RMIT UNIVERSITY

School of Science

COSC1295 Advanced Programming, S1, 2020

Assignment Part 1: Console Implementation

	Assessment Type: **Individual assignment; no group work**. Submit online via Canvas → Assignments → Assignment 1. Marks are awarded for meeting requirements as closely as possible according to assignment specifications and the supplied rubric. Clarifications/updates may be made via announcements/relevant discussion forums.
	Due date: **end of Week 5 11:59pm Sunday 5th April 2020**. Late submissions are handled as per usual RMIT regulations - 10% deduction (10 marks) per day. You are only allowed to have 5 late days maximum unless special consideration has been granted.
	Weighting: 100 marks (15% of your final semester grade)

1. Overview

NOTE: Carefully read this document. In addition, regularly follow the Canvas assignment discussion board for assignment related clarifications and discussion.

This assignment requires you to design and implement an object-oriented console application using the Java programming language. The application is named **UniLink**, which allows all students in a university to create and publish various types of posts and reply to those posts. In this assignment, UniLink supports three types of posts as shown below:

- **Event**: the post creator specifies information about an upcoming event. Other students can see that post in the UniLink system and reply to the post to attend that event.
- **Sale**: when a student has an item to sell, that student can create a sale post in the system and provide necessary details about the item. The post is visible to other students and each of them can reply with a price they want to pay for the item. More sale rules are described in the Sale class specifications below.
- **Job**: this is another type of post to enable a student to describe a job which the student needs to hire someone to do and the price at which the student expects to pay for that job. Other students can see the job post in the system and reply with different prices they want to get paid to do the job. More requirements about job posts are described in the Job class specifications below.

Users of the UniLink system can also view all the posts, manage their own posts, review all replies to their posts, close and delete their own posts.

See the next sections for more information about system features and implementation requirements.

Disclaimer: *the specification in this assignment is intended to represent a simplified version of a system in real life and thus is not meant to be a 100% accurate simulation of any real system or service that is being used commercially.*

2. Assessment Criteria

This assessment will determine your ability to implement Object-Oriented Java code according to the specifications shown below. In addition to functional correctness (i.e. getting your code to work) you will also be assessed on code quality. Specifically:

- You should aim to provide high cohesion and low coupling.
- You should aim for maximum encapsulation and information hiding.
- You should rigorously avoid code duplication.
- You should use superclass methods to retrieve and/or manipulate superclass properties from within subclass methods where necessary.
- You should take advantage of polymorphism wherever possible when invoking methods upon objects that have been created.
- You should comment important sections of your code remembering that clear and readily comprehensible code is preferable to a comment.

3. Learning Outcomes

This assessment is relevant to the following Learning Outcomes:

CL01: use the Java programming language in the implementation of small to medium sized application programs that illustrate professionally acceptable coding and performance standards.

CL02: demonstrate knowledge of the basic principles of the object-oriented development process and apply this understanding to the analysis and design of solutions for small to medium scale problems.

CLO3: describe and apply basic algorithms and data structures, in particular simple searching and sorting of data stored in data structures and manipulating data.

4. Assessment details

The UniLink system allows students to create and publish many types of posts. All those posts share a set of common attributes and methods and also have their own different sets of attributes and behaviors. The following sections describe the requirements of the Post superclass and its subclasses that you must implement in your code.

RMIT UNIVERSITY

Post class

Note:

I've created several videos as examples to make it easier to understand and implement the assignment requirements.
https://drive.google.com/drive/folders/1cRbdnPdKtQT_4wBqd8ySwc2GKSLPLJCk?usp=sharing
All videos are in the folder shown above. You should watch the LogInLogOut first, then Create Event/Sale/Job post, then Respond to Event/Sale/Job post.

