Lab 3: Abstract and Interface

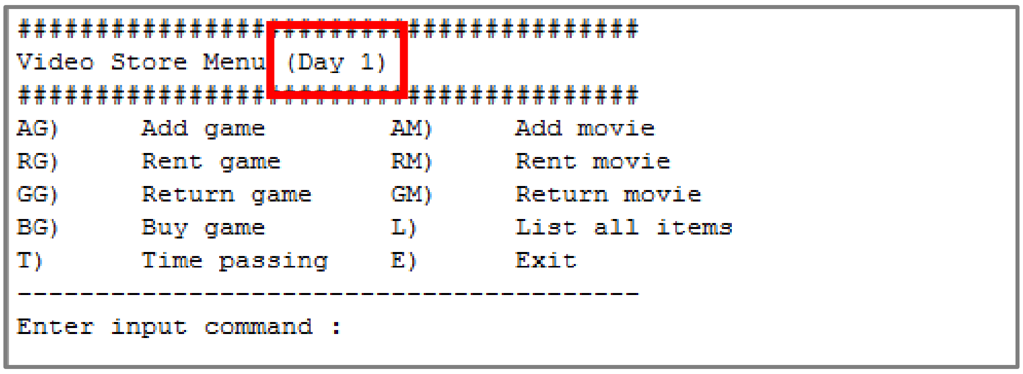
Instruction

1. Click the provided link on MyCourseVille to automatically create your own repository for this lab
2. Using a git command to clone this repository to your local computer
3. Open Eclipse and then "import existing project" from the cloned folder (in Step 2) into your Eclipse workspace
4. Rename an Eclipse Java project named "2110215_Lab3_2017_{ID}_{FIRSTNAME}" using this repository's root folder as the location of the project (Example: "2110215_Lab3_2017_5931234521_John")
5. Implement all classes and methods in package "model" and "main" following the details given
6. Check your codes with the provided JUnit Test Case
7. After implementing all classes, create a UML Diagram using "ObjectAid" and put the result image (UML.png) in your project folder
    - Interface Comparable should also be included in your generated class diagram
8. After finishing the program, export your project into a runnable jar file called "Lab3_2017_{ID}.jar"
9. Commit and push it to your GitHub repository

1 Problem Statement: Blockbuster Video Store

Blockbuster is a video rental store. There are games and movies provided for sale and rental. Write a “Point-Of-Sale” (POS) program that is used by staffs to operate daily tasks (inventory, rental service, sale, and utilities) for the store. Figure 1 shows the main menu of the program as follows:

<u>Inventory Service</u>	
• Option “AG”	- add game to the store
• Option “AM”	- add movie to the store
<u>Rental Service</u>	
• Option “RG”	- rent game
• Option “RM”	- rent movie
• Option “GG”	- return (give back) game
• Option “GM”	- return (give back) movie
<u>Sale Service</u>	
• Option “BG”	- sale game to customers *Note that the store does not sale movies.* *Also, only the games that have never been rented can sale.*
<u>Utilities Service</u>	
• Option “L”	- list all items in the store
• Option “T”	- simulate the store’s scenario by adding the number of passing days For example, if T=10, 10 days have passed and the current day is 11th day (Day 11), which is shown in the program title.
• Option “E”	- exit the program



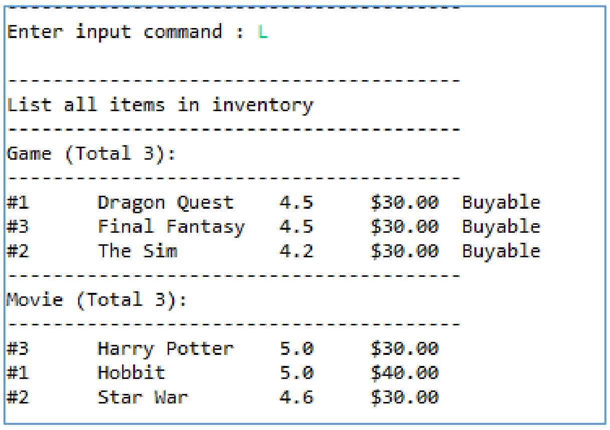
```
########################################
Video Store Menu (Day 1)
########################################
AG)     Add game        AM)     Add movie
RG)     Rent game       RM)     Rent movie
GG)     Return game     GM)     Return movie
BG)     Buy game        L)      List all items
T)      Time passing    E)      Exit
----------------------------------------
Enter input command :
```

