**WEEK 1 Hands On**

**Algorithms and Data Structures**

**Exercise 1: Inventory Management System**

**CODE**

**Product.java**

package DSA.InventoryManagementSystem;

public 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 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 "ID: " + productId + ", Name: " + productName + ", Quantity: " + quantity + ", Price: Rs." + price;

    }

}

**InventoryManager.java**

package DSA.InventoryManagementSystem;

import java.util.HashMap;

import java.util.Map;

public class InventoryManager {

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

    // Add

    public void addProduct(Product product) {

        if (inventory.containsKey(product.getProductId())) {

            System.out.println("Product already exists.");

        } else {

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

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

        }

    }

    // Update

    public void updateProduct(String productId, String name, int quantity, double price) {

        if (inventory.containsKey(productId)) {

            Product product = inventory.get(productId);

            product.setProductName(name);

            product.setQuantity(quantity);

            product.setPrice(price);

            System.out.println("Product updated.");

        } else {

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

        }

    }

    // Delete

    public void deleteProduct(String productId) {

        if (inventory.remove(productId) != null) {

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

        } else {

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

        }

    }

    // View all products

    public void viewAllProducts() {

        if (inventory.isEmpty()) {

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

        } else {

            for (Product product : inventory.values()) {

                System.out.println(product);

            }

        }

    }

}

**InventoryTest.java**

package DSA.InventoryManagementSystem;

public class InventoryTest {

    public static void main(String[] args) {

        InventoryManager manager = new InventoryManager();

        Product p1 = new Product("P001", "Laptop", 10, 59999.99);

        Product p2 = new Product("P002", "Mouse", 50, 499.99);

        Product p3 = new Product("P003", "Keyboard", 30, 999.99);

        manager.addProduct(p1);

        manager.addProduct(p2);

        manager.addProduct(p3);

        System.out.println("\nInventory:");

        manager.viewAllProducts();

        System.out.println("\nUpdating product P002...");

        manager.updateProduct("P002", "Wireless Mouse", 60, 599.99);

        System.out.println("\nInventory after update:");

        manager.viewAllProducts();

        System.out.println("\nDeleting product P001...");

        manager.deleteProduct("P001");

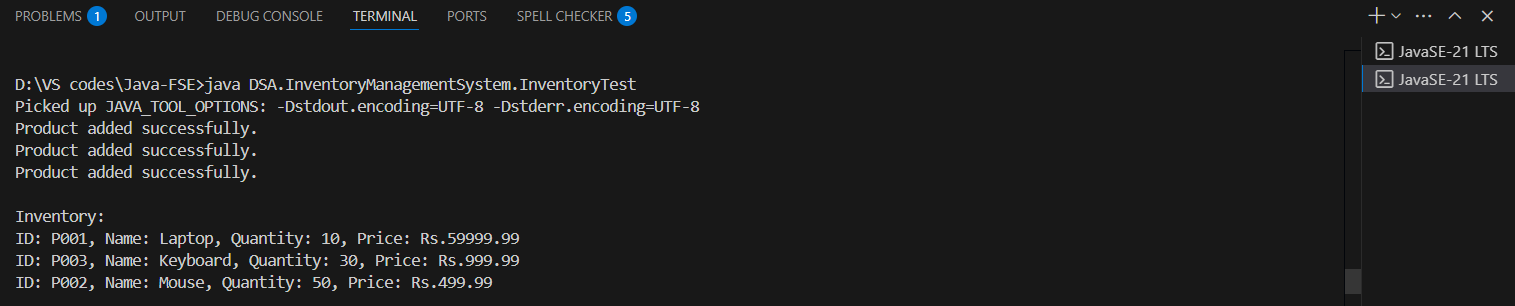
        System.out.println("\nFinal Inventory:");