Attributes

Each post object has the following attributes:

- **Id**: a string which uniquely identifies each post. The UniLink system must generate a unique id for each newly created post according to the following format:
  - An event post id begins with the string "`EVE`" followed by a three-digit, zero-padding number. The first event post has id `EVE001`, the subsequent event posts have id `EVE002`, `EVE003`, `EVE004`, `EVE005`…
  - A sale post id begins with the string "`SAL`" followed by a three-digit, zero-padding number. The first sale post has id `SAL001`, the subsequent sale posts have ids `SAL002`, `SAL003`, `SAL004`, `SAL005`…
  - A job post id begins with the string "`JOB`" followed by a three-digit, zero-padding number. For example, the first job post has id `JOB001`, the subsequent job posts have id `JOB002`, `JOB003`, `JOB004`, `JOB005`…
- **Title**: a short string to summarise the post. It can be a name of an event in an event post, or name of an item in a sale post or name of a job in a job post.
- **Description**: a longer string which provides more information about the post. It can be a description of an event, or an item on sale or a job specification.
- **Creator id**: a string which is the id of the student who created the post in the UniLink system. Student id begins with the character 's' followed by a number, for example, `s1`, `s2`, `s3`… are all acceptable student id in this assignment.
- **Status**: indicates whether the post is open to receive replies or closed and no longer accepts replies. In your implementation, you must make sure that a post status can only be either `OPEN` or `CLOSED`.
- **Replies**: a data structure of your choice, for example an array or array list, to store all replies to a post. Each reply is an object of the Reply class. More details about this class are shown in the "Reply class" section further down this document.

Constructors

At least one constructor to initialize all attributes shown above. By default, a post status is set to `OPEN` when a post is created.

Your implementation should not allow constructors in the Post class to be used directly to create Post objects. Instead, these constructors should only be called in constructors of subclasses of the Post

class, such as the Event, Sale and Job classes (more details below). Remember that one of our objectives is to reduce code duplication, especially when initializing common attributes of all post objects.

Methods

```
public String getPostDetails()
```

This method should build and return a string which contains information about a post. In this Post class, which is a superclass, the getPostDetails method only build and return a string containing the post id, title, description, creator id and status, which are common attributes of all types of post as shown above. This method is overridden in subclasses of the Post class to add more specific details relevant to each type of post, which will be described in the next sections.

The returned string should be formatted in a human readable form as shown in the examples below.

Example: calling the method getPostDetails() on an event post object will return the following String:

```
ID:             EVE001
Title:          Movie on Sunday
Description:    Join us to see a movie this
                Sunday with ticket
                discounts applied to all
                group members!
Creator ID:     s3456223
Status:         OPEN

Venue:          Village Cinema
Date:           29/03/2020
Capacity:       3
Attendees:      0
```

Note: details about event id, title, description, creator id and status are obtained by calling the getPostDetails() of the Post class.

The getPostDetails() method must be overridden in the Event subclass of the Post class to display more specific details about the event such as venue, date, capacity and the current number of attendees. More in the Event class specifications below.

Abstract methods

```
public abstract boolean handleReply(Reply reply)
```

Each type of post has a different way to handle a reply, therefore in this Post class, which is the superclass, this method is set to abstract and will be implemented in concrete subclasses of the Post class, i.e. in Event, Sale and Job classes

This method receives a reference to a Reply object, which store all details of a reply to this post. Specifications of the Reply class are shown in the "Reply class" section further down. This method returns true to indicate that the reply is processed and added to the collection of replies. The method returns false otherwise. More details about the implementation of this method are described in subclasses of the Post class in the sections further down this document.

```
public abstract String getReplyDetails()
```

This method should only be called by the post creator on their own posts. It returns a String containing details of all replies to that post. Each type of post has a different way to display reply details to the post creator, therefore this method must be set to abstract in this Post class and overridden in subclasses of the Post class. More details about how this method should be implemented will be shown in later sections.

Accessor and mutator methods

Simple accessor and mutator methods must be implemented in this Post class. You should only implement necessary accessor and mutator methods according to the Post class attributes shown above. There must be no redundant accessor and mutator methods in your code.

Reply class

When a student replies to a post, the following information is stored in an object of the Reply class:

- **Post id**: the id of the post which the reply is associated with.
- **Value**: a decimal number. For the sake of simplicity, in this assignment, a responder needs to enter a positive value as part of a reply to a post. Specifically, your program should require a responder to enter value 1 to join an event in an event post, or a price he or she is willing to pay for an item in a sale post or an amount of money he or she wants to be paid to complete the job advertised in a job post. See also the sale rules and job rules in the sections below.
- **Responder id**: the id of the student who replies.