Figure 1 Store's menus

1.1 Menu "L" (List Item)

When the program starts, we assume that there are 3 games and 3 movies available in the store. Figure 2a shows the list of initial items. Each record is displayed in the format below. For example, the first game record is "Dragon Quest" with "ID1" and user rating "4.5". Its price is $30, it can be sold to customers (Buyable), and there is no comment. Note that the lists of games and movies are separated, and they are ordered by (1) user rating and (2) name.

```
<ID>    <Name> <User Rating> <Sale Price>   <Status> <Comments on Rental/Sale Details>
```

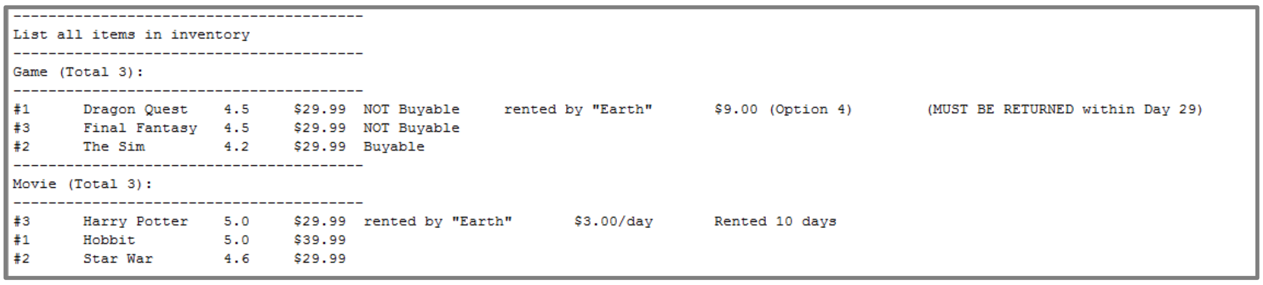


```
----------------------------------------
Enter input command : L

----------------------------------------
List all items in inventory
----------------------------------------
Game (Total 3):
----------------------------------------
#1      Dragon Quest    4.5     $30.00  Buyable
#3      Final Fantasy   4.5     $30.00  Buyable
#2      The Sim         4.2     $30.00  Buyable
----------------------------------------
Movie (Total 3):
----------------------------------------
#3      Harry Potter    5.0     $30.00
#1      Hobbit          5.0     $40.00
#2      Star War        4.6     $30.00
```

Figure 2a List menu

The item list in Figure 2b shows that Game "Dragon Quest" is already rented by Customer "Earth". The rental fee is $9 for 1 month rent (Option 4) and he must return the game within 29th day. More details about the rental option will be discussed later.



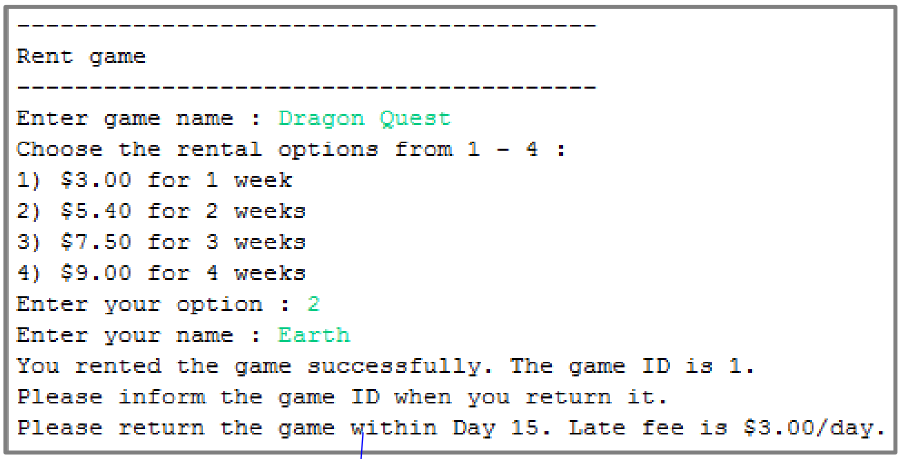
```
----------------------------------------
List all items in inventory
----------------------------------------
Game (Total 3):
----------------------------------------
#1      Dragon Quest    4.5     $29.99  NOT Buyable     rented by "Earth"       $9.00 (Option 4)        (MUST BE RETURNED within Day 29)
#3      Final Fantasy   4.5     $29.99  NOT Buyable
#2      The Sim         4.2     $29.99  Buyable
----------------------------------------
Movie (Total 3):
----------------------------------------
#3      Harry Potter    5.0     $29.99  rented by "Earth"       $3.00/day       Rented 10 days
#1      Hobbit          5.0     $39.99
#2      Star War        4.6     $29.99
```