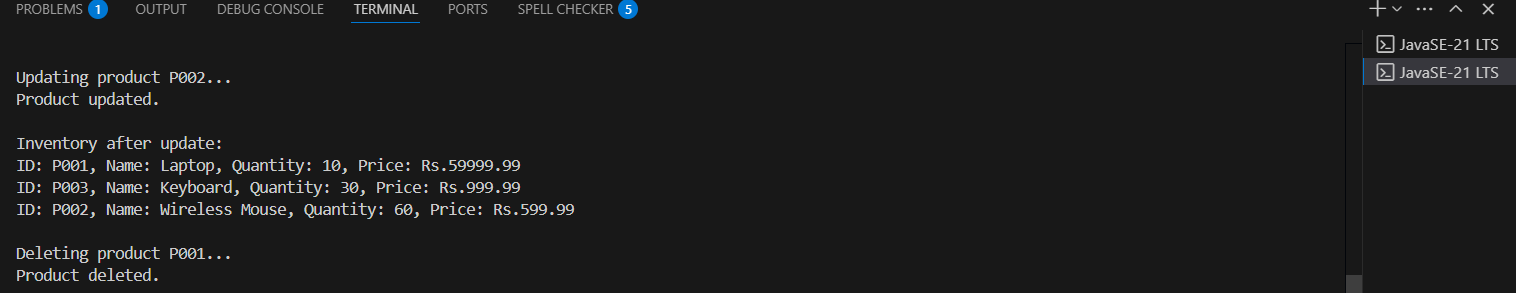
        manager.viewAllProducts();

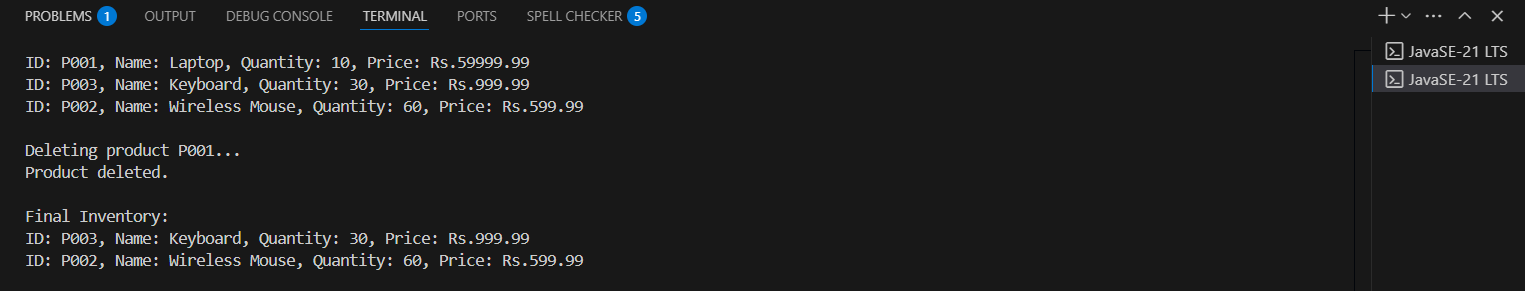
    }

}

**OUTPUT**







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

**CODE**

**Product.java**

package DSA.EcommerceSearchExample;

public 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 getProductName() {

        return productName;

    }

    @Override

    public int compareTo(Product other) {

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

    }

    @Override

    public String toString() {

        return "[" + productId + "] " + productName + " (" + category + ")";

    }

}

**SearchAlgorithms.java**

package DSA.EcommerceSearchExample;

public class SearchAlgorithms {

    // Linear Search

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

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

            if (products[i].getProductName().equalsIgnoreCase(name)) {

                return i;

            }

        }

        return -1;

    }

    // Binary Search

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

        int low = 0, high = products.length - 1;

        while (low <= high) {

            int mid = (low + high) / 2;

            int result = products[mid].getProductName().compareToIgnoreCase(name);

            if (result == 0)

                return mid;

            else if (result < 0)

                low = mid + 1;

            else

                high = mid - 1;

        }

        return -1;

