K.RAMAKRISHNAN
COLLEGE OF TECHNOLOGY
An Autonomous Institution
Affiliated to Anna University Chennai, Approved by AICTE New Delhi,
ISO 9001:2015 & ISO 14001:2015 Certified Institution, Accredited with 'A+' grade by NAAC
Samayapuram, Tiruchirappalli – 621 112, Tamilnadu, India.

A Project Report

on

Stock Management System

Submitted in partial fulfillment of requirements for the award of the course

of

EGB1201 – JAVA PROGRAMMING

Under the guidance of

Ms. Hema R., M.E.,

Assistant Professor / Information Technology

Submitted By

THARUN KUMAR M (ECB23114)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY
(Autonomous)

TRICHY - 621112

DECEMBER 2024

K.RAMAKRISHNAN
COLLEGE OF TECHNOLOGY
An Autonomous Institution
Affiliated to Anna University Chennai, Approved by AICTE New Delhi,
ISO 9001:2015 & ISO 14001:2015 Certified Institution, Accredited with 'A+' grade by NAAC
Samayapuram, Tiruchirappalli – 621 112, Tamilnadu, India.

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(Autonomous Institution affiliated to Anna University, Chennai)

TRICHY - 621112

BONAFIDE CERTIFICATE

Certified that this project report on **"STOCK MANAGEMENT SYSTEM"** is the Bonafide work of **THARUN KUMAR M(ECB23114)** who carried out the project work during the academic year 2024 - 2025 under my supervision.

Signature

Ms. HEMA R., M.E.,

SUPERVISOR,

Department of Information Technology,

K. Ramakrishnan College of Technology,

Trichy - 621112

Signature

Dr. SYEDAKBAR S., M.E.,Ph.D.,

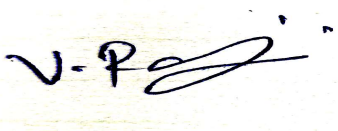
HEAD OF THE DEPARTMENT,

Department of ECE,

K. Ramakrishnan College of Technology,

Trichy - 621112

Submitted for the viva-voce examination held on 07.12.24.



INTERNAL EXAMINER **EXTERNAL EXAMINER**

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

VISION OF THE INSTITUTION

To emerge as a leader among the top institutions in the field of technical education

MISSION OF THE INSTITUTION

- Produce smart technocrats with empirical knowledge who can surmount the global challenges
- Create a diverse, fully-engaged, learner-centric campus environment to provide quality education to the students
- Maintain mutually beneficial partnerships with our alumni, industry, and Professional associations

VISION OF THE DEPARTMENT

To create innovative and socially responsible Electronics and Communication Engineers with design skills and research focus to meet Societal and Industrial needs.

MISSION OF THE DEPARTMENT

- M1: To provide high quality education and professional ethics to students through enhanced learning environment
- M2: To impart a creative environment towards centre of excellence in department with design skill and exposure for research.
- M3: To nurture required employable skills of students to satisfy the industry and social needs with ethical and human values.

PROGRAM EDUCATIONAL OBJECTIVES (PEOS)

- PEO1: Core Knowledge Development: Graduates will have enhanced engineering skills in the field of electronics, communication and interdisciplinary areas to serve the society with global standards.

- PEO2: Professional development: Graduates will apply the technical knowledge for continuous up gradation of their professional skills to become an inimitable employee, researcher or entrepreneur.
- PEO3: Analytical Thinking: Graduates will have analytic and thinking skills to provide the innovative solutions for industry and societal requirements.

PROGRAM OUTCOMES

Engineering students will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

- PSO1: To analyse, design and develop solutions by applying foundational concepts of electronics and communication engineering.
- PSO2: To apply design principles and best practices for developing quality products for scientific and business applications.

