Final Project - Work order Project

This program is a Work Order Request System which uses the concept of object oriented programming to layout a system that can be used by a client of any work environment. Here, the client is provided with a graphical interface that shows all the work orders they have requested and whether they are completed. The user will have the ability to add, delete and edit work orders. He will also be able to request different types of work orders including Repair, Housekeeping, IT support.

Each user will be an object having a name, username , email address and password. There exists a login or sign up page too that will grant entry into the system.

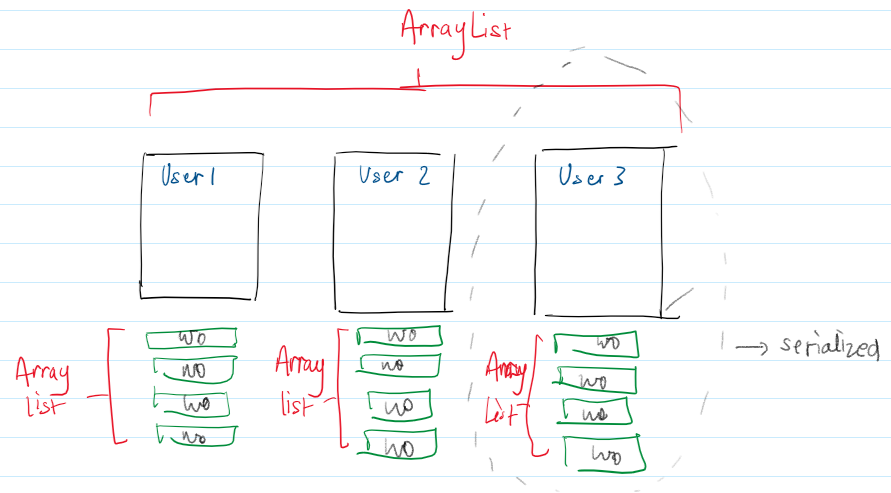
Each work order also an object is made up of a unique work order number (five digit String), date of entry, work order name and category(work order type), completion status ( completed or incomplete).

Language chosen (Java):

The main reason was to save instances of a class as elements of an ArrayList and then serialize them. My choice of such a type of data persistence was due to the ease of use, and customization.

Concepts used:

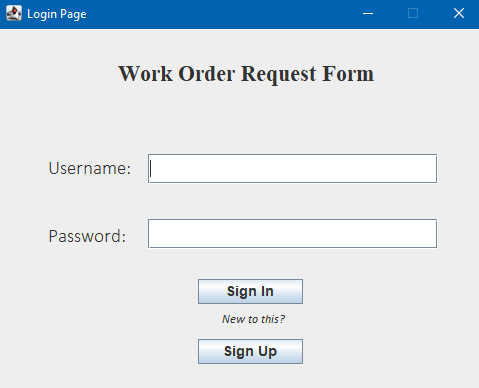
1. Encapsulation : This was done by using setters and getters to access my data members that were by default all protected. This was done during setting the various attributes of both the account class and the work order class.
2. Polymorphism - Overriding some of the methods that exist in the interface and abstract classes. Due to my extensive use of Swing GUI, most actionListeners() were of the form of an abstract class or an interface. Due to this, I was required to not only add but override them in some cases.
3. Abstraction - The work order class will be an abstract class that will act as the parent class for all its subsequent classes which are the specific work orders like electric, IT support and all. The GUI had a great deal of abstractions, interfaces like the addActionListener class that has the method ActionPerformed.
4. Serialization - implementing data persistence in this project will involve object serialization. In my project, there exists an ArrayList of instances of a class that in turn contained an ArrayList of instances of another class. It is an ArrayList of UserAccounts that contain an ArrayList of WorkOrders. Serialization was the best choice due to that fact that I can serialize and deserialize these objects without worrying about loss of data or finding any data stored.
5. Interfaces - Swing GUI has many instances when I am to use interfacing and implementation.
6. Inheritance: JFrame is a child class of awt Frame which is a child class of Window continued all the way till Java.lang.object. Understanding this concept was huge since I was able to use methods from any of the parent class without particularly importing or creating the library.
7. Memory Allocation : Since my programs have instances and objects, I understand that these are all stored in the Heap. The reference to these objects are only stored in the Stack.
8. Aggregation: In my program, Main.java is connected to UserAccount.java and then consequently connected to the WorkOrder.java.
9. ArrayList - Each instance of the object is stored as an element in the ArrayList and each instance subclass of the work order too will be stored in the same fashion. This enables the delete, add and edit functionality.



