A Project
SmartBill

CSE-342 : Advance Programming Lab

Submitted By:

Name	ID	Intake	Section
MD.Jahidul Islam Shihab	20234103347	52	09
MD. Al Nasir Uddin Siam	20234103349	52	09
Munim Halder	20234103351	52	09
Yeasir Ibna Hasibur Rahman	20234103358	52	09

Supervised By :

Humayra Ferdous

Lecturer

Department of **CSE**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BANGLADESH UNIVERSITY OF BUSINESS AND TECHNOLOGY

(BUBT)

ABSTRACT

We present “SmartBill”, an advanced billing platform designed to streamline retail transactions through a unified console-based interface. The system features product selection, inventory management, dynamic discount application for Regular and Premium customers, and automated receipt generation. Administrators can maintain and update inventory, while users benefit from accurate calculations, professional receipts, and real-time stock updates. With file-based data storage and robust exception handling, SmartBill ensures reliable operation, efficient billing, and effective inventory tracking. Overall, the platform simplifies retail management, enhances transaction accuracy, and supports informed decision-making for small to medium retail businesses.

DECLARATION

I hereby declare that the project entitled "SmartBill" submitted for the degree of Bachelor of Science in Computer Science and Engineering in the faculty of Computer Science and Engineering of Bangladesh University of Business and Technology (BUBT), is our original work. I affirm that all the content presented in this project is created by us and does not contain any materials previously published or written by any other person. Furthermore, we assert that this project does not include any content that has been accepted for the award of the degree or diploma to any other candidate. Any external sources or references utilized in this project have been duly cited and acknowledged.

---------------------------------------	---------------------------------------
Md. Jahidul Islam Shihab	MD. Al Nasir Uddin Siam
ID: 20234103347	ID: 20234103349
Intake-52	Intake-52
Section-9	Section-9

---------------------------------------	---------------------------------------
Munim Halder	Yeasir Ibna Hasibur Rahman
ID: 20234103351	ID: 20234103358
Intake-52	Intake-52
Section-9	Section-9

CERTIFICATION

This project report titled “SmartBill” submitted by MD.Jahidul Islam Shihab ,Md. Al Nasir Uddin Siam ,Munim Halder and Yeasir Ibna Hasibur Rahman students of the Department of Computer Science and Engineering, Bangladesh University of Business and Technology (BUBT), under the supervision of Humayra Ferdous, Lecturer, Department of Computer Science and Engineering, has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering.

------------------------------------	------------------------------------
Humayra Ferdous	Md. Saifur Rahman
Lecturer	Assistant Professor
Department of CSE	&
	Chairman
	Department of CSE

DEDICATION

This project is dedicated to our parents for their unconditional love, support, and inspiration throughout our journey. We also dedicate this work to our well-wishers and mentors who guided, encouraged, and supported us whenever needed. Their constant motivation has been invaluable in completing this project successfully.

ACKNOWLEDGEMENTS

We would like to express our sincere gratitude to Humayra Ferdous, Lecturer and our project supervisor, for her guidance, expertise, and encouragement, which were instrumental in the successful completion of this project.

We also thank all the lecturers and instructors of the Department of Computer Science and Engineering, Bangladesh University of Business and Technology (BUBT), for their valuable guidance and support in teaching the fundamentals of coding and software development.

Finally, We grateful to our family and friends for their continuous encouragement throughout this journey.

APPROVAL

This project report titled “SmartBill” submitted by MD.Jahidul Islam Shihab (ID:20234103347) ,Md. Al Nasir Uddin Siam (ID:20234103349) ,Munim Halder (ID:20234103351) and Yeasir Ibna Hasibur Rahman (ID:20234103358) students of the Department of Computer Science and Engineering (CSE), Bangladesh University of Business and Technology (BUBT), under the supervision of Humayra Ferdous, Lecturer, Department of CSE, has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science (B.Sc.) in Computer Science and Engineering. This report has been approved as to its style and contents.

Supervisor:
Humayra Ferdous
Lecturer
Department of Computer Science and Engineering (CSE)
Bangladesh University of Business and Technology (BUBT)
Mirpur-2, Dhaka-1216, Bangladesh

Chairman:
Md. Saifur Rahman
Assistant Professor and Chairman
Department of Computer Science and Engineering (CSE)
Bangladesh University of Business and Technology (BUBT)
Mirpur-2, Dhaka-1216, Bangladesh

Abbreviations and Nomenclature

SMS – Super Market System

OOP – Object-Oriented Programming

API – Application Programming Interface

I/O – Input / Output

IDE – Integrated Development Environment

Qty – Quantity

Tk – Taka

INV – Inventory

BR – BufferedReader

PW – PrintWriter

LHM – LinkedHashMap

SC – Scanner

POLY – Polymorphism (Overriding & Overloading)

EXH – Exception Handling

TABLE OF CONTENTS

List of Figures

List of Snippets

Chapter 1: Introduction

1.1 Introduction

SmartBill is a **console-based receipt and billing system** designed to automate the billing process in retail shops and small supermarkets. Billing is an essential function of any retail business, and errors in manual billing can lead to financial loss and customer dissatisfaction. SmartBill addresses this problem by automating calculations, applying discounts, generating itemized receipts, and maintaining inventory records.

The system is implemented in **Java**, leveraging **Object-Oriented Programming (OOP)** principles such as **inheritance, polymorphism, abstraction, and encapsulation**, making it modular, maintainable, and easy to extend.

1.2 Objective

The main objectives of SmartBill are:

1. **Automate billing**: Reduce human errors by automating calculations of totals and discounts.
2. **Inventory management**: Update stock automatically as products are sold.
3. **Customer management**: Differentiate discounts for Regular and Premium customers.
4. **Professional receipts**: Generate clear, structured receipts for customers.
5. **Demonstrate OOP concepts**: Use Java classes, interfaces, and exception handling in a practical scenario.

1.3 Project Scope

SmartBill is suitable for:

1. **Small to medium retail shops** that need automated billing.
2. Shops that want **inventory tracking** integrated with billing.
3. Systems requiring **discount management** based on customer type.
4. Situations where **accurate receipts** and stock updates are critical.

5. The system **cannot handle multi-user network environments** yet, and it is primarily **console-based**. Future expansions can include GUI or web-based interfaces.

1.4 Our Contributions

1. Developed **discountable customer classes** to handle different discount policies.
2. Created a **Product class** that handles stock and cost calculations.
3. Implemented **file-based inventory management** (inventory.txt).
4. Added **exception handling** for invalid input, stock shortage, and file operations.
5. Designed a **professional receipt format** with subtotal, discount, and grand total.

1.5 Conclusions

SmartBill demonstrates how **OOP principles** can solve practical problems in retail billing. It reduces manual errors, provides a clear receipt format, and maintains an updated inventory for every transaction.

Chapter 2: Existing Literature

2.1 Introduction

Billing and receipt systems have evolved significantly from **manual methods** to **automated digital solutions**. Effective billing software improves efficiency, reduces human error, and supports better customer service.

2.2 History of Billing / Receipt Systems

1. **Manual Billing**: Originally, shops used paper-based ledgers to record purchases.
2. **Electronic Cash Registers (ECRs)**: Automated total calculations and basic receipts.
3. **Software-Based Systems**: Modern systems integrate **inventory, customer management, and reporting**.
4. SmartBill fits into this trend by providing **automated billing and inventory management** using a console-based Java application.

2.3 Growth of Retail Billing Software

1. **Integrated systems**: Modern solutions combine billing with inventory and sales reporting.
2. **Discount and loyalty management**: Premium customers or loyalty programs receive automated discounts.
3. **Digital receipts**: Replacing paper receipts with printed or PDF receipts.