ABSTRACT

Effective stock management is crucial for maintaining the efficiency and profitability of any business that deals with inventory. This report presents the design and implementation of a Stock Management System (SMS), a software solution aimed at streamlining the processes of managing inventory, handling orders, and ensuring optimal stock levels. The application is developed using Java and integrates both console-based and graphical user interface (GUI)-based functionalities to cater to diverse user requirements. The system provides several core features, including adding new products with unique identifiers, tracking their quantities and prices, and updating stock levels dynamically. The SMS also supports creating and managing customer orders, ensuring that product availability is verified before order placement. Each order includes a breakdown of items purchased, their quantities, and the total amount. These features are aimed at improving inventory visibility, reducing manual errors, and ensuring seamless operations. Initially, the SMS was implemented as a console-based application using Java's basic I/O capabilities to interact with users. However, to enhance user experience and align with modern standards, the system was extended to include a GUI-based interface using Java Swing and AWT libraries. This transition replaces text-based inputs and outputs with interactive components like buttons, text fields, and labels, making it more user-friendly and visually appealing. The GUI version allows users to easily add products, view available inventory, create orders, and review past transactions through a set of intuitive controls.

The implementation emphasizes modularity and reusability of code, with the system being divided into distinct classes representing products, orders, and the management system itself. Each class encapsulates its functionalities, ensuring a clean and maintainable code structure. Event-driven programming is used in the GUI to respond to user interactions, while the back-end logic remains consistent across both console and GUI implementations. This report discusses the design process, features, and technical challenges faced during the development of the Stock Management System. It also evaluates the system's performance in handling typical inventory management tasks. By leveraging Java's robust programming capabilities and its extensive libraries, this SMS provides a scalable solution suitable for small- to medium-sized businesses. Future enhancements may include database integration for persistent storage, multi-user support, and advanced analytics for decision-making.

ABSTRACT WITH POs AND PSOs MAPPING

ABSTRACT	POs MAPPED	PSOs MAPPED
The **Stock Management System** is a Java-based application developed to streamline the management of inventory and orders. It provides functionality for adding, updating, and viewing products, as well as creating and managing customer orders. Built using **Swing**, this application offers an intuitive Graphical User Interface (GUI) for user interaction. The system employs **Object-Oriented Programming (OOP)** principles, such as encapsulation and polymorphism, to ensure modularity and maintainability. The application is suitable for small to medium-sized businesses looking to automate their stock and sales processes.	**PO1, PO3, PO5, PO9, PO12**	**PSO1, PSO2, PSO3**

Note: 1- Low, 2-Medium, 3- High

SUPERVISOR **HEAD OF THE DEPARTMENT**

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Objective

The objective of the Stock Management System is to develop a reliable, efficient, and user-friendly application that simplifies inventory and order management for businesses.This eliminates traditional, error-prone manual tracking methods, offering instead a computerized system that ensures data accuracy, reliability, and real-time availability. The system is designed to:

1. Allow businesses to add, view, and update product details effectively.
2. Enable order creation while automatically validating stock availability.
3. Provide a graphical user interface (GUI) for seamless interaction.
4. Maintain a record of all orders and product details to ensure transparency and accurate tracking.

This system is particularly beneficial for small- to medium-sized businesses that need a simple yet effective solution to manage their stock and sales without complex technical requirements.

1.2 Overview

The Stock Management System is a desktop application built using Java and AWT (Abstract Window Toolkit). It includes functionalities for:

Product Management: Adding and viewing product information such as ID, name, quantity, and price.

Stock Updates: Modifying the available quantity of products in the inventory.

Order Management: Creating orders with selected products and quantities while checking for stock availability.

Order History: Displaying past orders with product details and total amounts.

This emphasizes modularity, simplicity, and efficiency in design. By leveraging AWT, it provides a platform-independent graphical interface that can run on any system with a Java runtime environment.The use of in-memory data structures such as ArrayList and HashMap makes the system highly efficient for real-time operations like searching, updating, and displaying products and orders.

