DATA STRUCTURE AND ALGORITHM:

Exercise 1: Inventory Management System

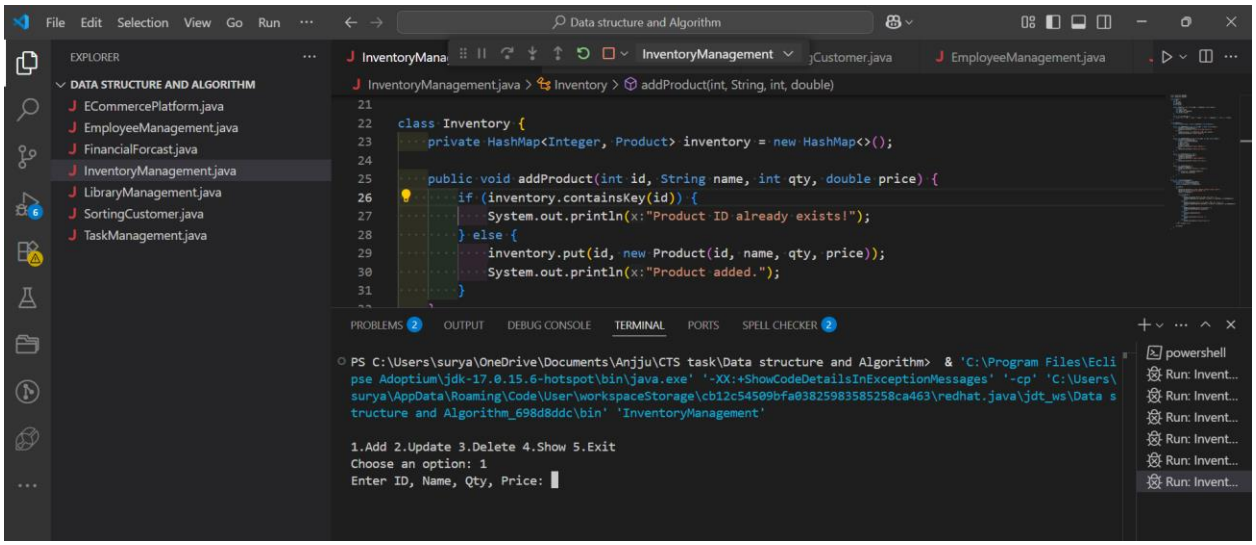
```
import java.util.HashMap;
import java.util.Scanner;

class Product {
    int id;
    String name;
    int quantity;
    double price;

    public Product(int id, String name, int quantity, double price) {
        this.id = id;
        this.name = name;
        this.quantity = quantity;
        this.price = price;
    }

    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Qty: " + quantity + ", Price: ₹" + price;
    }
}

class Inventory {
    private HashMap<Integer, Product> inventory = new HashMap<>();

    public void addProduct(int id, String name, int qty, double price) {
        if (inventory.containsKey(id)) {
            System.out.println("Product ID already exists!");
        } else {
            inventory.put(id, new Product(id, name, qty, price));
            System.out.println("Product added.");
        }
```

```
    }

    public void updateProduct(int id, String name, int qty, double price) {
        if (inventory.containsKey(id)) {
            Product p = inventory.get(id);
            p.name = name;
            p.quantity = qty;
            p.price = price;
            System.out.println("Product updated.");
        } else {
            System.out.println("Product not found.");
        }
    }

    public void deleteProduct(int id) {
        if (inventory.containsKey(id)) {
            inventory.remove(id);
            System.out.println("Product deleted.");
        } else {
            System.out.println("Product not found.");
        }
    }

    public void showInventory() {
        if (inventory.isEmpty()) {
            System.out.println("Inventory is empty.");
        } else {
            for (Product p : inventory.values()) {
                System.out.println(p);
            }
        }
    }
}

public class InventoryManagement {
```