2.4 Conclusions

SmartBill adopts essential features of modern billing systems, focusing on **accuracy, discounts, inventory tracking, and receipt formatting**.

Chapter 3: Proposed Model

3.1 Introduction

The Proposed Model for SmartBill focuses on developing an efficient, user-friendly, and modular billing system for retail environments. The main goal of this system is to simplify the billing process, accurately calculate customer totals, manage inventory in real-time, and generate professional receipts. This chapter introduces the architecture and design philosophy of SmartBill, explaining how its components interact to achieve a reliable and maintainable system.

3.1.1 Purpose of the Proposed Model

1. **Automation of Billing Process:** Eliminates manual errors and reduces the time required to calculate bills for multiple products.
2. **Customer-Centric Approach:** Supports different customer types (Regular and Premium) with automated discount application.
3. **Inventory Management:** Tracks product quantities, updates stock after each purchase, and ensures accurate stock reporting.
4. **Ease of Use:** Simple console-based interface for entering customer details, selecting products, and generating receipts.

3.1.2 Key Features

1. **Object-Oriented Design**

- Uses abstraction, inheritance, and polymorphism to represent customers and products.
- Modular classes allow easy maintenance and future upgrades.

2. **Discount Management**

- Applies discounts dynamically based on customer type and total purchase amount.
- Demonstrates runtime polymorphism through the Customer class hierarchy.

3. **Inventory Handling**

- Maintains product details such as name, quantity, price, and unit.
- Supports file-based storage, enabling persistence across sessions.

4. **Receipt Generation**

- Produces a clear, formatted receipt showing itemized products, quantities, prices, subtotal, discounts, and grand total.

5. **Exception Handling**

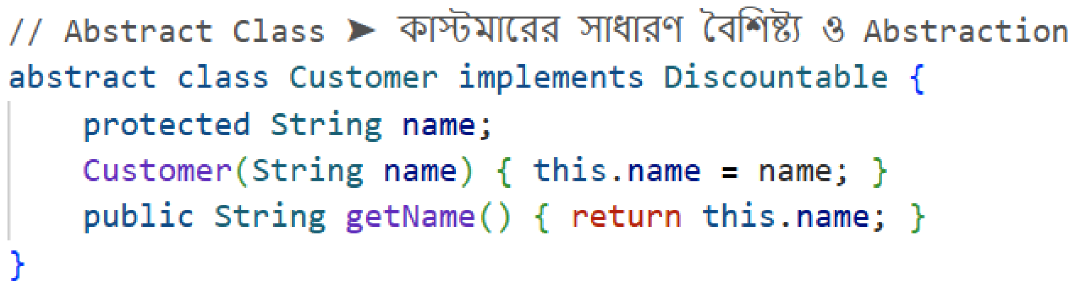
- Handles invalid inputs gracefully, such as negative quantities, exceeding stock, or invalid product names.

3.2 Software Requirements

- **Programming Language**: Java
- **File Storage**: inventory.txt for product inventory
- **IDE**: VS Code
- **OS**: Windows, Linux, or Mac
- **Console**: Command-line interface for interaction

3.2.1 System Structure (SS) of Implemented Components

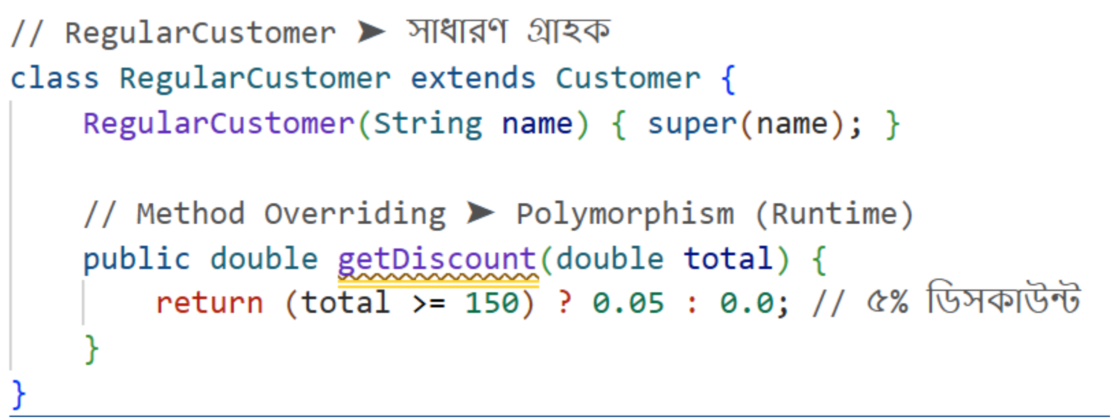
- **Customer (Abstract Class)**:



```java
// Abstract Class ➤ কাস্টমারের সাধারণ বৈশিষ্ট্য ও Abstraction
abstract class Customer implements Discountable {
    protected String name;
    Customer(String name) { this.name = name; }
    public String getName() { return this.name; }
}
```

Snippet-1: Base class for all customer types.

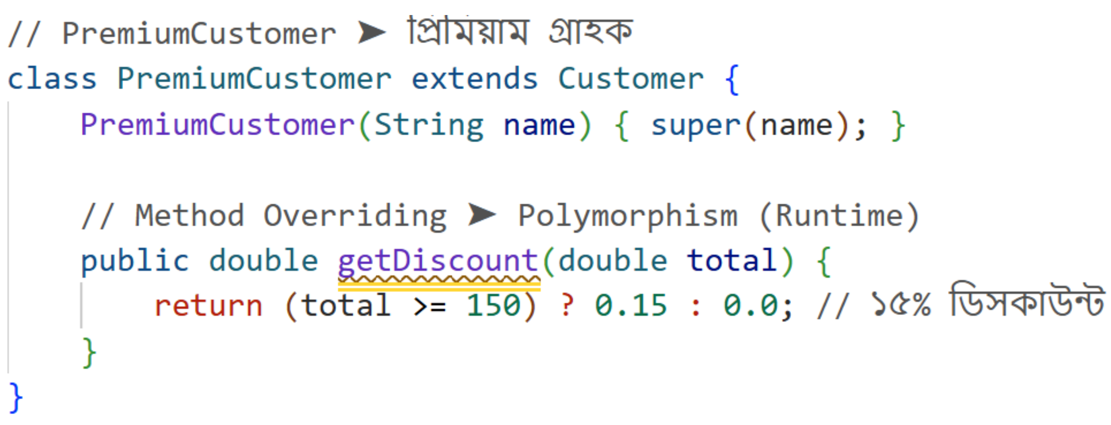
- **RegularCustomer**:



```
// RegularCustomer ➤ সাধারণ গ্রাহক
class RegularCustomer extends Customer {
    RegularCustomer(String name) { super(name); }

    // Method Overriding ➤ Polymorphism (Runtime)
    public double getDiscount(double total) {
        return (total >= 150) ? 0.05 : 0.0; // ৫% ডিসকাউন্ট
    }
}
```

Snippet-2: Calculate discounts for Regular Customer using polymorphism.

