**Exercise 1: Inventory Management System**

**Scenario:**

You are developing an inventory management system for a warehouse. Efficient data storage and retrieval are crucial.

**Steps:**

1. **Understand the Problem:**

Explain why data structures and algorithms are essential in handling large inventories.

When managing a large inventory (thousands of products) data structures helps in storing and organizing the data effectively and algorithms help process that data (search, sort, update, etc.) efficiently for the operations like:

* Searching for a product
* Updating its quantity
* Deleting discontinued items need to be fast and efficient.

Discuss the types of data structures suitable for this problem.

* Array List
* Hashmap

1. **Setup:**
   * Create a new project for the inventory management system.
2. **Implementation:**
   * Define a class Product with attributes like **productId**, **productName**, **quantity**, and **price**.

Script:

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;

}

public void display() {

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

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

}

}

* + Choose an appropriate data structure to store the products (e.g., ArrayList, HashMap).
  + Implement methods to add, update, and delete products from the inventory.

**Script:**

import java.util.HashMap;

public class InventoryManager {

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

public void addProduct(Product product) {

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

System.out.println("Product with ID " + product.productId + " already exists.");

} else {

inventory.put(product.productId, product);

System.out.println("Product added.");

}

}

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

if (inventory.containsKey(productId)) {

Product p = inventory.get(productId);

p.quantity = newQuantity;

p.price = newPrice;

System.out.println("Product updated.");

} else {

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

}

}

public void deleteProduct(int productId) {

if (inventory.containsKey(productId)) {

inventory.remove(productId);

System.out.println("Product removed.");

} else {

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

}

}

public void displayInventory() {

if (inventory.isEmpty()) {

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

} else {

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

p.display();

}

}

}

}

**Main Java:**

public class Main {

public static void main(String[] args) {

InventoryManager manager = new InventoryManager();

Product p1 = new Product(101, "Laptop", 5, 75000);

Product p2 = new Product(102, "Keyboard", 15, 1500);

Product p3 = new Product(103, "Mouse", 30, 800);

manager.addProduct(p1);

manager.addProduct(p2);

manager.addProduct(p3);

System.out.println("\nInventory:");

manager.displayInventory();

manager.updateProduct(102, 20, 1600);

manager.deleteProduct(103);

System.out.println("\nUpdated Inventory:");

manager.displayInventory();

}

}

1. **Analysis:**
   * Analyze the time complexity of each operation (add, update, delete) in your chosen data structure.
     + 1. Add, Update and Delete – O(1)
       2. Displaying all – O(n)
   * Discuss how you can optimize these operations.

* HashMap for **fast access.**
* LinkedHashMap if **insertion order** matters.