****

**CZ2002 Object Oriented Design & Programming**

**Report**

**BCG2**

**Chua Jie Yong (U1422170H)**

**Huang Kai (U1421072H)**

**Jordan Kwong Zheng Xi (U1422414C)**

**Yu Jia Qing (U1421648H)**

SCHOOL OF COMPUTER ENGINEERING

NANYANG TECHNOLOGICAL UNIVERSITY

**Contents**

[**1. Declaration of Original Work** 3](#_Toc435165062)

[**2. UML Class Diagram** 4](#_Toc435165063)

[**3. Considerations and Use of Concepts** 5](#_Toc435165064)

[3.1 Design Considerations 5](#_Toc435165065)

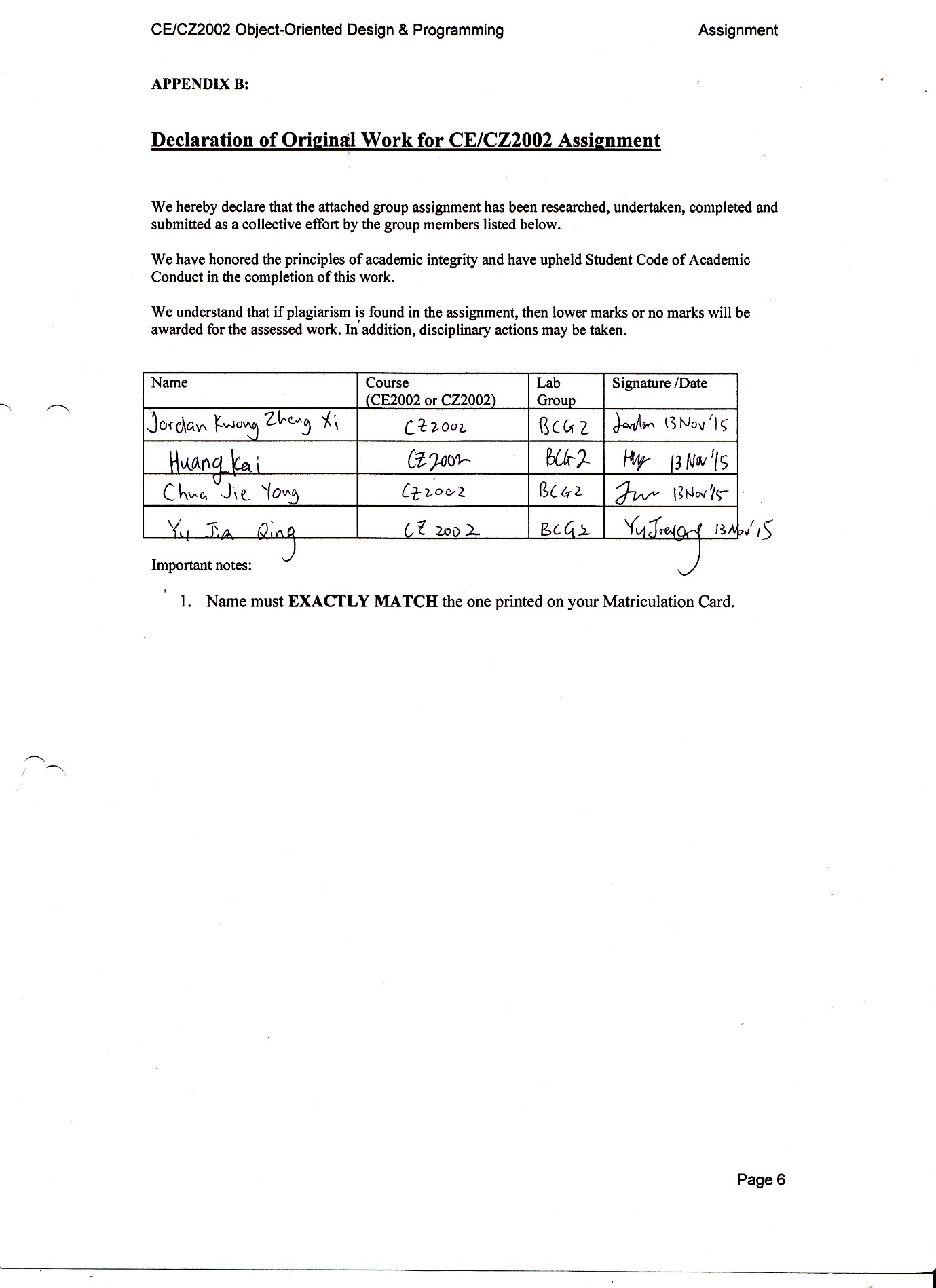
[3.2 Use of Object Oriented Concepts 6](#_Toc435165066)