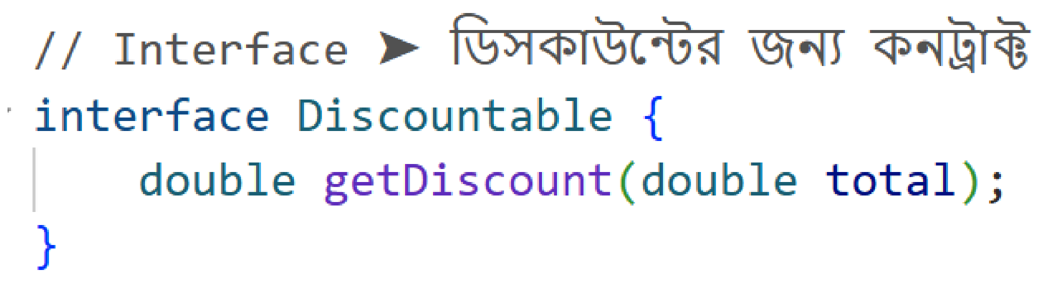
- **PremiumCustomer:**



```
// PremiumCustomer ➤ প্রিমিয়াম গ্রাহক
class PremiumCustomer extends Customer {
    PremiumCustomer(String name) { super(name); }

    // Method Overriding ➤ Polymorphism (Runtime)
    public double getDiscount(double total) {
        return (total >= 150) ? 0.15 : 0.0; // ১৫% ডিসকাউন্ট
    }
}
```

Snippet-3: Calculate discounts Premium Customer using polymorphism.

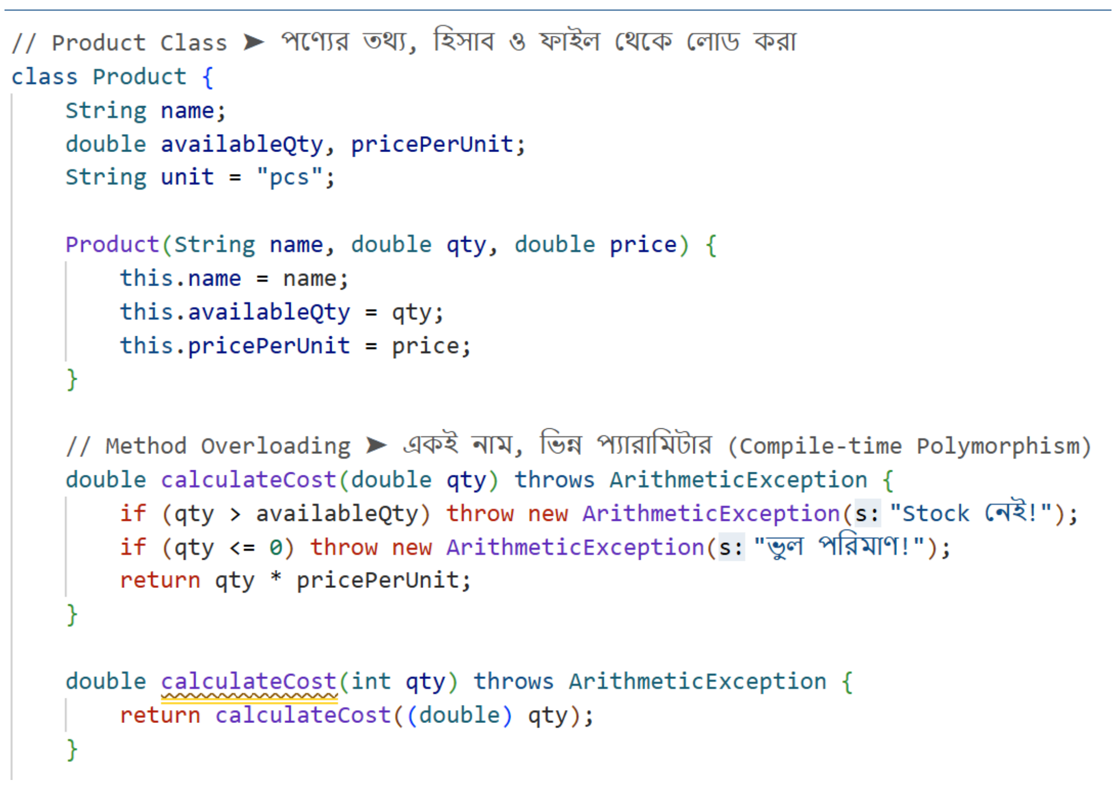
- **Discountable Interface**:



```
// Interface ➤ ডিসকাউন্টের জন্য কনট্রাক্ট
interface Discountable {
    double getDiscount(double total);
}
```

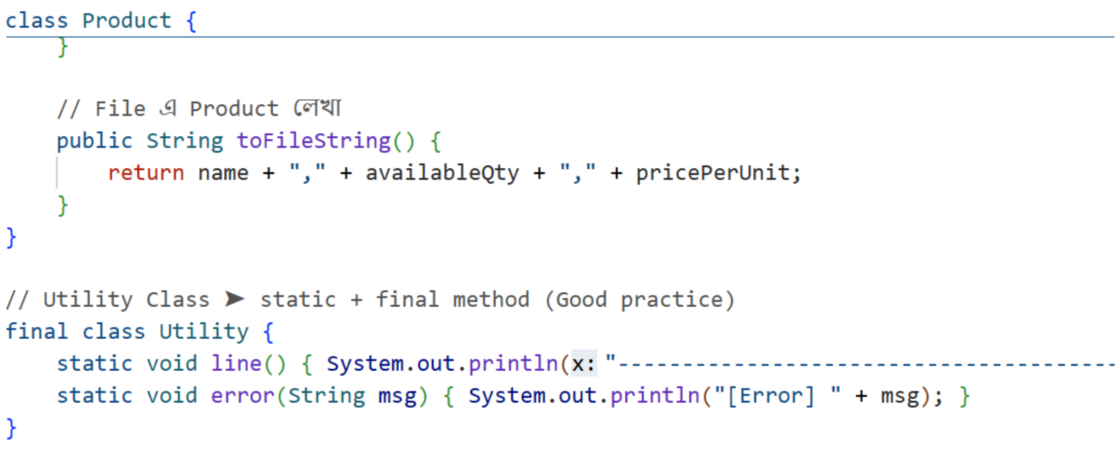
Snippet-4: Defines getDiscount() contract.

- **Product Class**: Manages product details, price, stock, and cost calculations.



```java
// Product Class ➤ পণ্যের তথ্য, হিসাব ও ফাইল থেকে লোড করা
class Product {
    String name;
    double availableQty, pricePerUnit;
    String unit = "pcs";

    Product(String name, double qty, double price) {
        this.name = name;
        this.availableQty = qty;
        this.pricePerUnit = price;
    }

    // Method Overloading ➤ একই নাম, ভিন্ন প্যারামিটার (Compile-time Polymorphism)
    double calculateCost(double qty) throws ArithmeticException {
        if (qty > availableQty) throw new ArithmeticException(s: "Stock নেই!");
        if (qty <= 0) throw new ArithmeticException(s: "ভুল পরিমাণ!");
        return qty * pricePerUnit;
    }

    double calculateCost(int qty) throws ArithmeticException {
        return calculateCost((double) qty);
    }
```

