



## GROUP ASSIGNMENT

### OBJECT ORIENTED DEVELOPMENT WITH JAVA

CT038-3-2-OODJ

UC2F2008CS(IS)

HAND IN DATE: 20 September 2020

HAND OUT DATE: 27 NOVEMBER 2020

MEMBER 1: TAN HOONG LOO TP051111

MEMBER 2: YAP XUAN MING TP051166

INTAKE NUMBER: UC2F2008CS(IS)

LECTURER NAME: USMAN HASHMI

**Workload Matrix**

	Tan Hoong Loo TP051111	Yap Xuan Ming TP0551166
Delivery Staff Edit Profile	x	
Delivery View order	x	
Customer Feedback		x
Feedback management	x	
Report Management		x
User Account Management	x	
Order Management		x
Login Page		x

## Table of Contents

<b>1.0 Assumption.....</b>	<b>4</b>
<b>2.0 Diagram.....</b>	<b>5</b>
<b>2.1 Use Case Diagram.....</b>	<b>5</b>
<b>Description of Use Case Diagram.....</b>	<b>6</b>
<b>2.2 Class Diagram.....</b>	<b>13</b>
<b>3.0 User Manual.....</b>	<b>14</b>
<b>3.1 Customer.....</b>	<b>14</b>
<b>3.2 Manager.....</b>	<b>16</b>
<b>3.3 Delivery Stuff.....</b>	<b>20</b>
<b>4.0 Description and Justification of the design and implementation codes.....</b>	<b>22</b>
<b>4.1 Inheritance.....</b>	<b>22</b>
<b>4.2 Encapsulation.....</b>	<b>24</b>
<b>4.3 Abstraction.....</b>	<b>27</b>
<b>4.4 Polymorphism.....</b>	<b>29</b>
<b>4.5 Object.....</b>	<b>30</b>
<b>5.0 Extra Features 1.....</b>	<b>31</b>
<b>5.1 Extra Features 2.....</b>	<b>32</b>
<b>6.0 Report of object-oriented programs.....</b>	<b>33</b>
<b>6.1 Inheritance.....</b>	<b>34</b>
<b>6.2 Encapsulation.....</b>	<b>35</b>
<b>6.3 Abstract.....</b>	<b>36</b>
<b>6.4 Polymorphism.....</b>	<b>37</b>
<b>7.0 Conclusions.....</b>	<b>39</b>
<b>8.0 References.....</b>	<b>41</b>

## 1.0 Assumption

In this courier service system, we assume that every order that the manager update will click “calculate” button in the order management class before they click the “update” button. After that, if the manager would like to print their report based on the order, they will print as pdf form. We assumed that the delivery staff will know which users they will be logging in as the system will show the name on top of the class. We assumed that each of the order made will be pending first, until the delivery staff clicked “Delivered” button in their page, then the status will change to delivered, as this is to prove that the delivery staff had successfully send the parcel to the customer. Moreover, we assumed that the customer would enter their feedback by login to their existing account, the feedback that every customer wrote will be anonymous, in case the manager or the staff will look for them. Therefore, we assumed that the order ID will be generated as random number and in case the ID had existed in the text file, there will be a notification that warn the users that the order ID is exist.

## 2.0 Diagram

### 2.1 Use Case Diagram

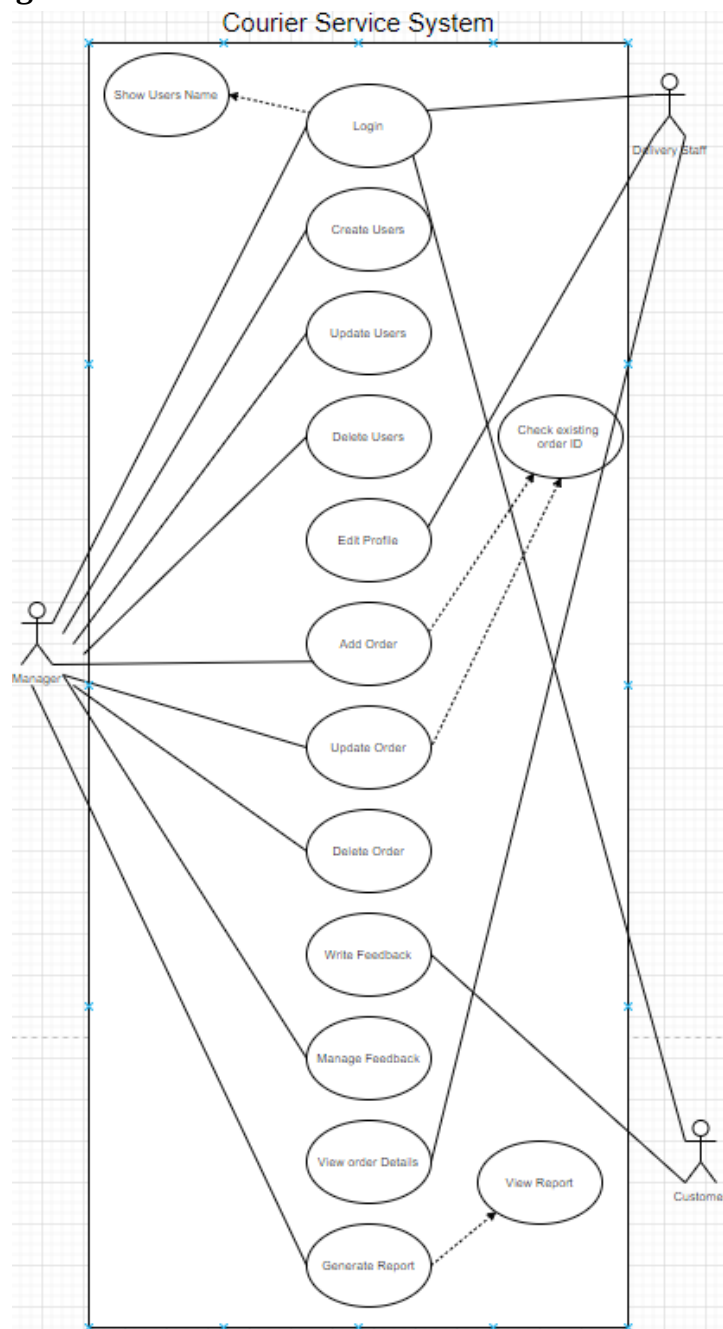


Figure 2.1, Use Case Diagram

### Description of Use Case Diagram

Use Case	<b>Login</b>
Brief Description	This is to allow the users to login the system
Actors	Delivery Staff, Manager and Customer
Precondition	Delivery Staff, Manager, and customer are necessary to be registered by an existing manager before they allow to access into the system
Main Flow	<ol style="list-style-type: none"> <li>1) The system requires the users to enter their login information (username and password).</li> <li>2) The users enter the details</li> <li>3) The system will check and authenticate the details. Next, the system will allow the users to login the system based on their particular role</li> <li>4) Use case end</li> </ol>
Alternative Flow	Wrong detail (Invalid username / password) <ul style="list-style-type: none"> <li>- System display error message of wrong username / password</li> <li>- Users re-enter the login information</li> </ul>

Use Case	<b>Show user's name</b>
Brief Description	This is to allow the system to show the current login name on top of the class
Actors	Manager, Customer, Delivery Staff
Precondition	The users are logged in as their specific role
Main Flow	<ol style="list-style-type: none"> <li>1) Users will need to log in as their individually account</li> <li>2) Use case end</li> </ol>
Alternative Flow	- N.A

Use Case	<b>Create Users</b>
Brief Description	This is to allow the manager to create account for others user.
Actors	Manager
Precondition	The users are logged in as manager
Main Flow	<ol style="list-style-type: none"> <li>1) Manager will be filled in the necessary details about the users that will create.</li> <li>2) Manager will be filled in all the section</li> <li>3) Use case end</li> </ol>
Alternative Flow	Wrong phone number format (Invalid format) <ul style="list-style-type: none"> <li>- System will display an error message of wrong phone number format (xxx-xxxxxxx)</li> </ul>

	<ul style="list-style-type: none"> <li>- Managers re-enter the phone number</li> </ul> <p>Wrong email address format (Invalid format)</p> <ul style="list-style-type: none"> <li>- System will display an error message of wrong email address format (aaa@aaa.com)</li> </ul> <p>Some section is not filled up by the manager</p> <ul style="list-style-type: none"> <li>- System will display an error message of the section are not filled up completely</li> <li>- Managers check missing information and enter the information</li> </ul>
--	---

Use Case	<b>Update Users</b>
Brief Description	This is to allow the manager to update account for others user.
Actors	Manager
Precondition	The users are logged in as manager
Main Flow	<ol style="list-style-type: none"> <li>1) Manager will be selecting the specific account in the table</li> <li>2) Manager will be updating the specific account</li> <li>3) Use case end</li> </ol>
Alternative Flow	<p>Wrong phone number format (Invalid format)</p> <ul style="list-style-type: none"> <li>- System will display an error message of wrong phone number format (xxx-xxxxxxx)</li> <li>- Managers re-enter the phone number</li> </ul> <p>Wrong email address format (Invalid format)</p> <ul style="list-style-type: none"> <li>- System will display an error message of wrong email address format (aaa@aaa.com)</li> </ul> <p>Some section is not gilled up by the manager</p> <ul style="list-style-type: none"> <li>- System will display an error message of the section are not filled up completely</li> <li>- Managers check missing information and enter the information</li> </ul>

