Data Structures Project

Inventory Management System

Done By:

Youssef Abdelaleem – 20100503

Abdelsalam Ahmed – 20100242

Rowan Badr – 20100239

Main Focus of the Project

The Main goal is to build a piece of software that tracks inventory, adds product values, and overall management of the inventory system, while also creating a user interface that allows a user to purchase and add items to cart. This was done using Java programming language, since it allows for defining classes, which helped while using our data structures of choice : (Queue, Built In ArrayList), as well as custom classes for objects like Item Product, etc.

Technologies used:

1. Java Swing for UI Design and Development
2. Intellij Idea for its superior IntelliSense technology and ease of use

Class AdminInterfacePanel

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.List;
2 usages
class AdminInterfacePanel {
    7 usages
    private JPanel adminPanel;
    3 usages
    private JButton generateLowStockReportButton;
    3 usages
    private JButton generateSalesReportButton;
    3 usages
    private JButton addNewProductButton;
    3 usages
    private JButton increaseProductQuantityButton;
    3 usages
    private JButton mainMenuButton;

    5 usages
    private InventoryManagementGUI mainGUI;
```

A graphical user interface (GUI) panel is a part of an inventory management system is represented by the class named AdminInterfacePanel, which is defined in this Java code. The panel has a number of buttons, each of which is linked to a distinct inventory management activity. With the Swing library, the GUI is constructed.

```java
public AdminInterfacePanel(InventoryManagementGUI mainGUI) {
    this.mainGUI = mainGUI;
    initialize();
    addListeners();
}

1 usage
private void initialize() {
    adminPanel = new JPanel(new GridLayout( rows: 5,  cols: 1));

    generateLowStockReportButton = new JButton( text: "Generate Low Stock Report");
    generateSalesReportButton = new JButton( text: "Generate Sales Report");
    addNewProductButton = new JButton( text: "Add New Product");
    increaseProductQuantityButton = new JButton( text: "Increase Product Quantity");
    mainMenuButton = new JButton( text: "Main Menu"); // New button for Main Menu

    adminPanel.add(generateLowStockReportButton);
    adminPanel.add(generateSalesReportButton);
    adminPanel.add(addNewProductButton);
    adminPanel.add(increaseProductQuantityButton);
    adminPanel.add(mainMenuButton); // Added button for Main Menu
}
```

AdminInterfacePanel object is initialized by the constructor It accepts an InventoryManagementGUI parameter called mainGUI, which stands for the primary inventory management graphical user interface and the administrative panel's graphical elements are configured via the initialize() method.

```java
private void addListeners() {
    generateLowStockReportButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) { showLowStock(); }
    });

    generateSalesReportButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) { showSales(); }
    });

    addNewProductButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) { showAddNewProductDialog(); }
    });

    increaseProductQuantityButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) { showIncreaseQuantityDialog(); }
    });

    mainMenuButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) { mainGUI.showMainMenu(); }
    });
```

This method is Attaching action listeners to different graphical user interface (GUI) buttons is the responsibility of this function. Every button on the inventory system is associated with a particular inventory management operation.

```java
private void showSales() {
    JTextArea productsTextArea = new JTextArea();
    productsTextArea.setEditable(false);

    List<List<Product>> cartHistory = CustomerInterfacePanel.getCart_History();

    for(int i = 0; i<cartHistory.size(); i++){
        productsTextArea.append("Sale " + i+1);
        for(int j = 0; j<cartHistory.get(i).size(); i++){
            productsTextArea.append(cartHistory.get(i).get(j).toString() + "\n");
        }
    }

    JScrollPane scrollPane = new JScrollPane(productsTextArea);
    JOptionPane.showMessageDialog( parentComponent: null, scrollPane, title: "Available Products", JOptionPane.PLAIN_MESSAGE);
}
```

This method present a sales report through a graphical user interface. In order to display data on the sales history retrieved from the CustomerInterfacePanel, it generates a text field.