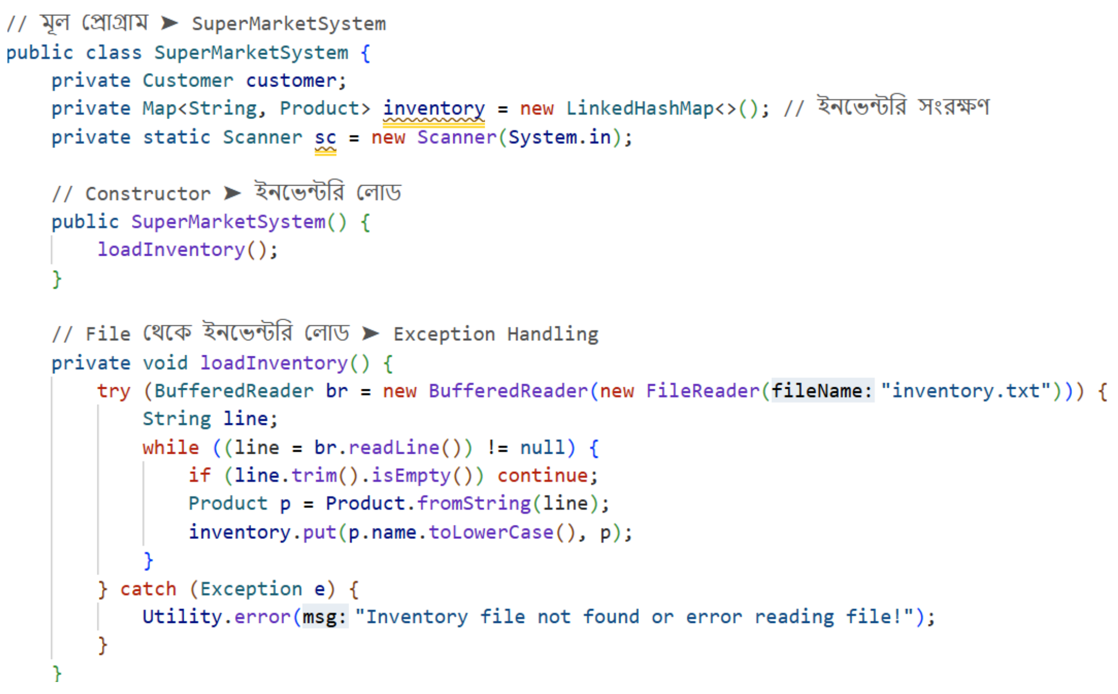
Snippet-5:



```java
class Product {
    }

    // File এ Product লেখা
    public String toFileString() {
        return name + "," + availableQty + "," + pricePerUnit;
    }
}

// Utility Class ➤ static + final method (Good practice)
final class Utility {
    static void line() { System.out.println(x: "--------------------------------------
    static void error(String msg) { System.out.println("[Error] " + msg); }
}
```

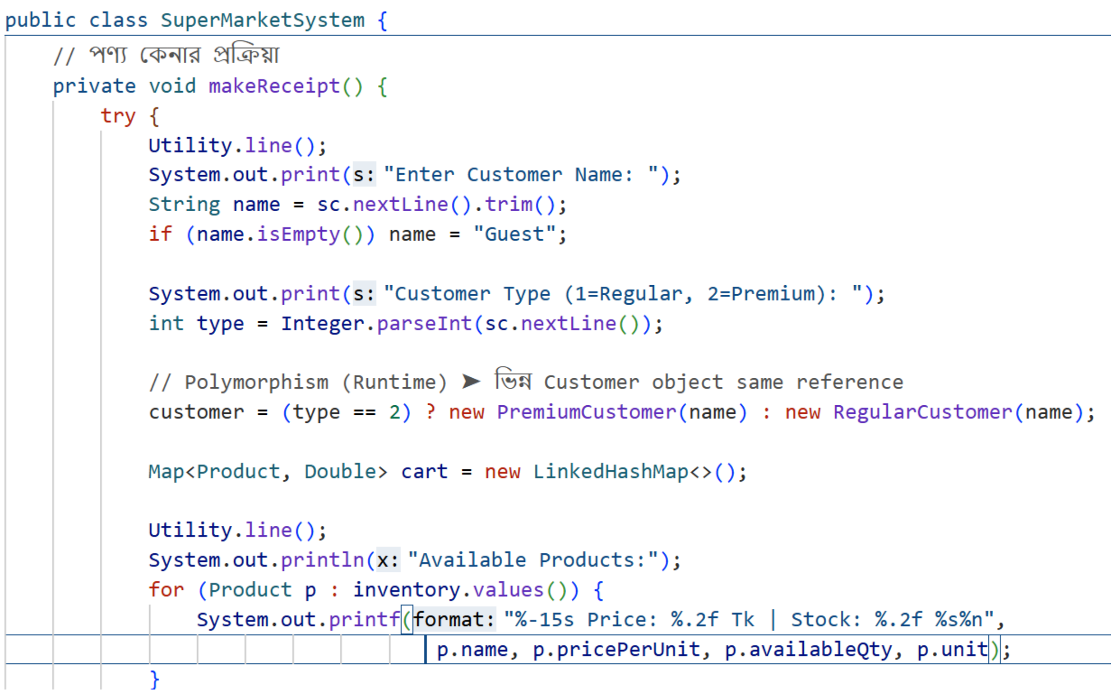
Snippet-6:

- **SuperMarketSystem Class**: Main driver handling input, cart, receipt, and inventory updates.



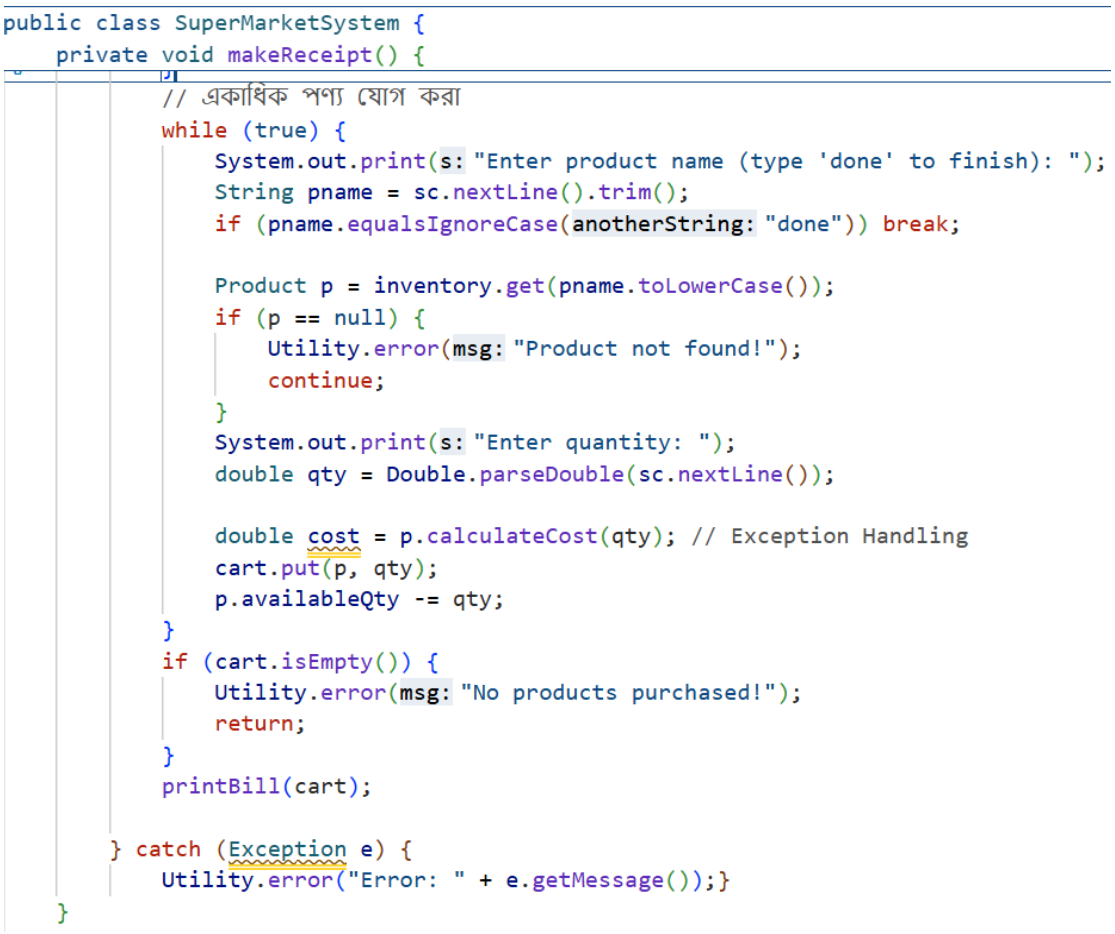
```java
// মূল প্রোগ্রাম ➤ SuperMarketSystem
public class SuperMarketSystem {
    private Customer customer;
    private Map<String, Product> inventory = new LinkedHashMap<>(); // ইনভেন্টরি সংরক্ষণ
    private static Scanner sc = new Scanner(System.in);

    // Constructor ➤ ইনভেন্টরি লোড
    public SuperMarketSystem() {
        loadInventory();
    }

    // File থেকে ইনভেন্টরি লোড ➤ Exception Handling
    private void loadInventory() {
        try (BufferedReader br = new BufferedReader(new FileReader(fileName: "inventory.txt"))) {
            String line;
            while ((line = br.readLine()) != null) {
                if (line.trim().isEmpty()) continue;
                Product p = Product.fromString(line);
                inventory.put(p.name.toLowerCase(), p);
            }
        } catch (Exception e) {
            Utility.error(msg: "Inventory file not found or error reading file!");
        }
    }
```

