

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.
- Discuss the types of data structures suitable for this problem.

2. Setup:

- Create a new project for the inventory management system.

3. 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.

4. Analysis:

- Analyze the time complexity of each operation (add, update, delete) in your chosen data structure.
- Discuss how you can optimize these operations.

→ Step 1: Understand the Problem

Why Data Structures and Algorithms are Essential

In a warehouse, thousands of products may need to be tracked by ID, name, quantity, etc. Efficient algorithms and data structures are essential for:

- **Fast retrieval** (e.g., get a product by ID in constant time)
- **Efficient storage** (avoid redundancy)
- **Quick updates** (restocking, price changes)
- **Scalability** (handling large datasets as inventory grows)

Suitable Data Structures

- **HashMap**: Ideal for fast lookup, insertion, and deletion by productId ($O(1)$ average case).
- **ArrayList**: Good for sequential data, but inefficient for frequent lookups and deletions ($O(n)$).
- **TreeMap**: If sorted order of products is required, though slower ($O(\log n)$).

- **Conclusion:** Use **HashMap<Integer, Product>** for best performance.

Step 2: Setup a Java Project

You can set up a Java project in any IDE like IntelliJ or Eclipse:

Commands:

```
mkdir InventoryManagementSystem
```

```
cd InventoryManagementSystem
```

Step 3: Implementation

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 setProductName(String productName) { this.productName = productName; }  
    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 +
        '}';
    }
}

```

InventoryManager.java

```

import java.util.HashMap;

public class InventoryManager {

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

    public void addProduct(Product product) {
        if (inventory.containsKey(product.getProductID())) {
            System.out.println("Product ID already exists!");
        } else {
            inventory.put(product.getProductID(), product);
            System.out.println("Product added.");
        }
    }

    public void updateProduct(int productId, String name, int quantity, double price) {
        Product product = inventory.get(productId);
        if (product != null) {
            product.setProductName(name);
            product.setQuantity(quantity);
            product.setPrice(price);
            System.out.println("Product updated.");
        } else {
            System.out.println("Product not found.");
        }
    }

    public void deleteProduct(int productId) {
        if (inventory.remove(productId) != null) {
            System.out.println("Product deleted.");
        }
    }
}

```

```

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

public void printInventory() {
    for (Product product : inventory.values()) {
        System.out.println(product);
    }
}
}

```

Main.java

```

public class Main {
    public static void main(String[] args) {
        InventoryManager manager = new InventoryManager();
        manager.addProduct(new Product(101, "Keyboard", 20, 29.99));
        manager.addProduct(new Product(102, "Mouse", 35, 19.99));
        manager.updateProduct(101, "Mechanical Keyboard", 25, 39.99);
        manager.deleteProduct(102);
        manager.printInventory();
    }
}

```

Step 4: Analysis

Time Complexity (using HashMap<Integer, Product>)

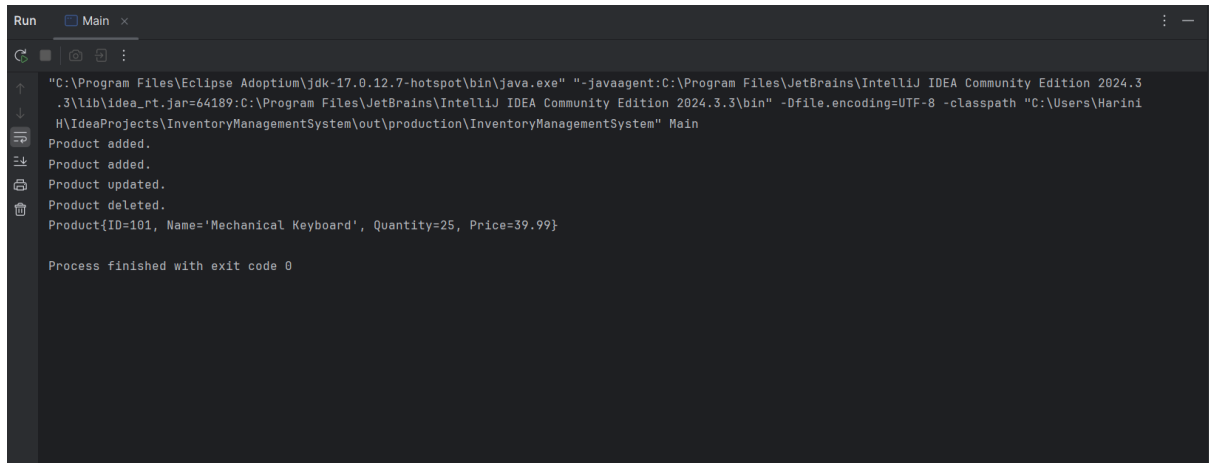
Operation	Time Complexity	Explanation
Add Product	O(1) avg	Uses put() method of HashMap
Update Product	O(1) avg	Uses get() and modifies fields
Delete Product	O(1) avg	Uses remove() by key
Print Inventory	O(n)	Iterates over all products

Worst-case time for HashMap operations is $O(n)$, but this is rare due to good hash functions and load factor management.

Optimization Tips

- Use **TreeMap** if sorted access by productId is required (but slower).
- Use **LinkedHashMap** if insertion order needs to be preserved.

OUTPUT:



```
Run Main x
"\"C:\\Program Files\\Eclipse Adoptium\\jdk-17.0.12-hotspot\\bin\\java.exe\" \"-javaagent:C:\\Program Files\\JetBrains\\IntelliJ IDEA Community Edition 2024.3\\lib\\idea_rt.jar=64189:C:\\Program Files\\JetBrains\\IntelliJ IDEA Community Edition 2024.3\\bin\" -Dfile.encoding=UTF-8 -classpath \"C:\\Users\\Harini\\IdeaProjects\\InventoryManagementSystem\\out\\production\\InventoryManagementSystem\" Main
Product added.
Product added.
Product updated.
Product deleted.
Product{ID=101, Name='Mechanical Keyboard', Quantity=25, Price=39.99}

Process finished with exit code 0
```