**Inventory Management System Solution**

**Why data structures and algorithms matter?**

* Large inventories need fast operations (adding, finding, updating items)
* Good data structures make these operations efficient
* Algorithms help with sorting, searching, and managing inventory data

**Suitable data structures:**

* HashMap: Best for quick lookup by product ID (O(1) time)
* ArrayList: Simple but slower for searches (O(n) time)
* TreeMap: Good if we need sorted products by ID

I'll choose HashMap because we need fast access by product ID, which is most common in inventory systems.

**Code Inventory Management System Solution**

import java.util.HashMap;

class Product {

String productId;

String productName;

int quantity;

double price;

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

this.productId = id;

this.productName = name;

this.quantity = qty;

this.price = price;

}

}

class InventorySystem {

private HashMap<String, Product> inventory;

public InventorySystem() {

inventory = new HashMap<>();

}

// Add a new product

public void addProduct(String id, String name, int qty, double price) {

if (!inventory.containsKey(id)) {

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

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

} else {

System.out.println("Product ID already exists!");

}

}

// Update product quantity

public void updateQuantity(String id, int newQty) {

if (inventory.containsKey(id)) {

Product p = inventory.get(id);

p.quantity = newQty;

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

} else {

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

}

}

// Delete a product

public void deleteProduct(String id) {

if (inventory.containsKey(id)) {

inventory.remove(id);

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

} else {

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

}

}

// Display a product

public void displayProduct(String id) {

Product p = inventory.get(id);

if (p != null) {

System.out.println("ID: " + p.productId);

System.out.println("Name: " + p.productName);

System.out.println("Quantity: " + p.quantity);

System.out.println("Price: $" + p.price);

} else {

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

}

}

}

public class Main {

public static void main(String[] args) {

InventorySystem myInventory = new InventorySystem();

// Test the system

myInventory.addProduct("P100", "Laptop", 50, 999.99);

myInventory.addProduct("P101", "Mouse", 100, 19.99);

myInventory.displayProduct("P100");

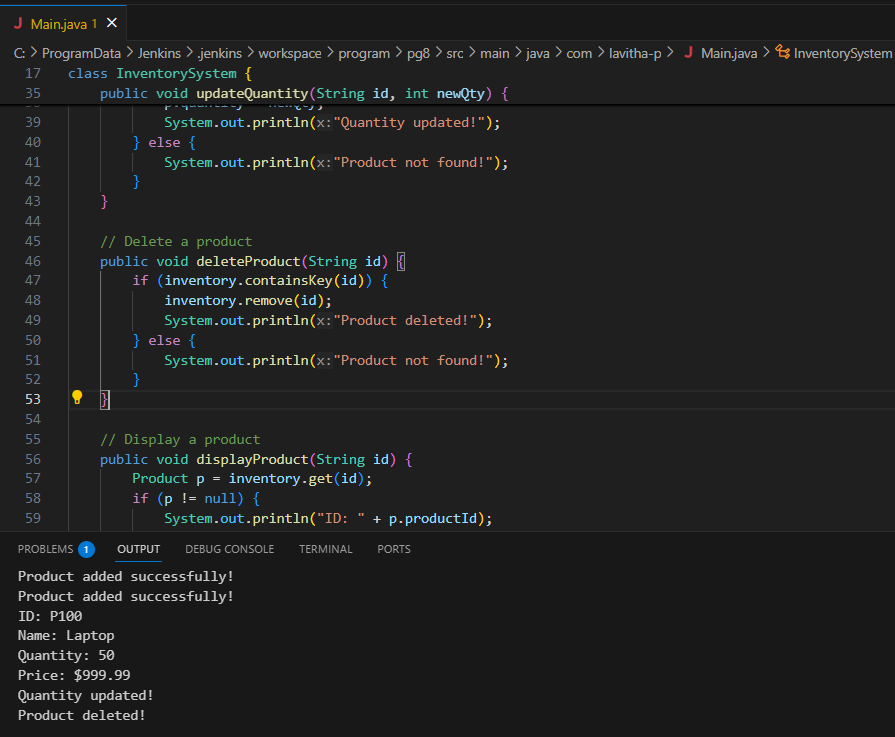
myInventory.updateQuantity("P100", 45);

myInventory.deleteProduct("P101");

}

}

**OUTPUT:**



**Analysis through this:**

Time Complexity:

* Add product: O(1) - HashMap insertion is constant time
* Update product: O(1) - HashMap lookup then update is constant time
* Delete product: O(1) - HashMap removal is constant time
* Display product: O(1) - HashMap lookup is constant time

**Optimization possibilities:**

1. If we need to frequently display all products sorted by ID, we could:

* Use a TreeMap instead (keeps keys sorted)
* But this would make operations O(log n) instead of O(1)

2. For searching by product name (not just ID), we could:

* Maintain a separate HashMap for name-to-ID mapping
* This would use more memory but allow fast name searches

3. For bulk operations, we could:

* Implement batch add/update methods
* Use multi-threading for very large inventories

This simple implementation gives us the core functionality with optimal performance for the most common inventory operations.