**EXERCISE 1-(E-COMMERCE PLATFORM SEARCH FUNCTION)**

import java.util.HashMap;

import java.util.Map;

public class InventoryManagementSystem {

public static 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 String getProductName() {

return productName;

}

public int getQuantity() {

return quantity;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

@Override

public String toString() {

return String.format("Product[id=%s, name=%s, quantity=%d, price=%.2f]",

productId, productName, quantity, price);

}

}

public static class Inventory {

private Map<String, Product> products;

public Inventory() {

products = new HashMap<>();

}

public void addProduct(Product product) {

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

}

public boolean updateProduct(String productId, Integer quantity, Double price) {

Product product = products.get(productId);

if (product == null) {

return false;

}

if (quantity != null) {

product.setQuantity(quantity);

}

if (price != null) {

product.setPrice(price);

}

return true;

}

public boolean deleteProduct(String productId) {

if (products.containsKey(productId)) {

products.remove(productId);

return true;

} else {

return false;

}

}

public Product getProduct(String productId) {

return products.get(productId);

}

public void printInventory() {

if (products.isEmpty()) {

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

} else {

System.out.println("Current Inventory:");

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

System.out.println(p);

}

}

}

}

public static void main(String[] args) {

Inventory inventory = new Inventory();

inventory.addProduct(new Product("P1001", "Laptop", 10, 999.99));

inventory.addProduct(new Product("P1002", "Smartphone", 25, 499.50));

inventory.addProduct(new Product("P1003", "Headphones", 50, 89.90));

inventory.printInventory();

System.out.println("\nUpdating product P1002...");

boolean updated = inventory.updateProduct("P1002", 30, 479.99);

if (updated) {

System.out.println("Updated product: " + inventory.getProduct("P1002"));

} else {

System.out.println("Product P1002 not found.");

}

System.out.println("\nDeleting product P1003...");

boolean deleted = inventory.deleteProduct("P1003");

if (deleted) {

System.out.println("Product P1003 deleted.");

**} else {**

System.out.println("Product P1003 not found.");

}

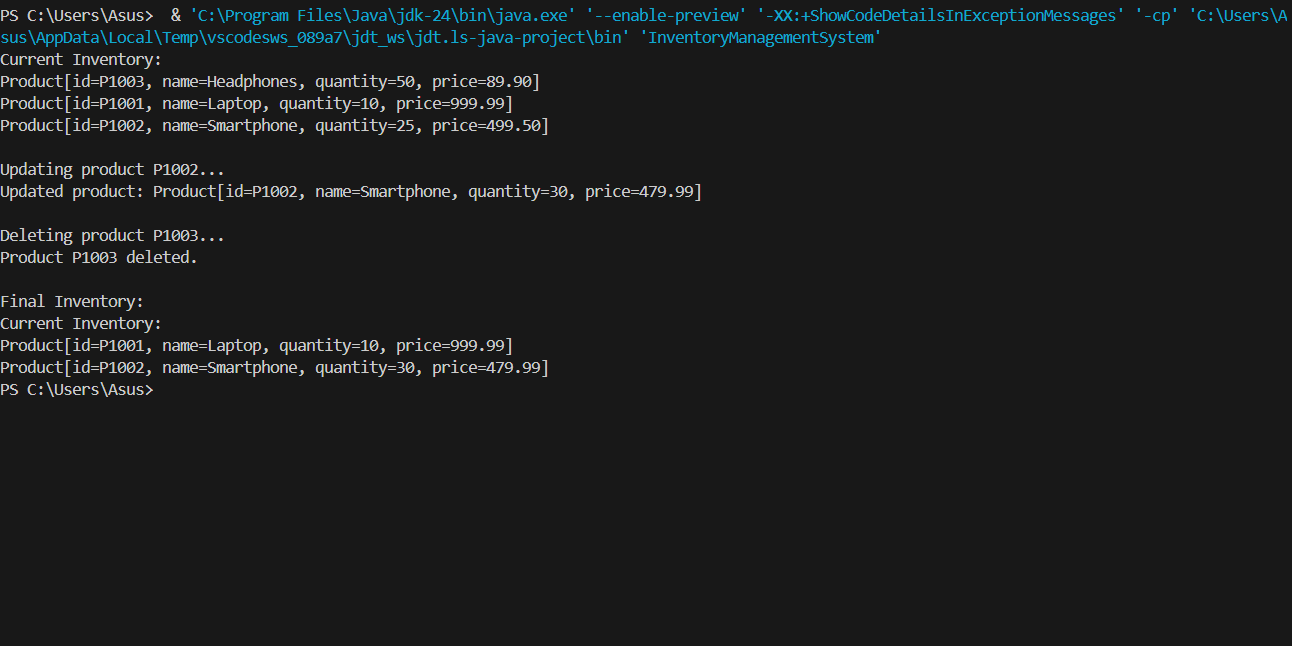
System.out.println("\nFinal Inventory:");

