Object Oriented Visual Programming

Mini Project

SuperMarket Management System

LECTURER

Deffa Rahadiyan

Group Member:

Michael Tampubolon (001202300151)

Muhammad Rafid Dypo Maulana (001202300051)

Ribathullah Ahmad Yasin (001202300028)

PRESIDENT OF UNIVERSITY

Jl. Ki Hajar Dewantara, Kota Jababeka, Cikarang Baru, Bekasi

17550-Indonesia Phone (021) 8910 9762-6. Fax (021) 8910 9768

www.presuniv.com | Email: presuniv@gmail.com

Application Description

a.Supermarket inventory management system Description

The Supermarket Inventory Management System is a comprehensive digital solution tailored to assist supermarket owners in efficiently recording and managing their products and customers. This system empowers users to transition from manual record-keeping to a streamlined digital approach, enhancing productivity and accuracy in their daily operations. Key features include the ability to swiftly create, update, and delete product records, enabling users to maintain a well-organized inventory database. With the Supermarket Inventory Management System, gone are the days of manual data entry on paper, as the system facilitates a faster and more efficient recording process. By leveraging this solution, supermarket owners can optimize their workflow, reduce errors, and focus on providing exceptional service to their customers.

B. Supermarket Inventory Management System Problem Solve

The Supermarket Inventory Management System addresses the challenges faced by supermarket owners and other users in need of an efficient recording system for their businesses. Traditionally, many supermarkets relied on manual methods involving paper and pencil to manage their inventory, resulting in inefficiencies and inaccuracies in their records. However, with the implementation of the Supermarket Inventory Management System, users can overcome these obstacles.

This system streamlines the recording process, significantly reducing the time and effort required to maintain product records. By transitioning from manual to digital recording methods, users can enjoy faster data entry and retrieval, leading to enhanced productivity and operational efficiency. Moreover, the Supermarket Inventory Management System ensures data accuracy by minimizing human errors commonly associated with manual data entry.

Providing a user-friendly interface and robust functionalities, the system empowers users to keep their inventory data up-to-date and easily accessible. With real-time updates and comprehensive reporting features, users can make informed business decisions and optimize their inventory management

processes. Overall, the Supermarket Inventory Management System serves as a valuable tool for supermarkets of all sizes, enabling them to streamline operations, reduce errors, and improve overall productivity.

c. Supermarket Inventory Management System Uniqueness

Supermarket inventory management system offering simplicity and ease of use for recording and managing user's products and customers. This system is best suited for startup supermarkets seeking a straightforward solution to record and manage their inventory. With this system, the cumbersome task of manually recording products on paper becomes obsolete. Users can effortlessly manage their inventory and customer data directly from their devices, ensuring a safer and more efficient process.

Implementation of OOP

OOP pillars is a principles that used in java programming to help manipulate and managing object so it will make the process easier and effective. There are 4 principles in oop, which is Abstraction, Polymorphism, Encapsulation, and Inheritance.

a.Encapsulation

Encapsulation It refers to the bundling of data (attributes or properties) and methods (functions or procedures) that operate on the data within a single unit or class. Encapsulation hides the internal state of an object from the outside world and only exposes the necessary functionality through well-defined interfaces.

For example class called "Car" in your program. Encapsulation means that you hide the inner workings of the "Car" class from the outside world. For instance, instead of directly accessing the variables inside the "Car" class, like its speed or fuel level, you provide methods like "getSpeed()" and "refuel()" to interact with these variables. This way, the implementation details of how speed is calculated or how fuel is refilled are hidden from the user of the "Car" class, promoting better organization and reducing the chances of unintended errors.

In this mini project the encapsulation principles can be seen in the product1.java where the product is being recorded. In the product class the attributes id, name, quantity, price and category are declared with the private access modifier.

```java
class Product {
    private int id;
    private String name;
    private int quantity;
    private int price;
    private String category;

    public Product(int id, String name, int quantity, int price, String category) {
        this.id = id;
        this.name = name;
        this.quantity = quantity;
        this.price = price;
        this.category = category;
    }

    // Getters and setters
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }
```

```
64      public int getPrice() {
65          return price;
66      }
67
68      public void setPrice(int price) {
69          this.price = price;
70      }
71
72      public String getCategory() {
73          return category;
74      }
75
76      public void setCategory(String category) {
77          this.category = category;
78      }
79  }
```

This makes them accessible only within the class and its subclasses, ensuring that their state is controlled by methods defined within the class.

b.Inheritance

Inheritance in OOP allows a new class (called a derived class or subclass) to inherit the attributes and methods of an existing class (called a base class or superclass). This means the subclass can reuse code from the superclass, promoting code reuse and maintaining a hierarchical structure.In this mini project, inheritance principles can be seen between the item class and the product class which extend the attribute.

For example, consider a superclass called "Animal" with attributes like "species" and methods like "eat()" and "sleep()". Now, you might have subclasses like "Dog" and "Cat" that inherit from the "Animal" class. These subclasses can add their own unique attributes and methods, like "bark()" for the "Dog" class and "meow()" for the "Cat" class, while still inheriting the common behaviors and properties from the "Animal" class.

For example in this mini project Inheritance it show in product1.java

```
81      public class product1 extends javax.swing.JFrame {
```

In this code it defines a class named "product1" that extends (or inherits from) the class "javax.swing.JFrame". In Java Swing, "JFrame" is a class that represents a window with decorations such as a title bar, borders, and buttons for closing, minimizing, and maximizing. By extending "JFrame", the "product1" class inherits all the properties and methods of the "JFrame" class, allowing you to customize and manipulate the window as needed.

