



PROJECT
**SALES, PURCHASE AND RECEIPT MANAGEMENT
SYSTEM.**

**OBJECT ORIENTED
PROGRAMMING**

**Mr. MUZAFAR
IQBAL.**

GROUP MEMBERS:

Khola Gohar

(SP22-BSE-022)

Eeman Basharat

(SP22-BSE-011)

Wajahat Ali Tahir

(SP22-BSE-051)

TASK DISTRIBUTION:

The whole system has been developed by equal contribution from all the members. In this regard proper sessions were convened to discuss the ongoing work on this project. Every member participated in it with a zeal to learn new things and to have hands-on Object-Oriented Programming and especially new things like File Handling and GUI.

PROJECT DESCRIPTION:

Our system, “**Sales, Purchase and Receipt Management**”, aims to address all the challenges faced by local cash and carry businesses, pharmacies, and shopping malls in their daily sales, purchase, and product management. Our system is designed to streamline operations, reduce errors, and increase profits for businesses. With our system, businesses can efficiently manage their inventory, set pricing, manage customer information, and analyze data. The system is user-friendly and can be customized to suit the specific needs of each business. Our system is equipped with a dashboard (GUI) that provides real-time data on sales, inventory levels, and customer operations.

MAIN ENTITIES IN THE SYSTEM:

The possible entities that exist in our Sales, Purchase, and Receipt Management System are as below. These specific entities and their attributes may vary depending on the business requirements and the scope of the system.

1. **Person:** The person class represents itself as a parent entity from which many other entities are inheriting attributes like Customer, Owner, Salesperson, and Supplier. The person class has the attributes like name, phone, and email address.
2. **Customer:** This entity represents the individuals who purchase products or services from the business. It can include information such as name, contact details, and purchase history.

3. **Owner:** This entity represents the individuals that can perform the operations like check for total sales, check for all the salespersons in store, add salespersons, delete salespersons, check all the products in the store, add any product, delete any product, view his profile and modify his account etc.
4. **Salespersons:** This entity represents the individuals responsible for selling products or services to customers. It can include information such as name, contact details. This class has the methods like to initialize the purchase, show products available and modifying his profile etc.
5. **Suppliers:** This entity represents the individuals or organizations who supply products or services to the business. It can include information such as name, specific ID and contact details.
6. **Store:** This entity represents the stock of products and it includes all the related information including products, the number of Sales Persons in store and the information of the store like name and address as well.
7. **Product:** This entity represents the items or services that the business sells. It can include information such as product name, description, price, stock, weight per item, and supplier information as well.
8. **Invoice/Receipt:** This entity represents the billing statements or receipt provided to customers for their purchases. It can include information such as invoice number, date, customer details, products purchased, quantity, price, total cost and Salesperson.

FEATURES:

The most important and significant features of our Project “**Sales, Purchase and Receipt Management System**” are as below:

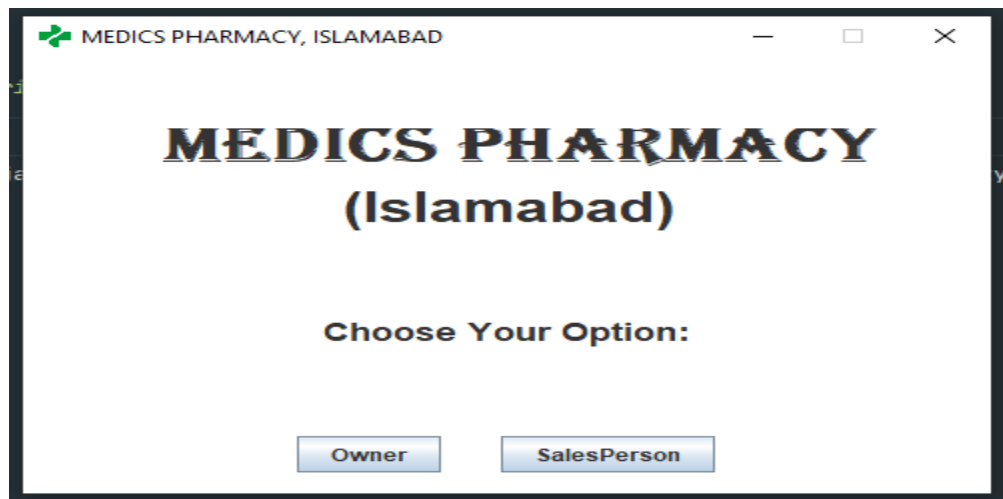
1. **Inventory Management:** To manage product inventory, including stock levels, prices, and product details, the system offers a centralized database. It enables simple inventory tracking and sends warnings when stock levels are low, ensuring prompt replenishment.
2. **Processing of Sales Orders:** The system makes it possible to create and control sales orders. New sales orders can be created by Salesperson, who can also add products,

define quantities, and compute totals. Invoices are generated for orders that have been completed, and it also makes it easier to follow orders from creation through fulfilment.

- 3. Management of Purchase Orders:** The system enables users (owners) to create and control purchase orders. It helps with choosing a vendor, ordering goods, and following purchase orders from creation to completion. It also helps manage supplier relationships and offers real-time notifications on the status of purchase orders.
- 4. Management of Customers and Vendors:** The system keeps an extensive database of Customers, Salespersons and Vendors. Users have access to and storage for crucial data such contact information, payment method and customers record and transaction history.

MAIN FUNCTIONALITIES

The program when run, opens the main Window of the system having the Store name and options for both Owners and Salespersons as depicted in figure below:



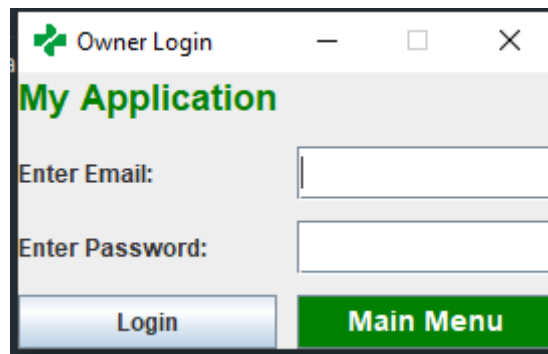
This system is designed in such a way that it can be used by the owner of the Store as well as the salespersons within store to manage the inventories. For this purpose, separate graphics interfaces have been developed. Let us we discuss the functionalities of the Owner Class and Salesperson classes in details:

OWNER OPTIONS:

The owner can have following options in his windows i.e.:

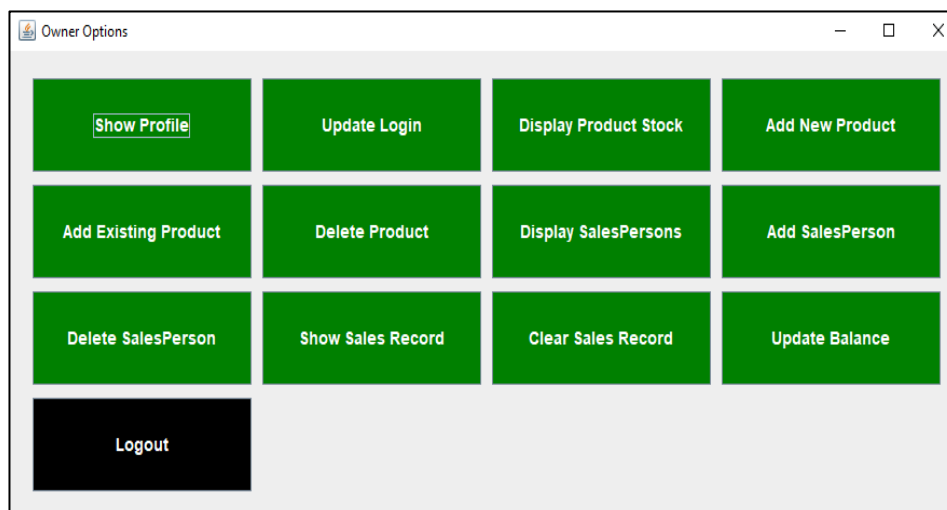
1) Login:

As soon as the owner selects or clicks the Owner Button on main Frame or main window, the system directs him to a new window pane asking for his login Email and login password as shown below. If the password and email match with the owner's record the system logs in otherwise it shows an error message and keeps the user at same window. Moreover, the user has the option to move back to the main Window. i.e.:



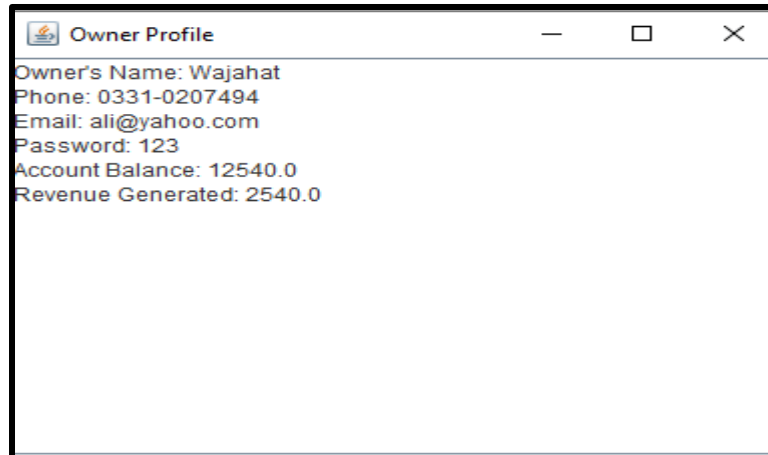
2) Logout:

After the successful login the system takes the owner to a new window showing Owner options. Here the user can find the logout button to logout of his account and move back to login window i.e.:



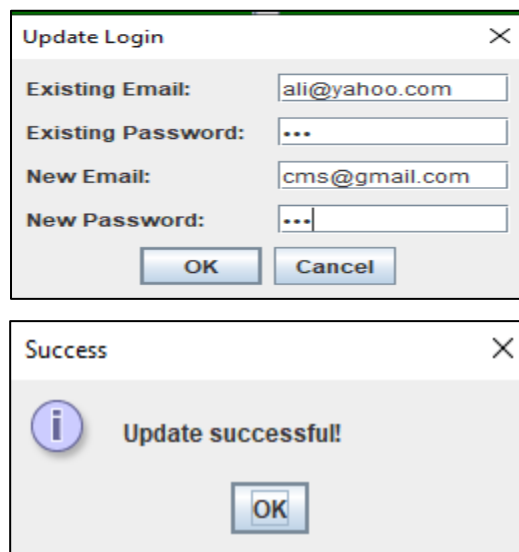
3) Show Profile:

In the Owner Option window, the owner can find a button Show Profile to view his profile . It displays the whole data of Owner including his name, phone, email, account balance, total revenue generated as well as his login password in a separate panel or window i.e.:



4) Update Login:

If the user or owner clicks the button named Update login from Owner Options, the system prompts the owner to enter his existing email ,existing password and new email and new password. The system then calls the method from Owner class through Interface. In that method if the users existing email and password match with his data in profile the password and email are updated to new email and password i.e.:



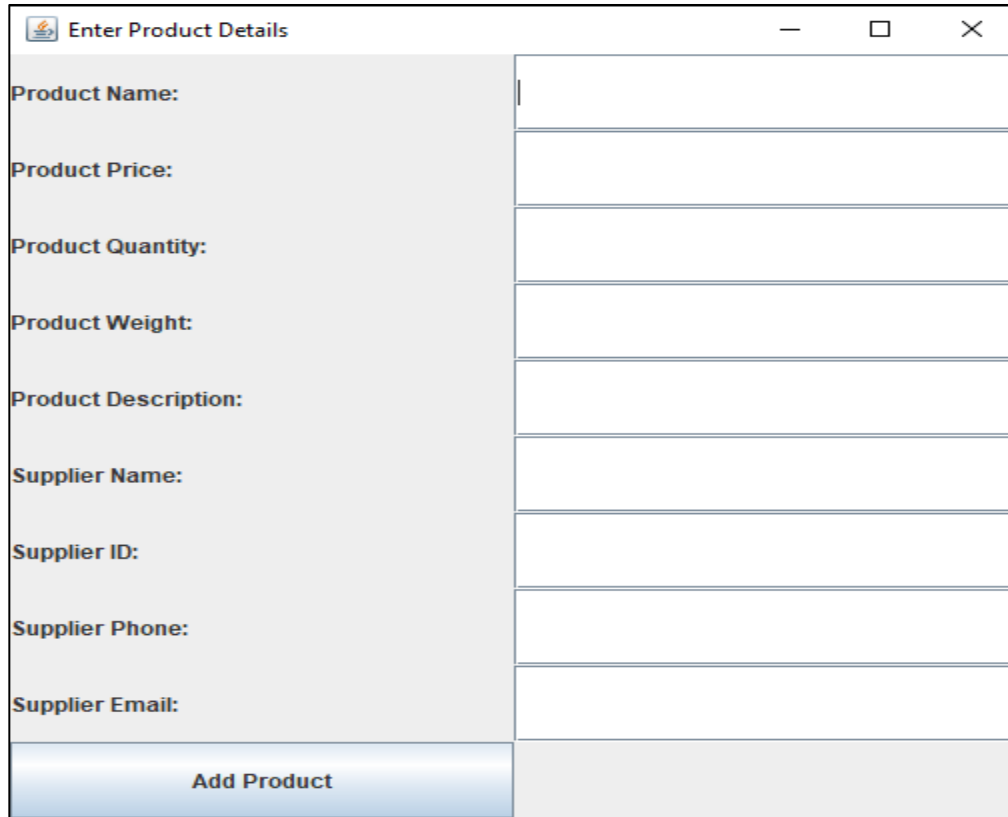
5) Display Product Stock:

If the user clicks the button Display Products, the system through action listeners calls the method in Utility class to display all the available products on a new window or console i.e.:



6) Add New Product:

In the same way if the user wants to add a new product in the stock, he will simply click the Add New Product button and the new console will ask him for the data like Product name, price, quantity, weight, description, Supplier name, email, phone and his ID. The system then prompts the method within owner class to add the product in the file already available. The system also shows a dialogue upon successful operation i.e.:



Enter Product Details

Product Name:

Product Price:

Product Quantity:

Product Weight:

Product Description:

