**APPENDIX B:**

**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

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Course  (CE2002 or CZ2002) | Lab Group | Signature /Date |
| Nguyen Ngoc Tram Anh | CE2002 | SSP3 |  |
| Chaitanya Malaviya | CE2002 | SSP3 |  |
| Li Bai Yang | CE2002 | SSP3 |  |
| Yuen Kai Him | CE2002 | SSP3 |  |

Important notes:

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



**CE/CZ2002 Object-Oriented Design & Programming Assignment Report**

Building an OO Application

Prepared By:

Nguyen Ngoc Tram Anh

Chaitanya Malaviya

Li Bai Yang

Yuen Kai Him

(SSP3 Group 6)

Contents

[**1. Introduction 2**](#_Toc371931913)

[**2. UML Class Diagram 2**](#_Toc371931914)

[**3. Design Considerations 2**](#_Toc371931915)

[**4. UML Sequence Diagram 2**](#_Toc371931916)

[**5. Test Cases 2**](#_Toc371931917)

# Introduction

* Application

MOvie Booking and LIsting Management Application (MOBLIMA) is an application allowing the online booking and purchase of movie tickets, listing of movies and sale reporting. This application will be used by the movie-goers as well as cinema staff.

* Function of application

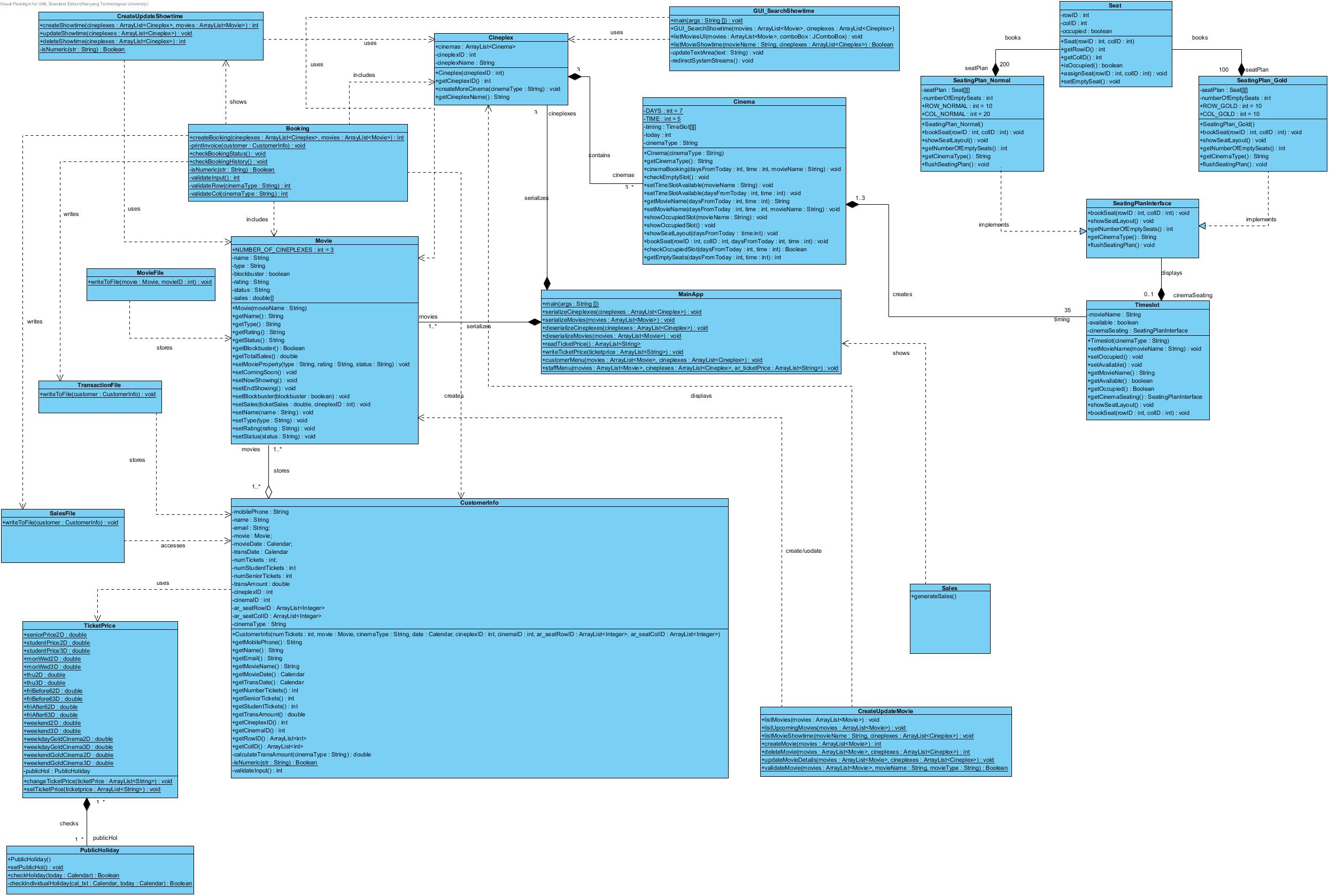
1. User application
2. View Coming Soon movies
3. Search movie for showtime & Create booking
4. Check booking status
5. Check booking history
6. View GUI for search/list movie showtime