    }

}

**SearchTest.java**

package DSA.EcommerceSearchExample;

import java.util.Arrays;

public class SearchTest {

    public static void main(String[] args) {

        Product[] products = {

            new Product("P001", "Laptop", "Electronics"),

            new Product("P002", "Shoes", "Fashion"),

            new Product("P003", "Smartphone", "Electronics"),

            new Product("P004", "Watch", "Accessories"),

            new Product("P005", "Backpack", "Bags")

        };

        // Linear Search

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

        int index1 = SearchAlgorithms.linearSearch(products, "Watch");

        System.out.println(index1 != -1 ? "Found: " + products[index1] : "Not Found");

        // Binary Search

        Arrays.sort(products);

        System.out.println("\nBinary Search (sorted by productName):");

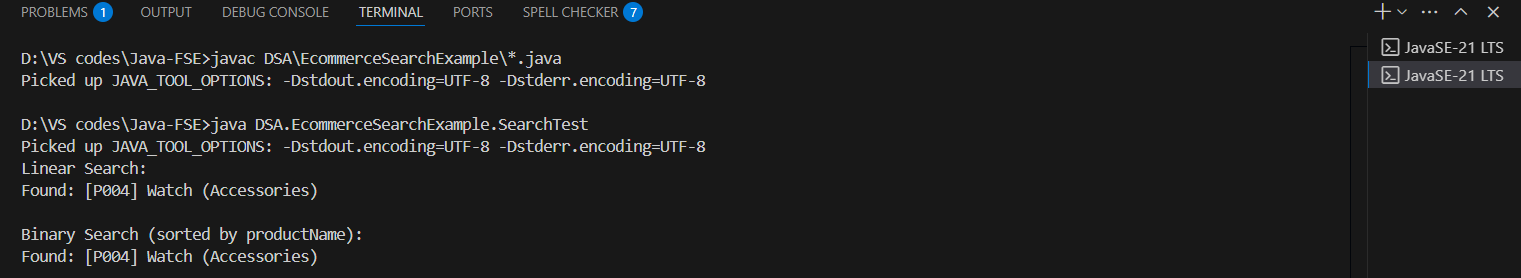
        int index2 = SearchAlgorithms.binarySearch(products, "Watch");

        System.out.println(index2 != -1 ? "Found: " + products[index2] : "Not Found");

    }

}

**OUTPUT**



**Exercise 3: Sorting Customer Orders**

**CODE**

**Order.java**

package DSA.Customer;

public class Order {

    private String orderId;

    private String customerName;

    private double totalPrice;

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

        this.orderId = orderId;

        this.customerName = customerName;

        this.totalPrice = totalPrice;

    }

    public double getTotalPrice() {

        return totalPrice;

    }

    @Override

    public String toString() {

        return "[" + orderId + "] " + customerName + " - ₹" + totalPrice;

    }

}

**OrderTest.java**

package DSA.Customer;

public class OrderTest {

    public static void main(String[] args) {

        Order[] orders = {

            new Order("O001", "Alice", 2500),

            new Order("O002", "Bob", 1500),

            new Order("O003", "Charlie", 8000),

            new Order("O004", "Diana", 5000),

            new Order("O005", "Ethan", 3000)

        };

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

        for (Order o : orders) System.out.println(o);

        // Test Bubble Sort

        System.out.println("\nSorted by Bubble Sort (Descending):");

        SortAlgorithms.bubbleSort(orders);

        for (Order o : orders) System.out.println(o);

        // Reset and Test Quick Sort

        Order[] orders2 = {

            new Order("O001", "Lingoth", 2500),

            new Order("O002", "Bobby", 1500),

            new Order("O003", "Charlie", 8000),

            new Order("O004", "Diana", 5000),

            new Order("O005", "Ethan", 3000)

        };

        System.out.println("\nSorted by Quick Sort (Descending):");

        SortAlgorithms.quickSort(orders2, 0, orders2.length - 1);

        for (Order o : orders2) System.out.println(o);

    }

}

**SortAlgorithms.java**

package DSA.Customer;

public class SortAlgorithms {

    public static void bubbleSort(Order[] orders) {

        int n = orders.length;