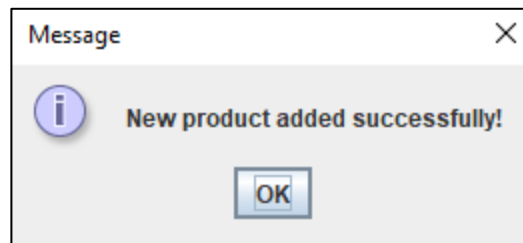
Supplier Name:

Supplier ID:

Supplier Phone:

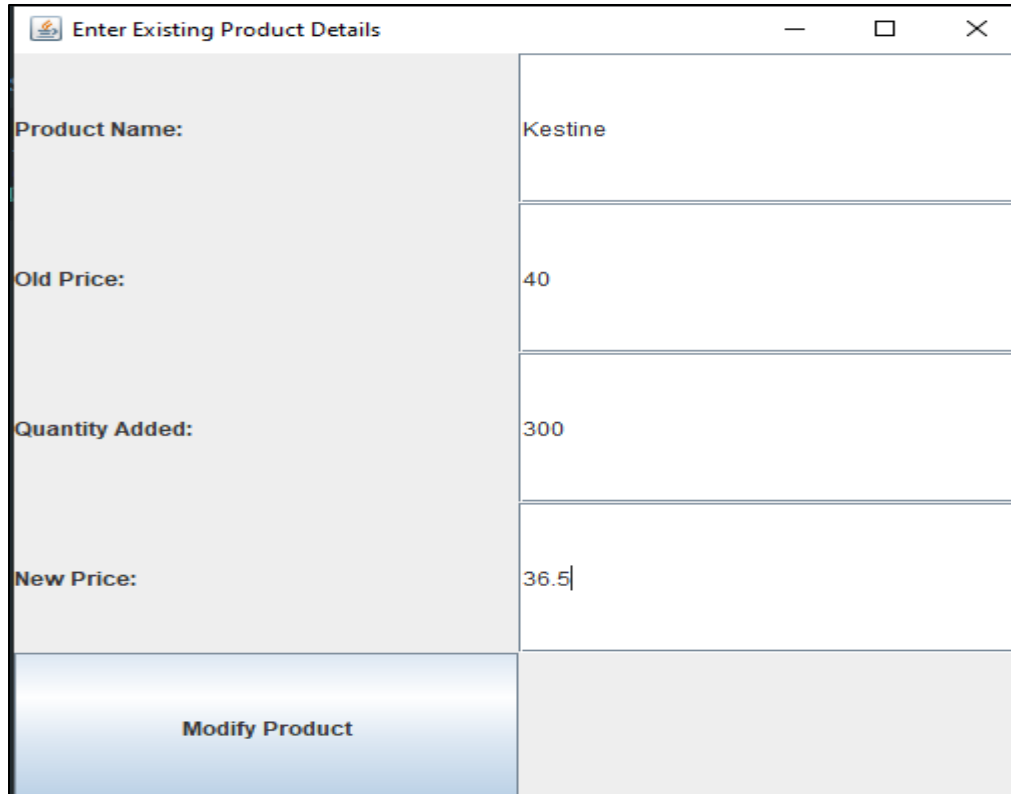
Supplier Email:

Add Product

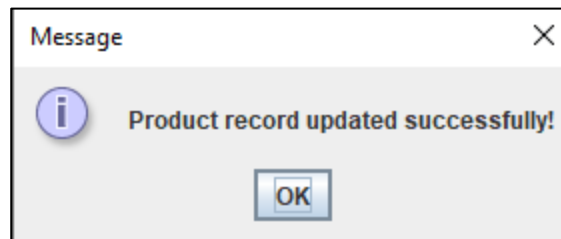


7) **Add Existing Product:**

In the same way, if the owner wants to add the stock of a product already available in store but in low amount, he will just click this button and the system will prompt him to enter the product name, old price and the new quantity and new stock to be added. The system then calls the method from Owner class to retrieve the data of objects from file and then after modifying the desired product it, will again put the modified data into file i.e., and it will show a success message as well:

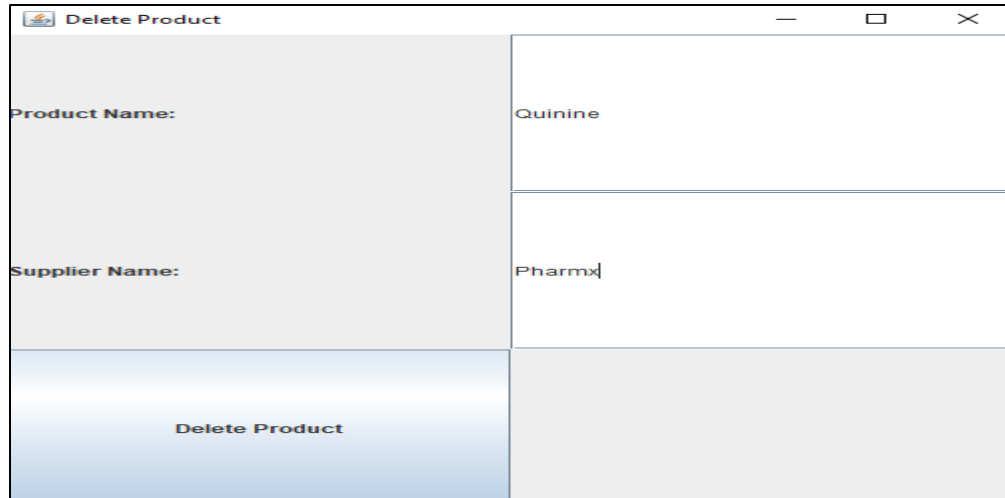


The screenshot shows a Java Swing dialog box titled "Enter Existing Product Details". It has a light gray background and standard window controls (minimize, maximize, close) in the top right corner. The dialog is divided into two main sections. The left section, which has a light gray background, contains four labels: "Product Name:", "Old Price:", "Quantity Added:", and "New Price:". The right section, which has a white background, contains four corresponding text input fields. The first field contains the text "Kestine", the second contains "40", the third contains "300", and the fourth contains "36.5". At the bottom of the dialog, there is a blue button with the text "Modify Product".

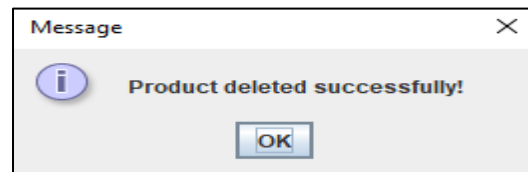


8) **Delete Product:**

If the Owner clicks the Delete Product Button, the system will prompt him to enter the product's name, and product's supplier name then action listener will call the method of delete product from Owner class which will take the date of products from the file and after deleting the mentioned product, add the new arrayList to file again while at the same time shows the success dialogue.

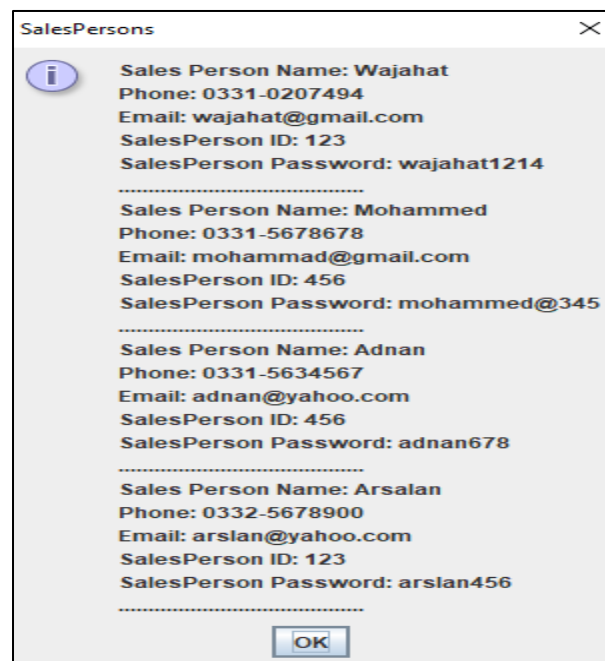


A dialog box titled "Delete Product" with a close button (X) in the top right corner. It contains two text input fields. The first field is labeled "Product Name:" and contains the text "Quinine". The second field is labeled "Supplier Name:" and contains the text "Pharmx". At the bottom left, there is a blue button labeled "Delete Product".



9) **Display Salesperson:**

This button when clicked through action listener calls the method in the Owner class to return the ArrayList of all the salespersons in store. The system then displays the salespersons in a new frame i.e:



A dialog box titled "SalesPersons" with a close button (X) in the top right corner. It features an information icon (i) on the left. The main content area displays the details of four salespersons, separated by horizontal lines. At the bottom, there is an "OK" button.

Sales Person Name: Wajahat
Phone: 0331-0207494
Email: wajahat@gmail.com
SalesPerson ID: 123
SalesPerson Password: wajahat1214

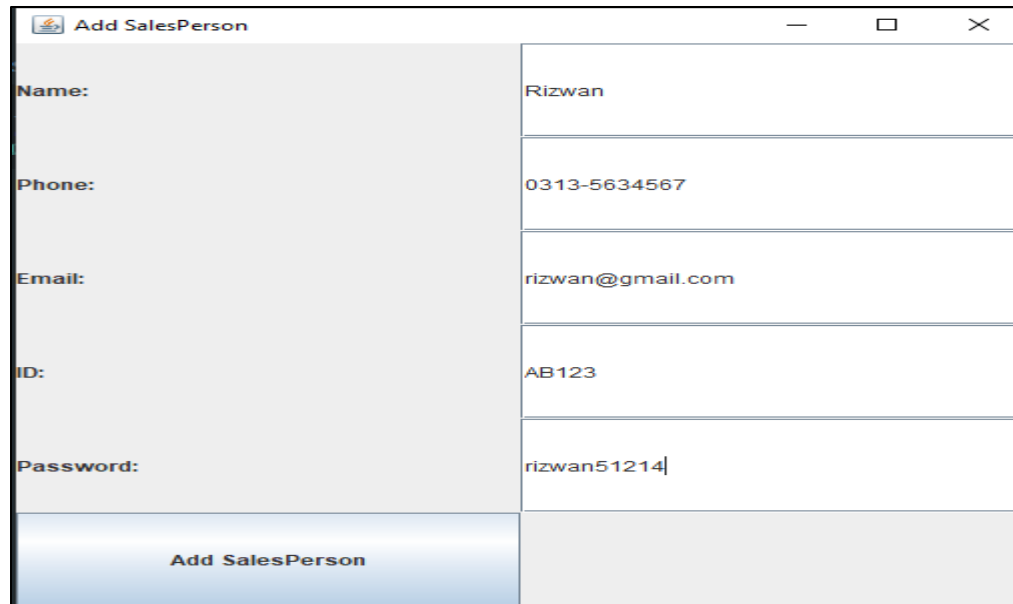
Sales Person Name: Mohammed
Phone: 0331-5678678
Email: mohammad@gmail.com
SalesPerson ID: 456
SalesPerson Password: mohammed@345

Sales Person Name: Adnan
Phone: 0331-5634567
Email: adnan@yahoo.com
SalesPerson ID: 456
SalesPerson Password: adnan678

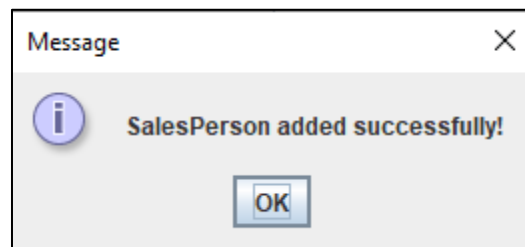
Sales Person Name: Arslan
Phone: 0332-5678900
Email: arslan@yahoo.com
SalesPerson ID: 123
SalesPerson Password: arslan456

10) Add Salesperson:

If the Owner clicks this button, the system prompts him to enter the data for a new salesperson i.e. his name, email, phone, ID, password, etc. The action listener then passes this object to a method which after saving the date in the file returns a true or false indicating the success of the process i.e.

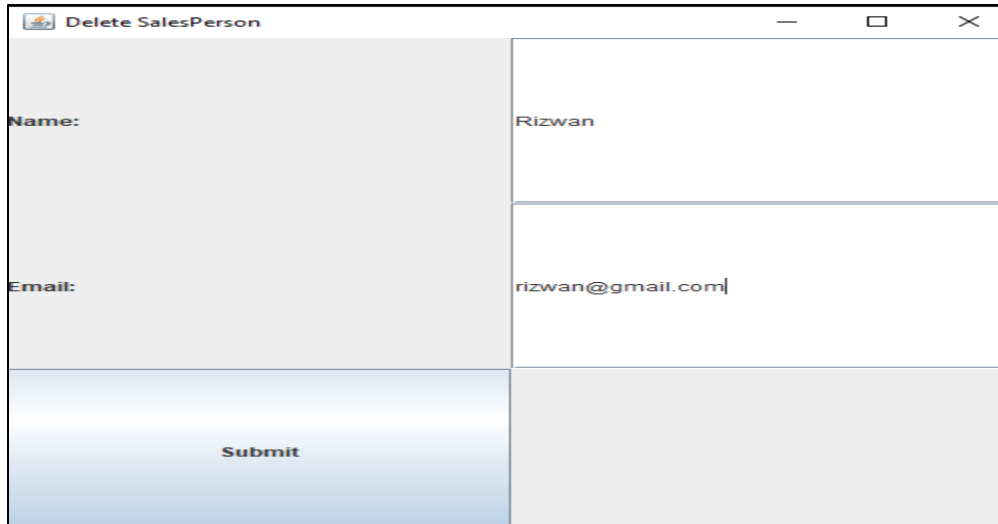


The screenshot shows a Java Swing window titled "Add SalesPerson". It contains five text input fields with labels "Name:", "Phone:", "Email:", "ID:", and "Password:". The fields are filled with the following text: "Rizwan", "0313-5634567", "rizwan@gmail.com", "AB123", and "rizwan51214". At the bottom of the window is a button labeled "Add SalesPerson".

