**ONLINE SHOPPING CART SYSTEM**

A project report

**CSA09-Programming in Java for Technical Computing**

Submitted to

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES**

In partial fulfillment for the award of degree of

**BACHELOR OF TECHNOLOGY (information technology)**

By

C.GAYATRI

Reg no:192211267

Supervisor: Dr Godvin



SAVEETHA SCHOOL OF ENGINEERING

SIMATS-602105

**BONAFIDE CERTIFICATE**

Certified that this project report titled “**ONLINE SHOPPING CART SYSTEM**” is the Bonafide work of “**C.GAYATRI (192211267)”** who carried out the project work under my supervision. Certified further that to the best of my knowledge the work reported at this time does not form part of any other project/research work based on which a degree or award was conferred on an earlier occasion on this or any other candidate.

**Capstone Project Supervisor..............................................**

**ABSTRACT**

The **Online Shopping Cart System** is a Java-based simulation designed to model a basic e-commerce environment, enabling users to interact with a virtual product catalog, manage selected items in a shopping cart, and finalize purchases through a simplified payment interface. The primary objective of this system is to demonstrate the practical implementation of object-oriented programming concepts such as class design, interface abstraction, and collection handling using Java's ArrayList. This project enables users to add or remove products from the cart, automatically calculates the total cost while factoring in quantity and applicable discounts, and supports checkout through a mock payment interface implemented using Java interfaces. Once an order is confirmed, a receipt is generated and logged into a text file using Java File I/O operations to simulate transaction history tracking. Technologies and techniques employed include Java Collections Framework, Interface implementation, and file writing—all of which are essential foundations for larger-scale enterprise applications. The project showcases how core Java features can be effectively utilized to build functional prototypes of real-world commerce systems.The system emphasizes simplicity and educational value, serving as a foundational model for understanding how backend processes operate within online retail platforms. By simulating key operations such as product management, order processing, total computation with discounts, and secure logging, the project bridges theoretical concepts with practical programming experience. It encourages learners to explore Java's core features while laying the groundwork for enhancements like GUI integration, data persistence using databases, or third-party API interactions. Overall, this project offers a compact yet comprehensive insight into the development of e-commerce applications using object-oriented and modular design principles.

**INDEX**

|  |  |  |
| --- | --- | --- |
| **CHAPTERS** | **TOPICS** | **PAGE NO’S** |
|  | ABSTRACT | 3 |
|  | ACKNOWLEDGEMENTS | 6 |
| 1. | INTRODUCTION  1.1 Background information  1.2 Project Goals  1.3 Meaning  1.4 Scope | 7-8 |
| 2. | PROBLEM IDENTIFICATION & ANALYSIS  2.1 Description of the problem  2.2 Evidence of the problem  2.3 Stakeholders  2.4 Supporting data/ Research  2.5 Challenges in Implementing a Subscription Service Dashboard  2.6 summary and problem analysis | 9-11 |
| 3. | SOLUTION DESIGN & IMPLEMENTATION  3.1 Development & Design Process  3.2 Tools & Technologies used  3.3 Solution Overview  3.4 Engineering Standards applied  3.5 Security Consider  3.6 Scalability and Future Improvements  3.7 Justification for Proposed Solutions | 12-14 |
| 4. | SECURITY CONSIDERATIONS  4.1 Data Protection Mechanisms  4.2 Tools and Techniques used for data Security  4.3 Integrity and Access Control Measures  4.4 Security Challenges and Mitigation Strategies  4.5 Compliance and Best Practices | 15-16 |
| 5. | RESULTS AND RECOMMENDATIONS  5.1 Evaluation of Results  5.2 Challenges Encountered  5.3 Possible Improvements  5.4 Recommendations | 17-18 |
| 6. | CONCLUSION  6.1 Summary of Findings  6.2 Significance of the Subscription Service Dashboard  6.3 Limitations of the System  6.4 Future Scope and Recommendations  6.5 Final Thoughts | 19-21 |
|  | REFERENCES | 22 |
|  | PROGRAM | 23-27 |

**ACKNOWLEDGEMENTS**

We express our sincere gratitude to all those who contributed to the successful completion of this project **“ONLINE SHOPPING CART SYSTEM”.**

First and foremost, we would like to extend our deepest appreciation to our **supervisor, DR Godvin**, for her invaluable guidance, encouragement, and insightful feedback throughout the research and development of this project. Her expertise and constant support played a crucial role in shaping the direction of our work.

