NANYANG TECHNOLOGICAL UNIVERSITY

CZ2002

MOBLIMA System *Report*

Cao Hannan U1622161K
Heng Weiliang U1620593L
Shen Youlin U1622165J
Zhang Xinye U1622118K

SS3 | GROUP 3

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	Course (CE2002 or CZ2002)	Lab Group	Signature /Date
ao Hannan	Z	SS	曹瀚楠 16/11/2017
Heng Weiliang	Z	SS	[illegible] 16/11/2017
Shen Youlin	Z	SS	[illegible] 16/11/2017
Zhang Xinye	Z	SS	张鑫业 16/11/2017

Important notes:

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

1. Introduction

MOBLIMA is an application to computerize the processes of making online booking and purchase of movie tickets, listing of movies and sale reporting. It will be used by the moviegoers and cinema staff. The application acts as a centralized 'location' for making bookings for all the Cineplexes in different locations managed by the vendor. In this report, design considerations, principles and use of object-oriented programming (OOP) concepts are to be demonstrated. The report also includes a detailed UML Class Diagram for the MOBLIMA and a UML Sequence Diagram for some essential features, followed by screen captures of testing done with test data provided.

2. Design Considerations

a. Implementation of SOLID design principles

i. Single Responsibility Principle (SRP)

The Single Responsibility Principle emphasises the cohesion of classes, such that a class should only focus on what it supposed to do and nothing else. Our whole MOBLIMA application is divided into 3 types of classes: ***model***, ***controller*** and ***view***, corresponding to the entity, controller and boundary classes. Take classes under ***view*** as an example, every class is for one and only one particular menu interacting with the customer or staff, so that there is at most 1 class to be modified in case of a change of user need.

ii. Open-Closed Principle (OCP)

Open-Closed Principle is applied in some of the classes, such that they are open for extension but close for modification. For example, the ***BaseMenu*** class is an abstract class that is the father class of all menus. This allows changing of class responsibility without changing the source code of the classes.

iii. Liskov Substitution Principle (LSP)

Liskov Substitution Principle is well implemented in our application.
For example, as all entity classes are extended from the ***Model*** class, they are also substitutable for the base interface ***Model***. In the ***Manager*** class, the ***getEntry()*** method,

public <T extends Model> T getEntry (Constant.Tables from, Predicate<T> where), takes in a predicate T extends from ***Model*** to filter out the correct entry from the database, and note that this method is capable to return any class type extends from ***Model*** class.

iv. Interface Segregation Principle (ISP)

Interface Segregation principle is achieved by using client specific interfaces instead of using general purpose interface. For example, our ***Model*** interface, implementing the ***Serializable*** interface from ***java.io***, is a better-fit customized interface, which also serves as a tagging interface of our entity classes. So the ***getEntry()*** method in the ***Manager*** class can easily identify whether the type of the incoming data type is a valid entity class or not.

v. Dependency Injection Principle (DIP)

The DIP principle indicates that a high-level module should not depend upon low-level modules. Both should depend upon abstractions. For example, all the menu classes are extended from the ***BaseMenu*** class, which contains an abstract method ***execute()*** to be implemented by the lower level classes for various purposes.

b. Design Patterns

i. Singleton Design Pattern

The Singleton Pattern restricts the instantiation of a class to one and only one object. It is very useful when exactly one object is needed to coordinate actions across the system. For example, operations regarding the database are all contained in the ***Manager*** class and ***PriceManager*** class. Therefore, only one instance of each is allowed to be created. If more than one instances of ***Manager/PriceManager*** class are created, it may lead to some data coherence problems.

c. Object-Oriented Concepts (Explanation of UML Diagram)

i. Inheritance

Inheritance is mainly used in the view classes, i.e. boundary classes. All the menu classes are extended from the abstract ***BaseMenu*** class so that they share the ***getPreviousMenu()*** defined in the base class to prevent code duplicate and redundancy.

ii. Encapsulation

Encapsulation is to restrict the access to the private data, keeping it safe from unwanted interference so that the private attributes are only accessible through getter and setter methods. Also, the implementation details of a class are hidden from outsiders. In our application, encapsulation is applied in almost every entity class.

