**Exercise 1: Inventory Management System**

**Scenario:**

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

**Understand the Problem:**

**Step 1: Understanding the Problem**

In a warehouse setting where inventory can span thousands of products, the choice of data structures and algorithms becomes essential for managing operations like insertion, search, update, and deletion. Efficient data handling ensures minimal response times, scalability, and improved user experience.

The key requirements of an inventory system are:

* Fast lookup of products based on unique identifiers
* Efficient updates of product quantity and pricing
* Quick deletion of discontinued or faulty products
* Minimal memory overhead and optimal runtime performance

To achieve these requirements, appropriate data structures should be selected.

**Suitable Data Structures**

1. **HashMap**:  
   A HashMap allows constant time average-case operations for insertion, lookup, update, and deletion when using a product's unique identifier such as productId as the key. This makes it highly suitable for large-scale inventory systems where fast retrieval is crucial.
2. **ArrayList**:  
   An ArrayList can be considered if the inventory size is relatively small or if the product lookup is not based on keys. However, in such cases, search operations become linear in time complexity which may lead to inefficiencies as the inventory size grows.

For this system, a HashMap is chosen due to its superior performance characteristics for key-based operations.

**Step 2: Project Setup**

A Java console-based project named InventoryManagementSystem is created. The project follows a modular structure with a clear separation between the data model and service layer.

**Step 3: Implementation**

**inventory/**

**├── Product.java**

**├── InventoryManager.java**

**└── Main.java**

**Product.java :-**

public class Product {

private int productId;

private String productName;

private int quantity;

private 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 int getProductId() { return productId; }

public String getProductName() { return productName; }

public int getQuantity() { return quantity; }

public double getPrice() { return price; }

public void setQuantity(int quantity) { this.quantity = quantity; }

public void setPrice(double price) { this.price = price; }

@Override

public String toString() {

return "Product ID: " + productId + ", Name: " + productName +

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

}

}

**ImventoryManager.java :-**

import java.util.HashMap;

public class InventoryManager {

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

public void addProduct(Product product) {

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

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

}

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

Product product = inventory.get(productId);

if (product != null) {

product.setQuantity(newQuantity);

product.setPrice(newPrice);

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

} else {

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

}

}

public void deleteProduct(int productId) {

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

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

} else {

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

}

}

public void displayInventory() {

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

System.out.println(product);

}

}

}

**Main.java :-**

public class Main {

public static void main(String[] args) {

InventoryManager manager = new InventoryManager();

Product p1 = new Product(101, "Laptop", 10, 75000);

Product p2 = new Product(102, "Keyboard", 50, 1500);

Product p3 = new Product(103, "Mouse", 75, 800);

manager.addProduct(p1);

manager.addProduct(p2);

manager.addProduct(p3);

manager.updateProduct(102, 60, 1400);

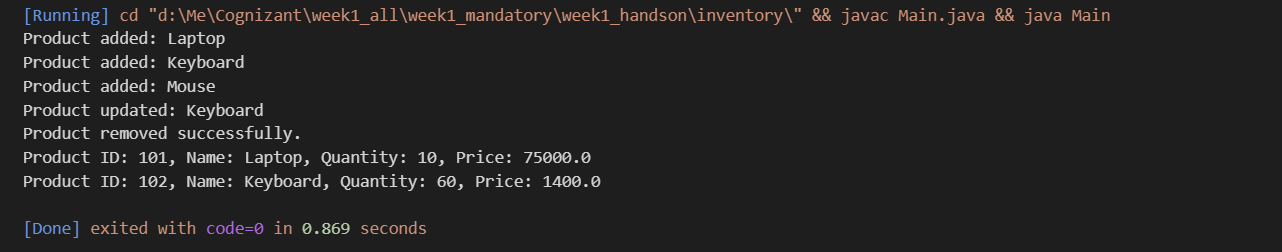
manager.deleteProduct(103);

manager.displayInventory();

}

}

**Output :-**



**Step 4: Analysis of Time Complexity**

**Add Product**:

* Operation: Inserting a product into the HashMap
* Time Complexity: Constant time on average

**Update Product**:

* Operation: Retrieving the product by key and modifying its fields
* Time Complexity: Constant time on average

**Delete Product**:

* Operation: Removing the product from the HashMap using its key
* Time Complexity: Constant time on average

**Display Inventory**:

* Operation: Iterating through all product entries
* Time Complexity: Linear time with respect to the number of products

**Optimizations**

* If product queries are based on attributes other than the product ID, consider maintaining secondary maps or indexes.
* Use synchronized or concurrent data structures in multi-threaded environments to avoid race conditions.
* For persistent storage, integrate with databases and implement caching strategies like LRU (Least Recently Used) to improve performance.