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    Inventory inventory = new Inventory();

    int choice;
    do {
        System.out.println("\n1.Add 2.Update 3.Delete 4.Show 5.Exit");
        System.out.print("Choose an option: ");
        choice = sc.nextInt();

        switch (choice) {
            case 1:
                System.out.print("Enter ID, Name, Qty, Price: ");
                inventory.addProduct(sc.nextInt(), sc.next(), sc.nextInt(), sc.nextDouble());
                break;
            case 2:
                System.out.print("Enter ID to update, Name, Qty, Price: ");
                inventory.updateProduct(sc.nextInt(), sc.next(), sc.nextInt(), sc.nextDouble());
                break;
            case 3:
                System.out.print("Enter ID to delete: ");
                inventory.deleteProduct(sc.nextInt());
                break;
            case 4:
                inventory.showInventory();
                break;
            case 5:
                System.out.println("Exiting...");
                break;
            default:
                System.out.println("Invalid choice.");
        }
    } while (choice != 5);

    sc.close();
```

```
    }
}
```

0utput:



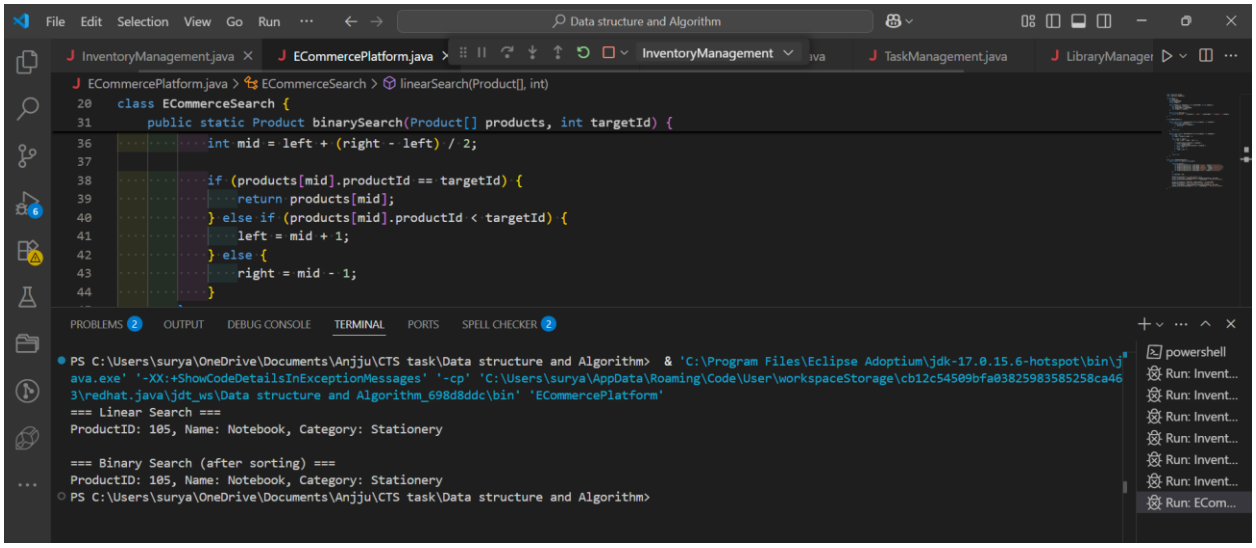
Exercise 2: E-commerce Platform Search Function

```
import java.util.Arrays;
import java.util.Comparator;

class Product {
    int productId;
    String productName;
    String category;

    public Product(int productId, String productName, String category) {
        this.productId = productId;
        this.productName = productName;
        this.category = category;
    }

    public String toString() {
```

```
        return "ProductID: " + productId + ", Name: " + productName + ", Category: " + category;
    }
}

class ECommerceSearch {

    public static Product linearSearch(Product[] products, int targetId) {
        for (Product p : products) {
            if (p.productId == targetId) {
                return p;
            }
        }
        return null;
    }

    public static Product binarySearch(Product[] products, int targetId) {
        int left = 0;
        int right = products.length - 1;

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

            if (products[mid].productId == targetId) {
                return products[mid];
            } else if (products[mid].productId < targetId) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }

        return null;
    }
}
```

```
public class ECommercePlatform {
    public static void main(String[] args) {