You're required to implement at least one constructor to initialize all the attributes shown above when a reply object is created.

You also need to implement simple accessor and mutator methods according to the attributes shown above.

Event class

A student participated in the UniLink system can create an event post to let other students know about an upcoming event. An event post is an object of the Event class, which is a concrete subclass of the abstract Post class. In addition to common information of a post such as id, title, description, creator id, status, replies as shown in the Post superclass, an event post object also stores the following information:

- **Venue**: a value of type String which is the location of the event
- **Date**: the date when the event happens. For simplicity, no need to store the event time. It is sufficient to use a value of type String to store a date in the format `dd/mm/yyyy`, for example `20/03/2020`.
- **Capacity**: an integer which is the maximum number of attendees of the event
- **Attendee count**: the current number of participants in the event. A newly created event has zero attendee count. When a student selects to join the event, attendee count increments by one. When the attendee count is equal to event capacity, the UniLink system will automatically close the event, i.e. the event status set to `CLOSED`. A closed event no longer accepts participants.

Constructors

At least one constructor to initialize all attributes shown above. When a new event post is created, the creator will enter an event name, description, venue, date, capacity. The UniLink system will generate a new unique event post id and set that id and creator id to the newly created post object.

To reduce code duplication, your implementation must make use of the constructors created in the Post superclass.

Methods

`public String getPostDetails()`

You must override this method, which is inherited from the superclass Post, to return a String containing all common details of a post as shown in the Post class, and additional details of an event post, i.e. venue, date, capacity and attendees according to the following example:

Example: output of a `getPostDetails()` method call on an event post object

```
ID:             EVE001
Title:          Programming Study Group Wednesday night
Description:    Let's meet this Wednesday afternoon to finish
                the programming assignment!
Creator ID:     s3477223
Status:         OPEN
Venue:          RMIT Library
Date:           15/04/2025
Capacity:       5
Attendees:      0
```

`public boolean handleReply(Reply reply)`

Your Event class must implement this abstract handleReply method shown previously in the Post class to handle a reply to an event post. When a student replies to an event post to join an event, details of that reply are stored in a Reply object passed to this method (see "Reply class" section). If the reply is valid, the event is not full and the student id is not yet recorded in that event, then this handleReply method must add this reply object to the replies collection of this event post and return true, otherwise the reply will not be added to the replies collection and a

false value is returned. When the event is full, i.e when the number of attendees is equal to capacity, event status is set to CLOSED.

```
public String getReplyDetails()
```

As shown in the Post class, this method should only be called by the post creator on their own posts. It returns a String containing details of all replies to that post. When calling this method on an event post, this method must return a comma-separated string of the ids of students who have sent replies to join the event and have been accepted by the system as attendees, according to the following format:

`Attendee list:` *[comma-separated string of attendee id values]*

If an event has no attendee, then your method should return the following string:

```
Attendee list: Empty
```

Example: when an event post creator chooses to view his own events, in addition to details of the posts as shown in the getPostDetails method requirements, the getReplyDetails method will also be called to return a string as shown below (in each event post, notice the status, capacity, attendees and attendee list fields below):

```
ID:             EVE001
Title:          Programming Study Group Wednesday night
Description:    Let's meet this Wednesday afternoon to
                finish the programming assignment
Owner ID:       s3477223
Status:         OPEN
Venue:          RMIT Library
Date:           15/04/2020
Capacity:       5
Attendees:      0
Attendee list:  Empty
---------------------------

Title:          Movie on Sunday
Description:    See a movie this Sunday with a ticket
                discount applied to all group members.
Owner ID:       s3477223
Status:         CLOSED
Venue:          Village Cinema
Date:           26/03/2020
Capacity:       4
Attendees:      4
Attendee list:  s3488456,s3599643,s3737458,s3483921
```

Accessor and mutator methods

Simple accessor and mutator methods must be implemented in this Event class. You should only implement necessary accessor and mutator methods according to the additional attributes shown above. There must be no redundant accessor and mutator methods in your code.