b. Staff application

1. View current movie listing
2. Create movie listing
3. Update movie listing
4. Remove movie listing
5. Create cinema showtime
6. Update cinema showtime
7. Remove cinema showtime
8. Print sales revenue report
9. Update ticket price
10. .Set upcoming holiday

# Design Consideration

## Class Diagram



# Explanation for Class Diagram

* The *Seat* class contains information (e.g row number, column number, if occupied) of a single seat;
* A 2-D array of *Seat* objects will be created in each of the SeatPlan classes (both *SeatPlan\_Gold* and *SeatPlan\_Normal*);
* A *TimeSlot* class is used to store the show time for a particular movie and the type of cinema(Gold or Normal) it will be showed in;
* A *Cinema* class instantiates a 2-D array of TimeSlot objects with the Day and Time attribute;
* A *Cineplex* class will contain an Array List of Cinema objects.
* The *Movie* class will store the name and type of a movie. The details of movies (create/update/delete) can be made through the *CreatUpdateMovie* class.
* Customer information, including name, mobile number, email, date of purchased movie ticket can be accessed through *CustomerInfo* class. A CustomerInfo object will be created every time a customer does a booking. After the booking is done, the object will not be stored as object but will be written to the text file Transaction.txt
* The *Booking* class is the main control class for booking which interacts with various classes to allow the user to book tickets, check their booking history and print the invoice.
* *CreateUpdateShowtime* and *CreateUpdateMovie* are 2 other control class that do all the work regarding create, update and delete of showtimes and movies
* All past transaction history will access *CustomerInfo* class, extract date of transaction, movie name, cineplex ID and transaction amount (saved as Sales) as one entry in the *“Sales.txt”* file. The Sales.txt file is to serve as a database for making the sales revenue report.
* *TicketPrice* class and *PublicHoliday* class will interact with CustomerInfo class, where ticket price will be generated according to day and time in the week (weekdays/weekends, before/after 6pm), customer’s age (senior/student vs normal), cinema type (Normal or Gold), movie type (digital or 3D.
* Printing of sales revenue report (by day/month) will be done by calling the *Sales* class in which searching of all entries from the Sales.txt file for on “a particular date”, “a particular movie” in “a particular cineplex” for “a particular sale amount” will be done and added together to get the total amount of “that movie”, in “that cineplex”, on “that day”. With this information stored in a 3-D array of parSale[i][j][k] ( i = number of dates/month, j = number of movies, k= number of cineplexes), the sales revenue could easily be displayed to the staff;
* The *main* class will initiate and execute the program according to the user input. The data will be read from, appended or overwritten depending on the customer/staff selection through interacting with the above mentioned classes.

## 2.2 Applied OO concepts

### Encapsulation

Encapsulation is the technique of making the fields in a class private and providing access to the fields through public methods. If a field is declared private, it cannot be accessed outside the class, thereby hiding the information of that field within the class. For this reason, encapsulation is also referred to as information hiding. Access to the data and code is tightly controlled by an interface.

Example from this application:

Layouts of seats in the cinema (Gold/Normal class) will be held as private attributes (2-D array), and will only be displayed through public methods inside the class because direct user access to the seat layout is not allowed in the application



The class TimeSlot and SeatingPlanInterface are hided from the public and from the code. The deepest level we access from the application is Cinema class. For example

In cinema class,

|  |
| --- |
| public void showSeatLayout(int daysFromToday, int time) {  int day = (today + daysFromToday) % 7;  timing[day][time].showSeatLayout();  } |

In TimeSlot class

|  |
| --- |
| public void showSeatLayout(){  cinemaSeating.showSeatLayout();  } |

In SeatingPlan\_Normal / SeatingPlan\_Gold

|  |
| --- |
| @Override  public void showSeatLayout(){  System.out.print(" ");  for(int j=0; j<COL\_GOLD; j++){  if(j==COL\_GOLD/2)  System.out.print(" ");  System.out.print(" "+(j + 1) + " ");}  System.out.println("");    for (int i = 0; i < ROW\_GOLD; i++) {  for (int j = 0; j < COL\_GOLD; j++) {  if(j==0){  System.out.printf("%2d ", (i+1));  }  if (j == COL\_GOLD/2) {  System.out.print(" ");  }  if(seatPlan[i][j].isOccupied() == true)  System.out.print("[X] ");  else  System.out.print("[\_] ");  }  System.out.println("");  }  } |

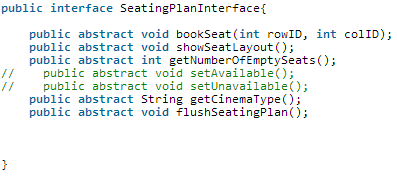
### b. Abstraction & Polymorphism

Abstraction in Java is achieved by using interfaces and abstract class. Interfaces can help us in creating polymorphism of our code, which will create the appropriate object and its behavior based on different conditions.

For our application, a *SeatingPlanInterface* is created which contains attributes and methods that would be implemented in different ways in *SeatingPlan\_Gold* class and S*eatingPlan\_Normal* class.

Polymorphism has been implemented in this application using method overriding through the *SeatingPlan* interface. The methods of concrete classes *SeatingPlan\_Normal* and *SeatingPlan\_Gold* implement the methods declared in the *SeatingPlan* interface.

The example below shows a different return String, depending on the type of cinema.



2.3 Other design approach

* For database, our application implements Serializable to serialize 2 classes (ArrayList of Cineplex class and ArrayList of Movie class)
* For the rest of the database, we use text file to store the data. Those include Sales.txt, Transaction.txt, PublicHolidays.txt and TicketPrice.txt. The use of text files for Public Holidays and Ticket price, especially, helps to update the above-mentioned even without booting up the application.
* Error checking is done thoroughly throughout the application. This includes non-numeric input for a situation that asks for an integer (e.g. inputing seat selection, option selection,…), inputing input that is out of required range (e.g. inputting number of student tickets that is larger than total number of seats booked earlier, inputting option that is out of range), booking of a fully booked ticket, creating/updating a cinema show time at a already booked slot, inputting day and time that is not showing any movie.
* Hard-coding is avoided as much as possible. Constants are used, such as DAYS and TIME for cinema class. The number of seats in a cinema seating plan is also put in constant declaration (SeatingPlan\_Normal and SeatingPlan\_Gold) and are used throughout the code (all the for loop in the class).
* Number of cinemas is not fixed at 3. For all the “for” loop, we always use “cineplexes.get(i).cinemas.size()” for the size of the cinema array. Hence, in the future different cineplexes with different number of cinemas (more than 3) will not give any error to our application.
* Method overloading is used in Cinema class for setTimeslotAvailable() method. Depending on the purpose, we can call setTimeSlotAvailable(String movieName) or setTimeSlotAvailable(int daysFromToday, int time)

2.3 Assumptions

1. This is a single-user application and doesn’t allow concurrent access;
2. There are three cineplexes
3. When printing sales revenue report, user-selection of period (by day or by month) will be done after the user-selection of “by movie” or “by cineplex”;
4. Mobile number will be used as the unique customer ID when checking one’s past booking history.

# Design Principles

- Single Responsibility Principle (SRP): All the classes have one specific responsibility, and the responsibility is entirely encapsulated by the class.

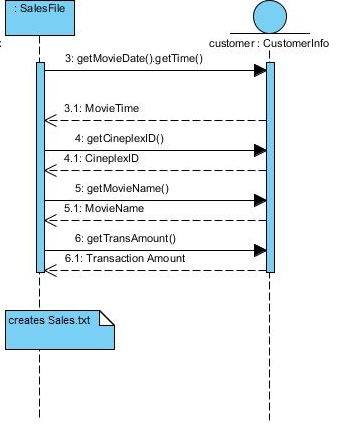
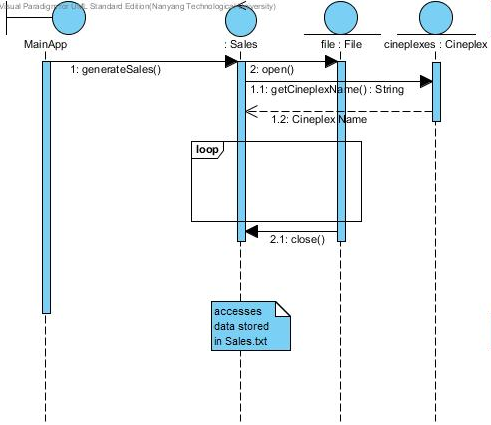
- Open-Closed Principle(OCP): Abstraction has been used in the program to show an entity can allow its behavior to be modified without altering its [source code](http://en.wikipedia.org/wiki/Source_code)

- Interface Segregation Principle(ISP): There is one interface in the program called *SeatingPlanInterface* which is implemented by SeatingPlan\_Gold and SeatingPlan\_Normal according to the class of the cinema. The program avoids the use of a general purpose interface.

- Don’t Repeat Yourself(DRY): Dependency relationships where some classes use methods of other classes, have been used wherever possible to avoid repetition of code

Besides, *delegation* has been used in several parts of the program while implementing “has a” relationship between classes.

**4. UML Sequence Diagram**



Flow of the sequence diagram:

* The method generateSales() in class Sales is called in the MainApp.
* The Cineplex class passes the Cineplex name to the Sales class
* The Sales class accesses a text file called Sales.txt to print the monthly sales revenue report.
* In sd2, the class SalesFile has created the text file Sales.txt upon getting data from the CustomerInfo class.
* The Sales class reads the Sales.txt file and retrieves data for calculating the sales report by month from January till the current month.

# 5. Test Cases

Error checking

* Non-numeric input for a situation that asks for an integer (e.g. inputting seat selection, option selection,…)
* inputting input that is out of required range (e.g. inputting number of student tickets that is larger than total number of seats booked earlier, inputting option that is out of range),
* booking of a fully booked ticket,
* creating/updating a cinema show time at a already booked slot,
* inputting day and time that is not showing any movie.
* booking on a different day of the week or holiday and type of cinema (eg suite)
* • Sale revenue report by different selection, especially period.

6. Future upgrade

The application supports adding of cinema to a Cineplex. There is a createMoreCinema(String cinemaType) method in Cineplex class. Hence, when need be, we can always call the method to add more cinema. Furthermore, our code uses the dynamically created number of cinemas instead of “3”.