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

            boolean swapped = false;

            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;

                    swapped = true;

                }

            }

            if (!swapped) break;

        }

    }

        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) {  // descending

                i++;

                Order temp = orders[i];

                orders[i] = orders[j];

                orders[j] = temp;

            }

        }

        Order temp = orders[i + 1];

        orders[i + 1] = orders[high];

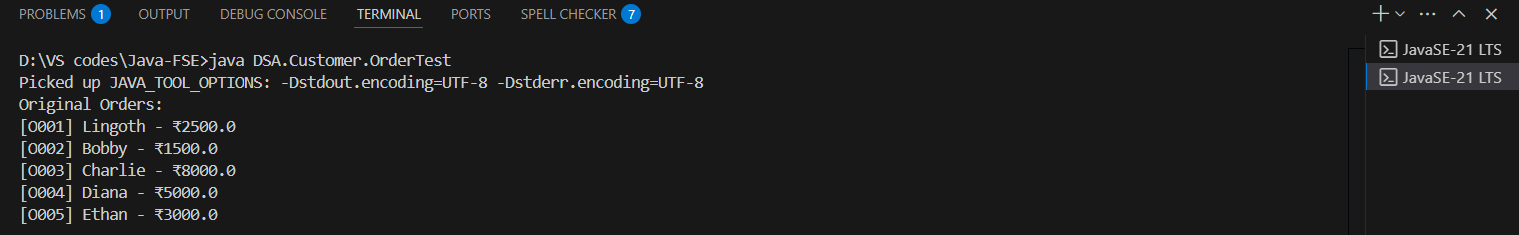
        orders[high] = temp;

        return i + 1;

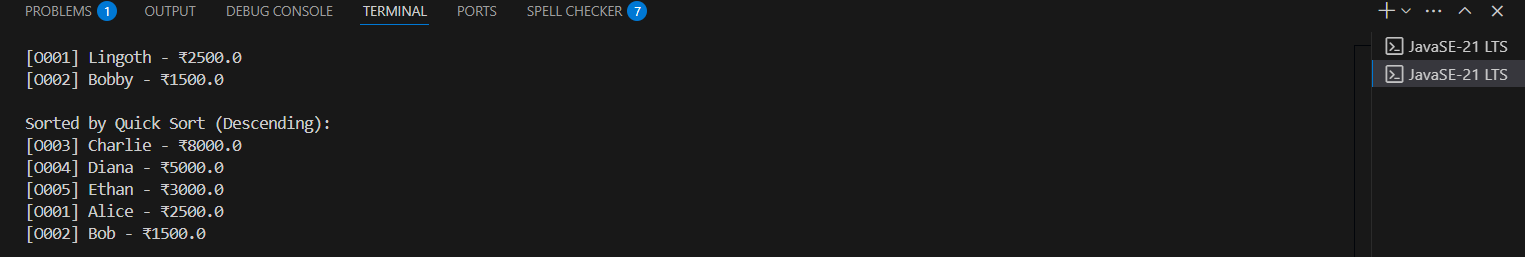
    }

}

**OUTPUT**







**Exercise 4: Employee Management System**

**CODE**

**Employee.java**

package DSA.EmployeeManagement;

public class Employee {

    private int employeeId;

    private String name;

    private String position;

    private double salary;

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

        this.employeeId = employeeId;

        this.name = name;

        this.position = position;

        this.salary = salary;

    }

    public int getEmployeeId() {

        return employeeId;

    }

    @Override

    public String toString() {

        return "ID: " + employeeId + ", Name: " + name + ", Position: " + position + ", Salary: ₹" + salary;

    }

}

**EmployeeManager.java**

package DSA.EmployeeManagement;

public class EmployeeManager {

    private Employee[] employees;

    private int count;

    public EmployeeManager(int capacity) {

        employees = new Employee[capacity];

        count = 0;

    }

    // Add employee

    public void addEmployee(Employee emp) {

        if (count >= employees.length) {

            System.out.println("Employee list is full!");

            return;