Sale class

When a student has an item to sell, he or she can create a sale post in the UniLink system to let other students know about the features of the item. Other students who are interested in that item can reply with an offer price they want to pay for the item.

In addition to common information of a post such as id, title, description, creator id, status and a replies collection as described in the Post superclass, a sale post object also stores the following information:

- **Asking price**: the price at which the post creator wants to sell the item. This asking price is not visible to other students (see the Sale post rules section below for more details).
- **Highest offer**: the highest price offered among all replies from students interested to buy the item.
- **Minimum raise**: the minimum amount by which a new offer price in a reply must be higher than the highest offer recorded in this sale post (see Sale post rules section below for more details).

Sale post rules

There is a special feature about this type of sale. Specifically, the asking price is not visible to students viewing the sale post. When a student views a sale post which is opened, he or she can only see the item name, description and the highest price offered at that point, and then chooses whether to reply with an offer price.

The UniLink system only accepts a reply to a sale post when the post is opened (i.e. post status is OPEN) and will process the new offer price in a reply according to the following rules:

- If the new offer price is less than or equal to the current highest offer, then the system rejects that offer price.
- If the new offer price is greater than the highest offer, then the system accepts the new offer price and set the highest offer value to the new offer price. At this point, if this new offer price is greater than or equal to the asking price, then the system will close the sale post and display a message telling the responder that the item has been sold to him or her.

Constructors

At least one constructor to initialize all attributes shown above. When a new sale post is created, the creator will enter the item name, description, asking price and minimum raise. The UniLink system will generate a new unique sale post id and set that id and creator id to the newly created sale post object.

To reduce code duplication, your implementation must make use of the constructors created in the Post superclass.

Methods

`public String getPostDetails()`

You must override this method, which is inherited from the superclass Post, to return a String containing all common details of a post as shown in the Post class, and additional details of a sale post which are the minimum raise and highest offer according to the following example:

Example: output of a getPostDetails() method call on a sale post object, which hasn't yet received any offer

```
ID:              SAL001
Title:           IPad Air 2 64GB
Description:     Excellent working condition, comes with box,
                 cable and charger.
Creator ID:      s3456223
Status:          OPEN
Minimum raise:   $15.00
Highest offer:   NO OFFER
```

`public boolean handleReply(Reply reply)`

Your Sale class must implement this abstract handleReply method inherited from the Post superclass to handle a reply to a sale post. When a reply with a valid offer price is given, this method must add that reply to the replies collection and close the post if the price offer is greater than or equal to the asking price of the item. This method returns false to indicate that the given reply has a price offer which is not valid and that reply will not be kept in the replies collection.

`public String getReplyDetails()`

As shown in the Post class, this method should only be called by the post creator on their own posts. It returns a String containing details of all replies to that post. When calling this method on a sale post, this method must return the list of replies sorted in descending order of offer prices as shown in the example below:

Example: when a sale post creator chooses to view his own sale post, in addition to details of the post as shown in the getPostDetails method requirements, the post creator also sees the asking price. Furthermore, the getReplyDetails method is also called to display the Offer History section shown below. There are 5 replies sorted in descending order of offer prices. The highest offer is above the asking price, therefore the post is closed and the item is sold to the corresponding responder, the student s4.

```
ID:              SAL001
Title:           IPad Air 2 64GB
Description:     Excellent working condition, comes with box,
                 cable and charger.
Creator ID:      s3456223
Status:          CLOSED
```

```
Minimum raise:  $15.00
Highest offer:  305.00

Asking price:   $300.00 (NOTE: only visible to the post creator)

-- Offer History --
s4: $305.00
s5: $290.00
s3: $250.00
s2: $180.00
s1: $100.00
```

Accessor and mutator methods

Simple accessor and mutator methods must be implemented in this Sale class. You should only implement necessary accessor and mutator methods according to the additional attributes shown above. There must be no redundant accessor and mutator methods in your code.