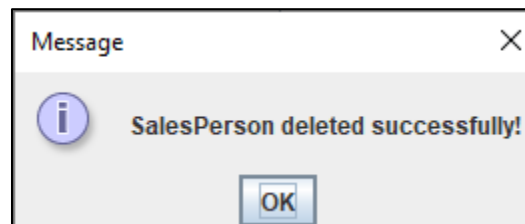


11) Delete Salesperson:

If the owner clicks the button i.e. Delete Salesperson, the system prompts him to enter the name and email of salesperson. The system then through action listener calls a method in Owner class to delete the salesperson with the name and email being same as those given by the user;

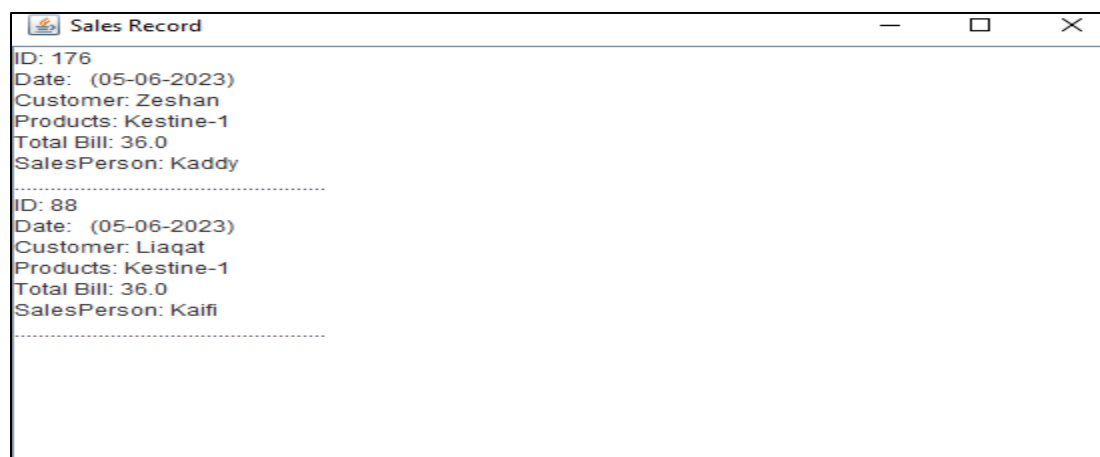


A dialog box titled "Delete SalesPerson" with a close button (X) in the top right corner. It contains two input fields: "Name:" with the text "Rizwan" and "Email:" with the text "rizwan@gmail.com". Below the input fields is a blue "Submit" button.



12) Show Sales Record:

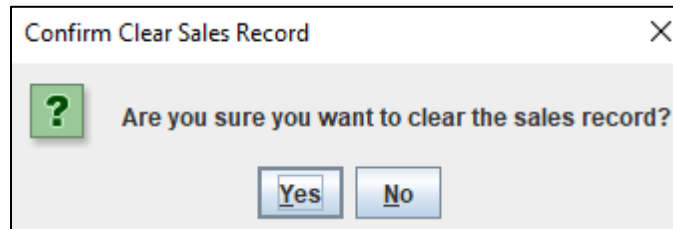
If the owner wants to see the record of all the sales which have been made after the last record was cleared by him, he will simply click the Show Sales Record Button to display the whole record showing the data of each order like customer information, the products bought by him, the product quantity sold, the amount for each product as well as the total bill for particular receipt i.e.:



A dialog box titled "Sales Record" with a close button (X) in the top right corner. It displays two sales records separated by dashed lines. The first record is for ID: 176, Date: (05-06-2023), Customer: Zeshan, Products: Kestine-1, Total Bill: 36.0, and SalesPerson: Kaddy. The second record is for ID: 88, Date: (05-06-2023), Customer: Liaqat, Products: Kestine-1, Total Bill: 36.0, and SalesPerson: Kaifi.

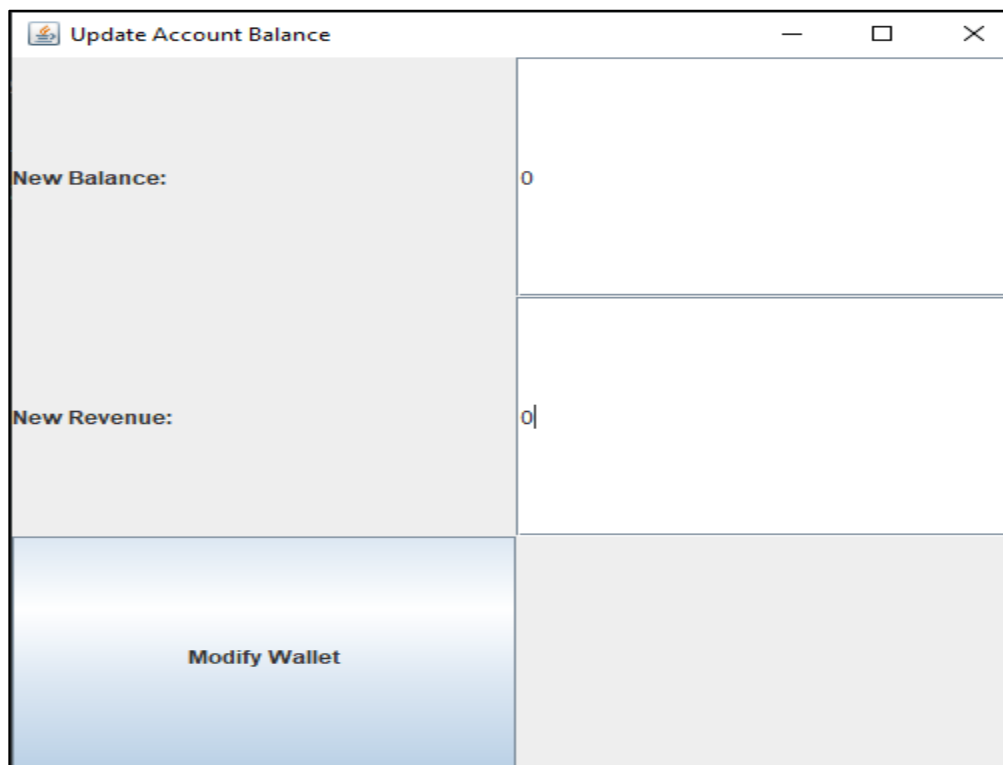
13) Clear Sales Record:

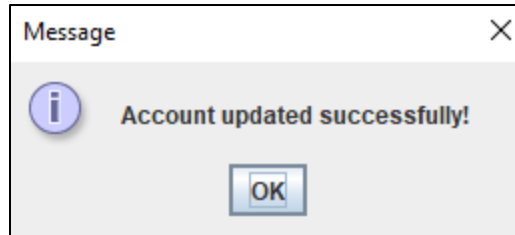
At the start of the day, the Owner might want to clear the salesRecord. So, if he clicks this button, the action listener will call a method from the owner class which clears the whole data in SalesRecord File. i.e.:



14) Update Sales Record:

Similarly the Owner may want to reset the account balance and total revenue in his profile or account at the start of the day. If he clicks this button, the action listener by calling a method from owner class changes the Owner's account balance and total revenue i.e.:



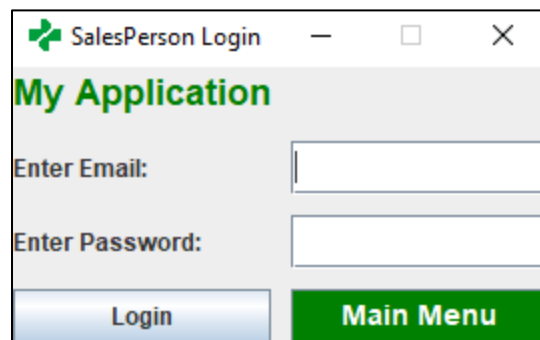


SALESPERSON OPTIONS:

The salesperson can perform following tasks in his windows i.e.:

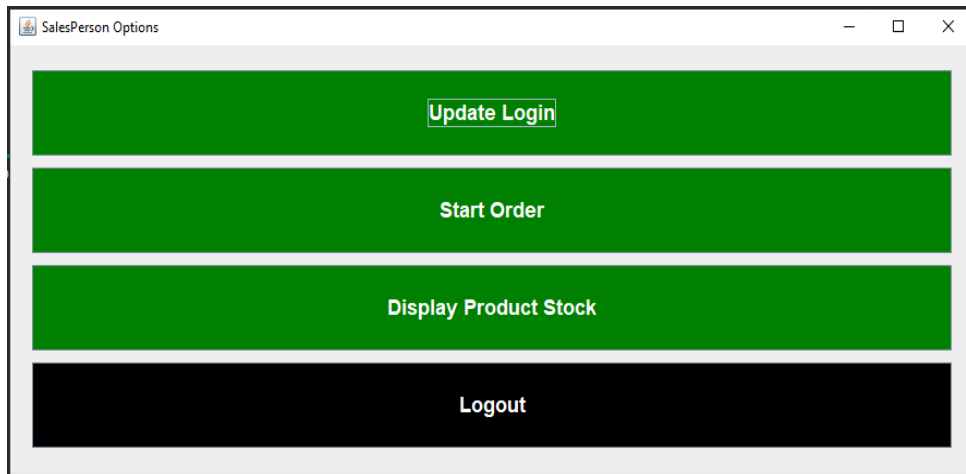
1) Login:

As soon as the SalesPerson selects or clicks the SalesPerson Button on main Frame or main window, the system directs him to a new window pane asking for his login Email and login password as shown below. If the password and email match with the salesperson's record the system logs in otherwise it shows an error message and keeps the user at same window. Moreover, the user has the option to move back to the main Window. i.e.:



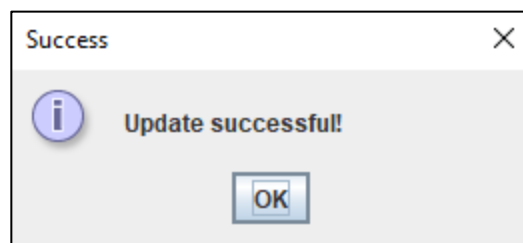
2) Logout:

After the successful login the system takes the Salesperson to a new window showing Salesperson options. Here the user can find the logout button to logout of his account and move back to login window i.e.:



3) Update Login:

If the user or salesPerson clicks the button named Update login from SalesPerson Options, the system prompts the salesperson to enter his existing email ,existing password and new email and new password. The system then calls the method from SalesPerson class through Interface. In that method if the users existing email and password match with his data in profile the password and email are updated to new email and password i.e.:

A screenshot of a dialog box titled "Update Login". It contains four input fields with labels: "Existing Email:" (containing "ali@gmail.com"), "Existing Password:" (containing "..."), "New Email:" (containing "cms@gmail.com"), and "New Password:" (containing "..."). At the bottom of the dialog are two buttons: "OK" and "Cancel".

4) Display Product Stock:

If the user clicks the button Display Products, the system through action listeners calls the method in Utility class to display all the available products on a new window or console i.e.:



5) Start Purchase:

If the salesperson clicks the button Start Purchase, system takes him to a new window asking for Salesperson Name which might want to be displayed on the receipt. After that the system takes all the necessary data from the salesperson and then passes this data to Salesperson class through action listeners. Where not only that data is stored in salesRecord file, but also displayed in separate window for customer i.e.:

SalesPerson Details

SalesPerson Name:

Customer Details

Customer Name:

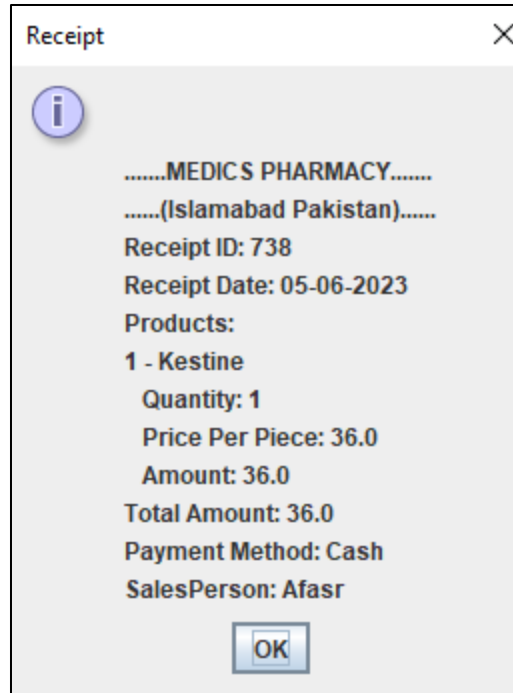
Customer Phone:

Customer Email:

Customer ID:

Payment Method:

Product Name	Quantity
<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>



CONCLUSION:

Organizations can effectively streamline their sales and purchase activities by using our “**Sales, Purchase and Receipt Management System**”. It increases operational efficiency, improves decision-making, and ultimately adds to the overall success of the organization by automating crucial operations and offering real-time insights.