```java
private void showAddNewProductDialog() {
    JTextField productIdField = new JTextField();
    JTextField productNameField = new JTextField();
    JTextField priceField = new JTextField();
    JTextField quantityField = new JTextField();

    JPanel panel = new JPanel(new GridLayout( rows: 4, cols: 2));
    panel.add(new JLabel( text: "Product ID:"));
    panel.add(productIdField);
    panel.add(new JLabel( text: "Product Name:"));
    panel.add(productNameField);
    panel.add(new JLabel( text: "Price:"));
    panel.add(priceField);
    panel.add(new JLabel( text: "Quantity:"));
    panel.add(quantityField);

    int result = JOptionPane.showConfirmDialog( parentComponent: null, panel, title: "Add New Product", JOptionPane.OK_CANCEL_OPTION);
    if (result == JOptionPane.OK_OPTION) {
        try {
            int productId = Integer.parseInt(productIdField.getText());
            String productName = productNameField.getText();
            double price = Double.parseDouble(priceField.getText());
            int quantity = Integer.parseInt(quantityField.getText());

            Product newProduct = new Product(productId, productName, price, quantity);
            mainGUI.getInventory().addProduct(newProduct);
            JOptionPane.showMessageDialog( parentComponent: null, message: "New product added successfully!");
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog( parentComponent: null,
                    message: "Invalid input. Please enter numeric values for price and quantity.");
        }
```

This method is to show the administrator a dialog box where they can enter information about a new product. After confirming the information, a new Product object is created and added to the inventory.

```java
private void showLowStock() {
    JTextArea productsTextArea = new JTextArea();
    productsTextArea.setEditable(false);

    List<Product> ListProd = mainGUI.getInventory().sortProductByStock();
    for (int i = 0; i < ListProd.size(); i++) {
        if(ListProd.get(i).quantityInStock > 10) {
            productsTextArea.append(ListProd.get(i).toString() + "\n");
        }
        else {
            productsTextArea.append(ListProd.get(i).toString() + "LOW STOCK!!\n");
        }
    }

    JScrollPane scrollPane = new JScrollPane(productsTextArea);
    JOptionPane.showMessageDialog( parentComponent: null, scrollPane, title: "Available Products", JOptionPane.PLAIN_MESSAGE);
}
```

A low stock report is created by the showLowStock() method and presented in a message dialog. It provides a list of inventory items with a

```java
private void showIncreaseQuantityDialog() {
    JTextField productIdField = new JTextField();
    JTextField quantityField = new JTextField();
    JPanel panel = new JPanel(new GridLayout( rows: 2,  cols: 2));
    panel.add(new JLabel( text: "Product ID:"));
    panel.add(productIdField);
    panel.add(new JLabel( text: "Quantity to Increase:"));
    panel.add(quantityField);
    int result = JOptionPane.showConfirmDialog( parentComponent: null, panel,  title: "Increase Product Quantity",
            JOptionPane.OK_CANCEL_OPTION);
    if (result == JOptionPane.OK_OPTION) {
        try {
            int productId = Integer.parseInt(productIdField.getText());
            int quantityToAdd = Integer.parseInt(quantityField.getText());

            Product existingProduct = mainGUI.getInventory().getProductInfo(productId);
            if (existingProduct != null) {
                existingProduct.quantityInStock += quantityToAdd;
                JOptionPane.showMessageDialog( parentComponent: null,  message: "Product quantity increased successfully!");
            } else {
                JOptionPane.showMessageDialog( parentComponent: null,  message: "Product not found with ID: " + productId);
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog( parentComponent: null,
                    message: "Invalid input. Please enter numeric values for product ID and quantity.");
        }
    }
}
1 usage
public JPanel getPanel() { return adminPanel; }
```

quantity in stock below a given cutoff (10 units in this example).

A dialog box allowing the administrator to enter data for increasing the amount of an existing product in the inventory is displayed by the showIncreaseQuantityDialog() method and the adminPanel instance is returned by the getPanel() getter function. Through this function, the panel can be accessed for integration by other classes or the main GUI.

Class CustomerInterfacePanel

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;
3 usages
class CustomerInterfacePanel {
    6 usages
    private JPanel customerPanel;
    3 usages
    private JButton displayProductsButton;
    3 usages
    private JButton addToCartButton;
    3 usages
    private JButton placeOrderButton;
    3 usages
    private JButton mainMenuButton; // New button for Main Menu

    6 usages
    private InventoryManagementGUI mainGUI;
    6 usages
    private List<Product> cart;

    2 usages
    private static List<List<Product>> Cart_History;
```

