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
Alice Chua Qin Hui	CZ2002	SS1 Group 2	alice 16/11/19
Foo Chuan Sheng	CZ2002	SS1 Group 2	16/11/19
Jiang Ji Xuan	CZ2002	SS1 Group 2	16/11/19
Tanay Bharadwaja	CZ2002	SS1 Group 2	16/11/19
Vincent Ribli	CZ2002	SS1 Group 2	16/11/19

Contents Page

A. Design Considerations

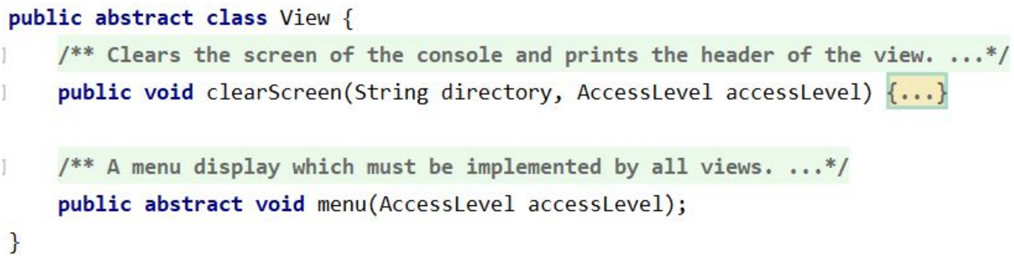
A working application may not mean that it is a well-designed application. It may be working fine at this instant, but inherently show symptoms of design issues such as symptoms of rigidity, fragility or immobility.

In our project, we have considered various basic OO concepts including abstraction, encapsulation, inheritance, and polymorphism, as well as design considerations and principles including the SOLID design principles when implementing two new features in our application. Before looking at what we can add to this application, one must understand the design considerations used. The code used in this project is split up into three main components:

1. Views, inheriting from `View` class
2. Systems, implementing the `MOBLIMASystem` interface
3. Entities

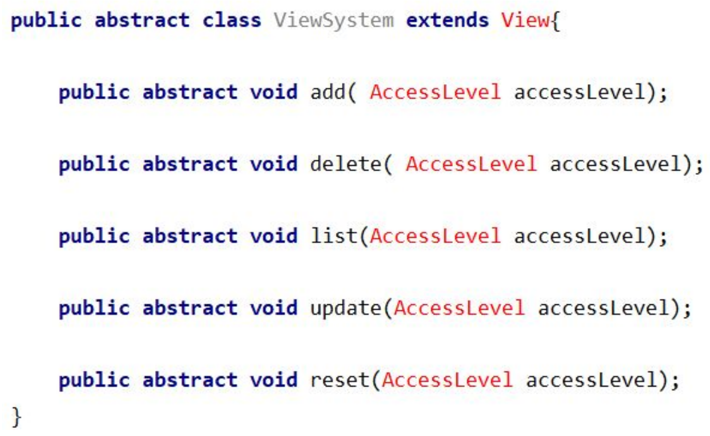
This ensures the Single Responsibility Principle, where view is responsible as a boundary input/output, systems are responsible for the logics and entities to represent the objects.

Views



```java
public abstract class View {
    /** Clears the screen of the console and prints the header of the view. ...*/
    public void clearScreen(String directory, AccessLevel accessLevel) {...}

    /** A menu display which must be implemented by all views. ...*/
    public abstract void menu(AccessLevel accessLevel);
}
```

Figure 1: `View` methods



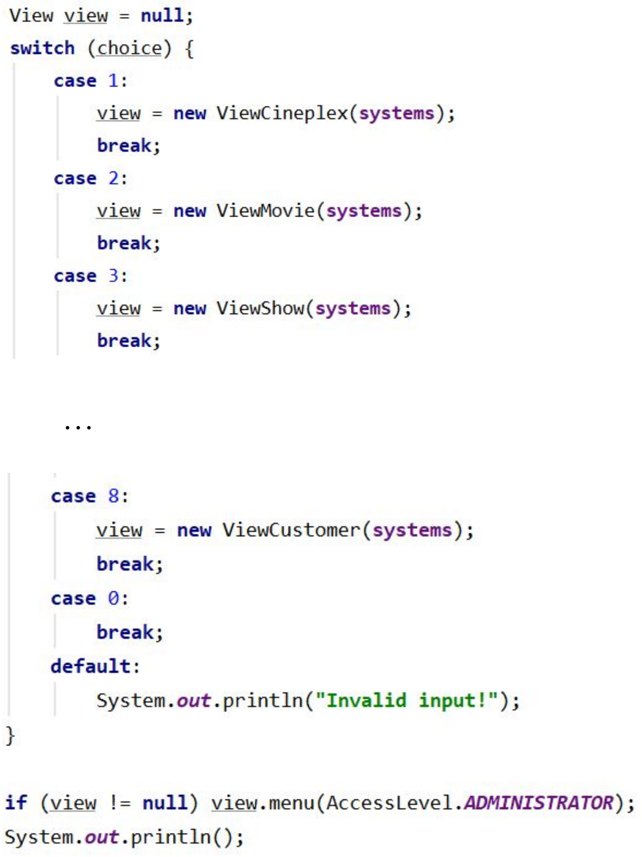
```java
public abstract class ViewSystem extends View{

    public abstract void add( AccessLevel accessLevel);

    public abstract void delete( AccessLevel accessLevel);

    public abstract void list(AccessLevel accessLevel);

    public abstract void update(AccessLevel accessLevel);

    public abstract void reset(AccessLevel accessLevel);
}
```

Figure 2: `ViewSystem` methods

Inheritance is a mechanism that defines a new class (child class) that inherits the properties and behaviours of a parent class. To generate the views for the systems, we have an abstract superclass called `ViewSystem` which inherits from `View`, containing abstract methods which is shared by all subclasses. Classes inheriting from the class `View` will inherit useful methods such as `clearScreen()` and are made to implement `menu()`, demonstrating inheritance.

`ViewSystem` parent class includes the universal methods of `add`, `delete`, `list`, `update`, and `reset` which must be implemented by all the view systems. These methods are left with an empty body, so that the responsibilities of each individual child class can be implemented separately. This is an example of the **Open/Closed Principle** from SOLID principles, as we will be able to change what the modules do without changing the abstract class.

A property of the **Dependency Injection Principle** (DIP) states that abstractions should not depend upon details. Details should depend upon abstractions. This can be seen from the `ViewSystem` class in Figure 2, where all abstract methods have empty bodies. These details are later added in the sub-classes where explicit responsibilities of each class are outlined. This ensures that we would not make the code more complicated than it needs to be; not having to work round the details.