Code Walkthrough:

I have four classes namely:

1. Main()
2. UserAccount();
3. WorkOrder();
4. Admin();

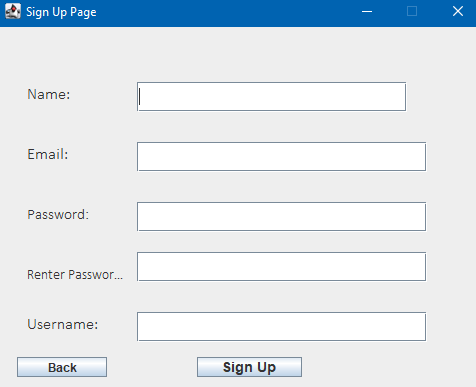
1.Main() class:

This class ties together the whole program, it basically starts off the program and provides the Login and Sign up userInterface.

First comes the Textfields of input, that uses access modifiers of userName and passWord and sets them to Strings for user authentication.

Authentication() is called on the clicking of the **Sign in button** that is connected to an actionListener().

Authentication contains a series of conditional and iterative logic that does the following :



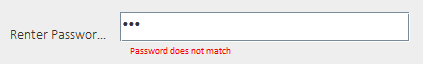
1. If admin and 12345 is entered, take me to the adminMenu() in the Admin class
2. If admin and the wrong password is entered, show an error message
3. If both field are empty, show an error message
4. If list is empty, tell user that no users exist and sign up is prompted
5. If list is not empty, then go through the arraylist and match a username
6. If the username is not found, tell the user that the username does not exist.
7. If username found, make boolean found = true
   1. If password does not match the username display “incorrect password”
   2. If password does match = go the userMenu() in UserAccount class

On clicking on the sign up button, the program presumes that the user does not have an existing account. User is taken to another UI called the sign up page UI

This consists of a number of text fields that are all put together using access modifiers and created into an object. The UI is pretty straightforward but there are some restrictions to what can be inputted:

1. In email, a proper email address with an “@” symbol and a “.” should be entered. If not a little error message occurs that blocks the user from signing up.
2. Password and reenter password must match or else the same will occur.
3. Username is should have not been already taken.

This works using Swing's keyListener which uses the abstract class called keyRelease which performs an action which in this case is input validation.

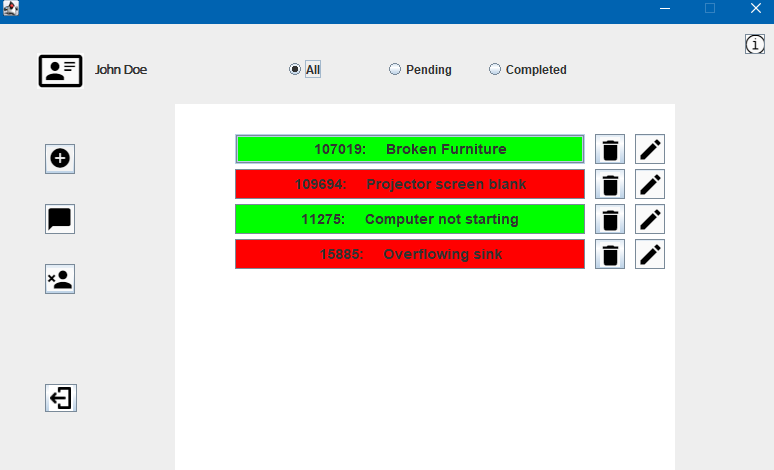


The back button just forgets whatever is entered and goes back to the login page.

On clicking the Sign Up button a new object of UserAccount is created with the fields of whatever was entered by the user. And again just like in the page, if there are any errors in input like the above or any empty text fields then, error messages are displayed that prompt the user in the direction of amendment. All previous frames and windows are disposed off and the login menu is restarted.This instance of the UserAccount class is saved as an element of an arraylist. The save() method is called too, which serializes the object into a .dat file and loads the lists.

On Valid Signing in :

menu() method is called from the UserAccount that provides an interface as shown in the picture below.

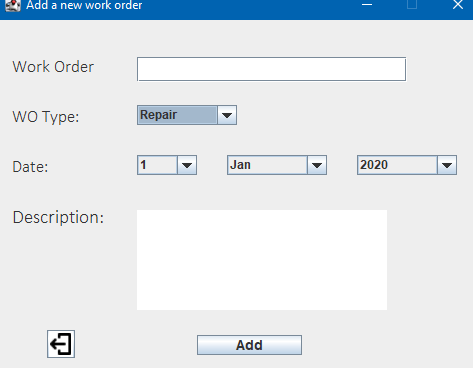


This is the general UI, that is displayed on clicking the sign in. All of the UI is comprised in the UserAccount class that have different methods for different components of the screen. The icons represent the functions of each button. The main part of this window is the listView(). This executes a for loop through the ArrayList and prints out a button with a color depending on the completion which the instances name, work id , date , description and so on. This is again retrieved from the access modifiers that provide encapsulation of data members. As a filter, I attached radio buttons that are straight forward :

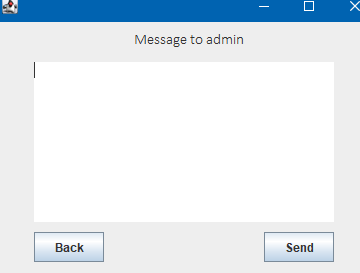
1. All - all the complete and incomplete work orders are presented on the screen
2. Pending - the incomplete ones are filtered and present in the color red.
3. Completed - only the completed ones are declared with the color green.

Attached aside each button are options that allow the user to delete or edit work orders according to their needs. This is also attached in the for loop and has an ActionListener to its consequent methods. On clicking each

On the left side is a panel of buttons that do the following:

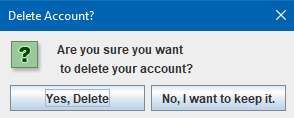
1. Add a new work order - This triggers a chain of methods that lead to a UI where there are prompts based on the work order properties. The user enters the name of the work order. Then from a dropdown menu, choose a type of work order which is at the end of the day inherited from the abstract class WorkOrder. The date dropdown allows the user to select the date of request. And finally the white textarea where the user is to describe the work order request in depth.

On clicking Add, a new work order is created depending on the type it may be ITSupport, Repair or HouseKeeping which will use access modifiers to be instantiated and added to the arraylist of work order that is unique for each user.



1. Sends a message to the admin

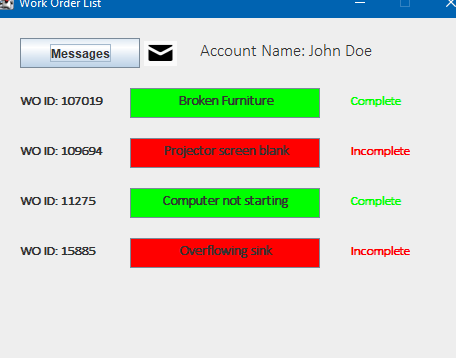
This function is straightforward in the sense that the user can send a message be it commending or complaining to the admin which will be stored as String in the accounts inbox. The back button cancels the action. Send saves the message using the access modifiers and then returns the user to the main menu().

1. Delete the account :

On clicking the delete account button, the user is able to remove his account from the database. The user is redirected to an OptionPane and asked again for confirmation. If no, then action is cancelled. If yes, then a static method in Main is called as delete which takes the index and removes it from the arraylist and reruns the program from the main page.

1. Log out to the system : This is just a new Main() call which in a way reruns the program and therefore logs the user of the page.

Admin Menu : If on the login page, the entered details is admin as the username and 12345 as the password, then up comes the adminMenu() which is from the Admin class. This is basically where all the controlling of the work order occurs after the user does his part. The admin menu is basically a list of all the registered users with a delete option next to it. On clicking each button of the account list, one will be redirected to the work orders for that particular account which in turn will also be in the form of a list with color coded info based on the completion. The admin can view messages sent by the userAccount() and also change the completion status of each work order through an JOptionPane. The back button simply reruns the code and takes a step back in the aggregation process.



UML :