Use Case	<b>Delete Users</b>
Brief Description	This is to allow the manager to delete account for others user.
Actors	Manager

Precondition	The users are logged in as manager
Main Flow	1) Manager will be deleting the specific users 2) Use case end
Alternative Flow	N.A

Use Case	<b>Edit profile</b>
Brief Description	This is to allow the delivery staff to edit their information
Actors	Delivery Staff
Precondition	The users are logged in as delivery staff
Main Flow	1) Delivery Staff will be filled in the necessary details the information they would like to edit. 2) Use case end
Alternative Flow	Wrong phone number format (Invalid format) <ul style="list-style-type: none"> <li>- System will display an error message of wrong phone number format (xxx-xxxxxxx)</li> <li>- Managers re-enter the phone number</li> </ul> Wrong email address format (Invalid format) <ul style="list-style-type: none"> <li>- System will display an error message of wrong email address format (aaa@aaa.com)</li> </ul>

Use Case	<b>Add order</b>
Brief Description	This is to allow the manager to add order.
Actors	Manager
Precondition	The users are logged in as manager
Main Flow	1) Manager will be filled in the necessary details about the order that will create. 2) Manager will be filled in all the section 3) Use case end
Alternative Flow	Wrong phone number format (Invalid format) <ul style="list-style-type: none"> <li>- System will display an error message of wrong phone number format (xxx-xxxxxxx)</li> <li>- Managers re-enter the phone number</li> </ul> Some section is not filled up by the Manager <ul style="list-style-type: none"> <li>- System will display an error message of the section are not filled up completely</li> <li>- Manager check missing information and enter the information</li> </ul>



--	--

Use Case	<b>Update order</b>
Brief Description	This is to allow the Manager to update the existing order
Actors	Manager
Precondition	The users are logged in as manager
Main Flow	<ol style="list-style-type: none"> <li>1) Manager will be selecting the specific account in the table</li> <li>2) manager will be updating in the necessary details about the order that created</li> <li>3) Manager will be filled in all the section</li> <li>4) Use case end</li> </ol>
Alternative Flow	<p>Wrong phone number format (Invalid format)</p> <ul style="list-style-type: none"> <li>- System will display an error message of wrong phone number format (xxx-xxxxxxx)</li> <li>- Managers re-enter the phone number</li> </ul> <p>Some section is not filled up by the Manager</p> <ul style="list-style-type: none"> <li>- System will display an error message of the section are not filled up completely</li> <li>- Manager check missing information and enter the information</li> </ul>

Use Case	<b>Delete order</b>
Brief Description	This is to allow the manager to delete the existing order
Actors	Manager
Precondition	The users are logged in as manager
Main Flow	<ol style="list-style-type: none"> <li>1) Manager will be selecting the specific account in the table</li> <li>2) Manager will be deleting the necessary details about the order that created</li> <li>3) Use case end</li> </ol>
Alternative Flow	- N.A

Use Case	<b>Check Existing Order ID</b>
----------	--------------------------------

Brief Description	This is to allow the system to check whether the order ID are existed
Actors	Manager
Precondition	The users are logged in as manager
Main Flow	1) Manager will check whether the order ID are existed in the system 2) Use case end
Alternative Flow	Same order ID are existed in the text file - System will be required manager to regenerate the order ID

Use Case	<b>Write Feedback</b>
Brief Description	This is to allow the customer to enter the feedback that would like to show to the manager
Actors	Customer
Precondition	The users are logged in as customer
Main Flow	1) Customers will write the feedback 2) Customer will click add to submit the feedback to the text file 3) Use case end
Alternative Flow	N.A

Use Case	<b>Manage Feedback</b>
Brief Description	This is to allow the manager to view and delete the feedback that wrote by the customer
Actors	Manager
Precondition	The users are logged in as manager
Main Flow	1) Manager will view the feedback 2) Manager can delete the feedback 3) Use case end
Alternative Flow	N.A

Use Case	<b>View order details</b>
Brief Description	This is to allow the delivery staff to view the order that created by

	the manager
Actors	Delivery staff
Precondition	The users are logged in as delivery staff
Main Flow	1) Delivery staff will view the order that created by delivery staff 2) Use case end
Alternative Flow	N.A

Use Case	<b>Generate Report</b>
Brief Description	This is to allow the manager to generate a report.
Actors	Manager
Precondition	The users are logged in as manager
Main Flow	1) Manager will generate a report 2) Use case end
Alternative Flow	N.A

Use Case	<b>View Report</b>
Brief Description	This is to allow the manager to view the report
Actors	Manager
Precondition	The users are logged in as manager
Main Flow	4) Manager will view the report by selecting any specific order in the table 5) Use case end
Alternative Flow	N.A

## 2.2 Class Diagram

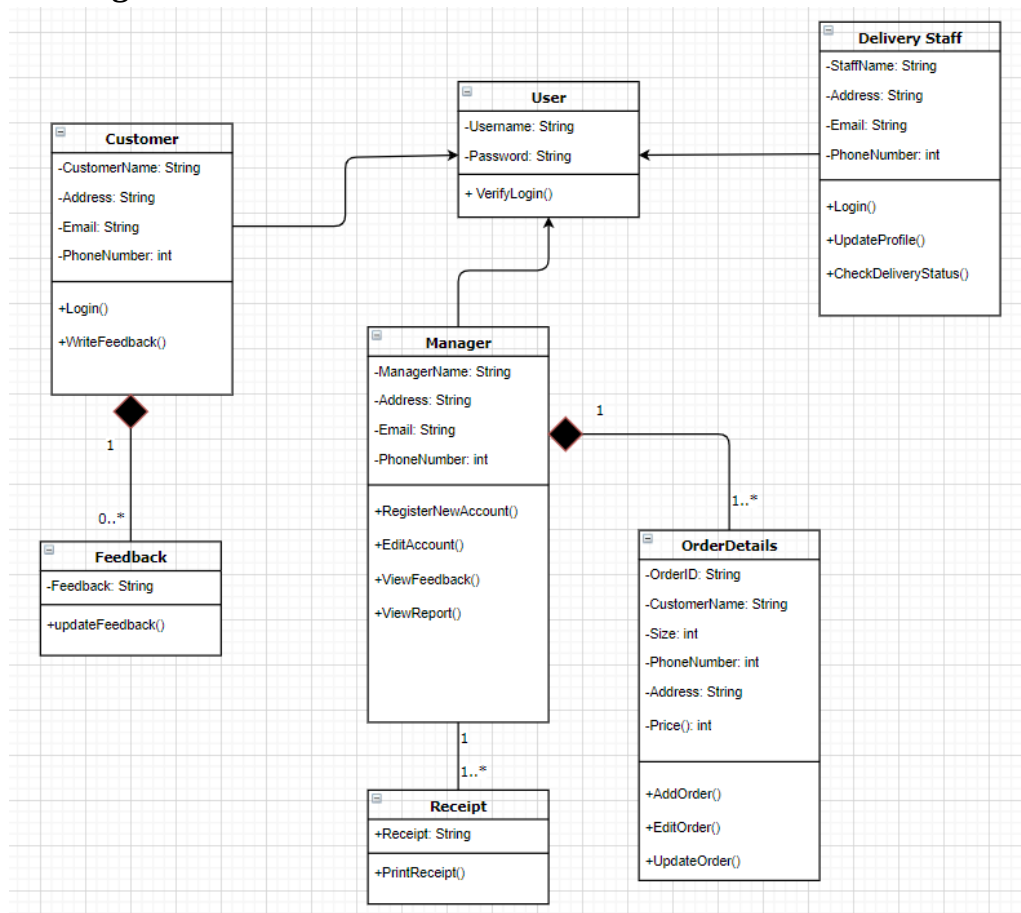


Figure 2.1, Class Diagram

We have a total of 3 Users, the Customer, Manager and the Delivery Staff. From the user table, the customer, manager and the delivery staff can be logging in to the system using their Username and Password created. Customer then can have the ability to write feedback for the manager. Feedback are composition to the customer, there will be no feedback if there are no customer available to write. It is one to zero to many because a customer can choose either not to write any feedback or write more than one feedback.

The manager will have their details like address, email and phone number recorded when registered, those information will be private. They can also edit account, view feedback and view report. Manager are in charge of the order details and every information of the order details will be made privately, the manager can also add more order, edit order and update order. Order details are also composition to the manager as there will be no order added if there are not any manager available. A manager can add 1 or more than 1 order at a time. After viewing the report, the manager can also print receipt of the report, a manager can print one or more than one receipt at a time.

Delivery Staff will also have their information kept privately after creating their account. The delivery staff can update their own individual profile and also have the ability to check the delivery status.

## 3.0 User Manual

### 3.1 Customer

To begin with, the customer will have to login with their own account in the login page like the picture in Figure 3.1.1.