SOURCE CODE:

The complete source Code for our project is given in the form of classes and Interfaces and GUI classes as well as below:

Class main

```
public class Main
{
    public static void main(String[] args)
    {
        MainFrame frame = new MainFrame();
    }
}
```

Class Person

```
import java.io.*;

public class Person implements Serializable
{
    private static final long serialVersionUID = -4037640016983222511L;
    protected String name;
    protected String phone;
    protected String email;

    // Constructor.....

    public Person()
    {
        name = "NULL";
        phone = "0000-00000000";
        email = "null@gmail.com";
    }

    public Person(String name, String phone, String email)
    {
        this.name = Utility.verifyName(name);
        this.phone = Utility.verifyPhone(phone);
        this.email = Utility.verifyEmail(email);
    }

    // Getters.....

    public String getName()
    {
        return name;
    }

    public String getPhone()
    {
        return phone;
    }
}
```

```

    }

    public String getEmail()
    {
        return email;
    }

    //.....Display Method.....
    public String toString()
    {
        String value = "Name: " + name;
        value += "\nPhone: " + phone;
        value += "\nEmail: " + email;
        return value;
    }
}

```

Class Customer

```

import java.io.Serializable;

import java.io.*;

class Customer extends Person implements Serializable
{

    private String custId;

    //Non parametrized constructor.....
    public Customer()
    {
        super();
        custId = "0000";
    }

    //Parametrized constructor.....
    public Customer(String name, String phone, String email,String id)

```

```

{
    super(name, phone, email);
    this.custId = id;
}

public void setCust_Id(String cust_Id)
{
    this.custId = custId;
}

public String getCust_Id()
{
    return custId;
}

@Override
public String toString()
{
    String displayString = "Customer ID: " + custId + "\n" +
        "Customer's " + super.toString();
    return displayString;
}
}

```

Interface

```

public interface LoginInterface
{
    boolean login(String email, String password);
    boolean updateLogin(String existingEmail, String existingPassword,String newMail,String
newPassword);
}

```

Class Owner

```

import java.util.ArrayList;
import java.io.*;

```

```

class Owner extends Person implements LoginInterface,Serializable
{
    private static final long serialVersionUID = 3436446231500268890L;

    protected String password;
    protected double accBalance;
    protected double totalRevenue;

    //Non parametrized constructor
    public Owner()
    {
        super();
        password = "0000";
        accBalance = 0;
        totalRevenue = 0;
    }
    //Parametrized constructor
    public Owner(String name, String phone, String email,String password, double
accBalance,double totalRevenue)
    {
        super(name, phone, email);
        this.password = password;
        this.accBalance = accBalance;
        this.totalRevenue = totalRevenue;
    }
    public void setPassword(String password)
    {
        this.password = password;
    }
    public void setAccBalance(double accBalance)
    {
        this.accBalance= accBalance;
    }
    public void setTotalRevenue(double totalRevenue)
    {
        this.totalRevenue = totalRevenue;
    }
    public String getPassword()
    {
        return password;
    }
    public double getAccBalance()
    {

```

```

        return accBalance;
    }
    public double getTotalRevenue()
    {
        return totalRevenue;
    }
    //.....Method for Login.....

    @Override
    public boolean login(String username, String password)
    {
        try
        {
            ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("OwnerRecord.txt"));
            Owner owner = new Owner();
            owner = (Owner) ois.readObject();
            if(owner.getEmail().equalsIgnoreCase(username) &&
owner.getPassword().equals(password))
            {
                return true;
            }
            else
            {
                return false;
            }
        }
        catch(IOException | ClassNotFoundException e)
        {
            System.out.println(e);
        }
        return false;
    }
    //.....Method for Update Login.....

    @Override
    public boolean updateLogin(String mail, String password,String newEmail,String
newPassword)
    {
        Owner owner = new Owner();
        try
        {
            ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("OwnerRecord.txt"));

```

```

        owner = (Owner) ois.readObject();
    }
    catch (IOException | ClassNotFoundException e)
    {
        System.out.println(e);
    }

    owner.email = Utility.verifyEmail(newEmail);
    owner.setPassword(newPassword);
    try
    {
        ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("OwnerRecord.txt"));
        oos.writeObject(owner);
        oos.close();
        return true; // Update successful
    }
    catch (Exception e)
    {
        System.out.println("Error with File");
        return false; // Update failed
    }
}

//.....Method to Add New Product.....
public void addNewProduct(Product newProduct)
{
    ArrayList<Product> products = new ArrayList<>();

    // Read existing products from file (if any)
    try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("ProductStore.txt")))
    {
        products = (ArrayList<Product>) ois.readObject();
    }
    catch (EOFException e)
    {
        // Handle the case where the file is empty
        products = new ArrayList<>();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}

```



```

// Append new product to the existing list
products.add(newProduct);

// Write the updated list to the file
try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("ProductStore.txt")))
{
    oos.writeObject(products);
}
catch (IOException e)
{
    throw new RuntimeException(e);
}
}

//.....Method to delete Product.....
public boolean deleteProduct(String productName, String supplierName)
{
    ArrayList<Product> productList = new ArrayList<>();
    boolean productFound = false;

    try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("ProductStore.txt")))
    {
        try
        {
            while (true)
            {
                ArrayList<Product> tempProductList = (ArrayList<Product>) ois.readObject();
                for (Product product : tempProductList)
                {
                    if (product.getProductName().equalsIgnoreCase(productName) &&
product.getSupplier().getName().equalsIgnoreCase(supplierName))
                    {
                        productFound = true;
                    }
                    else
                    {
                        productList.add(product);
                    }
                }
            }
        }
    }
}

```

```

        catch (EOFException e)
        {
            // No more objects in the file, so continue execution
        }
    }
    catch (IOException | ClassNotFoundException e)
    {
        e.printStackTrace();
        return false; // Return false if an exception occurred
    }
    catch (Exception e)
    {
        e.printStackTrace();
        return false; // Return false if an exception occurred
    }

    // Write the updated list to the file if the product was found and removed
    if (productFound)
    {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("ProductStore.txt")))
        {
            oos.writeObject(productList);
            return true; // Return true if the data was updated successfully
        }
        catch (IOException e)
        {
            e.printStackTrace();
            return false; // Return false if an exception occurred
        }
    }
    else
    {
        return false; // Return false if the product was not found
    }
}

```

```

//.....Method to Add Existing Product.....
public boolean addExistingProduct(String name, double oldPrice, int quantityAdded, double
newPrice)
{
    ArrayList<Product> products = new ArrayList<>();

```

```

    try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("ProductStore.txt")))
    {
        try
        {
            while (true)
            {
                ArrayList<Product> productList = (ArrayList<Product>) ois.readObject();
                boolean productFound = false; // Flag to track if a matching product is found
                for (Product product : productList)
                {
                    if (product.getProductName().equalsIgnoreCase(name) &&
Math.abs(product.getProductPrice() - oldPrice) < 0.0001)
                    {
                        product.setQuantity(product.getQuantityInStock() + quantityAdded);
                        product.setProductPrice(newPrice);
                        productFound = true;
                        break; // Exit the loop after updating the quantity and price
                    }
                }
                if (productFound)
                {
                    products.addAll(productList);
                }
            }
        }
        catch (EOFException e)
        {
            // No more objects in the file, so continue execution
        }
    }
    catch (IOException | ClassNotFoundException e)
    {
        e.printStackTrace();
        return false; // Return false if an exception occurred
    }
    catch (Exception e)
    {
        e.printStackTrace();
        return false; // Return false if an exception occurred
    }

    // Write the updated list to the file

```

```

    try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("ProductStore.txt")))
    {
        oos.writeObject(products);
        return true; // Return true if the record was updated successfully
    }
    catch (IOException e)
    {
        e.printStackTrace();
        return false; // Return false if an exception occurred
    }
}

//.....Method to Show SalesPersons.....
public ArrayList<SalesPerson> showSalesPerson()
{
    ArrayList<SalesPerson> salesPersons = new ArrayList<>();
    try
    {
        ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("SalesPersonRecord.txt"));
        try
        {
            while (true)
            {
                ArrayList<SalesPerson> salesPersonData = (ArrayList<SalesPerson>)
ois.readObject();
                salesPersons.addAll(salesPersonData);
            }
        }
        catch (EOFException e)
        {
            // No more objects in the file, so continue execution
        }
        ois.close();
    }
    catch (IOException | ClassNotFoundException e)
    {
        e.printStackTrace();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}

```

```

        return salesPersons;
    }

    //.....Method to Add SalesPersons.....
    public boolean addSalesPerson(SalesPerson salesPerson)
    {
        ArrayList<SalesPerson> salesPersons = new ArrayList<>();

        // Read existing SalesPersons from file (if any)
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("SalesPersonRecord.txt")))
        {
            salesPersons = (ArrayList<SalesPerson>) ois.readObject();
        }
        catch (EOFException e)
        {
            // Handle the case where the file is empty
            salesPersons = new ArrayList<>();
        }
        catch (Exception e)
        {
            e.printStackTrace();
            return false; // Return false if an exception occurred
        }

        // Append new SalesPerson to the existing list
        salesPersons.add(salesPerson);

        // Write the updated list to the file
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("SalesPersonRecord.txt")))
        {
            oos.writeObject(salesPersons);
            return true; // Return true if the process is completed successfully
        }
        catch (IOException e)
        {
            e.printStackTrace();
            return false; // Return false if an exception occurred
        }
    }

    //.....Method to Delete SalesPerson.....
    public boolean deleteSalesPerson(String name, String email)

```

```

{
    ArrayList<SalesPerson> salesPersonList = new ArrayList<>();
    boolean salesPersonFound = false;

    try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("SalesPersonRecord.txt")))
    {
        try
        {
            salesPersonFound = false;
            while (true)
            {
                ArrayList<SalesPerson> tempSalesPersonList = (ArrayList<SalesPerson>)
ois.readObject();
                for (SalesPerson salesPerson : tempSalesPersonList)
                {
                    if (salesPerson.getName().equalsIgnoreCase(name) &&
salesPerson.getEmail().equalsIgnoreCase(email))
                    {
                        salesPersonFound = true;
                    }
                    else
                    {
                        salesPersonList.add(salesPerson);
                    }
                }
            }
        }
        catch (EOFException e)
        {
            // No more objects in the file, so continue execution
        }
    }
    catch (IOException | ClassNotFoundException e)
    {
        e.printStackTrace();
        return false; // Return false if an exception occurred
    }
    catch (Exception e)
    {
        e.printStackTrace();
        return false; // Return false if an exception occurred
    }
}

```

```

        // Write the updated list to the file if the salesperson was found and removed
        if (salesPersonFound)
        {
            try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("SalesPersonRecord.txt")))
            {
                oos.writeObject(salesPersonList);
                return true; // Return true if the salesperson was deleted successfully
            }
            catch (IOException e)
            {
                e.printStackTrace();
                return false; // Return false if an exception occurred
            }
        }
        else
        {
            return false; // Return false if the salesperson was not found
        }
    }

//.....Method to Show SalesRecord.....
public ArrayList<String> showSalesRecord()
{
    ArrayList<String> salesRecordDetailsList = new ArrayList<>();
    ArrayList<Receipt> salesRecord;
    try
    {
        ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("SalesRecord.txt"));

        try
        {
            while (true)
            {
                salesRecord = (ArrayList<Receipt>) ois.readObject();
                if (salesRecord.isEmpty())
                {
                    salesRecordDetailsList.add("No data in Record.");
                } else
                {
                    for (Receipt record : salesRecord)
                    {
                        // Create a separate ArrayList to store the sales record details

```

```

        ArrayList<String> salesRecordDetails = new ArrayList<>();
        salesRecordDetails.add("ID: " + record.receiptId);
        salesRecordDetails.add("Date: " + " (" + record.getReceiptDate() + ")");
        salesRecordDetails.add("Customer: " + record.getCustomer().getName());
        for (Product purchased : record.getCart())
        {
            salesRecordDetails.add("Products: " + purchased.getProductName() + "-" +
purchased.getQuantityInStock());
        }
        salesRecordDetails.add("Total Bill: " + record.getAmount());
        salesRecordDetails.add("SalesPerson: " + record.getSalesOfficer().getName());
        salesRecordDetails.add(".....");

        // Add the sales record details ArrayList to the main list
        salesRecordDetailsList.addAll(salesRecordDetails);
    }
}
}
}
catch (EOFException e)
{
    // No more objects in the file, so continue execution
}

    ois.close();
}
catch (IOException | ClassNotFoundException e)
{
    e.printStackTrace();
}
catch (Exception e)
{
    e.printStackTrace();
}

    return salesRecordDetailsList;
}

```

```

//.....Method to Delete SalesRecord.....
public boolean deleteSalesRecord()
{
    try
    {

```



```

        // Open the file in write mode
        ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("SalesRecord.txt"));

        // Write an empty ArrayList to clear the file
        ArrayList<Receipt> emptySalesRecord = new ArrayList<>();
        oos.writeObject(emptySalesRecord);
        oos.close();
        return true; // Return true indicating successful deletion
    }
    catch (IOException e)
    {
        e.printStackTrace();
        return false; // Return false indicating failure
    }
}

//.....Method to Show Account Details.....

public static Owner displayAccount()
{
    Owner owner = null;

    try
    {
        ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("OwnerRecord.txt"));
        owner = (Owner) ois.readObject();
        ois.close();
    }
    catch (IOException | ClassNotFoundException e)
    {
        System.out.println(e);
    }

    return owner;
}

@Override
public String toString()
{
    String displayString = "Owner's " + super.toString() +

```

```

        "\nPassword: " + getPassword() + "\n" +
        "Account Balance: " + getAccBalance() + "\n" +
        "Revenue Generated: " + getTotalRevenue() + "\n";
    return displayString;
}
}

```

Class Salesperson

```

import java.io.*;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;

public class SalesPerson extends Person implements LoginInterface, Serializable
{
    private static final long serialVersionUID = 7194050316026614779L;

    private String ID;
    private String password;
    public SalesPerson()
    {
        super();
        ID = "admin";
        password = "0000";
    }
    public SalesPerson(String name, String phone, String email, String ID, String password)
    {
        super(name, phone, email);
        this.ID = ID;
        this.password = password;
    }
    //Creating Setter and Getter Methods.
    public void setID(String ID)
    {
        this.ID = ID;
    }
    public void setPassword(String password)
    {
        this.password = password;
    }
}

```

```

    }
    public String getID()
    {
        return ID;
    }
    public String getPassword()
    {
        return password;
    }
    //.....Method for Login.....
    @Override
    public boolean login(String email, String password)
    {
        ArrayList<SalesPerson> salesPersons;
        try
        {
            ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("SalesPersonRecord.txt"));

            try
            {
                while (true)
                {
                    salesPersons = (ArrayList<SalesPerson>) ois.readObject();
                    for (SalesPerson salesPeople : salesPersons)
                    {
                        if(salesPeople.getEmail().equalsIgnoreCase(email) &&
salesPeople.getPassword().equals(password))
                        {
                            return true;
                        }
                    }
                }
            }
            catch (EOFException e)
            {
                // No more objects in the file, so continue execution
            }

            ois.close();
        }
        catch (IOException | ClassNotFoundException e)
        {
            e.printStackTrace();
        }
    }

```

```

    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
    return false;
}
//.....Method for Update Login.....
@Override
public boolean updateLogin(String existingEmail, String existingPassword, String newMail,
String newPassword)
{
    ArrayList<SalesPerson> salesPersons = new ArrayList<>();
    boolean isUpdated = false;

    try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("SalesPersonRecord.txt")))
    {
        try
        {
            while (true)
            {
                ArrayList<SalesPerson> salesPeople = (ArrayList<SalesPerson>) ois.readObject();
                for (SalesPerson salesPerson : salesPeople)
                {
                    if (salesPerson.getEmail().equalsIgnoreCase(existingEmail) &&
salesPerson.getPassword().equals(existingPassword))
                    {
                        salesPerson.email = newMail;
                        salesPerson.setPassword(newPassword);
                        isUpdated = true;
                        break;
                    }
                }
                salesPersons.addAll(salesPeople);
            }
        }
        catch (EOFException e)
        {
            // No more objects in the file, so continue execution
        }
    }
    catch (IOException | ClassNotFoundException e)
    {

