**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.

**Code:  
package** io;

**import** java.util.HashMap;

**import** java.util.Scanner;

**class** Product {

**private** String productId;

**private** String productName;

**private** **int** quantity;

**private** **double** price;

**public** Product(String productId, String productName, **int** quantity, **double** price) {

**this**.productId = productId;

**this**.productName = productName;

**this**.quantity = quantity;

**this**.price = price;

}

**public** String getProductId() {

**return** productId;

}

**public** **void** setProductName(String name) {

**this**.productName = name;

}

**public** **void** setQuantity(**int** qty) {

**this**.quantity = qty;

}

**public** **void** setPrice(**double** price) {

**this**.price = price;

}

**public** String toString() {

**return** "ID: " + productId + ", Name: " + productName + ", Quantity: " + quantity + ", Price: " + price;

}

}

**class** Inventory {

**private** HashMap<String, Product> products;

**public** Inventory() {

products = **new** HashMap<>();

}

**public** **void** addProduct(Product p) {

products.put(p.getProductId(), p);

}

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

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

Product p = products.get(id);

p.setProductName(name);

p.setQuantity(qty);

p.setPrice(price);

}

}

**public** **void** deleteProduct(String id) {

products.remove(id);

}

**public** **void** displayAllProducts() {

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

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

}

}

}

**public** **class** Inventory\_Management\_System {

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

Inventory inventory = **new** Inventory();

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

**while** (**true**) {

System.***out***.println("\n1. Add Product\n2. Update Product\n3. Delete Product\n4. View All Products\n5. Exit");

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

sc.nextLine();

**if** (choice == 1) {

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

String id = sc.nextLine();

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));

} **else** **if** (choice == 2) {

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

String id = sc.nextLine();

System.***out***.print("Enter New 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();

inventory.updateProduct(id, name, qty, price);

} **else** **if** (choice == 3) {

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

String id = sc.nextLine();

inventory.deleteProduct(id);

} **else** **if** (choice == 4) {

inventory.displayAllProducts();

} **else** **if** (choice == 5) {

**break**;

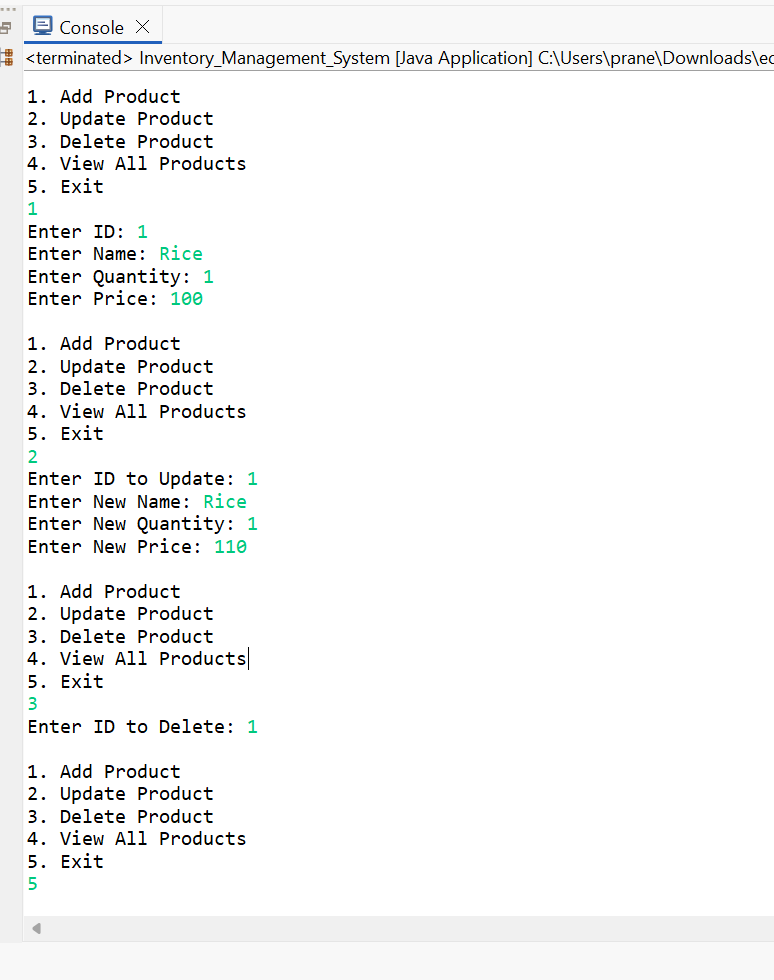
}

}

sc.close();

}

}

**Output:** 

**Exercise 2: E-commerce Platform Search Function**

**Scenario:**

You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

**Steps:**

1. **Understand Asymptotic Notation:**
   * Explain Big O notation and how it helps in analyzing algorithms.
   * Describe the best, average, and worst-case scenarios for search operations.
2. **Setup:**
   * Create a class **Product** with attributes for searching, such as **productId, productName**, and **category**.
3. **Implementation:**
   * Implement linear search and binary search algorithms.
   * Store products in an array for linear search and a sorted array for binary search.
4. **Analysis:**
   * Compare the time complexity of linear and binary search algorithms.
   * Discuss which algorithm is more suitable for your platform and why.