iii. Polymorphism

Polymorphism is to treat objects of different as a single general type, usually, their base type, and the behaviour of the objects depends on their actual type. In our application, the ***start()*** method from ***MenuLoop*** class uses the concept of polymorphism as it defined a ***BaseMenu*** reference to accommodate all types of menus and then call the execute() methods according to the real type of the menus.

public void start() {BaseMenu nextMenu = entryMenu; ...; }

iv. Abstraction

Abstraction is the development of classes in terms of their interfaces and functionality, instead of their implementation details. For example, the various methods in ***IOUtil*** class provide convenient I/O operations for all the menu (boundary) classes by just calling them, while the implementation and error checking mechanism is all hidden in the methods.

3. Data Storage

Because No database application nor JSON is to be used, we create a ***SerializeDB*** class to store several ArrayList of data we needed and then use ***ObjectOutputStream*** to write the ***SerializeDB*** object directly to a file (***db.ser***). Upon every time of restarting the application, the ***serializeDB*** will be read out from the file and stored in the singleton Manager instance. To interact with the information from the ***SerializeDB*** object, we implement various getter and remove methods in ***Manager*** class for the entity class to use.

4. Assumptions Made

a. The application is for single-user usage and not considering concurrent access.

b. The currency is in Singapore Dollar (SGD) with GST included.

c. Once confirm to book, payment will always be successful.

d. There are no interfaces provide to external systems.

e. Student and senior discount can be applied without validation during purchase.

f. All holidays have a duration of exactly one day.

g. Holidays may have the same name in case of a multi-day holiday.

h. The default ticket price is set according to the ticket price retrieved from the website of Cathay Cineplexes, http://www.cathaycineplexes.com.sg/faqs.aspx.

i. The time is based on 24-hour clock.

j. User can login to check their booking history by using their name and phone number.

k. Movies that are not allowed to book currently will not be displayed in the 'list top 5 sales/rating menu'.

5. UML Class Diagram

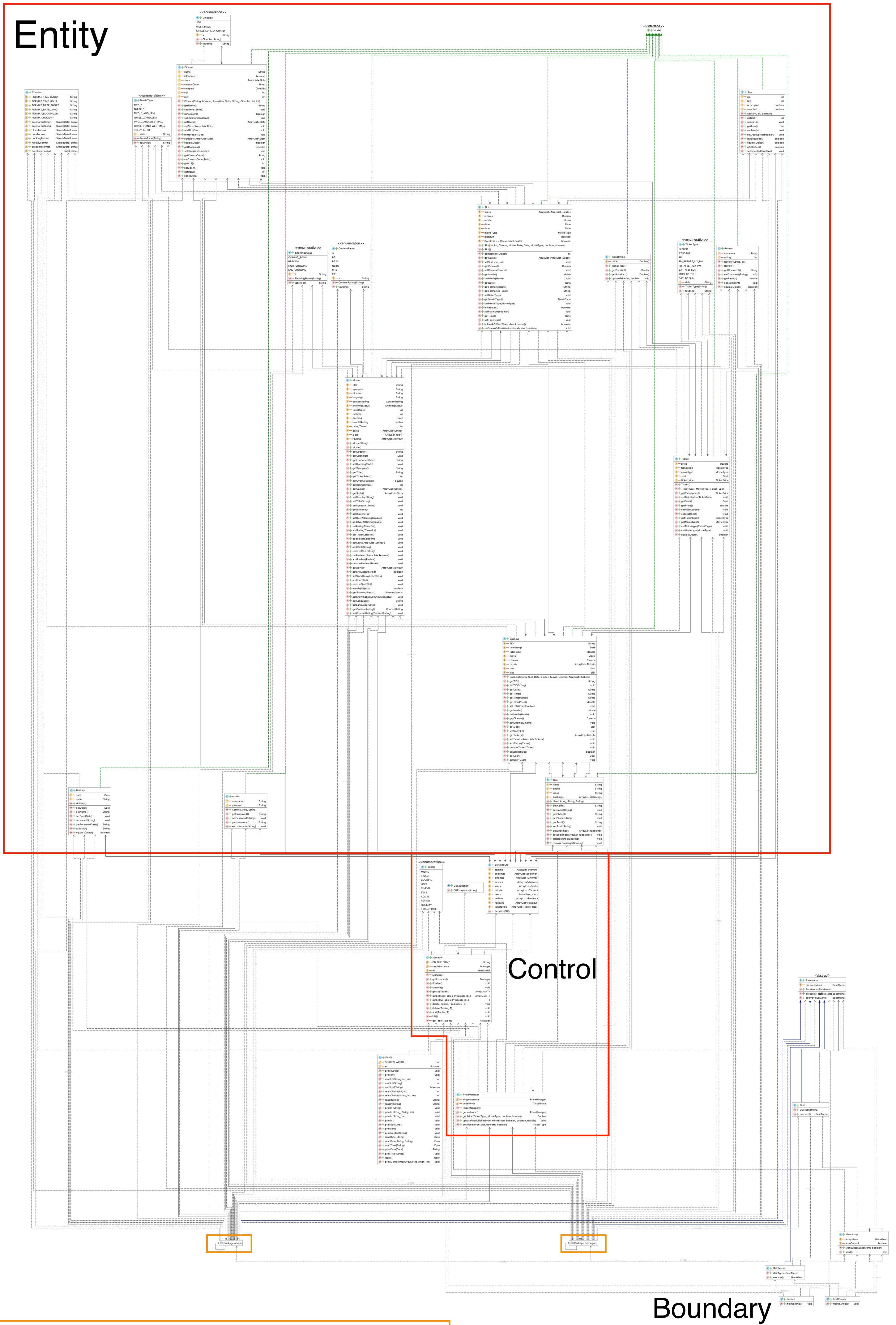
Please refer to the Appendix.