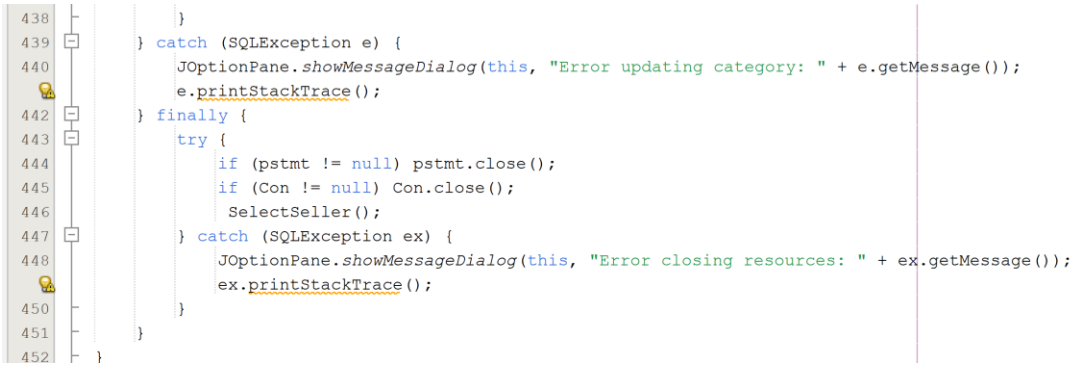
c.Polymorphism

Polymorphism is a fundamental concept in object-oriented programming (OOP) that allows objects of different types to be treated as objects of a common superclass. It enables methods to do different things based on the object that they are acting upon.

Consider a scenario where it have a base class named "Shape" with subclasses like "Circle" and "Rectangle". Each subclass might have its own implementation of a method called "calculateArea()". When you call "calculateArea()" on an object, the specific implementation executed depends on the actual type of the object, not just its declared type. This means that despite invoking the same method name, different objects may execute different code paths, tailored to their respective classes. This flexibility enhances code reusability, promotes cleaner design, and allows for easier extension of functionality. Polymorphism enables developers to write more generic and maintainable code, where behavior can adapt dynamically based on the context in which it is invoked.

In this mini project, polymorphism can be seen from the CustomerDAO class and DatabaseAccess class. The populateCustomerTable method overrides the select method from its superclass DatabaseAccess. This is indicated by the @Override annotation, which tells the compiler that the method is intended to override a method in the superclass.

```
    private void jButton2MouseClicked(java.awt.event.MouseEvent evt) {
420 if (CatId.getText().isEmpty() || CatName.getText().isEmpty() || CatDesc.getText().isEmpty()) {
421 JOptionPane.showMessageDialog(this, "Missing Information");
422 } else {
    Connection Con = null;
424 PreparedStatement pstmt = null;
425 try {
426     Con = DriverManager.getConnection("jdbc:mysql://localhost:3306/supermarketdb", "root", "");
427     String query = "UPDATE categorytbl SET CatName=?, CatDesc=? where CatID=?";
428     pstmt = Con.prepareStatement(query);
429     pstmt.setString(1, CatName.getText());
430     pstmt.setString(2, CatDesc.getText());
431     pstmt.setString(3, CatId.getText());
432
433     int rowsAffected = pstmt.executeUpdate();
434     if (rowsAffected > 0) {
435         JOptionPane.showMessageDialog(this, "Category Updated");
436     } else {
437         JOptionPane.showMessageDialog(this, "Failed to update category");
```



```
438         }
439     } catch (SQLException e) {
440         JOptionPane.showMessageDialog(this, "Error updating category: " + e.getMessage());
            e.printStackTrace();
442     } finally {
443         try {
444             if (pstmt != null) pstmt.close();
445             if (Con != null) Con.close();
446              SelectSeller();
447         } catch (SQLException ex) {
448             JOptionPane.showMessageDialog(this, "Error closing resources: " + ex.getMessage());
            ex.printStackTrace();
450         }
451     }
452 }
```

This code snippet demonstrates polymorphism in Java, specifically method polymorphism. Polymorphism allows a method to perform different actions based on the object that invokes it. In this case, the method jButton2MouseClicked is polymorphic because it can behave differently depending on the event (MouseEvent in this context).

The jButton2MouseClicked method is overridden to handle mouse click events. When the mouse button is clicked, this method gets invoked. This is an example of method overloading where the same method name (jButton2MouseClicked) is used for different types of events, but with different parameter types (in this case, MouseEvent evt).

Inside the jButton2MouseClicked method, there's a conditional statement (if-else) that checks if certain text fields (CatId, CatName, and CatDesc) are empty. Depending on whether they are empty or not, different actions are taken. This demonstrates conditional polymorphism, where the method behaves differently based on the conditions met during runtime.