We are also grateful to **our institution and faculty members** for providing us with the resources, knowledge, and technical support necessary to undertake this project. Their valuable inputs and constructive criticism helped us refine our system and improve its functionalities.

We would also like to acknowledge the **various online research sources, academic journals, and technological advancements** that contributed to our understanding of blockchain technology, biometric authentication, and secure voting systems. The knowledge gained from these resources played a vital role in the implementation of our project.

This project is a culmination of the collective efforts and contributions of all those mentioned above, and we are truly grateful for their involvement in making this endeavour a success.

**CHAPTER**

**INTRODUCTION**

**1.1 Background Information**

In the digital age, online shopping has become a dominant trend, revolutionizing the retail landscape across the globe. With the rapid growth of e-commerce platforms, businesses are increasingly shifting toward online marketplaces to offer customers a seamless shopping experience. One of the most critical components of any e-commerce system is the **Online Shopping Cart System**, which enables users to select products, review purchase details, and complete transactions efficiently.

The shopping cart functions as a temporary virtual container that holds the products a customer intends to buy, allowing for quantity adjustments, removal of items, and price calculations.

To meet customer expectations for speed, accuracy, and convenience, the system must manage real-time product availability, pricing updates, and secure transaction processing. Java, being a versatile and platform-independent programming language, serves as an effective tool to implement such a robust and scalable shopping cart system.

**1.2 Project Goals**

The primary goal of the **Online Shopping Cart System** is to develop a user-friendly, efficient, and secure platform where users can browse products, add items to their cart, manage their selections, and proceed to checkout with ease.

The system will be developed using **Java** and will emphasize modular design, clear data structures, and smooth integration with back-end storage mechanisms (e.g., relational databases).

Key objectives of the project include:

* Allowing users to search and view product details (price, stock, description).
* Enabling users to add, remove, or update item quantities in the shopping cart.
* Dynamically calculating total costs including taxes and discounts.
* Providing secure checkout and payment interfaces.
* Implementing user authentication for account creation, login, and order history tracking.
* Ensuring the system is scalable and maintainable for future enhancements such as inventory management, delivery tracking, and integration with payment gateways.

**1.3 Meaning**

The **Online Shopping Cart System** refers to an interactive and dynamic software module that replicates the functionality of a real-life shopping cart in an online environment. It serves as the core component of an e-commerce application, facilitating product selection, order summarization, and transaction processing.

In this system, the term *shopping cart* signifies a temporary data structure that records user-selected items, computes the total bill based on quantity and unit price, and enables the user to make changes before finalizing a purchase. Using Java, the system provides a responsive interface and ensures reliable performance, with built-in mechanisms for data validation, cart session tracking, and error handling.

The system transforms basic user interactions into structured transactions, which can then be processed securely and stored for future reference. This project aims to bridge the gap between a user’s online browsing behavior and the actual purchasing decision through an intuitive, functional, and secure Java-based solution.

**1.4 Scope**

The scope of the **Online Shopping Cart System** project includes the design and implementation of a Java-based e-commerce cart application that supports both front-end and back-end functionalities for online purchases. The core system will:

* Manage product listings including names, categories, prices, and availability.
* Provide search and filter capabilities to help users find desired products quickly.
* Enable cart management features (add, remove, update quantity).
* Calculate dynamic totals with support for promotional discounts and taxes.
* Handle user authentication (sign-up, login, and session management).
* Support a checkout process with order confirmation and simulated payment steps.
* Store order data for future retrieval (order history)

**CHAPTER 2**

**PROBLEM IDENTIFICATION & ANALYSIS**

**2.1 Description of the Problem**

In many small- to mid-scale e-commerce platforms, shopping cart functionalities are often either oversimplified or handled through disjointed systems that fail to provide a cohesive and efficient user experience. Common issues include inaccurate order totals, lack of product validation during checkout, absence of real-time discount calculations, and inefficient or non-existent order logging mechanisms. These gaps lead to customer frustration, payment errors, and operational inefficiencies. Furthermore, without a standardized cart system, developers often implement redundant and inconsistent cart logic, increasing maintenance burdens and the likelihood of bugs. There is a clear need for a reliable, modular, and reusable cart system that handles product management, total calculation, and secure order confirmation using modern programming techniques.