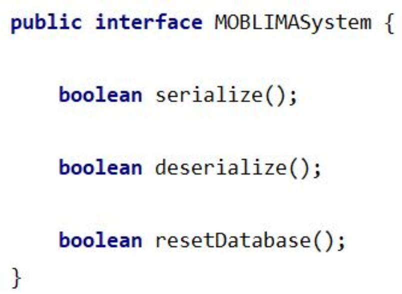
```
View view = null;
switch (choice) {
    case 1:
        view = new ViewCineplex(systems);
        break;
    case 2:
        view = new ViewMovie(systems);
        break;
    case 3:
        view = new ViewShow(systems);
        break;

     …

    case 8:
        view = new ViewCustomer(systems);
        break;
    case 0:
        break;
    default:
        System.out.println("Invalid input!");
}

if (view != null) view.menu(AccessLevel.ADMINISTRATOR);
System.out.println();
```

Figure 3: Selecting a View

The inheritance also allows for **polymorphism**, where each view performs the same method name menu, but with different implementations appropriate to its class. In the case of the `View` and `ViewSystem` class as shown in Figure 1, it contains the different possible methods which all `View` classes use, however, the implementation of these methods is different and are altered to perform operations appropriate to its class. As such the `'add'` method used in the `ViewBooking` class adds a new booking made by customers or admins, but the same add method can be used to add a date in the `ViewHoliday` class. Both classes are adding new things into their database, but it will done be in varied formats; showing how polymorphism allows for similar things to be done differently.

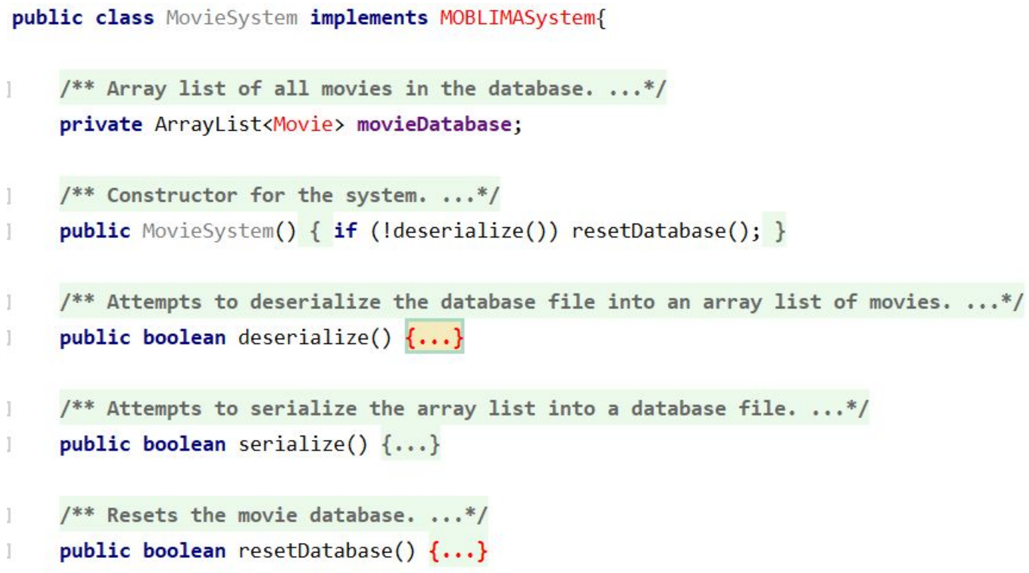
In Figure 3, we demonstrate how a view can be selected. As a view is instantiated, we can simply call `view.menu(AccessLevel)` for any of the inherited views, demonstrating polymorphism.

MOBLIMA Systems



```
public interface MOBLIMASystem {

    boolean serialize();

    boolean deserialize();

    boolean resetDatabase();
}
```

Figure 4: MOBLIMASystem Interface



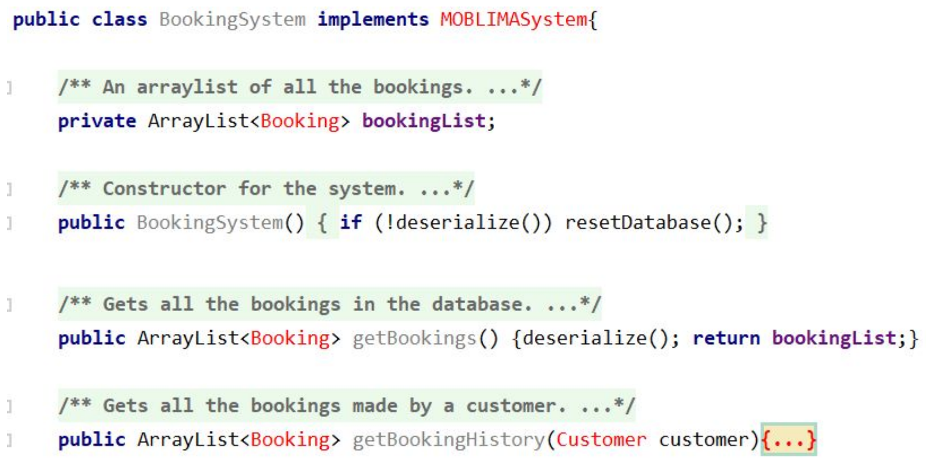
```
public class MovieSystem implements MOBLIMASystem{

    /** Array list of all movies in the database. ...*/
    private ArrayList<Movie> movieDatabase;

    /** Constructor for the system. ...*/
    public MovieSystem() { if (!deserialize()) resetDatabase(); }

    /** Attempts to deserialize the database file into an array list of movies. ...*/
    public boolean deserialize() {...}

    /** Attempts to serialize the array list into a database file. ...*/
    public boolean serialize() {...}

    /** Resets the movie database. ...*/
    public boolean resetDatabase() {...}
```

Figure 5: Movie System Using Methods from the MOBLIMASystem

We implemented a `MOBLIMASystem` interface for our systems. This helps to promote standardization throughout the project amongst the classes of similar purposes, important especially with multiple collaborators. In the `MOBLIMASystem` interface, methods declared must be implemented by all classes implementing the `MOBLIMASystem` interface; since all subclasses require similar methods. Through inheriting the methods from the superclass, we can make sure that these methods are correctly and efficiently implemented in the project across all systems.

