**WEEK-1 DESIGN PRINCIPLES & PATTERNS HANDS ON**

**Exercise 1: Implementing the Singleton Pattern**

class Logger {

    // a static private instance veriable

    private static *Logger* log = new Logger();

    // a private constructor

    private Logger() {

        System.out.println("Logger is initialized.");

    }

    // a public static getinstance method

    public static *Logger* getInstance() {

        return log;

    }

}

// a seperate class to test the Logger

public class TestLogger {

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

*Logger* log1 = Logger.getInstance();

*Logger* log2 = Logger.getInstance();

        if(log1 == log2) {

            System.out.println("Both the instances are equal.");

        } else {

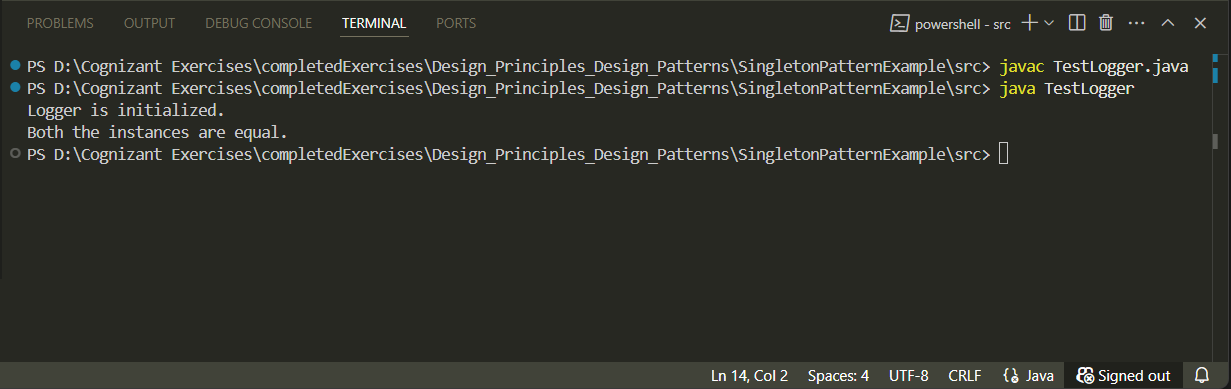
            System.out.println("Different instances.");

        }

    }

}

**Output :**

****

**Exercise 2: Implementing the Factory Method Pattern**

public abstract class WordDocument {

    public abstract *void* open();

}

public abstract class PDFDocument {

    public abstract *void* open();

}

public abstract class ExcelDocument {

    public abstract *void* open();

}

public class ExcelImplementation extends ExcelDocument {

    @*Override*

    public *void* open() {

        System.out.println("Opening Excel Document.");

    }

}

public class PDFImplementation extends PDFDocument {

    @*Override*

    public *void* open() {

        System.out.println("Opening PDF Document.");

    }

}

public class ExcelImplementation extends ExcelDocument {

    @*Override*

    public *void* open() {

        System.out.println("Opening Excel Document.");

    }

}

public abstract class DocumentFactory {

    public abstract Object createDocument();

}

public class ExcelFactory extends DocumentFactory {

    public ExcelDocument createDocument() {

        return new ExcelImplementation();

    }

}

public class PDFFactory extends *DocumentFactory* {

   public PDFFactory() {

   }

   public *PDFDocument* createDocument() {

      return new PDFImplementation();

   }

}

public class PDFFactory extends *DocumentFactory* {

   public PDFFactory() {

   }

   public *PDFDocument* createDocument() {

      return new PDFImplementation();

   }

}

public class TestClass {

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

        WordFactory wordDoc = new WordFactory();

        WordDocument word = wordDoc.createDocument();

        word.open();

        PDFFactory pdfDoc = new PDFFactory();

        PDFDocument pdf = pdfDoc.createDocument();

        pdf.open();

        ExcelFactory excelDoc = new ExcelFactory();