The Swing GUI makes the application intuitive for users who may not be familiar with command-line interfaces. The system's clear menus and dialog boxes allow for easy data entry and manipulation.

The design is scalable, meaning the system can easily be expanded to support more complex features such as multi-user access, data persistence (e.g., saving to a database), or integration with online platforms for e-commerce.

The system provides user feedback through dialog boxes for error handling. For example, if there is insufficient stock for an order, the system notifies the user with a message, preventing invalid operations.

By keeping track of product quantities and automatically updating them after each order, the system provides real-time inventory management and order tracking.
Order summaries allow businesses to keep track of sales data and maintain proper records.
As the application is currently designed, all data is stored in memory. However, for long-term use, it would be beneficial to implement data persistence using a database (e.g., MySQL, SQLite) or file-based storage to ensure data is not lost when the application is closed.

The system can be expanded to support multiple users. For example, different employees could have access to different parts of the system, such as product management, order management, and reporting.

The reporting functionality could be expanded to include detailed sales analytics, including the ability to generate reports based on product sales, monthly performance, etc.o protect sensitive data,

the system could be enhanced with user authentication and role-based access control, ensuring that only authorized users can perform certain actions (e.g., updating stock or viewing financial reports).

1.3 Java Programming Concepts

Object-Oriented Programming (OOP)

Object-Oriented Programming is a fundamental paradigm in Java, enabling the creation of reusable, maintainable, and modular code. In the Stock Management System, the key concepts of OOP—Encapsulation, Inheritance, Polymorphism, and Abstraction—were implemented to manage products, orders, and inventory.

Encapsulation:

Encapsulation involves bundling data (attributes) and methods that operate on the data into a single unit, often a class. It allows restricting access to certain components of the object and protecting the internal state of the object by exposing only necessary methods.

- **Product Class**: In the Product class, the fields like productId, name, quantity, and price are private, ensuring that they are not directly accessible from outside the class. Methods like getProductId(), getName(), getQuantity(), and setQuantity() are provided to access or modify these fields.

Inheritance:

Inheritance allows one class (subclass) to inherit the properties and behaviors of another class (superclass). Although inheritance is not heavily used in this particular example, it can be extended to introduce more specialized types of products (e.g., ElectronicProduct, FoodProduct) by inheriting from the Product class and adding extra attributes or behaviors.

Polymorphism:

Polymorphism allows a single method to perform different tasks based on the objects it is acting upon. In this case, the toString() method of both the Product and Order classes is an example of method overriding (a form of polymorphism). This allows for different string representations of the objects.

Abstraction:

Abstraction allows hiding the complex implementation details from the user and exposing only the essential functionality. The system abstracts the details of adding products to the inventory or creating orders, allowing users to interact with simple interfaces without worrying about the internal workings.

Swing for GUI Development

Swing is Java's built-in GUI toolkit that provides a set of graphical user interface components for building window-based applications. Swing enables the creation of user-friendly interfaces by providing components such as buttons, labels, text fields, and dialog boxes.

Event Handling

Event handling in Java allows applications to respond to user interactions, such as button clicks or menu selections. In this system, ActionListeners are used to handle user actions when interacting with menu items and dialog boxes.

For example, when a user clicks the "Add Product" menu item, the system triggers the addProductDialog() method to display a form for entering product details.

Exception Handling

Exception handling in Java allows a program to deal with unexpected errors and exceptions in a graceful manner. In the Stock Management System, the application checks for potential errors such as invalid input (e.g., non-numeric quantity or price) and handles them using try-catch blocks.For example, when parsing an integer from user input, a NumberFormatException may be thrown if the input is not a valid number:

CHAPTER 2

PROJECT METHODOLOGY

2.1 Proposed Work

The development of the Stock Management System follows a systematic approach:

1. Requirements Gathering

Understanding the key requirements of a stock management application, including:

- Managing dynamic product inventory.
- Handling stock updates and validations.
- Enabling seamless order creation and history tracking.