According to DIP, a high-level module should not depend upon a low-level module. When high-level modules (`MOBLIMASystem`) are independent of low-level modules (i.e. `MovieSystem`), the high-level modules can then be reusable for other low-level modules such as `CustomerSystem` and `ShowSystem`. This can be done through the implementation of the high-level modules.

By making more system classes, responsibilities are split up with the **Single Responsibility Principle** in mind. Each system inherits the methods to be used only for their purpose, i.e. the PriceSystem's sole responsibility is to handle all the base prices we have set. Despite having to split up into smaller roles within the class, but it only has one main responsibility; which is to handle all pricing used in our application.



```
public class BookingSystem implements MOBLIMASystem{

    /** An arraylist of all the bookings. ...*/
    private ArrayList<Booking> bookingList;

    /** Constructor for the system. ...*/
    public BookingSystem() { if (!deserialize()) resetDatabase(); }


    /** Gets all the bookings in the database. ...*/
    public ArrayList<Booking> getBookings() {deserialize(); return bookingList;}

    /** Gets all the bookings made by a customer. ...*/
    public ArrayList<Booking> getBookingHistory(Customer customer){...}
```

Figure 6: BookingSystem Encapsulation

Lastly, encapsulation is used throughout the project. **Encapsulation** is the concept of building a barrier to protect an object's private data; the idea of information hiding is to prevent the user from going through the details of the implementation of the different classes. In the case of our project, sensitive data like customers' and admins' personal data are all private data which should not be accessible to the public. Encapsulation only requires its users to know what each class does and how to access the methods to get the desired outputs. From Figure 4, using the `BookingSystem` class, the users do not need to know all the bookings stored in the database, but only the bookings that they have made.

With the design considerations, here are two features that we propose to be added.

- **Promotions**

  A new feature that we can foresee being added to our system would be the ability for the console admins to implement exciting promotions that will allow Golden Town to grow. The ability to add promotions would be essential modifiers to the price of a movie ticket in some circumstances, however, must be kept separate from the base price system to ensure efficient and sustained function of the system after the promotion has ended.

  The road to implementing such a feature is made easier through the implementation of design principles in our project. As seen above, the use of the `MOBLIMASystem` and `ViewSystem` class will make this process much easier and standardized with the rest of the application, for serialisation, adding, editing, deleting and listing all the available promotions, and the seamless implementation of the view to include the header of the system.

  We shall also ensure Single Responsibility Principle. The control takes place in the system and the input/output takes place in the view.

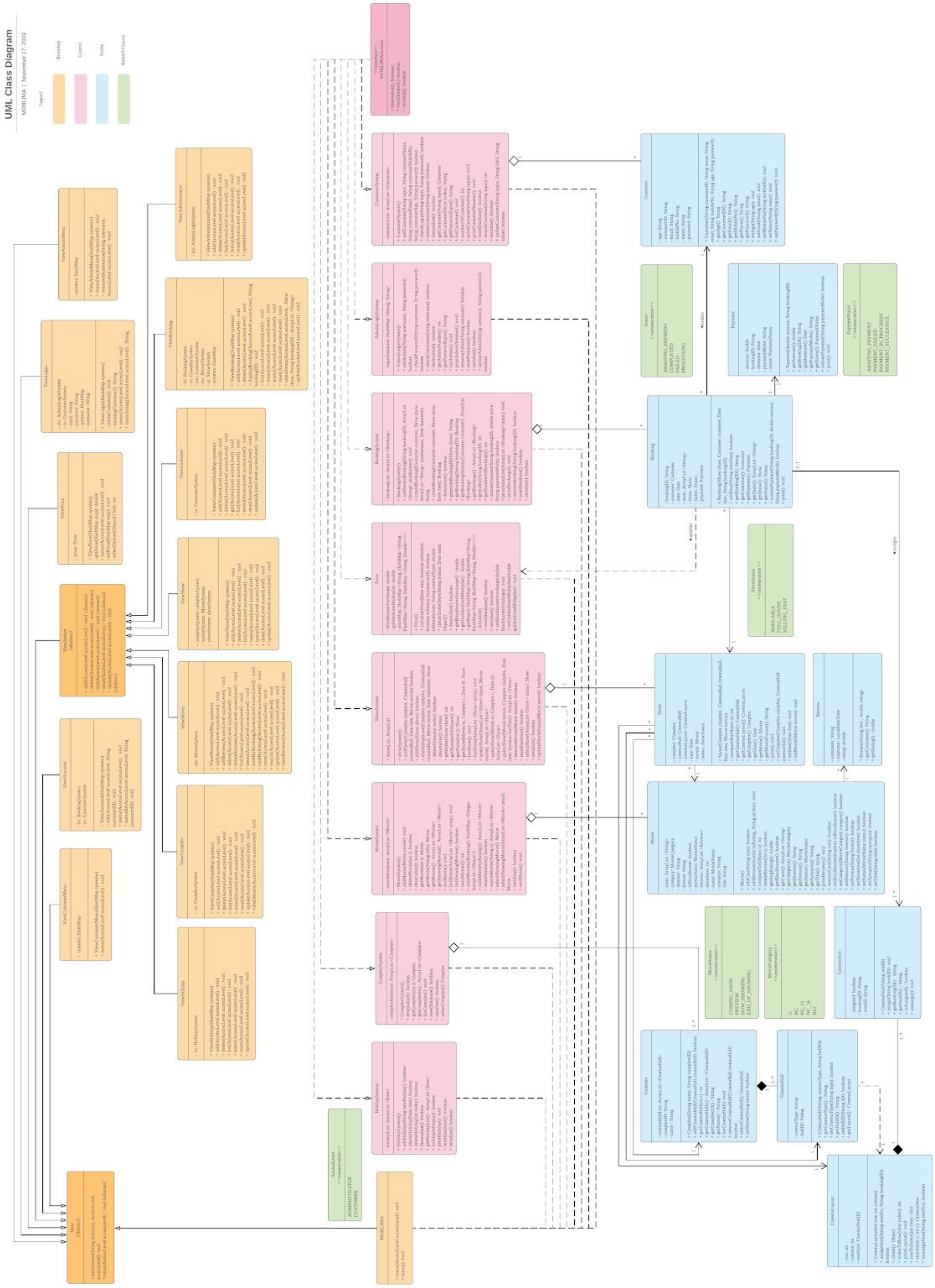
- **Extension of Payment**

  Another feature which can be added would be a payment extension page; which takes in the card details as well as the one-time password (OTP) from the customer when making payment with cards. Through this new feature, we can increase the security of the payment system.