6. UML Sequence Diagram

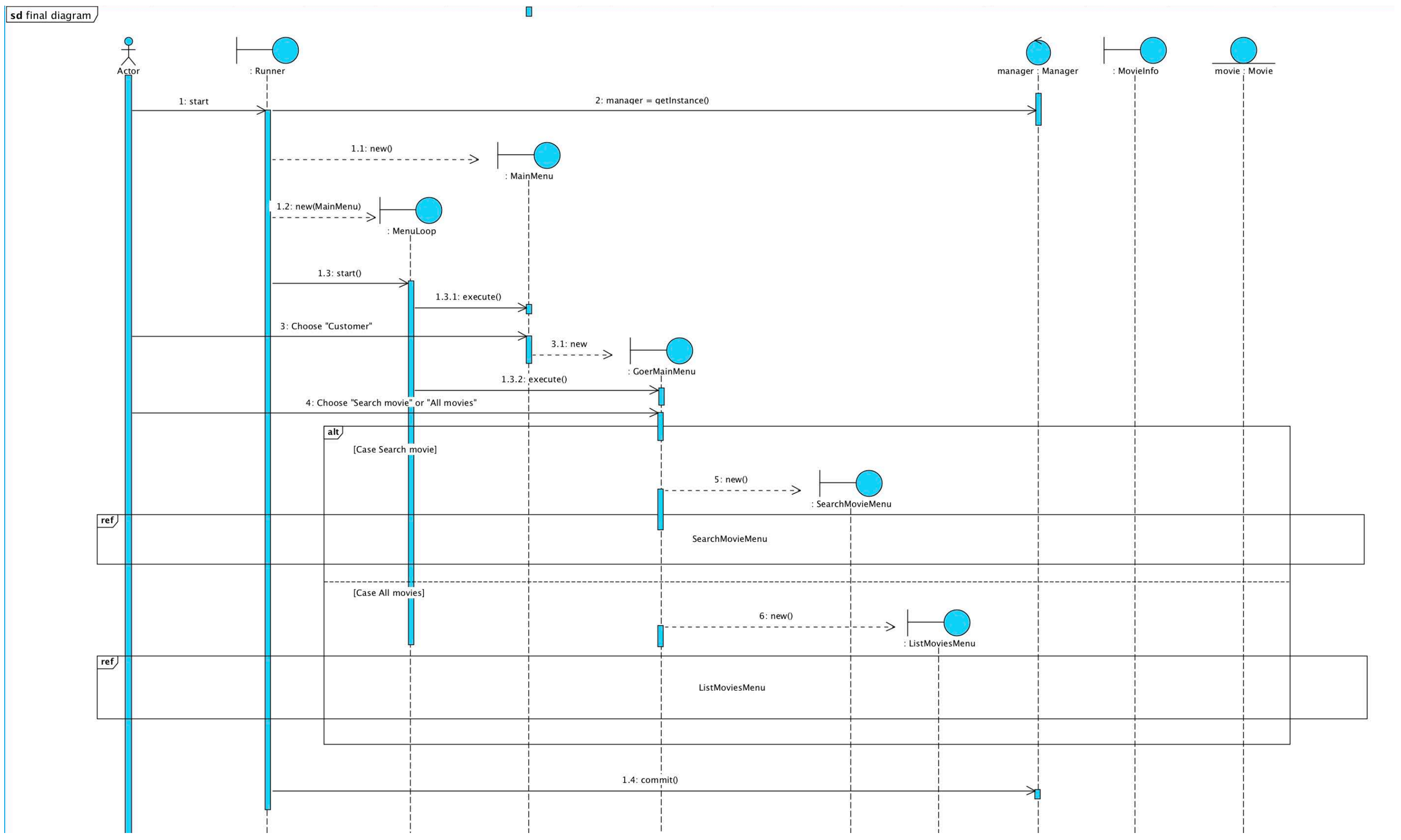
Please refer to the Appendix.