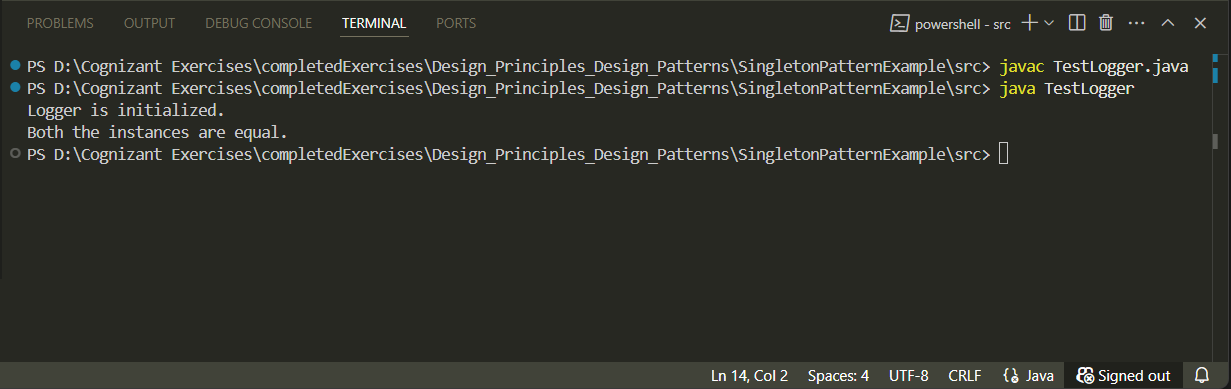
        ExcelDocument excel = excelDoc.createDocument();

        excel.open();

    }

}

**Output :**

****

**Exercise 3: Implementing the Builder Pattern**

public class Computer {

    private String CPU;

    private String ram;

    private String storage;

    private Computer(ComputerBuilder *builder*) {

        this.CPU = *builder*.CPU;

        this.ram = *builder*.ram;

        this.storage = *builder*.storage;

    }

    public *void* getSpecifications() {

        System.out.println("CPU : " + CPU + "\nRAM : " + ram + "\nStorage : " + storage);

    }

    public static class ComputerBuilder {

        private String CPU;

        private String ram;

        private String storage;

        public ComputerBuilder setCPU(String *CPU*) {

            this.CPU = *CPU*;

            return this;

        }

        public ComputerBuilder setRAM(String *ram*) {

            this.ram = *ram*;

            return this;

        }

        public ComputerBuilder setStorage(String *storage*) {

            this.storage = *storage*;

            return this;

        }

        public Computer build() {

            return new Computer(this);

        }

    }

}

public class TestClass {

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

        // creating a personal computer

        Computer personalPC = new Computer.ComputerBuilder()

                                .setCPU("Intel i5")

                                .setRAM("8GB")

                                .setStorage("512GB SSD")

                                .build();

        personalPC.getSpecifications();

        // creating a gaming computer

        Computer gamingPC = new Computer.ComputerBuilder()

                            .setCPU("Intel i9")

                            .setRAM("32GB")

                            .setStorage("1TB SSD")

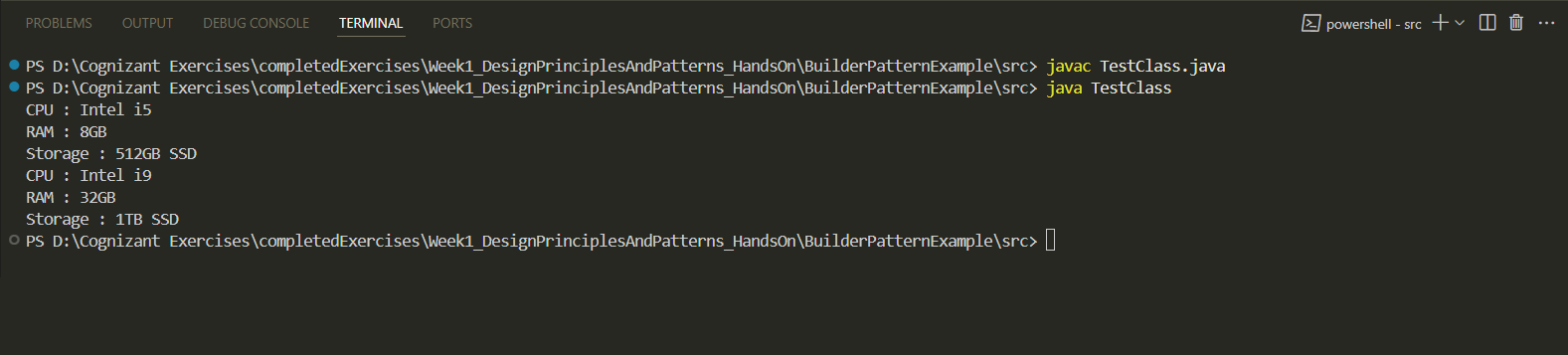
                            .build();

        gamingPC.getSpecifications();