  This feature can be an extension from the `Payment` entity which we have already implemented in our code. By modifying the `initiatePayment()` method and adding new functionalities in the Payment entity, a new view as well as a change in the details shown when showing bookings, we can immediately implement this new feature. This opens up exciting opportunities to collaborate with different payment APIs such as Mastercard and VISA.

Thus, with the principles and design considerations explained above, we are able to implement these two features efficiently and quickly.

B. **UML Class Diagram** (for higher resolution please refer to bit.ly/ss1g2class)

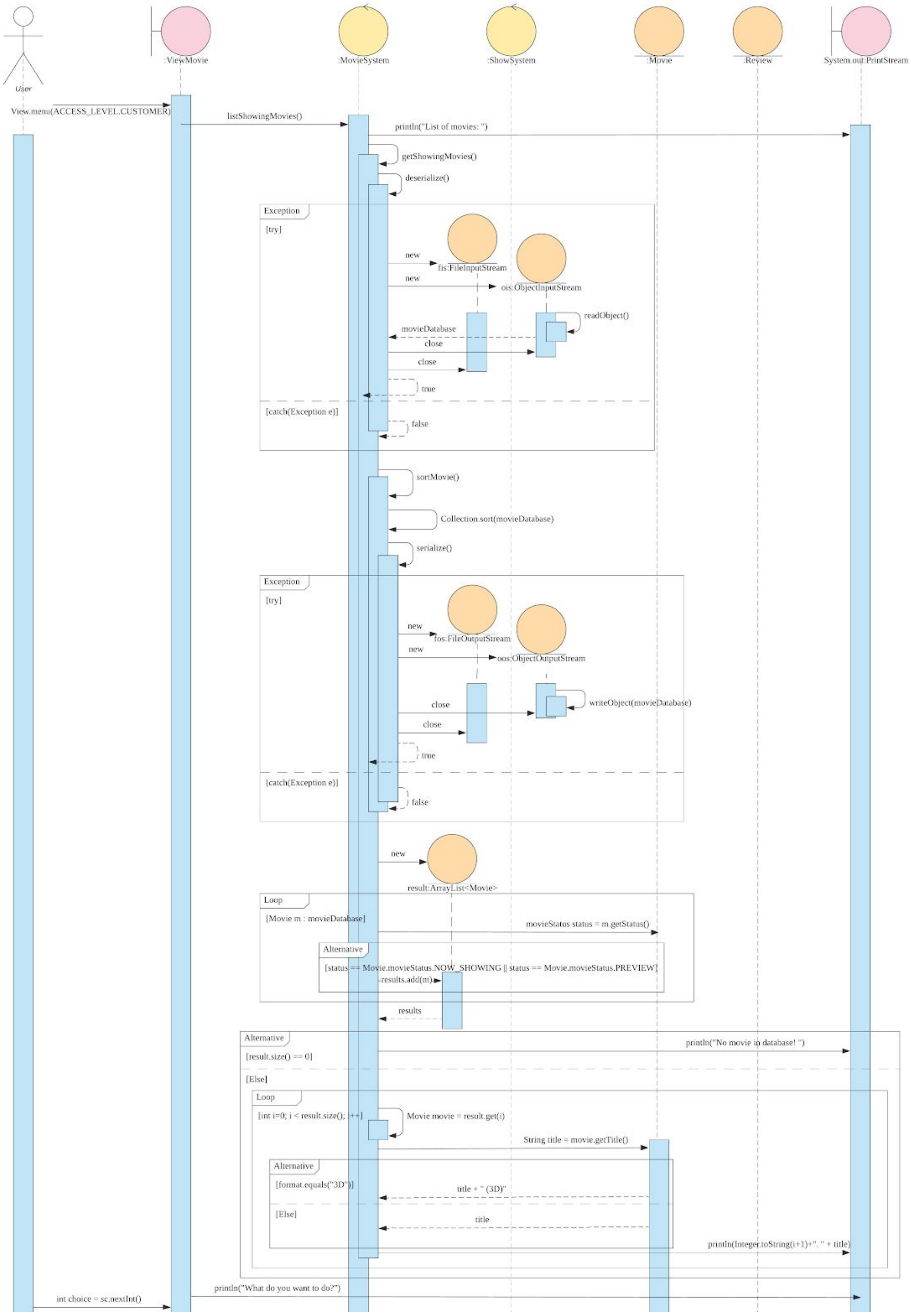


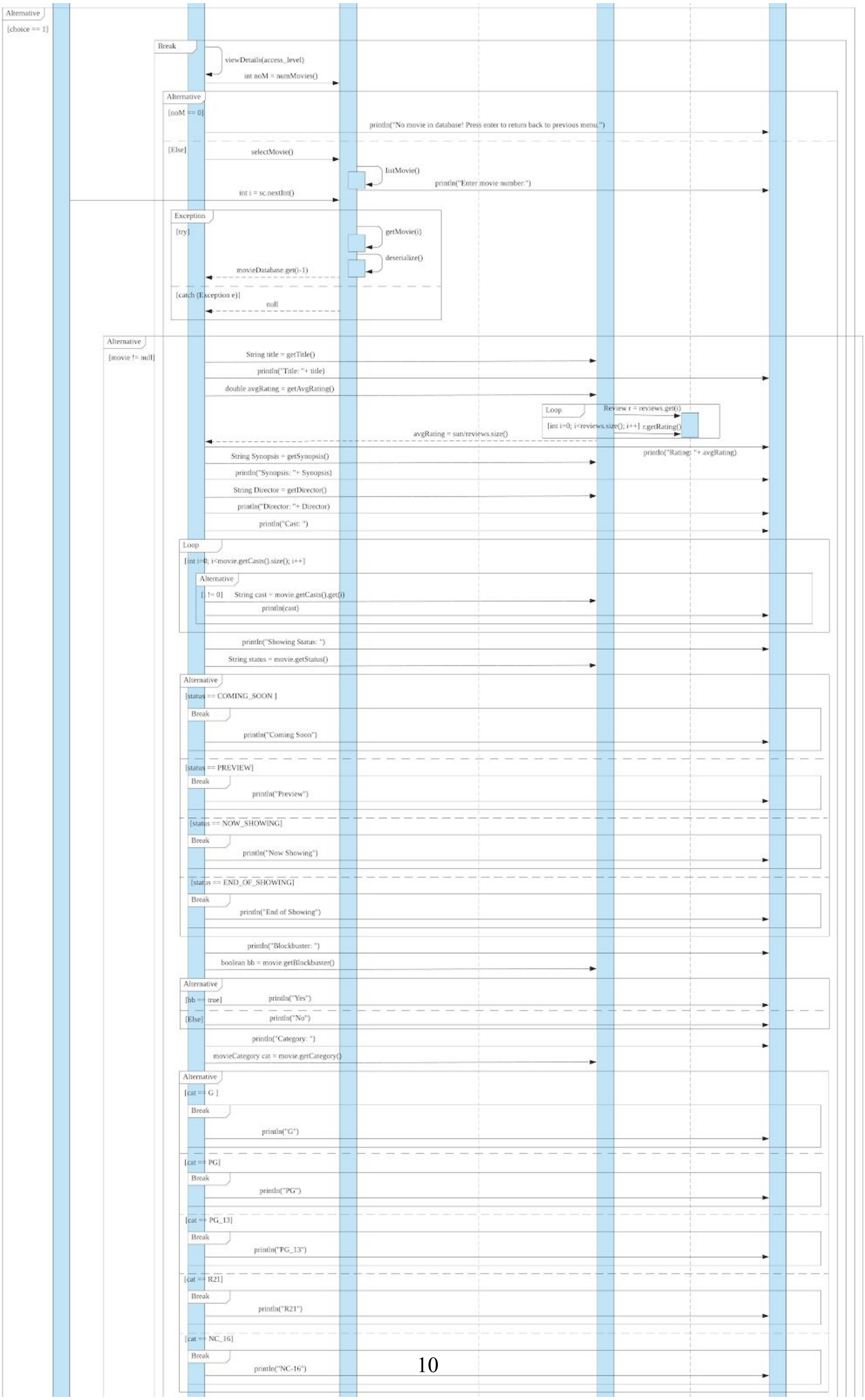
C. UML Sequence Diagram (for higher resolution please refer to bit.ly/ss1g2sequence)