A basic inventory management system's customer interface is represented by the CustomerInterfacePanel Java class, which is defined in the provided code. This class creates a graphical user interface (GUI) with buttons so that users can interact with the system using Swing components.

```java
public CustomerInterfacePanel(InventoryManagementGUI mainGUI) {
    this.mainGUI = mainGUI;
    initialize();
    addListeners();
    cart = new ArrayList<>();
}

1 usage
public static List<List<Product>> getCart_History() { return Cart_History; }

1 usage
private void initialize() {
    customerPanel = new JPanel(new GridLayout( rows: 4,  cols: 1));

    displayProductsButton = new JButton( text: "Display Available Products");
    addToCartButton = new JButton( text: "Add to Cart");
    placeOrderButton = new JButton( text: "Place Order");
    mainMenuButton = new JButton( text: "Main Menu"); // New button for Main Menu

    customerPanel.add(displayProductsButton);
    customerPanel.add(addToCartButton);
    customerPanel.add(placeOrderButton);
    customerPanel.add(mainMenuButton); // Added button for Main Menu
}
```

An instance of the CustomerInterfacePanel class, which represents the customer interface in an inventory management system, is initialized by the constructor.

The Cart_History property is the result of this static procedure. It eliminates the requirement for an instance of the CustomerInterfacePanel class in order for external classes to access the static field.

```
1 usage
private void addListeners() {
    displayProductsButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) { showAvailableProducts(); }
    });

    addToCartButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) { showAddToCartDialog(); }
    });

    placeOrderButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) { placeOrder(); }
    });

    mainMenuButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) { mainGUI.showMainMenu(); }
    });
}
```

The CustomerInterfacePanel class's graphical user interface (GUI) components are configured via the initialize() method.

The CustomerInterfacePanel class's buttons have action listeners attached to them thanks to the addListeners() method. Every button has a specific function that is activated when it is clicked.

```
private void showAvailableProducts() {
    JTextArea productsTextArea = new JTextArea();
    productsTextArea.setEditable(false);

    for (Product product : mainGUI.getInventory().getProducts()) {
        productsTextArea.append(product.toString() + "\n");
    }

    JScrollPane scrollPane = new JScrollPane(productsTextArea);
    JOptionPane.showMessageDialog( parentComponent: null, scrollPane,  title: "Available Products", JOptionPane.PLAIN_MESSAGE);
}
```

A dialog box with a list of the products in the inventory that are available is displayed by the showAvailableProducts() function.

```
1 usage
private void showAddToCartDialog() {
    JTextField productIdField = new JTextField();
    JTextField quantityField = new JTextField();

    JPanel panel = new JPanel(new GridLayout( rows: 2,  cols: 2));
    panel.add(new JLabel( text: "Product ID:"));
    panel.add(productIdField);
    panel.add(new JLabel( text: "Quantity:"));
    panel.add(quantityField);

    int result = JOptionPane.showConfirmDialog( parentComponent: null, panel,  title: "Add to Cart", JOptionPane.OK_CANCEL_OPTION);
    if (result == JOptionPane.OK_OPTION) {
        try {
            int productId = Integer.parseInt(productIdField.getText());
            int quantity = Integer.parseInt(quantityField.getText());

            Product product = mainGUI.getInventory().getProductInfo(productId);
            if (product != null && product.quantityInStock >= quantity) {
                cart.add(new Product(product.productId, product.productName, product.price, quantity));
                JOptionPane.showMessageDialog( parentComponent: null,  message: "Added to cart successfully!");
            } else {
                JOptionPane.showMessageDialog( parentComponent: null,  message: "Product not available or insufficient stock.");
            }
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog( parentComponent: null,
                    message: "Invalid input. Please enter numeric values for product ID and quantity.");
        }
    }
}
```

