COSC1284 Programming Techniques
Semester 1 (2018)
Assignment 2

Due: **Friday, May 18th**
11:59 p.m. AEST (Midnight)
ie. End of week 11.

Late submissions accepted until
Wednesday, May 23rd
11:59 p.m. AEST (Midnight)

Assignment Type: **Individual (Group work is not permitted)**

Total Marks: **150 marks (15%)**

Backround information

For this assignment you need to write an object-oriented console application in the Java programming language which adheres to basic object-oriented programming principles shown below:

a) Your code should follow good object-oriented principles such as: encapsulation, composition and cohesion.

b) Setting the visibility of all instance variables to private.

c) Using getter and setter methods only where needed and appropriate with consideration given to scope (visibility).

d) Implementing appropriate validation. Any methods that return a double value should return Double.Nan if the operation fails.

e) Avoiding the use of static variables and methods.
Only using static attributes and methods when no other solution can be found.
You should not need to use the static keyword in this assignment.

f) Encapsulating both the data for a class and the methods that work with and/or manipulate that data within the same class.

g) Using superclass methods to retrieve and/or manipulate superclass properties from within subclass methods. Avoiding access variables directly.

h) Taking advantage of polymorphism wherever possible when invoking methods upon objects that have been created and avoiding unnecessary casting.

NOTE: The inappropriate use of the static keyword will incur a 10% marking penalty.

NOTE: Any changes made to this specification after its release, will be posted on the discussion forum.

You must make sure that you keep upto date with any posts in the forum to ensure you are aware of any changes.

Overview

For this assignment you need to write a console application in the Java programming language which implements a simulation for hiring items from a movie store called **MovieMaster**.

The staff at **MovieMaster** will need to be able to maintain the details for rental items that are available for hire by **MovieMaster** customers.

A movie can be either a **new release** movie (which can only be borrowed for up to **2 days** before needing to be returned) or a **weekly** movie (which can be borrowed for up to **7 days** before needing to be returned).

- Weekly movies have a base rate of **$3.00** to hire.
- New release movies have a base rate of **$5.00** to hire.

A user can **'borrow'** in advance by specifying the number of days in the future that they will collect/pick up the item.

If a rental movie is returned late then **a late fee of 50% of the hire fee for the movie is charged for each day that has passed after the initial loan period** for the movie in question.

Each movie will maintain its own collection of Hiring records maintained for the last 10 transactions.

MovieMaster staff will need to be able to perform the following tasks:

- Add new movies to the system
- Display a summary of the details for **ALL** rental movies.
- Record details for movies that have been borrowed by **MovieMaster** customers.
- Record details for movies that have been returned by **MovieMaster** customers.

In the later stages of this assignment additional types of rental items will be introduced, but you should focus on these intital requirements first.

You will be addressing these requirements by implementing a series of classes designed to meet these needs. The classes need to be designed according to object oriented principles.

Please note that for the purposes of this assessment you **DO NOT** need to create membership system or customers. The customer will be represented by a simple 3 character id that will be input at the time of borrowing.

You must use an array to store any collections of data in your program.

You are not permitted to use streams from Java 1.8.
You must not use ArrayLists, Maps or any other data structures within the Java API or from 3rd party libraries.

Failing to comply with these restrictions will incur a 25% marking penalty.

Disclaimer:

While the scenario described is based upon the concept of a movie rental store management system, the specification for this task is intended to represent simplified version of such a system and thus is not meant to be a 100% accurate simulation of the any movie rental store management system or service that is being used at a "real life" movie rental store.

Assignment 2 – Getting Started

This assignment is divided up into several stages, each of which corresponds to the concepts discussed during the course as shown below:

• Stage 1 - Movie class & HiringRecord class	(Classes)	(25 marks)
• Stage 2 - MovieMasterSystem	(Arrays of objects)	(25 marks)
• Stage 3 - Item class & Game class	(Inheritance / subclasses)	(25 marks)
• Stage 4 - MovieMasterSystem (**Updated**)	(Polymorphism)	(10 marks)
• Stage 5 - Exception Handling	(Exception Handling)	(25 marks)
• Stage 6 - Persistence to file	(File Handling)	(25 marks)
• Code Quality	(General Concepts)	(7 marks)
• Demo	(Scheduled Demonstration)	(8 marks)

The assignment is designed to increase in complexity with earlier stages being easier and more prescriptive and the later stages being more difficult and less prescriptive. The later stages of the assignment will require you to be more independent and resolve design issues on your own.

There is no Startup code for this assignment you are expected to write the entire program yourself.

Working with Dates

A single class called **DateTime** has been provided to help make working with dates easier than using the inbuilt Java classes.

Creating a DateTime for the current day.

```
DateTime date = new DateTime();
```

Creating a DateTime for a specific date.

```
DateTime date = new DateTime(day, month, year));
```

Setting a DateTime for a specific number of days in the future.

```
DateTime date = new DateTime(11));
```

Getting an Australian formatted date.

```
DateTime date = new DateTime());
date.getFormattedDate();
```

Getting a date in the format of 8 digits.

```
DateTime date = new DateTime(5));
date.getEightDigitDate();
```

Please review the **DateTime** class to see what other features are available (you might need them :-)).

Returning Values

In the specifcation, please note that **"returning"** a value does not mean printing a value to the screen.

The value should be returned to the code that calls the function and the calling code is responsible for the actual printing and/or processing of the string.

Formatting Strings

The **formatting** of strings can be achieved by using the format method on the String class which operates in the same way as the printf statement you learned about earlier in the course.

A simple example of how to use the format method of the string class:

```
                              PH1    PH2    value1     value2

String.format("%-25s %s\n", "ID:", MOS);
                          Spacing    New Line Character
```

This would produce:

```
ID:                       MOS
```

25 spaces (data left aligned)

Your task begins with the implementation of the **Movie** & **HiringRecord** classes.

It of course goes without saying that working your way through the learning materials for the corresponding weeks highlighted above is strongly recommended before tackling each of the stages in this assignment.

You are strongly encourage to review the course materials for the corresponding weeks before starting on the relevant stage, to ensure that you have sufficient grasp of the underlying concepts to proceed.