    }

}

**Output :**

****

**Exercise 4: Implementing the Adapter Pattern**

public interface PaymentProcessor {

*void* processPayment(*double* *payment*);

}

public class GPay {

    public *void* makePayment(*double* *payment*) {

        System.out.println("Processing payment of " + *payment* + "rupees.");

    }

}

public class PayPal {

    public *void* makePayment(*double* *payment*) {

        System.out.println("Processing payment of " + *payment* + "rupees.");

    }

}

public class GPayAdapter implements PaymentProcessor {

    private GPay gpay;

    public GPayAdapter(GPay *gpay*) {

        this.gpay = *gpay*;

    }

    @*Override*

    public *void* processPayment(*double* *payment*) {

        gpay.makePayment(*payment*);

    }

}

public class PayPalAdapter implements PaymentProcessor {

    private PayPal paypal;

    public PayPalAdapter(PayPal *paypal*) {

        this.paypal = *paypal*;

    }

    @*Override*

    public *void* processPayment(*double* *payment*) {

        paypal.makePayment(*payment*);

    }

}

public class TestClass {

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

        PaymentProcessor paypal = new PayPalAdapter(new PayPal());

        paypal.processPayment(100000.0);

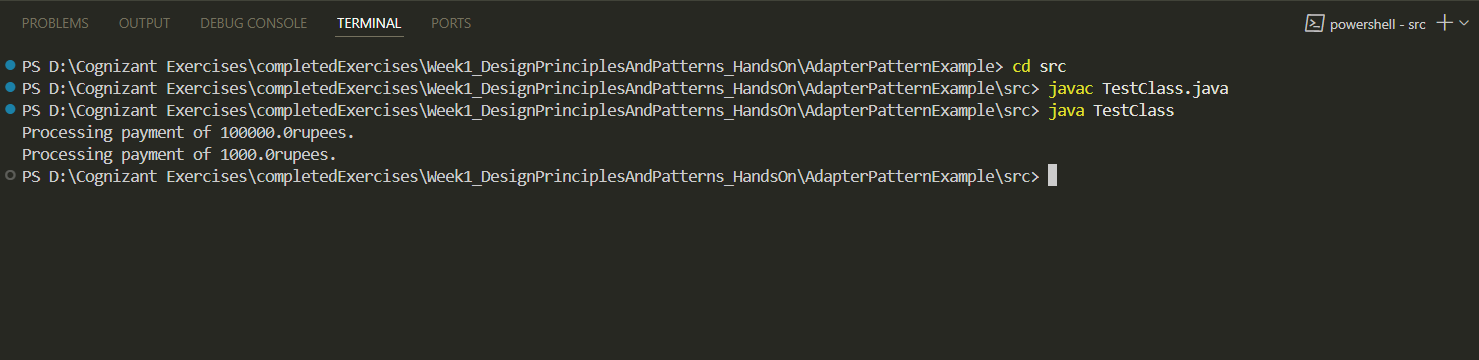
        PaymentProcessor gpay = new GPayAdapter(new GPay());

        gpay.processPayment(1000.0);

    }

}

**Output :**

****

**Exercise 5: Implementing the Decorator Pattern**

public interface Notifier {

    String send();

}

public class EmailNotifier implements Notifier {

    private String msg;

    public EmailNotifier(String *msg*) {

        this.msg = *msg*;

    }

    public String send() {

        return msg;

    }

}

public abstract class NotifierDecorator implements Notifier{

    private Notifier notifier;

    public NotifierDecorator(Notifier *notifier*) {

        this.notifier = *notifier*;

    }

    public String send() {

        return notifier.send();

    }

}

public class SMSNotifierDecorator extends NotifierDecorator {

    public SMSNotifierDecorator(Notifier *notifier*) {

        super(*notifier*);

    }

    public String send() {

        return "SMS " + super.send();