Job class

This is another type of post to enable the post creator to describe a job which he or she needs to hire someone to do and the price at which the post creator expects to pay. Other students can see this job post in the system and reply with different prices they want to get paid to complete the job.

In addition to common information of a post such as id, title, description, creator id, status and a replies collection as shown in the Post superclass, a job post object also stores the following information:

- **Proposed price**: the maximum amount of money the job post creator is willing to paid for the job.
- **Lowest offer**: the lowest amount offered among all replies from students interested in taking this job.

Job post rules

The UniLink system only accepts a reply to a job post when the post is opened (i.e. post status is OPEN) and will process the new offer price in a reply according to the following rules:

- If the new offer price is greater than or equal to the current lowest offer, then the system rejects that new offer price.
- If the new offer price is less than the current lowest offer, then the system accepts the new offer price and set the current lowest offer value to the new offer price.
- The system will not automatically close a job post. The creator of the job post will manually close the job post and select a responder with the lowest offer price.

Constructors

At least one constructor to initialize all attributes shown above. When a new job post is created, the creator will enter the job name, description and proposed price. The UniLink system will generate a new unique job post id and set that id and creator id to the newly created job post object.

To reduce code duplication, your implementation must make use of the constructors created in the Post superclass.

Methods

```
public String getPostDetails()
```

You must override this method, which is inherited from the superclass Post, to return a String containing all common details of a post as shown in the Post class, and additional details of a job post, which are the proposed price and lowest offer according to the following example:

Example: output of a getPostDetails() method call on a job post object, which has already got a number of offer prices, only the lowest offer is shown:

```
ID:              JOB001
Title:           Moving House
Description:     Need a person to help me move my belongings to a
                 new house
Creator ID:      s3
Status:          OPEN
Proposed price: $190.00
Lowest offer:    $180.00
```

```
public boolean handleReply(Reply reply)
```

Your Job class must implement this abstract handleReply method inherited from the Post superclass to handle a reply to a job post. When a reply with a valid offer price is given according to the Job post rules above, this method must add that reply to the replies collection. This method returns false in cases when a given reply cannot be accepted according to the Job post rules above

```
public String getReplyDetails()
```

As shown in the Post class, this method should only be called by the post creator on their own posts. It returns a String containing details of all replies to that post. When calling this method on a job post, this method must return the list of replies sorted in ascending order of offer prices as shown in the example below:

Example: when a job post creator chooses to view his own job post, in addition to details of the post as shown in the getPostDetails method requirements, the getReplyDetails method is also called to display the Offer History section shown below. There are 4 replies sorted in ascending order of offer prices.

```
ID:              JOB001
Title:           Moving House
Description:     Need a person to help me move my belongings to a
                 new house
Creator ID:      s3545897
Status:          OPEN
```

```
Proposed price: $190.00
Lowest offer:   165.00


-- Offer History --
s4: $165.00
s3: $178.00
s2: $180.00
s1: $185.00
```

Accessor and mutator methods

Simple accessor and mutator methods must be implemented in this Job class. You should only implement necessary accessor and mutator methods according to the additional attributes shown above. There must be no redundant accessor and mutator methods in your code.

Implementing the UniLink system class

In Assignment 1, the UniLink system class is a console application. All user interactions with the system will be via a menu described below. In Assignment 2, you will create a graphical user interface of this system using JavaFx.

You are required to implement a class named UniLink which contains **a single collection of type Post** to store post objects of the three types, i.e. Event, Sale and Job posts. Posts and replies will be created by users at runtime by selecting options shown in the menus shown below.

Your implementation must propose code reuse, reduce code duplication and apply polymorphism correctly, especially when calling methods such as handleRely, getPostDetails and getReplyDetails on post objects.

When your program is executed, a user will be asked to login. After logging in, the main menu will appear as shown in the example below. To make it easier for testing, we don't use password, just need a student number to log in.

Assumption: in this assignment, assume that only one user can log in at a time.