Do not leave it until the last minute to start the assignment, this is likely to take longer than you expect to complete.

Stage 1 of the Assignment begins on the next page ...

Stage 1 - HiringRecord (Design / Implementation) (10 marks)

Some tips below should help you to design and write the **Movie** & **HiringRecord** classes.

Some specific instructions are given below to get you started with the basics, but you will need to use the knowledge and skills you are developing in the course to further develop these classes and solve any problems you encounter as required.

You need to make sure that any modifications/changes you make adhere to good object oriented concepts and design.

After you write each class you should write some test code in the main method of your program to thoroughly test the class before moving on to the next section. Once your testing is complete you should remove any test code from your main method and move on to the next stage.

A) Create the HiringRecord Class (Supporting Class) (2.0 marks)

Each **Movie** will have a current hire record as well as a collection of the previous hiring records that includes up to the last ten hire transactions.

Because each **Movie** will have a **HiringRecord** you will need to create this class before creating the **Movie** class so that your **Movie** class can have an attribute that refers to these objects.

This **HiringRecord** class represents a single hiring record.

A **HiringRecord** is created at the time of the hire transaction.

The following attributes should be defined in your **HiringRecord** class.

You should not define any other instance variables unless there is a good reason to do so. You should think very carefully before adding any additional instance variables.

Instance variable	Type
id	String
rentalFee	double
lateFee	double
borrowDate	Date/Time
returnDate	Date/Time

B) Constructor (1.0 mark)

You should **NOT** have a no argument constructor. The constructor is responsible for initialising all the instance variables to appropriate values. **The borrow date** should be set to the date that the transaction is being made or an advance date if the user wishes to book the rental ahead of time.

The id for the hiring record should be a concatenation of the following three attributes:

id	+	memberId_	+	borrowDate (8 digit format)

For example:

id	+	memberId	+	borrowDate (8 digit format)
M_MOS_	+	HJC_	+	02032018

Example: M_MOS_HJC_02032018

C) Mutators & Accessors (setters & getters) (1.0 mark)

Simple accessors (getter methods) should **ONLY** be implemented if they are needed and used in your implementation. No simple setters should be implemented for any of the instance variables.

D) Implement a method for returning a movie (1.0 mark)

public double returnItem(**DateTime** returnDate, **double** lateFee)

This method will be responsible for completing all the necessary tasks that need to be performed when an item is returned including:

- validation (for example: late fee should not be less than zero)
- updating the state of all attributes related to this operation.

E) Implement a method for getting all the details of a Hiring Record (2.5 marks)

public String getDetails()

This method should build a string and **return** the string. The returned string should be formatted in a human readable form as shown below. This method **SHOULD NOT** do the actual printing.

The description must include labels and values for all the instance variables depending on whether it represents a hire that has yet to be returned or a hire that has been returned.

Sample output for a borrowing record that represents an item that **IS** currently on hire.

```
Hire Id:        M_MOS_RIC_12032018
Borrow Date:    12/03/2018
```

Sample output for a borrowing record that represents a movie that **HAS BEEN** returned.

```
Hire ID:        M_MOS_HJC_03032018
Borrow Date:    03/03/2018
Return Date:    11/03/2018
Fee:            $3.00
Late Fee:       $1.50
Total Fees:     $4.50
```

F) Implement a toString method (2.5 marks)

public String toString()

This method should build a string and **return** it to the calling method. The returned string should be formatted in a pre-defined format designed to be read and processed by a computer.

Format for the toString method is a colon seperated list of the instance variables as shown below:

id:borrowDate:returnDate:fee:lateFee

Sample output for a borrowing record that represents a movie the **IS** currently on hire.

```
M_MOS_RIC_03032018:03032018:none:none:none
```

Sample output for a borrowing record that represents a movie that **HAS BEEN** returned.

```
M_MOS_RIC_03032018:03032018:11032018:3.00:6.00
```

Stage 1 of the Assignment continues on the next page ...

Stage 1 - Movie (Design / Implementation) (15 marks)

Make a backup of your program before continuing

A) Create the Movie Class (2.0 marks)

The following attributes should be defined in your **Movie** class.

You should define other instance variables **ONLY** if necessary and where appropriate.

You should think very carefully before adding any additional instance variables to make sure that you are not creating additional variables that are unnecessary.

You should also create additional constants where appropriate avoiding the use of any hard coded numbers.

Instance variable	Type
id	String
title	String
genre	String
description	String
genre	String
isNewRelease	bool
currentlyBorrowed	HiringRecord
hireHistory	[HiringRecord]

Note: you must use an array

Constants	Type
NEW_RELEASE_SURCHARGE	double

B) Constructor (2.0 marks)

You should define a constructor in your **Movie** class.

You should **NOT** have a no argument constructor.

The constructor is responsible for initialising all the instance variables to appropriate values.

Think about what details will be provided by the user of the system and which details should be assigned default values.

The movie id should not be automatically generated, but the user of the system should supply this at the time the object is being constructed.

The id for the movie should be a concatenation of the following attributes:

M_ + id

For example:

prefix + id
M_ + MOS

Example: M_MOS

Stage 1 of the Assignment continues on the next page ...

C) Mutators & Accessors (setters & getters) (2.0 marks)

Simple accessors (getter methods) should **ONLY** be implemented if they are needed and used in your implementation.

No simple setters should be implemented for any of the instance variables.

D) Implement a method for borrowing a movie (2.5 marks)

public double borrow(String memberId)

This method should check for any pre-conditions to see if the **Movie** can be loaned such as checking to see if the movie is already on loan.

It should create a hire record and assign it to the currently borrowed attribute, then add that record to the historical hiring records for that particular **Movie**.

If there are more than 10 records it should delete the oldest one, insert the new one and maintain them in order of most recent.

The method should return **Double.*NaN*** if the **Movie** cannot be borrowed, otherwise it should return the borrowing fee.

E) Implement a method for returning a movie (2.5 marks)

public double returnItem(**DateTime** returnDate)

This method should check for any pre-conditions to see if the **Movie** can be returned such as attempting to return a movie on a date prior to the day on which it was borrowed or the movie is not currently on loan.