2. Design and Architecture

- Define a modular structure with separate classes for products, orders, and the management system.
- Use **AWT** for GUI design and event handling to connect the interface with backend operations.

3. Implementation

- Develop the system using core Java concepts, focusing on functionality, efficiency, and maintainability.
- Build the GUI with AWT components, ensuring a user-friendly interface.

4. Testing and Debugging

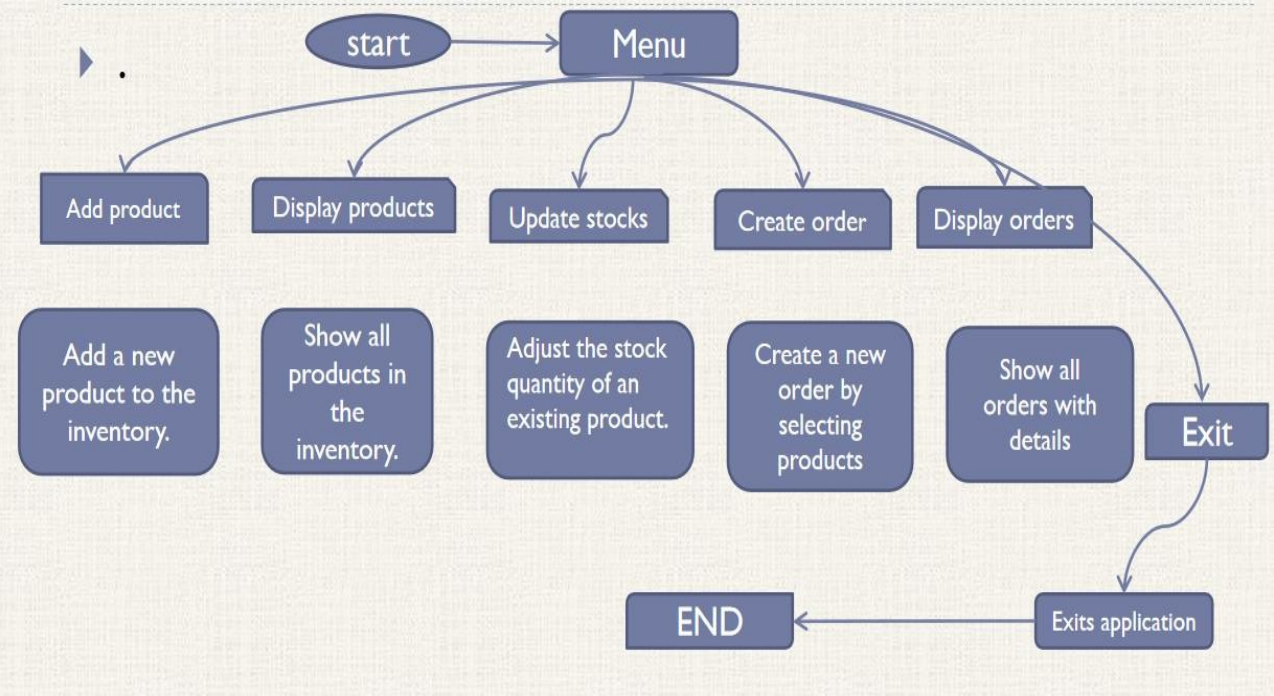
- Test individual modules (product management, stock updates, order creation) for functionality and edge cases.

- Validate GUI responsiveness and error handling.

5. Deployment

Package the application for deployment on any system with a Java runtime environment.

2.2 Block Diagram



CHAPTER 3

MODULE DESCRIPTION

3.1 Product Management

Objective: Add, view, and manage product details.

Explanation:

This module handles the core inventory functionality, allowing users to:

1. Add products by providing details such as ID, name, quantity, and price.
2. View all available products in the inventory, displayed in a formatted list.
3. Ensure no duplicate products are added by validating the product ID.

