# Declaration of Original Work for SC2002 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** | **Matriculation Number** | **Course** | **Lab Group** | **Signature/Date** |
| Khoo Jia Xuan Mavis | U2121376B | SC2002 | SSP3 | 9/11/2022 |
| Joel Tham Yew Hng | U2121679K | SC2002 | SSP3 | 9/11/2022 |
| Lian Hong Shen Jordan | U2122011E | SC2002 | SSP3 | 9/11/2022 |
| Chew Zhi Yi Mark | U2121947C | SC2002 | SSP3 | 9/11/2022 |
| Lim Wei Jun Wilson | U2121267J | SC2002 | SSP3 | 9/11/2022 |

* 1. **Design Considerations**
  2. **Assumptions**

1. This is a single-user application and there is no need to consider concurrent access.
2. THREE cineplexes will be created for the demonstration.
3. The currency will be in Singapore Dollar (SGD) and inclusive of Goods and Services Tax (GST).
4. A simple login for cinema staff will suffice.
5. Payment will always be successful.
6. There is no need to interface with external systems, e.g. Payment, printer, etc., but you can consider it in your design.
7. Senior citizen tickets can be purchased online without validation of identity or age. The validation will be done upon entering the cinema.
8. The ticket pricing structure is assumed to follow this hierarchy:
   1. Seat Type
   2. Weekend/holiday
   3. Senior Citizen
   4. Student
   5. Normal (basic)
      1. Cinema Type

# Abstraction

Abstraction is the technique of highlighting key aspects of an entity while concealing minor ones. The complexity of the design and implementation processes is lowered by abstraction. It is clear from our code that it uses our attributes, model, methods, UI, and models, encouraging code reuse while maintaining a straightforward and tidy design.

# Encapsulation & Information Hiding

Private information on an object is shielded by encapsulation and information hiding against unauthorised access by other classes. This is evident in our code where attributes are labelled as private, so that other classes and objects will not be able to access them and only the correct classes will be able to get the relevant information. This is also applicable to importing the correct modules so that certain objects do not have access to information privy to other objects.

# Inheritance & Polymorphism

Polymorphism and inheritance go hand in hand because polymorphism enables an object to take on numerous forms and when various classes are related to one another via inheritance. This can be seen in our Person, Admin and Guest classes where it adopts polymorphism and inheritance which allows our Admin and Guest class to inherit all attributes in the Person class, allowing us to construct and build on our current work without having the need to create new attributes in the class from scratch.

# Single Responsibility Principle (SRP)

Each one of our classes only has a single responsibility so as to achieve high cohesion such that there is only one reason for it to be changed. For instance, our CreateMovie, RemoveMovie and UpdateMovie only do their respective jobs and nothing else, such as creating movies, removing movies and updating movies respectively. This is so our classes are easier to maintain and have high reusability.

# Open-Closed Principle (OCP)

Our modules are open for extension and allow modification of functionality without having to change source code since the modules are also closed for modification. We have implemented a superclass for Events management which manages the creation, deletion, parsing and listing of events in the format of Name and Date. The HolidayUI inherits this class and overloads the methods to enable holidays to be added and removed without affecting the base class Events. Additional functionality can be extended from the superclass and the subclasses overload the superclass’ methods to implement their own variation and functionality, without changing existing code. For instance, other special types of events such as promotional dates can be easily implemented since the module is extensible.

# Liskov Substitution Principle (LSP)

All our derived subclasses can be completely substitutable for their superclasses since their pre-conditions are no stronger than the superclass methods and their post-conditions are no weaker than the superclass methods. This enables our subclasses to replace superclasses without breaking the implementation of our code. For example, our Person class is a superclass that both our subclasses Admin class and Guest class extend from. Since they enhance the functionality without reducing it, functions that use these superclasses can use the subclasses without knowing it and it would not stop the application from running.

# Interface Segregation Principle (ISP)

Our classes will not depend on interfaces that they do not need since we have many explicit interfaces that serve a single purpose rather than one general-purpose interface that encompasses all. For instance, we have both a SearchInterface and DisplayInterface interface so as to remove the possibility of classes having to implement methods from interfaces that they do not actually use. This is why our displayMovie class only implements the DisplayInterface since it prints out movie details but our SearchMovie class implements both the SearchInterface and DisplayInterface since its functionality requires a search for the required movie through user input before displaying its details. This design reduces the negative effects of utilising larger interfaces by breaking it down into smaller and more specific interfaces, increasing its flexibility.

# Dependency Injection Principle (DIP)

Our classes depend on abstraction rather than concretion through the use of interfaces that act as a framework to facilitate communication with the concrete classes. This ensures that our high level modules do not depend on low level modules which helps to reduce internal dependencies by enabling these dependencies to be injected externally, increasing the reusability of these high level modules. For instance, rather than hard-coding our displayMovie class, we have a DisplayInterface interface since it gives us the flexibility to add new models to display later on such as movie Sessions that can also provide their own functionality. This allows our main client classes to call these modules and functions without having to worry about how their dependencies are implemented due to the reduced coupling between classes and their dependencies.