It should calculate any **late fee** that is applicable.

The late fee for a movie is calculated as follows:

- The late period is charged based on its rental period i.e. weekly or new release.
- Weekly rentals have 7 days.
- New release rentals have 2 days.
- The late fee for a movie is 50% of the rental fee for every day past the due date.

It should update both the currently borrowed attribute and the collection of historic hiring records.

The method should return **Double.*NaN*** if the **Movie** cannot be returned, otherwise it should return the total fee that needs to be paid.

Stage 1 of the Assignment continues on the next page ...

F) Implement a method for getting all the details of a Movie (2.0 marks)

public String getDetails()

This method should build a string and **return** it to the calling method. The returned string should be formatted in a human readable form. This method **SHOULD NOT** do the actual printing.

The description must include labels and values for all the instance variables depending on whether it represents a hire that has yet to be returned or a hire that has been returned.

The borrowing records in the history should be built such that the first record is the most recent.

Sample outputs demonstrating the structure and content required in the final summary for a **`Movie`** are shown below:

```
ID:                 M_MOS
Title:              Man of Steel
Genre:              Action
Description:        Clark Kent (Henry Cavill) must ...
Standard Fee:       $3.00
On loan:            YES
Movie Type:         Weekly
Rental Period:      7 days

                    BORROWING RECORD
                    ________________________________________
                    Hire ID:        M_MOS_RIC_12032018
                    Borrow Date:    12/03/2018

                    ________________________________________
                    Hire ID:        M_MOS_HJC_03032018
                    Borrow Date:    03/03/2018
                    Return Date:    11/03/2018
                    Fee:            $3.00
                    Late Fee:       $1.50
                    Total Fees:     $4.50

                    ________________________________________
```

NOTE:

You do not need to match the formatting shown here exactly.

As long as it is neat and easy to read.

You should at least have two columns with the left column for labels and the right column for the data.

You do not need to format the description into a paragraph style.

```
ID:                 M_JLG
Title:              Justice League
Genre:              Action
Description:        Fueled by his restored faith in humanity....
Standard Fee:       $5.00
On loan:            NO
Movie Type:         New Release
Rental Period:      2 days

                    BORROWING RECORD
                    ________________________________________

                    Hire ID:        M_JLG_RIC_24032018
                    Borrow Date:    24/03/2018
                    Return Date:    04/04/2018
                    Fee:            $5.00
                    Late Fee:       $22.50
                    Total Fees:     $27.50
```

Stage 1 of the Assignment continues on the next page ...

G) Implement a toString method (2.0 marks)

public String toString()

This method should build a string and **return** it to the calling method. The returned string should be formatted in a pre-defined format designed to be read and processed by a computer.

Format for the **toString** method is a colon seperated list of the instance variables as shown below:

id:title:description:genre:standard rental fee:type of hire:loan status

Sample output for a borrowing record that represents a movie that

IS currently on weekly hire

```
M_MOS:Man of Steel:Clark Kent (Henry Cavill) must save .... :Action:3.0:WK:Y
```

IS NOT currently on hire **AND IS** a New Release

```
M_MOS:Man of Steel:Clark Kent (Henry Cavill) must save .... :Action:5.0:NR:N
```

MAKE SURE YOU HAVE COMPLETED THIS STAGE BEFORE MOVING ONTO STAGE 2

Stage 2 of the Assignment begins on the next page ...

Stage 2 - MovieMasterSystem class (basic functionality) (25 marks)

Make a backup of your program before continuing

The attribute and functionality required for a **Movie** and **HiringRecord** have been modelled and encapsulated in the Movie class described above in Stage 1.

Here, you will begin with the implementation of the **MovieMasterSystem** application class, which will use an array of **Movie** references called '**items**' to store and manage movies that are added to the system by the user.

You must use an array to store the collection of your Movies.

You are not permitted to use streams from Java 1.8.
You are not permitted to use inbuilt or 3rd party collections such as ArrayLists or Maps.

Failing to comply with these restrictions will incur a 25% marking penalty.

1.) You should:

A) create a class called **'MovieMaster'**

B) this class should present a menu to the user for interacting with the program.

2.) The menu options that are presented to the user are:

```
*** Movie Master System Menu ***

Add Item                    A
Borrow Item                 B
Return Item                 C
Display details             D
Seed Data                   E
Exit Program                X
Enter selection:
```

You must implement the menu as shown. **Do not change the options** or the inputs for selecting the menu options.
(Note: the letters on the right represent what the user must type to use that feature.
In other words, to add a new Movie, the user must input 'A'.

The solution should be case insensitive, that is the user should be able to enter either ‘a’ or ‘A’)

Your task is to work on the implementation of this initial **MovieMasterSystem** class by implementing the functionality for the features shown in the menu. A description of the functionality that needs to be implemented for each of these features is provided below.

Stage 2 of the Assignment continues on the next page ...

A) Add Item (5 marks)

This first feature **'Add Item** should prompt the user to enter all relevant details for a **Movie**.

You should perform data validation to ensure that:

1. The id entered is three characters in length.
2. The id entered does not already exist in the system.

If either of these checks fail the user **should not** be prompted for the additional information and the movie object should not be created. You should display an appropriate error message to the console and the program should go back to the menu immediately without creating or storing a new **Movie** object in the array.

You are not permitted to implement auto generated user inputs such as id numbers, but must validate those entered by the user.

This means that you should perform this check prior to creating the **Movie** object and prior to asking the user to input all of the data for the new movie.

If the initial data validation passes, then once the user has entered the required details, the program should then check the input for the new release to ensure the user input is either "Y" or "N". If an incorrect input is received the user should be reprompted until they enter a correct value. The user can also choose to just hit the enter key without inputing a response if they wish to cancel the operation and return to the main menu.

Once all the data validation has passed, then the movie object should be created with the supplied data.

