**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.

**Solution:**

1. **Why are Data Structures and Algorithms Essential?**

When you manage an inventory system, especially in a warehouse, you're likely dealing with thousands or millions of products. You’ll need to:

* Search for products quickly.
* Add, update, or remove products efficiently.
* Keep track of stock levels and prices.
* Handle real-time updates when sales happen.

Without using the right data structure, even simple tasks can become slow and inefficient.

Example:

If you stored all products in a basic array or list, finding a product would take O(n) time in the worst case. But with a HashMap, you can do it in O(1) on average.

| **Data Structure** | **Usage** | **Time Complexities (average)** | **Pros** | **Cons** |
| --- | --- | --- | --- | --- |
| **ArrayList** | Simple, ordered list | Search: O(n), Add: O(1), Delete: O(n) | Easy to use | Slow for searching or deleting |
| **LinkedList** | When you frequently insert/delete | Insert/Delete: O(1), Search: O(n) | Good for queues | Poor for search |
| **HashMap** (Best Choice) | Key-value pair (key = productId) | All: O(1) average | Fast lookup/update | No sorting |
| **TreeMap** | Sorted key access | All: O(log n) | Keeps items sorted by ID | Slightly slower |
| **Set** | Unique product IDs | O(1) with HashSet | Prevents duplicates | Not ideal for storing full product data |

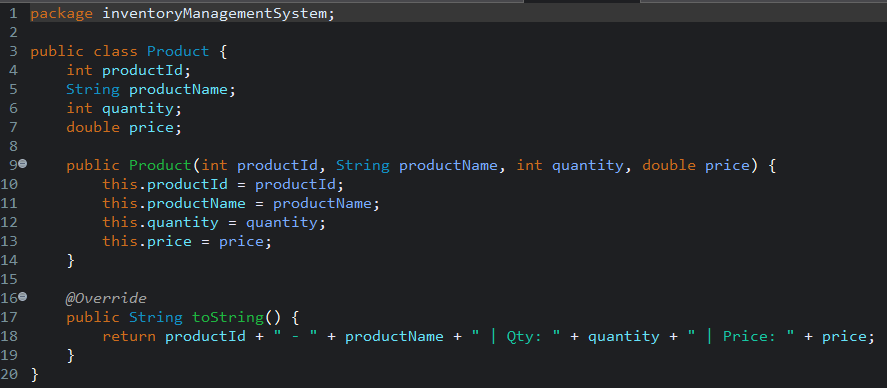
**2. Setup**

You need to:

1. Create a **Java Project** (use Eclipse, IntelliJ, or any IDE).
2. Add two classes:
   * Product → stores product details.
   * InventoryManager → manages the product list using a HashMap.

**Product Class (Product.java)**

This class represents the data model.



**b) InventoryManager Class (InventoryManager.java)**

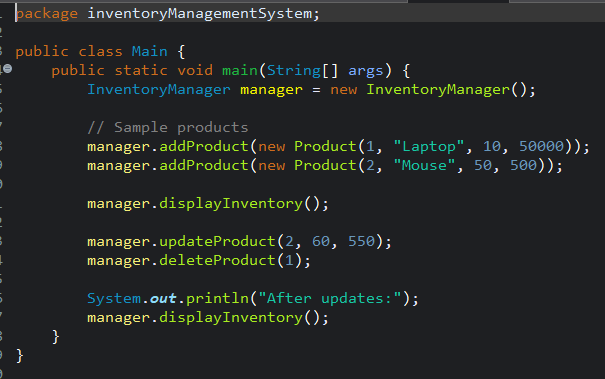
Manages all products using a HashMap.



Uses a HashMap<Integer, Product> to store products by their ID.

* Methods:
  + addProduct() – adds a product to the inventory.
  + updateProduct() – updates quantity and price.
  + deleteProduct() – removes a product.
  + displayInventory() – shows all products.

c) **Main Class(Main.java)**



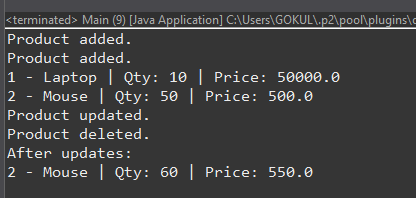
Creates an instance of InventoryManager.

Adds sample products.

Performs update and delete operations.

Displays products before and after changes.

**Output:**

****

**4.Analyse**

| **Operation** | **Data Structure** | **Time Complexity** | **Description** |
| --- | --- | --- | --- |
| **Add** | HashMap | O(1) avg / O(n) worst | Constant-time insert by key |
| **Update** | HashMap | O(1) avg / O(n) worst | Retrieve and modify object |
| **Delete** | HashMap | O(1) avg / O(n) worst | Remove by key |
| **Search** | HashMap | O(1) avg / O(n) worst | Fast key-based lookup |