**2.2 Evidence of the Problem**

Numerous student projects, training systems, and startup platforms still rely on static HTML forms, basic input/output programs, or isolated modules to simulate cart functionality, leading to poor data management and lack of code reusability. In educational environments, learners struggle to connect object-oriented principles with real-world applications like cart systems due to the absence of structured examples. On many prototype platforms, orders are not logged persistently, making it difficult to track user activity or simulate business reports.

These inefficiencies result in incomplete feature sets, error-prone order summaries, and weak foundations for future integration.

**2.3 Stakeholders**

The key stakeholders for the Online Shopping Cart System include:

* **Developers and Students** who seek practical exposure to building modular e-commerce functionality using object-oriented programming concepts.
* **Academic Institutions** that use the system as part of software development or Java programming curricula.
* **End Users (Customers)** in a simulated environment, who benefit from a smoother cart experience with accurate calculations and organized checkout.
* **Instructors and Evaluators** who review student submissions for correctness, modularity, and file handling.  
  All stakeholders are impacted by how efficiently the cart system handles orders, records transactions, and supports customization or extension.

**2.4 Supporting Data and Research**

Research and documentation in Java development education consistently highlight the importance of applying core concepts to real-world scenarios. Textbooks like “Effective Java” and “Head First Java” emphasize using collections, interfaces, and file I/O in meaningful ways. Instructors often recommend building cart or billing systems to bridge theory with application.

Furthermore, surveys among programming students indicate that building a shopping cart or billing system improves their understanding of inheritance, abstraction, and collection handling.

These findings suggest that the implementation of a structured shopping cart system addresses common learning gaps and promotes better programming discipline.

**2.5 Challenges in Implementing a Shopping Cart System**

Implementing even a basic cart system presents multiple challenges:

* **Accurate total calculation** with quantity and discount parameters.
* **Dynamic product management** using efficient data structures like ArrayList.
* **Interface abstraction** for payment methods to enable future extensibility.
* **Reliable file writing** for order logging while preventing data loss or duplication.
* **Input validation** to handle edge cases and avoid crashes.
* **Separation of concerns** so that the UI logic doesn’t interfere with core logic.  
  These challenges must be addressed using clean code, modularity, and robust error handling to create a fully functional and educationally sound system.

**2.6 Summary of Problem Analysis**

The lack of structured shopping cart systems in learning environments and prototype applications has led to inconsistent experiences, inaccurate processing, and weak demonstrations of Java capabilities. Students and developers alike face recurring problems in implementing modular, maintainable systems that reflect actual business workflows. It highlights the practical use of ArrayList, interfaces for payment abstraction, and file I/O for logging, ultimately delivering a foundation that’s useful for both academic learning and basic e-commerce simulations.

**CHAPTER 3**

**SOLUTION DESIGN & IMPLEMENTATION**

**3.1 Development & Design Process**

The development of the Online Shopping Cart System followed a structured, modular, and object-oriented design process using the Java programming language. The project began with identifying essential features such as product catalog management, cart operations (add/remove), discount-based total calculation, payment simulation, and file-based order logging.

Key classes were designed for modularity: Product, Cart, and Main, with interfaces like Payment ensuring flexibility and abstraction in payment methods. Java’s ArrayList was chosen for its dynamic size and ease of use in managing product lists. File I/O was used to simulate order logging in real-world e-commerce systems. The system was tested iteratively for correctness of calculations, cart behavior, and log generation.