Example Inputs/Outputs

```
Example 1:

Enter id:                       MOS
Enter title:                    Man of Steel
Enter genre:                    Action
Enter description:              Clark Kent (Henry Cavill) must save ....
Enter new release (Y/N):        N


New movie added successfully for movie id: M_MOS.


Example 2:

Enter id: MOS
Error - Id for M_MOS already exists in the system!


Example 3:

Enter id: MO
Error - The Id 'MO' is invalid. Please enter a 3 digit id.
```

Stage 2 of the Assignment continues on the next page ...

```
Example 4:

Enter id:                                MOS
Enter title:                             Man of Steel
Enter genre:                             Action
Enter description:                       Clark Kent (Henry Cavill) must save
                                         ....
Enter new release (Y/N):                 Z
    Error: You must enter 'Y' or 'N'
Is New Release (Y/N)?:                   Z
    Error: You must enter 'Y' or 'N'
Is New Release (Y/N)?:
    Exiting to main menu.
```

After the object has been created it should be added to the next (empty) spot in the array instance that you are using to store the **Movie** objects.

B) Borrow Item (5 marks)

This feature should begin by prompting the user to enter the identification number of the movie which they wish to borrow.

The user only needs to enter the three digit id and not the prefix that identifies it as a movie.

Once the user has entered the identification number, the feature should then attempt to locate the corresponding **Movie** object within the array of **Movie** references described above.

If a matching **Movie** with the specified identification number was not found then a suitable error message should be displayed to the screen.

```
Example 1:

Enter id: MSS
Error - The item with id number: MSS, not found
```

If the movie is found but was not able to be borrowed then it should print out a suitable error message indicating the borrowing operation could not be completed.

```
Example 2:

Enter id: MOS
Error: The item with id M_MOS is currently on loan.
```

If the movie is found and is able to be borrowed, then it should prompt the user for a member id.

If the borrowing operation was completed, then the rental fee should be returned to the caller of the method and a suitable message printed to the console that contains both the fee payable and the due date.

```
Example 3:

Enter id:               MOS
Enter member id:        RIC
Advance borrow (days):  0
The item Man of Steel costs $3.00 and is due on: 05/04/2018
```

Stage 2 of the Assignment continues on the next page ...

C) Return Item (5 marks)

This feature should begin by prompting the user to enter the id number of the item which they wish to return. Once the user has entered the id number, the feature should then attempt to locate the corresponding **Movie** object within the array of **Movie** references described above.

If a matching **Movie** with the specified id number was not found then a suitable error message should be displayed to the screen.

```
Example:

Enter id: MSS
Error - The item with id: MSS, not found
```

If the item was was not able to be returned then it should print out a suitable error message indicating the return operation could not be completed.

```
Example:

Enter id: MOS
Error: The item with id: M_MOS is NOT currently on loan.
```

If the movie was found, then you should attempt to return the movie at a date in the future input by the user.

You can use the convienience method of the DateTime class to create a date a number of days in advance. Assume a movie variable holds a reference to a **Movie**

```
movie.returnMovie(new DateTime(5));
```

This will create a return date 5 days in the future. Note the value should be supplied by the user.

If the return operation was completed, then the fee payable should be returned by the method call and printed to the console.

```
Example 1:

Enter id:                       M_MOS
Enter number of days on loan:  5
The total fee payable is $0.00

Example 2:

Enter id:                       M_MOS
Enter number of days on loan:  11
The total fee payable is $6.00
```

Stage 2 of the Assignment continues on the next page ...

D) Display Movie Information (5 marks)

This feature should display the details to the console for all objects currently stored in the array of movie references described above, by using a loop to step through the array and calling the **getDetails()** method for each object in the array and printing the returned string to the console.

Example Outputs:

Not Borrowed (Weekly)

```
_____________________________________________________________________

ID:              M_MOS
Title:           Man of Steel
Genre:           Action
Description:     Clark Kent (Henry Cavill) must save ....
Standard Fee:    $3.00
On loan:         NO

Movie Type:      Weekly
Rental Period:   7 days

                 BORROWING RECORD
                 NONE
```

Not Borrowed (New Release)

```
_____________________________________________________________________

ID:              M_JLG
Title:           Justice League
Genre:           Action
Description:     Fueled by his restored faith in humanity and inspired by
                 Superman's selfless act ...
Standard Fee:    $5.00
On loan:         NO

Movie Type:      NEW RELEASE
Rental Period:   2 days

                 BORROWING RECORD
                 NONE
```

Borrowed (Weekly)

```
_____________________________________________________________________

ID:              M_MOS
Title:           Man of Steel
Genre:           Action
Description:     Clark Kent (Henry Cavill) must save ....
Standard Fee:    $3.00
On loan:         YES

Movie Type:      Weekly
Rental Period:   7 days

                 BORROWING RECORD
                 ___________________________________________

                 Hire ID:             M_MOS_RIC_05032018
                 Borrow Date:         05/03/2018
```

Stage 2 of the Assignment continues on the next page ...

Borrowed & Returned (Weekly)

```
_____________________________________________________________

Movie ID:         M_MOS
Title:            Man of Steel
Genre:            Action
Description:      Clark Kent (Henry Cavill) must save ....
Standard Fee:     $3.00
On loan:          NO

Movie Type:       Weekly
Rental Period:    7 days

                  BORROWING RECORD
                  _____________________________________________
                  Borrow ID:            M_MOS_HJC_03032018
                  Borrow Date:          03/03/2018
                  Return Date:          11/03/2018
                  Fee:                  $3.00
                  Late Fee:             $1.50
                  Total Fees:           $4.50
                  _____________________________________________
```

Borrowing History (Weekly)

```
_____________________________________________________________

Movie ID:         M_MOS
Title:            Man of Steel
Genre:            Action
Description:      Clark Kent (Henry Cavill) must save ....
Standard Fee:     $3.00
On loan:          YES

Movie Type:       Weekly
Rental Period:    7 days

                  BORROWING RECORD
                  _____________________________________________

                  Hire ID:              M_MOS_HJC_10042018
                  Borrow Date:          10/04/2018
                  _____________________________________________

                  Borrow ID:            M_MOS_RIC_29032018
                  Borrow Date:          29/03/2018
                  Return Date:          09/04/2018
                  Fee:                  $3.00
                  Late Fee:             $6.00
                  Total Fees:           $9.00
                  _____________________________________________
```

Stage 2 of the Assignment continues on the next page ...

