**Data Structures Algorithms**

**EXERCISE 1:Inventory Management System**

**Source Code**

// Main.java

import java.util.\*;

// Step 3: Product class (Model)

class Product {

private String productId;

private String productName;

private int quantity;

private double price;

public Product(String productId, String productName, int quantity, double price) {

this.productId = productId;

this.productName = productName;

this.quantity = quantity;

this.price = price;

}

// Getters and setters

public String getProductId() {

return productId;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

public void setPrice(double price) {

this.price = price;

}

public void display() {

System.out.println("ID: " + productId + ", Name: " + productName +

", Quantity: " + quantity + ", Price: ₹" + price);

}

}

// Inventory Manager

class InventoryManager {

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

// Add product

public void addProduct(Product product) {

if (inventory.containsKey(product.getProductId())) {

System.out.println("Product ID already exists. Use updateProduct() instead.");

} else {

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

System.out.println("Product added: " + product.getProductId());

}

}

// Update product

public void updateProduct(String productId, int newQuantity, double newPrice) {

Product product = inventory.get(productId);

if (product != null) {

product.setQuantity(newQuantity);

product.setPrice(newPrice);

System.out.println("Product updated: " + productId);

} else {

System.out.println("Product not found: " + productId);

}

}

// Delete product

public void deleteProduct(String productId) {

if (inventory.remove(productId) != null) {

System.out.println("Product deleted: " + productId);

} else {

System.out.println("Product not found: " + productId);

}

}

// Display all products

public void displayAllProducts() {

if (inventory.isEmpty()) {

System.out.println("Inventory is empty.");

} else {

System.out.println("Inventory Contents:");

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

product.display();

}

}

}

}

// Step 6: Main class for testing

public class Main {

public static void main(String[] args) {

InventoryManager manager = new InventoryManager();

Product p1 = new Product("P001", "Mouse", 50, 599.99);

Product p2 = new Product("P002", "Keyboard", 30, 999.50);

Product p3 = new Product("P003", "Monitor", 15, 7899.00);

manager.addProduct(p1);

manager.addProduct(p2);

manager.addProduct(p3);

manager.addProduct(p1); // Duplicate test

System.out.println("\n--- After Adding Products ---");

manager.displayAllProducts();

manager.updateProduct("P002", 40, 950.00);

manager.updateProduct("P999", 10, 100.00); // Invalid update

System.out.println("\n--- After Updating Products ---");

manager.displayAllProducts();

manager.deleteProduct("P001");

manager.deleteProduct("P100"); // Invalid delete

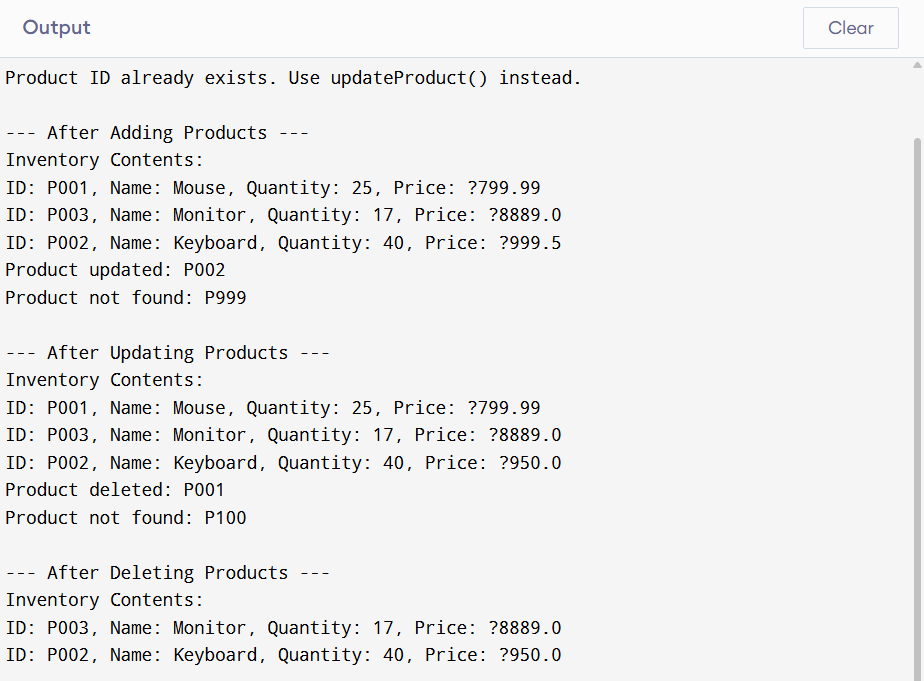
System.out.println("\n--- After Deleting Products ---");

manager.displayAllProducts();

}

}

**OUTPUT**

****

**ANALYSIS**

💡 Why HashMap?

1.Fast lookups: Average-case time complexity of O(1) for add, update, and delete operations.2.Optimized retrieval: Accessing products by their productId is efficient, even with a large dataset.🔍 Optimization Ideas:1.Use TreeMap instead of HashMap if you want sorted results by key.2.Add data validation logic.3.Load/store inventory from/to a file or database for persistence.