Exercise 1: Inventory Management System

Scenario:

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

1. Understand the Problem

Why Data Structures and Algorithms are Essential:

Efficiency: Large inventories may contain thousands of products; accessing or modifying these efficiently requires optimal data structures.

Scalability: A good algorithm ensures performance doesn’t degrade as inventory grows.

Search, Update, Delete: Frequent operations that need to be fast (ideally O(1) or O(log n)) to ensure a responsive system.

Suitable Data Structures:

ArrayList<Product>: Simple and ordered, but search, update, and delete operations are O(n).

HashMap<Integer, Product> (Recommended): Provides O(1) average time complexity for add, update, and delete using the productId as the key.

TreeMap<Integer, Product>: Maintains sorted order of keys with O(log n) operations.

Best choice: HashMap<Integer, Product> for fast lookup by productId.

2. Setup

Java project named InventoryManagementSystem.

files:

Product.java

Inventory.java

Main.java

3.Implementation:

Code:

Product.java

public class Product {

int id;

String name;

int quantity;

double price;

public Product(int id, String name, int quantity, double price) {

this.id = id;

this.name = name;

this.quantity = quantity;

this.price = price;

}

public void display() {

System.out.printf("ID: %d | Name: %s | Qty: %d | Price: ₹%.2f%n", id, name, quantity, price);

}

}

Inventory.java

import java.util.ArrayList;

public class Inventory {

private ArrayList<Product> products = new ArrayList<>();

public void addProduct(Product p) {

products.add(p);

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

}

public void viewAll() {

if (products.isEmpty()) {

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

return;

}

for (Product p : products) {

p.display();

}

}

public Product searchById(int id) {

for (Product p : products) {

if (p.id == id) return p;

}

return null;

}

public boolean deleteProduct(int id) {

Product p = searchById(id);

if (p != null) {

products.remove(p);

return true;

}

return false;

}

public boolean updateQuantity(int id, int newQty) {

Product p = searchById(id);

if (p != null) {

p.quantity = newQty;

return true;

}

return false;

}

}

Main.java

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Inventory inventory = new Inventory();

Scanner sc = new Scanner(System.in);

int choice;

do {

System.out.println("\n=== Inventory Menu ===");

System.out.println("1. Add Product");

System.out.println("2. View All Products");

System.out.println("3. Search Product by ID");

System.out.println("4. Update Product Quantity");

System.out.println("5. Delete Product");

System.out.println("0. Exit");

System.out.print("Enter choice: ");

choice = sc.nextInt();

switch (choice) {

case 1 -> {

System.out.print("Enter ID: ");

int id = sc.nextInt();

sc.nextLine(); // clear buffer

System.out.print("Enter Name: ");

String name = sc.nextLine();

System.out.print("Enter Quantity: ");

int qty = sc.nextInt();

System.out.print("Enter Price: ");

double price = sc.nextDouble();

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

}

case 2 -> inventory.viewAll();

case 3 -> {

System.out.print("Enter ID to search: ");

int id = sc.nextInt();

Product p = inventory.searchById(id);

if (p != null) p.display();

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

}

case 4 -> {

System.out.print("Enter ID to update: ");

int id = sc.nextInt();

System.out.print("Enter new quantity: ");

int newQty = sc.nextInt();

if (inventory.updateQuantity(id, newQty))

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

else

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

}

case 5 -> {

System.out.print("Enter ID to delete: ");

int id = sc.nextInt();

if (inventory.deleteProduct(id))

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

else

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

}

case 0 -> System.out.println(" Exiting...");

default -> System.out.println(" Invalid option.");

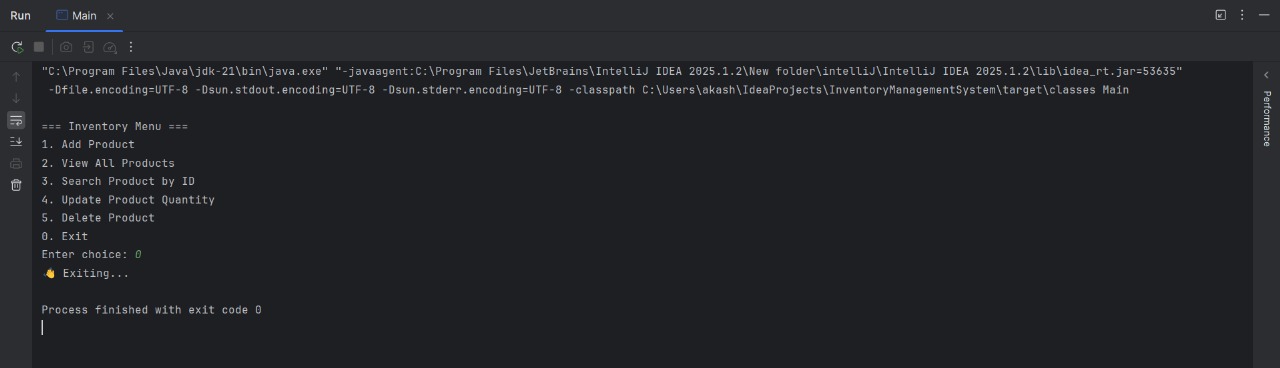
}

} while (choice != 0);

sc.close();

}

}

OUTPUT:

4. Analysis

Time Complexity (Using HashMap<Integer, Product>)

Operation Time Complexity

addProduct O(1)

updateProduct O(1)

deleteProduct O(1)

displayInventory O(n)