E) Seed Data method (5 marks)

You should implement a method called '**seedData'** that pre-populates the movies collection with 10 sample movies. When the seed data menu item is selected 10 hard coded movies (**Movie**) will be instantiated and added to the collection.

WEEKLY MOVIES

- One weekly movie that **HAS NOT been borrowed**
- One weekly movie that **HAS BEEN been borrowed, but not returned**
- One weekly movie that **HAS BEEN** borrowed, and **HAS BEEN** returned 5 days later
- One weekly movie that **HAS BEEN** borrowed, and **HAS BEEN** returned 10 days later
- One weekly movie that **HAS BEEN** borrowed, and **HAS BEEN** returned 10 days later and then borrowed by another member, but not yet returned.

NEW RELEASE MOVIES

- One new release movie that **HAS NOT been borrowed**
- One new release movie that **HAS BEEN been borrowed, but not returned**
- One new release movie that **HAS BEEN** borrowed, and **HAS BEEN** returned 1 days later
- One new release movie that **HAS BEEN** borrowed, and **HAS BEEN** returned 3 days later
- One new release movie that **HAS BEEN** borrowed, and **HAS BEEN** returned 3 days later and then borrowed by another member, but not yet returned.

If this option is not selected at runtime, then the collection of movies should be empty.

If the collection of movies already has data then this method should not overwrite the current data and an error message should be displayed to the user.

MAKE SURE YOU HAVE COMPLETED THIS STAGE BEFORE MOVING ONTO STAGE 3

Stage 3 of the Assignment begins on the next page ...

Stage 3 - Item Heirarchy (Design / Implementation) (15 marks)

Make a backup of your program before continuing

This stage of the assignment is less prescriptive than the previous stages.

You will be required to think carefully about the concepts you have learned in the course and make decisions in regards to modifying the classes to meet the stated requirements.

This stage is **preparation** for implementing a new class called **Game**

When you have finished this stage, your program should operate exactly as it did before but you will have re-organised and re-written parts of the code.

DO NOT implement the Game class yet, you must complete the prepatory work in this stage before you implement the new **Game** class. Make sure you finish the Item class and test that your program is still working correctly before proceeding to create the **Game** class.

The **Game** and **Movie** class share common attributes and behaviours as they are both '**hirable**' items.

A **Movie is a 'hirable item'**

A **Game is a 'hirable item'**.

In object-oriented programming whenever we see an '**is a**' relationship. We model this relationship by implementing a class heirarchy.

Your first task in this section is to refactor your current **Movie** class into a class heirarchy before implementing an additional class for **Game**.

A) Create the Item class and modify the Movie class to be a sub-class of Item (6.0 marks)

Create a new class called **Item**

It should not be possible to instantiate this class as every item will have a different implementation of returning an item and some additional behaviours.

Modify your **Movie** class so that it is a sub-class of the new **Item** class.

Move any attributes that are currently in the **Movie** class that will be common to both **Game** and to the **Item** class. (*These have already been identified for you :-)*

Instance variable	Type
id	String
title	String
description	String
genre	String
fee	double
hireHistory	[HiringRecord]
currentlyBorrowed	HiringRecord

NOTE: 'New Release' has not been added to the common properties as some hireable types in the future will not have a new release property such as hiring a gaming console.

You will need to modify the code in your **Movie** class to resolve any compilation errors as a result of moving the above attributes to the **Item** class.

Stage 3 of the Assignment continues on the next page ...

You will need to make sure that you maintain good object-oriented design by keeping the scope of the attributes to the most restrictive scope you can.

You should implement appropriately scoped getters and setters ONLY where they are necessary.

B) Move the method for borrowing (3.0 marks)

You should now implement the **borrow** method in the **Item** class.

The **borrow** method in the **Item** class will perform any business logic that is common to all items.

C) Returning an item (3.0 marks)

You need to consider polymorphism here when deciding how to handle the returning of an item in this class.

D) Refactor the method for getting details and the toString method (3.0 marks)

You should now implement the **getDetails** method in the Item class and move any code from the **Movie** class into its super class version of this method.

After refactoring, you should have a **getDetails** method in both the **Item** and the **Movie** classes.

The **getDetails** method in the **Item** class will perform any business logic that is common to all items.

Think about the attributes that have been moved into the class to guide your decisions.

The **getDetails** method in the **Movie** class should make a call to its super class version and then perform any additional operations that are unique to **Movie**'s

You should also refactor the toString method in a similar fashion.

After refactoring your program should still be operating the same as it did at then end of the previous stage.

Do not continue with the design of the GAME class until you have fully tested your program to ensure it is still operating correctly after your refactoring work.

Stage 3 of the Assignment continues on the next page ...

Stage 3 - Game Class (Design / Implementation) (10 marks)

Make a backup of your program before continuing

A) Create the Game Class (1.0 mark)

The following attributes should be defined in your **Game** class. The extended property indicates that a user has chosen extended borrowing option at the time of borrowing. The extended borrowing feature provides discounted late fees when the item is returned.

You should also create new constants where appropriate. You can create additonal attributes if they are required.

Instance variable	Type
platforms	[String]
extended	boolean

Note: you must use an array

B) Constructor (2.0 marks)

You should define a constructor in your **Game** class. You should **NOT** have a no argument constructor. The constructor is responsible for initialising all the instance variables to appropriate values. Think about what details will be provided by the user of the system and which details should get default values.

The game id should not be automatically generated, but the user of the system should supply this at the time the object is being constructed.

C) Mutators & Accessors (setters & getters) (1.0 mark)

Simple accessors (getter methods) should **ONLY** be implemented if they are needed and used in your implementation. No simple setters should be implemented for any of the instance variables unless needed.

D) Implement a method for returning a game (3.0 marks)

public double returnItem(**DateTime** returnDate)

This method should check for any pre-conditions to see if the **Game** can be returned.

It should calculate the **late fee**. The late fee for a game is calculated as follows:

- The late fee for a game is a flat $1.00 fee for every day past the due date.
- An additional late fee of $5.00 for every 7 days past the due date is applied as a surcharge.
- A 50% discount on the fee is applied if the item had been enabled for extended hire.