        }

        employees[count++] = emp;

        System.out.println("Employee added.");

    }

    // Search by ID

    public Employee searchEmployee(int id) {

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

            if (employees[i].getEmployeeId() == id) {

                return employees[i];

            }

        }

        return null;

    }

    // Traverse and print all employees

    public void listEmployees() {

        if (count == 0) {

            System.out.println("No employees to display.");

            return;

        }

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

            System.out.println(employees[i]);

        }

    }

    // Delete by ID

    public void deleteEmployee(int id) {

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

            if (employees[i].getEmployeeId() == id) {

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

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

                }

                employees[--count] = null;

                System.out.println("Employee deleted.");

                return;

            }

        }

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

    }

}

**EmployeeTest.java**

package DSA.EmployeeManagement;

public class EmployeeTest {

    public static void main(String[] args) {

        EmployeeManager manager = new EmployeeManager(5);

        // Add employees

        manager.addEmployee(new Employee(101, "Lingoth", "Developer", 50000));

        manager.addEmployee(new Employee(102, "Bobby", "Manager", 65000));

        manager.addEmployee(new Employee(103, "Charlie", "Designer", 48000));

        System.out.println("\nAll Employees:");

        manager.listEmployees();

        System.out.println("\nSearching for Employee ID 102:");

        Employee emp = manager.searchEmployee(102);

        System.out.println(emp != null ? emp : "Employee not found.");

        System.out.println("\nDeleting Employee ID 102...");

        manager.deleteEmployee(102);

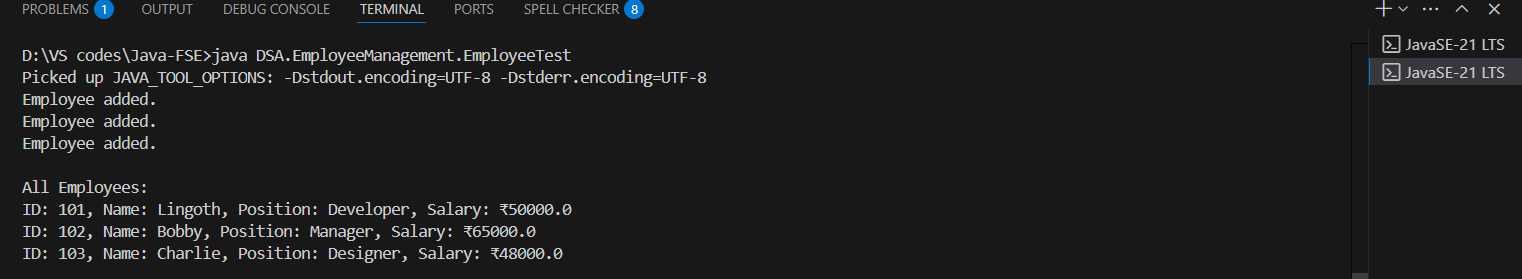
        System.out.println("\nEmployees after Deletion:");

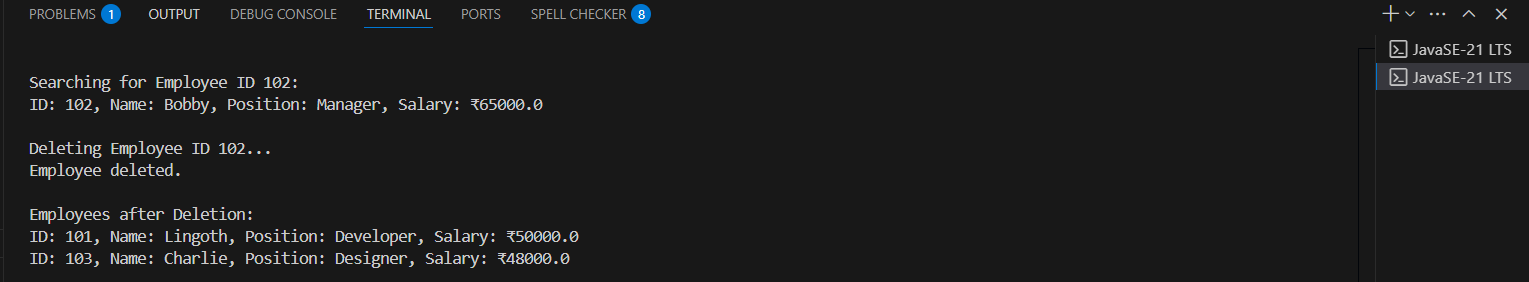
        manager.listEmployees();