```

```

        e.printStackTrace();
    }
    catch (Exception e) {
        e.printStackTrace();
    }

    if (isUpdated)
    {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("SalesPersonRecord.txt")))
        {
            oos.writeObject(salesPersons); // Write the updated list to the file
        }
        catch (IOException e)
        {
            throw new RuntimeException(e);
        }
    }

    return isUpdated;
}

```

//Method for SalesPerson to Start Order.....

```

public static Receipt startPurchase(String SalesPersonName ,String name, String phone, String
email, String id, ArrayList<Product> orderedProducts, String PaymentMethod)
{
    Customer C2 = new Customer(name, phone, email, id);
    SalesPerson salesPerson = new SalesPerson();
    salesPerson.name = SalesPersonName;

    LocalDate currentDate = LocalDate.now();
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy");
    String date = currentDate.format(formatter);

    Receipt receipt = new Receipt();
    receipt.customer = C2;
    receipt.date = date;
    receipt.paymentMethod = PaymentMethod;
    receipt.setSalesOfficer(salesPerson);
}

```

```

double totalBill = 0;

boolean productFound = false;
ArrayList<Product> ProductStock = new ArrayList<>();
ArrayList<Product> cartProduct = new ArrayList<>();
try
{
    ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("ProductStore.txt"));
    ProductStock = (ArrayList<Product>) ois.readObject();
    for (Product ordered : orderedProducts)
    {
        for (Product stock : ProductStock)
        {
            if (ordered.getProductName().equalsIgnoreCase(stock.getProductName()) &&
ordered.getQuantityInStock() <= stock.getQuantityInStock()) {
                productFound = true;
                Product productForCart = new Product(); // Create a new Product object
                productForCart.setProductName(stock.getProductName());
                productForCart.setProductPrice(stock.getProductPrice());
                productForCart.setQuantity(ordered.getQuantityInStock());
                totalBill += ordered.getQuantityInStock() * stock.getProductPrice();
                stock.setQuantity(stock.getQuantityInStock() - ordered.getQuantityInStock());
                cartProduct.add(productForCart);
            }
        }
    }
}
catch(Exception e)
{
    System.out.println(e);
}

try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("ProductStore.txt")))
{
    oos.writeObject(ProductStock);
    oos.close();
}
catch (IOException e)
{
    e.printStackTrace();
}

```

```

    try
    {
        ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("OwnerRecord.txt"));
        Owner owner;
        owner = (Owner) ois.readObject();
        double newBalance = owner.getAccBalance() + totalBill;
        double newRevenue = owner.getTotalRevenue() + totalBill;
        ois.close();
        Utility.updateBalances(newBalance, newRevenue);

    }
    catch (IOException | ClassNotFoundException e)
    {
        System.out.println(e);
    }

    receipt.cart = cartProduct;
    receipt.amount = totalBill;

    ArrayList<Receipt> salesRecord = new ArrayList<>();

    try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("SalesRecord.txt")))
    {
        salesRecord = (ArrayList<Receipt>) ois.readObject();
    }
    catch (FileNotFoundException e)
    {
        salesRecord = new ArrayList<>();
    }
    catch (EOFException e)
    {
        salesRecord = new ArrayList<>();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }

    salesRecord.add(receipt);

```

```

        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("SalesRecord.txt")))
        {
            oos.writeObject(salesRecord);
        }
        catch (IOException e)
        {
            throw new RuntimeException(e);
        }

        return receipt;
    }

```

```

@Override
public String toString()
{
    String displayString = "Sales Person " + super.toString() + "\n" +
        "SalesPerson ID: " + ID + "\n" +
        "SalesPerson Password: " + getPassword() + "\n" +
        ".....";
    return displayString;
}
}

```

Class Supplier

```

import java.io.*;
public class Supplier extends Person implements Serializable
{
    private static final long serialVersionUID = 6947560519416110057L;

    private String ID;

    public Supplier()
    {
        super();
        ID = "0000";
    }
    public Supplier(String name, String phone, String email,String id)
    {

```



```

        super(name, phone, email);
        this.ID = id;
    }
    public void setID(String ID)
    {
        this.ID = ID;
    }
    public String getID()
    {
        return ID;
    }

    @Override
    public String toString()
    {
        String displayString = "Supplier ID: " + ID + "\n" +
            "Supplier " + super.toString();
        return displayString;
    }
}

```

Class Store

```

import java.util.*;
import java.io.*;
public class Store implements Serializable
{
    protected static String storeName;
    protected static String storeAddress;
    protected static Product [] productStock;
    protected static SalesPerson [] salesOfficer;

    public Store()
    {
        storeName = "MEDICS PHARMACY";
        storeAddress = "Islamabad Pakistan";
        productStock = new Product[0];
        salesOfficer = new SalesPerson[0];
    }
    public Store(String storeName, String storeAddress, Product [] productStock, SalesPerson[]
salesOfficer)

```

```

{
    this.storeName = storeName;
    this.storeAddress = storeAddress;
    this.productStock = productStock;
    this.salesOfficer = salesOfficer;
}
public void setStoreName(String storeName)
{
    Store.storeName = storeName;
}
public void setStoreAddress(String storeAddress)
{
    Store.storeAddress = storeAddress;
}
public void setProductStock(Product[] productStock)
{
    Store.productStock = productStock;
}
public void setSalesOfficer(SalesPerson[] salesOfficer)
{
    Store.salesOfficer = salesOfficer;
}

public static String getStoreName()
{
    return storeName;
}
public static String getStoreAddress()
{
    return storeAddress;
}
public static Product[] getProductStock()
{
    return productStock;
}

public static SalesPerson[] getSalesOfficer()
{
    return salesOfficer;
}

public String displaySalesperson()
{
    String value = "";

```

```

    for(int i = 0; i < salesOfficer.length ; i++ )
    {
        value += "\n.....SalesPerson " + (i+1) + " .....";
        value += "\nSalesPerson Name: " + salesOfficer[i].getName();
        value += "\nSalesPerson ID: " + salesOfficer[i].getID();
        value += "\n.....";
    }
    return value;
}
public String displayStock()
{
    String value = "";
    for(int i = 0; i < productStock.length ; i++ )
    {
        value += "\n.....Product " + (i+1) + " .....";
        value += "\nProduct Name: " + productStock[i].getProductName();
        value += "\nProduct Price: " + productStock[i].getProductPrice();
        value += "\nProduct Quantity: " + productStock[i].getQuantityInStock();
        value += "\nProduct Weight: " + productStock[i].getWeight();
        value += "\nProduct Description: " + productStock[i].getDescription();
        value += "\nProduct Supplier: " + productStock[i].getSupplier().getName();
        value += "\n.....";
    }
    return value;
}
}

```

Class Product

```

import java.io.*;

public class Product implements Serializable
{
    private static final long serialVersionUID = 8785257001383939684L;

    private String productName;
    private double productPrice;
    private int quantityInStock;

```

```
private double weight;
private String description;
private Supplier supplier;

public Product()
{
}

public Product(String productName, double productPrice, int quantityInStock, double weight,
String description, Supplier supplier)
{
    this.productName = productName;
    this.productPrice = productPrice;
    this.quantityInStock = quantityInStock;
    this.weight = weight;
    this.description = description;
    this.supplier = supplier;
}

public void setProductName(String productName)
{
    this.productName = productName;
}

public void setProductPrice(double productPrice)
{
    this.productPrice = productPrice;
}

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

```
public void setWeight(double weight)
{
    this.weight = weight;
}
public void setDescription(String description)
{
    this.description = description;
}
public void setSupplier(Supplier supplier)
{
    this.supplier = supplier;
}
public String getProductName()
{
    return productName;
}
public double getProductPrice()
{
    return productPrice;
}
public int getQuantityInStock()
{
    return quantityInStock;
}
public double getWeight()
{
    return weight;
}
public String getDescription()
{
    return description;
}
```

```

    }

    public Supplier getSupplier()
    {
        return supplier;
    }

    public String display()
    {
        String value = "";
        value += "\nProduct Name==> " + productName;
        value += "\nProduct Price: " + productPrice;
        value += "\nProduct Weight: " + weight;
        value += "\nProduct in Stock: " + quantityInStock;
        value += "\nDescription: " + description;
        value += "\nSupplier ID: " + supplier.getID();
        value += "\nSupplier Name: " + supplier.getName();
        value += "\nSupplier Phone: " + supplier.getPhone();
        value += "\n.....";
        return value;
    }
}

```

Class Receipt

```

import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.io.*;
import java.util.ArrayList;

public class Receipt implements Serializable
{
    private static final long serialVersionUID = 5018065893772484716L;

```

```

protected int receiptId;
protected String date;
protected ArrayList<Product> cart ;
protected double amount;
protected String paymentMethod;
protected SalesPerson salesOfficer;
protected Customer customer;

public Receipt()
{
    receiptId = (int) (Math.random()*1000);
    LocalDate currentDate = LocalDate.now();
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MM-yyyy");
    String dateString = currentDate.format(formatter);
    date = dateString;
    cart = null;
    amount = 0;
    paymentMethod = "NULL";
}
public Receipt(int receiptId, String date, ArrayList<Product> products, double amount, String
paymentMethod, Customer customer, SalesPerson salesperson)
{
    this.receiptId = receiptId;
    this.date = date;
    this.cart = products;
    this.amount = amount;
    this.paymentMethod = paymentMethod;
    this.customer = customer;
    this.salesOfficer = salesperson;
}

// Getters and Setters

public void setCart(ArrayList<Product> cart)
{
    this.cart = cart;
}

public void setSalesOfficer(SalesPerson salesOfficer)
{
    this.salesOfficer = salesOfficer;
}
public void setPaymentMethod(String paymentMethod)

```

```

{
    this.paymentMethod = paymentMethod;
}

public ArrayList<Product> getCart()
{
    return cart;
}

public int getReceiptId()
{
    return receiptId;
}
public String getReceiptDate()
{
    return date;
}
public double getAmount()
{
    return amount;
}
public String getPaymentMethod()
{
    return paymentMethod;
}
public SalesPerson getSalesOfficer()
{
    return salesOfficer;
}
public Customer getCustomer()
{
    return customer;
}

public String toString()
{
    String value = "";
    Store store = new Store();
    value += "\n\n....." + store.getStoreName() + ".....";
    value += "\n.....(" + store.getStoreAddress() + ").....";
    value += "\nReceipt ID: " + receiptId;
    value += "\nReceipt Date: " + date;
    value += "\nProducts:";
    int i = 0;

```



```

    for (int j = 0; j < cart.size(); j++)
    {
        value += "\n" + (j+1) + " - " + cart.get(j).getProductName() + "\n  Quantity: " +
cart.get(j).getQuantityInStock() + "\n  Price Per Piece: " + cart.get(j).getProductPrice() + "\n
Amount: " + cart.get(j).getQuantityInStock() * cart.get(j).getProductPrice();
    }
    value += "\nTotal Amount: " + amount;
    value += "\nPayment Method: " + paymentMethod;
    value += "\nSalesPerson: " + salesOfficer.getName();
    return value;
}
}

```

Utility Class

```

import java.io.*;
import java.util.ArrayList;
import java.util.Scanner;

public class Utility implements Serializable
{
    //.....Method to Modify Account Details of Owner.....
    public static boolean updateBalances(Double updatedBalance, Double updatedRevenue)
    {
        try
        {
            ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("OwnerRecord.txt"));
            Owner owner = (Owner) ois.readObject();
            ois.close();

            owner.accBalance = updatedBalance;
            owner.totalRevenue = updatedRevenue;

            ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("OwnerRecord.txt"));
            oos.writeObject(owner);
            oos.close();

            return true; // Success
        }
        catch (Exception e)
        {

```

```

        System.out.println("Error with File");
        return false; // Failure
    }
}

//.....Method to Show ProductStock.....
public static ArrayList<Product> showProductStock()
{
    ArrayList<Product> productList = new ArrayList<>();

    try
    {
        ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("ProductStore.txt"));

        try
        {
            while (true)
            {
                productList.addAll((ArrayList<Product>) ois.readObject());
            }
        }
        catch (EOFException e)
        {
            // No more objects in the file, so continue execution
        }

        ois.close();
    }
    catch (IOException | ClassNotFoundException e)
    {
        e.printStackTrace();
    }
    catch (Exception e) {
        e.printStackTrace();
    }

    return productList;
}

// Input validation methods.....

//.....Method to Validate Name.....
public static String verifyName(String name)
{

```

```

Scanner input = new Scanner(System.in);
int count;
//Validating only alphabets in input.....
while(true)
{
    if(name.length() > 30)
    {
        return "NULL";
    }
    else
        break;
}
while(true)
{
    count = 0;
    for(int i = 0; i < name.length(); i++)
    {
        if (!(Character.isLetter(name.charAt(i)) || name.charAt(i) == '.' || name.charAt(i) == '
'))
        {
            count++;
            break;
        }
    }
    if (count != 0)
    {
        return "NULL";
    }
    else
        break;
}
return name;
}
//.....Method to Validate Phone Number.....
public static String verifyPhone(String contactNumber)
{
    Scanner input = new Scanner(System.in);
    int count;
    //Validation of Correct format of phone Number.....
    while(true)
    {
        if (((contactNumber.length() == 12 && contactNumber.charAt(4) == '-' &&
contactNumber.charAt(0) == '0'

```

```

        && contactNumber.charAt(1) == '3')&&(contactNumber.charAt(2) == '0' ||
contactNumber.charAt(2) == '2' ||
        contactNumber.charAt(2) == '1' || contactNumber.charAt(2) == '4' ||
contactNumber.charAt(2) == '3'))
    {
        return "NULL";
    }
    else
    {
        count = 0;
        for(int i =0; i < contactNumber.length(); i++)
        {
            if (!(Character.isDigit(contactNumber.charAt(i))))
            {
                if (i == 4)
                {
                    continue;
                }
                count++;
                break;
            }
        }
        if (count != 0)
        {
            return "NULL";
        }
        else
        {
            break;
        }
    }
}
return contactNumber;
}

```

//.....Method to Validate Email.....