Example: a student with the username `'s1'` logs in to the UniLink system and is presented with the UniLink System menu with the following two options:

```
** UniLink System **
1. Log in
2. Quit
Enter your choice: 1
Enter username: s1
```

As shown above, the user has logged in as student s1. Then the Student Menu is displayed:

```
Welcome s1!
** Student Menu **
1. New Event Post
```

```
2. New Sale Post
3. New Job Post
4. Reply To Post
5. Display My Posts
6. Display All Posts
7. Close Post
8. Delete Post
9. Log Out
Enter your choice:
```

To execute a feature of the system as shown in the menu above, the user needs to enter the number at the beginning of the corresponding menu item. If an invalid number is entered, an error message should be displayed, and the menu should be re-displayed. When a valid number is entered, your program should execute the corresponding method and then return to this student menu. Your program must stop when the user chooses to log out and quit the application.

This is a console application so all output data should be printed to the standard output.

Following is a description of each feature provided by the menu above:

Adding a new Event/Sale/Job post

From the student menu shown above, a student selects option 1, 2, 3 to add a new Event, Sale, Job post respectively. When a new post is created, your program must meet the following requirements:

- You program needs to ask the user for the correct information depending on the type of post.
- When the new post is created and saved successfully in the system, your program must show a message to let the user know about the id of the newly created post.
- User input validation must be implemented to make sure the user enters correct details for all the post.

Example: creating an event post

```
[Student menu]
Enter your choice: 1

Enter details of the event below:
Name: Friday night out
Description: Let's meet this Friday
Venue: Melbourne Central
Date: 10/04/2020
Capacity: 5
Success! Your event has been created with id EVE001
[Student menu is shown again]
```

Example: creating a sale post

```
[Student menu]
Enter your choice: 2
```

```
Enter details of the item to sale below:
Name: 13-inch Macbook Pro 2018 sale
Description: In good condition, ready to use
Asking price: 1850
Minimum raise: 40
Success! Your new sale has been created with id SAL001
[Student menu is shown again]
```

Example: creating a job post

```
Enter your choice: 3

Enter details of the job below:
Name: Moving House
Description: Need a person to help me move my belongings to a
new house
Propose price: 90
Success! Your job has been created with id JOB001
[Student menu is shown again]
```

Reply to a post

When a user replies to a post, he or she needs to enter the post id. Users can see all post id values when they select the Display All Posts option (described later). Your program must meet the following requirements:

- You program must ask the user for the post id only and use that id to search for the post in the posts collection. An error message will be displayed if user enters an invalid post id or a post not found.
- Your program must read the first 3 character in the id to determine the type of post to show the user only the required information to reply to a post. See examples below for more details about what information to show the user in each type of post.
- All exceptional cases are dealt with and appropriate messages are shown to users, for example, when a post is already closed, when the user has already replied to the post or if post creators are allowed to reply to their own posts.

Example: reply to a job post

```
Enter your choice: 4
Enter post id or 'Q' to quit: JOB001
Name: Moving House
Proposed price: $90.00
Lowest offer: $85.00
Enter your offer or 'Q' to quit: 80
Offer accepted!
[Student menu is shown again]
```

Display My Posts and Display All Posts

When the Display All Posts option is selected, your program needs to iterate through each post in the posts collection of the UniLink system and call the method getPostDetails on each post object. This is a polymorphic method call because the getPostDetails method is implemented differently in each subclass of the Post class. Details of all type of posts must be displayed correctly according to the requirements in each subclass shown above. However, replies details will not be shown until the user selects the Display My Posts option.

After the user logs in to the UniLink system by entering a student id as username, that user becomes the current user. When the Display My Posts option is selected, your program must retrieve only the posts created by the current user and display all details of those posts together with all replies details. Specifically, your program must call both method getPostDetails and getRepliesDetails on each post object created by the current user.