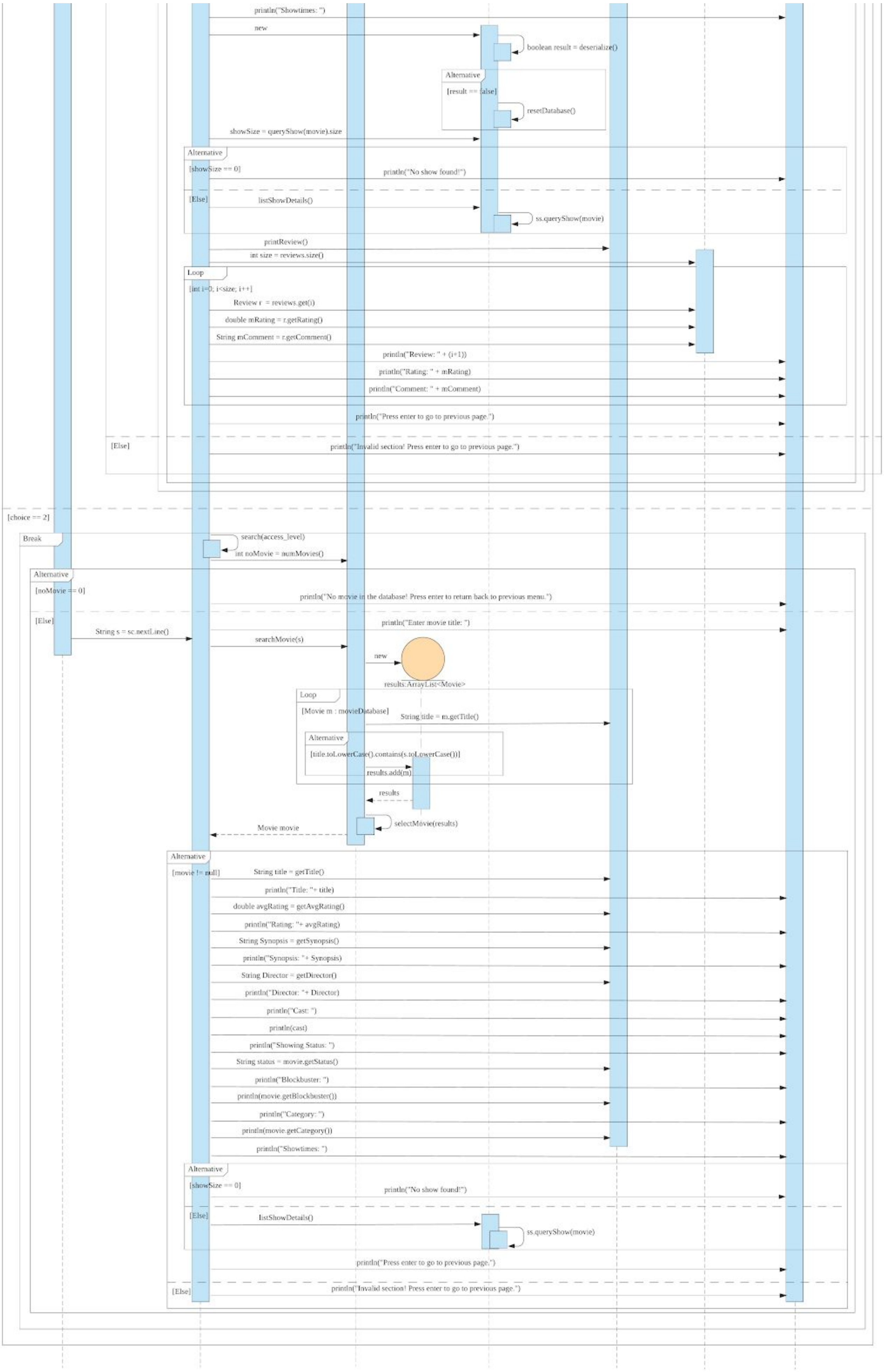
The use case below is when a user uses our interface to check out our movies. Users can choose to list all showing movies and select a movie to view its details, or search using our search tool based on the title. The program has the following flow:

- When user selects the movie module, the movie system is deserialised.
- The movie list is sorted based on title, and serialised back to the file.
- Iterating through the array, movies which are 'Now Showing or 'Preview' are added to a result array.
- The content of the result array is displayed to the user as a list of showing movies.
- Users have two choices, to select a movie from the list or to search using the title.
- If user chooses to select a movie from the list, the following happens.
  - The movie database is first checked. If there is no movie is in the database, user is asked to return back to previous menu.
  - If there are movies, the movies are displayed and user is asked to select a movie number.
  - The movie database array list is queried using the get operator and using the given index.
  - Details such as title, synopsis, director, cast, showing status, average rating, category, showtimes and reviews are displayed.
  - To obtain the average rating and reviews, the attribute 'reviews' of each movie, which is an array list of the object Review, is called.
- If user chooses to search a movie based on title, the following happens.
  - User is asked to give a string for search. It is then checked against every movie in the database if it is a substring of the title. If it is, it is added to a results array and listed to the user.
  - User is then asked to select a movie from the results array.
  - After selection, details of the movie is shown similar to the select movie function above.