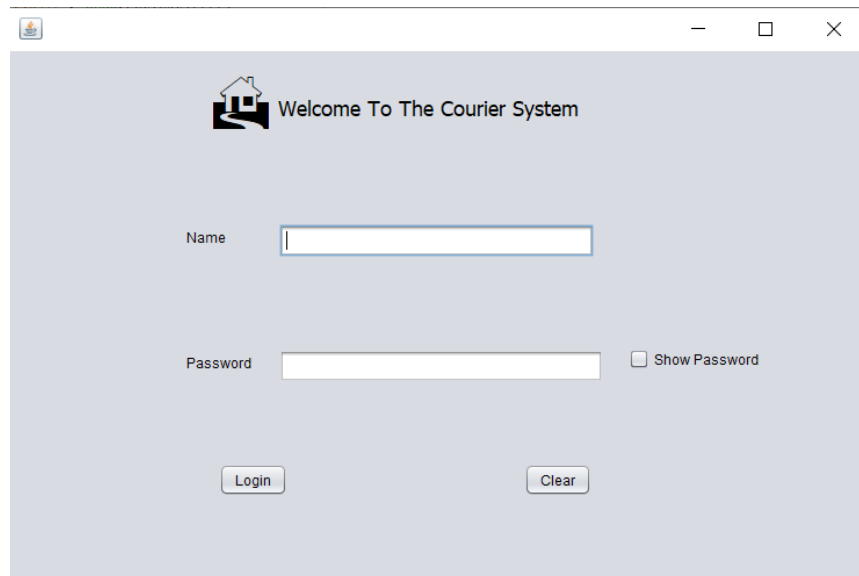


Figure 3.1.1, Login Page

The code for login will be show in Figure 3.1.2. Their information will be collected from the text file and being scanned by each array to see if it's the same data, the customer page will show if the data between the text file are identical.

```
else if(username.equals(tempArr[0]) && password.equals(tempArr[3]) && "customer".equals(tempArr[4])){  
    nm = tempArr[0];  
    pw = tempArr[3];  
    em = tempArr[1];  
    pn = tempArr[2];  
    dl = tempArr[4];  
    confirmation = true;  
    JOptionPane.showMessageDialog(rootPane, "You\'re Logged In!", "Customer Login Successful", JOptionPane.INFORMATION_MESSAGE);  
    Customer c = new Customer();  
    c.setVisible(true);  
    this.dispose();  
}
```

Figure 3.1.2, Code Customer Login

Figure 3.1.3 is a picture of the customer page. The customer can make feedback by interacting with the write feedback button. Customer name will be show after logging in to the account, Figure 3.1.3 just got logged in from Josh.

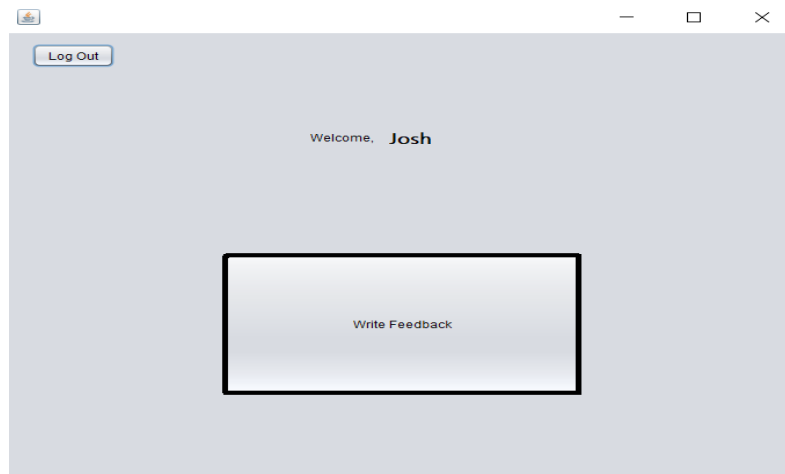


Figure 3.1.3, Feedback Page

In figure 3.1.4 is a picture of the Feedback page after interacting with the Write Feedback button. Customer can write their feedback as anonymous where their name cannot be seen by the manager.

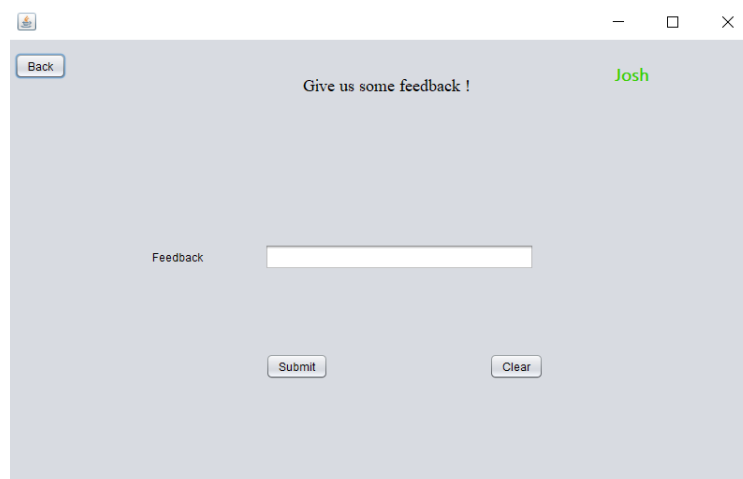


Figure 3.1.4, Feedback page

The code for creating the submit feedback are being use in figure 3.1.5.

```

FileWriter file = null;
if (checkEmpty()==true){
    JOptionPane.showMessageDialog(rootPane, "Please enter the field to submit", "Incomplete Fields", JOptionPane.ERROR_MESSAGE);
    //set cursor
    feedback.requestFocus(true);
    return;
}
else{
    try{
        JFeedback jf = new JFeedback(feedback.getText());
        file = new FileWriter("Feedback.txt", true);
        PrintWriter pw = new PrintWriter(file);
        pw.println(jf.getFeedback());
        file.close();
        pw.close();
        JOptionPane.showMessageDialog(rootPane, "Feedback Submitted!", "Add Successful", JOptionPane.INFORMATION_MESSAGE);
    }
    catch (IOException ex){
        JOptionPane.showMessageDialog(rootPane, ex.toString());
    }
}

```

Figure 3.1.5, Code for submit

### 3.2 Manager

Manager will have to login first just like how the customer will be logging in, code will be using the same as the customer too. In figure 3.2.1 is a picture of the Manager Home Page after logging in.

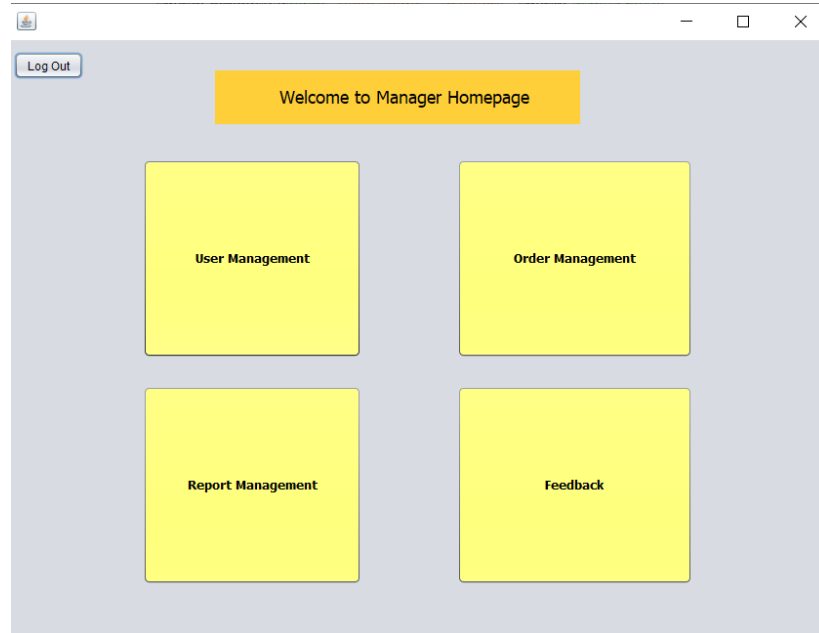


Figure 3.2.1, Manager Homepage

Firstly, the manager can create edit or delete their workers account by clicking the User Management button. Figure 3.2.2 is what will the User Management Page looks like.