    }

}

public class SlackNotifierDecorator extends NotifierDecorator {

    public SlackNotifierDecorator(Notifier *notifier*) {

        super(*notifier*);

    }

    public String send() {

        return "Slack " + super.send();

    }

}

public class TestClass {

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

        Notifier notification = new EmailNotifier("Notification sent.");

        Notifier smsNotification = new SMSNotifierDecorator(notification);

        System.out.println(smsNotification.send());

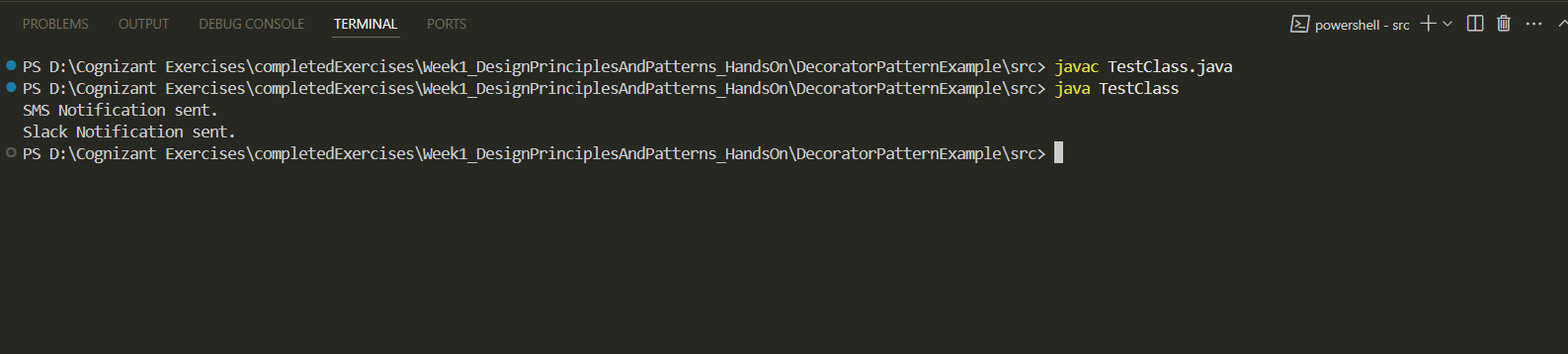
        Notifier slackNotification = new SlackNotifierDecorator(notification);

        System.out.println(slackNotification.send());

    }

}

**Output :**

****

**Exercise 7: Implementing the Observer Pattern**

public interface Stock {

*void* register(Observer *observer*);

*void* deregister(Observer *observer*);

*void* notifyObservers();

}

import java.util.ArrayList;

import java.util.List;

public class StockMarket implements Stock {

    private List<Observer> observers = new ArrayList<Observer>();

    private *int* state;

    public *void* setState(*int* *newState*) {

        this.state = *newState*;

        notifyObservers();

    }

    @*Override*

    public *void* register(Observer *observer*) {

        observers.add(*observer*);

    }

    @*Override*

    public *void* deregister(Observer *observer*) {

        observers.remove(*observer*);

    }

    @*Override*

    public *void* notifyObservers() {

        for(Observer observer : observers) {

            observer.update(state);

        }

    }

}

public interface Observer {

*void* update(*int* *state*);

}

public class MobileApp implements Observer {

    public String name;

    public MobileApp(String *name*) {

        this.name = *name*;

    }

    @*Override*

    public *void* update(*int* *state*) {

        System.out.println(name + ": Received update. New state is " + *state*);

    }

}

public class WebApp implements Observer {

    public String name;

    public WebApp(String *name*) {

        this.name = *name*;

    }

    @*Override*

    public *void* update(*int* *state*) {

        System.out.println(name + ": Received update. New state is " + *state*);

    }

}

public class TestClass {

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

        Notifier notification = new EmailNotifier("Notification sent.");

        Notifier smsNotification = new SMSNotifierDecorator(notification);

        System.out.println(smsNotification.send());