Implementation:

a. **Class**: Product to encapsulate product attributes.
b. **GUI**: A form to input product details, and a display area (text area) for listing products.

3.2 Stock Update

Objective: Update product quantities in the inventory.

Explanation:

This module ensures that stock levels remain accurate by allowing users to:

1. Search for a product using its ID.
2. Update the quantity by adding or subtracting stock.

Implementation:

a. **GUI**: Text fields for inputting the product ID and the quantity to update.
b. **Validation**: Check if the product ID exists and handle invalid inputs.

3.3 Order Creation

Objective: Create customer orders and manage stock automatically.

Explanation:

This module facilitates the creation of new orders. Users can:

1. Add products to an order by specifying their ID and quantity.
2. Ensure stock availability before adding a product to the order.
3. Automatically deduct the ordered quantity from the stock.

Implementation:

a. **Class**: Order to manage ordered products and calculate the total amount.

b. **GUI**: Text fields for product ID and quantity input, with dynamic updates to the order summary.

3.4 Display Orders

Objective: View all past orders with details.

Explanation:

This module allows users to track completed orders, displaying:

1. The order ID.
2. A list of ordered products and their quantities.
3. The total cost of the order.

Implementation:

a. **GUI**: A display area (text area) to show the list of all orders.

3.5 Graphical User Interface

Objective: Provide a user-friendly interface for all operations.

Explanation:

The GUI is the backbone of user interaction. It includes:

1. A main menu with buttons for each functionality (Add Product, Update Stock, Create Order, Display Orders).
2. Forms and text areas for input and output.
3. Buttons linked to backend operations via event listeners.

Implementation:

a. **AWT Components**: Frame, Button, Label, TextField, TextArea.
b. **Event Handling**: Implemented using ActionListener for all buttons.

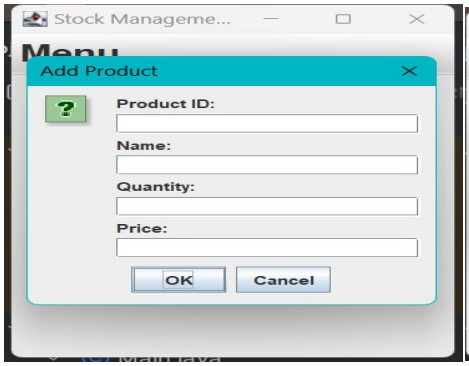
CHAPTER 4

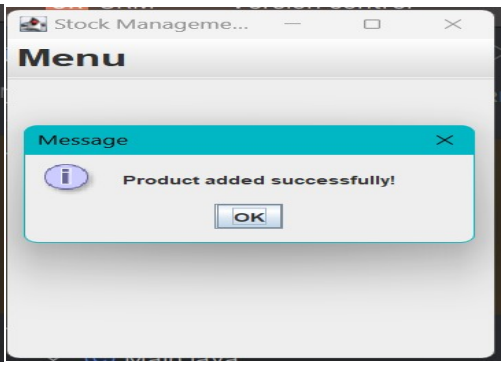
RESULTS AND DISCUSSION

Menu

1. Adding Products

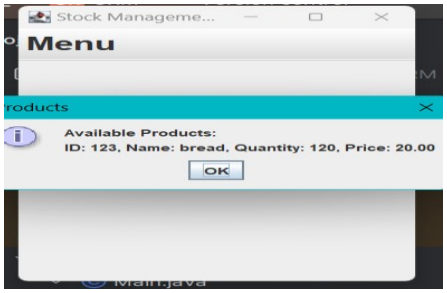
Successfully added products with unique IDs. Duplicate IDs were rejected