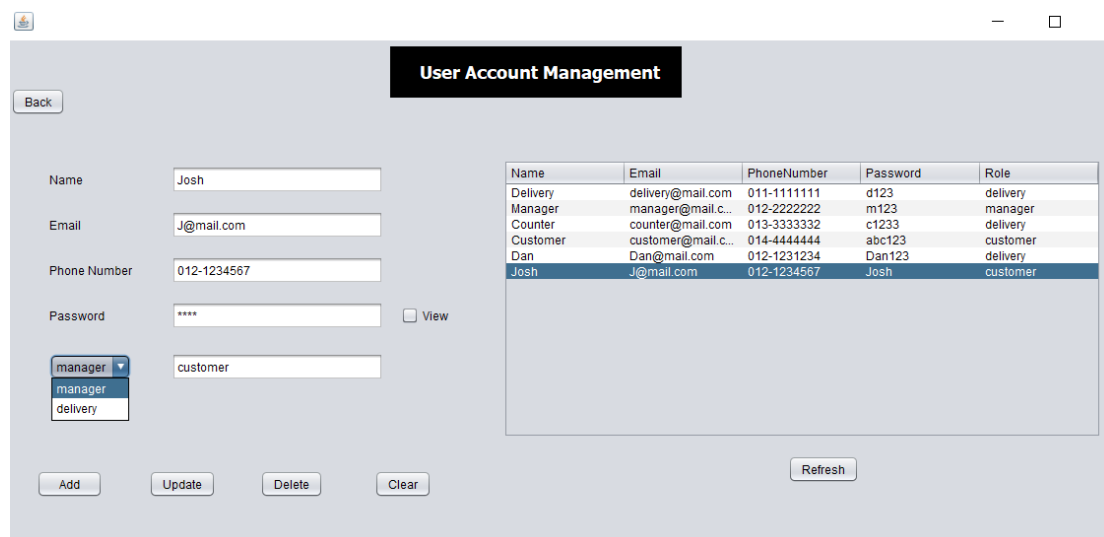


Figure 3.2.2, User Management

The code for add function in Figure 3.2.3 with validation.

```
FileWriter file = null;
if (checkEmpty() == true) {
    JOptionPane.showMessageDialog(rootPane, "Please complete all fields", "Incomplete Fields", JOptionPane.ERROR_MESSAGE);
    //set cursor
    name.requestFocus(true);
    return;
}
else{
    if (!(Pattern.matches("[a-zA-Z0-9]+@[0-9]{1}+[a-zA-Z0-9]+.[0-9]{1}+[a-zA-Z0-9]+$", email.getText())) {
        JOptionPane.showMessageDialog(rootPane, "Enter a valid email", "Error", JOptionPane.ERROR_MESSAGE);
    }
}
else{
    if (!(Pattern.matches("\\d{3}-\\d{7}", pnum.getText())) {
        JOptionPane.showMessageDialog(rootPane, "Phone Number format must be xxx-xxxxxxx", "Error", JOptionPane.ERROR_MESSAGE);
    }
}
else{
    try{
        file = new FileWriter("login.txt", true);
        PrintWriter pw = new PrintWriter(file);
        pw.print(l.getName() + ",");
        pw.print(l.getEmail() + ",");
        pw.print(l.getPhoneNumber() + ",");
        pw.print(l.getPassword() + ",");
        pw.println(l.getRole() + ",");
        file.close();
        pw.close();
        JOptionPane.showMessageDialog(rootPane, "Data Added", "Add Successful", JOptionPane.INFORMATION_MESSAGE);
    }
}
```

Figure 3.2.3, Add Function

Code for delete function in Figure 3.2.4.

```
ArrayList<String> tempArray = new ArrayList<>();

try{
    try(FileReader fr = new FileReader("login.txt")){
        Scanner sc = new Scanner(fr);
        String line;
        String[] lineArr;

        while((line = sc.nextLine())!=null){
            lineArr =line.split(",");
            if(!lineArr[0].equals(Name)){
                tempArray.add(line);
            }
        }
    }
}
```

Figure 3.2.4, Delete Function

And code for update function in Figure 3.2.5.

```
try (FileReader fr = new FileReader("login.txt")) {
    Scanner reader = new Scanner(fr);
    String line;
    String[] lineArr;

    while((line=reader.nextLine())!=null){
        lineArr = line.split(",");
        if(lineArr[0].equals(Name)){
            tempArray.add(
                Name + ", " +
                Email + ", " +
                PhoneNumber + ", " +
                Password + ", " +
                Role);
        }else{
            tempArray.add(line);
        }
    }
}
```

Figure 3.2.5, Update Function



The Manager can also add, update and delete order by clicking the Order Management button from the Manager Page. Figure 3.2.6 below is the picture of the Order Management page.

**Manage Order Here**

Customer Name: Jack

Phone Number: 111-1111111

Address: Jalan here

Order ID:  3851

Size (kg): 8

Date & Time: Thu Nov 26 14:01:08 SGT 2020

Total Price (RM):  24.0

Status: Delivered

Customer...	Phone Nu...	Address	Order ID	Size	Date&Time	Total Price	Status
Jack	111-1111...	Jalan here	3851	8	Thu Nov 2...	24.0	Delivered
Rock	222-2222...	Jalan there	5789	7	Thu Nov 2...	20.0	Delivered
ii	111-1111...	qewqr	0164	3.0	Thu Nov 2...	9.0	Delivered
ppp	111-1111...	aaaaa	2636	34.0	Thu Nov 2...	102.0	Pending
ff	321-2134...	2e1wdw	9933	43.0	Thu Nov 2...	129.0	Pending

Figure 3.2.6, Order Management

Other than that, manager can also view the report and print receipt by clicking the Report Managing button. Figure 3.2.7 is the example of the page.

**Report Management**

Customer Name: Jack

Phone Number: 111-1111111

Address: Jalan here

Order ID: 3851

Size: 8

Date & Time: Thu Nov 26 14:01:08 SGT 2020

Total Price: 24.0

Customer N...	Phone Num...	Address	Order ID	Size	Date&Time	Total Price
Jack	111-1111...	Jalan here	3851	8	Thu Nov 2...	24.0
Rock	222-2222...	Jalan there	5789	7	Thu Nov 2...	20.0
ii	111-1111...	qewqr	0164	3.0	Thu Nov 2...	9.0
ppp	111-1111...	aaaaa	2636	34.0	Thu Nov 2...	102.0
ff	321-2134...	2e1wdw	9933	43.0	Thu Nov 2...	129.0

\*\*\*\*\*  
\* Receipt \*  
\*\*\*\*\*  
Customer Name: Jack  
Phone Number: 111-1111111  
Address: Jalan here  
Order ID: 3851  
Size: 8  
Date and Time: Thu Nov 26 14:01:08 SGT 2020  
Total Price: 24.0

Figure 3.2.7, Report Management

Lastly, manager can be able to read and delete all the feedback that entered from the customer. Figure 3.2.7 is the picture of the View Feedback page.

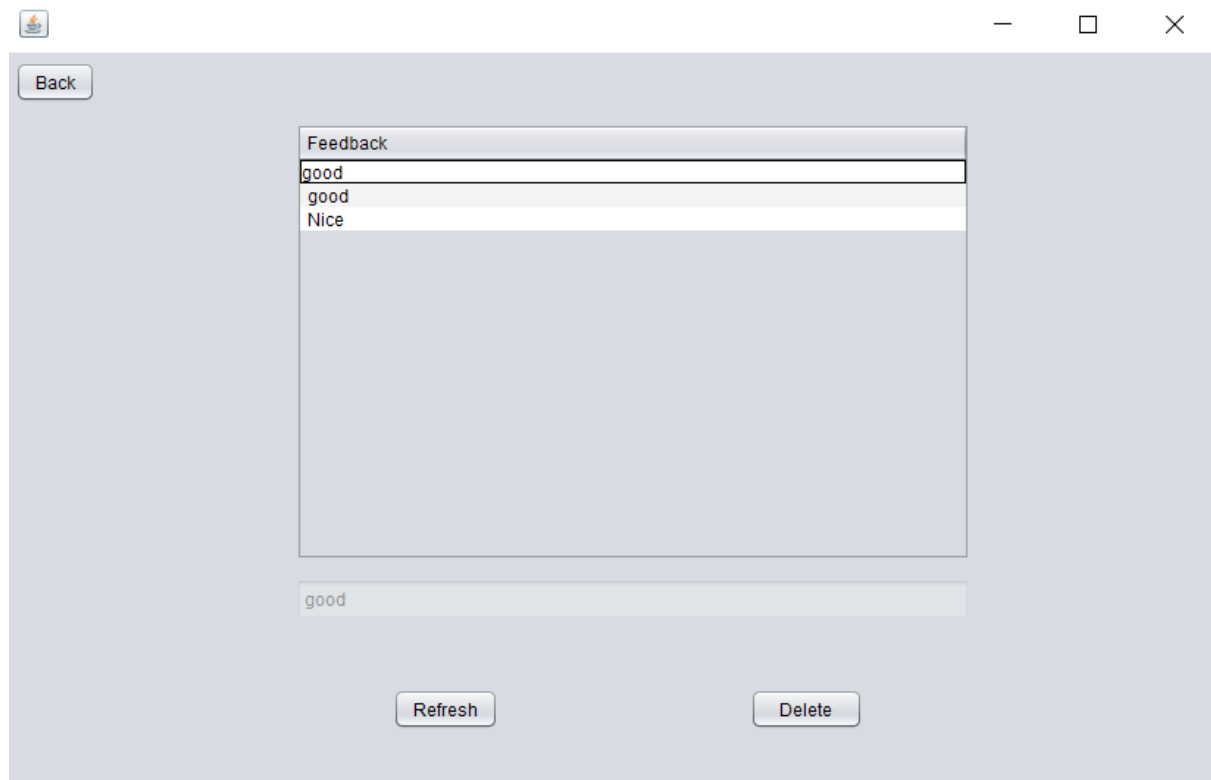


Figure 3.2.7, Viewing Feedback

### 3.3 Delivery Stuff

As usual the Delivery Stuff will have to log in first like how customer and manager log in to their account. Figure 3.3.1 is a picture of the Delivery Stuff page, the name will be appearing after login on the top right corner.

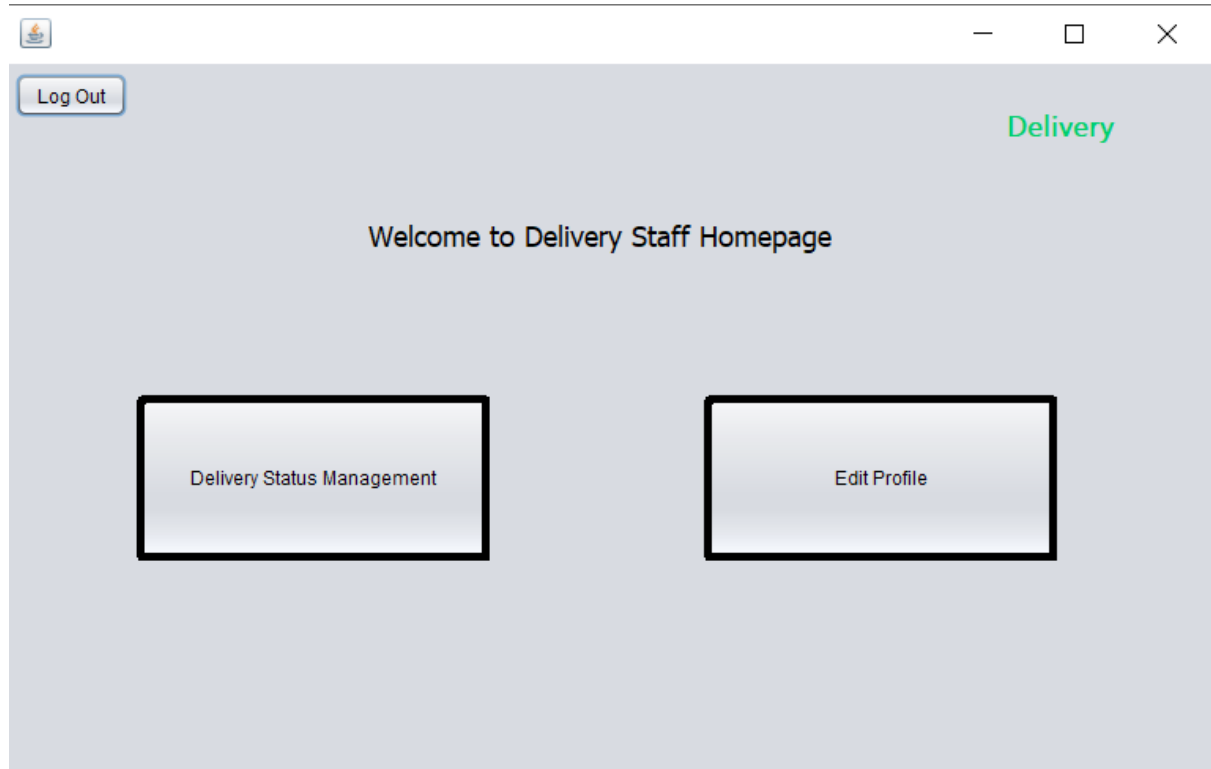


Figure 3.3.1, Delivery Staff Home Page

The delivery stuff can view all the order that has place by their manager by interacting the Delivery Status Management button. Figure 3.3.2 is what the page will look like. Delivery stuff can update their status by Delivered or Pending

Back

Customer Name

Rock

Phone Number

222-222222

Address

Jalan there

Order ID

5789

Size

7

Date

Thu Nov 26 14:01:08 SGT 2020

Price

20.0

Status

Delivered

Delivered

Pending

Delivered

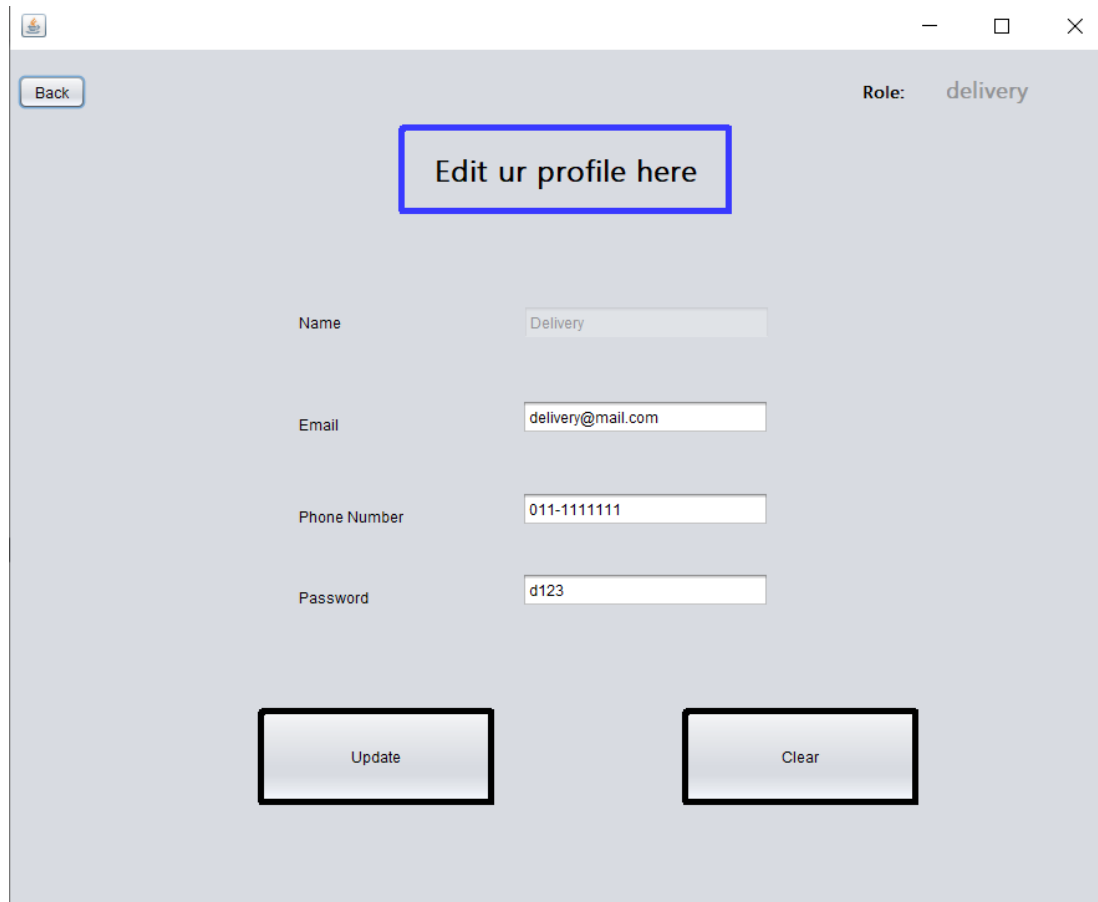
Customer...	Phone Nu...	Address	Order ID	Size	Date&Time	Total Price	Status
Jack	111-1111...	Jalan here	3851	8	Thu Nov 2...	24.0	Delivered
Rock	222-2222	Jalan there	5789	7	Thu Nov 2...	20.0	Delivered
ii	111-1111...	qewqr	0164	3.0	Thu Nov 2...	9.0	Delivered
ppp	111-1111...	aaaaa	2636	34.0	Thu Nov 2...	102.0	Pending
ff	321-2134...	2e1dw	9933	43.0	Thu Nov 2...	129.0	Pending

Save

Refresh

Figure 3.3.2, Delivery Status Management

Lastly for the Delivery Staff, they can edit their own individual account after interacting the Edit Profile button. Below in Figure 3.3.3 is the picture of editing their own individual



The screenshot shows a web application window titled "Edit ur profile here" with a blue border. In the top left corner is a "Back" button. In the top right corner, it says "Role: delivery". The form contains four input fields: "Name" with the value "Delivery", "Email" with the value "delivery@mail.com", "Phone Number" with the value "011-1111111", and "Password" with the value "d123". At the bottom of the form are two buttons: "Update" and "Clear", both with black borders.

Figure 3.3.3, Editing Individual Account

profile.