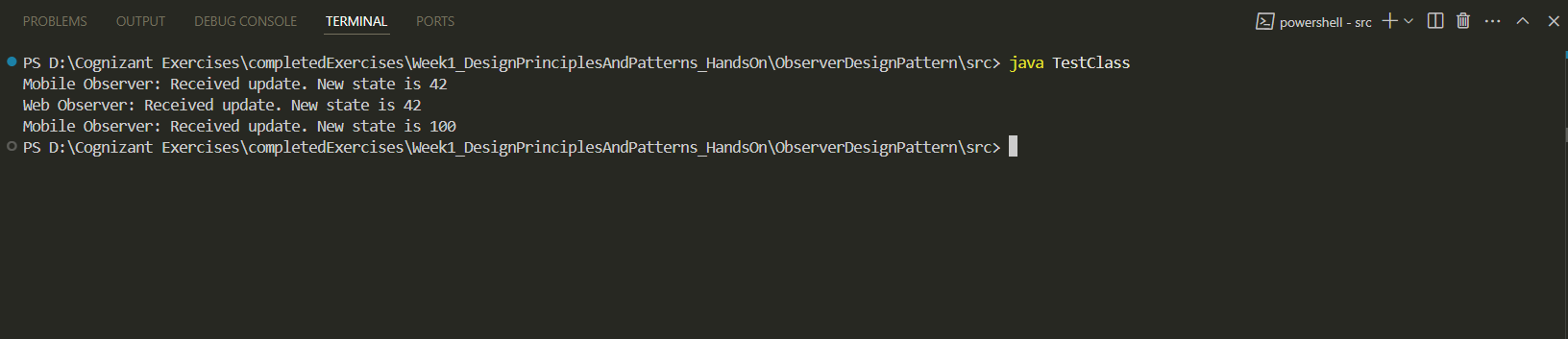
        Notifier slackNotification = new SlackNotifierDecorator(notification);

        System.out.println(slackNotification.send());

    }

}

**Output :**

****

**Exercise 8: Implementing the Strategy Pattern**

public interface PaymentStrategy {

*void* pay();

}

public class CreditCardPayment implements PaymentStrategy {

    public *void* pay() {

        System.out.println("You are paying through Credit Card.");

    }

}

public class PayPalPayment implements PaymentStrategy {

    public *void* pay() {

        System.out.println("You are paying through PayPal.");

    }

}

public class PaymentContext {

    private PaymentStrategy payment;

    public PaymentContext(PaymentStrategy *payment*) {

        this.payment = *payment*;

    }

    public *void* setPayment(PaymentStrategy *strategy*) {

        this.payment = *strategy*;

    }

    public *void* makePayment() {

        payment.pay();

    }

}

public class TestClass {

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

        PaymentContext payment = new PaymentContext(new CreditCardPayment());

        payment.makePayment();

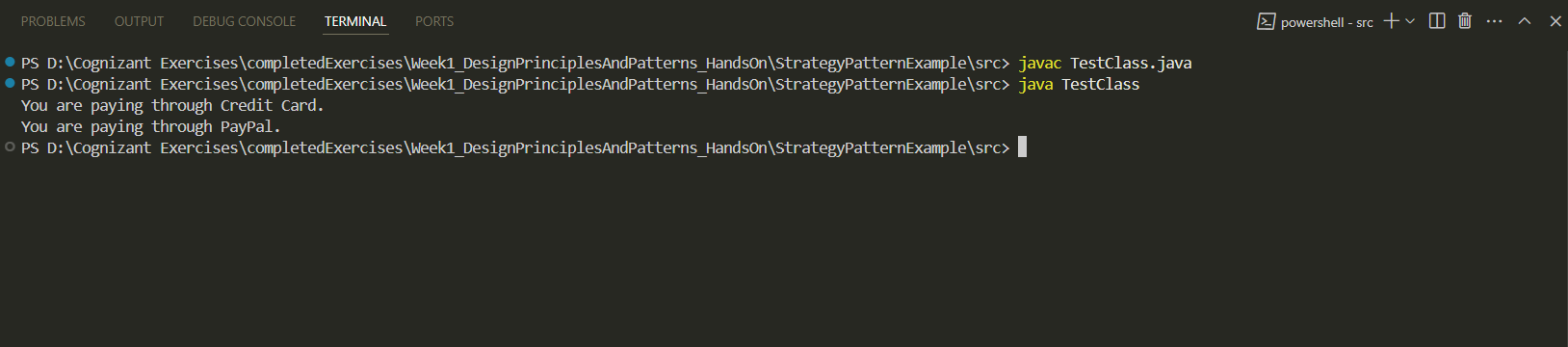
        payment.setPayment(new PayPalPayment());

        payment.makePayment();