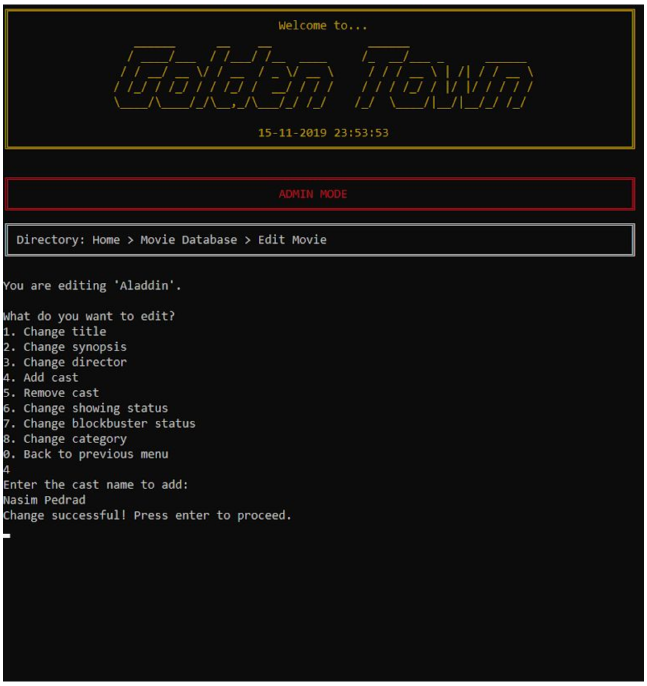
The UML sequence diagram on the next page illustrates the above use case. For methods which are performed more than once, they detailed sequence will only be shown once. For user input, the scanner class was assumed and not presented in the following sequence diagram.


User
:ViewMovie
:MovieSystem
:ShowSystem
:Movie
:Review
System.out:PrintStream
View.menu(ACCESS_LEVEL.CUSTOMER)
listShowingMovies()
println("List of movies: ")
getShowingMovies()
deserialize()
Exception
[try]
new
fis:FileInputStream
new
ois:ObjectInputStream
readObject()
movieDatabase
close
close
true
[catch(Exception e)]
false
sortMovie()
Collection.sort(movieDatabase)
serialize()
Exception
[try]
new
fos:FileOutputStream
new
oos:ObjectOutputStream
close
writeObject(movieDatabase)
close
true
[catch(Exception e)]
false
new
result:ArrayList<Movie>
Loop
[Movie m : movieDatabase]
movieStatus status = m.getStatus()
Alternative
[status == Movie.movieStatus.NOW_SHOWING || status == Movie.movieStatus.PREVIEW]
results.add(m)
results
Alternative
println("No movie in database! ")
[result.size() == 0]
[Else]
Loop
[int i=0; i < result.size(); i++]
Movie movie = result.get(i)
String title = movie.getTitle()
Alternative
[format.equals("3D")]
title + " (3D)"
[Else]
title
println(Integer.toString(i+1)+". " + title)
println("What do you want to do?")
int choice = sc.nextInt()

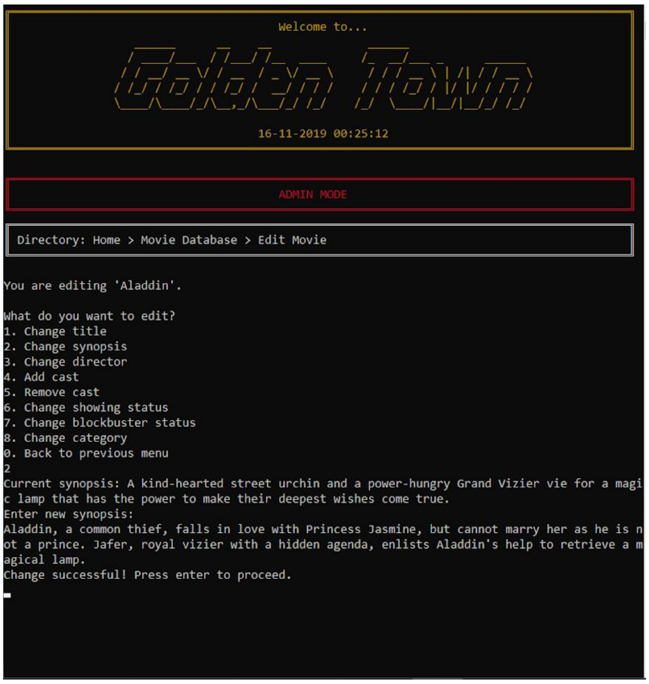

Alternative
[choice == 1]
Break
viewDetails(access_level)
int noM = numMovies()
Alternative
[noM == 0]
println("No movie in database! Press enter to return back to previous menu.")
[Else]
selectMovie()
listMovie()
println("Enter movie number:")
int i = sc.nextInt()
Exception
[try]
getMovie(i)
deserialize()
movieDatabase.get(i-1)
[catch (Exception e)]
null
Alternative
[movie != null]
String title = getTitle()
println("Title: "+ title)
double avgRating = getAvgRating()
Loop
Review r = reviews.get(i)
[int i=0; i<reviews.size(); i++]
r.getRating()
avgRating = sun/reviews.size()
println("Rating: "+ avgRating)
String Synopsis = getSynopsis()
println("Synopsis: "+ Synopsis)
String Director = getDirector()
println("Director: "+ Director)
println("Cast: ")
Loop
[int i=0; i<movie.getCasts().size(); i++]
Alternative
[i != 0]
String cast = movie.getCasts().get(i)
println(cast)
println("Showing Status: ")
String status = movie.getStatus()
Alternative
[status == COMING_SOON]
Break
println("Coming Soon")
[status == PREVIEW]
Break
println("Preview")
[status == NOW_SHOWING]
Break
println("Now Showing")
[status == END_OF_SHOWING]
Break
println("End of Showing")
println("Blockbuster: ")
boolean bb = movie.getBlockbuster()
Alternative
[bb == true]
println("Yes")
[Else]
println("No")
println("Category: ")
movieCategory cat = movie.getCategory()
Alternative
[cat == G]
Break
println("G")
[cat == PG]
Break
println("PG")
[cat == PG_13]
Break
println("PG_13")
[cat == R21]
Break
println("R21")
[cat == NC_16]
Break
println("NC-16")