Snippet-7:



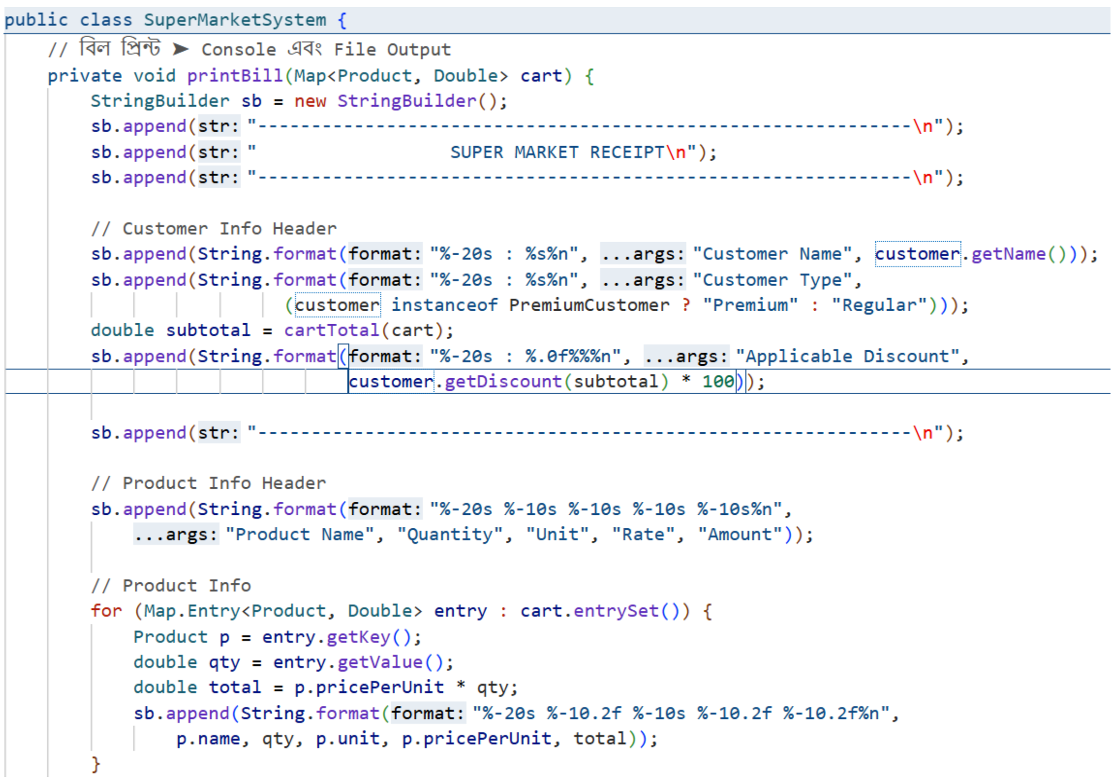
```java
public class SuperMarketSystem {
    // পণ্য কেনার প্রক্রিয়া
    private void makeReceipt() {
        try {
            Utility.line();
            System.out.print(s: "Enter Customer Name: ");
            String name = sc.nextLine().trim();
            if (name.isEmpty()) name = "Guest";

            System.out.print(s: "Customer Type (1=Regular, 2=Premium): ");
            int type = Integer.parseInt(sc.nextLine());

            // Polymorphism (Runtime) ➤ ভিন্ন Customer object same reference
            customer = (type == 2) ? new PremiumCustomer(name) : new RegularCustomer(name);

            Map<Product, Double> cart = new LinkedHashMap<>();

            Utility.line();
            System.out.println(x: "Available Products:");
            for (Product p : inventory.values()) {
                System.out.printf(format: "%-15s Price: %.2f Tk | Stock: %.2f %s%n",
                                  p.name, p.pricePerUnit, p.availableQty, p.unit);
            }
```

Snippet-8:



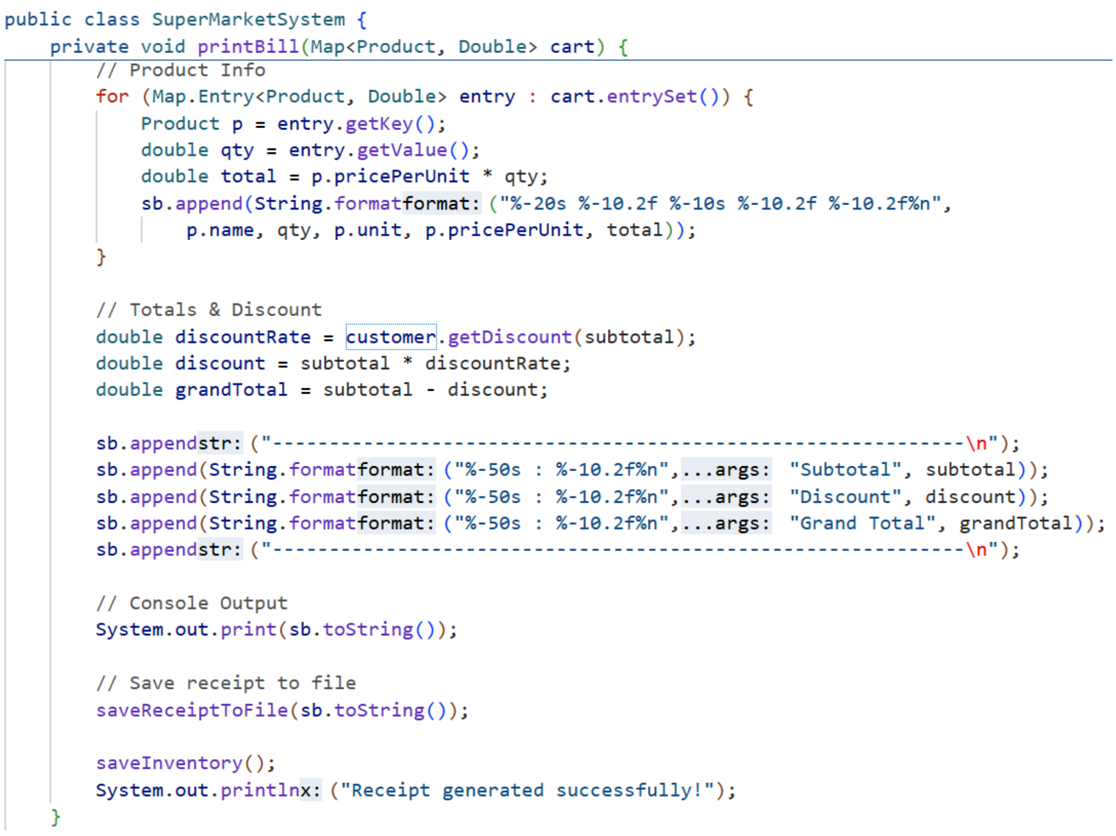
```
public class SuperMarketSystem {
    private void makeReceipt() {
            // একাধিক পণ্য যোগ করা
            while (true) {
                System.out.print(s: "Enter product name (type 'done' to finish): ");
                String pname = sc.nextLine().trim();
                if (pname.equalsIgnoreCase(anotherString: "done")) break;

                Product p = inventory.get(pname.toLowerCase());
                if (p == null) {
                    Utility.error(msg: "Product not found!");
                    continue;
                }
                System.out.print(s: "Enter quantity: ");
                double qty = Double.parseDouble(sc.nextLine());

                double cost = p.calculateCost(qty); // Exception Handling
                cart.put(p, qty);
                p.availableQty -= qty;
            }
            if (cart.isEmpty()) {
                Utility.error(msg: "No products purchased!");
                return;
            }
            printBill(cart);

        } catch (Exception e) {
            Utility.error("Error: " + e.getMessage());}
    }
```