inventory.printInventory();

}

}

**OUTPUT:**



**EXERCISE 2-(FINANCIAL FORECASTING)**

import java.util.Arrays;

public class SearchFunctionality {

public static class Product implements Comparable<Product> {

private String productId;

private String productName;

private String category;

public Product(String productId, String productName, String category) {

this.productId = productId;

this.productName = productName;

this.category = category;

}

public String getProductId() {

return productId;

}

public String getProductName() {

return productName;

}

public String getCategory() {

return category;

}

@Override

public String toString() {

return String.format("Product[id=%s, name=%s, category=%s]", productId, productName, category);

}

@Override

public int compareTo(Product other) {

return this.productId.compareTo(other.productId);

}

}

public static Product linearSearch(Product[] products, String productId) {

for (Product product : products) {

if (product.getProductId().equals(productId)) {

return product;

}

}

return null;

}

public static Product binarySearch(Product[] products, String productId) {

int left = 0;

int right = products.length - 1;

while (left <= right) {

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

int cmp = products[mid].getProductId().compareTo(productId);

if (cmp == 0) {

return products[mid];

}

if (cmp < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

Product[] products = {

new Product("123", "Laptop", "Electronics"),

new Product("456", "Mouse", "Electronics"),

new Product("789", "Keyboard", "Electronics"),

new Product("101", "Smartphone", "Electronics")

};

Product foundProductLinear = linearSearch(products, "456");

System.out.println("Linear Search: " + (foundProductLinear != null ? foundProductLinear : "Product not found"));

Arrays.sort(products);

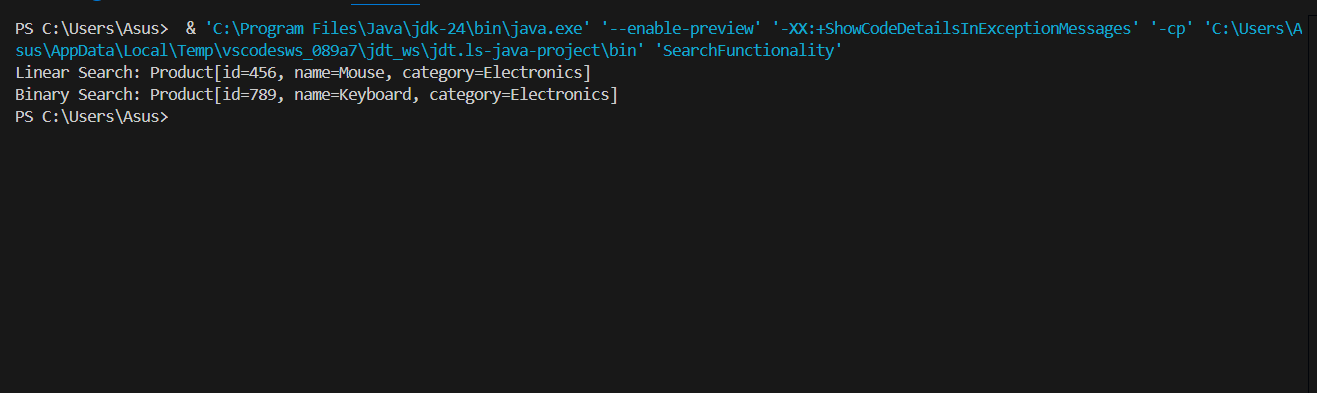
Product foundProductBinary = binarySearch(products, "789");