The showAddToCartDialog() function makes it easier for customers to interact and add items to their shopping carts. A dialog box asking the user to enter the product ID and quantity is displayed.

```
private void placeOrder() {
    if (cart.isEmpty()) {
        JOptionPane.showMessageDialog( parentComponent: null,  message: "Cart is empty. Please add items before placing an order.");
        return;
    }

    String customerName = JOptionPane.showInputDialog("Enter Your Name:");
    if (customerName != null && !customerName.isEmpty()) {
        int orderId = mainGUI.getOrderQueue().getQueueSize() + 1;
        Order order = new Order(orderId, customerName);

        for (Product product : cart) {
            order.addProduct(product, product.quantityInStock);
            product.quantityInStock -= product.quantityInStock; // Deduct stock from inventory
        }

        mainGUI.getOrderQueue().enqueueOrder(order);
        Cart_History.add(cart);
        cart.clear(); // Clear the cart after placing the order
        JOptionPane.showMessageDialog( parentComponent: null,  message: "Order placed successfully. Order ID: " + orderId);
    } else {
        JOptionPane.showMessageDialog( parentComponent: null,  message: "Invalid customer name. Please enter a valid name.");
    }
}

1 usage
public JPanel getPanel() { return customerPanel; }
```

The customer's ability to place an order is handled by the placeOrder() method. It asks for the user's name, adds the items from the cart to an order, removes stock from the inventory, queues the order, and gives feedback on how well the order was placed.

External classes are able to access and show the customer interface panel because the getPanel() method returns the customerPanel.

Class Inventory

```java
import java.util.ArrayList;
import java.util.List;
3 usages
class Inventory {
    8 usages
    private List<Product> products;
    1 usage
    public Inventory() {
        products = new ArrayList<>();
        // Initialize with some sample products
        products.add(new Product( productId: 1,  productName: "Product A",  price: 10.0,  quantityInStock: 50));
        products.add(new Product( productId: 2,  productName: "Product B",  price: 15.0,  quantityInStock: 30));
        products.add(new Product( productId: 3,  productName: "Product C",  price: 20.0,  quantityInStock: 20));
    }
    1 usage
    public List<Product> getProducts() { return products; }
```

Products that are offered at a store are represented by the Inventory class. It gives techniques to access the product list and starts with some sample products.

private List<Product> products: A list where products of the Product class instances that correspond to inventory items are kept.

public Inventory():starts the inventory by making a blank product list and adding a few sample items to it.

public List<Product> The inventory's product list can be accessed via the getProducts() function.

```java
public Product getProductInfo(int productId) {
    for (Product product : products) {
        if (product.productId == productId) {
            return product;
        }
    }
    return null;
}
1 usage
public void addProduct(Product product) { products.add(product); }
1 usage
public List<Product> sortProductByStock() {
    List<Product> sortedList = products;
    for (int i = 0; i < sortedList.size(); i++) {
        for (int j = 1; j < sortedList.size(); j++) {
            if (sortedList.get(i).quantityInStock > sortedList.get(j).quantityInStock) {
                Product temp = sortedList.get(i);
                sortedList.set(i, sortedList.get(j));
                sortedList.set(j, temp);
            }
        }
    }
    return sortedList;
}
```

public Product getProductInfo(int productId): This method uses the product's unique identifier (productId) to retrieve product information.

public void addProduct(Product product):Adds a new product to the inventory.

public List<Product> Sorts the inventory list of items according to the quantity in stock (quantityInStock) using the sortProductByStock() function.

Class inventoryManagementGUI

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

5 usages
public class inventoryManagementGUI {
    17 usages
    private JFrame frame;
    10 usages
    private JPanel mainPanel;
    6 usages
    private JButton adminButton;
    6 usages
    private JButton customerButton;
    6 usages
    private JButton exitButton;
    2 usages
    private Inventory inventory;
    2 usages
    private OrderQueue orderQueue;
    1 usage
    public inventoryManagementGUI() {
        initialize();
        inventory = new Inventory();
        orderQueue = new OrderQueue();
    }
    1 usage
```

The graphical user interface (GUI) of an inventory management system is represented by this class. It offers a straightforward user interface with buttons to access customer, administrative, and application exit functionalities.

```java
private void initialize() {
    frame = new JFrame( title: "Inventory Management System");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize( width: 500,  height: 350);

    mainPanel = new JPanel(new GridLayout( rows: 3,  cols: 1));

    adminButton = new JButton( text: "Admin");
    customerButton = new JButton( text: "Customer");
    exitButton = new JButton( text: "Exit");

    adminButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) { showAdminInterface(); }
    });
    customerButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) { showCustomerInterface(); }
    });
    exitButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) { System.exit( status: 0); }
    });
    mainPanel.add(adminButton);
    mainPanel.add(customerButton);
    mainPanel.add(exitButton);

    frame.getContentPane().add(mainPanel, BorderLayout.CENTER);
    frame.setVisible(true);
}
```

The graphical user interface (GUI) elements of an inventory management system's main frame are initialized using this approach.