**Code:  
package** io;

**import** java.util.Arrays;

**import** java.util.Scanner;

**class** SearchProduct **implements** Comparable<SearchProduct> {

**private** String productId;

**private** String productName;

**private** String category;

**public** SearchProduct(String productId, String productName, String category) {

**this**.productId = productId;

**this**.productName = productName;

**this**.category = category;

}

**public** String getProductName() {

**return** productName;

}

**public** **int** compareTo(SearchProduct other) {

**return** **this**.productName.compareToIgnoreCase(other.productName);

}

**public** String toString() {

**return** "ID: " + productId + ", Name: " + productName + ", Category: " + category;

}

}

**public** **class** E\_commerce\_Platform {

**public** **static** **int** linearSearch(SearchProduct[] products, String targetName) {

**for** (**int** i = 0; i < products.length; i++) {

**if** (products[i].getProductName().equalsIgnoreCase(targetName)) {

**return** i;

}

}

**return** -1;

}

**public** **static** **int** binarySearch(SearchProduct[] products, String targetName) {

**int** left = 0;

**int** right = products.length - 1;

**while** (left <= right) {

**int** mid = (left + right) / 2;

**int** cmp = products[mid].getProductName().compareToIgnoreCase(targetName);

**if** (cmp == 0) **return** mid;

**else** **if** (cmp < 0) left = mid + 1;

**else** right = mid - 1;

}

**return** -1;

}

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

SearchProduct[] products = {

**new** SearchProduct("101", "Laptop", "Electronics"),

**new** SearchProduct("102", "Shoes", "Fashion"),

**new** SearchProduct("103", "Mobile", "Electronics"),

**new** SearchProduct("104", "Watch", "Accessories"),

**new** SearchProduct("105", "Tshirt", "Fashion")

};

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

System.***out***.print("Enter product name to search: ");

String name = sc.nextLine();

**int** indexLinear = *linearSearch*(products, name);

**if** (indexLinear != -1) {

System.***out***.println("Linear Search: Found -> " + products[indexLinear]);

} **else** {

System.***out***.println("Linear Search: Product not found");

}

Arrays.*sort*(products);

**int** indexBinary = *binarySearch*(products, name);

**if** (indexBinary != -1) {

System.***out***.println("Binary Search: Found -> " + products[indexBinary]);

} **else** {

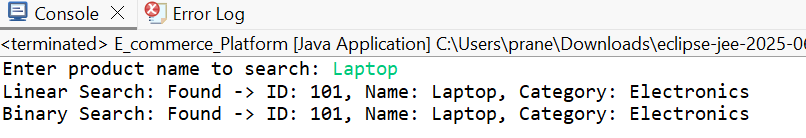
System.***out***.println("Binary Search: Product not found");

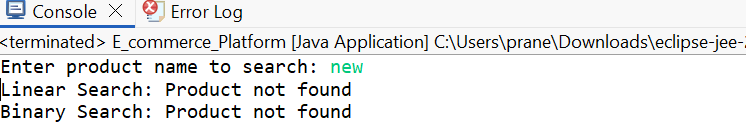
}

sc.close();

}

}

**Output:**



**Exercise 1: Implementing the Singleton Pattern**

**Scenario:**

You need to ensure that a logging utility class in your application has only one instance throughout the application lifecycle to ensure consistent logging.

**Steps:**

1. **Create a New Java Project:**
   * Create a new Java project named **SingletonPatternExample**.
2. **Define a Singleton Class:**
   * Create a class named Logger that has a private static instance of itself.
   * Ensure the constructor of Logger is private.
   * Provide a public static method to get the instance of the Logger class.
3. **Implement the Singleton Pattern:**
   * Write code to ensure that the Logger class follows the Singleton design pattern.
4. **Test the Singleton Implementation:**
   * Create a test class to verify that only one instance of Logger is created and used across the application.

**Code:  
package** patterns;

**public** **class** SingletonPattern {

// Singleton Logger class

**static** **class** Logger {

**private** **static** Logger *instance*;

**private** Logger() {

System.***out***.println("Logger instance created");

}

**public** **static** Logger getInstance() {

**if** (*instance* == **null**) {

*instance* = **new** Logger();

}

**return** *instance*;

}

**public** **void** log(String message) {

System.***out***.println("[LOG]: " + message);

}

}

// Main method to test the Singleton

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

Logger logger1 = Logger.*getInstance*();

logger1.log("First message");

Logger logger2 = Logger.*getInstance*();

logger2.log("Second message");

**if** (logger1 == logger2) {

System.***out***.println("Both logger instances are the same (Singleton verified)");

} **else** {

System.***out***.println("Different instances (Singleton violated)");

}

}

}

**Output:**