    }

}

**Output :**

****

**Exercise 9: Implementing the Command Pattern**

public interface Command {

    public *void* execute();

}

public class LightOnCommand implements Command{

    private Light light;

    public LightOnCommand(Light *light*) {

        this.light = *light*;

    }

    public *void* execute() {

        this.light.turnOn();

    }

}

public class LightOffCommand implements Command {

    private Light light;

    public LightOffCommand(Light *light*) {

        this.light = *light*;

    }

    public *void* execute() {

        this.light.turnOff();

    }

}

public class RemoteControl {

    Command command;

    public *void* setCommand(Command *command*) {

        this.command = *command*;

    }

    public *void* pressSwitch() {

        this.command.execute();

    }

}

public class Light {

    public *void* turnOn() {

        System.out.println("Light is turned on.");

    }

    public *void* turnOff() {

        System.out.println("Light is turned off.");

    }

}

public class TestClass {

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

        Light light = new Light();

        RemoteControl control = new RemoteControl();

        Command onCommand = new LightOnCommand(light);

        Command offCommand = new LightOffCommand(light);

        control.setCommand(onCommand);

        control.pressSwitch();

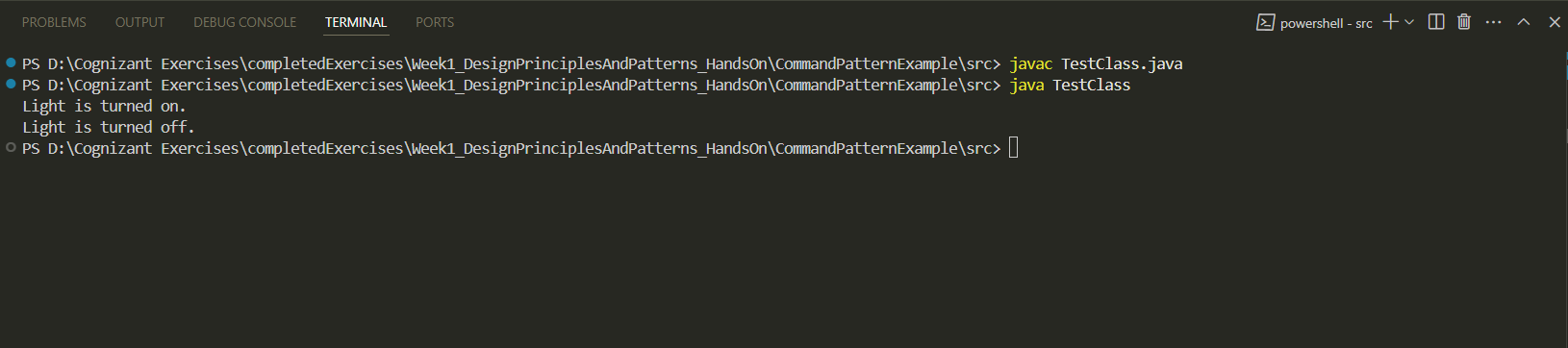
        control.setCommand(offCommand);

        control.pressSwitch();

    }

}

**Output :**

****

**Exercise 10: Implementing the MVC Pattern**

public class Student {

    public String name;

    public *int* id;

    public String grade;

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

        this.name = *name*;

    }

    public *void* setID(*int* *id*) {

        this.id = *id*;

    }

    public *void* setGrade(String *grade*) {

        this.grade = *grade*;

    }

    public String getName() {

        return this.name;

    }

    public *int* getID() {

        return this.id;

    }

    public String getGrade() {

        return this.grade;

    }

}

public class StudentView {

    public *void* displayStudentDetails(String *name*, *int* *id*, String *grade*) {

        System.out.println("Student Name : " + *name* + ",\nStudent ID : " + *id* + ",\nStudent Grade : " + *grade* + "\n");

    }

}

public class StudentController {

    private Student model;

    private StudentView view;

    public StudentController(Student *model*, StudentView *view*) {

        this.model = *model*;

        this.view = *view*;

    }

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

        model.setName(*name*);