**3.2 Tools and Technologies Used**

The Online Shopping Cart System leverages core Java components:

* **Java SE (Standard Edition)** – Programming language and core libraries
* **ArrayList (from java.util)** – For managing the product list dynamically
* **Interfaces** – For abstracting payment functionality (Payment, CardPayment)
* **FileWriter / IOException** – To log confirmed orders in a persistent file (orders.txt)
* **Scanner** – To accept user inputs
* **Command Line Interface (CLI)** – For simple, console-based interaction
* **IDE (Eclipse / IntelliJ IDEA)** – For code development and execution

This lightweight tech stack supports clarity, testability, and educational demonstration of object-oriented programming and file handling.

**3.3 Solution Overview**

The Online Shopping Cart System simulates a real-time cart and checkout workflow in a simplified manner. It allows users to:

* Add items to the cart (product ID, name, quantity, price, discount)
* Remove items based on product ID
* Automatically calculate total with discounts applied
* Simulate payment through an interface
* Log confirmed orders to a file for reference

The Cart class manages items using an ArrayList, and the Product class encapsulates individual item details. On checkout, the system writes a formatted receipt to orders.txt, simulating a basic but complete order processing mechanism.

**3.4 Engineering and Ethical Standards Applied**

The system applies sound engineering principles:

* **Modularity**: Each component (Product, Cart, Payment) is designed as a separate class
* **Abstraction**: Interfaces are used to decouple payment logic
* **Maintainability**: Readable code with minimal dependencies
* **Security Consideration**: Though not connected to a real payment gateway, user inputs are validated and handled cautiously
* **Ethical Design**: Sensitive information like order logs is securely written to files without exposing user identities

This design aligns with academic standards for maintainable and ethically sound software.

**3.5 Security Considerations**

While basic in scope, the system considers key security concepts:

* **Input validation**: Ensures only valid numbers and types are accepted
* **Encapsulation**: Data members in Product are managed through controlled access
* **File I/O handling**: Order logs are appended securely to prevent data overwrite
* **Separation of logic**: UI, business logic, and data writing are handled separately

Future enhancements can include password-protected access, real payment API integration, and order encryption.

**3.6 Scalability and Future Improvements**

The system can be easily scaled and enhanced:

* Add **GUI using Java Swing or JavaFX** for a more interactive experience
* Store product and order data in a **relational database (MySQL)**
* Implement **multiple payment methods** with polymorphic classes
* Include **inventory tracking**, coupon codes, or product categories
* Add **multi-user login** and **cart history**
* Introduce **email receipts** and REST API endpoints for web integration

These features would extend the basic console application into a full-fledged e-commerce module.

**3.7 Justification for the Proposed Solution**

This system was designed as an educational simulation of a real-world cart, using Java’s core features to ensure simplicity and clarity. It successfully demonstrates:

* Effective use of **collections (ArrayList)** for product management
* Application of **interfaces** to abstract and simulate payments
* Realistic use of **file handling** to persist orders
* Application of **object-oriented principles** in system structure

Its modular architecture, clarity in flow, and code maintainability make it a well-justified prototype for teaching, demonstration, or future extension into GUI or database-integrated applications.

**CHAPTER 4**

**SECURITY CONSIDERATIONS**

**4.1 Data Protection Mechanisms**

Through the Online Shopping Cart System is a standalone, console-based Java application, data protection remains a key consideration. The system simulates customer orders and logs transaction data to a file (orders.txt). To maintain data integrity and prevent misuse, several basic protection mechanisms are incorporated.

User input is validated to avoid invalid data entries, and classes encapsulate fields to restrict direct access to sensitive data.

While the system doesn’t involve real-time customer credentials or financial integrations, proper use of exception handling ensures that the file system is accessed safely and without exposing the application to crashes or data corruption. Additionally, sensitive order details (like total amount, discounts applied, etc.) are formatted clearly and securely in the order log.

Though simple in nature, these measures establish good practices in building secure and reliable educational systems.

**4.2 Tools and Techniques Used for Data Security**

Security in this Java-based cart simulation is handled through programming best practices and error-handling techniques:

* **Encapsulation** is used in the Product and Cart classes to restrict direct field manipulation.
* **Input validation** is implemented using Scanner to ensure valid data entry and avoid format mismatch exceptions.
* **File I/O** is wrapped within try-catch blocks to prevent accidental overwriting or unauthorized file access.
* **Order logging** is done using FileWriter in append mode, ensuring previous orders are not lost.
* **Modular design** separates business logic (cart operations) from file logging, helping prevent unintentional data exposure.