```

public static String verifyEmail(String email)
{
    Scanner input = new Scanner(System.in);
    int count = 0;

    while (true)
    {
        boolean containsAtSymbol = false;

```

```

for (int i = 0; i < email.length(); i++)
{
    if (email.charAt(i) == '@')
    {
        containsAtSymbol = true;
        break;
    }
}

if (email.length() <= 10 || !containsAtSymbol)
{
    return "NULL";
}
else
{
    break;
}
}

while (true)
{
    for (int i = 0; i < email.length(); i++)
    {
        char c = email.charAt(i);

        if (!(Character.isLetter(c) || Character.isDigit(c) || c == '.' || c == '@' || c == '-'))
        {
            count++;
            break;
        }
    }
    if (count != 0)
    {
        return "NULL";
    }
    else
    {
        break;
    }
}

return email;

```

```
}  
}
```

Class OwnerGUI

```
import javax.swing.*;  
import java.awt.*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.util.ArrayList;  
import java.util.Scanner;  
  
public class OwnerGUI extends JFrame  
{  
    private JTextField usernameField;  
    private JPasswordField passwordField;  
  
    OwnerGUI()  
    {  
        this.setVisible(true);  
        this.setSize(400, 200);  
        this.setTitle("Owner Login");  
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        this.setResizable(false);  
        this.setLayout(new BorderLayout());  
  
        ImageIcon logo = new ImageIcon("Logo.png");  
        this.setIconImage(logo.getImage());  
  
        JPanel panel = new JPanel();  
        panel.setLayout(new GridLayout(4, 2, 10, 10));  
  
        JLabel appNameLabel = new JLabel("My Application");  
        appNameLabel.setFont(new Font("Arial", Font.BOLD, 18));  
        appNameLabel.setHorizontalAlignment(SwingConstants.CENTER);  
  
        JLabel nameLabel = new JLabel("Enter Email:");  
        usernameField = new JTextField();  
  
        JLabel passwordLabel = new JLabel("Enter Password:");
```

```

passwordField = new JPasswordField();

JButton loginButton = new JButton("Login");
JButton mainMenuButton = new JButton("Main Menu");

// Styling options
Font buttonFont = new Font("Arial", Font.BOLD, 14);
Color buttonColor = new Color(0, 128, 0);

appNameLabel.setForeground(buttonColor);
mainMenuButton.setFont(buttonFont);

mainMenuButton.setBackground(buttonColor);
mainMenuButton.setForeground(Color.WHITE);

// Main Menu button action listener
mainMenuButton.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        dispose(); // Close the login window
        MainFrame mainFrame = new MainFrame(); // Open the main menu window
    }
});

// Login button action listener
loginButton.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        String username = usernameField.getText();
        String password = new String(passwordField.getPassword());
        Owner owner = new Owner();
        boolean loginSuccessful = owner.login(username, password);
        if (loginSuccessful)
        {
            dispose(); // Close the login window
            showOwnerOptions(); // Open the owner options window
        }
        else
        {
            JOptionPane.showMessageDialog(OwnerGUI.this, "Invalid username or password.

```

```

Please try again.", "Login Failed", JOptionPane.ERROR_MESSAGE);
    }
}
});

panel.add(appNameLabel);
panel.add(new JLabel()); // Empty label for spacing
panel.add(nameLabel);
panel.add(usernameField);
panel.add(passwordLabel);
panel.add(passwordField);
panel.add(loginButton);
panel.add(mainMenuButton);

add(panel, BorderLayout.CENTER);
pack();
setLocationRelativeTo(null);
}

private void showOwnerOptions()
{
    JFrame optionsFrame = new JFrame();
    optionsFrame.setTitle("Owner Options");
    optionsFrame.setSize(900, 400);
    optionsFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    optionsFrame.setLayout(new BorderLayout());

    JPanel optionsPanel = new JPanel();
    optionsPanel.setLayout(new GridLayout(4, 2, 10, 10));
    optionsPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

    JButton showProfileButton = new JButton("Show Profile");
    JButton updateLoginButton = new JButton("Update Login");
    JButton displayProductStockButton = new JButton("Display Product Stock");
    JButton addNewProductButton = new JButton("Add New Product");
    JButton addExistingProductButton = new JButton("Add Existing Product");
    JButton deleteProductButton = new JButton("Delete Product");
    JButton displaySalesPersonsButton = new JButton("Display SalesPersons");
    JButton addSalesPersonButton = new JButton("Add SalesPerson");
    JButton deleteSalesPersonButton = new JButton("Delete SalesPerson");
    JButton showSalesRecordButton = new JButton("Show Sales Record");
    JButton clearSalesRecordButton = new JButton("Clear Sales Record");
    JButton updateAccountButton = new JButton("Update Balance");
    JButton logoutButton = new JButton("Logout");

```



```

// Styling options
Font buttonFont = new Font("Arial", Font.BOLD, 14);
Color buttonColor = new Color(0, 128, 0);

JButton[] buttons =
{
    showProfileButton,
    updateLoginButton,
    displayProductStockButton,
    addNewProductButton,
    addExistingProductButton,
    deleteProductButton,
    displaySalesPersonsButton,
    addSalesPersonButton,
    deleteSalesPersonButton,
    showSalesRecordButton,
    clearSalesRecordButton,
    updateAccountButton,
    logoutButton
};

FontMetrics fontMetrics = updateLoginButton.getFontMetrics(buttonFont);
int maxButtonWidth = 0;

for (JButton button : buttons)
{
    int buttonWidth = fontMetrics.stringWidth(button.getText());
    maxButtonWidth = Math.max(maxButtonWidth, buttonWidth);
    button.setFont(buttonFont);
    button.setBackground(buttonColor);
    button.setForeground(Color.WHITE);
}

Dimension buttonSize = new Dimension(maxButtonWidth + 20, 40);

for (JButton button : buttons)
{
    button.setPreferredSize(buttonSize);
    optionsPanel.add(button);
}
showProfileButton.setBackground(buttonColor);
updateLoginButton.setBackground(buttonColor);
displayProductStockButton.setBackground(buttonColor);

```

```

addNewProductButton.setBackground(buttonColor);
addExistingProductButton.setBackground(buttonColor);
deleteProductButton.setBackground(buttonColor);
displaySalesPersonsButton.setBackground(buttonColor);
addSalesPersonButton.setBackground(buttonColor);
deleteSalesPersonButton.setBackground(buttonColor);
showSalesRecordButton.setBackground(buttonColor);
clearSalesRecordButton.setBackground(buttonColor);
updateAccountButton.setBackground(buttonColor);
logoutButton.setBackground(Color.black);

```

```

showProfileButton.setForeground(Color.WHITE);
updateLoginButton.setForeground(Color.WHITE);
displayProductStockButton.setForeground(Color.WHITE);
addNewProductButton.setForeground(Color.WHITE);
addExistingProductButton.setForeground(Color.WHITE);
deleteProductButton.setForeground(Color.WHITE);
displaySalesPersonsButton.setForeground(Color.WHITE);
addSalesPersonButton.setForeground(Color.WHITE);
deleteSalesPersonButton.setForeground(Color.WHITE);
showSalesRecordButton.setForeground(Color.WHITE);
clearSalesRecordButton.setForeground(Color.WHITE);
updateAccountButton.setForeground(Color.WHITE);
logoutButton.setForeground(Color.WHITE);

```

//Show Profile Action Listener.....

```

showProfileButton.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        Owner o1 = new Owner();
        Owner owner = o1.displayAccount();
        if (owner != null)
        {
            // Create a new window to display the account details
            JFrame profileFrame = new JFrame("Owner Profile");
            profileFrame.setSize(400, 300);
            profileFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

            // Create a text area to display the account details
            JTextArea profileTextArea = new JTextArea(owner.toString());
            profileTextArea.setEditable(false);
        }
    }
}

```

```

JScrollPane scrollPane = new JScrollPane(profileTextArea);

// Add the text area to the frame
profileFrame.add(scrollPane);

// Set the frame visible
profileFrame.setLocationRelativeTo(null);
profileFrame.setVisible(true);
}
else
{
    JOptionPane.showMessageDialog(OwnerGUI.this, "Failed to retrieve account
details.", "Error", JOptionPane.ERROR_MESSAGE);
}
}
});

//UpdateLogin Action Listener.....
updateLoginButton.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        while (true)
        {
            JPanel updatePanel = new JPanel();
            updatePanel.setLayout(new GridLayout(4, 2, 10, 10));

            JLabel existingEmailLabel = new JLabel("Existing Email:");
            JTextField existingEmailField = new JTextField();

            JLabel existingPasswordLabel = new JLabel("Existing Password:");
            JPasswordField existingPasswordField = new JPasswordField();

            JLabel newEmailLabel = new JLabel("New Email:");
            JTextField newEmailField = new JTextField();

            JLabel newPasswordLabel = new JLabel("New Password:");
            JPasswordField newPasswordField = new JPasswordField();

            updatePanel.add(existingEmailLabel);
            updatePanel.add(existingEmailField);
            updatePanel.add(existingPasswordLabel);

```

```

        updatePanel.add(existingPasswordField);
        updatePanel.add(newEmailLabel);
        updatePanel.add(newEmailField);
        updatePanel.add(newPasswordLabel);
        updatePanel.add(newPasswordField);

        int result = JOptionPane.showConfirmDialog(OwnerGUI.this, updatePanel, "Update
Login",
            JOptionPane.OK_CANCEL_OPTION, JOptionPane.PLAIN_MESSAGE);

        if (result == JOptionPane.OK_OPTION)
        {
            String existingEmail = existingEmailField.getText();
            String existingPassword = new String(existingPasswordField.getPassword());
            String newEmail = newEmailField.getText();
            String newPassword = new String(newPasswordField.getPassword());

            String verifiedExistingEmail = Utility.verifyEmail(existingEmail);
            String verifiedNewEmail = Utility.verifyEmail(newEmail);

            if (verifiedExistingEmail.equalsIgnoreCase("NULL") ||
verifiedNewEmail.equalsIgnoreCase("NULL"))
            {
                JOptionPane.showMessageDialog(OwnerGUI.this, "Invalid email format.
Please enter a valid email.", "Error", JOptionPane.ERROR_MESSAGE);
                continue; // Restart the loop to get input again
            }

            Owner owner = new Owner();
            boolean updateSuccessful = owner.updateLogin(verifiedExistingEmail,
existingPassword, verifiedNewEmail, newPassword);

            if (updateSuccessful)
            {
                JOptionPane.showMessageDialog(OwnerGUI.this, "Update successful!",
"Success", JOptionPane.INFORMATION_MESSAGE);
            }
            else
            {
                JOptionPane.showMessageDialog(OwnerGUI.this, "Failed to update.", "Error",
JOptionPane.ERROR_MESSAGE);
            }
            break; // Exit the loop after successful update
        }

```

```

        else
        {
            break; // Exit the loop if cancel is clicked
        }
    }
}
});

// Display Product Stock Action Listener.....
displayProductStockButton.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {

        ArrayList<Product> products = Utility.showProductStock();
        if (products.isEmpty())
        {
            JOptionPane.showMessageDialog(null, "No products available.");
        }
        else
        {
            StringBuilder stockText = new StringBuilder();
            for (Product product : products)
            {
                stockText.append(product.display()).append("\n\n");
            }
            showProductStockWindow(stockText.toString());
        }
    }
});

// Method to display product stock in a new window
public void showProductStockWindow(String stockText)
{
    JFrame frame = new JFrame("Product Stock");
    JTextArea stockTextArea = new JTextArea(stockText);
    stockTextArea.setEditable(false);
    JScrollPane scrollPane = new JScrollPane(stockTextArea);
    frame.getContentPane().add(scrollPane);
    frame.setSize(400, 300);
    frame.setLocationRelativeTo(null);
    frame.setVisible(true);
}
});

```

```

// Add New Product Action Listener.....
addNewProductButton.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // Create a new window for input
        JFrame inputFrame = new JFrame("Enter Product Details");
        inputFrame.setSize(520,520);
        inputFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        // Create input fields and labels
        JTextField nameField = new JTextField();
        JTextField priceField = new JTextField();
        JTextField quantityField = new JTextField();
        JTextField weightField = new JTextField();
        JTextField descriptionField = new JTextField();
        JTextField supplierNameField = new JTextField();
        JTextField idField = new JTextField();
        JTextField phoneField = new JTextField();
        JTextField emailField = new JTextField();

        // Create a button for submitting the input
        JButton submitButton = new JButton("Add Product");
        submitButton.addActionListener(new ActionListener()
        {
            @Override
            public void actionPerformed(ActionEvent e)
            {
                // Retrieve the input values
                String name = nameField.getText();
                double price = Double.parseDouble(priceField.getText());
                int quantity = Integer.parseInt(quantityField.getText());
                double weight = Double.parseDouble(weightField.getText());
                String description = descriptionField.getText();
                String supplierName = supplierNameField.getText();
                String id = idField.getText();
                String phone = phoneField.getText();
                String email = emailField.getText();

                // Verify name, phone, and email
            }
        }
    }
}

```

```

String nameError = Utility.verifyName(name);
String phoneError = Utility.verifyPhone(phone);
String emailError = Utility.verifyEmail(email);

if (nameError.equalsIgnoreCase("NULL") ||
phoneError.equalsIgnoreCase("NULL") || emailError.equalsIgnoreCase("NULL"))
{
    // Display error message in a separate window
    JFrame errorFrame = new JFrame("Error");
    JOptionPane.showMessageDialog(errorFrame, "Please enter correct
information. Verification failed.");
}
else
{
    // Create a new supplier and product
    Supplier supplier = new Supplier(supplierName, phone, email, id);
    Product newProduct = new Product(name, price, quantity, weight, description,
supplier);

    Owner owner = new Owner();

    // Call the modified method to add the new product
    owner.addNewProduct(newProduct);

    // Display success message in a separate window
    JFrame successFrame = new JFrame("Success");
    JOptionPane.showMessageDialog(successFrame, "New product added
successfully!");

    // Close the input window
    inputFrame.dispose();
}
}
});

// Create a layout for the input window
JPanel inputPanel = new JPanel(new GridLayout(10, 2));
inputPanel.add(new JLabel("Product Name: "));
inputPanel.add(nameField);
inputPanel.add(new JLabel("Product Price: "));
inputPanel.add(priceField);
inputPanel.add(new JLabel("Product Quantity: "));
inputPanel.add(quantityField);
inputPanel.add(new JLabel("Product Weight: "));
inputPanel.add(weightField);

```

