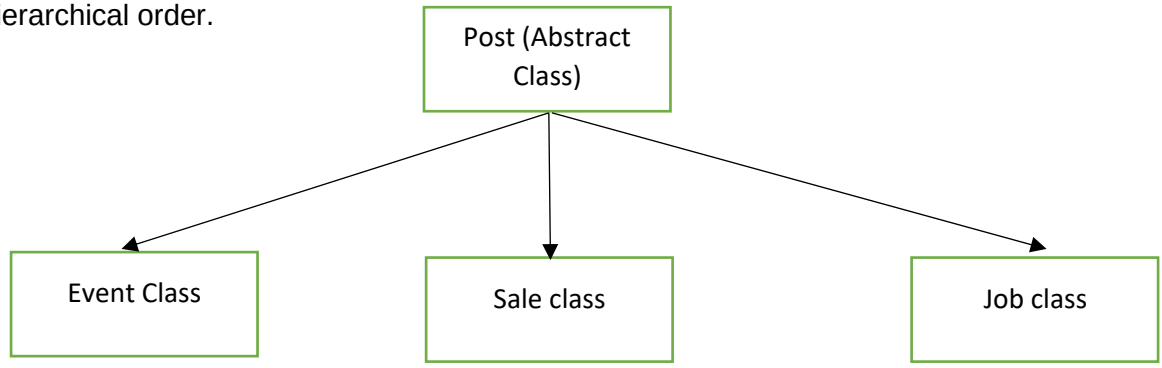
Advanced Programming
Assignment 3

Name : Varun Ramesh
Student Id : s3793675

Task 1

A.

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance, the information is made manageable in a hierarchical order.




```
public abstract class Post
public class Sale extends Post
public class Event extends Post
public class Job extends Post
```

B.

The benefits of inheritance is to reduce duplicate code in an application which share a common code among classes. If a similar code exists in 2 related classes, a parent class can hold up the common elements and be called when a method needs to be used in any of the child classes. This makes the code smaller and simpler.

Inheritance can also make application code more flexible to change because classes that inherit from a common superclass can be used interchangeably. If the return type of a method is superclass

Reusability

Extensibility

Data hiding

Overriding

Each of the child class (Event, Job, Sale) has function which are common like post details method, handle reply method and get reply details method. Although each of these classes had their own attributes, they had 5 attributes which are common all throughout the posts. This made the code easier to read and more organized.

Task 2

A.

Polymorphism in Java means many forms. There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism.

In Post Abstract class :

```
@Override
public String getPostDetails() {
    return ("ID: " + this.id + "\nTitle: " + this.title +
"\nDescription: "
            + this.description + "\nCreator ID: " + this.creatorId +
"\nStatus: " + this.status);

}
```

- In Event Class extends Post

```
@Override
public String getPostDetails() {
    return super.getPostDetails()+ ("\nVenue: " + this.venue +
"\nDate " + this.date
            +"\nCapacity: " + this.capacity + "\nAttendees: " +
this.attendeeCount) ;
}
```

- In Sale Class extends Post

```
@Override
public String getPostDetails() {

    return super.getPostDetails() + ("\nHighest Offer: " +
this.highestOffer + "\nMinimum Raise: "
            + this.minimumRaise);
}
```

- In Job Class extends Post

```
@Override
public String getPostDetails() {

    if(this.lowestOffer > this.proposedPrice){
        return super.getPostDetails() + "\nProposed Price: " +
this.proposedPrice;
    }
```

We can see that the same method has different forms which are being taken. In each class the same method performs different functions. Polymorphism can also be used with the data members. Looking at the example of a method in Unilink (model package)

```
public ObservableList<PostDetails> addFilter(String type, String status,
String creator, String username) {
```

Looking at the above method, we can see the method is of type ObservableList of PostDetails. To this we can add all types of posts (event, job and sale) this makes it easier and convenient to just use one object type instead of using different ones for each post.

B.

• It was useful in reusing the code, classes, methods written once, tested and implemented. They may be reused in many ways.

• The single object can be used to store variables of multiple data types and objects.

• Polymorphism helps in reducing the coupling between different functionalities.

• Method overloading can be implemented on constructors allowing different ways to initialize objects of a class. This enables you to define multiple constructors for handling different types of initializations.

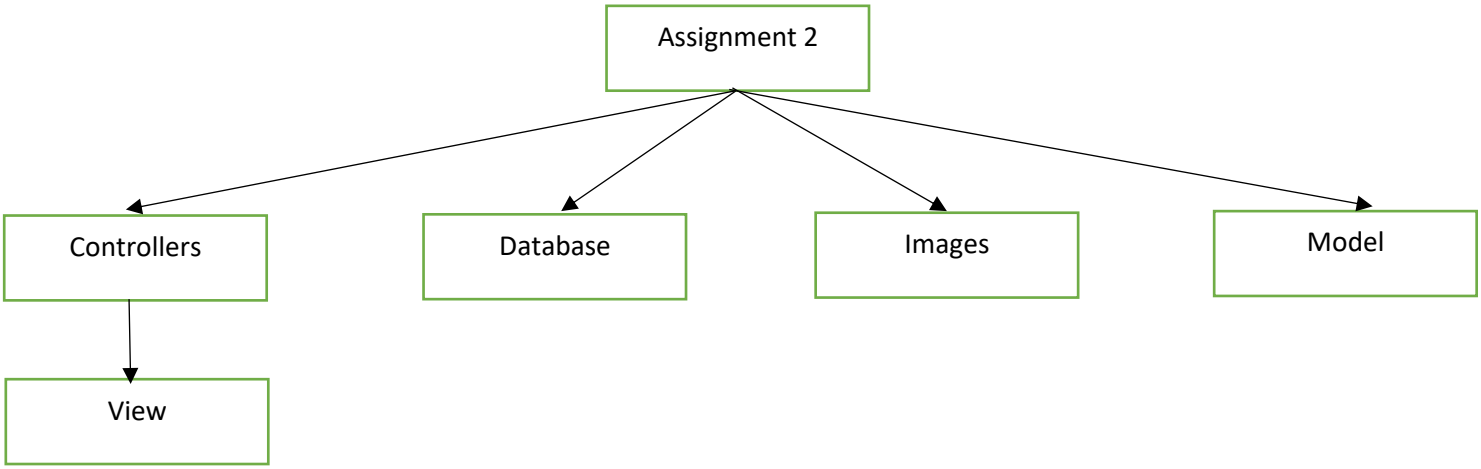
Task 3

A.