Snippet-9:



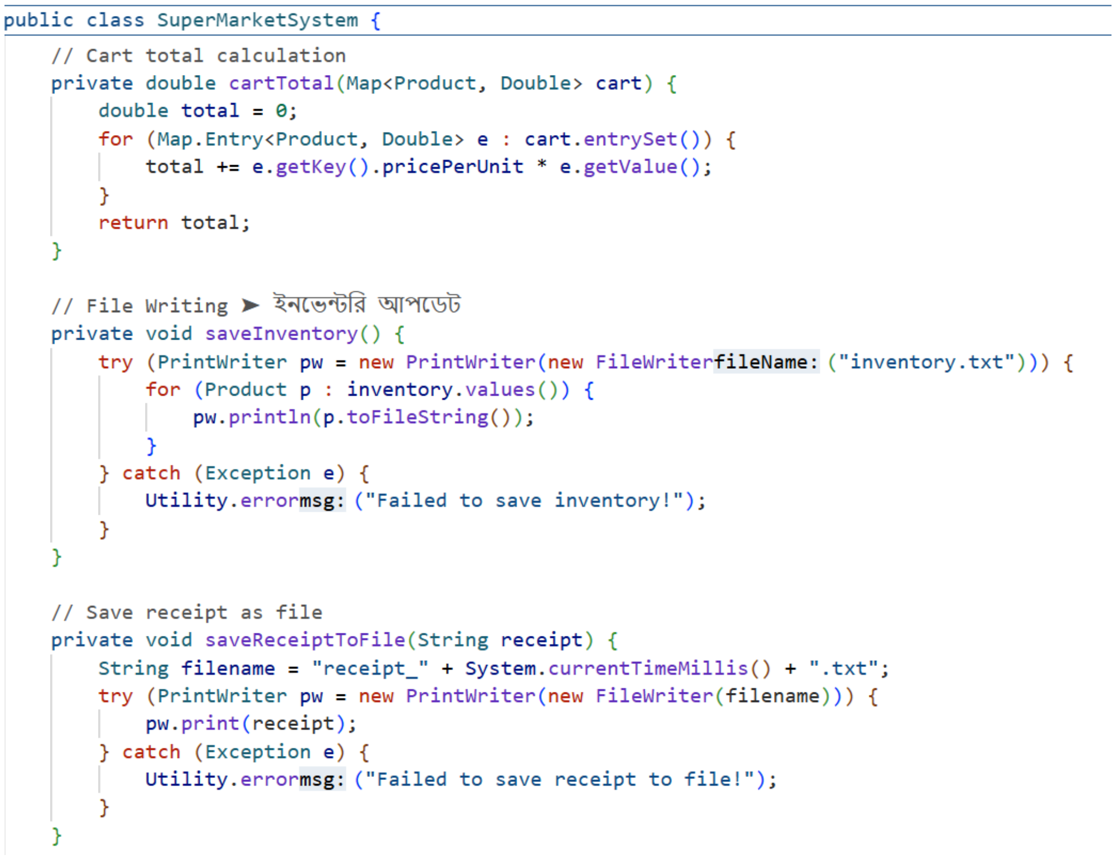
```
public class SuperMarketSystem {
    // বিল প্রিন্ট ➤ Console এবং File Output
    private void printBill(Map<Product, Double> cart) {
        StringBuilder sb = new StringBuilder();
        sb.append(str: "------------------------------------------------------------\n");
        sb.append(str: "                  SUPER MARKET RECEIPT\n");
        sb.append(str: "------------------------------------------------------------\n");

        // Customer Info Header
        sb.append(String.format(format: "%-20s : %s%n", ...args: "Customer Name", customer.getName()));
        sb.append(String.format(format: "%-20s : %s%n", ...args: "Customer Type",
                                  (customer instanceof PremiumCustomer ? "Premium" : "Regular")));
        double subtotal = cartTotal(cart);
        sb.append(String.format(format: "%-20s : %.0f%%%n", ...args: "Applicable Discount",
                                        customer.getDiscount(subtotal) * 100));

        sb.append(str: "------------------------------------------------------------\n");

        // Product Info Header
        sb.append(String.format(format: "%-20s %-10s %-10s %-10s %-10s%n",
            ...args: "Product Name", "Quantity", "Unit", "Rate", "Amount"));

        // Product Info
        for (Map.Entry<Product, Double> entry : cart.entrySet()) {
            Product p = entry.getKey();
            double qty = entry.getValue();
            double total = p.pricePerUnit * qty;
            sb.append(String.format(format: "%-20s %-10.2f %-10s %-10.2f %-10.2f%n",
                p.name, qty, p.unit, p.pricePerUnit, total));
        }
```

Snippet-10:



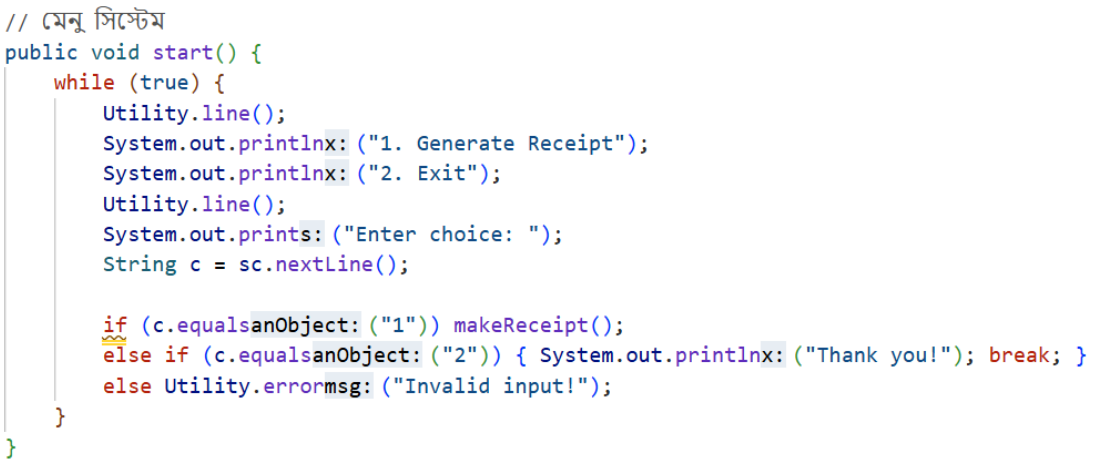
```
public class SuperMarketSystem {
    private void printBill(Map<Product, Double> cart) {
        // Product Info
        for (Map.Entry<Product, Double> entry : cart.entrySet()) {
            Product p = entry.getKey();
            double qty = entry.getValue();
            double total = p.pricePerUnit * qty;
            sb.append(String.formatformat: ("%-20s %-10.2f %-10s %-10.2f %-10.2f%n",
                p.name, qty, p.unit, p.pricePerUnit, total));
        }

        // Totals & Discount
        double discountRate = customer.getDiscount(subtotal);
        double discount = subtotal * discountRate;
        double grandTotal = subtotal - discount;

        sb.appendstr: ("--------------------------------------------------------------\n");
        sb.append(String.formatformat: ("%-50s : %-10.2f%n",...args:  "Subtotal", subtotal));
        sb.append(String.formatformat: ("%-50s : %-10.2f%n",...args:  "Discount", discount));
        sb.append(String.formatformat: ("%-50s : %-10.2f%n",...args:  "Grand Total", grandTotal));
        sb.appendstr: ("--------------------------------------------------------------\n");

        // Console Output
        System.out.print(sb.toString());

        // Save receipt to file
        saveReceiptToFile(sb.toString());

        saveInventory();
        System.out.printlnx: ("Receipt generated successfully!");
    }
```

Snippet-11:



```
public class SuperMarketSystem {
    // Cart total calculation
    private double cartTotal(Map<Product, Double> cart) {
        double total = 0;
        for (Map.Entry<Product, Double> e : cart.entrySet()) {
            total += e.getKey().pricePerUnit * e.getValue();
        }
        return total;
    }

    // File Writing ➤ ইনভেন্টরি আপডেট
    private void saveInventory() {
        try (PrintWriter pw = new PrintWriter(new FileWriterfileName: ("inventory.txt"))) {
            for (Product p : inventory.values()) {
                pw.println(p.toFileString());
            }
        } catch (Exception e) {
            Utility.errormsg: ("Failed to save inventory!");
        }
    }

    // Save receipt as file
    private void saveReceiptToFile(String receipt) {
        String filename = "receipt_" + System.currentTimeMillis() + ".txt";
        try (PrintWriter pw = new PrintWriter(new FileWriter(filename))) {
            pw.print(receipt);
        } catch (Exception e) {
            Utility.errormsg: ("Failed to save receipt to file!");
        }
    }
```

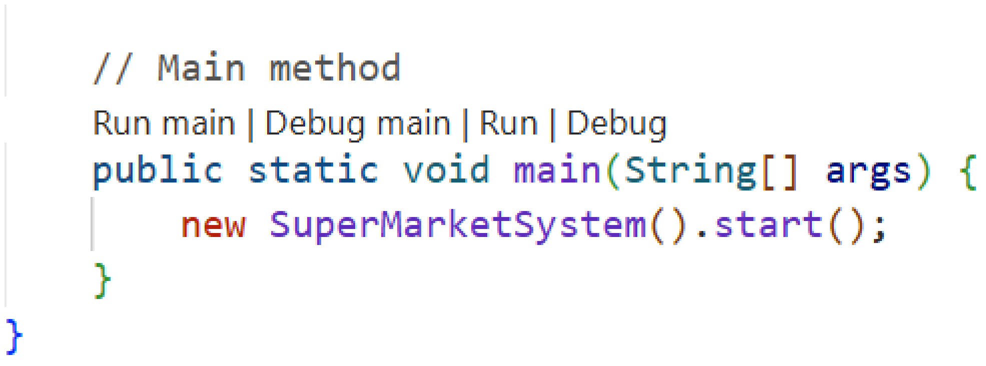
Snippet-12:



```
// মেনু সিস্টেম
public void start() {
    while (true) {
        Utility.line();
        System.out.printlnx: ("1. Generate Receipt");
        System.out.printlnx: ("2. Exit");
        Utility.line();
        System.out.prints: ("Enter choice: ");
        String c = sc.nextLine();

        if (c.equalsanObject: ("1")) makeReceipt();
        else if (c.equalsanObject: ("2")) { System.out.printlnx: ("Thank you!"); break; }
        else Utility.errormsg: ("Invalid input!");
    }
}
```

Snippet-13:

- **Main Class**:



```
    // Main method
    Run main | Debug main | Run | Debug
    public static void main(String[] args) {
        new SuperMarketSystem().start();
    }
}
```

Snippet-14:

3.3 Conclusions

The proposed model of **SmartBill** demonstrates a **modular, object-oriented approach** to building a receipt management system. By dividing the system into clear components—**Customer classes, Product management, Inventory handling, and Billing logic**—the model ensures:

1. **Scalability:** New products, customer types, or features can be added with minimal changes.
2. **Maintainability:** Each class has a specific responsibility, making the code easier to understand and debug.
3. **Accuracy:** Automated subtotal, discount, and grand total calculation reduces human errors.
4. **Reusability:** Components like Utility and Discountable interface can be reused in other projects.

Chapter 4: Implementation of SmartBill

4.1 Introduction

The system is implemented entirely in Java using **OOP concepts**. Key features include **runtime and compile-time polymorphism**, **exception handling**, and **file-based inventory management**.

4.2 Overview

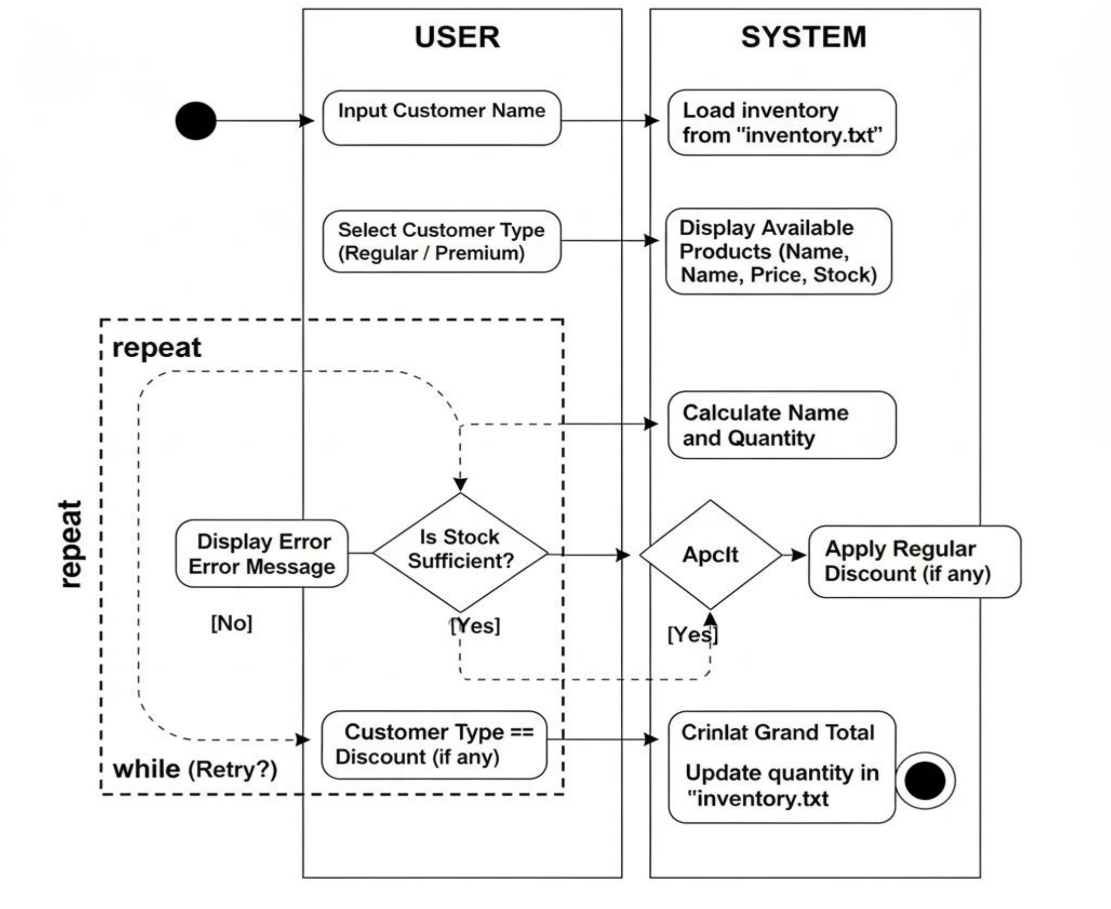

USER
SYSTEM
Input Customer Name
Load inventory from "inventory.txt"
Select Customer Type (Regular / Premium)
Display Available Products (Name, Name, Price, Stock)
repeat
Calculate Name and Quantity
repeat
Display Error Error Message
Is Stock Sufficient?
Apclt
Apply Regular Discount (if any)
[No]
[Yes]
[Yes]
Customer Type == Discount (if any)
while (Retry?)
Crinlat Grand Total
Update quantity in "inventory.txt

Figure-1: Overview of the SmartBill working process

1. **Load inventory** from inventory.txt.
2. Ask the **customer name and type** (Regular / Premium).
3. Display **available products** with prices and stock.
4. Input **product name and quantity**, validating stock availability.
5. Calculate **subtotal, discount, and grand total**.
6. Print **receipt** and update inventory in the file.

4.3 Classes and Methods Explanation

1. **Customer (Abstract Class)**: Stores customer name, implements Discountable.
2. **RegularCustomer / PremiumCustomer**: Override getDiscount() to apply 5% or 15% discount.
3. **Product Class**:

   - calculateCost(double qty) – Computes product cost; throws exception if invalid quantity.
   - fromString(String line) – Creates product from file data.
   - toFileString() – Converts product data for saving to file.

4. **SuperMarketSystem**: Handles shopping flow, manages cart, generates receipt, saves inventory.
5. **Utility**: Static helper methods for printing lines and error messages.

4.4 Exception Handling and Polymorphism Usage

1. **Runtime Polymorphism**: Customer discount calculation based on type.
2. **Compile-time Polymorphism**: Product calculateCost() method overloading.
3. **Exception Handling**:

   - Invalid quantity (<=0)
   - Quantity exceeding available stock
   - File read/write errors

4.5 Conclusions

1. SmartBill demonstrates OOP concepts in action.
2. Provides robust error handling and accurate calculations.
3. Flexible enough for enhancements and GUI integration.

Chapter 5: Experimental Results and Evaluation

5.1 Introduction

The experimental evaluation of SmartBill is an essential step to ensure that the system functions correctly, reliably, and efficiently under realistic usage scenarios. The primary purpose of this chapter is to demonstrate that the billing system is capable of handling multiple customer types, managing inventory accurately, calculating discounts, and generating well-formatted receipts. This chapter provides a detailed description of the testing process, sample transactions, observed results, and their analysis.

5.2 Objectives of Testing

1. Verify Functional Accuracy:

- Ensure that SmartBill correctly calculates the total cost of purchased products.
- Verify that discounts are applied accurately for both Regular and Premium customers.
- Confirm that inventory is updated correctly after each purchase.

2. Test System Robustness:

- Check how the system handles invalid inputs such as negative quantities, quantities exceeding stock, or non-existent product names.
- Validate that exception handling works as intended without crashing the system.

3. Evaluate User Interaction:

- Assess the clarity of prompts and ease of use for entering customer information and selecting products.
- Ensure that the receipt output is readable, properly formatted, and contains all necessary details (customer name, type, items, quantity, rate, subtotal, discount, grand total).

4. Simulate Real-World Scenarios:

- Conduct multiple transactions with different customer types to simulate daily retail operations.
- Include edge cases such as purchasing all stock of a product, buying multiple different products, or attempting invalid transactions.

5.3 Methodology

1. Inventory Preparation:

- A sample inventory file is prepared with 10–15 products, including items with varying prices and stock quantities.

Example:

```
Rice, 50, 60

Sugar, 30, 50

Oil, 20, 120

Milk, 40, 35

Biscuits, 60, 25
```

- This ensures a realistic variety of products for testing.

2. Customer Simulation:

- Two customer types are considered:
  - ➢ Regular Customer: receives 5% discount if total ≥ 150
  - ➢ Premium Customer: receives 15% discount if total ≥ 150

- Multiple test cases are designed for both customer types.

3. Transaction Execution:

- Customers select multiple products and specify quantities.
- The system calculates the total cost for each product, applies the applicable discount, and generates the receipt.
- Inventory quantities are updated in the file after each transaction.

4. Exception and Error Handling

- Attempt to purchase quantities exceeding available stock.
- Enter negative or zero quantities.
- Enter invalid product names.
- Observe how the system responds and prevents errors.

5.4 Result Analysis

1. Correct calculation of **subtotal, discount, and grand total**.
2. Inventory updated accurately after purchases.
3. Appropriate **error messages** for invalid input or stock issues.
4. Discounts correctly applied for **Regular (5%) and Premium (15%) customers**.

5.5 Sample Receipt Output

```
------------------------------------------------------------
                  SmartBill
------------------------------------------------------------
Customer Name        : Hasib
Customer Type        : Premium
Applicable Discount  : 15%
------------------------------------------------------------
Product Name         Quantity   Unit       Rate       Amount
Cycle                1.00       pcs        8000.00    8000.00
Yoga Mat             2.00       pcs        800.00     1600.00
Football             2.00       pcs        799.00     1598.00
Torch                2.00       pcs        250.00     500.00
------------------------------------------------------------
Subtotal                                            : 11698.00
Discount                                            : 1754.70
Grand Total                                         : 9943.30
------------------------------------------------------------
```

Snippet-15: Smartbill

5.6 Conclusions

1. The testing of SmartBill demonstrates that the system is accurate, reliable, and user-friendly.

2. It correctly calculates subtotal, discount, and grand total for both Regular and Premium customers.

3. Exception handling effectively manages invalid inputs such as negative quantities, exceeding stock, or incorrect product names.

4. Inventory is updated accurately after each transaction, ensuring realistic stock management.

5. Receipts are well-formatted and easy to understand.

6. The system successfully applies OOP principles like polymorphism, abstraction, and method overloading.

Chapter 6: Future Scope and Limitation

6.1 Future Scope of SmartBill

1. Upgrade to **GUI or web-based interface** for better usability.
2. Integrate **barcode scanner support** for faster billing.
3. Multiple payment options (cash, card, mobile banking).
4. **Sales and inventory reporting** dashboards.
5. Multi-user and networked store support.

6.2 Limitation

1. Limited to **single-user console environment**.
2. Inventory management **relies on text file**.
3. Not suitable for **large-scale retail chains** without further development.

6.3 Conclusions

SmartBill is a reliable, small-scale billing system that demonstrates **core OOP concepts**, provides **accurate receipts**, and **maintains inventory** effectively. Future upgrades can turn it into a comprehensive retail management system.

Bibliography

- https://www.geeksforgeeks.org/java/object-oriented-programming-oops-concept-in-java/
- https://www.tutorialspoint.com/java/java_oops_concepts.htm
- Effective Java by **Joshua Bloch**