```

        inputPanel.add(new JLabel("Product Description: "));
        inputPanel.add(descriptionField);
        inputPanel.add(new JLabel("Supplier Name: "));
        inputPanel.add(supplierNameField);
        inputPanel.add(new JLabel("Supplier ID: "));
        inputPanel.add(idField);
        inputPanel.add(new JLabel("Supplier Phone: "));
        inputPanel.add(phoneField);
        inputPanel.add(new JLabel("Supplier Email: "));
        inputPanel.add(emailField);
        inputPanel.add(submitButton);

        // Set the layout for the input window
        inputFrame.getContentPane().add(inputPanel);
//        inputFrame.pack();
        inputFrame.setVisible(true);
    }
});

// Add Existing Product Action Listener.....
addExistingProductButton.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // Create a new window for input
        JFrame inputFrame = new JFrame("Enter Existing Product Details");
        inputFrame.setSize(520,520);
        inputFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        // Create input fields and labels
        JTextField nameField = new JTextField();
        JTextField oldPriceField = new JTextField();
        JTextField quantityAddedField = new JTextField();
        JTextField newPriceField = new JTextField();

        // Create a button for submitting the input
        JButton submitButton = new JButton("Modify Product");
        submitButton.addActionListener(new ActionListener()
        {
            @Override
            public void actionPerformed(ActionEvent e)
            {

```



```

        // Retrieve the input values
        String name = nameField.getText();
        double oldPrice = Double.parseDouble(oldPriceField.getText());
        int quantityAdded = Integer.parseInt(quantityAddedField.getText());
        double newPrice = Double.parseDouble(newPriceField.getText());

        // Call the addExistingProduct method
        Owner owner = new Owner();
        boolean success = owner.addExistingProduct(name, oldPrice, quantityAdded,
newPrice);

        // Display error or success dialogue based on the result
        if (success)
        {
            JOptionPane.showMessageDialog(inputFrame, "Product record updated
successfully!");
        }
        else
        {
            JOptionPane.showMessageDialog(inputFrame, "Failed to update product
record. Please try again.");
        }

        // Close the input window
        inputFrame.dispose();
    }
});

// Create a panel and add input fields and submit button to it
JPanel panel = new JPanel();
panel.setLayout(new GridLayout(5, 2));
panel.add(new JLabel("Product Name:"));
panel.add(nameField);
panel.add(new JLabel("Old Price:"));
panel.add(oldPriceField);
panel.add(new JLabel("Quantity Added:"));
panel.add(quantityAddedField);
panel.add(new JLabel("New Price:"));
panel.add(newPriceField);
panel.add(submitButton);

// Add the panel to the input frame
inputFrame.getContentPane().add(panel);
//      inputFrame.pack();

```

```

        inputFrame.setVisible(true);
    }
});

// Delete Product Action Listener.....
deleteProductButton.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // Create a new window for input
        JFrame inputFrame = new JFrame("Delete Product");
        inputFrame.setSize(520,520);
        inputFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        // Create input fields and labels
        JTextField productNameField = new JTextField();
        JTextField supplierNameField = new JTextField();

        // Create a button for submitting the input
        JButton submitButton = new JButton("Delete Product");
        submitButton.addActionListener(new ActionListener()
        {
            @Override
            public void actionPerformed(ActionEvent e)
            {
                // Retrieve the input values
                String productName = productNameField.getText();
                String supplierName = supplierNameField.getText();

                // Call the deleteProduct method
                Owner owner = new Owner();
                boolean success = owner.deleteProduct(productName, supplierName);

                // Display error or success dialogue based on the result
                if (success)
                {
                    JOptionPane.showMessageDialog(inputFrame, "Product deleted
successfully!");
                }
                else
                {
                    JOptionPane.showMessageDialog(inputFrame, "Failed to delete product.
Please check the input values.");
                }
            }
        });
    }
});

```

```

        }

        // Close the input window
        inputFrame.dispose();
    }
});

// Create a panel and add input fields and submit button to it
JPanel panel = new JPanel();
panel.setLayout(new GridLayout(3, 2));
panel.add(new JLabel("Product Name:"));
panel.add(productNameField);
panel.add(new JLabel("Supplier Name:"));
panel.add(supplierNameField);
panel.add(submitButton);

// Add the panel to the input frame
inputFrame.getContentPane().add(panel);
//    inputFrame.pack();
inputFrame.setVisible(true);
}
});

// Display SalesPersons Action Listener.....
displaySalesPersonsButton.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        Owner owner = new Owner();
        ArrayList<SalesPerson> salesPersons = owner.showSalesPerson();

        StringBuilder message = new StringBuilder();
        for (SalesPerson salesPerson : salesPersons)
        {
            message.append(salesPerson.toString()).append("\n");
        }

        JOptionPane.showMessageDialog(null, message.toString(), "SalesPersons",
JOptionPane.INFORMATION_MESSAGE);
    }
});

```

```

// Add SalesPerson Action Listener.....
addSalesPersonButton.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // Create a new window for input
        JFrame inputFrame = new JFrame("Add SalesPerson");
        inputFrame.setSize(520,520);
        inputFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        // Create input fields and labels
        JTextField nameField = new JTextField();
        JTextField phoneField = new JTextField();
        JTextField emailField = new JTextField();
        JTextField idField = new JTextField();
        JTextField passwordField = new JTextField();

        // Create a button for submitting the input
        JButton submitButton = new JButton("Add SalesPerson");
        submitButton.addActionListener(new ActionListener()
        {
            @Override
            public void actionPerformed(ActionEvent e)
            {
                // Retrieve the input values
                String name = nameField.getText();
                String phone = phoneField.getText();
                String email = emailField.getText();
                String id = idField.getText();
                String password = passwordField.getText();

                // Validate input fields
                if (name.isEmpty() || phone.isEmpty() || email.isEmpty())
                {
                    JOptionPane.showMessageDialog(inputFrame, "Please enter all required
fields.");
                    return;
                }
                if (Utility.verifyName(name).equalsIgnoreCase("NULL") ||
Utility.verifyEmail(email).equalsIgnoreCase("NULL") ||
Utility.verifyPhone(phone).equalsIgnoreCase("NULL"))
                {

```

```

        JOptionPane.showMessageDialog(inputFrame, "Please Enter Valid fields.");
        return;
    }

    // Create a SalesPerson object
    SalesPerson salesPerson = new SalesPerson(name, phone, email, id, password);

    // Call the addSalesPerson method
    Owner owner = new Owner();
    boolean success = owner.addSalesPerson(salesPerson);

    // Display error or success dialogue based on the result
    if (success)
    {
        JOptionPane.showMessageDialog(inputFrame, "SalesPerson added
successfully!");
    }
    else
    {
        JOptionPane.showMessageDialog(inputFrame, "Failed to add SalesPerson.
Please try again.");
    }

    // Close the input window
    inputFrame.dispose();
}

});

// Create a panel and add input fields and submit button to it
JPanel panel = new JPanel();
panel.setLayout(new GridLayout(6, 2));
panel.add(new JLabel("Name:"));
panel.add(nameField);
panel.add(new JLabel("Phone:"));
panel.add(phoneField);
panel.add(new JLabel("Email:"));
panel.add(emailField);
panel.add(new JLabel("ID:"));
panel.add(idField);
panel.add(new JLabel("Password:"));
panel.add(passwordField);
panel.add(submitButton);

// Add the panel to the input frame and display it

```

```

        inputFrame.getContentPane().add(panel);
//        inputFrame.pack();
        inputFrame.setVisible(true);
    }
});

// Delete SalesPerson Action Listener.....
deleteSalesPersonButton.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // Create a new window for input
        JFrame inputFrame = new JFrame("Delete SalesPerson");
        inputFrame.setSize(520,520);
        inputFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        // Create input fields and labels
        JTextField nameField = new JTextField();
        JTextField emailField = new JTextField();

        // Create a button for submitting the input
        JButton submitButton = new JButton("Submit");
        submitButton.addActionListener(new ActionListener()
        {
            @Override
            public void actionPerformed(ActionEvent e)
            {
                // Retrieve the input values
                String name = nameField.getText();
                String email = emailField.getText();

                // Call the deleteSalesPerson method
                Owner owner = new Owner();
                boolean success = owner.deleteSalesPerson(name, email);

                // Display error or success dialogue based on the result
                if (success)
                {
                    JOptionPane.showMessageDialog(inputFrame, "SalesPerson deleted
successfully!");
                }
                else

```

```

        {
            JOptionPane.showMessageDialog(inputFrame, "Failed to delete SalesPerson.
Please try again.");
        }

        // Close the input window
        inputFrame.dispose();
    }
});

// Create a panel and add input fields and submit button to it
JPanel panel = new JPanel();
panel.setLayout(new GridLayout(3, 2));
panel.add(new JLabel("Name:"));
panel.add(nameField);
panel.add(new JLabel("Email:"));
panel.add(emailField);
panel.add(submitButton);

// Add the panel to the input frame and display it
inputFrame.getContentPane().add(panel);
//    inputFrame.pack();
inputFrame.setVisible(true);
}
});

// Show Sales Record Action Listener.....
showSalesRecordButton.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        Owner owner = new Owner();
        ArrayList<String> salesRecordDetails = owner.showSalesRecord();

        // Create a new window to display the sales record details
        JFrame salesRecordFrame = new JFrame("Sales Record");
        salesRecordFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        JTextArea salesRecordTextArea = new JTextArea(20, 50);
        salesRecordTextArea.setEditable(false);

        // Add the sales record details to the text area
        for (String record : salesRecordDetails)

```

```

    {
        salesRecordTextArea.append(record + "\n");
    }

    // Add the text area to a scroll pane
    JScrollPane scrollPane = new JScrollPane(salesRecordTextArea);
    salesRecordFrame.getContentPane().add(scrollPane);

    // Set the size and make the window visible
    salesRecordFrame.pack();
    salesRecordFrame.setVisible(true);
}
});

// Clear Sales Record Action Listener.....
clearSalesRecordButton.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        int choice = JOptionPane.showConfirmDialog(null, "Are you sure you want to clear
the sales record?", "Confirm Clear Sales Record", JOptionPane.YES_NO_OPTION);

        if (choice == JOptionPane.YES_OPTION)
        {
            Owner owner = new Owner();
            boolean success = owner.deleteSalesRecord();

            if (success)
            {
                JOptionPane.showMessageDialog(null, "Sales record cleared successfully!");
            }
            else
            {
                JOptionPane.showMessageDialog(null, "Failed to clear sales record. Please try
again.");
            }
        }
    }
});

// Update Account Balance Action Listener.....
updateAccountButton.addActionListener(new ActionListener()

```



```

{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // Create a new window for input
        JFrame inputFrame = new JFrame("Update Account Balance");
        inputFrame.setSize(520,520);
        inputFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        // Create input fields and labels
        JTextField balanceField = new JTextField();
        JTextField revenueField = new JTextField();

        // Create a button for submitting the input
        JButton submitButton = new JButton("Modify Wallet");
        submitButton.addActionListener(new ActionListener()
        {
            @Override
            public void actionPerformed(ActionEvent e)
            {
                // Retrieve the input values
                Double updatedBalance = Double.parseDouble(balanceField.getText());
                Double updatedRevenue = Double.parseDouble(revenueField.getText());

                // Call the updateBalances method
                boolean success = Utility.updateBalances(updatedBalance, updatedRevenue);

                // Display success or error message based on the result
                if (success)
                {
                    JOptionPane.showMessageDialog(inputFrame, "Account updated
successfully!");
                }
                else
                {
                    JOptionPane.showMessageDialog(inputFrame, "Failed to update account.
Please try again.");
                }

                // Close the input window
                inputFrame.dispose();
            }
        });
    }
}

```

```

        // Create a panel and add input fields and submit button to it
        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(3, 2));
        panel.add(new JLabel("New Balance:"));
        panel.add(balanceField);
        panel.add(new JLabel("New Revenue:"));
        panel.add(revenueField);
        panel.add(submitButton);

        // Add the panel to the input frame and display it
        inputFrame.getContentPane().add(panel);
//        inputFrame.pack();
        inputFrame.setVisible(true);
    }
});

// Logout button action listener.....
logoutButton.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        optionsFrame.dispose(); // Close the options window
        OwnerGUI ownerGUI = new OwnerGUI(); // Open the login window
    }
});

optionsPanel.add(showProfileButton);
optionsPanel.add(updateLoginButton);
optionsPanel.add(displayProductStockButton);
optionsPanel.add(addNewProductButton);
optionsPanel.add(addExistingProductButton);
optionsPanel.add(deleteProductButton);
optionsPanel.add(displaySalesPersonsButton);
optionsPanel.add(addSalesPersonButton);
optionsPanel.add(deleteSalesPersonButton);
optionsPanel.add(showSalesRecordButton);
optionsPanel.add(clearSalesRecordButton);
optionsPanel.add(updateAccountButton);
optionsPanel.add(logoutButton);

```

```

optionsFrame.add(optionsPanel, BorderLayout.CENTER);
optionsFrame.setLocationRelativeTo(null);
optionsFrame.setVisible(true);
}
}

```

Class SalesPersonGUI

```

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.Scanner;

public class SalesPersonGUI extends JFrame
{
    private JTextField usernameField;
    private JPasswordField passwordField;

    SalesPersonGUI()
    {
        this.setVisible(true);
        this.setSize(400, 200);
        this.setTitle("SalesPerson Login");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setResizable(false);
        this.setLayout(new BorderLayout());

        ImageIcon logo = new ImageIcon("Logo.png");
        this.setIconImage(logo.getImage());

        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(4, 2, 10, 10));

        JLabel appNameLabel = new JLabel("My Application");
        appNameLabel.setFont(new Font("Arial", Font.BOLD, 18));
        appNameLabel.setHorizontalAlignment(SwingConstants.CENTER);

        JLabel nameLabel = new JLabel("Enter Email:");

```

```

usernameField = new JTextField();

JLabel passwordLabel = new JLabel("Enter Password:");
passwordField = new JPasswordField();

JButton loginButton = new JButton("Login");
JButton mainMenuButton = new JButton("Main Menu");

// Styling options
Font buttonFont = new Font("Arial", Font.BOLD, 14);
Color buttonColor = new Color(0, 128, 0);

appNameLabel.setForeground(buttonColor);
mainMenuButton.setFont(buttonFont);

mainMenuButton.setBackground(buttonColor);
mainMenuButton.setForeground(Color.WHITE);