```java
private void showAdminInterface() {
    frame.getContentPane().removeAll();
    frame.repaint();

    AdminInterfacePanel adminPanel = new AdminInterfacePanel( mainGUI: this);
    frame.getContentPane().add(adminPanel.getPanel(), BorderLayout.CENTER);
    frame.setVisible(true);
}
2 usages
private void showCustomerInterface() {
    frame.getContentPane().removeAll();
    frame.repaint();

    CustomerInterfacePanel customerPanel = new CustomerInterfacePanel( mainGUI: this);
    frame.getContentPane().add(customerPanel.getPanel(), BorderLayout.CENTER);
    frame.setVisible(true);
}
```

By using button clicks, these techniques make it easier to move between the graphical user interface (GUI)'s various panels (the Admin and Customer interfaces).

They pass the running instance of InventoryManagementGUI to the AdminInterfacePanel and CustomerInterfacePanel instances, respectively, enabling communication across the panels.

In order to create a dynamic GUI that responds to user inputs, the current content is removed and new panels are added to the content pane.

```java
public void showMainMenu() {
    frame.getContentPane().removeAll();
    frame.repaint();

    mainPanel = new JPanel(new GridLayout( rows: 3,  cols: 1));

    adminButton = new JButton( text: "Admin");
    customerButton = new JButton( text: "Customer");
    exitButton = new JButton( text: "Exit");

    adminButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) { showAdminInterface(); }
    });
    customerButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) { showCustomerInterface(); }
    });
    exitButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) { System.exit( status: 0); }
    });

    mainPanel.add(adminButton);
    mainPanel.add(customerButton);
    mainPanel.add(exitButton);

    frame.getContentPane().add(mainPanel, BorderLayout.CENTER);
    frame.setVisible(true);
}
```

The main menu in the GUI is displayed via this technique, which also gives the user the option to quit the application or switch between the administrative and customer interfaces.

```java
public static void main(String[] args) { SwingUtilities.invokeLater(() -> new inventoryManagementGUI()); }

5 usages
public Inventory getInventory() { return inventory; }

2 usages
public OrderQueue getOrderQueue() { return orderQueue; }
```

The main method serves as the application's entry point.

The order queue and inventory are regulated access points that other classes can use to communicate with these elements in the proper way thanks to the getter methods getInventory and getOrderQueue.

Class OrderQueue

```java
import java.util.ArrayList;
import java.util.List;

3 usages
class OrderQueue {
    5 usages
    private List<Order> queue;

    1 usage
    public OrderQueue() { queue = new ArrayList<>(); }

    1 usage
    public void enqueueOrder(Order order) { queue.add(order); }

    public Order dequeueOrder() {
        if (!queue.isEmpty()) {
            return queue.remove( index: 0);
        }
        return null;
    }

    1 usage
    public int getQueueSize() { return queue.size(); }
}
```

An elementary implementation of a queue for order storing is provided by the OrderQueue class.

An order is added to the end of the queue using the enqueueOrder method.

The order at the front of the queue is removed and returned using the dequeueOrder function.

The order queue's current size is returned via the getQueueSize function.

Class Order

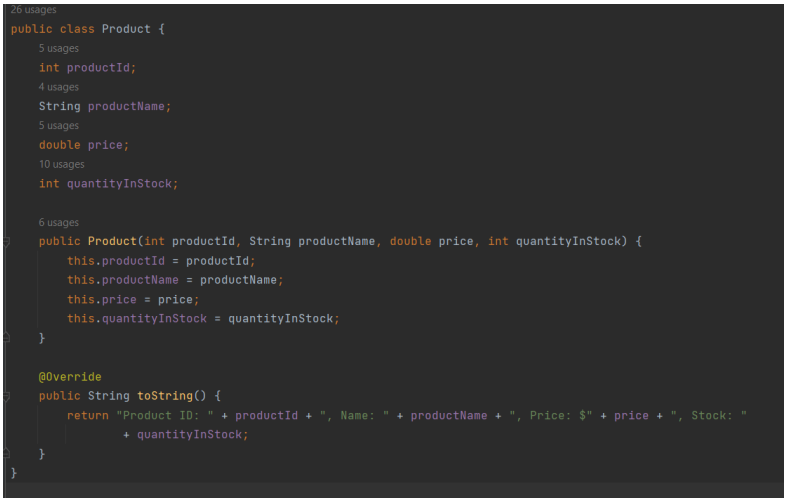
```java
import java.util.ArrayList;
import java.util.List;