It should also ensure that the state of the object is updated appropriately.

The method should return **Double.*NaN*** if the **Game** cannot be returned, otherwise it should return the total of the late fees paid.

Stage 3 of the Assignment continues on the next page ...

E) Implement a method for getting all the details of a Game (1.5 marks)

public String getDetails()

This method should build a string and **return** it to the calling method. The returned string should be formatted in a human readable form.

The description must include labels and values for all the instance variables depending on whether it represents a hire that has yet to be returned or a hire that has been returned.

The on loan description should now include the following possible values:

YES
NO
EXTENDED

The borrowing records in the history should be built such that the first record is the most recent.

Sample output demonstrating the structure and content required in the final summary for a **`Game`** is shown below:

Example Output:

```
ID:                     G_IGA
Title:                  Injustice Gods Among Us
Genre:                  Fighting
Description:            What if our greatest heroes became
                        our greatest threat? ...

Standard Rental Fee:    $20.00
Platforms:              XBox 360, PS4
Rental Period:          22 days
On loan:                EXTENDED

                        BORROWING RECORD
                        ________________________________
                        Borrow ID:   RIC02032018
                        Borrow Date: 24/04/2018

                        ________________________________
                        Borrow ID:   G_IGA_RIC_25032018
                        Borrow Date: 25/03/2018
                        Return Date: 23/04/2018
                        Fee:         $20.00
                        Late Fee:    $12.00
                        Total Fees:  $32.00

                        ________________________________
```

NOTE:

You do not need to match the formatting shown here exactly.

As long as it is neat and easy to read.

You should at least have two columns with the left column for labels and the right column for the data.

You do not need to format the description into a paragraph style.

Stage 3 of the Assignment continues on the next page ...

G) Implement a toString method (1.5 marks)

public String toString()

This method should build a string and **return** it to the calling method. The returned string should be formatted in a pre-defined format designed to be read and processed by a computer.

Format for the **toString** method is a colon seperated list of the instance variables as shown below:

id:title:description:genre:platforms:standard rental fee:loan status

Sample output for a borrowing record that represents a movie that

IS currently on hire

```
G_IGA:Injustice Gods Among Us: What if our greatest heroes became our
greatest threat?:Fighting:20.00: XBox 360, PS4:Y
```

Sample output for a borrowing record that represents a movie that

IS currently on extended hire

```
G_IGA:Injustice Gods Among Us: What if our greatest heroes became our
greatest threat?:Fighting:20.00: XBox 360, PS4:E
```

Sample output for a borrowing record that represents a movie that

IS NOT currently on hire

```
G_IGA:Injustice Gods Among Us: What if our greatest heroes became our
greatest threat?:Fighting:20.00: XBox 360, PS4:N
```

MAKE SURE YOU HAVE COMPLETED THIS STAGE BEFORE MOVING ONTO STAGE 4

Stage 4 of the Assignment begins on the next page ...

Stage 4 - MovieMasterSystem class (updated) (10 marks)

Make a backup of your program before continuing

The next stage of this task is to update the **MovieMaster** application class implementation described in stage 2 so that it incorporates the ability to work with **Games**.

A description of the functions that need to be implemented for each of these features is provided below.

A) Add Item (Updated) (5 marks)

The add item function of your menu is now being used for adding either a **Movie** or a **Game**.

Update your add function to now prompt the user to indicate if they wish to create a **Game** or a **Movie.** Based on the user's choice it will now prompt for the appropriate input to finish creating the correct type of object.

Example 1:

```
Enter id:                   IGA
Enter title:                Injustice Gods Among Us
Enter genre:                Fighting
Enter description:          What if our greatest heroes became our greatest
                            threat?
Movie or Game (M/G)?        G
Enter Game Platforms:       Xbox 360, PS4

New game added successfully for the game entitled: Injustice Gods Among Us
```

Example 2:

```
Enter id:                   MOS
Enter title:                Man of Steel
Enter genre:                Action
Enter description:          Clark Kent (Henry Cavill) must save ...
Movie or Game (M/G)?        M
Is New Release (Y/N)?:      N

New movie added successfully for the movie entitled: Man of Steel
```

Example 3:

```
Enter id:                   BVS
Enter title:                Batman vs Superman
Enter genre:                Action
Enter description:          Superheroes at odds, who will help us now ...
Movie or Game (M/G)?        M
Is New Release (Y/N)?:      Y

New movie added successfully for the movie entitled: Batman vs Superman
```

The object should be stored in the next available (empty) position in the same `Item` array described previously in Stage 2. In other words, both your Movie and Game objects should be stored in the same array.

Stage 4 of the Assignment continues on the next page ...

B) Borrow Item (Updated) (2.5 marks)

You will need to update this feature to be able to handle the new feature of a game that prompts the user for the option to choose extended hire.

C) Seed Data (2.5 marks)

Update your seed method so that it now pre-populates the collection with an additional 4 Games. The seeded data must contain the following variety of games.

GAMES

- One game that **HAS NOT been borrowed**
- One game that **HAS BEEN been borrowed, but not returned**
- One game that **HAS BEEN** borrowed, and **HAS BEEN** returned 19 days later
- One game that **HAS BEEN** borrowed, and **HAS BEEN** returned 32 days later

These dates should be 19 and 32 days at the time the program is being run/exectued, not from the day that you wrote the code :-).

If this option is not selected at runtime, then the collection of items should be empty.

If the collection of items already has data then, this method should not overwrite the current data and an error message should be displayed to the user.

MAKE SURE YOU HAVE COMPLETED THIS STAGE BEFORE MOVING ONTO STAGE 5

Stage 5 of the Assignment begins on the next page ...

Stage 5 - Adding exception handling (25 marks)

Make a backup of your program before continuing

It has been identified that there are potential issues with the booking process. Currently, these have been handled by returning Double.Nan to the user to indicate the presence of a problem.

It has been decided that a better solution is to generate an exception with a custom error message that will inform the user of the problem that has occurred.

You should comply with these requests by making the following changes to your program.