## 4.0 Description and Justification of the design and implementation codes

### 4.1 Inheritance

Inheritance is one of the object-oriented programs that allows the code to be reused in other class and maintained a connection between separate classes. In inheritance it included parent class and children class, it defined as a parent can have many children. In Java, a Super class can have many of subclasses.[ CITATION Tho17 \l 17417 ]

The object-oriented program that we had used in our login and users details class is inheritance. The login page used the login class which is the parent class of information. The home page will only get the details of name and password from the users.

```
13
14     class login extends information{
15     private String name;
16     private String password;
17
18     public String getName() {
19         return name;
20     }
21
22     public void setName(String name) {
23         this.name = name;
24     }
25
26     public String getPassword() {
27         return password;
28     }
29
30     public void setPassword(String password) {
31         this.password = password;
32     }
33
34     void setVisible(boolean b) {
35         throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
36     }
37
38     }
39
```

Figure 4.1.1, Inheritance

In the information classes which is the children class of the login, will be used in the user management class that allow the managers to collect the user information like email and phone number.

```

6 package couriersystem;
7
8 /**
9  *
10  * @author Admin
11  */
12 public class information {
13     private String email;
14     private String phonenumber;
15     private String role;
16
17     public String getEmail() {
18         return email;
19     }
20
21     public void setEmail(String email) {
22         this.email = email;
23     }
24
25     public String getPhoneNumber() {
26         return phonenumber;
27     }
28
29     public void setPhoneNumber(String phonenumber) {
30         this.phonenumber = phonenumber;
31     }
32
33     public String getRole() {
34         return role;
35     }
36
37     public void setRole(String role) {
38         this.role = role;
39     }
40 }

```

Figure 2.1.2

This is the code that used in the user management class which the information of the users was used to get and set the data.

```

private void addActionPerformed(java.awt.event.ActionEvent evt) {
    login l = new login();
    l.setName(name.getText());
    l.setEmail(email.getText());
    l.setPhoneNumber(pnum.getText());
    l.setPassword(pword.getText());
    l.setRole(role.getText());
}

```

Figure 4.1.3

## 4.2 Encapsulation

Encapsulation is an object-oriented program that combine multiple of data and convert it into a single unit. This type of object-oriented program used regularly to form of a class. The benefits of using encapsulation is to prevent the access from users to get connection with the value. However, the getter setter function is to retrieve and update the value to a certain variable within a class. [CITATION NANA1 \l 17417 ]

This is the function of encapsulation that used in our courier service system, which is to get the data of the order details and set the data to the following classes. We used encapsulation in this class is to prevent the data from accessed by other object or users.

```
12 public class Order {
13     private String name;
14     private String pnum;
15     private String addr;
16     private String orderid;
17     private String size;
18     private String date;
19     private String tp;
20
21     //constructor
22     public Order(String name, String pnum, String addr, String orderid, String size, String date, String tp) {
23         this.name = name;
24         this.pnum = pnum;
25         this.addr = addr;
26         this.orderid = orderid;
27         this.size = size;
28         this.date = date;
29         this.tp = tp;
30     }
31
32     Order(String text, String text0, String text1, String text2, String text3, String text4) {
33         throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
34     }
35
36
37     public String getName() {
38         return name;
39     }
40
41     public void setName(String name) {
42         this.name = name;
43     }
44
45     public String getPhone() {
46         return pnum;
47     }
48
49     public void setPhone(String pnum) {
50         this.pnum = pnum;
51     }
52
53     public String getAddress() {
54         return addr;
55     }
```

Figure 4.2.1

This is the code that used in the order management class that using encapsulation. The purpose of using encapsulation in this class is to update the order details or make any changes to the order.

```
515 private void updateActionPerformed(java.awt.event.ActionEvent evt) {  
516     String Name = name.getText();  
517     String Pnum = pnum.getText();  
518     String Addr = addr.getText();  
519     String Id = id.getText();  
520     String Size = size.getText();  
521     String Date = date1.getText();  
522     String tp = totalprice.getText();  
523  
524     ArrayList<String> tempArray = new ArrayList<>();  
525  
526     try {  
527         try (FileReader fr = new FileReader("order.txt")) {  
528             Scanner reader = new Scanner(fr);  
529             String line;  
530             String[] lineArr;  
531  
532             while ((line=reader.nextLine())!=null) {  
533                 lineArr = line.split(",");  
534                 if(lineArr[0].equals(Name)) {  
535                     tempArray.add(  
536                         Name + "," +  
537                         Pnum + "," +  
538                         Addr + "," +  
539                         Id + "," +  
540                         Size + "," +  
541                         Date + "," +  
542                         tp);  
543                     }else{  
544                         tempArray.add(line);  
545                     }
```

Figure 4.2.2



This is another example of encapsulation that used in the system, which is to get and set the feedback that created by the customers.

```
package couriersystem;

/**
 *
 * @author Admin
 */
public class JFeedback {
    private String feedback;

    //constructor
    public JFeedback(String fb) {
        this.feedback = fb;
    }

    public String getFeedback() {
        return feedback;
    }

    public void setFeedback(String fb) {
        this.feedback = fb;
    }
}
```

Figure 4.2.3

This object-oriented program was implemented in the Feedback class, which is to set the data that was entered by the customers. Other than that, the manager able to view and delete the record of the feedback that created by customers.

```
FileWriter file = null;
if (checkEmpty() == true) {
    JOptionPane.showMessageDialog(rootPane, "Please enter the field to submit", "Incomplete Fields", JOptionPane.ERROR_MESSAGE);
    //set cursor
    feedback.requestFocus(true);
    return;
}
else {
    try {
        JFeedback jf = new JFeedback(feedback.getText());
        file = new FileWriter("Feedback.txt", true);
        PrintWriter pw = new PrintWriter(file);
        pw.println(jf.getFeedback());
        file.close();
        pw.close();
        JOptionPane.showMessageDialog(rootPane, "Feedback Submitted!", "Add Successful", JOptionPane.INFORMATION_MESSAGE);
    }
    catch (IOException ex) {
        JOptionPane.showMessageDialog(rootPane, ex.toString());
    }
}
```

Figure 4.2.4

### 4.3 Abstraction

Data Abstraction is a property that only show the essentials of information to the users. However, the non-essentials data are not been displayed to the users. Abstraction also identified as the process of finding only the necessary attributes of an object without the unrelated of details.

In our case, we will be writing the information we want to have in our data. Below in Figure 4.3.1 is where we are creating the String information before extending the class. These information can be counted as hidden function too. To use Abstract as our object-oriented programming we have to declare it by adding the Abstract keyword in front of the public class. For an example, “Public Abstract Class Order {}”, with this abstract we can then allow other classes to being “inheritate” to the class.

```
public abstract class Order {  
    private String name;  
    private String pnum;  
    private String addr;  
    private String orderid;  
    private double size;  
    private String date;  
    private String status;  
  
    //constructor  
    public Order(String name, String pnum, String addr, String orderid, double size, String date, String status) {  
        this.name = name;  
        this.pnum = pnum;  
        this.addr = addr;  
        this.orderid = orderid;  
        this.size = size;  
        this.date = date;  
        this.status = status;  
    }  
  
    public abstract double Price();  
}
```

Figure 4.3.1, String Information

Then we created another class called Calculate to have all the other related information saved in the Calculated Java class. Below in Figure 11 is a picture of us extending the Order Class.

```
public class Calculate extends Order {  
    Calculate(String name, String pnum, String addr, String orderid, double size, String date, String status) {  
        super(name, pnum, addr, orderid, size, date, status );  
    }  
  
    @Override  
    public double Price() {  
        return getSize() * 3 ;  
    }  
}
```

Figure 4.3.2, Extending Class

## 4.4 Polymorphism

Polymorphism is a concept which we can perform a single action in different ways, the word polymorphism means many forms. Based on our system that we had created, polymorphism help us in providing calculation functions for us and at the same time providing various of string functions for another purpose in another form.

In Figure 4.4.1 below is a picture on how we implement the polymorphism object-oriented programming in our system. To call out a method we have to use the static polymorphism which have the function of overloading or overriding to provide more functions that we had stored in another class.

```
@Override
public double Price() {
    return getSize() * 3 ;
}
```

Figure 4.4.1, Override

From the picture above in Figure 4.4.1 we can see that the word on top “@Override” which mean we had already override the method in the sub class to get more function than just here. We can see this information in the Calculate Java Class calling out from the Order Java Class.

```
Double size = Double.parseDouble(s.getText());
Calculate r = new Calculate(name.getText(), pnum.getText(), addr.getText(), id.getText(), size, date1.getText(), tp.getText());
file = new FileWriter("order.txt", true);
PrintWriter pw = new PrintWriter(file);
pw.print(r.getName()+",");
pw.print(r.getPhone()+",");
pw.print(r.getAddress()+",");
pw.print(r.getOrderid()+",");
pw.print(r.getSize()+",");
pw.print(r.getDate()+",");
pw.print(r.Price()+",");
pw.println("Pending");
file.close();
pw.close();
```

Figure 4.4.2

From the picture above in Figure 13, we are using the formula of the calculation from the Calculate Java Class. We have to call out the Java Class by writing the name of your Java Class to one of the line in Figure 4.4.2. By obtaining the information from the Order Java Class we can start to calculate the prize between the size.