Figure 2b List menu

1.2 Menu “RG” (Rent Game)

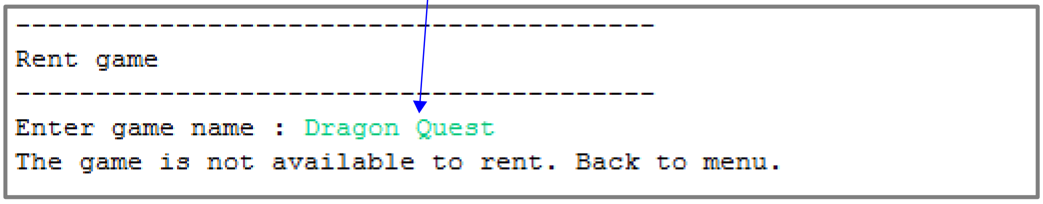
In this menu, there are 4 rental options: 1, 2, 3, and 4 weeks with different rental fees. An example scenario is shown below:

- Figure 3a shows that Game “Dragon Quest” is rented by Customer “Earth” for 2 weeks. Note that the late fee is 10% of the item price per day.
- Next, Figure 3b illustrates that customer cannot rent Game “Dragon Quest” anymore since it is already rented (not available).



```
----------------------------------------
Rent game
----------------------------------------
Enter game name : Dragon Quest
Choose the rental options from 1 - 4 :
1) $3.00 for 1 week
2) $5.40 for 2 weeks
3) $7.50 for 3 weeks
4) $9.00 for 4 weeks
Enter your option : 2
Enter your name : Earth
You rented the game successfully. The game ID is 1.
Please inform the game ID when you return it.
Please return the game within Day 15. Late fee is $3.00/day.
```

Figure 3a Rent game menu



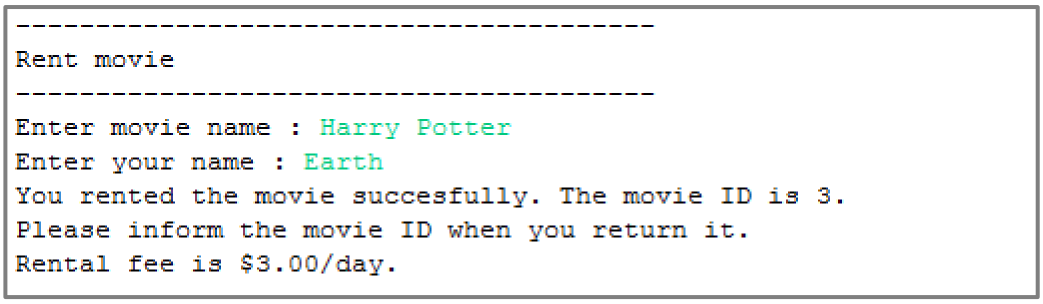
```
----------------------------------------
Rent game
----------------------------------------
Enter game name : Dragon Quest
The game is not available to rent. Back to menu.
```

Figure 3b Rent game menu (If the informed game isn’t available)

1.3 Menu “RM” (Rent Movie)

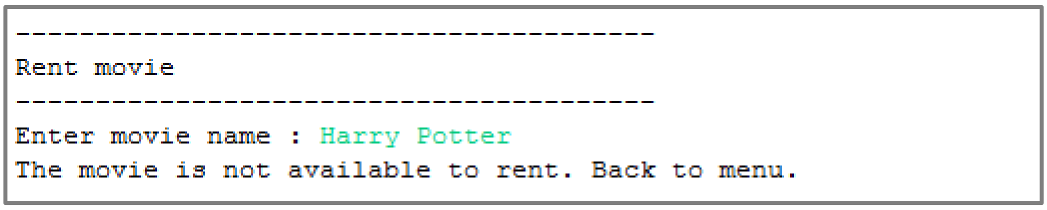
The rental menus for movie and game are different. In the movie menu, the rental fee is fixed at 10% of the item price per day. An example scenario is shown below:

- Figure 4a shows that Movie “Harry Potter” is rented by Customer “Earth”
- Next, Figure 4b illustrates that customer cannot rent Movie “Harry Potter” anymore since it is already rented (not available).



```
----------------------------------------
Rent movie
----------------------------------------
Enter movie name : Harry Potter
Enter your name : Earth
You rented the movie succesfully. The movie ID is 3.
Please inform the movie ID when you return it.
Rental fee is $3.00/day.
```