        Product[] products = {
            new Product(103, "Laptop", "Electronics"),
            new Product(101, "Shampoo", "Personal Care"),
            new Product(102, "Keyboard", "Electronics"),
            new Product(105, "Notebook", "Stationery"),
            new Product(104, "Mouse", "Electronics")
        };

        int searchId = 105;

        System.out.println("=== Linear Search ===");
        Product resultLinear = ECommerceSearch.linearSearch(products, searchId);
        System.out.println(resultLinear != null ? resultLinear : "Product not found");

        Arrays.sort(products, Comparator.comparingInt(p -> p.productId));
        System.out.println("\n=== Binary Search (after sorting) ===");
        Product resultBinary = ECommerceSearch.binarySearch(products, searchId);
        System.out.println(resultBinary != null ? resultBinary : "Product not found");
    }
}
```

Output:



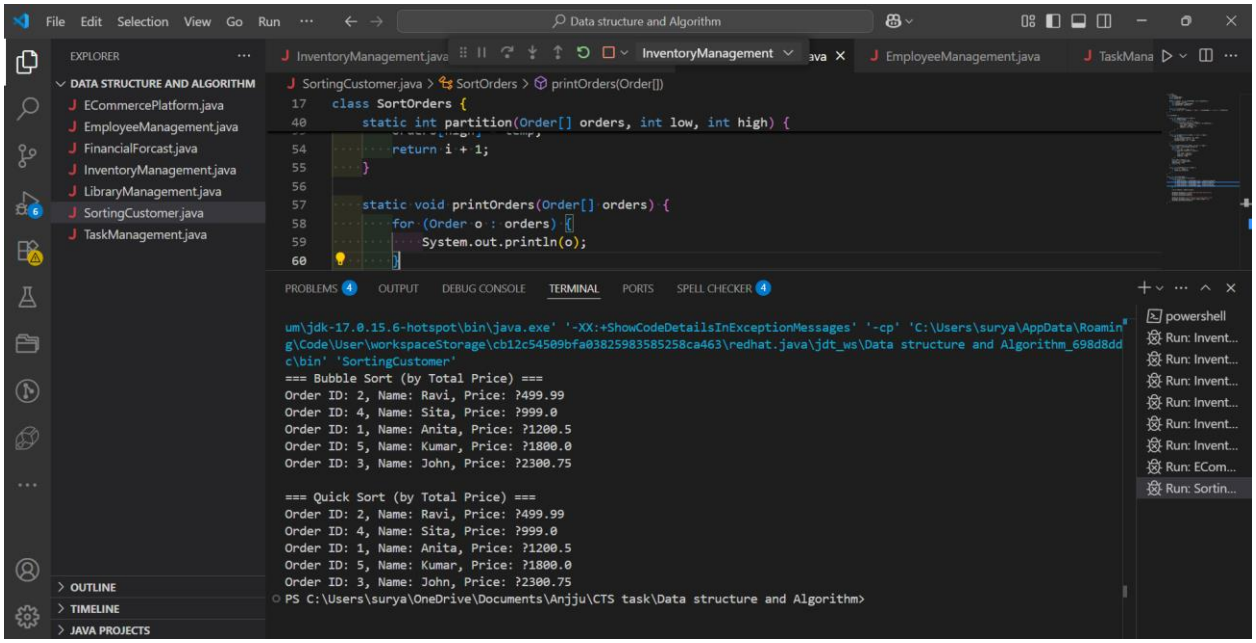
Exercise 3: Sorting Customer Orders

```
class Order {
    int orderId;
    String customerName;
    double totalPrice;

    Order(int orderId, String customerName, double totalPrice) {
        this.orderId = orderId;
        this.customerName = customerName;
        this.totalPrice = totalPrice;
    }

    public String toString() {
        return "Order ID: " + orderId + ", Name: " + customerName + ", Price: ₹" + totalPrice;
    }
}

class SortOrders {

    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].totalPrice > orders[j + 1].totalPrice) {
                Order temp = orders[j];
                orders[j] = orders[j + 1];
                orders[j + 1] = temp;
            }
        }
    }
}