7. Test Cases

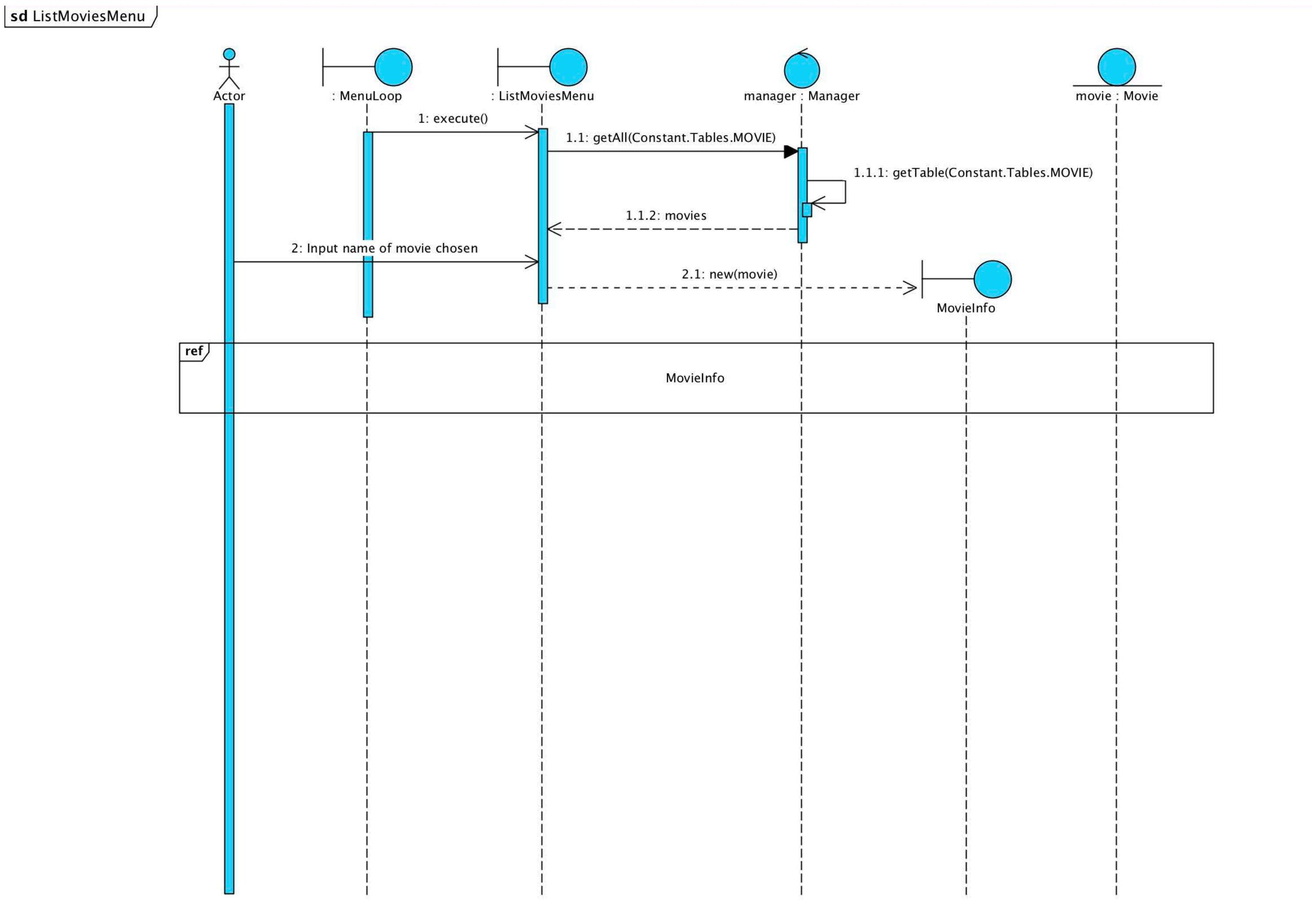
Please refer to the video.