5 usages
class Order {
    1 usage
    private int orderId;
    1 usage
    private String customerName;
    2 usages
    private List<Product> productsOrdered;
    1 usage
    private double totalAmount;

    1 usage
    public Order(int orderId, String customerName) {
        this.orderId = orderId;
        this.customerName = customerName;
        this.productsOrdered = new ArrayList<>();
    }

    1 usage
    public void addProduct(Product product, int quantity) {
        productsOrdered.add(new Product(product.productId, product.productName, product.price, quantity));
        totalAmount += product.price * quantity;
    }
}
```

In the inventory management system, an order is represented by the Order class.

An order is initialized by the constructor with an empty list of products, the customer's name, and an order ID.

Products can be added to the order using the addProduct method, which also determines the overall cost based on the quantity of the new products.

Class Product



```java
26 usages
public class Product {
    5 usages
    int productId;
    4 usages
    String productName;
    5 usages
    double price;
    10 usages
    int quantityInStock;

    6 usages
    public Product(int productId, String productName, double price, int quantityInStock) {
        this.productId = productId;
        this.productName = productName;
        this.price = price;
        this.quantityInStock = quantityInStock;
    }

    @Override
    public String toString() {
        return "Product ID: " + productId + ", Name: " + productName + ", Price: $" + price + ", Stock: "
                + quantityInStock;
    }
}
```

A product in an inventory management system is simply represented by the Product class.

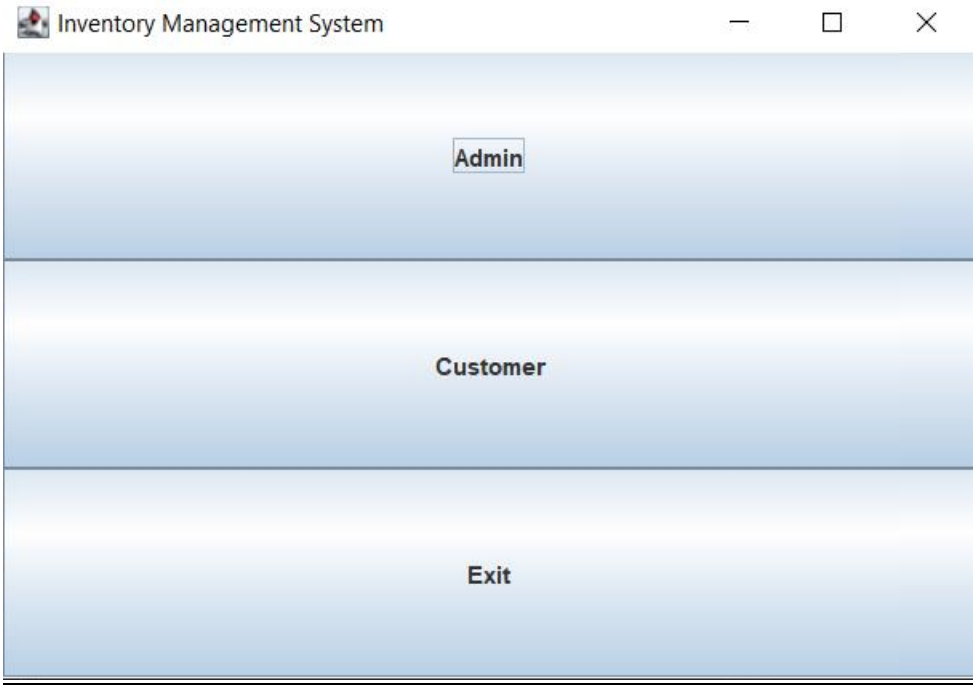
A product's constructor initializes it with certain information.

To produce a formatted string representation of the product, the toString method overrides the Object class's default toString method.

The fundamental characteristics and actions of a product inside the system are contained in this class. For debugging or user interfaces, for example, the toString method is especially helpful for showing product information in an understandable format for humans.

Finally running code

As a admin:



As a customer: