**A logo for a university

Description automatically generated**

**Alliance School of Advanced Computing**

**Department of Computer Science and Engineering**

**Mini Project Report**

**Course Code:**

**Course Title: JAVA FULL STACK DEVELOPMENT**

**Semester : 4th semester**

**Course : CSE-GENERAL**

**Faculty Guide:**

**Submitted by**

**Team: Java Jaguars**

**•  Soundarya N (2023BCSE07AED604)**

**• Anantapur shaik shane Rabbani (2023BCSE07AED631)**

**2025-26**

**QUESTION:**

**Online Store System**

**Requirements:**

**Interface: Store Operations with methods add Product, buy Product viewCart, and checkout**

**Abstract Class: Product with attributes Vike productip, name price and stock**

**Concrete Class: Onlinestore to manage inventory and customer purchases**

**Main Class: Allow users to browse, add products to cart, and checkout**

**Java Concepts Used:**

**1. Object-Oriented Programming (OOP) Concepts:**

* **Classes and Objects**:
  + Classes like Product, Electronics, Clothing, etc., are defined to represent different types of products.
  + Objects are instances of these classes. For example, laptop, tshirt, and notebook are objects of the respective product classes.
* **Inheritance**:
  + The Electronics, Clothing, Footwear, and Stationary classes all inherit from the Product class, which is a fundamental OOP concept.
  + Inheritance allows these subclasses to reuse the properties and behaviors (methods) of the Product class while also adding their own specific properties (like brand for Electronics and size for Clothing and Footwear).
* **Polymorphism**:
  + The method displayProduct() is defined as an abstract method in the Product class and is overridden in each subclass. This allows the program to call displayProduct() on any type of Product (whether it's Electronics, Clothing, etc.), and the correct version of the method (based on the object's actual type) will be executed.
  + This is an example of *method overriding*, which is a form of polymorphism.
* **Encapsulation**:
  + The Product class and its subclasses use private fields (like productId, name, price, and stock) and provide public getter and setter methods to access and modify these fields. This is a key aspect of encapsulation, which hides the internal state of an object and provides controlled access to it.
* **Abstraction**:
  + The Product class is abstract, meaning it cannot be instantiated directly but serves as a blueprint for the subclasses. This is an example of abstraction, which allows you to define common functionality while leaving the details to the subclasses.

**2. Data Structures:**

* **Arrays** and **ArrayLists**:
  + An ArrayList<CartItem> is used in the OnlineStore class to store the shopping cart items. This allows dynamic resizing of the cart.
  + A HashMap<Integer, Product> is used to store the inventory of products, with the product's ID as the key and the Product object as the value. This allows for efficient lookups of products by ID.
* **Lists** and **Maps** are part of the Java Collections Framework, which provides flexible data structures for handling data in different ways (in this case, dynamically sized lists and key-value mappings).

**3. Constructors:**

* Each class defines a constructor to initialize objects with specific values. For example, the Electronics class has a constructor to initialize the product's ID, name, price, stock, and brand. Constructors are used to set up the initial state of objects when they are created.

**4. Access Modifiers:**

* The use of access modifiers like public, protected, and private is evident:
  + public is used for methods that need to be accessed from outside the class (e.g., displayProduct() and addProduct()).
  + protected is used in the Product class to allow access to the fields from subclasses.
  + private is used to hide internal data and ensure encapsulation (e.g., the fields in the Product class).

**5. Static Methods and Variables:**

* The CartItem class is marked as static. This means it is nested inside the OnlineStore class, but its instances do not depend on instances of OnlineStore itself. It's used here because each cart item doesn’t need to access any non-static members of the OnlineStore class.
* The inventory and cart variables in the OnlineStore class are instance variables (non-static), meaning each instance of OnlineStore will have its own separate inventory and cart.

**6. Scanner for User Input:**

* The Scanner class is used to read user input from the console. It is commonly used in Java for handling user interactions in console applications. The nextInt() method is used to read integer input for product IDs and quantities.

**7. Control Flow:**

* **Conditionals** (if, else) are used for checking conditions like whether the cart is empty or if there is enough stock to add a product to the cart.
* **Switch-Case** is used in the main() method to allow the user to choose different actions (view inventory, add to cart, checkout, etc.).

**8. Looping:**

* **While Loop**: The main menu uses a while (true) loop to continuously show the user options until they choose to exit the program (choice 5). This is a common technique in console applications for continuous interaction.
* **For-Each Loop**: The viewInventory() and viewCart() methods use for (Product product : inventory.values()) and for (CartItem item : cart) to iterate over the collection of products and cart items, respectively.

**9. Exception Handling (Could be added):**

* While there is no explicit exception handling in the code, there are potential opportunities for exception handling, such as when input is invalid or if an operation like adding an item with insufficient stock is attempted. Handling such situations with try-catch blocks could be a useful enhancement.

**10. Java Standard Library Classes:**

* **HashMap**: Used in OnlineStore to store the inventory, with the product ID as the key and the Product object as the value.
* **ArrayList**: Used to store the cart items in OnlineStore.
* **Scanner**: Used to read user input from the console.

**11. String Formatting:**

* String concatenation and basic formatting is used to generate output strings, such as when displaying product details in displayProduct().

**12. Primitive Data Types:**

* The code uses primitive types like int, double, and String. For example, productId, price, and stock are all basic data types, while name is a reference type (String).

**SOURCE CODE:**

**import java.util.\*;**

**// Abstract Product Class**

**abstract class Product {**

**protected int productId;**

**protected String name;**

**protected double price;**

**protected int stock;**

**public Product(int productId, String name, double price, int stock) {**

**this.productId = productId;**

**this.name = name;**

**this.price = price;**

**this.stock = stock;**

**}**

**public abstract void displayProduct();**

**public int getProductId() {**

**return productId;**

**}**

**public String getName() {**

**return name;**

**}**

**public double getPrice() {**

**return price;**

**}**

**public int getStock() {**

**return stock;**

**}**

**public void setStock(int stock) {**

**this.stock = stock;**

**}**

**}**

**// Electronics Class (inherits from Product)**

**class Electronics extends Product {**

**private String brand;**

**public Electronics(int productId, String name, double price, int stock, String brand) {**

**super(productId, name, price, stock);**

**this.brand = brand;**

**}**

**@Override**

**public void displayProduct() {**

**System.out.println(name + " (" + brand + ") - Price: $" + price + " - Stock: " + stock);**

**}**

**public String getBrand() {**

**return brand;**

**}**

**}**

**// Clothing Class (inherits from Product)**

**class Clothing extends Product {**

**private String size;**

**public Clothing(int productId, String name, double price, int stock, String size) {**

**super(productId, name, price, stock);**

**this.size = size;**

**}**

**@Override**

**public void displayProduct() {**

**System.out.println(name + " (Size: " + size + ") - Price: $" + price + " - Stock: " + stock);**

**}**

**public String getSize() {**

**return size;**

**}**

**}**

**// Footwear Class (inherits from Product)**

**class Footwear extends Product {**

**private String size;**

**public Footwear(int productId, String name, double price, int stock, String size) {**

**super(productId, name, price, stock);**

**this.size = size;**

**}**

**@Override**

**public void displayProduct() {**

**System.out.println(name + " (Size: " + size + ") - Price: $" + price + " - Stock: " + stock);**

**}**

**public String getSize() {**

**return size;**

**}**

**}**

**// Stationary Class (inherits from Product)**

**class Stationary extends Product {**

**private String type;**

**public Stationary(int productId, String name, double price, int stock, String type) {**

**super(productId, name, price, stock);**

**this.type = type;**

**}**

**@Override**

**public void displayProduct() {**

**System.out.println(name + " (" + type + ") - Price: $" + price + " - Stock: " + stock);**

**}**

**public String getType() {**

**return type;**

**}**

**}**

**// OnlineStore Class to manage inventory and cart**

**class OnlineStore {**

**private Map<Integer, Product> inventory = new HashMap<>();**

**private List<CartItem> cart = new ArrayList<>();**

**// Add product to inventory**

**public void addProduct(Product product) {**

**inventory.put(product.getProductId(), product);**

**}**

**// View all products in inventory**

**public void viewInventory() {**

**if (inventory.isEmpty()) {**

**System.out.println("No products available.");**

**} else {**

**System.out.println("\nAvailable Products:");**

**for (Product product : inventory.values()) {**

**product.displayProduct();**

**}**

**}**

**}**

**// Add product to shopping cart**

**public void addToCart(int productId, int quantity) {**

**Product product = inventory.get(productId);**

**if (product != null) {**

**if (product.getStock() >= quantity) {**

**cart.add(new CartItem(product, quantity));**

**product.setStock(product.getStock() - quantity);**

**System.out.println(quantity + " " + product.getName() + "(s) added to the cart.");**

**} else {**

**System.out.println("Insufficient stock for " + product.getName());**

**}**

**} else {**

**System.out.println("Product not found.");**

**}**

**}**

**// View shopping cart**

**public void viewCart() {**

**if (cart.isEmpty()) {**

**System.out.println("Your cart is empty.");**

**} else {**

**System.out.println("\nYour Cart:");**

**for (CartItem item : cart) {**

**System.out.println(item.getProduct().getName() + " - " + item.getQuantity() + " x $" + item.getProduct().getPrice() + " each");**

**}**

**}**

**}**

**// Checkout and display total cost**

**public void checkout() {**

**if (cart.isEmpty()) {**

**System.out.println("Your cart is empty.");**

**} else {**

**double total = 0;**

**System.out.println("\nCheckout Summary:");**

**for (CartItem item : cart) {**

**System.out.println(item.getProduct().getName() + " - " + item.getQuantity() + " x $" + item.getProduct().getPrice());**

**total += item.getProduct().getPrice() \* item.getQuantity();**

**}**

**System.out.println("Total: $" + total);**

**cart.clear();**

**}**

**}**

**// CartItem class to hold product and quantity in cart**

**private static class CartItem {**

**private Product product;**

**private int quantity;**

**public CartItem(Product product, int quantity) {**

**this.product = product;**

**this.quantity = quantity;**

**}**

**public Product getProduct() {**

**return product;**

**}**

**public int getQuantity() {**

**return quantity;**

**}**

**}**

**}**

**// Main Class (User Interface)**

**public class Main {**

**public static void main(String[] args) {**

**Scanner scanner = new Scanner(System.in);**

**OnlineStore store = new OnlineStore();**

**// Adding some products to the store**

**Electronics laptop = new Electronics(101, "Laptop", 1200.0, 10, "Dell");**

**Electronics smartphone = new Electronics(102, "Smartphone", 800.0, 20, "Samsung");**

**Clothing tshirt = new Clothing(201, "T-Shirt", 25.0, 50, "M");**

**Clothing jeans = new Clothing(202, "Jeans", 45.0, 30, "L");**

**Footwear sneakers = new Footwear(301, "Sneakers", 60.0, 40, "42");**

**Footwear sandals = new Footwear(302, "Sandals", 25.0, 25, "38");**

**Stationary notebook = new Stationary(401, "Notebook", 5.0, 100, "Writing");**

**Stationary pen = new Stationary(402, "Pen", 1.5, 200, "Writing");**

**store.addProduct(laptop);**

**store.addProduct(smartphone);**

**store.addProduct(tshirt);**

**store.addProduct(jeans);**

**store.addProduct(sneakers);**

**store.addProduct(sandals);**

**store.addProduct(notebook);**

**store.addProduct(pen);**

**while (true) {**

**// User Menu**

**System.out.println("\nWelcome to the Online Store!");**

**System.out.println("1. View Inventory");**

**System.out.println("2. Add Product to Cart");**

**System.out.println("3. View Cart");**

**System.out.println("4. Checkout");**

**System.out.println("5. Exit");**

**System.out.print("Enter your choice: ");**

**int choice = scanner.nextInt();**

**switch (choice) {**

**case 1:**

**store.viewInventory(); // Display all products in inventory**

**break;**

**case 2:**

**// Add product to cart**

**System.out.print("Enter product ID to add to cart: ");**

**int productId = scanner.nextInt();**

**System.out.print("Enter quantity: ");**

**int quantity = scanner.nextInt();**

**store.addToCart(productId, quantity);**

**break;**

**case 3:**

**store.viewCart(); // Display cart items**

**break;**

**case 4:**

**store.checkout(); // Checkout and display total amount**

**break;**

**case 5:**

**System.out.println("Thank you for shopping!"); // Exit the program**

**scanner.close();**

**return;**

**default:**

**System.out.println("Invalid choice, please try again.");**

**}**

**}**

**}**

**}**

**OUTPUT**

Welcome to the Online Store!

1. View Inventory

2. Add Product to Cart

3. View Cart

4. Checkout

5. Exit

Enter your choice: 1

Available Products:

Notebook (Writing) - Price: $5.0 - Stock: 100

Pen (Writing) - Price: $1.5 - Stock: 200

Laptop (Dell) - Price: $1200.0 - Stock: 10

Smartphone (Samsung) - Price: $800.0 - Stock: 20

T-Shirt (Size: M) - Price: $25.0 - Stock: 50

Jeans (Size: L) - Price: $45.0 - Stock: 30

Sneakers (Size: 42) - Price: $60.0 - Stock: 40

Sandals (Size: 38) - Price: $25.0 - Stock: 25

Welcome to the Online Store!

1. View Inventory

2. Add Product to Cart

3. View Cart

4. Checkout

5. Exit

Enter your choice: 2

Enter product ID to add to cart: 101

Enter quantity: 2

2 Laptop(s) added to the cart.

Welcome to the Online Store!

1. View Inventory

2. Add Product to Cart

3. View Cart

4. Checkout

5. Exit

Enter your choice: 2

Enter product ID to add to cart: 201

Enter quantity: 1

1 T-Shirt(s) added to the cart.

Welcome to the Online Store!

1. View Inventory

2. Add Product to Cart

3. View Cart

4. Checkout

5. Exit

Enter your choice: 2

Enter product ID to add to cart: 301

Enter quantity: 2

2 Sneakers(s) added to the cart.

Welcome to the Online Store!

1. View Inventory

2. Add Product to Cart

3. View Cart

4. Checkout

5. Exit

Enter your choice: 2

Enter product ID to add to cart: 401

Enter quantity: 2

2 Notebook(s) added to the cart.

Welcome to the Online Store!

1. View Inventory

2. Add Product to Cart

3. View Cart

4. Checkout

5. Exit

Enter your choice: 3

Your Cart:

Laptop - 2 x $1200.0 each

T-Shirt - 1 x $25.0 each

Sneakers - 2 x $60.0 each

Notebook - 2 x $5.0 each

Welcome to the Online Store!

1. View Inventory

2. Add Product to Cart

3. View Cart

4. Checkout

5. Exit

Enter your choice: 4

Checkout Summary:

Laptop - 2 x $1200.0

T-Shirt - 1 x $25.0

Sneakers - 2 x $60.0

Notebook - 2 x $5.0

Total: $2555.0

Welcome to the Online Store!

1. View Inventory

2. Add Product to Cart

3. View Cart

4. Checkout

5. Exit

Enter your choice: 5

Thank you for shopping!

=== Code Execution Successful ===

**CONCLUSION**

The program simulates an online store, demonstrating key **object-oriented programming (OOP)** concepts like **inheritance**, **polymorphism**, **encapsulation**, and **abstraction**. It allows users to view products, add them to a cart, and checkout with a total cost summary.

Key features of the program:

1. **Product Management**: The OnlineStore class acts as the central point to manage the store's inventory, where products are added and stock levels are updated based on user actions.
2. **Shopping Cart**: Users can add products to a cart, view their cart, and proceed to checkout, displaying a summary of the selected items and calculating the total cost.
3. **User Interaction**: The program interacts with the user through a text-based menu, allowing them to view products, add items to the cart, and proceed with the checkout.
4. **Use of Java Collections**: The program uses Java's HashMap to store the inventory and ArrayList to manage the cart. These collections ensure that data can be managed efficiently and dynamically.
5. **Flexibility**: Thanks to polymorphism, the program can easily be extended with more product types in the future by simply adding new subclasses of Product without major changes to the core logic.

**Strengths:**

* **Clear OOP Structure**: The program demonstrates good OOP practices with clear class hierarchies and separation of concerns between product types and the online store functionality.
* **Extensibility**: The code is easily extensible, allowing new product categories or features to be added with minimal changes.
* **User-Friendly Interface**: The text-based interface provides an interactive way for users to view products, manage their cart, and checkout.

**Potential Improvements:**

* **Input Validation**: Adding better input validation (e.g., for numeric input and valid product IDs) would enhance user experience by preventing errors and crashes.
* **Exception Handling**: Including more robust exception handling (e.g., for invalid input or out-of-stock situations) could make the program more resilient.
* **Advanced Features**: Features like removing items from the cart, updating quantities, and applying discounts could further enhance the functionality.
* **User Interface (UI)**: Transitioning to a graphical user interface (GUI) would make the program more user-friendly, especially for non-technical users.