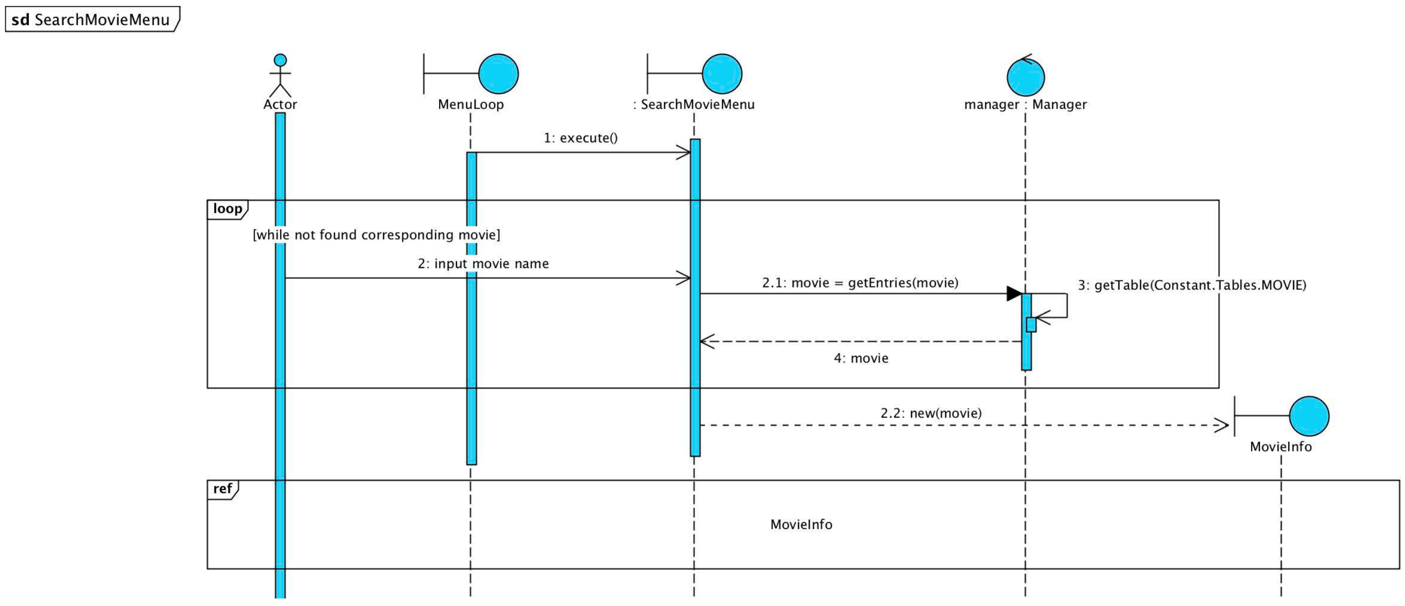
Note: for the readability of the class diagram, specific menus for moviegoers and staffs are encapsulated in their corresponding packages.