    }

}

**OUTPUT**





**Exercise 5: Task Management System**

**CODE**

**Task.java**

package DSA.TaskManagement;

public class Task {

    private int taskId;

    private String taskName;

    private String status;

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

        this.taskId = taskId;

        this.taskName = taskName;

        this.status = status;

    }

    public int getTaskId() {

        return taskId;

    }

    @Override

    public String toString() {

        return "[ID: " + taskId + ", Name: " + taskName + ", Status: " + status + "]";

    }

}

**TaskManager.java**

package DSA.TaskManagement;

public class TaskManager {

    // Node class

    private static class Node {

        Task task;

        Node next;

        Node(Task task) {

            this.task = task;

            this.next = null;

        }

    }

    private Node head = null;

    // Add task at the end

    public void addTask(Task task) {

        Node newNode = new Node(task);

        if (head == null) {

            head = newNode;

        } else {

            Node curr = head;

            while (curr.next != null) {

                curr = curr.next;

            }

            curr.next = newNode;

        }

        System.out.println("Task added: " + task);

    }

    // Search task by ID

    public Task searchTask(int taskId) {

        Node curr = head;

        while (curr != null) {

            if (curr.task.getTaskId() == taskId) {

                return curr.task;

            }

            curr = curr.next;

        }

        return null;

    }

    // Delete task by ID

    public void deleteTask(int taskId) {

        if (head == null) {

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

            return;

        }

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

            head = head.next;

            System.out.println("Task deleted.");

            return;

        }

        Node curr = head;

        while (curr.next != null && curr.next.task.getTaskId() != taskId) {

            curr = curr.next;

        }

        if (curr.next == null) {

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

        } else {

            curr.next = curr.next.next;

            System.out.println("Task deleted.");

        }

    }

    // Traverse and display all tasks

    public void listTasks() {

        if (head == null) {

            System.out.println("No tasks available.");

            return;

        }

        Node curr = head;

        while (curr != null) {

            System.out.println(curr.task);

            curr = curr.next;

        }

    }

}

**TaskTest.java**

package DSA.TaskManagement;

public class TaskTest {

    public static void main(String[] args) {

        TaskManager manager = new TaskManager();

        manager.addTask(new Task(1, "Design UI", "Pending"));

        manager.addTask(new Task(2, "Develop Backend", "In Progress"));

        manager.addTask(new Task(3, "Write Tests", "Pending"));

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

        manager.listTasks();

        System.out.println("\nSearching Task ID 2:");

        Task t = manager.searchTask(2);

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

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

        manager.deleteTask(2);

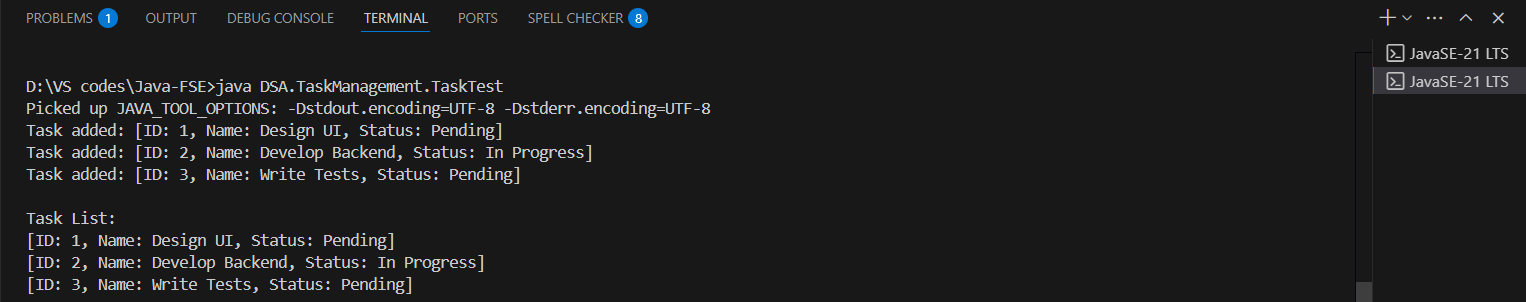
        System.out.println("\nTask List After Deletion:");