    }

    public *void* setStudentID(*int* *id*) {

        model.setID(*id*);

    }

    public *void* setStudentGrade(String *grade*) {

        model.setGrade(*grade*);

    }

    public *void* getStudentName() {

        model.getName();

    }

    public *void* getStudentID() {

        model.getID();

    }

    public *void* getStudentGrade() {

        model.getGrade();

    }

    public *void* updateView() {

        view.displayStudentDetails(model.name, model.id, model.grade);

    }

}

public class TestClass {

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

        Student student\_model = new Student();

        student\_model.setName("Devi Saranya");

        student\_model.setID(6137379);

        student\_model.setGrade("A");

        StudentView student\_view = new StudentView();

        StudentController student\_controller = new StudentController(student\_model, student\_view);

        student\_controller.updateView();

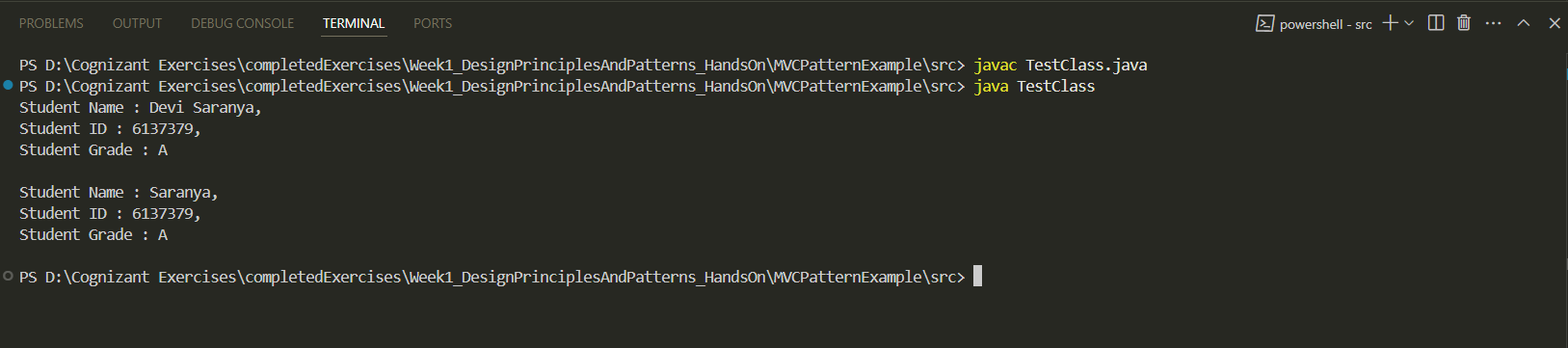
        student\_controller.setStudentName("Saranya");

        student\_controller.updateView();

    }

}

**Output :**

****

**Exercise 11: Implementing Dependency Injection**

public interface CustomerRepository {

    String findCustomerByID(*int* *id*);

}

public class CustomerRepositoryImpl implements CustomerRepository {

    @*Override*

    public String findCustomerByID(*int* *id*) {

        return "Customer " + *id* + " Devi Saranya Yanamadala.";

    }

}

public class CustomerService {

    private final CustomerRepository customerRepo;

    public CustomerService(CustomerRepository *customerRepo*) {

        this.customerRepo = *customerRepo*;

    }

    public *void* getCustomerDetails(*int* *id*) {

        String customer = customerRepo.findCustomerByID(*id*);

        System.out.println("Customer found. " + customer);

    }

}

public class TestClass {

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

        CustomerRepositoryImpl customerImpl = new CustomerRepositoryImpl();