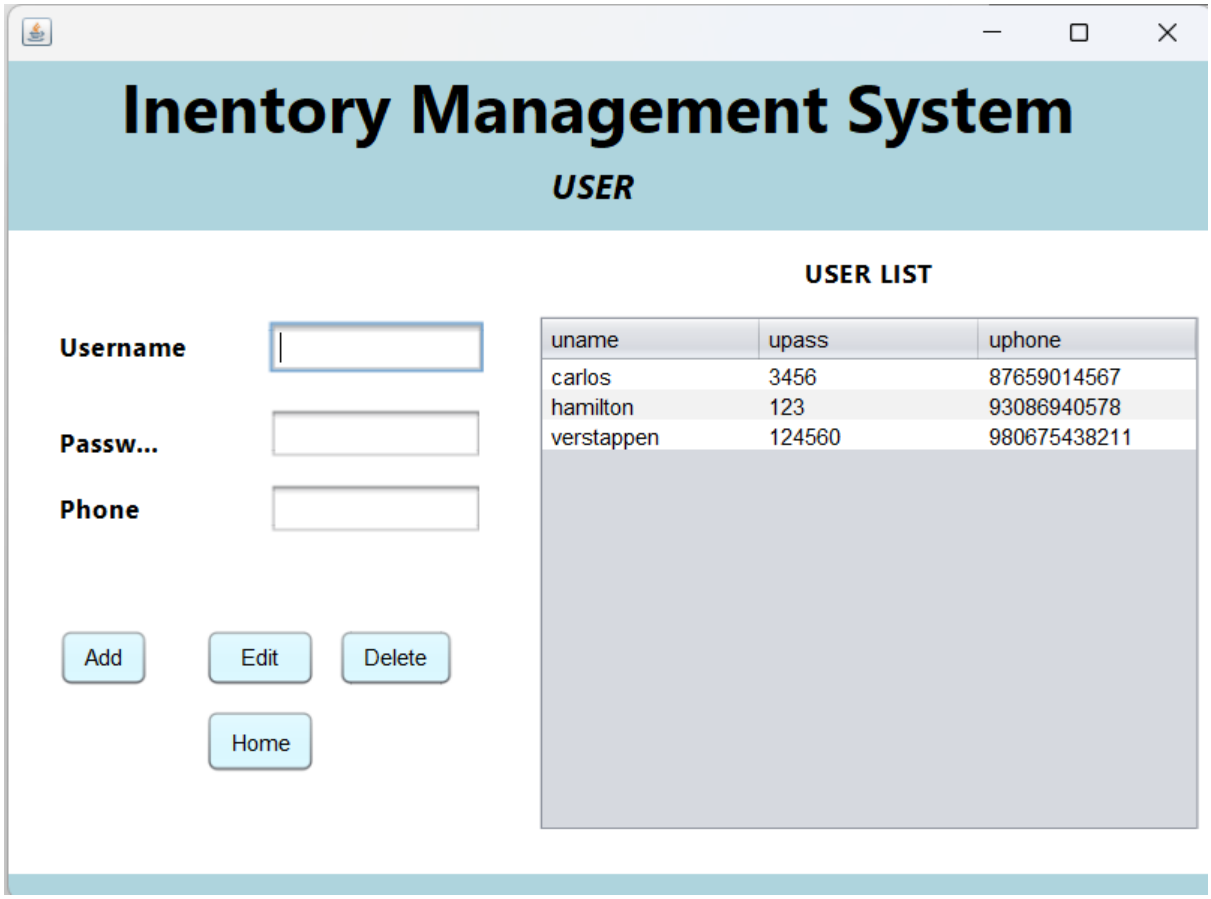
Within the try block, there's a call to executeUpdate() method of the PreparedStatement object (pstmt). This method executes an SQL statement in the database. The specific behavior of executeUpdate() depends on the underlying database state and the SQL statement provided. This is an example of dynamic method invocation, where the behavior of a method is determined at runtime based on the actual object it operates on.

d.Abstraction

Abstraction refers to the process of hiding the complex implementation details and showing only the essential features of an object. It allows developers to focus on what an object does rather than how it does it. Abstraction is achieved through abstract classes and interfaces, which define a blueprint for classes to follow without providing specific implementations.

Take example of abstract class "Vehicle" with abstract methods like "start" and "stop." Concrete subclasses like "Car" and "Motorcycle" implement these methods according to their specific functionalities. Users interact with vehicles through these abstract methods without needing to know the intricate details of how each vehicle type starts or stops.

In this mini project, abstraction principle are implemented in the button that interact with the user. This system hide the detail of each button such as add, edit, delete, and home from user, so the user don’t have to know how the buttons are working.



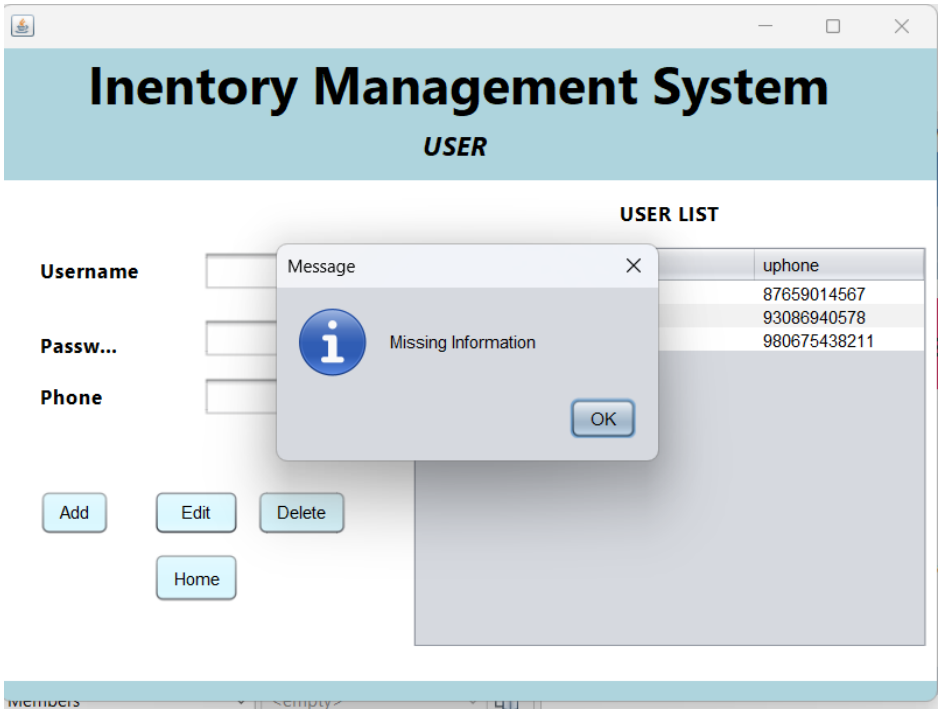
From the image user only need to type and click the button to insert, update or delete the data, without have to know how's the button working.