[**4. UML Sequence Diagram** 8](#_Toc435165067)

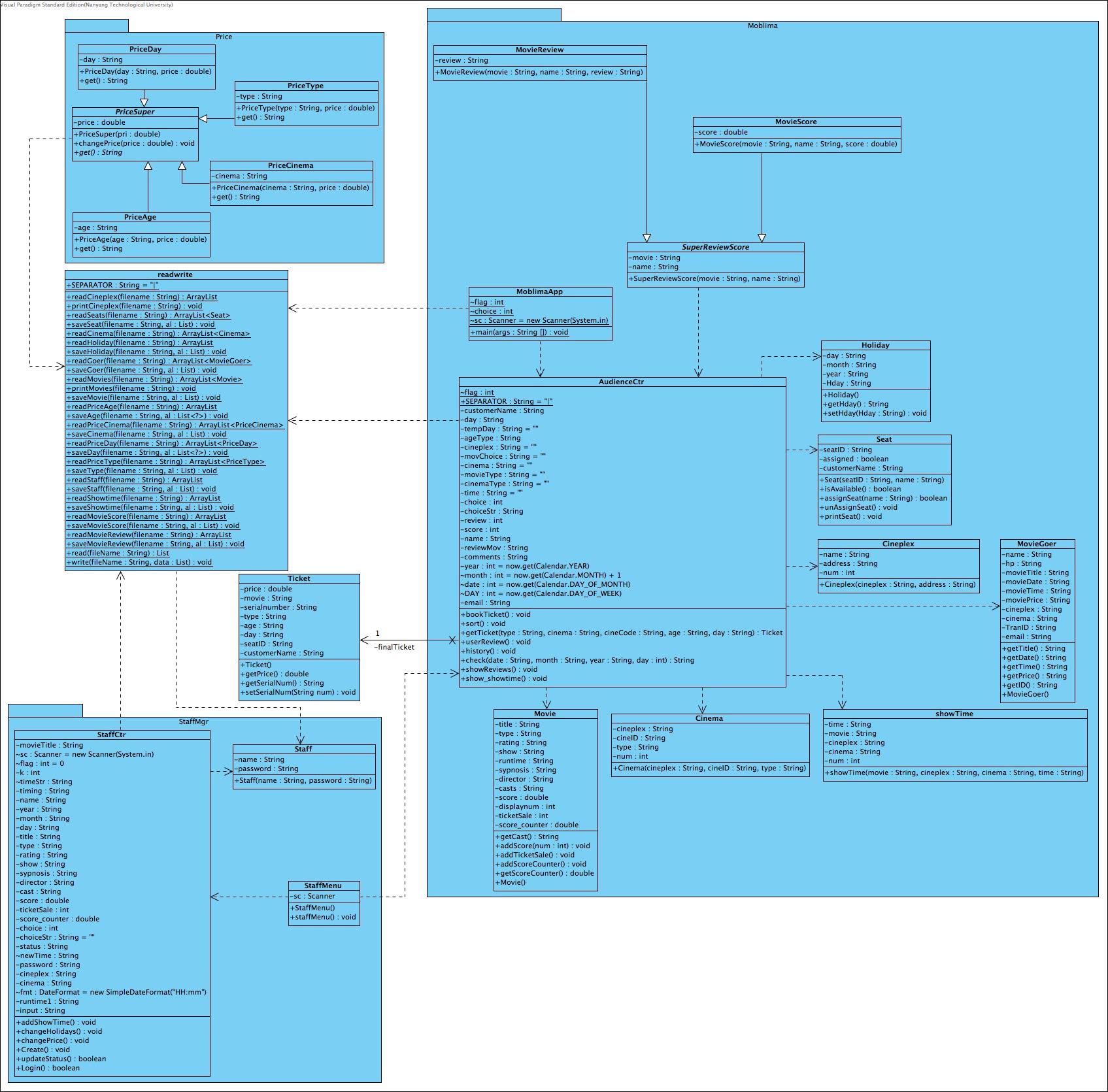
[**5. Test Cases** 9](#_Toc435165068)

[**6. Reference** 14](#_Toc435165069)

# 1. Declaration of Original Work



# **2. UML Class Diagram**



# 3. Considerations and Use of Concepts

For our object oriented application, we made use of several design considerations, namely: Don’t Repeat Yourself, Single Responsibility Principle, Open-Closed Principle, Liskov Substitution Principle and Interface Segregation Principle. Moreover, we coupled the design considerations with the use of object oriented concepts learned in the lecture. This ensures that we create a quality object oriented application.

## 3.1 Design Considerations

**Don’t Repeat Yourself Principle** is the principle where the code in our application should be written only once and be clearly written. Our code follows this principle in many areas, allowing us to have a smaller chance of failure as there is less code. It also avoid any duplication in the code, allowing the code to be “cleaner”. This can be shown by putting the code in various methods so that we do not have to repeat the code again if we use it twice.

For instance, we have a PriceSuper superclass omits the need to duplicate the code to the subclasses as they use the same get methods. This is shown in Figure 1 below:



*Figure 1: PriceSuper abstract class*

**Open Closed Principle** is the principle where classes should be opened for extension, but closed for modification. This means that an entity can allow its behavior to be extended without modifying its source code. An example of this can also be shown in Figure 1 above, which is to create a base class. This will close the base class for modification, but allowing extension to more base classes.

**Liskov Substitution Principle** states that if a program module is using a base class, then the reference to the base class can be replaced with a derived class without affecting the functionality of the program module. This can also be shown in Figure 1 above, where PriceSuper is an abstract class that allows overriding by other classes. This will not affect PriceSuper’s behaviour.

## 3.2 Use of Object Oriented Concepts

An **abstraction** denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer. In our application, objects are differentiated into different classes. For example, our PriceSuper class has an abstract method get() which allows overriding by other controller classes such as the PriceAge.get(), PriceDay.get(), PriceType.get(), PriceCinema.get(). Different classes calling the get() method will exhibit different behaviours.

**Encapsulation** builds a barrier to protect an object’s private data, access to private data can be done through the public methods of the object’s class get and set methods. **Information hiding** hides the detail or implementation of the class from users. This also means that users do not need to know the implementation details for the methods. This can be shown throughout the application.

*Figure 2: Seat Class*

As shown in Figure 2, the attributes are set to private, which hides the variables of the class from other classes. Accessing the private variables can only be done through the use of public methods, such as public String getSeatID(). Other classes can access or modify its attributes through its get and set methods. This will give us control over how the accessing of each data is done in each class.

**Inheritance** is a mechanism that defines a new class which inherits the properties and behaviours of a parent class. This is shown in MovieScore and MovieReview (subclasses) extends SuperReviewScore (superclass) to call the get and set method that is implemented by the superclass SuperReviewScore. This omits extensive duplication of code since the subclasses are able to reuse the superclass’s code. Figure 3 below shows how the implementation is done:

*Figure 3: Implementation of Inheritance*

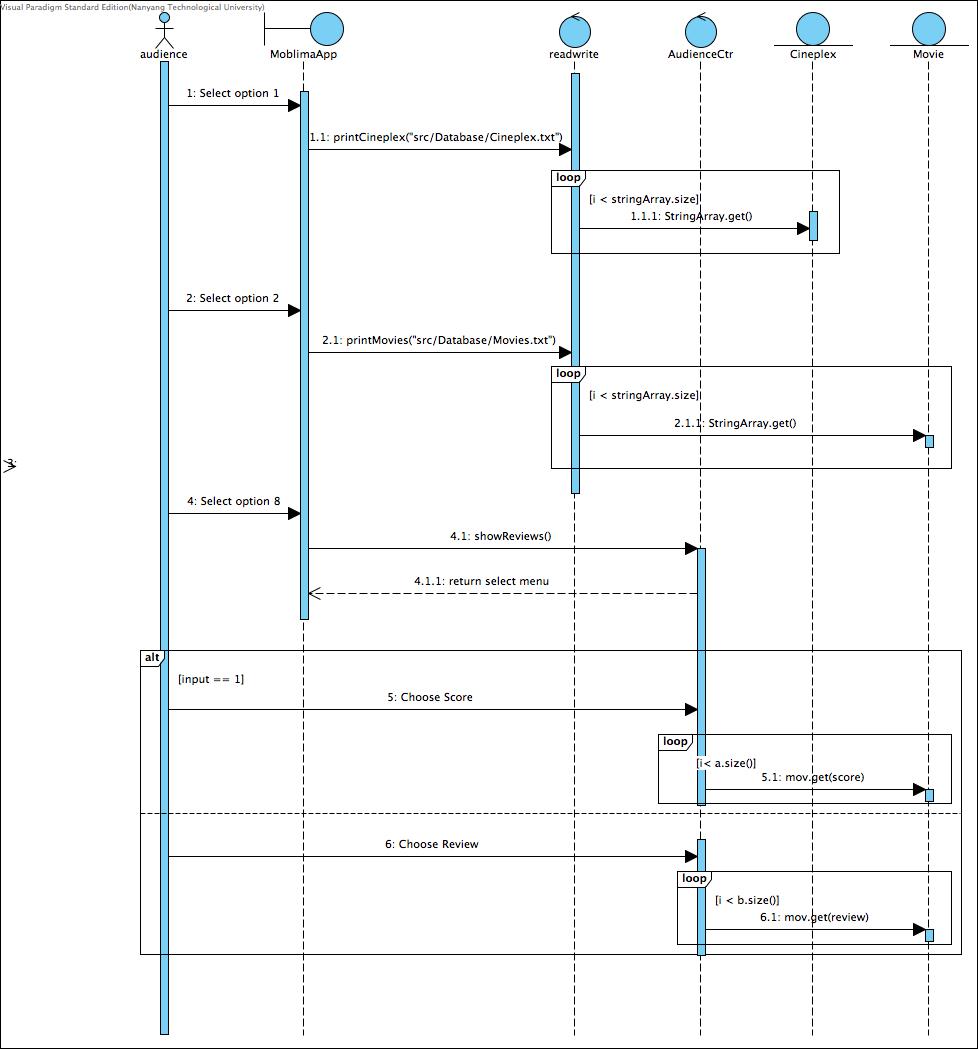
**Polymorphism** is the ability to take more than one form. An operation can show different behaviours in different instances used in implementing inheritance, and overriding is a necessary tool for polymorphism. For example in Figure 1 above, PriceAge, PriceCinema, PriceDay, PriceType (subclasses) overrides the abstract get() method in PriceSuper (abstract class).

*Figure 4: Abstract method in PriceSuper*

*Figure 5: Abstract method in PriceDay*

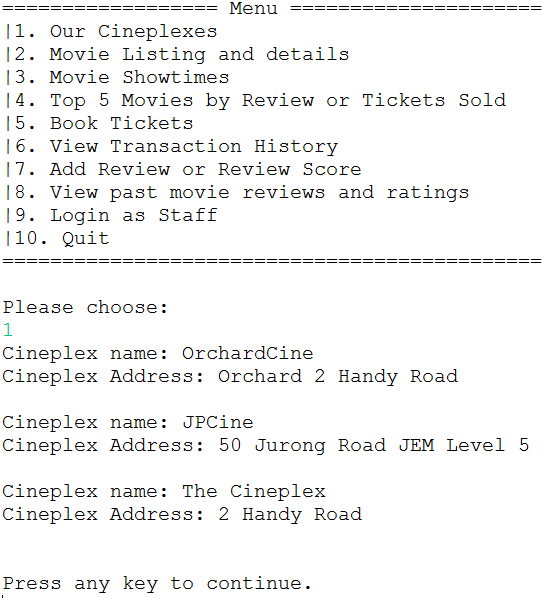
As seen in Figure 4 and Figure 5 above, the get() method in PriceDay overrides the abstract method in PriceSuper.

# 4. UML Sequence Diagram



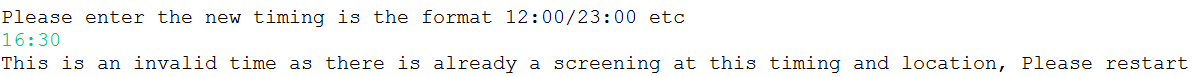
# 5. Test Cases

1. Changing same movies and location to within screening time:



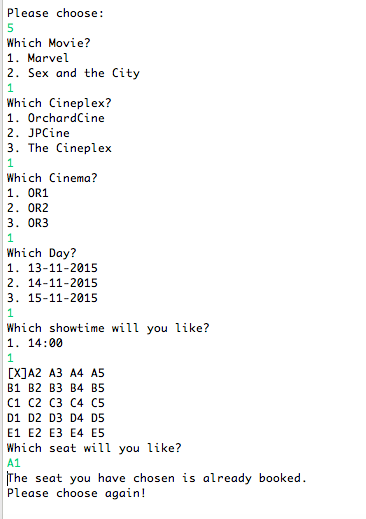
*Figure 6: Cineplex’s Addresses*





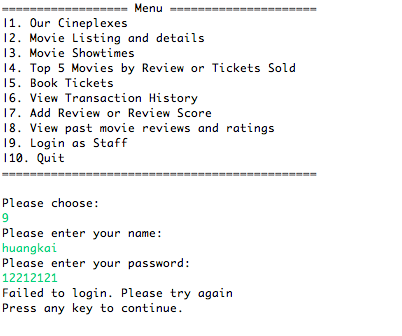
*Figure 7: Changing same movies and same location to within screening time (error)*

* If the staff changes the same movies and same location to within screening time. The system will show an error message and ask staff to restart.

1. Input seat ID which has been booked:

*Figure 8: Enter the SeatID which has been booked*

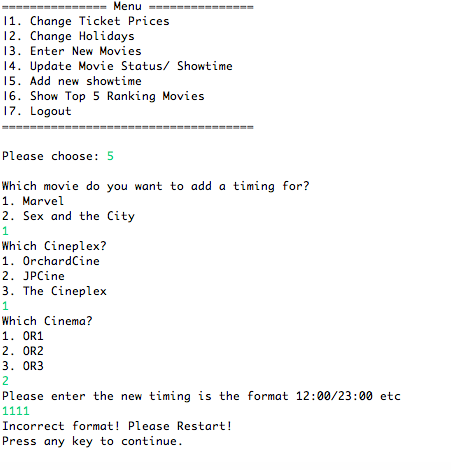
* If user enter the SeatID which has been booked by other user. The system will show that ‘The seat you have chosen is already booked.’ And it will let user to choose a new seat.

1. **Incorrect input of staff login password:

*Figure 9: Incorrect password*

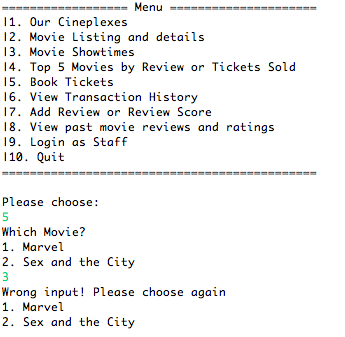
* If the staff enters the wrong password. The system will show an error message and return back to the main menu.

1. Invalid timing format:



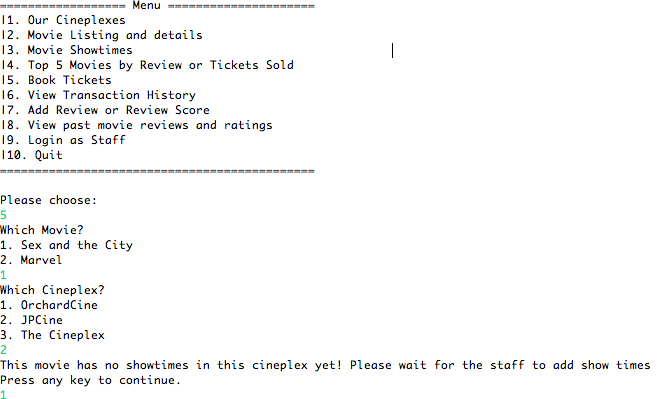
*Figure 10: Invalid format*

* If the staff enters the invalid timing format. The system will show an error message and return back to the staff menu.

1. Invalid input for “Choose a movie”:

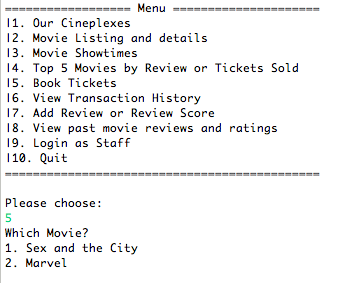
*Figure 11: Invalid movie input*

* If the user choose a movie which is not in the list. The system will show an error message and ask the user to choose again.

1. Select a Cineplex which the chosen movie has no show time:

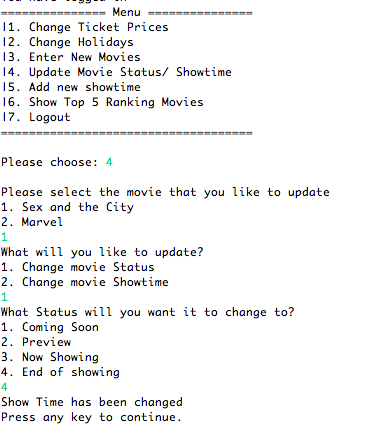
*Figure 12: movie has no show times in the Cineplex*

1. Configuring “End of Showing” date and the movie should not be listed for booking



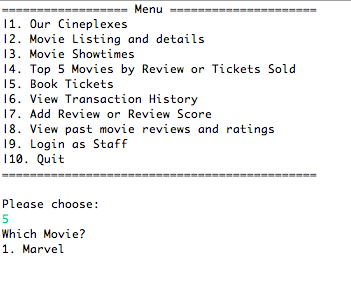
*Figure 13: Current movies available to watch*

* Initially we have two movies currently showing.



*Figure 14: Changing movie status*

* Then we change the status of “Sex and the City” to “End of showing”:

**

*Figure 15: Currently movie available to watch*

* The movie for user to choose is left with only one. The movie “Sex and the City” will not be listed for booking because it has ended for showing.

# 6. Reference

* NTU CZ2002 Lecture Notes
* Wikipedia