// Main Menu button action listener
mainMenuButton.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        dispose(); // Close the login window
        MainFrame mainFrame = new MainFrame(); // Open the main menu window
    }
});

// Login button action listener
loginButton.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        String username = usernameField.getText();
        String password = new String(passwordField.getPassword());
        SalesPerson salesPerson = new SalesPerson();
        boolean loginSuccessful = salesPerson.login(username, password);
        if (loginSuccessful)
        {
            dispose(); // Close the login window
            showOwnerOptions(); // Open the owner options window
        }
    }
});

```

```

        else
        {
            JOptionPane.showMessageDialog(SalesPersonGUI.this, "Invalid username or
password. Please try again.", "Login Failed", JOptionPane.ERROR_MESSAGE);
        }
    }
});

panel.add(appNameLabel);
panel.add(new JLabel()); // Empty label for spacing
panel.add(nameLabel);
panel.add(usernameField);
panel.add(passwordLabel);
panel.add(passwordField);
panel.add(loginButton);
panel.add(mainMenuButton);

add(panel, BorderLayout.CENTER);
pack();
setLocationRelativeTo(null);
}

private void showOwnerOptions()
{
    JFrame optionsFrame = new JFrame();
    optionsFrame.setTitle("SalesPerson Options");
    optionsFrame.setSize(900, 400);
    optionsFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    optionsFrame.setLayout(new BorderLayout());

    JPanel optionsPanel = new JPanel();
    optionsPanel.setLayout(new GridLayout(4, 2, 10, 10));
    optionsPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

    JButton updateLoginButton = new JButton("Update Login");
    JButton displayProductStockButton = new JButton("Display Product Stock");
    JButton startOrder = new JButton("Start Order");
    JButton logoutButton = new JButton("Logout");

    // Styling options
    Font buttonFont = new Font("Arial", Font.BOLD, 18);
    Color buttonColor = new Color(0, 128, 0);

    JButton[] buttons =

```

```

{
    updateLoginButton,
    displayProductStockButton,
    startOrder,
    logoutButton
};

FontMetrics fontMetrics = updateLoginButton.getFontMetrics(buttonFont);
int maxButtonWidth = 0;

for (JButton button : buttons)
{
    int buttonWidth = fontMetrics.stringWidth(button.getText());
    maxButtonWidth = Math.max(maxButtonWidth, buttonWidth);
    button.setFont(buttonFont);
    button.setBackground(buttonColor);
    button.setForeground(Color.WHITE);
}

Dimension buttonSize = new Dimension(maxButtonWidth + 20, 40);

for (JButton button : buttons)
{
    button.setPreferredSize(buttonSize);
    optionsPanel.add(button);
}

updateLoginButton.setBackground(buttonColor);
displayProductStockButton.setBackground(buttonColor);
startOrder.setBackground(buttonColor);
logoutButton.setBackground(Color.black);

updateLoginButton.setForeground(Color.WHITE);
displayProductStockButton.setForeground(Color.WHITE);
startOrder.setForeground(Color.WHITE);
logoutButton.setForeground(Color.WHITE);

//UpdateLogin Action Listener.....
updateLoginButton.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {

```

```

while (true)
{
    JPanel updatePanel = new JPanel();
    updatePanel.setLayout(new GridLayout(4, 2, 10, 10));

    JLabel existingEmailLabel = new JLabel("Existing Email:");
    JTextField existingEmailField = new JTextField();

    JLabel existingPasswordLabel = new JLabel("Existing Password:");
    JPasswordField existingPasswordField = new JPasswordField();

    JLabel newEmailLabel = new JLabel("New Email:");
    JTextField newEmailField = new JTextField();

    JLabel newPasswordLabel = new JLabel("New Password:");
    JPasswordField newPasswordField = new JPasswordField();

    updatePanel.add(existingEmailLabel);
    updatePanel.add(existingEmailField);
    updatePanel.add(existingPasswordLabel);
    updatePanel.add(existingPasswordField);
    updatePanel.add(newEmailLabel);
    updatePanel.add(newEmailField);
    updatePanel.add(newPasswordLabel);
    updatePanel.add(newPasswordField);

    int result = JOptionPane.showConfirmDialog(SalesPersonGUI.this, updatePanel,
"Update Login",
        JOptionPane.OK_CANCEL_OPTION, JOptionPane.PLAIN_MESSAGE);

    if (result == JOptionPane.OK_OPTION)
    {
        String existingEmail = existingEmailField.getText();
        String existingPassword = new String(existingPasswordField.getPassword());
        String newEmail = newEmailField.getText();
        String newPassword = new String(newPasswordField.getPassword());

        String verifiedExistingEmail = Utility.verifyEmail(existingEmail);
        String verifiedNewEmail = Utility.verifyEmail(newEmail);

        if (verifiedExistingEmail.equalsIgnoreCase("NULL") ||
verifiedNewEmail.equalsIgnoreCase("NULL"))
        {
            JOptionPane.showMessageDialog(SalesPersonGUI.this, "Invalid email format.

```

```

Please enter a valid email.", "Error", JOptionPane.ERROR_MESSAGE);
        continue; // Restart the loop to get input again
    }

    SalesPerson salesPerson = new SalesPerson();
    boolean updateSuccessful = salesPerson.updateLogin(verifiedExistingEmail,
existingPassword, verifiedNewEmail, newPassword);

    if (updateSuccessful)
    {
        JOptionPane.showMessageDialog(SalesPersonGUI.this, "Update successful!",
"Success", JOptionPane.INFORMATION_MESSAGE);
    }
    else
    {
        JOptionPane.showMessageDialog(SalesPersonGUI.this, "Failed to update.",
"Error", JOptionPane.ERROR_MESSAGE);
    }
    break; // Exit the loop after successful update
}
else
{
    break; // Exit the loop if cancel is clicked
}
}
}
});

// Display Product Stock Action Listener.....
displayProductStockButton.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        ArrayList<Product> products = Utility.showProductStock();
        if (products.isEmpty())
        {
            JOptionPane.showMessageDialog(null, "No products available.");
        }
        else
        {
            StringBuilder stockText = new StringBuilder();
            for (Product product : products)
            {

```



```

        stockText.append(product.display()).append("\n\n");
    }
    showProductStockWindow(stockText.toString());
}
}

```

```

// Method to display product stock in a new window
public void showProductStockWindow(String stockText)
{
    JFrame frame = new JFrame("Product Stock");
    JTextArea stockTextArea = new JTextArea(stockText);
    stockTextArea.setEditable(false);
    JScrollPane scrollPane = new JScrollPane(stockTextArea);
    frame.getContentPane().add(scrollPane);
    frame.setSize(400, 300);
    frame.setLocationRelativeTo(null);
    frame.setVisible(true);
}
});

```

```

// ActionListener for the "Start Order" button
startOrder.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // Create a separate window to input salesperson details
        JFrame salespersonDetailsWindow = new JFrame("SalesPerson Details");
        salespersonDetailsWindow.setSize(500, 300);

salespersonDetailsWindow.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        salespersonDetailsWindow.setLayout(new GridBagLayout());

        // Create a GridBagConstraints object for consistent component placement
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.gridx = 0;
        gbc.gridy = 0;
        gbc.insets = new Insets(5, 5, 5, 5); // Add spacing between components

        // Create labels and text fields for salesperson details
        salespersonDetailsWindow.add(new JLabel("SalesPerson Name:"), gbc);
        gbc.gridx++;
    }
});

```

```

        JTextField salespersonNameField = new JTextField();
        salespersonNameField.setPreferredSize(new Dimension(200, 30)); // Set preferred size
for text field
        salespersonDetailsWindow.add(salespersonNameField, gbc);

// Create confirm button
        gbc.gridx = 0;
        gbc.gridy++;
        gbc.gridwidth = 2; // Span the button across two columns
        gbc.anchor = GridBagConstraints.CENTER; // Center-align the button

        JButton confirmButton = new JButton("Confirm");
        salespersonDetailsWindow.add(confirmButton, gbc);

// Set action listener for the confirm button
        confirmButton.addActionListener(new ActionListener()
        {
            @Override
            public void actionPerformed(ActionEvent e)
            {
                // Get salesperson name from the text field
                String salespersonName = salespersonNameField.getText();

                // Close the salesperson details window
                salespersonDetailsWindow.dispose();

                // Create a separate window to input customer details and product information
                JFrame customerDetailsWindow = new JFrame("Customer Details");
                customerDetailsWindow.setSize(500, 900);

customerDetailsWindow.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
                customerDetailsWindow.setLayout(new GridBagLayout());

                // Create a GridBagConstraints object for consistent component placement
                GridBagConstraints gbc = new GridBagConstraints();
                gbc.gridx = 0;
                gbc.gridy = 0;
                gbc.insets = new Insets(5, 5, 5, 5); // Add spacing between components

                // Create labels and text fields for customer details
                customerDetailsWindow.add(new JLabel("Customer Name:"), gbc);
                gbc.gridx++;
                JTextField nameField = new JTextField();
                nameField.setPreferredSize(new Dimension(200, 30)); // Set preferred size for text

```

field

```
customerDetailsWindow.add(nameField, gbc);
gbc.gridx = 0;
gbc.gridy++;
customerDetailsWindow.add(new JLabel("Customer Phone:"), gbc);
gbc.gridx++;
JTextField phoneField = new JTextField();
phoneField.setPreferredSize(new Dimension(200, 30)); // Set preferred size for
```

text field

```
customerDetailsWindow.add(phoneField, gbc);

gbc.gridx = 0;
gbc.gridy++;
customerDetailsWindow.add(new JLabel("Customer Email:"), gbc);
gbc.gridx++;
JTextField emailField = new JTextField();
emailField.setPreferredSize(new Dimension(200, 30)); // Set preferred size for
```

text field

```
customerDetailsWindow.add(emailField, gbc);

gbc.gridx = 0;
gbc.gridy++;
customerDetailsWindow.add(new JLabel("Customer ID:"), gbc);
gbc.gridx++;
JTextField idField = new JTextField();
idField.setPreferredSize(new Dimension(200, 30)); // Set preferred size for text
```

field

```
customerDetailsWindow.add(idField, gbc);

gbc.gridx = 0;
gbc.gridy++;
customerDetailsWindow.add(new JLabel("Payment Method:"), gbc);
gbc.gridx++;
JTextField paymentField = new JTextField();
paymentField.setPreferredSize(new Dimension(200, 30)); // Set preferred size for
```

text field

```
customerDetailsWindow.add(paymentField, gbc);

// Create labels and text fields for product information
gbc.gridx = 0;
gbc.gridy++;
customerDetailsWindow.add(new JLabel("Product Name"), gbc);
gbc.gridx++;
customerDetailsWindow.add(new JLabel("Quantity"), gbc);
```

```

    JTextField[] productFields = new JTextField[10];
    JTextField[] quantityFields = new JTextField[10];

    for (int i = 0; i < 10; i++)
    {
        gbc.gridx = 0;
        gbc.gridy++;
        productFields[i] = new JTextField();
        productFields[i].setPreferredSize(new Dimension(200, 30)); // Set preferred
size for text field
        customerDetailsWindow.add(productFields[i], gbc);
        gbc.gridx++;
        quantityFields[i] = new JTextField();
        quantityFields[i].setPreferredSize(new Dimension(200, 30)); // Set preferred
size for text field
        customerDetailsWindow.add(quantityFields[i], gbc);
    }
    // Create confirm button
    gbc.gridx = 0;
    gbc.gridy++;
    gbc.gridwidth = 2; // Span the button across two columns
    gbc.anchor = GridBagConstraints.CENTER; // Center-align the button

    JButton confirmButton = new JButton("Confirm");
    customerDetailsWindow.add(confirmButton, gbc);

    // Set action listener for the confirm button in the customer details window
    confirmButton.addActionListener(new ActionListener()
    {
        @Override
        public void actionPerformed(ActionEvent e)
        {
            // Get customer details from the text fields
            String name = nameField.getText();
            String phone = phoneField.getText();
            String email = emailField.getText();
            String id = idField.getText();
            String paymentMethod = paymentField.getText();

            // Get product information from the text fields and create Product objects
            ArrayList<Product> selectedProductsList = new ArrayList<>();
            for (int i = 0; i < 10; i++)
            {

```

```

String productName = productFields[i].getText();
String quantityText = quantityFields[i].getText();

// Check if both the product name and quantity are empty
if (productName.isEmpty() && quantityText.isEmpty())
{
    break; // Stop processing if both fields are empty
}

// Check if the quantity field is empty or not a valid number
if (productName.isEmpty() || quantityText.isEmpty())
{
    JOptionPane.showMessageDialog(null, "Please fill in all product
information.", "Error", JOptionPane.ERROR_MESSAGE);
    return; // Stop further processing
}

int quantity;
try
{
    quantity = Integer.parseInt(quantityText);
}
catch (NumberFormatException ex)
{
    JOptionPane.showMessageDialog(null, "Invalid quantity for product: "
+ productName, "Error", JOptionPane.ERROR_MESSAGE);
    return; // Stop further processing
}

Product product = new Product();
product.setProductName(productName);
product.setQuantity(quantity);
selectedProductsList.add(product);
}

// Perform the purchase
SalesPerson salesPerson = new SalesPerson();
Receipt receipt = salesPerson.startPurchase(salespersonName, name, phone,
email, id, selectedProductsList, paymentMethod);

// Display the receipt in a dialog box
JOptionPane.showMessageDialog(null, receipt.toString(), "Receipt",
JOptionPane.INFORMATION_MESSAGE);

```

```

        // Clear the selected products list
        selectedProductsList.clear();

        // Close the customer details window
        customerDetailsWindow.dispose();
    }
});

// Make the customer details window visible
customerDetailsWindow.setVisible(true);
}
});

// Make the salesperson details window visible
salespersonDetailsWindow.setVisible(true);
}
});

// Logout button action listener.....
logoutButton.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        optionsFrame.dispose(); // Close the options window
        OwnerGUI ownerGUI = new OwnerGUI(); // Open the login window
    }
});

optionsPanel.add(updateLoginButton);
optionsPanel.add(startOrder);
optionsPanel.add(displayProductStockButton);
optionsPanel.add(logoutButton);

optionsFrame.add(optionsPanel, BorderLayout.CENTER);
optionsFrame.setLocationRelativeTo(null);
optionsFrame.setVisible(true);
}
}

```

Class Diagram