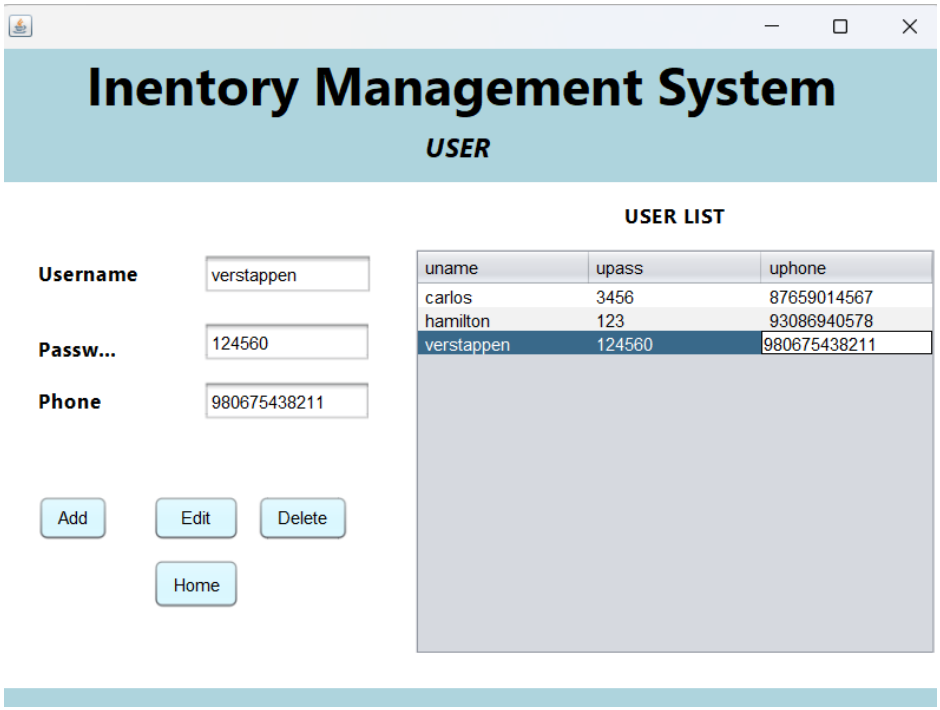
Meanwhile on the other side, these are the hidden code for making the buttons for the user.

```
527
    private void deletebtnMouseClicked(java.awt.event.MouseEvent evt) {
529         if (prodid.getText().isEmpty()) {
530             JOptionPane.showMessageDialog(this, "Select item to delete");
531         } else {
532             try {
533                 int id = Integer.parseInt(prodid.getText());
534                 productDAO.deleteProduct(id);
535                 SelectProd();
536                 JOptionPane.showMessageDialog(this, "Product deleted successfully");
537             } catch (NumberFormatException e) {
538                 JOptionPane.showMessageDialog(this, "Invalid input for product ID");
539             }
540         }
541     }
542
```

```
479
    private void addbtnMouseClicked(java.awt.event.MouseEvent evt) {
481         try {
482             Product product = new Product();
483             product.setId(Integer.parseInt(prodid.getText()));
484             product.setName(prodname.getText());
485             product.setQuantity(Integer.parseInt(prodqty.getText()));
486             product.setDescription(proddesc.getText());
487             product.setCategory(prodcat.getSelectedItem().toString());
488             productDAO.addProduct(product);
489             JOptionPane.showMessageDialog(this, "Product successfully added");
490             SelectProd();
491         } catch (NumberFormatException e) {
492             JOptionPane.showMessageDialog(this, "Invalid input for product ID or quantity");
493         }
494     }
495
```

In this code the button for delete and edit are checking the Jtextfield first, is it empty or not. If the text field is empty the system will prompt user to select the record first from the tables.





After getting the id from the text field, the code will call the deleteProduct method to delete the record from the database.

```
public void deleteProduct(int productId) {
    try (PreparedStatement preparedStatement = connection.prepareStatement("DELETE FROM product WHERE prodid=?")) {
        preparedStatement.setInt(1, productId);
        preparedStatement.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

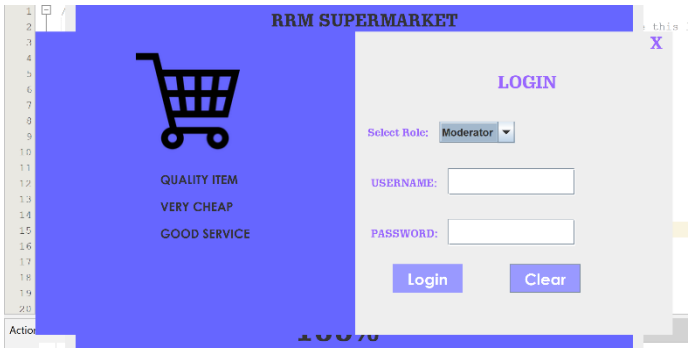
Program Explanation

Supermarket inventory management system are a program that consist product managing, customer managing, user, and category managing. This program aid the customer and cashier to record the business's product. The program consist of splash.java, login.java, selling.java, product.java, category.java, seller.java, and seller.java.

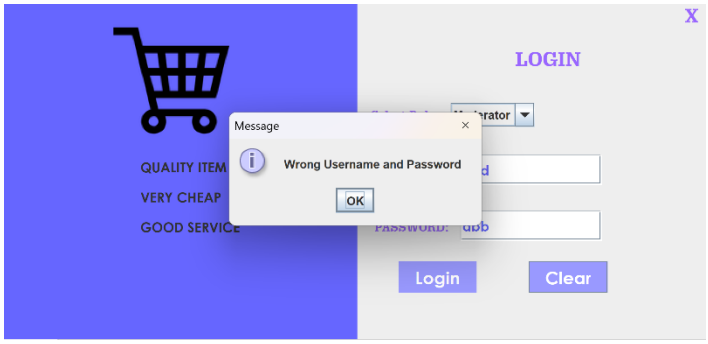
Splash.java

This file act as a starter for user when opening the program. After the start.java user will redirect to the login.java where user need to login first so they could access the dashboard.

Login.java

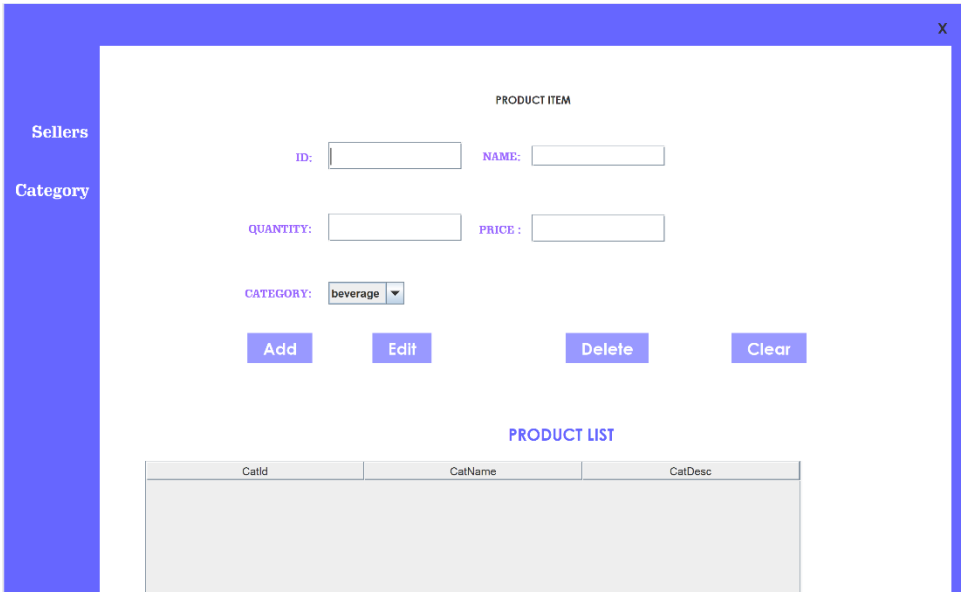


In this page there are text field for user to fill their name and password. There's also two button, the login button and clear button. The login button need to be click after user inserting the data, meanwhile the clear button is used to clear all the text field if it necessary. The user will have to choose between becoming moderator, or and seller which is the customer



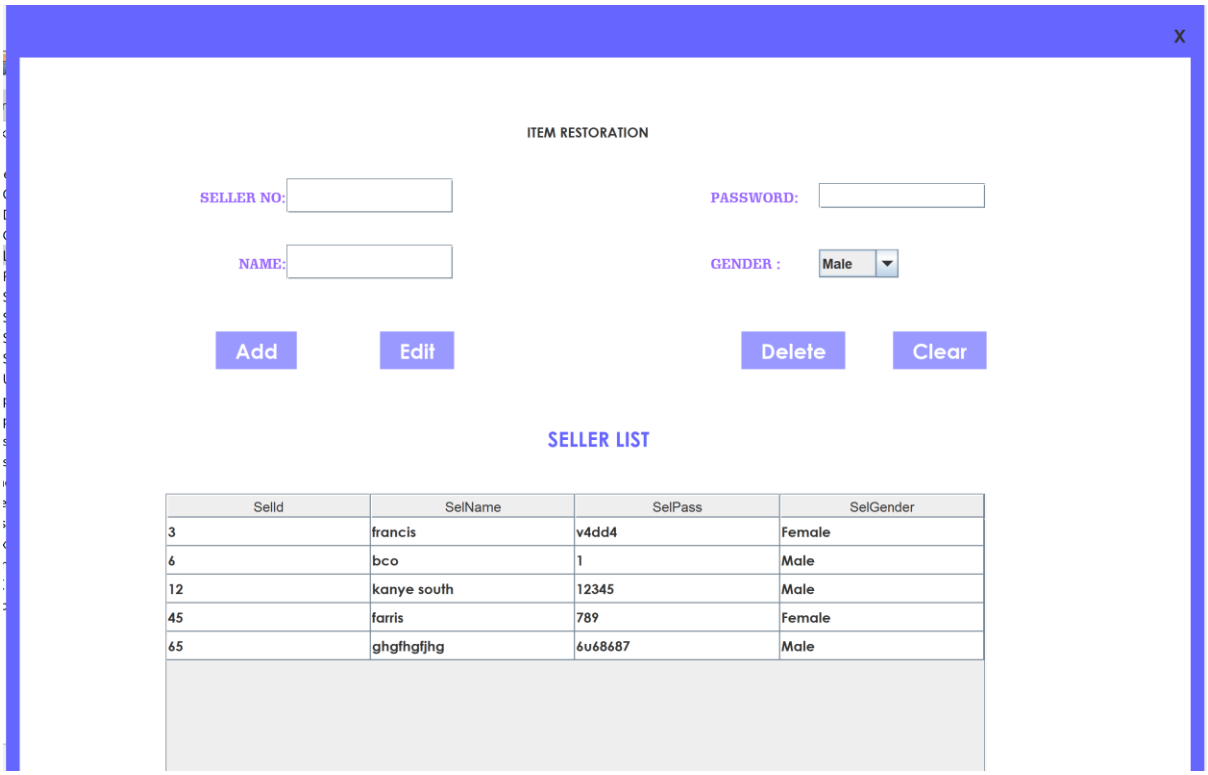
If user success to login, the user will redirect to the dashboard where there are several option in there.

Product1.java



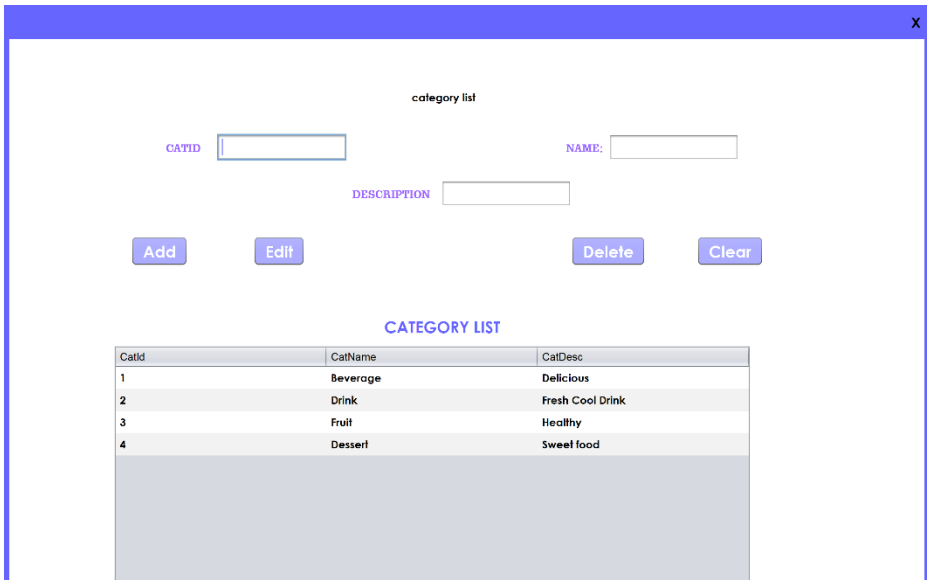
After choosing the correct username and password the user will be then directed into the Product1,In this page user are offered with four option to choose, add, edit, delete, and clear. Each of it will be put into the table down below to other file where user could insert a record corresponding to the option. The product icon will redirect user to Product1.java where user can Add, Edit, and Delete the product information. The user will then have the choice to stay in this program or choose to a different program in that page. The user have a choice between, Sellers or Category. Each have different outcome

Seller.java



Upon selecting the "Seller" button, users are seamlessly directed to the seller account page, where they encounter a user-friendly interface designed for optimal efficiency. Here, easily accessible input buttons streamline integration with the table, facilitating swift and organized customer management. Users are empowered with a range of options including adding, editing, deleting, or clearing entries within the list, affording them heightened control over data management processes.

Categories.java

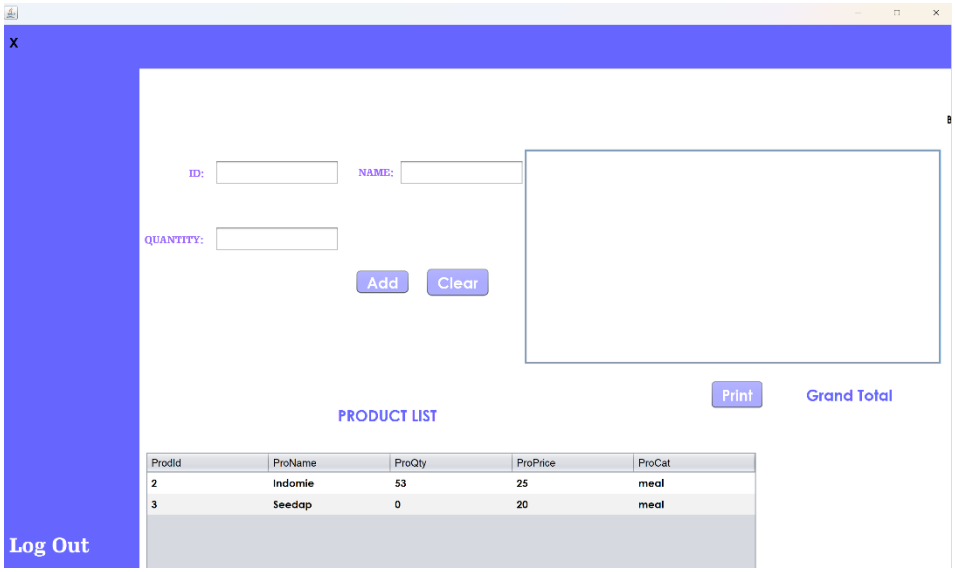


Upon selecting a different option, users will smoothly navigate to the categories list, a vital component for organizing products effectively. In this interface, users will find user-friendly tools that make managing categories straightforward.

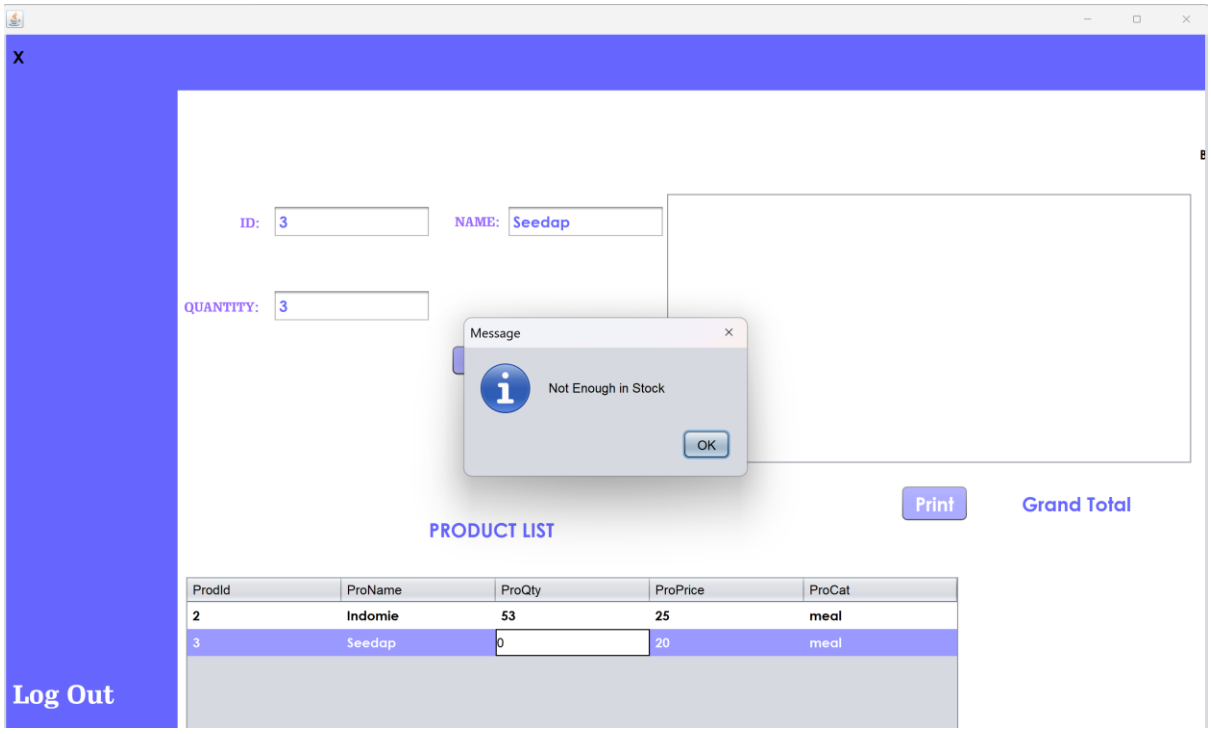
Here, users can easily add, edit, or delete categories as needed. They also have the option to introduce new categories or clear existing ones. This flexibility ensures that users can adapt the category list to their changing needs and preferences.

In summary, the categories list simplifies product organization, allowing users to maintain a well-structured inventory with ease and efficiency.

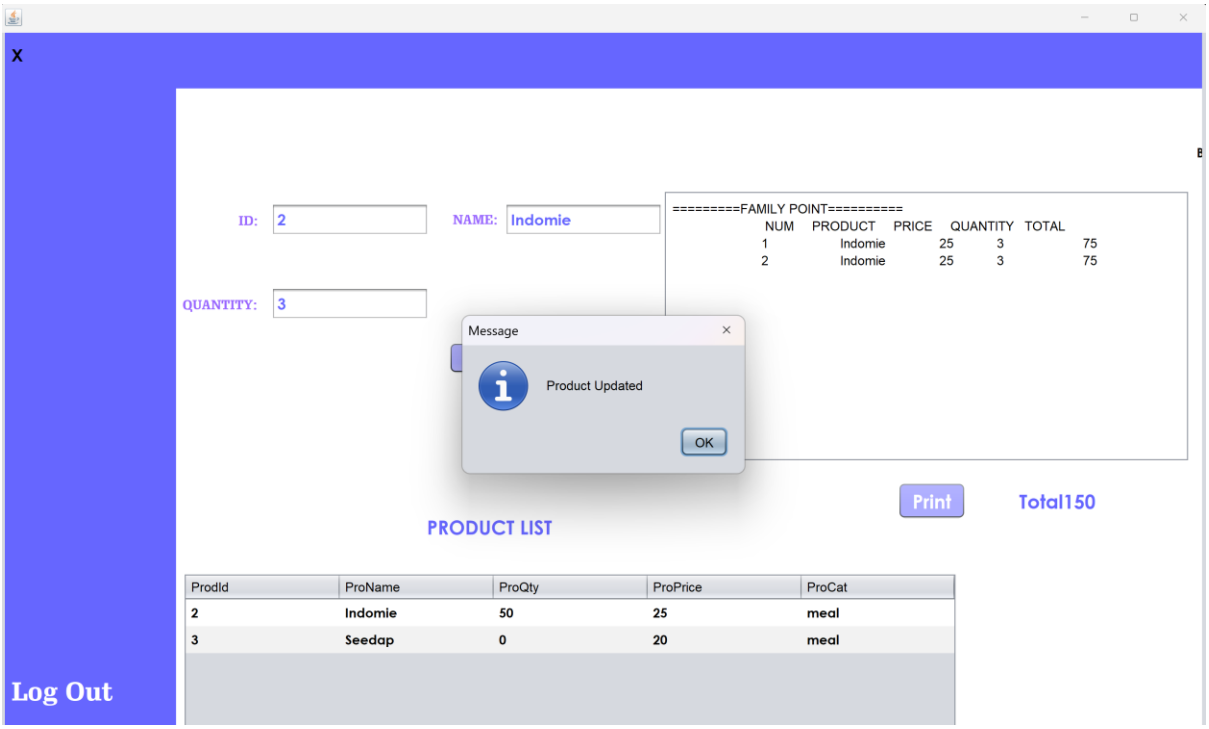
Selling1.java



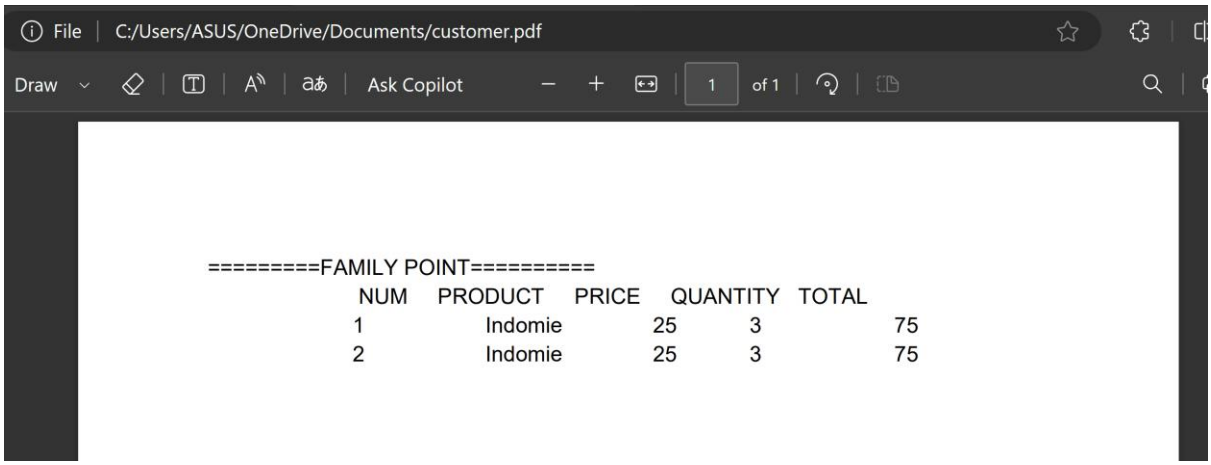
Here the user can manage what product the customer gonna buy, the user can seen the product list that available in product if the product quantity is zero it will get a message of “not enough in stock”



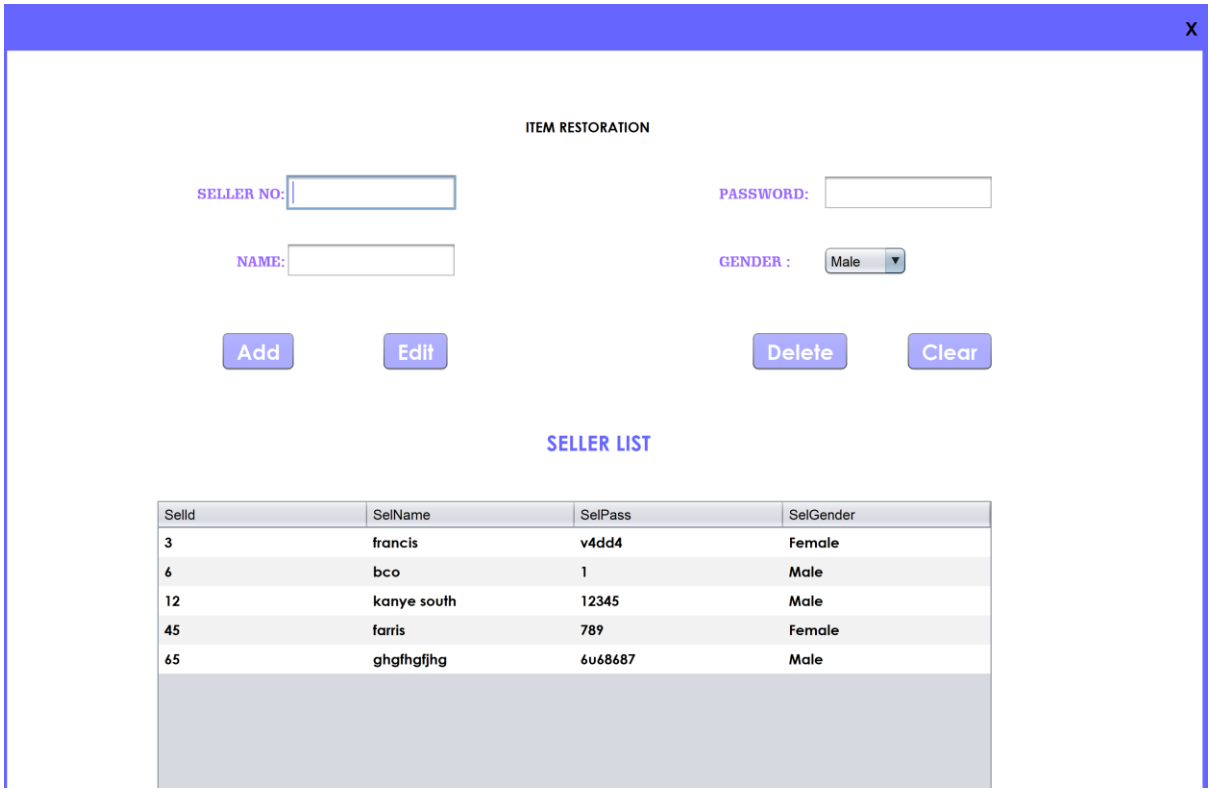
If the product quantity it will added to the bills



After the user finished tha all product of customers want user can easily print the customer history

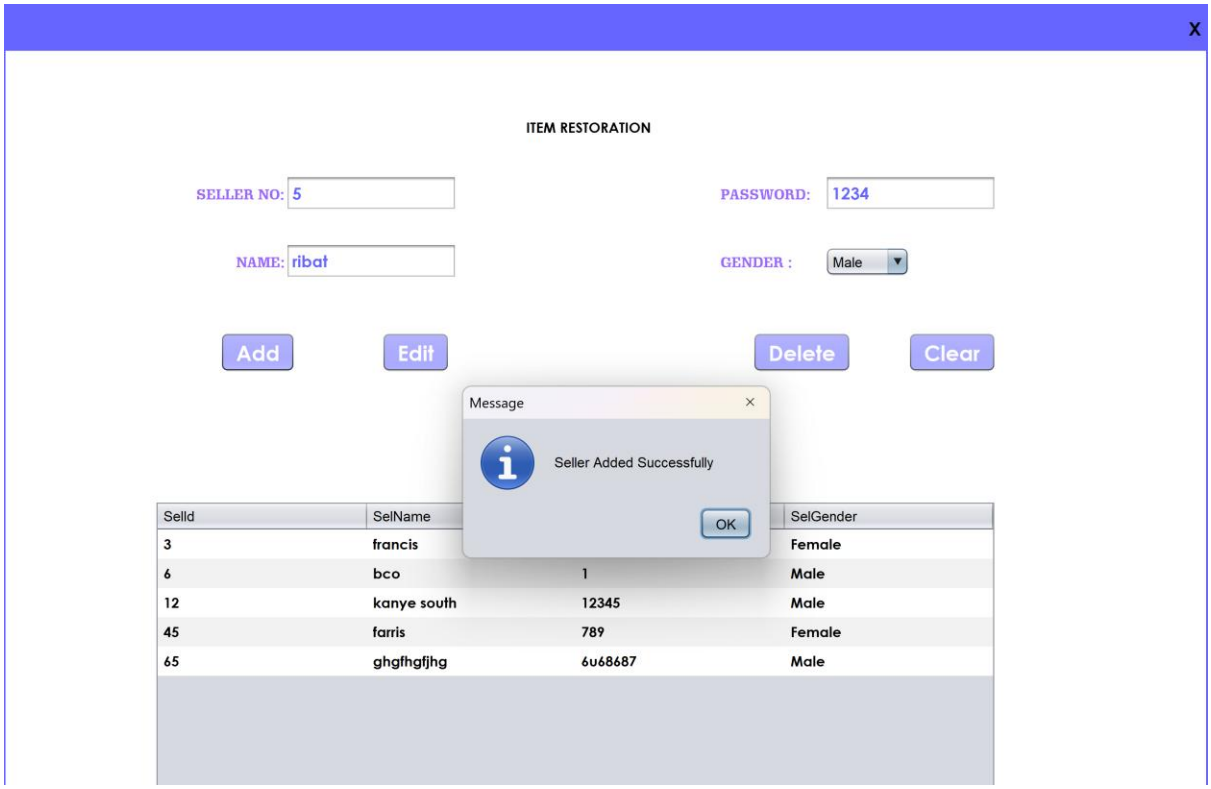


Seller.java



In this page user can manage their own information to login by adding user information such as their name, their own preferred password and their gender. This page consist of three text field, on combo box, four buttons and one table showing the user data such as seller name, seller password, and seller gender.

Same as the previous page if user want to insert new data they need to fill in the text field first and then click the add button to add the data to the database. To edit and delete the data user need to choose the data first from the table and then the data can edited or delete. To save the new data just click edit button and to delete data just click the delete button. There will be also a message pop up if the text field is not yet filled or the data is successfully added, edited, or deleted.



To go back to the dashboard, user could simply click the home button to select other option in the dashboard.