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);
    }
}

static int partition(Order[] orders, int low, int high) {
    double pivot = orders[high].totalPrice;
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (orders[j].totalPrice <= 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;
}

static void printOrders(Order[] orders) {
```

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

public class SortingCustomer {
    public static void main(String[] args) {
        Order[] orders1 = {
            new Order(1, "Anita", 1200.50),
            new Order(2, "Ravi", 499.99),
            new Order(3, "John", 2300.75),
            new Order(4, "Sita", 999.00),
            new Order(5, "Kumar", 1800.00)
        };

        Order[] orders2 = orders1.clone();

        System.out.println("=== Bubble Sort (by Total Price) ===");
        SortOrders.bubbleSort(orders1);
        SortOrders.printOrders(orders1);

        System.out.println("\n=== Quick Sort (by Total Price) ===");
        SortOrders.quickSort(orders2, 0, orders2.length - 1);
        SortOrders.printOrders(orders2);
    }
}
```

Output:



Exercise 4: Employee Management System

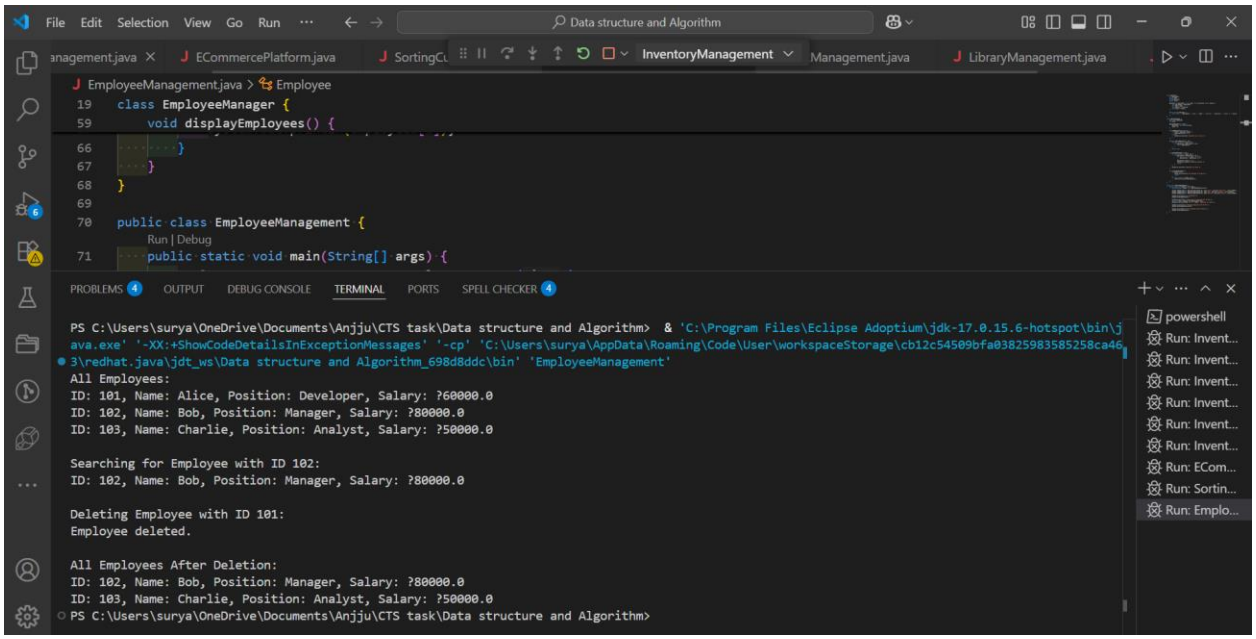
```
class Employee {
    int employeeId;
    String name;
    String position;
    double salary;

    Employee(int employeeId, String name, String position, double salary) {
        this.employeeId = employeeId;
        this.name = name;
        this.position = position;
        this.salary = salary;
    }

    public String toString() {
        return "ID: " + employeeId + ", Name: " + name + ", Position: " + position + ", Salary: ₹" + salary;
    }
}

class EmployeeManager {
    Employee[] employees;
```

```
int count;