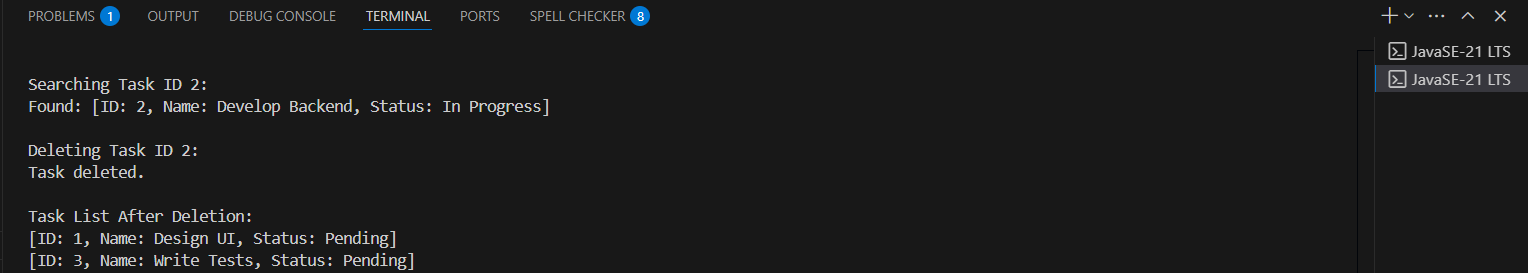
        manager.listTasks();

    }

}

**OUTPUT**





**Exercise 6: Library Management System**

**CODE**

**Book.java**

package DSA.LibraryManagement;

public class Book implements Comparable<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 String getTitle() {

        return title;

    }

    @Override

    public int compareTo(Book other) {

        return this.title.compareToIgnoreCase(other.title);

    }

    @Override

    public String toString() {

        return "[" + bookId + "] " + title + " by " + author;

    }

}

**SearchLibrary.java**

package DSA.LibraryManagement;

public class SearchLibrary {

    // Linear Search

    public static Book linearSearch(Book[] books, String targetTitle) {

        for (Book book : books) {

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

                return book;

            }

        }

        return null;

    }

    // Binary Search

    public static Book binarySearch(Book[] books, String targetTitle) {

        int low = 0;

        int high = books.length - 1;

        while (low <= high) {

            int mid = (low + high) / 2;

            int cmp = books[mid].getTitle().compareToIgnoreCase(targetTitle);

            if (cmp == 0) {

                return books[mid];

            } else if (cmp < 0) {

                low = mid + 1;

            } else {

                high = mid - 1;

            }

        }

        return null;

    }

}

**LibraryTest.java**

package DSA.LibraryManagement;

import java.util.Arrays;

public class LibraryTest {

    public static void main(String[] args) {

        Book[] books = {

            new Book(101, "The Hobbit", "J.R.R. Tolkien"),

            new Book(102, "Harry Potter", "J.K. Rowling"),

            new Book(103, "A Tale of Two Cities", "Charles Dickens"),

            new Book(104, "Pride and Prejudice", "Jane Austen"),

            new Book(105, "The Alchemist", "Paulo Coelho")

        };

        // Linear Search

        System.out.println("Linear Search for 'Pride and Prejudice':");

