Exercise 1: Inventory Management System

1. Understand the Problem

Efficient data structures and algorithms are crucial when dealing with large inventories to ensure fast and reliable storage and retrieval of product data. With thousands of products in a warehouse, operations such as searching, adding, updating, or deleting must be optimized to avoid performance issues.

Data Structures Suitable for Inventory:

* ArrayList: Maintains insertion order and is good for iteration.
* HashMap: Allows quick access by key (e.g., productId), making add/update/delete faster.

In this solution, we use HashMap for constant-time operations based on productId.

1. Setup

Create a new Java project named `InventoryManagementSystem`. The project will consist of a `Product` class and a main class to handle the inventory logic.

1. Implementation

import java.util.HashMap; import java.util.Scanner;

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 String toString() {

return "ID: " + productId + ", Name: " + productName + ", Qty: " + quantity + ", Price: " + price;

}

}

public class InventoryManagementSystem {

static HashMap<Integer, Product> inventory = new HashMap<>(); static Scanner sc = new Scanner(System.in);

public static void addProduct() {

System.out.print("Enter ID, Name, Quantity, Price: "); int id = sc.nextInt();

String name = sc.next(); int qty = sc.nextInt();

double price = sc.nextDouble();

inventory.put(id, new Product(id, name, qty, price)); System.out.println("Product added.");

}

public static void updateProduct() { System.out.print("Enter ID to update: "); int id = sc.nextInt();

if (inventory.containsKey(id)) {

System.out.print("Enter New Name, Quantity, Price: "); String name = sc.next();

int qty = sc.nextInt();

double price = sc.nextDouble();

inventory.put(id, new Product(id, name, qty, price));

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

} else {

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

}

}

public static void deleteProduct() { System.out.print("Enter ID to delete: "); int id = sc.nextInt();

if (inventory.remove(id) != null) { System.out.println("Product deleted.");

} else {

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

}

}

public static void displayInventory() { for (Product p : inventory.values()) {

System.out.println(p);

}

}

public static void main(String[] args) { while (true) {

System.out.println("\n1.Add 2.Update 3.Delete 4.Display 5.Exit"); int choice = sc.nextInt();

switch (choice) {

case 1: addProduct(); break; case 2: updateProduct(); break; case 3: deleteProduct(); break;

case 4: displayInventory(); break; case 5: return;

}

}

}

}

1. Analysis

Time Complexity Analysis for HashMap-based Inventory:

* Add Product: O(1) — Direct insertion using productId as key.
* Update Product: O(1) — Replaces value for given key.
* Delete Product: O(1) — Removes entry using key.
* Display Inventory: O(n) — Iterates over all values.

Optimization:

* Using `HashMap` ensures constant-time operations for insert, update, and delete.
* For large-scale systems, additional features like indexing, caching, or database normalization can be applied.