NOTE: When you implement exceptions into your program you will need to modify the other classes in your program to work with the exeptions. This will include removing any pre-checking from the MovieMaster system such as checking for invalid id's.

Exceptions will be generated and thrown in the **Item**, **Movie** and **Game** classes.

A) Creating the Exception Classes (IdException, BorrowException) (5 marks)

Define your own custom Exception subclasses type called **IdException, BorrowException** which represents various errors that can occur in your program
.

All of these custom exception types should allow an error message to be specified when the exception is created.

B) Generating Exceptions (10 marks)

You will need to look for areas in your program where you need to handle various errors such as :

- invalid id's (custom exception)
- borrowing errrors (custom exception)

Then you should determine where in your program **CAN** these errors occur and generate appropriate exceptions and then throw the exception back to the appropriate class to be handled.

C) Handling Exceptions (10 marks)

The various exceptions should then be allowed to propagate back to the appropriate class (ie. the exception **should not be caught locally** within the class that generated the exception).

Any exceptions will need to be caught and handled in an appropriate manner by displaying the error message contained within the various exception objects that have been propagated up from the relevant method call).

MAKE SURE YOU HAVE COMPLETED THIS STAGE BEFORE MOVING ONTO STAGE 6

Stage 6 of the Assignment begins on the next page ...

Stage 6 - Persistence (File I/O) (25 marks)

Make a backup of your program before continuing

This section of the program is designed to challenge and extend your ability to take a problem and workout how to solve it.

It is expected that **you will design your own solution** for this feature.

Your tutor can comment on your solution, but will not tell you how to write this section of the assignment.

Your program should incorporate file handling functionality so that it writes the details for each **Item** object currently in the **MovieMaster** System out to file when the program terminates.
(i.e. when the user selects the "Exit" option in the menu).

All **Item** data should be written out to and read in from (the same) text file (containing the details for both **Movie** and **Game** objects in the **MovieMaster** system).

The text files should be saved locally in your project directory.

The program should create two files with identical data. The first should be the main data file. The second should be a file with the same name appended with '_backup' to the file name.

The data that was previously written out to file should then be read back in automatically when the program is started up again and the **Item** information within should be used to create an appropriate set of **Movie** and **Game** objects, which should be stored in the array or collection of **Item** references described in Stage 4 above.

If the item data file is not found in the local folder then the program should:

1.) First check for the presence of a backup file and if found use this backup file for loading the data.

2.) If loading from the backup file, display a message indicating that the data was loaded from a backup file.

3.) If no backup file is found, display a message indicating that no **Item** data was loaded and continue on to the program menu without reconstructing any objects.

The format that you write vehicle data out in is entirely at your own discretion as long as it is done to a text file.

One aspect of this task is to record any changes that have been made during the previous run of the **MovieMaster** application, so your file handling functionality must be able to handle the writing out and reading in of all details for both types of **Items** in such a way that the state of the **MovieMaster** system at the point where the program was last exited is reconstructed in full when the program is started up again.

You can make any changes or include any additional methods that you deem necessary to the **Item**, **Movie**, and **Game** classes to facilitate the writing out and reading in of details for both **Item** types to satisfy this part of the specification.

Note that the program design described for the in stages 1 - 5 does not fully support what is required in this stage, so part of this task is identifying what aspects need to be considered when designing / implementing a solution.

Remember that you will still be required to adhere to basic object-oriented programming principles (eg. encapsulation, information hiding, etc) when designing your solution for this aspect of the program.

IMPORTANT NOTES:

These file reading / writing features are advanced functionality.

Marks will only be awarded for this section if the following conditions are met.

A) You must have completed the basic functionality of your program (Stages 1 – 5) before attempting this section.

B) Non-functional code or code which needs to be commented out in order to get the rest of the program to compile / run will receive zero marks in this section, as will code that just writes data out without any real intent of structuring that data to facilitate object persistence.

D) You should also use appropriate exception handling for any errors that are likely to occur when either reading a file or writing to a file.

C) Also note that **you are not permitted to use automatic serialisation, binary files or third party libraries** when implementing your file handling mechanism - you must use a **PrintWriter** for writing data out to a text file and a **BufferedReader** or **Scanner** when reading the data back in from the same text file that data was written out to previously.

A) Writing out to a file **(5 marks)**

Writing out the object state of each object in the **Item** array.
This part does not require writing out of the borrowing history.

B) Reading in from a file **(5 marks)**

Reading in the object state of each object represented in the file and restoration into the **Item** array.
This part does not require reading and restoration of the borrowing history.

C) Writing out of borrowing history **(5 marks)**

Writing out of the full borrowing history for each object such that it is associated with its parent object.

D) Restoration of borrowing history **(5 marks)**

Reading in and restoration of the full borrowing history for each object.

E) Maintainance of good object oriented principles **(5 marks)**

The code has maintained good object oriented practices of cohesion, encapsulation, maintainability and readability.

Coding style requirements begin on the next page ...

Coding Style (8 marks)

Your program should demonstrate appropriate coding style, which includes:

- **Formatting**
  Indentation levels of 3 or 4 spaces used to indent or a single tab provided the tab space is not too large - you can set up your IDE/editor to automatically replace tabs with levels of 3 or 4 spaces.

  A new level of indentation added for each new class/method/ control structure used.

  Indentation should return to the previous level of at the end of a class/method/control structure (before the closing brace if one is being used). Going back to the previous level of indentation at the end of a class/method/control structure (before the closing brace if one is being used)

  Block braces should be aligned and positioned consistently in relation to the method/control structure they are opening/closing.

  Lines of code not exceeding 80 characters in length - lines which will exceed this limit are split into two or more segments where required (this is a guideline - it's ok to stray beyond this by a small amount occasionally, but try to avoid doing so by more than 5-6 characters or doing so on a consistent basis).

  Expressions are well spaced out and source is spaced out into logically related segments

- **Good Coding Conventions**

  Identifiers themselves should be meaningful without being overly explicit (long) – you should avoid using abbreviations in identifiers as much as possible (an exception to this rule is a “generic” loop counter used in a for-loop).

  Use of appropriate identifiers wherever possible to improve code readability.