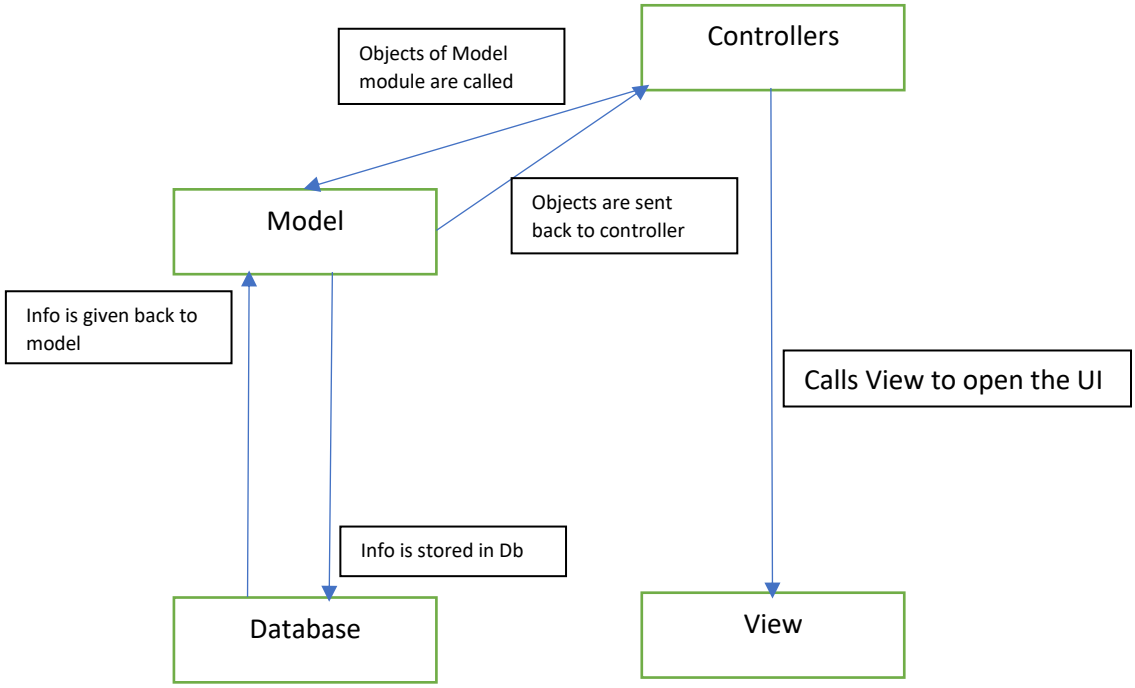
The reasons that my code is extensible because:

- Use of open close principle by using inheritance of different classes from the same parent class.
- Each of the methods created has been used in more than one place. This makes it easier to edit one method so that all the functionalities can be changed.
- Using appropriate Encapsulation to make sure data is not lost or prevent direct access of another object component. All the classes including the controller module has been loosely coupled. However there was some data that needed to be pushed through the classes because of opening one scene to the next. This was because of the requirement asked to display the username in the main page. Also, the logged in Username needed to be transferred between scenes to make appropriate functions.
- Following global conditions in writing the program. The beginning of the program has the developer name. All the redundant classes are removed. All import statements on the beginning of each class. Static and non-static imports are in different blocks. Braces are also used where needed. Most variables which are unused have been removed.

B.



Now looking at the flow of the whole program:



Now to understand by chunks of code for when a new Event Post Button is pressed:

- Controller (Main Page)

```
public void newEventPostPressed(javafx.event.ActionEvent event)
throws IOException {
    FXMLLoader loader = new
FXMLLoader(getClass().getResource("../view/eventNew.fxml"));
    Parent root = (Parent) loader.load();
    EventNewPost secController = loader.getController();
    secController.setUsernameEvent(username);
```

  - A new window is opened using Controller (Event New Post)
    - We can see here that all the information from Event New Post is being sent to Unilink (model module)

```
uniLinkOb.addEventPost(eventName, eventDescription, username,
"Open", venue, date, Integer.parseInt(capacity), 0,
imageEvent);
```

- Model (Unilink)
    - Here the objects from Controller is taken and retrieved here in the method 'addEventPost'
    - This method now takes all this information and stores in the database

```
"INSERT INTO \"event\"
```

    - Meanwhile this information is added to an array list of post of type event. This is later called Main Page to retrieve this information and then the 'filter new post' method is used to send it back to main page
- Now moving back to Main Page (controllers)
    - We are setting this in a list view by obtaining this information from Unilink.

We can see from the above flow that there is an equal balance of cohesion and coupling. We can see that all the information is protected. It is also noticed each of the modules have a purpose and consists of classes that are highly independent.

References

- *Java Polymorphism*. W3schools.com. Retrieved 14 June 2020, from https://www.w3schools.com/java/java_polymorphism.asp.
- *Java Polymorphism*. W3schools.com. Retrieved 14 June 2020, from https://www.w3schools.com/java/java_inheritance.asp