Close and Delete Post

Only the post creator can close or delete their own posts. Your program must meet the following requirements:

- When a user selects an option to either close or delete a post, your program must ask for a post id and use that id to retrieve the corresponding post and confirm with the user whether to close or delete the post.
- When a post is closed, its status is changed to CLOSED and replies are no longer accepted for this post.
- When post is deleted, its details and corresponding replies are removed from the system.
- Your program must not allow a post to be closed or deleted by a user who is not the creator of the post.
- Appropriate messages must be displayed, for example when the current user is not the post creator, the post is not found or already closed, and in any other case that needs to be addressed

Log Out

When a user selects the Log Out option, that user is no longer able to access the Student Menu. Your program will show the UniLink menu as shown below:

```
** Student Menu **
1. New Event Post
2. New Sale Post
...
...
8. Delete Post
9. Log Out
Enter your choice: 9
You have successfully logged out!

** UniLink System **
1. Log in
2. Quit
```

```
Enter your choice: 2
System exited! Thanks for using UniLink system
```

Note: if the Log In option is selected, then the user will be required to enter the student id as username and the Student Menu will reappear.

Start-up Class

You should create a startup class which contains a main method in which an object of the UniLink class is created and a single method is called on that object to run the entire UniLink application.

For testing purposes when we mark your program, you should hard-code some data. For example, when your program executes, there are the following:

- One event with a few participants
- One sale with a few offers and is still open
- One job with a few offers and is still open

Error Handling and User Input Validation

All user inputs must be validated, and appropriate messages are shown in error cases or when there are invalid inputs. Specifically, we will be checking the following:

- Validate user selection in menus, for example, in the UniLink System menu, there are two options, 1. Log in and 2. Quit, if the user enters an invalid input, you program must keep asking the user to enter either 1 or 2, until the user enters the correct number. Similar requirements apply to the Student Menu.
- If the user is required to enter a string, for example, post name or post description, your program must reject the empty string, i.e. user cannot press the Enter key to skip a post name or description.
- If a positive integer is required from the user, for example, event capacity, then your program must check to make sure that the user can only enter a positive integer.
- If a positive decimal number is required from the user, for example, the asking price or minimum raise in a Sale post, then your program must check to make sure that the user can only enter a positive number of type Double.
- Date in an event post must be in the format dd/mm/yyyy
- Your program displays correct messages in error cases or when there is an invalid input, for example when an invalid or non-existent post id is entered, when a sale post or job post rule is violated or when replying to a closed post...
- Your program should not crash at any point given any user input.

5. Referencing guidelines

You are free to refer to textbooks or notes and discuss the design issues (and associated general solutions) with your fellow students or on Canvas; however, the assignment should be your OWN WORK.

The submitted assignment must be your own work. For more information, please visit http://www.rmit.edu.au/academicintegrity.

Plagiarism is treated very seriously at RMIT. Plagiarism includes copying code directly from other students, internet or other resources without proper reference. Sometimes, students study and work on assignments together and submit similar files which may be regarded as plagiarism. Please note that you should always create your own assignment work even if you have very similar ideas.

Plagiarism-detection tools will be used for all submissions. Penalties may be applied in cases of plagiarism.

6. Submission format

You must submit a zip file of your project via Canvas.

You can submit your assignment as many times as you would like prior to the due date. The latest submission will be graded.

You MUST NOT submit compiled files (*.class files). Check your submission carefully, make sure all java files have been included.

7. Academic integrity and plagiarism (standard warning)

Academic integrity is about honest presentation of your academic work. It means acknowledging the work of others while developing your own insights, knowledge and ideas. You should take extreme care that you have:

- Acknowledged words, data, diagrams, models, frameworks and/or ideas of others you have quoted (i.e. directly copied), summarised, paraphrased, discussed or mentioned in your assessment through the appropriate referencing methods,
- Provided a reference list of the publication details so your reader can locate the source if necessary. This includes material taken from Internet sites.

If you do not acknowledge the sources of your material, you may be accused of plagiarism because you have passed off the work and ideas of another person without appropriate referencing, as if they were your own. RMIT University treats plagiarism as a very serious offence constituting misconduct. Plagiarism covers a variety of inappropriate behaviours, including:

- Failure to properly document a source
- Copyright material from the internet or databases
- Collusion between students

For further information on our policies and procedures, please refer to the University website.

8. Assessment declaration

When you submit work electronically, you agree to the assessment declaration.