**Exercise 1: Inventory Management System**

**InventoryManagementSystem.java**

**package** mypackage;

**import** java.util.\*;

**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;

}

}

**public** **class** InventoryManagementSystem {

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

**private** **static** Scanner *sc* = **new** Scanner(System.***in***);

**public** **static** **void** addProduct() {

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

**int** id = *sc*.nextInt();

*sc*.nextLine();

**if** (*inventory*.containsKey(id)) {

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

**return**;

}

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

String name = *sc*.nextLine();

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

**int** qty = *sc*.nextInt();

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

**double** price = *sc*.nextDouble();

Product product = **new** Product(id, name, qty, price);

*inventory*.put(id, product);

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

}

**public** **static** **void** updateProduct() {

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

**int** id = *sc*.nextInt();

*sc*.nextLine();

**if** (!*inventory*.containsKey(id)) {

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

**return**;

}

Product product = *inventory*.get(id);

System.***out***.print("Enter new Product Name: ");

String name = *sc*.nextLine();

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

**int** qty = *sc*.nextInt();

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

**double** price = *sc*.nextDouble();

product.setProductName(name);

product.setQuantity(qty);

product.setPrice(price);

System.***out***.println("Product updated successfully!");

}

**public** **static** **void** deleteProduct() {

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

**int** id = *sc*.nextInt();

**if** (*inventory*.remove(id) != **null**) {

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

} **else** {

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

}

}

**public** **static** **void** displayInventory() {

**if** (*inventory*.isEmpty()) {

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

**return**;

}

**for** (Product p : *inventory*.values()) {

System.***out***.println(p);

}

}

**public** **static** **void** main(String[] args) {

**int** choice;

**do** {

System.***out***.println("\nInventory Management System");

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

System.***out***.println("2. Update Product");

System.***out***.println("3. Delete Product");

System.***out***.println("4. Display Inventory");

System.***out***.println("5. Exit");

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

choice = *sc*.nextInt();

**switch** (choice) {

**case** 1: *addProduct*(); **break**;

**case** 2: *updateProduct*(); **break**;

**case** 3: *deleteProduct*(); **break**;

**case** 4: *displayInventory*(); **break**;

**case** 5: System.***out***.println("Exiting..."); **break**;

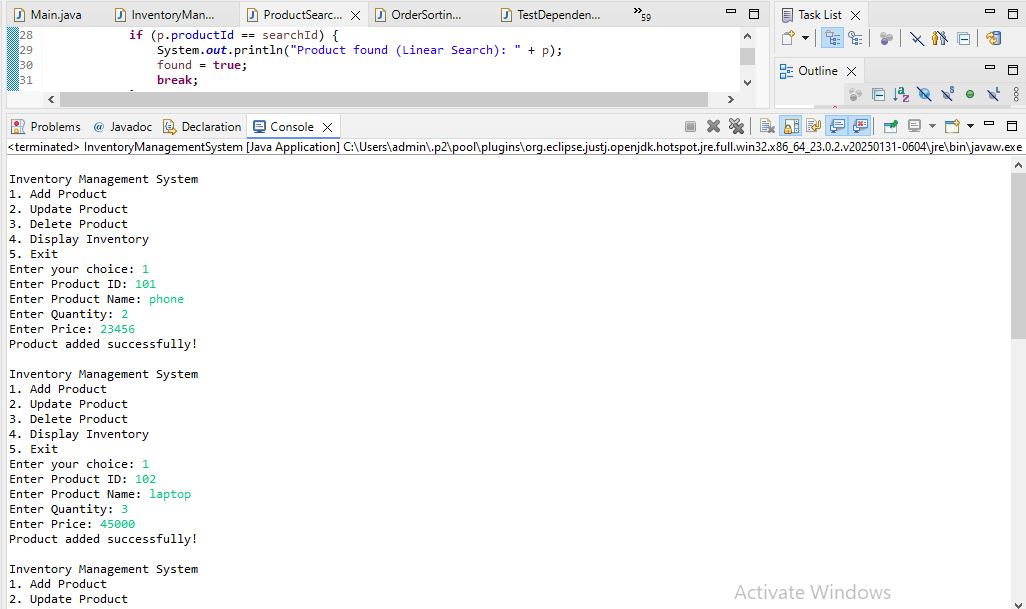
**default**: System.***out***.println("Invalid choice!");

