

In this application, we agreed on seven design principles to ensure that our parallel development process is congruent and consistent across all development phases, and also to reduce any inevitable technical debt. The seven design principles (we are excluding 'D' of SOLID) are:

1. SOLI
2. YAGNI

### 3. Composition Over Inheritance

## **SOLID**

### **Single Responsibility Principle (SRP)**

In our application, we applied this principle Boundary and Entity classes. All boundary and entities have a single responsibility only.

For example, a boundary class “EmailVerification” only requests for a user to enter an email, and passes it to the respective control class. A successful email verification then redirect to another boundary class “BookingHistory”. In this example, “EmailVerification” only handles the email verification part of the program, and nothing more.

Another example would be the entity class “Staff”. As the name suggests, “Staff” is used for storing information on Staff users. As such, the “Staff” class only has attributes necessary for a Staff and nothing more (i.e. user, password and cineplexId).

### **Open-Closed Principle (OCP)**

In our application, this principle is implemented by creating a “View” class as the parent class of all other boundary classes such as MoviegoerMenu class, AdminMenu class. This principle is also shown in the composition of various control class’s methods applied in boundary classes and other control classes.

In this way, all its subclasses extends the behavior of the parent class without having to modify the source code of the parent class - in the case of boundary inheriting “View” class, the source code of “View” class is never altered.

In addition, protected and private methods are adopted in most entity and boundary classes to restrict access and modification.

### **Liskov Substitution Principle (LSP)**

This principle is closely related to OCP stated above. In our application, this principle is implemented in a way that in the View package (refer to Class Diagram, Chapter 2), View is an abstract class which allows its method to be override by its subclasses, such as “MoviegoerMenu” and “AdminMenu”.

For example, display() is an abstract method of “View” class which is overridden by its subclasses.

### **Interface Segregation Principle (ISP)**

In our application, interfaces are not applied, however some abstract classes are adopted in a decomposite manner to fulfill this principle.

While we do not use an interface for our program, each boundary class represents a specific part of the user interface. This was mentioned earlier in the OCP principle, whereby the “EmailVerification” class only handled the verification of user email user interface, and nothing more.

In addition, each of the boundary classes implements some of the abstract methods such as display() function which helps which helps when writing specific code to implement those view classes and methods.

### **YAGNI**

YAGNI stands for “You aren’t gonna need it” and is a design principle that promotes clean programming practices. YAGNI dictates that unused functionalities should not be implemented until deemed necessary - it discourages programmers to program in anticipation of additional functionality.

As a result, YAGNI allows us to spend time on functionalities that actually matter at that point in time, and to improve the overall simplicity of the code. This is especially helpful for our

collaboration as codes are more readable, and troubleshooting done during the integration of parallelly programmed classes simplified.

An example of the YAGNI principle in our codes would be the omission of unnecessary getter and setter methods. There are also no unused attributes or methods in any class.

## **Composition Over Inheritance**

Composition over inheritance is useful because it allows for a malleable program, over the rigid restrictions using inheritance would cause. Changes can be made easily, without affecting any child classes.

A clear example of this principle in our code would be the boundary classes using “Navigation” class’s methods through composition.

## **Suggested Improvements**

### **1. Invite Friends**

This suggested improvement is a functionality for when a movie-goer books more than one seat. At the check out menu, the movie-goer enters his email, and then a different email for each ticket if he so wishes. Tickets not allocated an email is by default tied under his email account. If a second movie-goer’s email was entered, this second movie-goer can log in to view the receipt of the purchase in his own booking history.

This improvement taps on several design principles. Firstly, as SRP was closely followed, we have existing entity class called “Customer”. This can easily be instantiated should a new email be entered. In addition, with Composition over inheritance, we have static methods in “BookingController” that can create a new “Customer” upon call. In addition with an ECB design pattern, logical changes made to the Boundary class “EnterParticulars” can be done without affecting any other classes.

As a result, this functionality will only require minor adjustments to “EnterParticulars” boundary class, and perhaps more method calling through composition, to the “BookingController”.

## **2. Sending Confirmation Email Upon Successful Booking**

This suggested improvement adds the functionality of a confirmation email sent to the movie-goer’s entered email upon successful booking. This improvement will require other java imports. But for this discussion, we will be discussing how the existing infrastructure - as a result of good design principles and practices - can facilitate this development.

Similar to the previous suggestion, this improvement greatly benefits as a result of SRP allowing for clearly defined entity classes - “Customer” entity. However, a new controller will be required to actually handle the mailing of email, in line with SRP practices. Using the entity class “Customer” we can fetch the email address of the customer before sending the email out. Moreover, the error checking of a valid email syntax can be handed over to the “BookingController” control class via composition akin to the OCP.

Should it be decided to create a new boundary class for this functionality, it would be wise to inherit from the abstract class “View”. This allows for this new boundary class to be able to behave consistently with the overall theme of the project (i.e. the printing format, the style of error messages).

## **Use of OOP Concepts**

### **Abstraction**

Abstraction means using simple things to represent complexity. In Java, abstraction works by creating useful, reusable tools which is implemented by creating various variables, methods and classes.

For example, we have a Showtime class to represent a showtime object. Inside the Showtime class, the information of the showtime is stored in its attributes, such as date of showtime, start time and 2D array of seat layout. By using abstraction, we can represent complex system by using multiple classes and also avoid repeating the same work multiple times.

## **Encapsulation**

Encapsulation is the practice of keeping certain fields within a class private and provide public method to access. It acts as a protective barrier to keep the data and code safe within the class itself.

In our application, encapsulation is well implemented in the classes. For example, for classes under Model, all attributes are set to be private to restrict access from other classes. To modify or access private fields from other classes, public methods are added to achieve this.

For example, in Cineplex class under Model package, there are getter and setter methods for controllers to access private fields of models.

## **Inheritance**

Inheritance allows programmers to create new classes that share some of existing attributes and methods to reuse existing classes. In our application, inheritance is widely used for boundary.

For example, the View class contains protected methods that can get input of different data types from user. All of the other view classes such as MoviegoerMenu is a subclass of View , hence the other view classes inherit the protected methods from View.

In addition to the methods and attributes inherited from its base class, the other view classes has its own fields and methods. This helps to avoid duplicated codes and improve code reusability.



## Polymorphism

Polymorphism means the ability to take more than one form. An operation may exhibit different behaviors in different instances. Polymorphism is extensively used in implementing Inheritance and overriding is a necessary tool for polymorphism.

In the view package, all of the other view classes such as BaseMenu, MoviegoerMenu, AdminMenu and some other views overrides the display() method which is implemented in View (abstract class) as an abstract method.

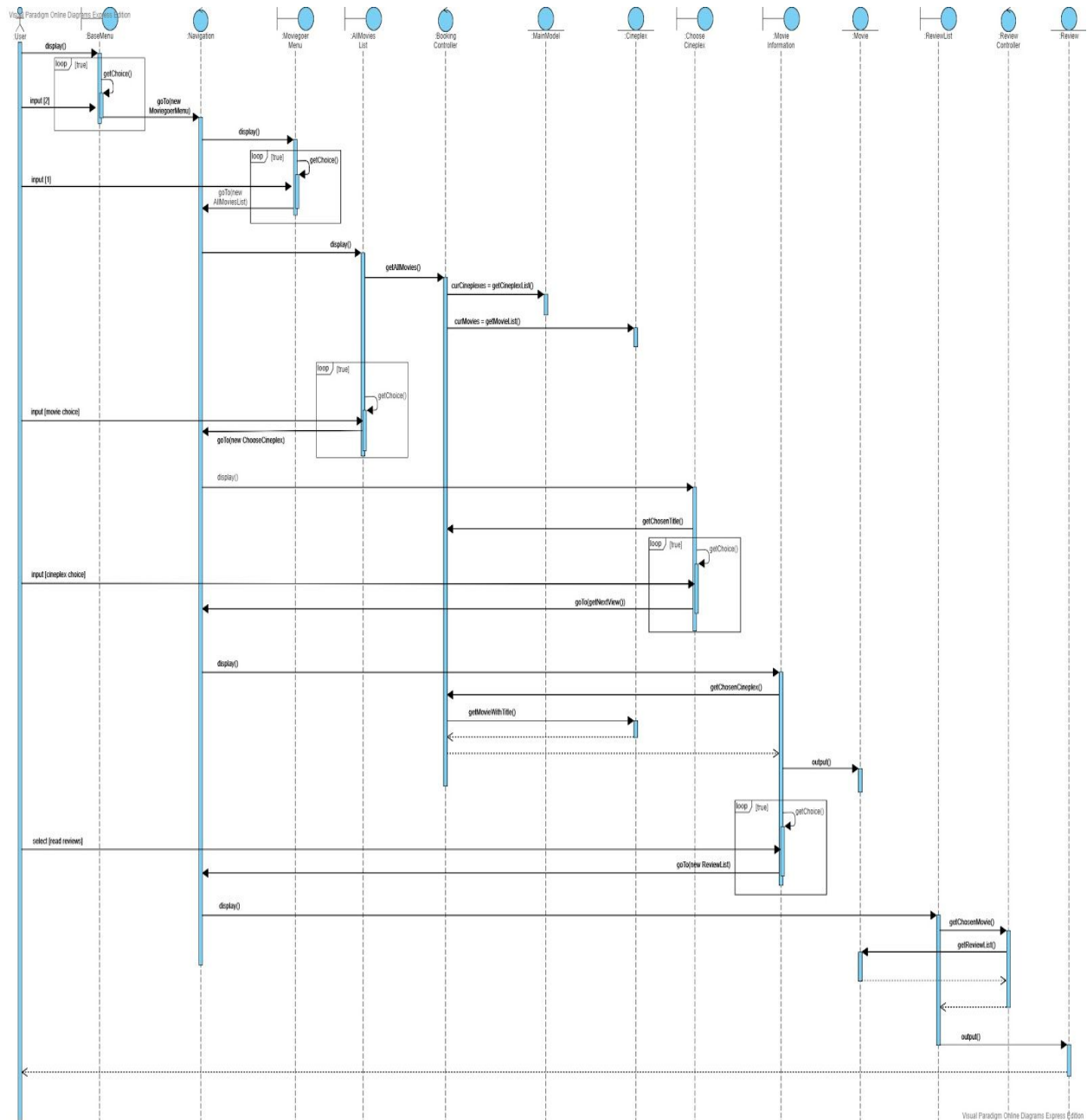
## Assumptions

1. This is a single-user application. As such, data will be read from the local database at the start of the program only and saved to the local database at the end of the program.
2. THREE cineplexes are created and under these cineplexes, THREE cinemas are created.
3. The currency used will be in Singapore Dollar (SGD) and inclusive of Good and Services Tax (GST).
4. The staff is assigned to a single cineplex. A staff belonging to a cineplex cannot access data from another cineplex. Login and password for each cineplex admin page are set beforehand since there is no registration.
5. Payment will always be successful.
6. Validation of the Customer's age will not be done through the application.
7. A maximum of 5 seats can be booked in a single transaction.
8. Different classes of Cinema (IMAX, Gold, Regular) have the same seat layout.
9. Only the base ticket price specific to a cineplex can be changed by the cineplex's staff.
10. For movies, only showing statuses *Now Showing* and *Preview* can be booked.
11. Customers can only book for showtimes that are shown within a week.
12. Customer doesn't need to verify or register their email addresses to make a payment. To check the booking history, only the email address that was used to book is enough

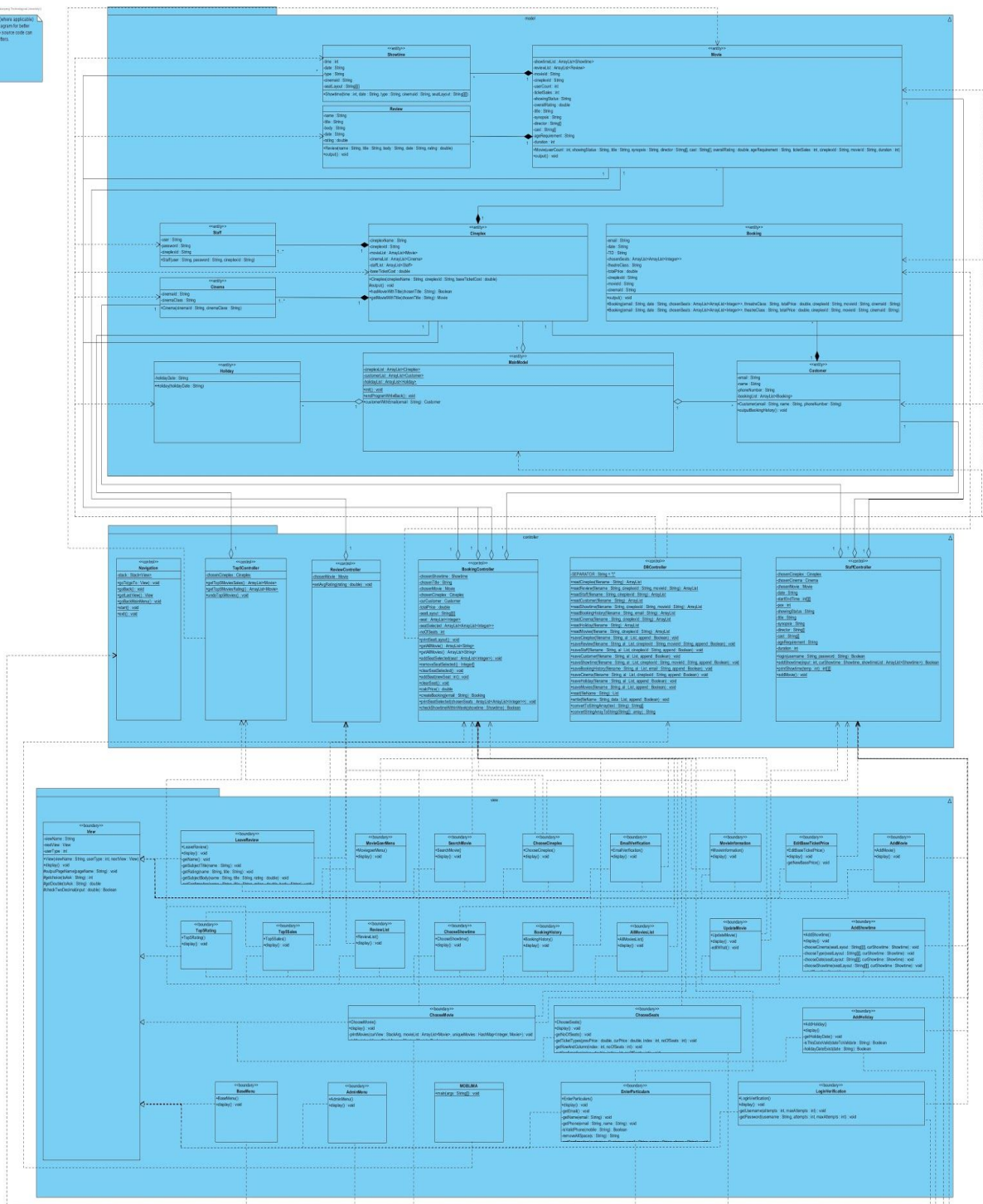
13. The ticket pricing table can be referred to in **Appendix A** (options with the same color are mutually exclusive). The maximum and minimum possible costs are \$25 and \$8.
14. Information about movies, such as title, synopsis, cast, etc, are stored universally in the system. An update in Jurong Point's movie title 'Avengers' to 'Avengers: Endgame' will be reflected in all other cineplexes as well.

## 2. Supplementary Diagrams

### UML Sequence Diagram



Getters and Setters method (where applicable) or not shown on this class diagram for better readability. For reference, the source code can be viewed for Getters and Setters.



### 3. Sample Tests

#### Email checking at the end of purchase

After the customer purchases ticket(s), they need to input email address to store the booking for reference later. We check for the validity of the email address in the method *getEmail()* in *EnterParticulars.java*.

```
=====
-----Booking: Enter Particulars-----
=====

Chosen movie: 'Lion King'
Total booking price: $16.0

Please input your email address (Input 0 to go back):
```

**Accepted output:** The system continues with asking to enter the name and phone.

**Rejected output:** “Not a valid email address.”

Input	Expected output	Actual output
lowbenedict12@gmail.com	Accept	Accept
johnsonjack@gmail.com	Accept	Accept
jack@12r	Reject	Reject
abc@gmail.srt	Accept	Accept
ben@yahoo.sg	Accept	Accept
23423	Reject	Reject
ab2@123.123	Reject	Reject
9	Reject	Reject
aaaaaaaa	Reject	Reject

## Staff login verification

Before giving an access to admin functionalities, we ask to verify using appropriate username and password set for each cineplex in methods `getUsername()` and `getPassword()` in `LoginVerification.java`.

```
=====
-----Choose a Cineplex Location-----
=====

(0) Back
(1) 'Jurong Point Cinema'
(2) 'Paya Lebar Cinema'
(3) 'Vivo City Cinema'
Please select a cineplex: 1

=====
-----Login Verification-----
=====

(0) Back

Please input your username:
```

**Accepted output:** “Login Successful. Welcome *[username]!*”

**Rejected output:** “Invalid username/password. Number of attempts left: *[X]*”

Chosen cineplex	Username	Password	Expected output	Actual output
1	staffA	password1	Accept	Accept
2	staffB	password2	Accept	Accept
3	staffC	password3	Accept	Accept
1	a	password	Reject	Reject
2	staffA	password1	Reject	Reject

## Date checking when adding a new showtime

When the admin tries to add a new showtime for a specific movie, we first check for the correctness of the date entered in *chooseDate()* in *AddShowtime.java*.

```
=====
-----Add Showtime-----
=====

Chosen movie: 'Abominable'

(0) Back
(1) Cinema ID: 001, Cinema Class: Regular
(2) Cinema ID: 002, Cinema Class: Gold
(3) Cinema ID: 003, Cinema Class: Platinum
Please select a cinema: 2

(0) Back
(1) Digital
(2) 3D
(3) IMAX
Choose type of movie: 1

(0) Back
Enter a date (dd/mm/yyyy):
|
```

**Accepted output:** The system continues with asking to enter the start time of the movie.

**Rejected output:** “Please enter a valid input”





Date	Expected output	Actual output
30/11/2019	Accept	Accept
01/01/2020	Accept	Accept
18/11/2019	Accept	Accept
1/01/2020	Reject	Reject
12	Reject	Reject
ab	Reject	Reject
32/11/2019	Reject	Reject
17/11/2019	Reject	Reject

## 4. 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
Low Yu Benedict	CZ2002	SS4	 17/11/2019
Madibek Nurbakyt	CZ2002	SS4	 17/11/2019
Mohamed Fazli Bin Abd Razak	CZ2002	SS4	 17/11/2019
Swa Ju Xiang	CZ2002	SS4	 17/11/2019



## Appendix A

Note: Options in the same color are considered mutually exclusive.

Base Ticket Price	\$8
<b>Type of Movie</b>	
IMAX	+\$5
3D	+\$3
Digital	+\$0
<b>Type of Cinema</b>	
Platinum Class	+\$5
Gold Class	+\$3
Regular	+\$0
<b>Type of Customer</b>	
Adult	+\$2
Student	+\$1
Senior Citizen	+\$0
<b>Type of Day</b>	
Weekend/Public Holiday/Eve	+\$3
Weekday	+\$0
<b>Time of Day</b>	
After 6p.m.	+\$2
Before 6p.m.	+\$0