## 4.5 Object

Object are generally defined by as an instance of a class. An object in object-oriented programs is a self-contained component which includes of methods and properties to make a specific type of data valuable. An object can contain a data structure, function, or a variable. [ CITATION GurNA \l 17417 ]

The object that we used in the system is by creating a set of String and implemented it in a add button. For example, nm is the name of the string which is the name that variable that we get from the user input.

```
19 public class Home extends javax.swing.JFrame {  
20     public static String nm;  
21     public static String pw;  
22     public static String em;  
23     public static String pn;  
24     public static String dl;
```

Figure 4.5.1

After creating the string, we implement the code in the add button. For example, we get name and password, after the system found the user input are same with the text file database, the users may login to the system successfully.

```
try{  
    File f = new File("login.txt");  
    Scanner sc = new Scanner(f);  
    String temp;  
    boolean confirmation = false;  
    while(sc.hasNext()){  
        temp = sc.nextLine();  
        String tempArr[] =temp.split(",");  
        if(username.equals(tempArr[0]) && password.equals(tempArr[3]) && "manager".equals(tempArr[4])){  
            nm = tempArr[0];  
            pw = tempArr[3];  
            dl = tempArr[4];  
            confirmation = true;  
            JOptionPane.showMessageDialog(rootPane, "You\'re Logged In!", "Manager Login Successful", JOptionPane.INFORMATION_MESSAGE);  
            Manager m = new Manager();  
            m.setVisible(true);  
            this.dispose();  
            break;  
        }  
    }  
}
```

Figure 4.5.2

## 5.0 Extra Features 1

The extra features that we had added into this system is generate random ID number. This to prevent the users to enter duplicate or same order ID that existed in the system. However, if the order ID generate same as the existed one, an notification will pop out to show the users that this order ID is currently existed in the data, they will have to generate a random number.

```
651 private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {  
652     String contain="1234567890";  
653     Random rd=new Random();  
654     StringBuilder sb = new StringBuilder(4);  
655     for(int i=0; i<4; i++)  
656     {  
657         sb.append(contain.charAt(rd.nextInt(contain.length())));  
658     }  
659     String random=sb.toString();  
660     id.setText(random);
```

Figure 5.0.1

This is the output of the code when the users clicked “generate ID” button, which the order ID will be only represents number value. Therefore, the users will be unable to edit the textfield due to the prevention of duplicate order ID.

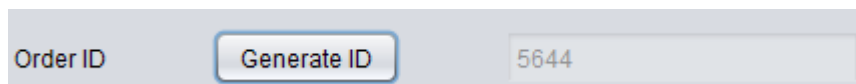


Figure 5.0.2

## 5.1 Extra Features 2

This is another extra features that the managers are able to print the receipt as pdf format or others format.

```
314 private void printActionPerformed(java.awt.event.ActionEvent evt) {  
315     try  
316     {  
317         area.print();  
318     }  
319     catch (Exception e)  
320     {  
321     }  
322 }
```

Figure 5.1.1

The users will be asked whether they would like to print this receipt at what format as they desired.

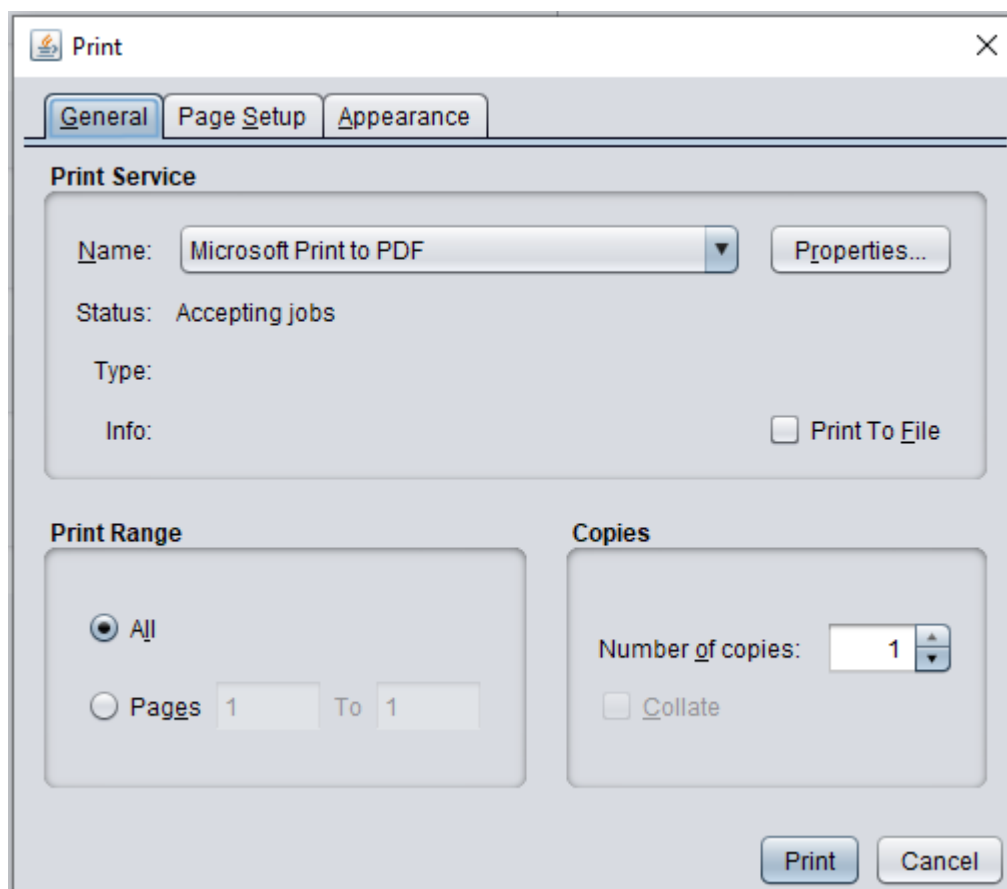


Figure 5.1.2

## 6.0 Report of object-oriented programs

Based on my research on object-oriented programming and its concepts, object-oriented programming (OOP) defined as an programming language that has been constructed around the object and classes rather than hard coding in the system, and genuine data which is more useful than encoding language. The purpose of implementing OOP in the system is to minimize the time usage of human effort rather than typing the code in a difficult way. Other than that, the main function of the used of OOP is to be focused how the way that the object transmitted and distributed the data to each class. The benefits of using OOP is that they valued the procedure of execution on each coding. The object oriented approach brought a new path and provide more prominence to the objects.[ CITATION Roo15 \l 17417 ] OOP is a transformative development in compelling developing. Moreover, based on some of the object-oriented approach were developed and created by multiple of software development knowledge that has been through several years, that conducts to the incentive of innovation of aspects. For example, hidden data, modules, closed methods, and modules. [ CITATION kaf96 \l 17417 ]

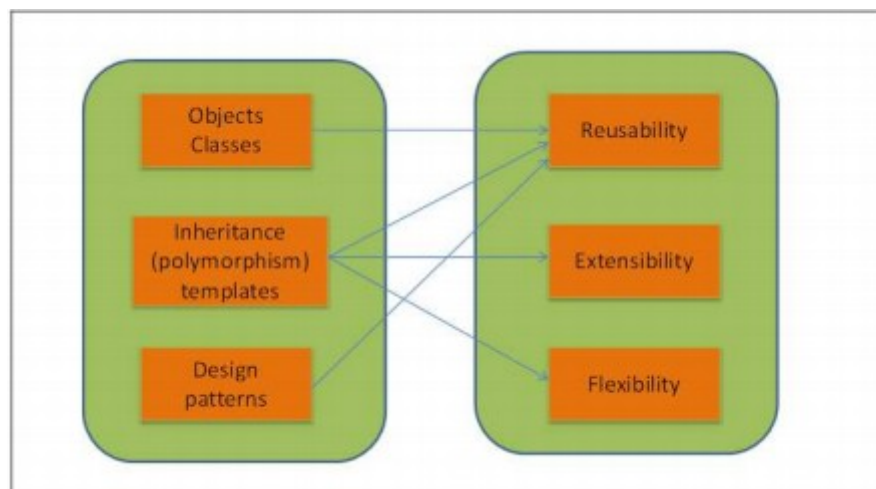


Figure 6.1, Software Engineering Goals

The benefits of using object-oriented programming is that this language is more practical and very attractive among the developers and designers. It tries to expand systems more expected developed in a way the users understand and work along. [ CITATION Anb07 \l 17417 ] Moreover, the benefits of using OOP in the system is that the developer can program the code safely without being interrupted by others part of the program, however to all the current generation it is very safe and reliable to use OOP in their system.



OOP plays an important role in JAVA programming language; it is because object-oriented programs provide a very simple and useful of design patterns which are a metaphorical configuration of the design of an object. For example, class is one of the types in object-oriented program that can gives a constructor for the users to implement different objects and it also very user approachable.

## 6.1 Inheritance

Inheritance is an object-oriented program that provides any users the obligation to apply the same code in the program or in the same class without the same variables. The procedure of creating a brand-new class from the existing main class has multiple of advantages. For example, the time usage by implementing inheritance will get lesser than hard coding in the system, the cause of error in the systems will simultaneously be lesser than using the traditional ways, as in the system will be implementing the same attributes which have been compiled by the system, and the system will not be alter by any errors. [ CITATION GMa12 \l 17417 ] Therefore, we can now build a new class using the same function from the current class with the different attributes, this could benefit the developers to work easier and faster during their coding process. However, the existing class can be treated as a reference class to build a new class as it contains the same function but only different of values. This is a type of reusability methods that the developers can reuse the same function without wasting their time and energy on unwanted place. This not only can bring easier work to the developers but also achievable and it can provide a neat figure of the classes with the proper definition when representing to the users. Inheritance also defined as the existence of an order of classes that relocate attribute to other sub-class which is like a children class that can have multiple of attribute. Inheritance are generally used on the event when transferring part from one class to another. “Specialization” is another word that can truly describe what inheritance are meaning in the object-oriented programs concept. For example, we have a class name “user information” in the system, which includes id, password, address and also phone number. If a class that requires only login information, we can move the id and password to a new class, and now the login class will only require to have only the data of id and password, which will not generally use the information other than id and password. If another login class are applied in the system, the developers will not need to apply the user information class, but only the login data, which is a more efficient and easier method to apply whenever there are nothing necessary needed.