println("Showtimes: ")
new
boolean result = deserialize()
Alternative
[result == false]
resetDatabase()
showSize = queryShow(movie).size
Alternative
[showSize == 0]
println("No show found!")
[Else]
listShowDetails()
ss.queryShow(movie)
printReview()
int size = reviews.size()
Loop
[int i=0; i<size; i++]
Review r = reviews.get(i)
double mRating = r.getRating()
String mComment = r.getComment()
println("Review: " + (i+1))
println("Rating: " + mRating)
println("Comment: " + mComment)
println("Press enter to go to previous page.")
[Else]
println("Invalid section! Press enter to go to previous page.")
[choice == 2]
Break
search(access_level)
int noMovie = numMovies()
Alternative
[noMovie == 0]
println("No movie in the database! Press enter to return back to previous menu.")
[Else]
println("Enter movie title: ")
String s = sc.nextLine()
searchMovie(s)
new
results:ArrayList<Movie>
Loop
[Movie m : movieDatabase]
String title = m.getTitle()
Alternative
[title.toLowerCase().contains(s.toLowerCase())]
results.add(m)
results
selectMovie(results)
Movie movie
Alternative
[movie != null]
String title = getTitle()
println("Title: "+ title)
double avgRating = getAvgRating()
println("Rating: "+ avgRating)
String Synopsis = getSynopsis()
println("Synopsis: "+ Synopsis)
String Director = getDirector()
println("Director: "+ Director)
println("Cast: ")
println(cast)
println("Showing Status: ")
String status = movie.getStatus()
println("Blockbuster: ")
println(movie.getBlockbuster())
println("Category: ")
println(movie.getCategory())
println("Showtimes: ")
Alternative
[showSize == 0]
println("No show found!")
[Else]
listShowDetails()
ss.queryShow(movie)
println("Press enter to go to previous page.")
[Else]
println("Invalid section! Press enter to go to previous page.")

D. Screen Captures of Testing

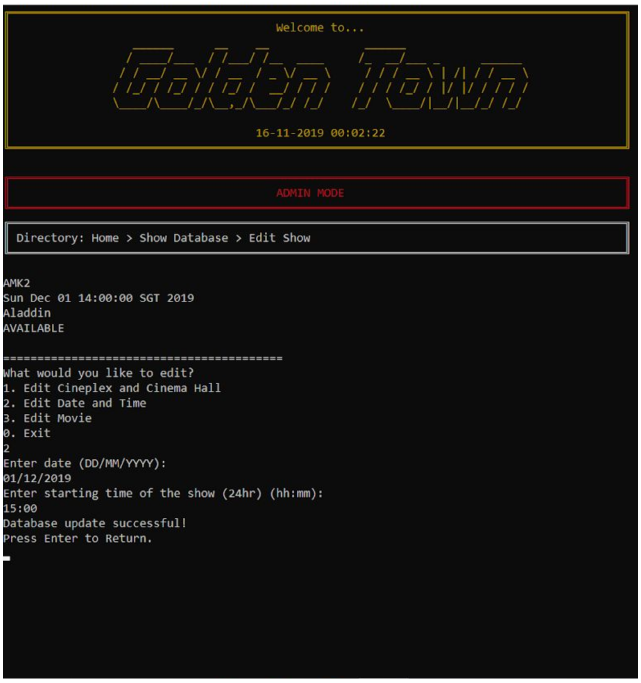


```
Welcome to...

15-11-2019 23:53:53

ADMIN MODE

Directory: Home > Movie Database > Edit Movie

You are editing 'Aladdin'.

What do you want to edit?
1. Change title
2. Change synopsis
3. Change director
4. Add cast
5. Remove cast
6. Change showing status
7. Change blockbuster status
8. Change category
0. Back to previous menu
4
Enter the cast name to add:
Nasim Pedrad
Change successful! Press enter to proceed.
```

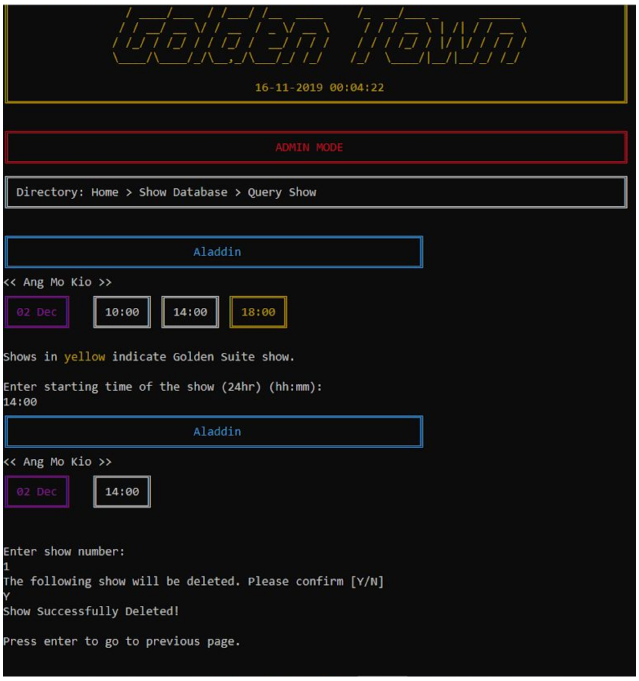


```
Welcome to...

16-11-2019 00:25:12

ADMIN MODE

Directory: Home > Movie Database > Edit Movie

You are editing 'Aladdin'.

What do you want to edit?
1. Change title
2. Change synopsis
3. Change director
4. Add cast
5. Remove cast
6. Change showing status
7. Change blockbuster status
8. Change category
0. Back to previous menu
2
Current synopsis: A kind-hearted street urchin and a power-hungry Grand Vizier vie for a magi
c lamp that has the power to make their deepest wishes come true.
Enter new synopsis:
Aladdin, a common thief, falls in love with Princess Jasmine, but cannot marry her as he is n
ot a prince. Jafer, royal vizier with a hidden agenda, enlists Aladdin's help to retrieve a m
agical lamp.
Change successful! Press enter to proceed.
```

Within administrator mode, we are able to edit movie details such as its title, synopsis, director etc. For example, we added a cast to the movie database as well as updating its synopsis in the pictures above.