EmployeeManager(int size) {
    employees = new Employee[size];
    count = 0;
}

void addEmployee(Employee e) {
    if (count < employees.length) {
        employees[count++] = e;
    } else {
        System.out.println("Employee list is full.");
    }
}

Employee searchEmployee(int id) {
    for (int i = 0; i < count; i++) {
        if (employees[i].employeeId == id) {
            return employees[i];
        }
    }
    return null;
}

void deleteEmployee(int id) {
    for (int i = 0; i < count; i++) {
        if (employees[i].employeeId == 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.");
    }

    void displayEmployees() {
        if (count == 0) {
            System.out.println("No employees to display.");
            return;
        }
        for (int i = 0; i < count; i++) {
            System.out.println(employees[i]);
        }
    }
}

public class EmployeeManagement {
    public static void main(String[] args) {
        EmployeeManager manager = new EmployeeManager(5);

        manager.addEmployee(new Employee(101, "Alice", "Developer", 60000));
        manager.addEmployee(new Employee(102, "Bob", "Manager", 80000));
        manager.addEmployee(new Employee(103, "Charlie", "Analyst", 50000));

        System.out.println("All Employees:");
        manager.displayEmployees();

        System.out.println("\nSearching for Employee with ID 102:");
        Employee found = manager.searchEmployee(102);
        System.out.println(found != null ? found : "Employee not found.");

        System.out.println("\nDeleting Employee with ID 101:");
        manager.deleteEmployee(101);

        System.out.println("\nAll Employees After Deletion:");
        manager.displayEmployees();
    }
```

```
}
```