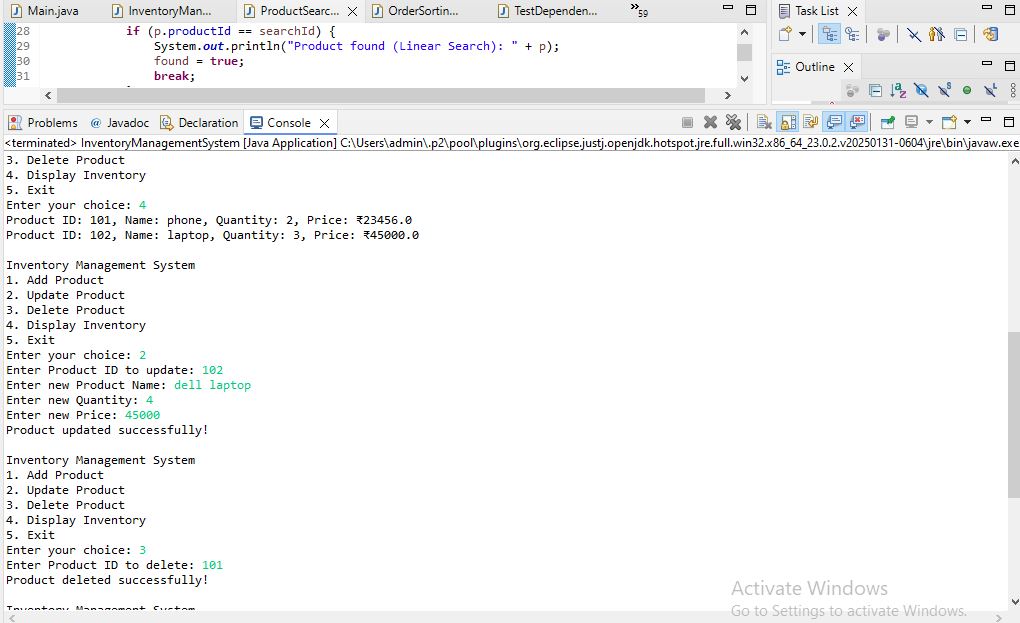
}

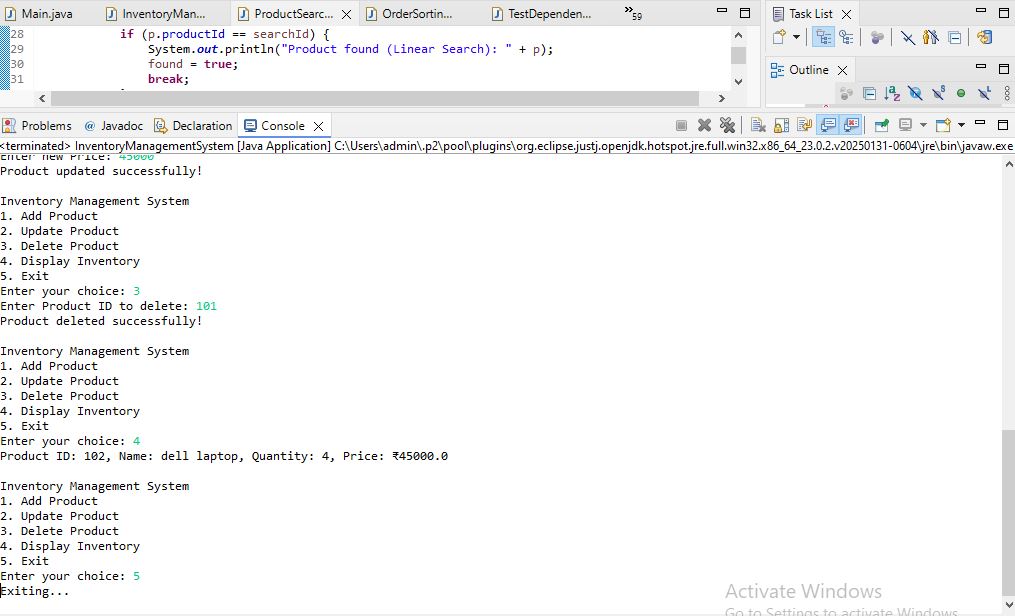
} **while** (choice != 5);

}

}







**➔ Why are data structures and algorithms essential in handling large**

**inventories?**

* In a warehouse, there can be thousands or even millions of products.
* Efficient storage and retrieval of product information is critical for:
  + Fast searching
  + Quick updates
  + Real-time stock management
* **Data structures** help organize data for faster access.
* **Algorithms** help perform operations (like search, insert, update, delete) efficiently.

Without proper data structures, operations may take longer time, causing system slowdowns and operational delays.

**➔ Suitable data structures for this problem:**

| **Data Structure** | **Use Case** | **Advantage** |
| --- | --- | --- |
| **ArrayList** | Simple, ordered list of products | Easy to implement, good for small data |
| **HashMap** | Key-value pair storage | Fast lookup, insertion, deletion using productId as key |
| **TreeMap** | Sorted key-value storage | Allows fast sorted operations |

In this system, we use **HashMap** because:

* Product ID is unique.
* Fast access by key (productId).
* Insertion, update, delete, and search are very efficient.

**2. Setup**

* A new project is created for the Inventory Management System.
* The main class is named InventoryManagementSystem.
* A Product class is defined to store product details.

**3. Implementation**

**➔ Product class**

Attributes:

* productId (int): Unique ID for each product.
* productName (String): Name of product.
* quantity (int): Stock quantity available.
* price (double): Price of product.

**➔ Data Structure**

* We use **HashMap<Integer, Product> inventory**:
  + Key: productId
  + Value: Product object
* HashMap allows quick access based on productId.

**➔ Operations Implemented**

| **Operation** | **Description** |
| --- | --- |
| **Add Product** | Adds new product into inventory |
| **Update Product** | Updates details of existing product |
| **Delete Product** | Removes product from inventory |
| **Display Inventory** | Displays all products currently in inventory |

**4. Analysis**

**➔ Time Complexity of Operations using HashMap**

| **Operation** | **Time Complexity** | **Explanation** |
| --- | --- | --- |
| **Add Product** | O(1) | Inserting into HashMap |
| **Update Product** | O(1) | Access by key and update fields |
| **Delete Product** | O(1) | Remove by key |
| **Display Inventory** | O(n) | Traverses all products |

HashMap provides **constant time (O(1))** for add, update, and delete operations.  
Only display requires linear time (O(n)) since we need to go through all entries.

**➔ How to optimize further?**

* In real-world systems, as the data grows larger:
  + Use **database indexing** for very large datasets.
  + Use **concurrent data structures** for multi-user systems.
  + Use **persistent storage** (like SQL or NoSQL databases) for durability.
  + Implement **caching** for frequent queries.

For simple warehouse systems with moderate data size, HashMap provides excellent performance.