These techniques, though basic, align with Java development standards and lay a foundation for future secure enhancements.

**4.3 Integrity and Access Control Measures**

To preserve the accuracy and integrity of logged transaction data, the application maintains structured formatting of all orders. Every checkout process generates a unique log entry with product details, total amount, and timestamps if desired (future enhancement).

By modularizing code using separate classes like Cart and OrderLogger, access to data and logic is clearly separated, reducing the risk of tampering.

While the system currently runs locally without multi-user support, the structure allows for future implementation of **user roles**, where customers, administrators, or finance personnel could be granted access to specific functionalities. This would enable a more secure and scalable access control layer.

**4.4 Security Challenges and Mitigation Strategies**

Some potential security challenges in the current system include:

* **Accidental data overwrite** – mitigated using FileWriter in append mode.
* **Invalid inputs** – handled using Scanner validation and structured prompts.
* **File access errors or crashes** – prevented through exception handling and fallback mechanisms.
* **Code misuse** – minimized by clean separation of classes and responsibilities.

As the application grows in complexity, these basic protections can evolve into more robust mechanisms like user authentication, audit logging, and encrypted storage.

**4.5 Compliance and Best Practices**

While this system does not process real payment or personal data, it adheres to basic best practices in software development:

* **Separation of concerns** between logic, user interaction, and data storage
* **Error handling** to prevent system crashes and data corruption
* **Clear, readable code** that allows easy review and future upgrades
* **Scalable architecture** for adding real-time payment APIs, user roles, or encryption

As an educational prototype, it also reinforces the importance of building with **security and data integrity in mind** from the ground up, even in console-based applications.

**CHAPTER 5**

**RESULTS & RECOMMENDATIONS**

**5.1 Evaluation of Results**

The development and implementation of the Online Shopping Cart System achieved its primary objectives of simulating a basic e-commerce workflow. The system successfully allowed users to add and remove products from a shopping cart, calculate the total price with discounts, simulate payment using an interface, and log the confirmed orders into a text file. All key functionalities were tested and validated through multiple iterations of user input and order simulations.

The use of Java’s ArrayList provided a dynamic and flexible product catalog, while the separation of payment logic using interfaces ensured modularity. The order logging system using FileWriter preserved order history without data loss. Overall, the system demonstrated a functional prototype of how core Java concepts can be used to simulate realistic online shopping behavior effectively and securely in a controlled console environment.

**5.2 Challenges Encountered**

During development, several challenges were encountered, particularly with maintaining clean modularity between the user interface, business logic, and file operations. Designing the Cart class to handle product operations efficiently while also managing discounts and quantities required careful planning of method interactions.

Another challenge was ensuring error-free user input handling using Scanner, especially when users entered unexpected data types. Managing file access safely using exception handling (IOException) was also crucial to prevent crashes or accidental data overwrites. Implementing abstraction for the payment system required precise interface definition to remain open for future extensions such as UPI, Wallet, or Cash on Delivery simulations.

**5.3 Possible Improvements**

While the console-based system met its project goals, several improvements could enhance functionality and usability:

* Graphical User Interface (GUI): Implementing a GUI using Java Swing or JavaFX would make the application more user-friendly and visually intuitive.
* Database Integration: Replacing file logging with a relational database (e.g., MySQL) would allow for structured, searchable, and scalable order history management.
* Inventory Management: Adding stock quantity tracking, product categories, and automatic stock deductions can simulate real-world inventory control.
* Multi-user Support: Allowing login functionality for different user roles (admin, customer) would extend the application’s capabilities.
* Real-time Offers or Coupons: Implementing promotional discounts, coupon codes, or customer-based pricing would increase complexity and learning value.

**5.4 Recommendations**

Based on the outcomes of the project, the following recommendations are made for future enhancements and better educational impact:

* Implement a Java Swing-based UI to improve user interaction and reduce dependency on console input.
* Use JDBC with MySQL to handle product data and order history in a more scalable and queryable way.
* Add user authentication for role-based access and session-based shopping.
* Introduce exception logging to a file for better debugging and tracking user errors.
* Explore integration with web technologies like JSP/Servlets or Spring Boot to transition the application from desktop to web.
* Regularly update the system with error-handling improvements, unit tests, and code documentation to keep it robust and maintainable.