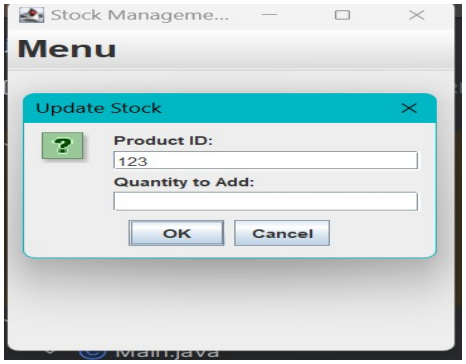
2. View products

Diplays all products available



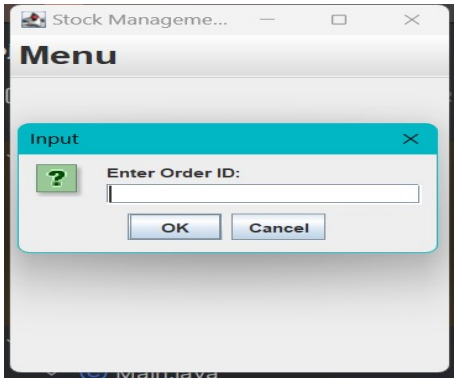
3. Updating stockes

Handled both positive and negative stock updates accurately. Successfully updated.



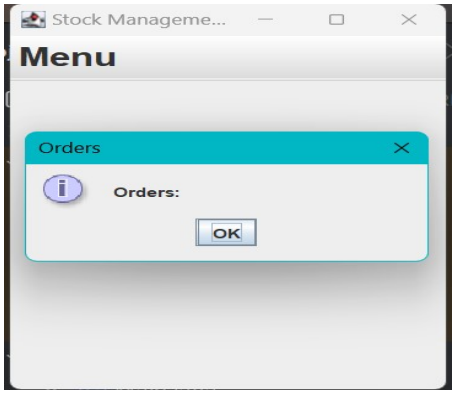
4. Create order

Validated stock availability before adding products to orders. Insufficient stock errors were displayed clearly. Ordered by order ID, Product ID. Bill will be generated.



5. View orders

Displays all orders placed. With amount and quantity.



6. Existing

Exits from the application.

CHAPTER 5

CONCLUSION

The Stock Management System developed using Java effectively applies core programming concepts such as Object-Oriented Programming (OOP), Swing GUI, Java Collections Framework, Event Handling, and Exception Handling to create a robust and user-friendly application for managing product inventory and customer orders. By leveraging OOP principles like Encapsulation, Inheritance, and Polymorphism, the system ensures modular, maintainable code, while the use of Swing provides an intuitive interface for users to interact with. The ArrayList and HashMap data structures are utilized for efficient product and order management, respectively, and event-driven programming allows for real-time response to user actions. Error handling ensures the system remains stable even with invalid inputs. The design is extensible, allowing for future enhancements like adding new product categories or integrating with external databases, making it a scalable solution for businesses.

Key Takeaways:

- Streamlined inventory management processes.
- Accurate stock validation and order processing.
- A simple yet effective GUI for ease of use.

Future Scope:

- Integrate a database for persistent data storage.
- Enhance the GUI with Swing or JavaFX for better aesthetics and functionality.
- Add advanced features like reports, user authentication, and analytics.

REFERENCES

"Java: The Complete Reference" by Herbert Schildt-book

Why Use It?: Comprehensive resource for all Java topics, including GUI development with Swing.**Publisher**: McGraw-Hill Education

Java Swing Tutorial - GeeksforGeeks-website

Why Use It?: Provides step-by-step tutorials and code examples for using Swing in Java.

Oracle Java Documentation**.**

Why Use It?: Official documentation for Swing components and Java UI programming.