System.out.println("Binary Search: " + (foundProductBinary != null ? foundProductBinary : "Product not found"));

}

}

**OUTPUT:**



**EXERCISE 3**

import java.util.Arrays;

public class OrderSorting {

    public static class Order {

        private int orderId;

        private String customerName;

        private double totalPrice;

        public Order(int orderId, String customerName, double totalPrice) {

            this.orderId = orderId;

            this.customerName = customerName;

            this.totalPrice = totalPrice;

        }

        public int getOrderId() {

            return orderId;

        }

        public String getCustomerName() {

            return customerName;

        }

        public double getTotalPrice() {

            return totalPrice;

        }

        @Override

        public String toString() {

            return String.format("Order ID: %d, Customer: %s, Total Price: %.2f", orderId, customerName, totalPrice);

        }

    }

    public static void bubbleSort(Order[] orders) {

        int n = orders.length;

        for (int i = 0; i < n - 1; i++) {

            for (int j = 0; j < n - i - 1; j++) {

                if (orders[j].getTotalPrice() > orders[j + 1].getTotalPrice()) {

                    Order temp = orders[j];

                    orders[j] = orders[j + 1];

                    orders[j + 1] = temp;

                }

            }

        }

    }

    public static void quickSort(Order[] orders, int low, int high) {

        if (low < high) {

            int pi = partition(orders, low, high);

            quickSort(orders, low, pi - 1);

            quickSort(orders, pi + 1, high);

        }

    }

    private static int partition(Order[] orders, int low, int high) {

        double pivot = orders[high].getTotalPrice();

        int i = (low - 1);

        for (int j = low; j < high; j++) {

            if (orders[j].getTotalPrice() <= pivot) {

                i++;

                Order temp = orders[i];

                orders[i] = orders[j];

                orders[j] = temp;

            }

        }

        Order temp = orders[i + 1];

        orders[i + 1] = orders[high];

        orders[high] = temp;

        return i + 1;

    }

    public static void main(String[] args) {

        Order[] orders = {

            new Order(1, "Alice", 250.00),

            new Order(2, "Bob", 150.00),

            new Order(3, "Charlie", 300.00),

            new Order(4, "David", 200.00)

        };

        System.out.println("Original Orders:");

        System.out.println(Arrays.toString(orders));

        Order[] bubbleSortedOrders = Arrays.copyOf(orders, orders.length);

        bubbleSort(bubbleSortedOrders);

        System.out.println("\nBubble Sorted Orders:");

        System.out.println(Arrays.toString(bubbleSortedOrders));

        Order[] quickSortedOrders = Arrays.copyOf(orders, orders.length);

        quickSort(quickSortedOrders, 0, quickSortedOrders.length - 1);

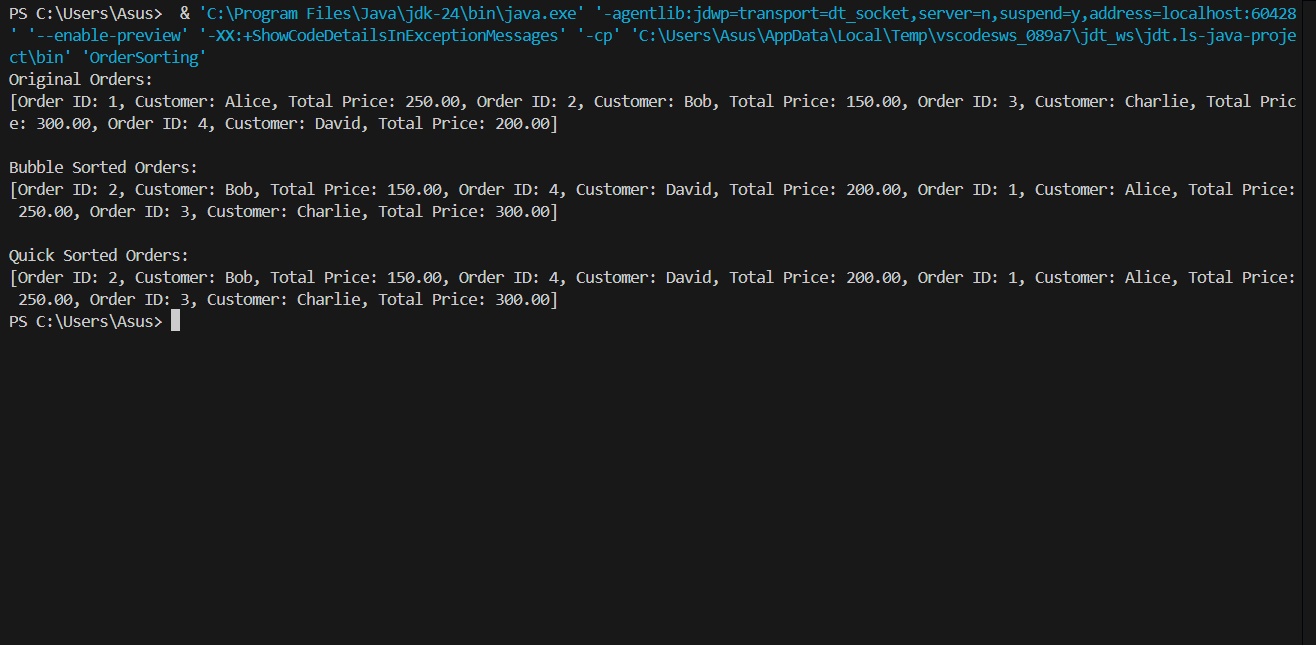
        System.out.println("\nQuick Sorted Orders:");