Output:



Exercise 5: Task Management System

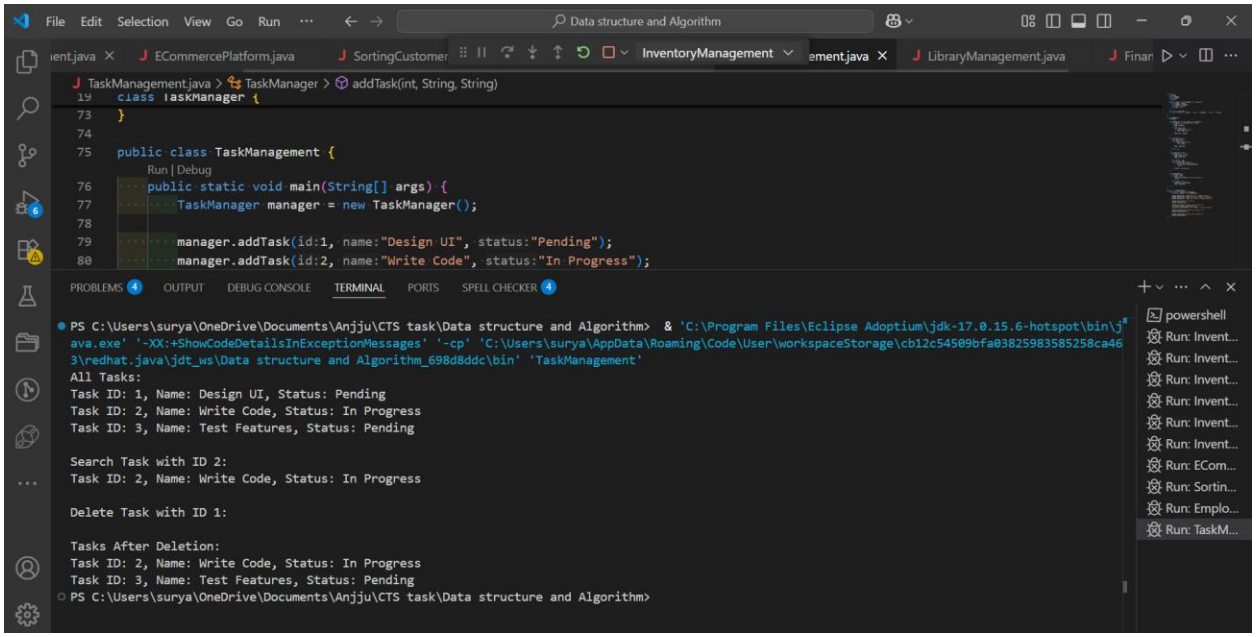
```
class Task {
    int taskId;
    String taskName;
    String status;
    Task next;

    Task(int taskId, String taskName, String status) {
        this.taskId = taskId;
        this.taskName = taskName;
        this.status = status;
        this.next = null;
    }

    public String toString() {
        return "Task ID: " + taskId + ", Name: " + taskName + ", Status: " + status;
```

```
    }
}

class TaskManager {
    Task head;

    void addTask(int id, String name, String status) {
        Task newTask = new Task(id, name, status);
        if (head == null) {
            head = newTask;
        } else {
            Task temp = head;
            while (temp.next != null) {
                temp = temp.next;
            }
            temp.next = newTask;
        }
    }

    Task searchTask(int id) {
        Task temp = head;
        while (temp != null) {
            if (temp.taskId == id) {
                return temp;
            }
            temp = temp.next;
        }
        return null;
    }

    void deleteTask(int id) {
        if (head == null) return;
        if (head.taskId == id) {
            head = head.next;
            return;
```

```
        }
        Task current = head;
        while (current.next != null) {
            if (current.next.taskId == id) {
                current.next = current.next.next;
                return;
            }
            current = current.next;
        }
    }

    void displayTasks() {
        Task temp = head;
        if (temp == null) {
            System.out.println("No tasks available.");
            return;
        }
        while (temp != null) {
            System.out.println(temp);
            temp = temp.next;
        }
    }
}

public class TaskManagement {
    public static void main(String[] args) {
        TaskManager manager = new TaskManager();

        manager.addTask(1, "Design UI", "Pending");
        manager.addTask(2, "Write Code", "In Progress");
        manager.addTask(3, "Test Features", "Pending");

        System.out.println("All Tasks:");
        manager.displayTasks();
```

```
        System.out.println("\nSearch Task with ID 2:");
        Task found = manager.searchTask(2);
        System.out.println(found != null ? found : "Task not found.");

        System.out.println("\nDelete Task with ID 1:");
        manager.deleteTask(1);

        System.out.println("\nTasks After Deletion:");
        manager.displayTasks();
    }
}
```

Output:



Exercise 6: Library Management System

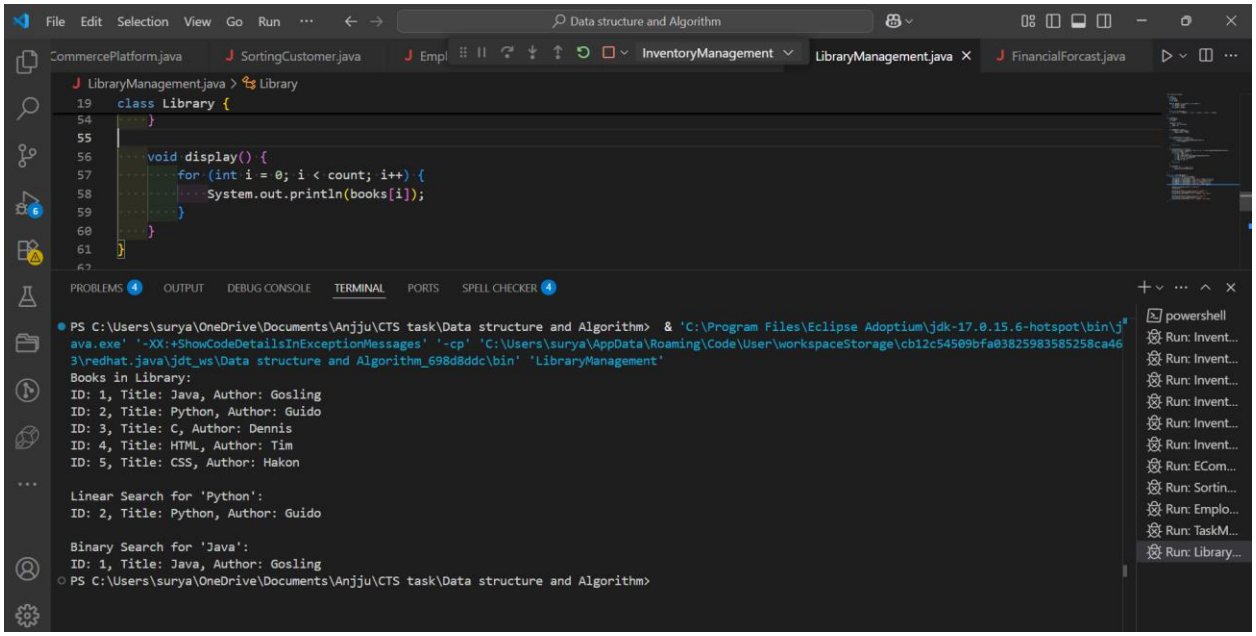
```
import java.util.Arrays;

class Book {
    int bookId;
    String title;
```

```
    String author;

    Book(int bookId, String title, String author) {
        this.bookId = bookId;
        this.title = title;
        this.author = author;
    }

    public String toString() {
        return "ID: " + bookId + ", Title: " + title + ", Author: " + author;
    }
}

class Library {
    Book[] books;
    int count;

    Library(int size) {
        books = new Book[size];
        count = 0;
    }

    void addBook(Book book) {
        if (count < books.length) {
            books[count++] = book;
        }
    }

    Book linearSearch(String title) {
        for (int i = 0; i < count; i++) {
            if (books[i].title.equalsIgnoreCase(title)) {
                return books[i];
            }
        }
        return null;
```

```
    }

    Book binarySearch(String title) {
        Arrays.sort(books, 0, count, (a, b) -> a.title.compareToIgnoreCase(b.title));
        int low = 0, high = count - 1;
        while (low <= high) {
            int mid = (low + high) / 2;
            int cmp = books[mid].title.compareToIgnoreCase(title);
            if (cmp == 0) return books[mid];
            if (cmp < 0) low = mid + 1;
            else high = mid - 1;
        }
        return null;
    }

    void display() {
        for (int i = 0; i < count; i++) {
            System.out.println(books[i]);
        }
    }
}

public class LibraryManagement {
    public static void main(String[] args) {
        Library lib = new Library(10);
        lib.addBook(new Book(1, "Java", "Gosling"));
        lib.addBook(new Book(2, "Python", "Guido"));
        lib.addBook(new Book(3, "C", "Dennis"));
        lib.addBook(new Book(4, "HTML", "Tim"));
        lib.addBook(new Book(5, "CSS", "Hakon"));

        System.out.println("Books in Library:");
        lib.display();

        System.out.println("\nLinear Search for 'Python':");
```

```
    Book result1 = lib.linearSearch("Python");
    System.out.println(result1 != null ? result1 : "Not found");

    System.out.println("\nBinary Search for 'Java':");
    Book result2 = lib.binarySearch("Java");
    System.out.println(result2 != null ? result2 : "Not found");
  }
}
```

Output:

