**Exercise 1: Inventory Management System**

**Scenario:**

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

**Steps:**

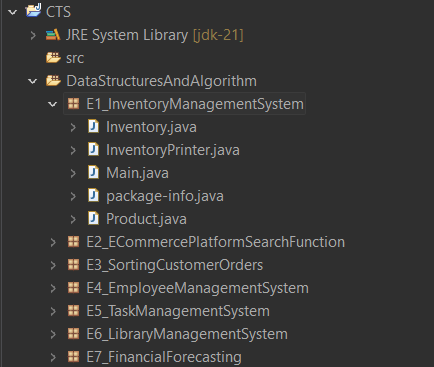
1. **Understand the Problem:**
   * **Explain why data structures and algorithms are essential in handling large inventories.**

Data structures and algorithms play a vital role in developing efficient software by reducing complexity and increasing performance. There are various types of data structures available for handling large inventories, each with its own advantages and disadvantages. Choosing the right data structure for a given problem or even a sub-problem is crucial for optimal performance.

* + **Discuss the types of data structures suitable for this problem.**

Arrays are one of the most commonly used data structures, but when handling large inventories, they are less efficient for search operations due to their linear search time **O(n)**. A HashMap is a better choice in such cases, as it provides **O(1)** time complexity for search, insert, and delete operations on average. A TreeMap can also be used when sorted ordering is required, but it is slightly slower with a time complexity of **O(log n)** for most operations.

1. **Setup:**
   * **Create a new project for the inventory management system.**

****

1. **Implementation:**
   * **Define a class Product with attributes like productId, productName, quantity, and 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.**

**Product.java**

package E1\_InventoryManagementSystem;

public class Product{

int productId;

String productName;

int productQuantity;

int productAmount;

public int getId() {

return productId;

}

public void setQty(int quantity) {

this.productQuantity = quantity;

}

public void setPrice(int amount) {

this.productAmount = amount;

}

public Product(int id, String name, int quantity, int amount){

this.productId = id;

this.productName = name;

this.productQuantity = quantity;

this.productAmount = amount;

}

public String toString(){

return "ID: " + productId + ", Name: " + productName + ", Quantity: " + productQuantity + ", Amount: " + productAmount;

}

}

**InventoryPrinter.java**

package E1\_InventoryManagementSystem;

import java.util.\*;

public class InventoryPrinter {

public void show(Collection<Product> products) {

if (products.isEmpty()) {

System.*out*.println("No products in the inventory!");

} else {

for (Product p : products) {

System.*out*.println(p);

}

}

}

}

**Inventory.java**

package E1\_InventoryManagementSystem;

import java.util.\*;

public class Inventory {

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

public void add(Product p) {

stock.put(p.getId(), p);

}

public void update(int id, int qty, int price) {

Product p = stock.get(id);

if (p != null) {

p.setQty(qty);

p.setPrice(price);

}

}

public void delete(int id) {

stock.remove(id);

}

public Collection<Product> all() {

return stock.values();

}

}

**Main.java**

package E1\_InventoryManagementSystem;

public class Main {

public static void main(String[] args) {

Inventory inventory = new Inventory();

InventoryPrinter printer = new InventoryPrinter();

Product pencil = new Product(1, "Pencil", 50, 15);

Product pen = new Product(2, "Pen", 100, 20);

Product scale = new Product(3, "Scale", 150, 30);

Product eraser = new Product(4, "Eraser", 110, 10);

inventory.add(pencil);

inventory.add(pen);

inventory.add(scale);

inventory.add(eraser);

inventory.update(1, 70, 10); // Update Pencil

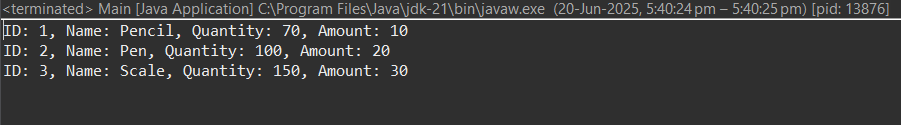
inventory.delete(4); // Delete Eraser

printer.show(inventory.all());

}

}

**Output:**

****

1. **Analysis:**
   * **Analyze the time complexity of each operation (add, update, delete) in your chosen data structure.**
2. **Add** – Time Complexity: O(1) – Direct key-based insertion
3. **Update** – Time Complexity: O(1) – Quick lookup by ID
4. **Delete** – Time Complexity: O(1) – Fast removal using product ID
5. **Show All** – Time Complexity: O(n) – Iterates over all products to display them
   * **Discuss how you can optimize these operations.**

* Use HashMap for fast access by productId.
* For sorted views, use TreeMap or sort ArrayList as needed.
* Use indexing for frequent searches on other fields (e.g., productName).