        System.out.println(Arrays.toString(quickSortedOrders));

    }

}

**OUTPUT:**



**EXERCISE 4**

class Employee {

    int employeeId;

    String name;

    String position;

    double salary;

    public Employee(int employeeId, String name, String position, double salary) {

        this.employeeId = employeeId;

        this.name = name;

        this.position = position;

        this.salary = salary;

    }

}

class EmployeeManagementSystem {

    private Employee[] employees;

    private int size;

    public EmployeeManagementSystem(int capacity) {

        employees = new Employee[capacity];

        size = 0;

    }

    public void addEmployee(Employee employee) {

        if (size < employees.length) {

            employees[size] = employee;

            size++;

        } else {

            System.out.println("Array is full. Cannot add more employees.");

        }

    }

    public Employee searchEmployee(int employeeId) {

        for (int i = 0; i < size; i++) {

            if (employees[i].employeeId == employeeId) {

                return employees[i];

            }

        }

        return null;

    }

    public void traverseEmployees() {

        for (int i = 0; i < size; i++) {

       System.out.println(employees[i].employeeId + " " + employees[i].name + " " + employees[i].position + " " + employees[i].salary);

        }

    }

    public void deleteEmployee(int employeeId) {

        for (int i = 0; i < size; i++) {

            if (employees[i].employeeId == employeeId) {

                for (int j = i; j < size - 1; j++) {

                    employees[j] = employees[j + 1];

                }

                employees[size - 1] = null;

                size--;

                return;

            }

        }

        System.out.println("Employee not found.");

    }

    public static void main(String[] args) {

        EmployeeManagementSystem ems = new EmployeeManagementSystem(5);

        ems.addEmployee(new Employee(1, "Alice", "Manager", 50000));

        ems.addEmployee(new Employee(2, "Bob", "Developer", 60000));

        ems.addEmployee(new Employee(3, "Charlie", "Analyst", 40000));

        System.out.println("All employees:");

        ems.traverseEmployees();

        System.out.println("\nSearching for employee with ID 2:");

        Employee employee = ems.searchEmployee(2);

        if (employee != null) {

            System.out.println(employee.name);

        }

        System.out.println("\nDeleting employee with ID 2:");

        ems.deleteEmployee(2);

