Exercise 1: Inventory Management System

# 1. Understand the Problem

Efficient data structures and algorithms are vital for handling large inventories as they directly impact performance in terms of speed and memory usage. Choosing the right data structure ensures quick access, insertion, and deletion of data.

Suitable Data Structures:

- ArrayList: Maintains order, good for small or sequential access.

- HashMap: Provides fast lookup, insertion, and deletion using unique keys like productId.

# 2. Setup

Create a Java project named `InventoryManagementSystem` with necessary classes to manage product data.

# 3. Implementation

* Product Class:

public class Product {  
 int productId;  
 String productName;  
 int quantity;  
 double price;  
  
 public Product(int productId, String productName, int quantity, double price) {  
 this.productId = productId;  
 this.productName = productName;  
 this.quantity = quantity;  
 this.price = price;  
 }  
  
 @Override  
 public String toString() {  
 return productId + " - " + productName + " | Qty: " + quantity + " | Price: ₹" + price;  
 }  
}

* Inventory Manager Class:

import java.util.HashMap;  
  
public class InventoryManager {  
 HashMap<Integer, Product> inventory = new HashMap<>();  
  
 public void addProduct(Product product) {  
 inventory.put(product.productId, product);  
 }  
  
 public void updateProduct(int productId, int quantity, double price) {  
 Product product = inventory.get(productId);  
 if (product != null) {  
 product.quantity = quantity;  
 product.price = price;  
 }  
 }  
  
 public void deleteProduct(int productId) {  
 inventory.remove(productId);  
 }  
  
 public void displayInventory() {  
 for (Product p : inventory.values()) {  
 System.out.println(p);  
 }  
 }  
}

* Test Class:

public class InventoryTest {  
 public static void main(String[] args) {  
 InventoryManager manager = new InventoryManager();  
 manager.addProduct(new Product(1, "Monitor", 10, 7500));  
 manager.addProduct(new Product(2, "Keyboard", 20, 1500));  
  
 manager.displayInventory();  
  
 manager.updateProduct(2, 25, 1400);  
 manager.deleteProduct(1);  
  
 manager.displayInventory();  
 }  
}  
4. Analysis

Using HashMap allows efficient operations due to key-based indexing:

- Add: O(1) average time

- Update: O(1) average time

- Delete: O(1) average time

Optimizations:

- Ensure proper hash function to avoid collisions.

- Use LinkedHashMap if order preservation is needed.

- Scale to databases for very large inventories.

# Output