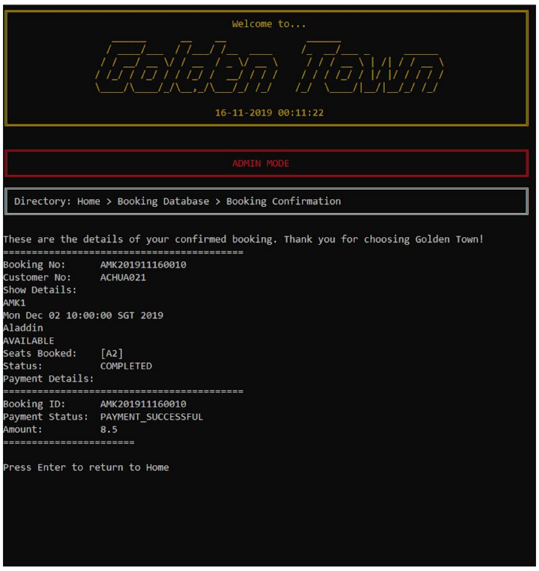
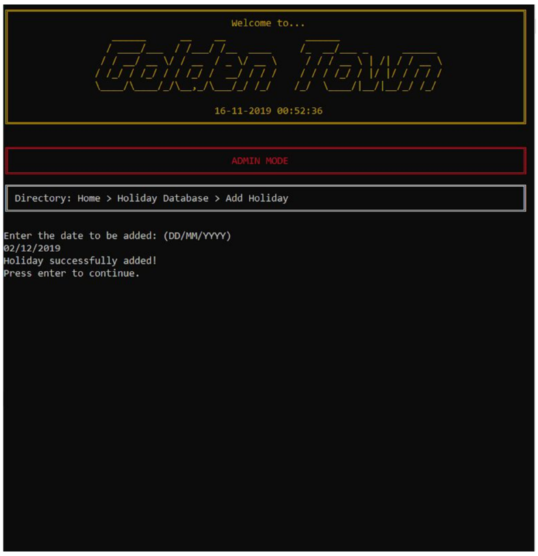
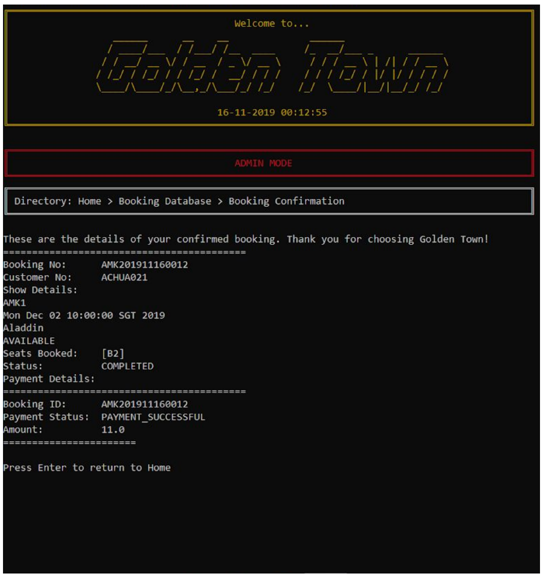
```
Welcome to...

16-11-2019 00:02:22

ADMIN MODE

Directory: Home > Show Database > Edit Show

AMK2
Sun Dec 01 14:00:00 SGT 2019
Aladdin
AVAILABLE

========================================
What would you like to edit?
1. Edit Cineplex and Cinema Hall
2. Edit Date and Time
3. Edit Movie
0. Exit
2
Enter date (DD/MM/YYYY):
01/12/2019
Enter starting time of the show (24hr) (hh:mm):
15:00
Database update successful!
Press Enter to Return.
```



```
16-11-2019 00:04:22

ADMIN MODE

Directory: Home > Show Database > Query Show

Aladdin
<< Ang Mo Kio >>
02 Dec    10:00   14:00   18:00

Shows in yellow indicate Golden Suite show.

Enter starting time of the show (24hr) (hh:mm):
14:00
Aladdin
<< Ang Mo Kio >>
02 Dec    14:00


Enter show number:
1
The following show will be deleted. Please confirm [Y/N]
Y
Show Successfully Deleted!

Press enter to go to previous page.
```

Other than that, within the administrator mode, show details such as cineplex details, date and movies can be updated. For illustration purposes, we have updated the starting showtime for Aladdin movie on December 01 from 14:00 to 15:00. We have also demonstrated the deletion of a showing for Aladdin on December 02 14:00.

The table below illustrates how setting a holiday changes the price of the show ticket.

 Welcome to... Golden Town 16-11-2019 00:11:22 ADMIN MODE Directory: Home > Booking Database > Booking Confirmation These are the details of your confirmed booking. Thank you for choosing Golden Town! == Booking No: AMK201911160010 Customer No: ACHUA021 Show Details: AMK1 Mon Dec 02 10:00:00 SGT 2019 Aladdin AVAILABLE Seats Booked: [A2] Status: COMPLETED Payment Details: == Booking ID: AMK201911160010 Payment Status: PAYMENT_SUCCESSFUL Amount: 8.5 ======================= Press Enter to return to Home	The picture on the left shows the original price of $8.50 for booking a seat at Ang Mo Kio Cineplex on December 02 10:00 before we have assigned December 02 as a holiday through administrator mode.
 Welcome to... Golden Town 16-11-2019 00:52:36 ADMIN MODE Directory: Home > Holiday Database > Add Holiday Enter the date to be added: (DD/MM/YYYY) 02/12/2019 Holiday successfully added! Press enter to continue.	The picture on the left shows the process of assigning December 02 as a holiday through administrator mode.
 Welcome to... Golden Town 16-11-2019 00:12:55 ADMIN MODE Directory: Home > Booking Database > Booking Confirmation These are the details of your confirmed booking. Thank you for choosing Golden Town! == Booking No: AMK201911160012 Customer No: ACHUA021 Show Details: AMK1 Mon Dec 02 10:00:00 SGT 2019 Aladdin AVAILABLE Seats Booked: [B2] Status: COMPLETED Payment Details: == Booking ID: AMK201911160012 Payment Status: PAYMENT_SUCCESSFUL Amount: 11.0 ======================= Press Enter to return to Home	The picture on the left shows the new price of $11.00 for booking a seat at Ang Mo Kio Cineplex on December 02 10:00 after we have assigned December 02 as a holiday through administrator mode.