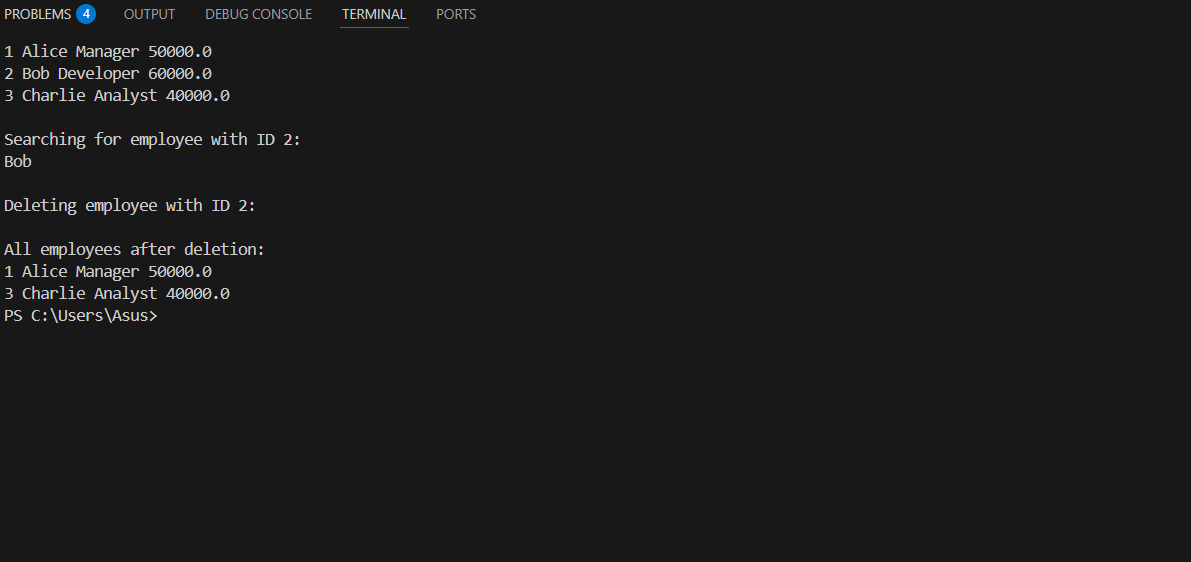
        System.out.println("\nAll employees after deletion:");

        ems.traverseEmployees();

    }

}

**OUTPUT:**



**EXERCISE 5**

public class TaskManager {

    public static class Task {

        private String taskId;

        private String taskName;

        private String status;

        public Task(String taskId, String taskName, String status) {

            this.taskId = taskId;

            this.taskName = taskName;

            this.status = status;

        }

        public String getTaskId() {

            return taskId;

        }

        public String getTaskName() {

            return taskName;

        }

        public String getStatus() {

            return status;

        }

        @Override

        public String toString() {

            return String.format("Task[id=%s, name=%s, status=%s]", taskId, taskName, status);

        }

    }

    private class Node {

        Task task;

        Node next;

        Node(Task task) {

            this.task = task;

            this.next = null;

        }

    }

    private Node head;

    public void addTask(Task task) {

        Node newNode = new Node(task);

        if (head == null) {

            head = newNode;

        } else {

            Node current = head;

            while (current.next != null) {

                current = current.next;

            }

            current.next = newNode;

        }

    }

    public Task searchTask(String taskId) {

        Node current = head;

        while (current != null) {

            if (current.task.getTaskId().equals(taskId)) {

                return current.task;

            }

            current = current.next;

        }

        return null;

    }

    public void traverseTasks() {

        Node current = head;

        while (current != null) {

            System.out.println(current.task);

            current = current.next;

        }

    }

    public boolean deleteTask(String taskId) {

        if (head == null) {

            return false;

        }

        if (head.task.getTaskId().equals(taskId)) {

            head = head.next;

            return true;

        }

        Node current = head;

        while (current.next != null) {

            if (current.next.task.getTaskId().equals(taskId)) {

                current.next = current.next.next;

                return true;

            }

            current = current.next;

        }

        return false;

    }

    public static void main(String[] args) {

        TaskManager taskManager = new TaskManager();

        taskManager.addTask(new Task("T1001", "Design UI", "Pending"));

        taskManager.addTask(new Task("T1002", "Implement Backend", "Pending"));

        taskManager.addTask(new Task("T1003", "Test Application", "Pending"));

        System.out.println("Task List:");

        taskManager.traverseTasks();

        System.out.println("\nSearching for task T1002:");

        Task foundTask = taskManager.searchTask("T1002");

        System.out.println(foundTask != null ? foundTask : "Task not found.");

        System.out.println("\nDeleting task T1001:");

        boolean deleted = taskManager.deleteTask("T1001");