Youtube cahnnels: error makes clever, code.io

APPENDIX

(Coding)

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

// Product class remains unchanged
class Product {
    private String productId;
    private String name;
    private int quantity;
    private double price;

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

    public String getProductId()
        {return productId;
    }

    public String getName()
        {return name;
    }

    public int getQuantity()
        {return quantity;
    }

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

    public double getPrice()
        {return price;
    }
```

```
    public void updateStock(int newQuantity)
        {this.quantity += newQuantity;
    }

    @Override
    public String toString() {
        return String.format("ID: %s, Name: %s, Quantity: %d, Price: %.2f", productId, name, quantity,
price);
    }
}

// Order class remains unchanged
class Order {
    private String orderId;
    private Map<Product, Integer> orderedProducts;
    private double totalAmount;

    public Order(String orderId)
        {this.orderId = orderId;
        this.orderedProducts = new HashMap<>();
        this.totalAmount = 0.0;
    }

    public void addProductToOrder(Product product, int quantity)
        {if (product.getQuantity() >= quantity) {
            orderedProducts.put(product, quantity);
            product.setQuantity(product.getQuantity() - quantity);
            totalAmount += product.getPrice() * quantity;
        } else {
            System.out.println("Insufficient stock for product: " + product.getName());
        }
    }

    public double getTotalAmount()
        {return totalAmount;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("Order ID: ").append(orderId).append("\n");
        for (Map.Entry<Product, Integer> entry : orderedProducts.entrySet())
            { sb.append(entry.getKey().getName()).append("                -                Quantity:
").append(entry.getValue()).append("\n");
        }
        sb.append("Total Amount: ").append(totalAmount).append("\n");
```

```
        return sb.toString();
    }
}


public class Main extends JFrame
    {private List<Product> products;
    private List<Order> orders;

    public Main() {
        products = new ArrayList<>();
        orders = new ArrayList<>();
        initializeUI();
    }



    private void initializeUI()
        { setTitle("Stock Management
        System");setSize(600, 600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());
        Font largeFont = new Font("Arial", Font.BOLD, 25);
        // Menu bar
        JMenuBar menuBar = new JMenuBar();

        JMenu menu = new
            JMenu("Menu"){@Override
            public Dimension getPreferredSize() {
                return new Dimension(100, 40); // Set width and height for the menu button

            }
        };
        menu.setFont(largeFont); // Apply larger font to the menu

        // Set the menu bar for the frame
        menuBar.add(menu);
        setJMenuBar(menuBar);
        JMenuItem addProduct = new JMenuItem("Add Product");
        JMenuItem viewProducts = new JMenuItem("View Products");
        JMenuItem updateStock = new JMenuItem("Update Stock");
        JMenuItem createOrder = new JMenuItem("Create Order");
        JMenuItem viewOrders = new JMenuItem("View Orders");
        JMenuItem exit = new JMenuItem("Exit");

        addProduct.setBackground(Color.CYAN);
        addProduct.setForeground(Color.BLACK);
```

```
    viewProducts.setBackground(Color.GREEN);
    viewProducts.setForeground(Color.BLACK);
    updateStock.setBackground(Color.YELLOW);
    updateStock.setForeground(Color.BLACK);
    createOrder.setBackground(Color.ORANGE);
    createOrder.setForeground(Color.BLACK);
    viewOrders.setBackground(Color.PINK);
    viewOrders.setForeground(Color.BLACK);
    exit.setBackground(Color.RED);
    exit.setForeground(Color.WHITE);




    addProduct.setFont(largeFont);
    viewProducts.setFont(largeFont);
    updateStock.setFont(largeFont);
    createOrder.setFont(largeFont);
    viewOrders.setFont(largeFont);
    exit.setFont(largeFont);

    menu.add(addProduct);
    menu.add(viewProducts);
    menu.add(updateStock);
    menu.add(createOrder);
    menu.add(viewOrders);
    menu.addSeparator();
    menu.add(exit);
    menuBar.add(menu);
    setJMenuBar(menuBar);

    // Event Listeners
    addProduct.addActionListener(e -> addProductDialog());
    viewProducts.addActionListener(e -> displayProducts());
    updateStock.addActionListener(e -> updateStockDialog());
    createOrder.addActionListener(e -> createOrderDialog());
    viewOrders.addActionListener(e -> displayOrders());
    exit.addActionListener(e -> System.exit(0));
}

private void addProductDialog()
    { JTextField idField = new JTextField();
    JTextField nameField = new JTextField();
    JTextField qtyField = new JTextField();
    JTextField priceField = new JTextField();
```