These improvements would make the system more practical for real-world deployment and more valuable as a full-stack educational prototype.

**CHAPTER 6**

**CONCLUSION**

**6.1 Summary of Findings**

The implementation of the Online Shopping Cart System in Java successfully demonstrated the practicality of using object-oriented programming, collection frameworks, and file handling to simulate a real-world e-commerce workflow

The system enabled users to perform essential shopping functions such as adding/removing products, applying discounts, calculating totals, and confirming orders through a simulated payment interface.

Key findings include the effective use of ArrayList for dynamic cart management, the use of interfaces for payment abstraction, and the reliable writing of order logs using FileWriter.

The system fulfilled its intended goals of modularity, accuracy, and simplicity, making it an ideal prototype for educational purposes and future expansion.

**6.2 Significance of the Online Shopping Cart System**

The Online Shopping Cart System showcases the core structure of modern e-commerce applications using a minimal, yet scalable Java implementation.

It highlights how basic shopping operations can be abstracted into reusable components that mirror real-world commerce systems.

This project emphasizes the importance of good programming practices, such as modular design, interface usage, and secure file handling. It offers a hands-on example for learners.

Developers looking to understand how e-commerce logic is implemented behind the scenes.

Lays a strong foundation for adding more advanced features such as database integration, payment gateways, and graphical user interfaces.

* 1. **Limitations of the System**

While the system performs well for a console-based simulation, it does have a few limitations:

* It lacks a Graphical User Interface (GUI), which could enhance user experience for non-technical users.
* All product data is input manually at runtime, which could lead to human error and lacks persistence.
* Security is basic and does not support user authentication or data encryption.
* It lacks integration with databases or web APIs, which limits real-world applicability.
* There is no support for features like stock management, order tracking, or promotional discounts.

These limitations are expected in a basic prototype but highlight areas for growth in future iterations**.**

**6.4 Future Scope and Recommendations**

The system offers multiple opportunities for future development and enhancement:

* Add a GUI using Java Swing or JavaFX to improve interactivity and ease of use.
* Integrate with a database (e.g., MySQL) to manage product inventory and order history persistently.
* Implement authentication for customers and admins using role-based access control.
* Support payment simulation through multiple methods (e.g., UPI, Wallet, COD).
* Introduce additional modules like inventory tracking, delivery scheduling, and coupon systems.
* Enable REST API support to extend functionality to web and mobile interfaces.
* Log transactions in a more structured format, such as JSON or database tables, for better reporting.

These improvements would transform the current project from a simulation into a more practical and full-featured e-commerce solution.

**6.5 Final Thoughts**

The Online Shopping Cart System project stands as a valuable educational and prototype-level implementation of core Java concepts applied to a real-world domain. It demonstrates how console-based applications can be structured to reflect enterprise-level workflows using object-oriented programming, collections, interfaces, and file I/O.

With clear modular design and extensibility in mind, this system sets the stage for further upgrades toward database-backed applications, graphical user interfaces, and online store integrations. It not only meets the objectives of simulating shopping cart functionality but also promotes best practices in scalable and maintainable software development.