        System.out.println(deleted ? "Task T1001 deleted." : "Task not found.");

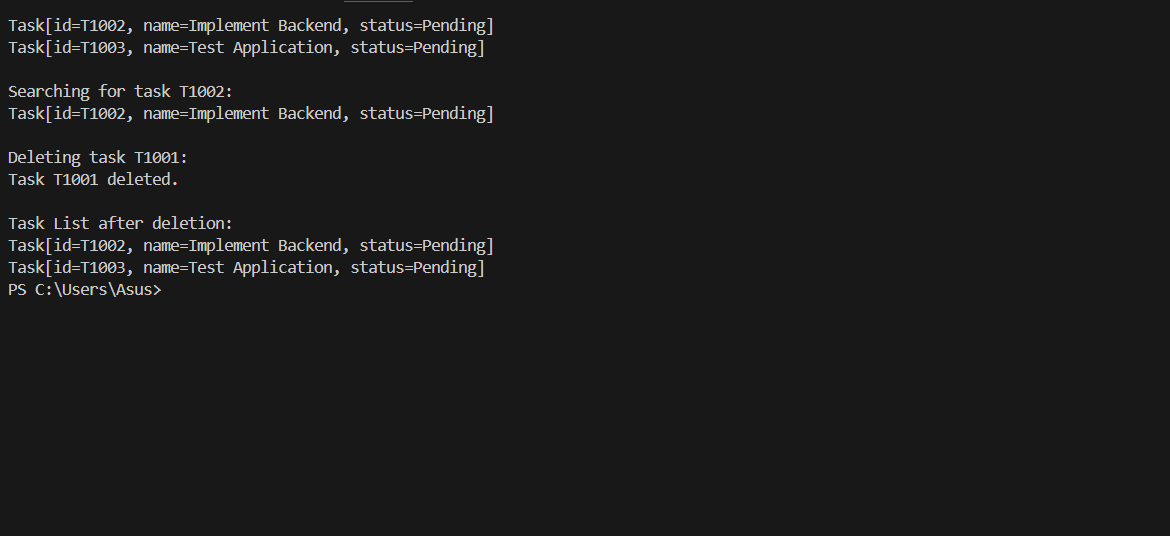
        System.out.println("\nTask List after deletion:");

        taskManager.traverseTasks();

    }

}

**OUTPUT:**



**EXERCISE 6**

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

public class LibrarySearch {

    public static class Book {

        private int bookId;

        private String title;

        private String author;

        public Book(int bookId, String title, String author) {

            this.bookId = bookId;

            this.title = title;

            this.author = author;

        }

        public int getBookId() {

            return bookId;

        }

        public String getTitle() {

            return title;

        }

        public String getAuthor() {

            return author;

        }

        @Override

        public String toString() {

            return String.format("Book ID: %d, Title: \"%s\", Author: \"%s\"", bookId, title, author);

        }

    }

    public static List<Book> linearSearchByTitle(List<Book> books, String title) {

        List<Book> results = new ArrayList<>();

        for (Book book : books) {

            if (book.getTitle().equalsIgnoreCase(title)) {

                results.add(book);

            }

        }

        return results;

    }

    public static Book binarySearchByTitle(List<Book> books, String title) {

        int low = 0;

        int high = books.size() - 1;

        title = title.toLowerCase();

        while (low <= high) {

            int mid = (low + high) >>> 1;

            String midTitle = books.get(mid).getTitle().toLowerCase();

            int cmp = midTitle.compareTo(title);

            if (cmp == 0) {

                return books.get(mid);

            } else if (cmp < 0) {

                low = mid + 1;

            } else {

                high = mid - 1;

            }

        }

        return null;

    }

    public static void main(String[] args) {

        List<Book> books = new ArrayList<>();

        books.add(new Book(101, "Effective Java", "Joshua Bloch"));

        books.add(new Book(102, "Clean Code", "Robert C. Martin"));

        books.add(new Book(103, "Java Concurrency in Practice", "Brian Goetz"));

        books.add(new Book(104, "Design Patterns", "Erich Gamma"));

        books.add(new Book(105, "Refactoring", "Martin Fowler"));

        books.add(new Book(106, "Clean Architecture", "Robert C. Martin"));