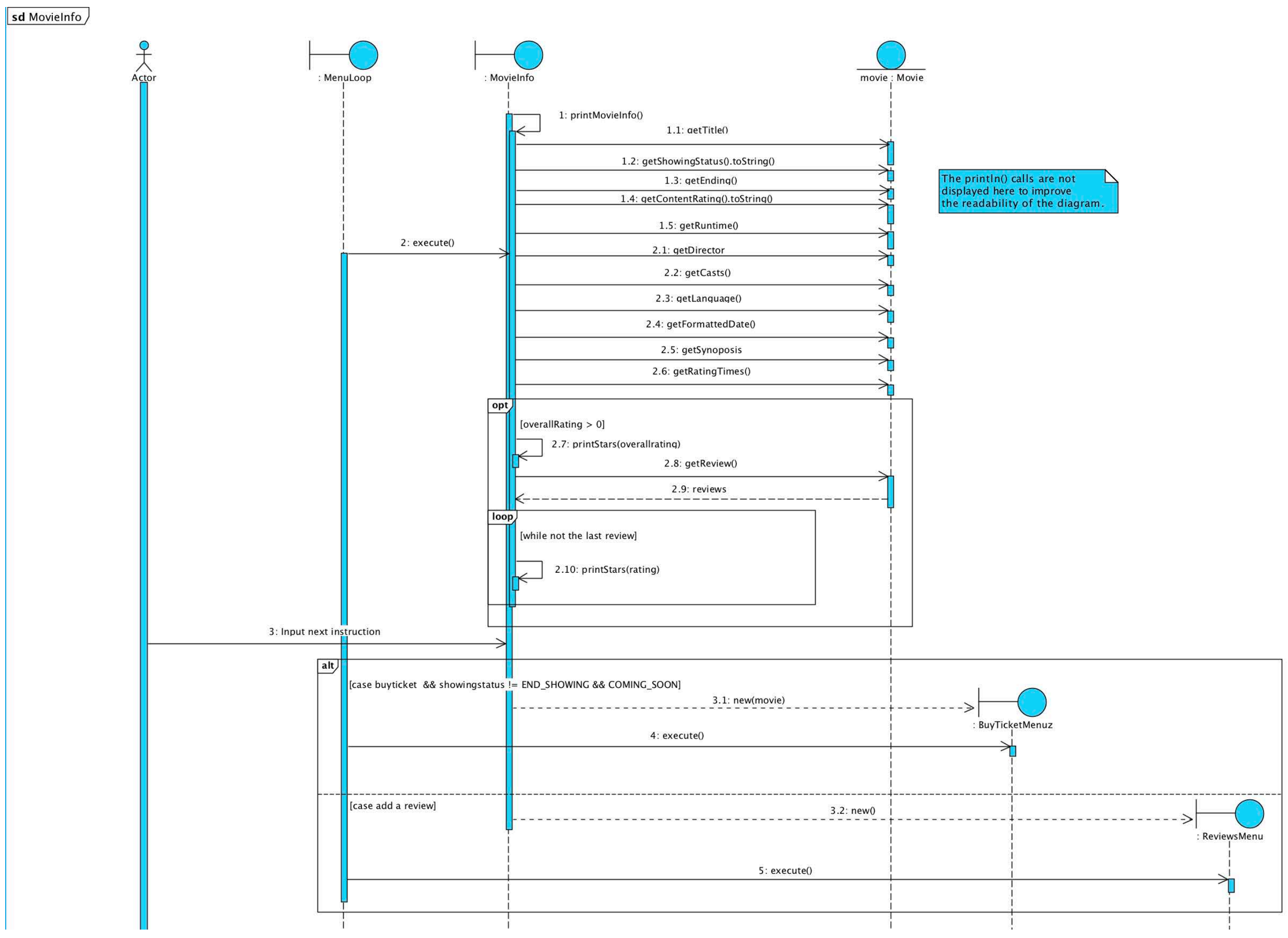
Entry



Movie Listing



Search Movies



Show Movie information