Figure 4a Rent movie menu

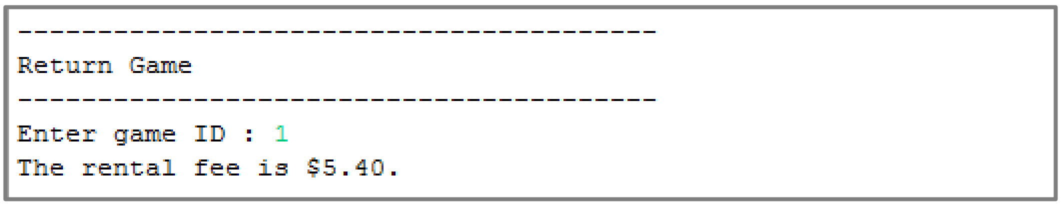


```
----------------------------------------
Rent movie
----------------------------------------
Enter movie name : Harry Potter
The movie is not available to rent. Back to menu.
```

Figure 4b Rent movie menu (If the informed movie isn't available)

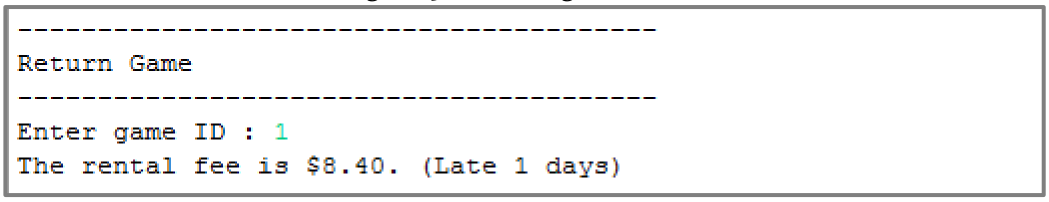
1.4 Menu "GG" (Give back Game - Return)

For the return menu, the program input is "ID" (not "Name") and the output shows the total fee including late fee. Figure 5a and 5b demo the result with/without late fee. Figure 5c shows the warning message when customer tries to return the item that has never been rented.



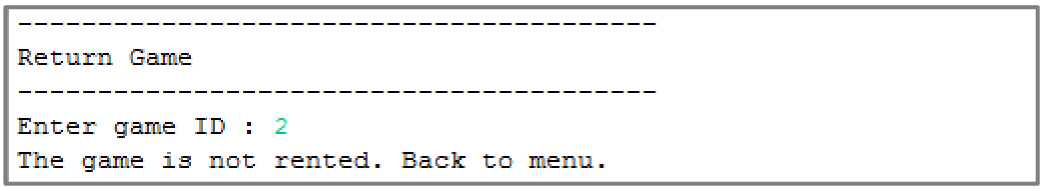
```
----------------------------------------
Return Game
----------------------------------------
Enter game ID : 1
The rental fee is $5.40.
```

Figure 5a Return game menu



```
----------------------------------------
Return Game
----------------------------------------
Enter game ID : 1
The rental fee is $8.40. (Late 1 days)
```

Figure 5b Return game menu (If the game is returned late)

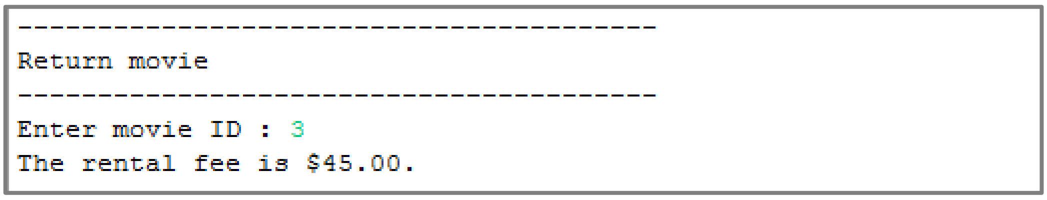


```
----------------------------------------
Return Game
----------------------------------------
Enter game ID : 2
The game is not rented. Back to menu.
```

Figure 5c Return game menu (If the game with the informed id isn't rented)

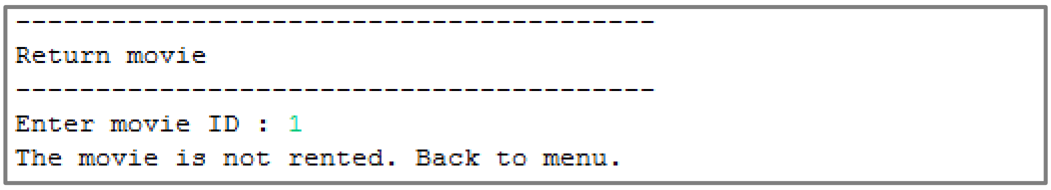
1.5 Menu "GM" (Give back Movie - Return)

It is similar to the return menu for game, except there is no late fee. Figure 6a demos the usual result and Figure 6b demonstrate the result when customer tries to return the item that has never been rented.



```
----------------------------------------
Return movie
----------------------------------------
Enter movie ID : 3
The rental fee is $45.00.
```

Figure 6a Return movie menu

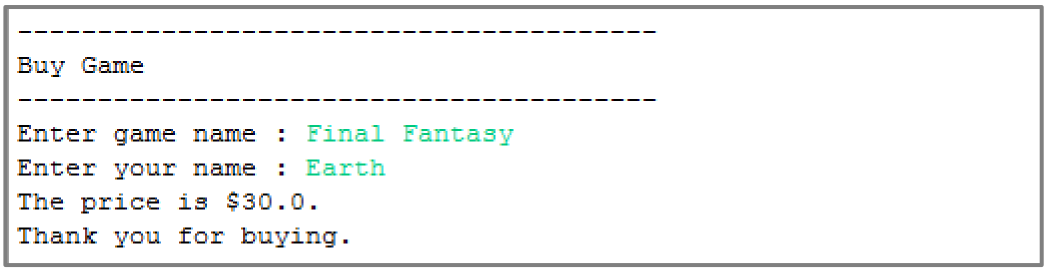


```
----------------------------------------
Return movie
----------------------------------------
Enter movie ID : 1
The movie is not rented. Back to menu.
```

Figure 6b Return movie menu (If the movie with the informed id isn't rented)

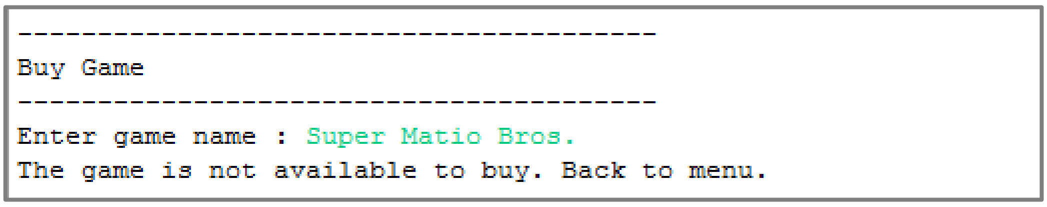
1.6 Menu "BG" (Buy Game)

Figure 7a demonstrates an example that Customer "Earth" bought Game "Final Fantasy". Also, the game that has ever been rented cannot be sold like in Figure 7b.



```
----------------------------------------
Buy Game
----------------------------------------
Enter game name : Final Fantasy
Enter your name : Earth
The price is $30.0.
Thank you for buying.
```

Figure 7a Buy game menu

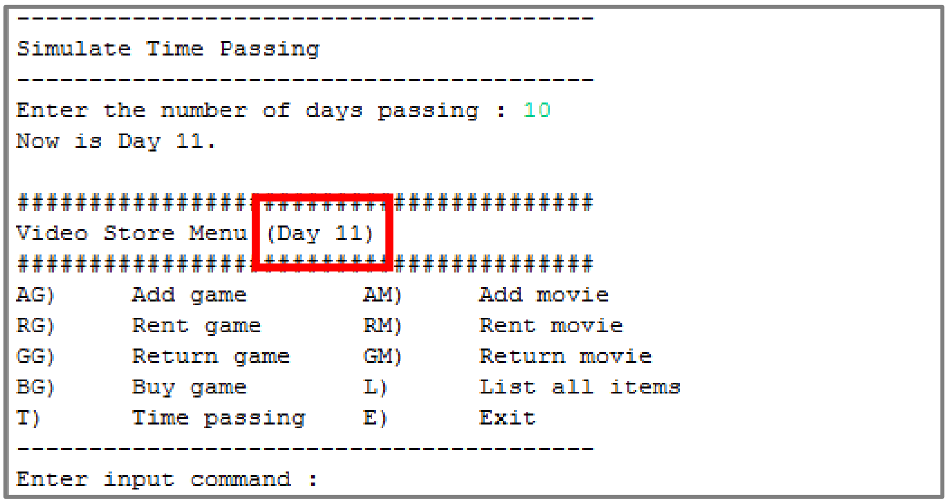


```
----------------------------------------
Buy Game
----------------------------------------
Enter game name : Super Matio Bros.
The game is not available to buy. Back to menu.
```

Figure 7b Buy game menu (If the informed game isn't available for sale)

1.7 Menu "T" (Time Passing)

This menu is used for simulate the store's scenario by adding the number of passing days. Figure 8 shows the case when T=10; this means that 10 days have passed and the current day is 11th day (Day 11), which is shown in the program title.



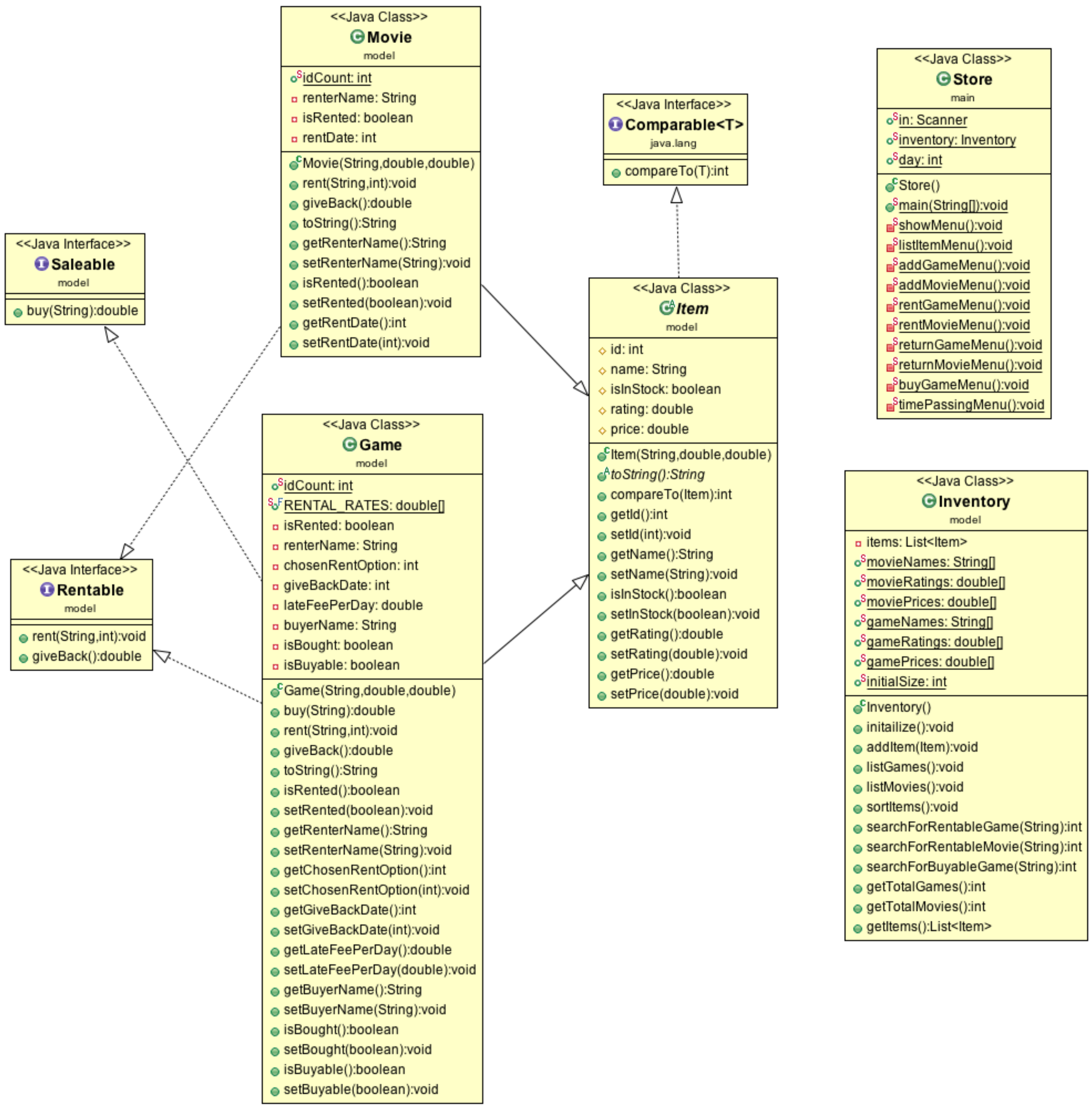
```
----------------------------------------
Simulate Time Passing
----------------------------------------
Enter the number of days passing : 10
Now is Day 11.

########################################
Video Store Menu (Day 11)
########################################
AG)     Add game        AM)     Add movie
RG)     Rent game       RM)     Rent movie
GG)     Return game     GM)     Return movie
BG)     Buy game        L)      List all items
T)      Time passing    E)      Exit
----------------------------------------
Enter input command :
```

Figure 8 Time passing menu

2 Implementation Detail

The diagram of the program is illustrated below. There are 5 classes: Item, Movie, Game, Inventory and Store. Also, there are 3 interfaces, where one is a Java interface (Comparable) and the rest are our custom interfaces (Saleable, Rentable).



2.1 Class *Item*

2.1.1 Field

# int id	Item's primary key. (The primary keys of game and movie are distinguish.)
# String name	Item's name.
# boolean isInStock	True if the item is in the inventory.
# double rating	Item's rating.
# double price	Item's price.

2.1.2 Constructor

+ Item(String name, double rating, double price)	Initialize all fields **<u>except id.</u>** (the default value for ***isInStock*** is **true**)

2.1.3 Method

+ *String toString()*	
+ int compareTo(Item o)	**If both items are the same kind**, compare two items based on their **<u>ratings</u>** and **<u>names</u>**. - Their ratings must be first considered descendingly. If they have the same **<u>ratings</u>**, their **<u>names</u>** are then compared lexicographically. **Otherwise, a game should always precedes a movie.** The result is: - **<u>negative integer</u>**, if this item precedes the given item. - **<u>postive integer</u>**, if this items follows the given game. - **<u>zero</u>**, if the items are equal.
+ Getter & Setter methods for every field	

2.2 Interface Rentable

2.2.1 Method

+ *void rent(String renterName, int rentOption)*	
+ *double giveBack()*	

2.3 Interface Saleable

2.3.1 Method

+ *double buy(String buyerName)*	

2.4 Class Game (Partially provided)

2.4.1 Field

<u>+ int idCount</u>	This field indicates the primary key of the new game in the inventory. (Starting from 1)
<u>+ double RENTAL_RATE[4]</u>	This variable is used for computing a game rental fee which composes of 4 options - 0.1 (10%) for 1 week - 0.18 (18%) for 2 weeks - 0.25 (25%) for 3 weeks - 0.30 (30%) for 4 weeks
- boolean isRented	True if the game is rented by any customer.
- String renterName	The name of customer who rented the game. **<u>If the game isn't rented by any customer, this string is empty.</u>**
- int chosenRentOption	The rental option customer chosen. There are 4 options available. - Option 1: Rents for one week. - Option 2: Rents for two weeks. - Option 3: Rents for 3 weeks. - Option 4: Rents for 4 weeks. **<u>If the game isn't rented by any customer, this field is zero.</u>**
- int giveBackDate	The date which the game should be returned within. **<u>If the game isn't rented by any customer, this field is zero.</u>**
- double lateFeePerDay	The fee customer would be charged if the game was returned late. (Per day) The fee is 10% of the game's price.
- String buyerName	The name of customer who bought the game. **<u>If the game isn't bought by any customer, this string is empty.</u>**
- boolean isBought	True if the game is bought by any customer.
- boolean isBuyable	True if the game has never been rented by any customer.

2.4.2 Constructor

+ Game(String name, double rating, double price)	**/*Fill Code*/** Initialize all fields, set *id* to the current value of *idCount* then increase *idCount* by one. Assume that when this game is initialized, it is not rented or bought by any customer **<u>Note: You should utilize its parent constructor.</u>**

2.4.3 Method

+ *String toString()*	
+ void rent(String renterName, int rentOption)	**/*Fill Code*/** To set all corresponding fields (*renterName*, *chosenRentOption*, *isInStock*, *isRented* and *giveBackDate*).
+ double giveBack()	**/*Fill Code*/** To set all corresponding fields (*renterName*, *chosenRentOption*, *isInStock*, *isRented* and *giveBackDate*) and return total rental fee (Late fee included).
+ double buy(String buyerName)	**/*Fill Code*/** To set all corresponding fields (*buyerName*, *isInStock*, *isBought* and *isBuyable*) and return the game's price.
+ Getter & Setter method for every field except *idCount* and *RENTAL_RATE*	

2.5 Class Movie (Partially provided)

2.5.1 Field

+ int idCount	This field indicates the primary key of the new movie in the inventory. (Starting from 1)
- boolean isRented	True if the movie is rented by any customer.
- String renterName	The name of customer who rented the movie. **If the movie isn't rented by any customer, this string is empty.**
- int rentDate	The date when the movie is rented by customer. **If the movie isn't rented by any customer, this field is zero.**

2.5.2 Constructor

+ Movie(String name, double rating, double price)	**/*Fill Code*/** Initialize all fields, set *id* to the current value of *idCount* then increase *idCount* by one. Assume that when this movie is initialized, it is not rented by any customer **Note: You should utilize its parent constructor.**

2.5.3 Method

+ *String toString()*	
+ void rent(String renterName, int rentOption)	**/*Fill Code*/** To set all corresponding fields (*renterName*, *isInStock*, *isRented* and *rentDate*). **Note: The rentOption is zero.**
+ double giveBack()	**/*Fill Code*/** To set all corresponding fields (*renterName*, *isInStock*, *isRented* and *rentDate*) and return total rental fee. **Note: If the movie is returned in the same day as it was rented, It would be counted as it has been rented for one day.**
+ Getter & Setter method for every field except *idCount*	

2.6 Class Inventory (Partially provided)

This class stores items (games and movies) that are available to customers for rent or sale.

2.6.1 Field

- List<Item> items	Stores all games and movies in the inventory.

2.6.2 Constructor

+ Inventory()	**/*Fill Code*/** Initializes *items* then call initialize()

2.6.3 Method

+ void initialize()	To generate and sort the initial games and movies in the inventory.
+ void addItem(Item item)	**/*Fill Code*/** To add the item into the inventory. **<u>Note: The item could be either a game or movie.</u>**
+ void listGames()	**/*Fill Code*/** To iterate and print all games in the inventory. **<u>Tip: If you directly print an object, toString() of the object would be called automatically.</u>**
+ void listMovies()	**/*Fill Code*/** To iterate and print all movies in the inventory. **<u>Tip: If you directly print an object, toString() of the object would be called automatically.</u>**
+ void sortItems()	**/*Fill Code*/** To sort all items in the inventory according to the detail of *compareTo()* of class Item **<u>Note: You should utilize *compareTo()*.</u>**
+ int searchForRentableGame (String gameName)	**/*Fill Code*/** To return the index of the game which has the same name as customer informed and is still in the inventory. **<u>If no game is available, return -1 instead.</u>**
+ int searchForRentableMovie (String movieName)	**/*Fill Code*/** To return the index of the movie which has the same name as customer informed and is still in the inventory. **<u>If no movie is available, return -1 instead.</u>**
+ int searchForBuyableGame (String gameName)	**/*Fill Code*/** To return the index of the game which has the same name as customer informed, is still in the in the inventory **<u>and has never been rented by any customer.</u>** **<u>If no game is available, return -1 instead.</u>**
+ int getTotalGames()	**/*Fill Code*/** To return the total number of the games in the inventory.
+ int getTotalMovies()	**/*Fill Code*/** To return the total number of the movies in the inventory.
+ Getter method for *items*	

2.7 Class Store (Partially provided)

It is the main class showing the program menu to user and calling method corresponding to user's input.

2.7.1 Field

+ Scanner in	Keyboard-input scanner.
+ Inventory inventory	Store's inventory.
+ int day	The current date. (Starting from Day 1)

2.7.2 Method

+ void main()	To show all available menus, get user input command and execute the command repeatedly until the program is terminated.
- void showMenu()	To show all menus. **(Figure 1)**
- void listItemMenu()	To list all games and movies in the inventory in descending order. **(Figure 2)**
- void addGameMenu()	To randomly add a game from the predetermined set of games.
- void addMovieMenu()	To randomly add a movie from the predetermined set of movies.
- void rentGameMenu()	**/*Fill Code*/** To ask for the game's name then ask for customer's name and process the game-rental procedure, if the game customer informed is available. **(Figure 3a)** **Note: The customer's name won't be asked if the game customer informed isn't available.** **(Figure 3b)**
- void rentMovieMenu()	**/*Fill Code*/** To ask for the movie's name then ask for customer's name and process the movie-rental procedure. **(Figure 4a)** **Note: The customer's name won't be asked if the movie customer informed isn't available.** **(Figure 4b)**
- void returnGameMenu()	**/*Fill Code*/** To ask for the game's id, process the game-return procedure and calculate the total rental fee (Late fee included). **(Figure 5a and 5b)** **Note: If the game with the informed id isn't rented, The returning would be unsuccessful.** **(Figure 5c)**
- void returnMovieMenu()	**/*Fill Code*/** To ask for the movie's id, process the movie-return procedure and calculate the total rental fee. **(Figure 6a)** **Note: If the movie with the informed id isn't rented, The returning would be unsuccessful.** **(Figure 6b)**

- void buyGameMenu()	**/*Fill Code*/** To ask for the game's name then ask for customer's name and process the game-buying procedure, if the game customer informed is available. **(Figure 7a)** **Note: The customer's name won't be asked if the game customer informed isn't available.** **(Figure 7b)**
- void timePassingMenu()	To simulate the time passing in the program. **(Figure 8)**