```
        Object[] message =
                { "Product ID:",
                idField,"Name:",
                nameField,
                "Quantity:", qtyField,
                "Price:", priceField,
        };

        int option = JOptionPane.showConfirmDialog(this, message, "Add Product",
JOptionPane.OK_CANCEL_OPTION);
        if (option == JOptionPane.OK_OPTION)
            {String id = idField.getText();
            String name = nameField.getText();
            int quantity = Integer.parseInt(qtyField.getText());
            double price = Double.parseDouble(priceField.getText());
            products.add(new Product(id, name, quantity, price));
            JOptionPane.showMessageDialog(this, "Product added successfully!");
        }
    }

    private void displayProducts() {
        StringBuilder productList = new StringBuilder("Available Products:\n");
        for (Product product : products) {
            productList.append(product).append("\n");
        }
        JOptionPane.showMessageDialog(this, productList.toString(), "Products",
JOptionPane.INFORMATION_MESSAGE);
    }

    private void updateStockDialog()
        { JTextField idField = new JTextField();
        JTextField qtyField = new JTextField();

        Object[] message =
                { "Product ID:",
                idField,
                "Quantity to Add:", qtyField,
        };

        int option = JOptionPane.showConfirmDialog(this, message, "Update Stock",
JOptionPane.OK_CANCEL_OPTION);
        if (option == JOptionPane.OK_OPTION)
            {String id = idField.getText();
            int quantity = Integer.parseInt(qtyField.getText());
            Product product = findProductById(id);
            if (product != null)
                { product.updateStock(quantity);
                JOptionPane.showMessageDialog(this, "Stock updated successfully!");
            } else {
```

```
            JOptionPane.showMessageDialog(this, "Product not found!", "Error",
JOptionPane.ERROR_MESSAGE);
        }
    }
}

private void createOrderDialog() {
    String orderId = JOptionPane.showInputDialog(this, "Enter Order ID:");
    if (orderId != null && !orderId.isEmpty()) {
        Order order = new Order(orderId);

        while (true) {
            String productId = JOptionPane.showInputDialog(this, "Enter Product ID (or type 'done'
to finish):");
            if (productId == null || productId.equalsIgnoreCase("done"))
                {break;
            }

            Product product = findProductById(productId);
            if (product != null) {
                String quantityStr = JOptionPane.showInputDialog(this, "Enter Quantity:");
                int quantity = Integer.parseInt(quantityStr);
                order.addProductToOrder(product, quantity);
            } else {
                JOptionPane.showMessageDialog(this, "Product not found!", "Error",
JOptionPane.ERROR_MESSAGE);
            }
        }

        if (order.getTotalAmount() > 0)
            {orders.add(order);
            JOptionPane.showMessageDialog(this, "Order created successfully!\n" + order, "Order",
JOptionPane.INFORMATION_MESSAGE);
        } else {
            JOptionPane.showMessageDialog(this, "No products added to the order.", "Error",
JOptionPane.ERROR_MESSAGE);
        }
    }
}

private void displayOrders() {
    StringBuilder orderList = new StringBuilder("Orders:\n");
    for (Order order : orders) {
        orderList.append(order).append("\n");
    }
    JOptionPane.showMessageDialog(this, orderList.toString(), "Orders",
JOptionPane.INFORMATION_MESSAGE);
```

```
    }

    private Product findProductById(String productId)
        {for (Product product : products) {
            if (product.getProductId().equals(productId))
                {return product;
            }
        }
        return null;
    }

    public static void main(String[] args)
        {SwingUtilities.invokeLater(() -> {
            Main gui = new Main();
            gui.setVisible(true);
        });
    }
}
```