# Further Enhancements

* 1. **Membership Program**

A membership program allows guests to enjoy discounted rates or other perks. A guest can be a member, and every member is a guest. Membership can be implemented by having a Member class that inherits the Guest class and implements all methods of Guest. This means that the Guest class can be substituted by the Member class and the program will still work. The member class can also include additional functionality and methods such as keeping track of their membership and loyalty points in order to provide further perks down the road.

The LSP is satisfied for our current implementation and would still be satisfied after this new feature, allowing the easy substitution of base classes by derived classes. This ensures the easy addition of this new feature into our current application. The OCP is also satisfied because the Member class will inherit all the methods of the Guest class and can overload them to customise functionality to be specific to members. This extensibility of modules provides a simple way for us to introduce new features into our booking system.

# Birthday Dates

To further entice customers for using our services, we can reward them by providing benefits on their birthday month. Since we already have a superclass Events.java, we can utilise this to create a subclass called Birthdays.java which will implement the methods in the parent class but adapted for user’s birthdays. Entries should contain their Name and Birthday, and should not be modifiable once entered except by a user with admin privileges.

The OCP is satisfied because the new Birthday class can overload the method implementations of the Events class and customise it for its own use. Once again, the extensibility affords an easy addition of new features into our system.

# 3.0 Detailed UML Class Diagram

A picture containing diagram

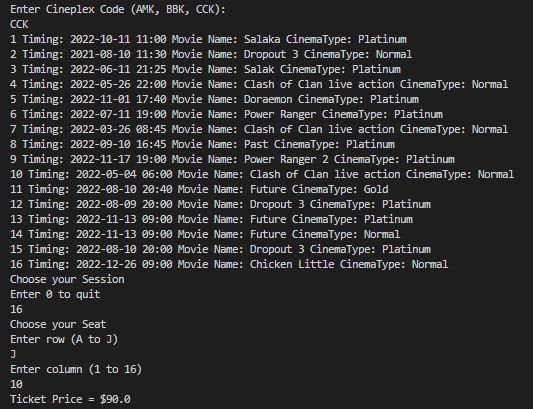
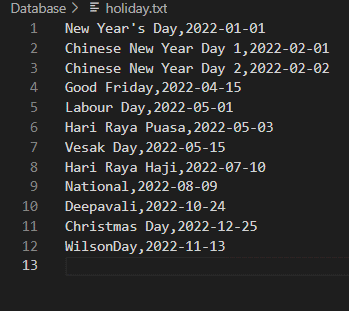
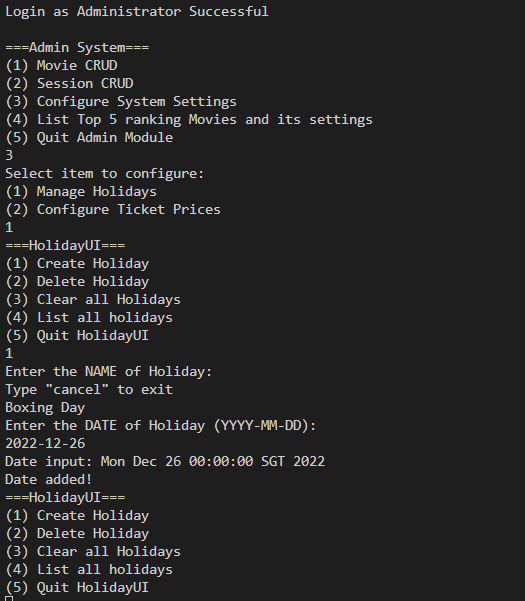
Description automatically generated

*Refer to SSP3-grp2.png in the file*

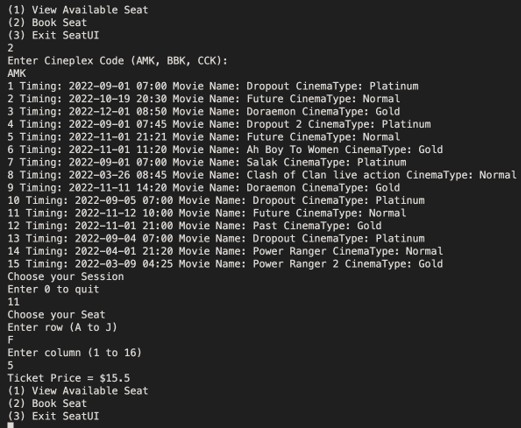
* 1. **Test Cases**

Compulsory Test cases (Inputs in Embedded Excel)