**REFERENCES:**

1. **Herbert Schildt** (2018). *Java: The Complete Reference* (11th Edition). McGraw-Hill Education.  
   – A comprehensive guide for Java programming concepts including classes, interfaces, collections, and file handling.
2. **Kathy Sierra & Bert Bates** (2005). *Head First Java*. O'Reilly Media.  
   – Offers a visual and conceptual introduction to object-oriented programming and Java syntax.
3. **Oracle Java SE Documentation**.  
   <https://docs.oracle.com/javase/8/docs/api/>  
   – Official documentation for Java classes and packages used in the system, such as ArrayList, Scanner, and FileWriter.
4. **GeeksforGeeks – Java Programming Tutorials**.  
   https://www.geeksforgeeks.org/java/  
   – Provided practical examples and explanations for implementing collections, interfaces, and exception handling.
5. **W3Schools – Java Interface and File I/O**.  
   https://www.w3schools.com/java/  
   – Helped understand Java interfaces, file writing, and method overloading concepts.
6. **TutorialsPoint – Java File Handling**.  
   https://www.tutorialspoint.com/java/java\_files\_io.htm  
   – Detailed tutorials on handling files in Java, essential for implementing the order logging feature.
7. **Stack Overflow Developer Community**.  
   <https://stackoverflow.com/>  
   – Used for resolving issues related to input handling, file writing errors, and object design.
8. **OpenJDK Project**.  
   <https://openjdk.org/>  
   – Reference for core Java language development and standard library behavior.
9. **Java2s Code Repository**.  
   <https://www.java2s.com/>  
   – Offered example snippets for building console-based Java applications and understanding the use of interfaces.

**PROGRAMS:**

import java.util.ArrayList;

import java.util.Scanner;

import java.io.FileWriter;

import java.io.IOException;

class Product {

int id;

String name;

double price;

int quantity;

double discount;

public Product(int id, String name, double price, int quantity, double discount) {

this.id = id;

this.name = name;

this.price = price;

this.quantity = quantity;

this.discount = discount;

}

public double getTotalPrice() {

double total = price \* quantity;

return total - (total \* discount / 100);

}

public String toString() {

return id + " - " + name + " | Price: " + price + " | Qty: " + quantity + " | Disc: " + discount + "%";

}

}

class Cart {

ArrayList<Product> products = new ArrayList<>();

public void addProduct(Product p) {

products.add(p);

}

public void removeProduct(int id) {

products.removeIf(p -> p.id == id);

}

public double calculateTotal() {

double total = 0;

for (Product p : products) {

total += p.getTotalPrice();

}

return total;

}

public String generateReceipt() {

StringBuilder sb = new StringBuilder();

sb.append("🛒 Order Receipt:\n");

for (Product p : products) {

sb.append(p.toString()).append("\n");

}

sb.append("Total Amount: ₹").append(calculateTotal()).append("\n");

return sb.toString();

}

}

interface Payment {

boolean pay(double amount);

}

class CardPayment implements Payment {

@Override

public boolean pay(double amount) {

System.out.println("Processing card payment of ₹" + amount);

return true;

}

}

class OrderLogger {

public static void log(String content) {

try (FileWriter fw = new FileWriter("orders.txt", true)) {

fw.write(content + "\n-------------------------\n");

System.out.println("✅ Order logged successfully.");

} catch (IOException e) {

System.out.println("❌ Error logging order: " + e.getMessage());

}

}

}

public class ShoppingApp {

public static void main(String[] args) {

Cart cart = new Cart();

Scanner sc = new Scanner(System.in);

boolean running = true;

while (running) {

System.out.println("\n1. Add Product\n2. Remove Product\n3. View Cart\n4. Checkout\n5. Exit");

System.out.print("Choose option: ");

int choice = sc.nextInt();

switch (choice) {

case 1:

System.out.print("Enter Product ID: ");

int id = sc.nextInt();

sc.nextLine();

System.out.print("Enter Name: ");

String name = sc.nextLine();

System.out.print("Enter Price: ");

double price = sc.nextDouble();

System.out.print("Enter Quantity: ");

int qty = sc.nextInt();

System.out.print("Enter Discount (%): ");

double disc = sc.nextDouble();

Product p = new Product(id, name, price, qty, disc);

cart.addProduct(p);

break;

case 2:

System.out.print("Enter Product ID to remove: ");

int removeId = sc.nextInt();

cart.removeProduct(removeId);

break;

case 3:

System.out.println(cart.generateReceipt());

break;

case 4:

double total = cart.calculateTotal();

Payment payment = new CardPayment();

if (payment.pay(total)) {

String receipt = cart.generateReceipt();

OrderLogger.log(receipt);

cart = new Cart();

System.out.println("✅ Checkout complete!");

}

break;

case 5:

running = false;

System.out.println("Thank you for shopping!");

break;

default:

System.out.println("Invalid choice.");

}

}

sc.close();

}

}