  All identifiers should adhere to the naming conventions discussed in the course notes such as ‘camel case’ for variables, and all upper case for constants etc.

  Complex expressions are assigned to meaningful variable names prior to being used to enhance code readability and debugging.

  Code blocks have been refactored into methods to improve readability and demonstrate a cohesive approach to method design.

- **Commenting**
  Note: the examples provided below do not reflect actual code you should have in your assignment, but are used for demonstration purposes only in regards to best practice commenting.

  Class Level commenting
  Each file in your program should have a comment at the top listing your name, student number and a brief (1-2 line) description of the contents of the file (generally the purpose of the class you have implemented).

```
/*
 * Class:        HiringRecord
 * Description:  The class represents a single hiring record for
 *               any type of item that can be hired.
 * Author:       [student name] - [student number]
 */
```

Method Level commenting

At least one method in each class should have an algorithm written in psuedocode as a multi-line comment. You don't need to specify test cases for every algorithm, only if it is appropriate.

```
/*
 * ALGORITHM
 * BEGIN
 *      GET age of dog
 *      COMPUTE human years equivalent
 *          IF dog is two years old or younger
 *              COMPUTE human years is dog's age multiplied by eleven
 *          ELSE
 *              COMPUTE human years is equal to age minus two then multiplied by five
 *              COMPUTER human years is equal to human years plus twenty two
 *      DISPLAY age
 * END
 *
 * TEST
 *      AGE is equal to 0, computation should not be performed
 *      AGE is equal to 1, human years is equal to 11
 *      AGE is equal to 2, human years is equal to 22
 *      AGE is equal to 3, human years is equal to 27
 */
```

Code block commenting

At least one method in each class should provide in line commenting to explain the purpose of a series of multiple statements, or a code block such as a loop or branching statement.

```
/* calculate the total value of the room rental
 * prior to applying surcharge.*/
int standardRate = 200;
int numberOfNights = 4;
int totalBeforeSurcharge = standardRate * numberOfNights;
```

Commented out code

Any incomplete or non functional code should be removed from your implementation prior to your final submission.

Code that is not being used in your final implementation that is left in the submission but just commented out will attract marking penalties.

Examples of bad commenting

An example of a bad comment and/or coding style:

```
// declare an int value for storing the basic nightly rate for a
room
int standardRoomRate = 200;

// declare and assign an initial account balance.
double x = 500.0;
```

Submission instructions/requirements on the next page ...

What to Submit

You are being assessed on your ability to write a program in an object-oriented manner in this assignment. Writing the program in a procedural style is not permitted and will be considered inadmissable receiving a zero grade.

You should stick to using the standard Java API (version 1.8) when implementing your program. Note the restrictions noted earlier in this document in relation to the Java Collection Framework, Streams and the use of 3rd party libraries.

You should export your entire eclipse project to a zip archive and submit the resulting zip file - do this from within eclipse while it is running, not by trying to copy or move files around in the eclipse workspace directly, as you may corrupt your entire workspace if you do something wrong.

All students are also advised to check the contents of their zip files by opening them and viewing the files contained within before submitting to make sure they have done it correctly and that the correct (latest) version of the source code file is present to avoid any unpleasant surprises later on.

You should also download a copy of your submission from Canvas after submission, so that you can double check that you have submitted the correct version.

Technical issues that result in the loss of your work or submitting an incorrect version **WILL NOT** be considered a basis for extensions or re-marking.

We are obliged to accept each submission in the form it is sent to us in, so make sure you submit the corrrect final version of your program!

The name of both your Eclipse project and your zip archive should be your student number followed by an underscore, then append A2. For example:

Project Name: s123456_A2
Zip Archive: s123456_A2.zip

Your submission will be penalised heavily if you go against any of the above guidelines.

Submission details

This assignment will be marked out of a total of **150 marks** and contributes **15%** towards your final result for this course.

Please make sure that you are familiar with both the contents of this specification document and the marking rubric. They both contribute to the requirements for this assessment and are meant to compliment each other. You should not rely on just one of them for understanding the assessible requirements.

Due Date

This due date of the assignment is listed on the first page of this specification. You must submit your final version to Canvas prior to this date/time to avoid incurring late penalties.

Late Submission Period

There is a late submission period of 5 days for this assignment.

Late submissions that are received before the end of this late submission period will attract a late penalty of 10% per day (or part thereof) of the total marks awarded on the assignment, unless an extension has been granted by the school or as part of an existing EAA provision (see below for details).

Submissions that are received after the late submission period expires will not be assessed unless some prior arrangement has been organised with the head tutor or lecturer in response to a request for an extension.

Extensions (General)

If you are unable to submit by the due date you **ARE REQUIRED** to provide additional documentation such as a medical certificate as per university policies.

Extensions **WILL NOT BE GRANTED** without supporting documentation.

The details of the extension requirements and an outline of the policies and procedures can be found here:

http://www1.rmit.edu.au/students/assessment/extension

Upon application for an extension you should continue to work on your asssessment and submit your latest version by the original due date specified at the top of this document whilst you await the outcome of your application.

If you have not received an outcome by the due date you should continue to work on your submission until you have received the outcome.

Extensions (Up to 7 Days)

Extensions can only be granted up to a maximum of 7 days provided the correct form and documentation has been submitted.

If you require an extension you must apply prior to the due date by filling out an 'extension of time' request and attaching the requisite documentation.

As outlined here:

http://www1.rmit.edu.au/students/assessment/extension

The form and documentation should be e-mailed to rodneyian.cocker@rmit.edu.au.

Extensions (Greater than 7 Days)

For extensions of greater than 7 days or for those with EAA provisions organised by the DLU:

http://www.rmit.edu.au/students/specialconsideration

If you are a student who has EAA provisions organised by the DLU then you must negotiate any extension you may require with the instructor well in advance of the submission deadline (at least 72 hours in advance of the on-time submission deadline would be ideal).

All questions regarding this specification should be directed to the Assignment 2 Specification forum on Canvas.

Any requests for assistance regarding implementing the requirements set out in this assignment or problems you are having with your program should be directed to the Assignment 2 Help / Discussion forum on Canvas.