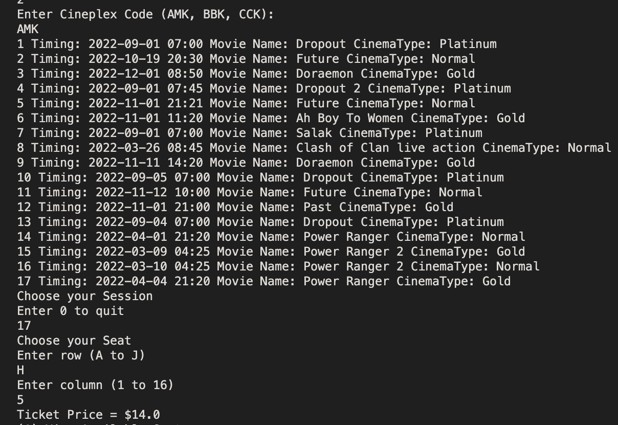
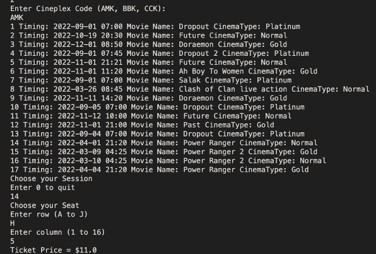
* + 1. Configuring a holiday date and the ticket price is shown correctly when booking is done on that date



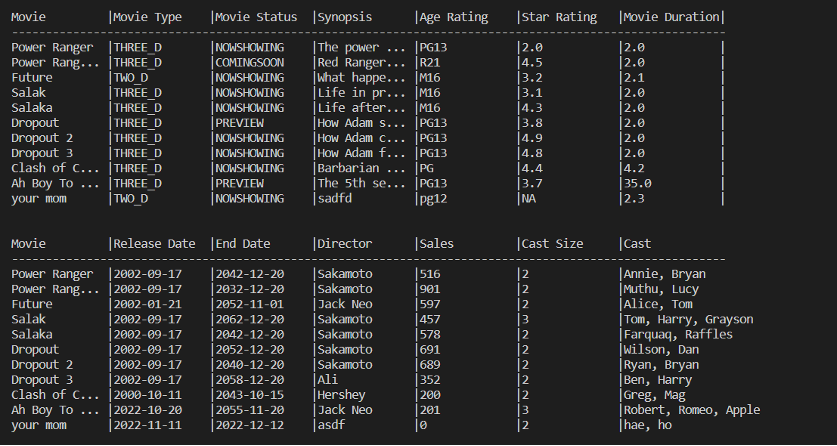
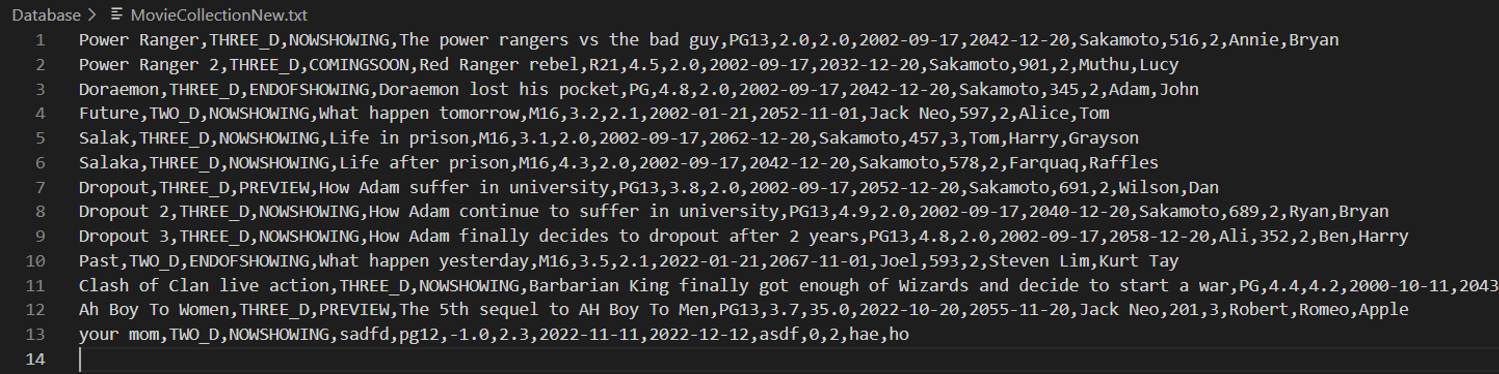
* + 1. Booking on a different day of the week or holiday and type of cinema (eg Gold Class to demonstrate the differences in prices)
       1. Show the different days of the week (Weekends more expensive than Weekdays)



* + - 1. Show different cinema type (Normal class cheaper than Gold class)



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



* + 1. Booking only allowed for “Preview” and “Now Showing” status.