        String searchTitle = "Clean Code";

        System.out.println("=== Linear Search ===");

        List<Book> linearResults = linearSearchByTitle(books, searchTitle);

        if (linearResults.isEmpty()) {

            System.out.println("No books found with title \"" + searchTitle + "\"");

        } else {

            for (Book book : linearResults) {

                System.out.println(book);

            }

        }

       Collections.sort(books, Comparator.comparing(Book::getTitle, String.CASE\_INSENSITIVE\_ORDER));

        System.out.println("\n=== Binary Search ===");

        Book binaryResult = binarySearchByTitle(books, searchTitle);

        if (binaryResult == null) {

            System.out.println("No book found with title \"" + searchTitle + "\"");

        } else {

            System.out.println(binaryResult);

        }

        System.out.println("\n=== Search Analysis ===");

        System.out.println("Linear Search time complexity: O(n) - checks every book until a match is found or list is exhausted.");

        System.out.println("Binary Search time complexity: O(log n) - repeatedly divides the sorted list to find the book quickly.");

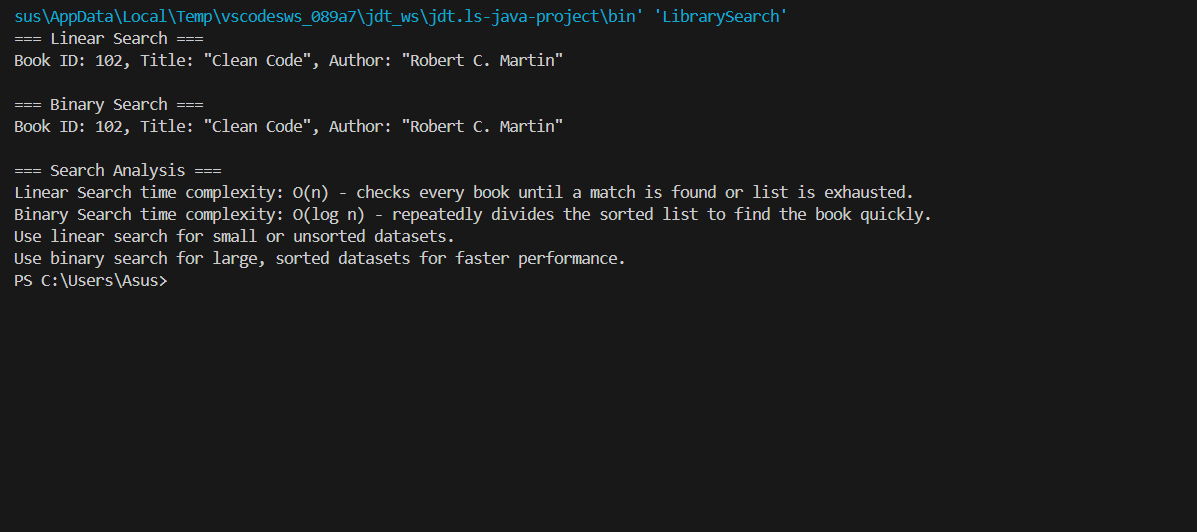
        System.out.println("Use linear search for small or unsorted datasets.");

        System.out.println("Use binary search for large, sorted datasets for faster performance.");

    }

}

**OUTPUT:**



**EXERCISE 7**

import java.util.HashMap;

import java.util.Map;

public class FinancialForecasting {

    private static Map<Integer, Double> memo = new HashMap<>();

    public static double calculateFutureValue(double presentValue, double growthRate, int periods) {

        if (memo.containsKey(periods)) {

            return memo.get(periods);

        }

        if (periods == 0) {

            return presentValue;

        }

        double futureValue = calculateFutureValue(presentValue \* (1 + growthRate), growthRate, periods - 1);

        memo.put(periods, futureValue);

        return futureValue;

    }

    public static void main(String[] args) {

        double presentValue = 1000.0;

        double growthRate = 0.05;

        int periods = 10;

        double futureValue = calculateFutureValue(presentValue, growthRate, periods);

        System.out.printf("Future Value after %d years: %.2f%n", periods, futureValue);

    }

}

**OUTPUT:**