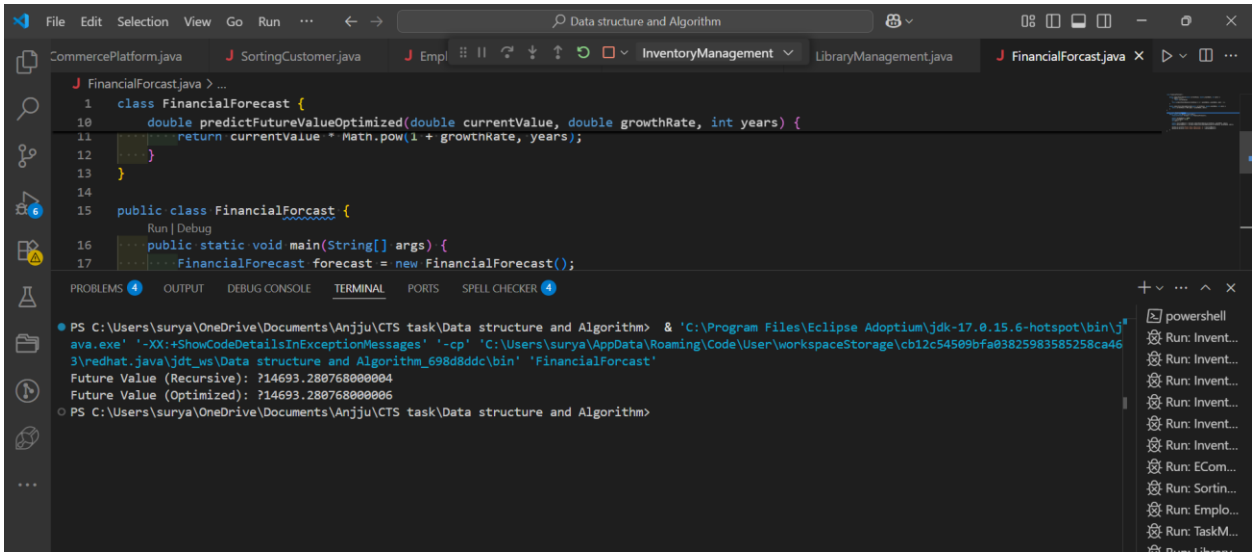


Exercise 7: Financial Forecasting

```
class FinancialForecast {

  double predictFutureValue(double currentValue, double growthRate, int years) {
    if (years == 0) {
      return currentValue;
    }
    return predictFutureValue(currentValue * (1 + growthRate), growthRate, years - 1);
  }
```

```
    double predictFutureValueOptimized(double currentValue, double growthRate, int years) {
        return currentValue * Math.pow(1 + growthRate, years);
    }
}

public class FinancialForcast {
    public static void main(String[] args) {
        FinancialForecast forecast = new FinancialForecast();

        double currentValue = 10000;
        double growthRate = 0.08;
        int years = 5;

        double recursiveResult = forecast.predictFutureValue(currentValue, growthRate, years);
        double optimizedResult = forecast.predictFutureValueOptimized(currentValue, growthRate, years);

        System.out.println("Future Value (Recursive): ₹" + recursiveResult);
        System.out.println("Future Value (Optimized): ₹" + optimizedResult);
    }
}
```

Output:


File Edit Selection View Go Run
Data structure and Algorithm
CommercePlatform.java
SortingCustomer.java
InventoryManagement
LibraryManagement.java
FinancialForcast.java
FinancialForcast.java > ...
class FinancialForecast {
double predictFutureValueOptimized(double currentValue, double growthRate, int years) {
return currentValue * Math.pow(1 + growthRate, years);
}
}
public class FinancialForcast {
Run | Debug
public static void main(String[] args) {
FinancialForecast forecast = new FinancialForecast();
PROBLEMS 4
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
SPELL CHECKER 4
PS C:\Users\surya\OneDrive\Documents\Anjju\CTS task\Data structure and Algorithm> & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.15.6-hotspot\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\surya\AppData\Roaming\Code\User\workspaceStorage\cb12c54509bfa03825983585258ca463\redhat.java\jdt_ws\Data structure and Algorithm_698d8ddc\bin' 'FinancialForcast'
Future Value (Recursive): ?14693.280768000004
Future Value (Optimized): ?14693.280768000006
PS C:\Users\surya\OneDrive\Documents\Anjju\CTS task\Data structure and Algorithm>
powershell
Run: Invent...
Run: Invent...
Run: Invent...
Run: Invent...
Run: Invent...
Run: Invent...
Run: ECom...
Run: Sortin...
Run: Emplo...
Run: TaskM...