## 6.2 Encapsulation

Encapsulation is a skill that can minimize the mutuality among all of the individually well written components by having a precise exterior interface. However, what users see is the number of aspects that are compressed in a capsule to form an object that able to function well in the system. The object provides various of services on the outside overlay by linking up however the implementation codes and information are concealed behind the background of the system. The meaning of encapsulation is to keep the complicated data behind the system while only the operation will be used on the system exterior. Therefore, the purpose of keeping data away from the users is to let them implement it without knowing what the difficult and challenging part in the system is. To define in another way, whatever the interface designed for human interaction of a system is available to every single of the system that the users are not having any clue what and how the system developed. For example, a customer bought a vehicle, they will not know how the vehicle are made and what are the vehicle made of, their responsible are just by driving the vehicle without having any concern. Therefore, the user's interface is a vital part of the system but not the developing procedure and this procedure of coding will not change the border of a human. Nonetheless, the technique that how the user combines the coding part and employs the data into a particular unit process, generally will called as the encapsulation. The features of applying encapsulation in the system are the safety and defences of data. By utilizing encapsulation, user do not have to worry about the data will be alter by other object or users, by applying getter and setter function, the data or information will be secure without any breached. Based on the analysis, this encapsulation method is suitable for big company which having a large database and system, and also data that need to be stores for a long time like membership. In conclusion, many of the developers or programmer are still not clear what polymorphism is and how to use it, but once they understand the concept and the function of this approach, they will feel a lot easier and not complicated for their future code develop.

## 6.3 Abstract

Data abstraction is one of the useful of method in object-oriented programming. The function of an abstract data entity is fully labelled by a set of abstract actions that described on the

object. The operator of an object is not required to fully well-known on how the operation of abstract implement or how to the object is characterized, the complicated part in the process are hidden behind the system. Any changes made on the representation of the object or the coding part will not interrupt the developers of the objects. For example, the specification of a computer is the complicated part of an object, like the graphic card, a processor, ram and many more. The exterior part which is the switch on and off button on the outside of the computer. The main purpose of this example is that the data abstraction approach will only show the relevant information and the others unrelated details that would confuse the users will not be handle by them as will bring much more unclear idea. In a physical scenario issues resolving, data abstraction plays an important role that it will only show essential and appropriate of description, instead of displaying the muddiness of implementation code that are not vital. In conclusion, object in most of the object-oriented program are always encapsulated, secure, and data will not be leaked. However, the peripheral interface included a set of functions that will be executed by the users. Changes that will be made to the implementations code of the object that uphold the exterior functions will not affect the exterior part of the class. This makes that the data abstract approach are well-used by the developers, as the customer details will not be altered easily.

## **6.4 Polymorphism**

Polymorphism defined as an object-oriented concept that allows an object or variable to be managed differently by data classes or types that provide an empty box that allow developer to input object. Other than that, polymorphism also provide the function to exemplify the variables that input in the class as a grouping or any other ordinary way to characterize the

variables.[ CITATION NMi15 \l 17417 ] To explain polymorphism in a much more understanding way, is that this approach allow developers that like to implement their code in a lazy or to say easier ways which allows them to spend lesser time and efforts in giving each variable or an object a much more exact distinct denotation and explanation. The code usage gives the programmer to utilize their time and produce their result rapidly. [ CITATION Tan06 \l 17417 ] It allows us or the developers to define only one particular procedure for many various types of variable to execute an action of implementation in a faster way. However, this also depends on what type of the data through the method and provides the capability to reimplement a method within a derived class. Nonetheless, we can avoid the intricacy and attach the object in a group by allowing the ordinary tasks to execute for variety of objects or variables. This can change over the traditional method by typing long and hard coding which is unnecessary, but using the polymorphism method, programmer can merge many objects together and make it easier of understanding and much more time saving rather than implementing a very convoluted of coding. Moreover, polymorphism is a class object that can belong to the same categorized tree that can have many functions with the similar name, but it can implement on different actions. For example,

```
/**
```

```
*The following source code obtained from [ CITATION edpNA \l 17417 ]
```

```
**/
```

```
Class Shape
```

```
{
```

```
Public:
```

```
Shape(){} 
```

```
Virtual void Draw(){{cout<<"Drawing a Shape"<<endl;}
```

```
};
```

```
class Rectangle: public Shape
```

```
{
```

```
public: Rectangle(){} 
```

```
virtual void Draw(){cout<<"Drawing a Rectangle"<<endl;}
```

```
};
```

```
class Triangle: public Shape
```

```
{
```

```
public:
    Triangle(){}
    virtual void Draw(){cout<<"Drawing a Triangle"<<endl;}
};
```

The following code represents a class name Shape which includes rectangle and triangle, both have different of name, but they are still under the same parent class. Both shapes can implement on different classes, but they are still under the same parent class. By providing an admittance to the users on classifying their object restrictions to specific classes, the user will get the influence to use the identical object at different locations with different of events. [ CITATION MVM89 \l 17417 ] The advantages of using polymorphism in the system is that it can help the programmers and developers to reprocess the code and classes once the application code are written, debug and tested without any errors existed. Which can allow the users to use without having any trouble or issues rather than coding individually. Other than that, a single type of variable name is allowed to be used to store variables of numerous data type like float, long, int and double. Lastly, the benefits of using polymorphism method is that can help compile a more effective and convoluted preoccupations from simpler ones. This meaning that a simple of coding can be convert into more useful of method.

## 7.0 Conclusions

Java allow us to create a system for everyone to use for their company, their shop, or any system they needed to use. In this assignment, will we be developing a system for the courier service. This system will help the staffs to work in a more convenient way such as like checking the customer, review feedback from customer, view report of the order from the

customer and many more functions that a courier service must have in their company. There are plenty of ways to create a system like the one we develop for the courier service company, sometimes developing one will be hard, messy, and even confused. With the help of the Oriented Object Programming, we will be able to develop a system easier and neatly. Coding with oriented object programming are way simpler, implement with the oriented object programming will also help strengthen the structure. There are many types of oriented object programming for an example Object and Class, Encapsulation, Inheritance, Abstraction and Polymorphism. All of those types of oriented object programming are already being implemented in the system for the courier service company. By using the system that we developed for the courier service will be more convenient than using paperwork or recording information manually.

## 8.0 References

edpresso, N.A. *What is Polymorphism?*. [Online]

Available at: <https://www.educative.io/edpresso/what-is-polymorphism>

[Accessed 26 November 2020].

Guru99, N.A. *What is Class and Object in Java OOPS? Learn with Example*. [Online]

Available at: <https://www.guru99.com/java-oops-class-objects.html>

[Accessed 20 November 2020].

Janssen, T., 2017. *OOP Concept for Beginners: What is Inheritance?*. [Online]

Available at: <https://stackify.com/oop-concept-inheritance/>

[Accessed 12 November 2020].

kafura, 1996. *Object-Oriented Programming and Software Engineering*. [Online]

Available at: <http://people.cs.vt.edu/kafura/cs2704/oop.swe.html>.

[Accessed 20 November 2020].

Laimek, R., 2015. *Internal ballistics simulation based on object oriented programming*. [Online]

Available at: <https://www.semanticscholar.org/paper/Internal-ballistics-simulation-based-on-object-Laimek-Pawgasame/1be413ad9768b5d6b5ee46ebd49869cd63d56416>

[Accessed 21 November 2020].

logic, s., N.A. *Encapsulation*. [Online]

Available at: <https://www.sumologic.com/glossary/encapsulation/>

[Accessed 21 November 2020].

M.V.Mannino, 1989. *An overview of the Object-Oriented Functional Data Language*. [Online]

Available at: <https://ieeexplore.ieee.org/document/47196>

[Accessed 26 November 2020].

Makkar, G., 2012. *Object oriented inheritance metric-reusability perspective*. [Online]

Available at: <https://www.semanticscholar.org/paper/Object-oriented-inheritance-metric-reusability-Makkar-Chhabra/091cbc8bd0b562338aaa85b1281e10cbec6154cf>

[Accessed 23 November 2020].

Milojkovic, N., 2015. *Polymorphism in the Spotlight: Studying Its Prevalence in Java and Smalltalk*.

[Online]

Available at: <https://doi.ieeecomputersociety.org/10.1109/ICPC.2015.29>

[Accessed 25 November 2020].

Pillay, A., 2007. *Object Oriented Programming*. [Online]

Available at: [http://math.hws.edu/eck/cs124/downloads/OOP2\\_from\\_Univ\\_KwaZulu-Natal.pdf](http://math.hws.edu/eck/cs124/downloads/OOP2_from_Univ_KwaZulu-Natal.pdf)

[Accessed 22 November 2020].

Yu, T. Y., 2006. *Architectural Support on Object-Oriented Programming in a JAVA Processor*. [Online]

Available at: <https://ieeexplore.ieee.org/document/4019533>

[Accessed 25 November 2020].