        Book result1 = SearchLibrary.linearSearch(books, "Pride and Prejudice");

        System.out.println(result1 != null ? "Found: " + result1 : "Not found.");

        // Sort for Binary Search

        Arrays.sort(books);

        // Binary Search

        System.out.println("\nBinary Search for 'Pride and Prejudice':");

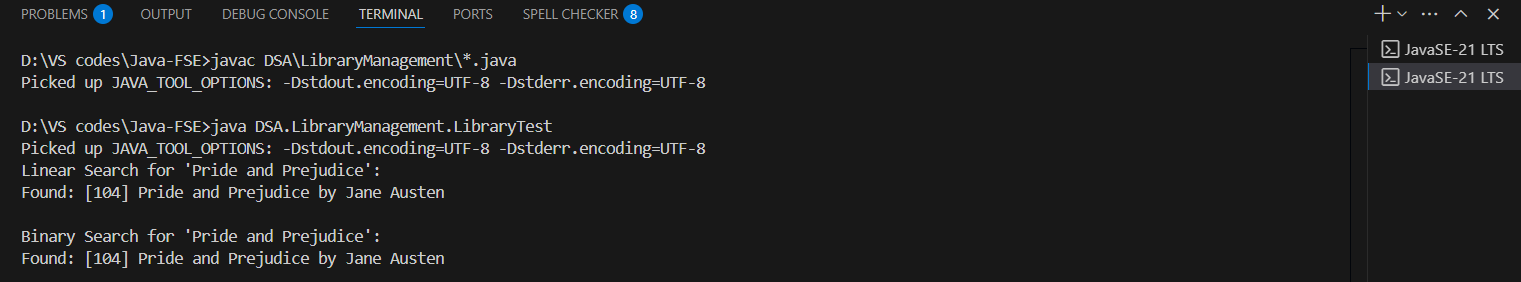
        Book result2 = SearchLibrary.binarySearch(books, "Pride and Prejudice");

        System.out.println(result2 != null ? "Found: " + result2 : "Not found.");

    }

}

**OUTPUT**



**Exercise 7: Financial Forecasting**

**CODE**

**FinancialForecast.java**

package DSA.FinancialForecasting;

public class FinancialForecast {

    // Recursive method

    public static double forecastRecursive(double presentValue, double growthRate, int years) {

        if (years == 0) {

            return presentValue;

        }

        return forecastRecursive(presentValue, growthRate, years - 1) \* (1 + growthRate);

    }

    public static double forecastMemo(double presentValue, double growthRate, int years, double[] memo) {

        if (years == 0) {

            return presentValue;

        }

        if (memo[years] != 0) {

            return memo[years];

        }

        memo[years] = forecastMemo(presentValue, growthRate, years - 1, memo) \* (1 + growthRate);

        return memo[years];

    }

    // Iterative method

    public static double forecastIterative(double presentValue, double growthRate, int years) {

        double result = presentValue;

        for (int i = 1; i <= years; i++) {

            result \*= (1 + growthRate);

        }

        return result;

    }

}

**ForecastTest.java**

package DSA.FinancialForecasting;

public class ForecastTest {

    public static void main(String[] args) {

        double presentValue = 10000;

        double growthRate = 0.10;

        int years = 5;

        System.out.println("Recursive Forecast:");

        double result1 = FinancialForecast.forecastRecursive(presentValue, growthRate, years);

        System.out.printf("Year %d forecast: ₹%.2f\n", years, result1);

        System.out.println("\nOptimized Recursive Forecast with Memoization:");

        double[] memo = new double[years + 1];

        double result2 = FinancialForecast.forecastMemo(presentValue, growthRate, years, memo);

        System.out.printf("Year %d forecast: ₹%.2f\n", years, result2);

        System.out.println("\nIterative Forecast:");

        double result3 = FinancialForecast.forecastIterative(presentValue, growthRate, years);

        System.out.printf("Year %d forecast: ₹%.2f\n", years, result3);

    }

}

**OUTPUT**