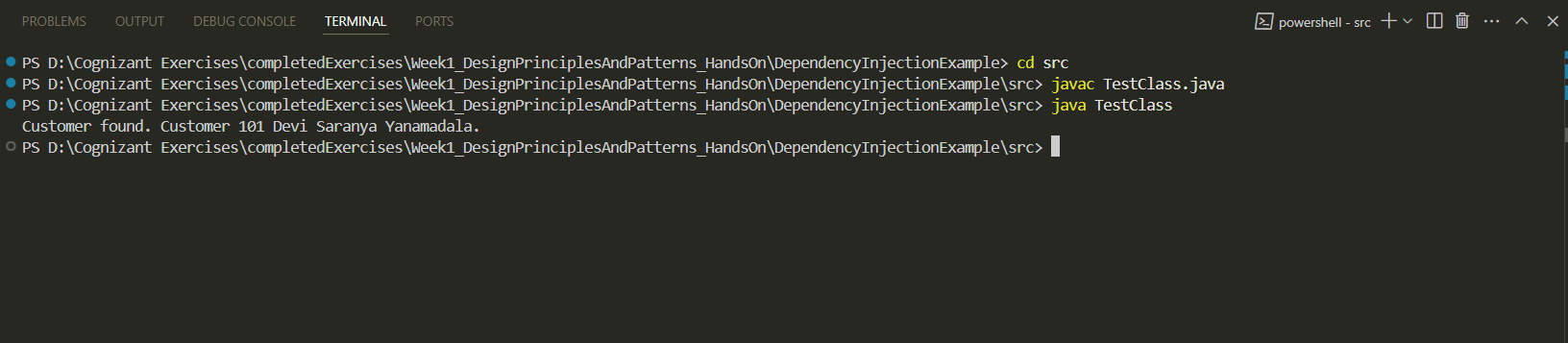
        CustomerService customerServ = new CustomerService(customerImpl);

        customerServ.getCustomerDetails(101);

    }

}

**Output :**



**WEEK 1 DATA STRUCTURES & ALGORITHMS HANDS ON**

**Exercise 1: Inventory Management System**

public 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;

    }

    // setter methods

    public *void* setProductID(*int* *productID*) {

        this.productID = productID;

    }

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

        this.productName = productName;

    }

    public *void* setProductQuantity(*int* *quantity*) {

        this.quantity = quantity;

    }

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

        this.price = price;

    }

    // getter methods

    public *int* getProductID() {

        return this.productID;

    }

    public *String* getProductName() {

        return this.productName;

    }

    public *int* getProductQuantity() {

        return this.quantity;

    }

    public *double* getProductPrice() {

        return this.price;

    }

    public *String* getProductDetails(*Product* *product*) {

        return "Product ID : " + product.productID + ", "

                + "\nProduct Name : " + product.productName + ", "

                + "\nProduct Quantity : " + product.quantity + ", "

                + "\nProduct Price : " + product.price  + ".\n";

    }

}

import java.util.HashMap;

public class ProductImplementation {

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

    public ProductImplementation() {

        products = new *HashMap*<>();

    }

    // Add product

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

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

    }

    // Update product

    public *void* updateProduct(*int* *productID*, *String* *productName*, *int* *quantity*, *double* *price*) {

*Product* updatingProduct = products.get(productID);

        // if the product is not present

        if(updatingProduct == null) {

            System.out.println("Product with product ID " + productID + " is not found.");

        } else {

            updatingProduct.setProductName(productName);

            updatingProduct.setProductQuantity(quantity);

            updatingProduct.setProductPrice(price);

        }

    }

    // Remove product

    public *void* removeProduct(*int* *productID*) {

*Product* remove = products.get(productID);

        // If product is not present

        if(remove == null) {

            System.out.println("Product with product ID " + productID + " is not found.");

        } else {

            products.remove(productID);

        }

    }

    // Get products

    public *void* getAvailableProducts() {

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

            System.out.println(product.getProductDetails(product));

        }

    }

}

public class InventoryManagement {

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

        ProductImplementation inventory = new ProductImplementation();

        Product jeans = new Product(1, "H&M High Waist Jeans", 10, 1000.0);

        Product tops = new Product(2, "Floral Top", 5, 250.0);

        Product sandles = new Product(18, "Black Sandles", 5, 500.0);

        Product tshirts = new Product(52, "Women T-Shirts", 15, 250.0);

        // Add all the products to the list

        inventory.addProduct(jeans);

        inventory.addProduct(tops);

        inventory.addProduct(sandles);

        inventory.addProduct(tshirts);

        // Get all the available products

        inventory.getAvailableProducts();

        // Update the jeans product with new quantity and price

        inventory.updateProduct(1, "H&M High Waist Jeans", 50, 2000.0);

        // Get all the available products after inventory update

        System.out.println("After updation of Inventory.");

        inventory.getAvailableProducts();

        // deleting sandles product

        inventory.removeProduct(18);

        // Available products

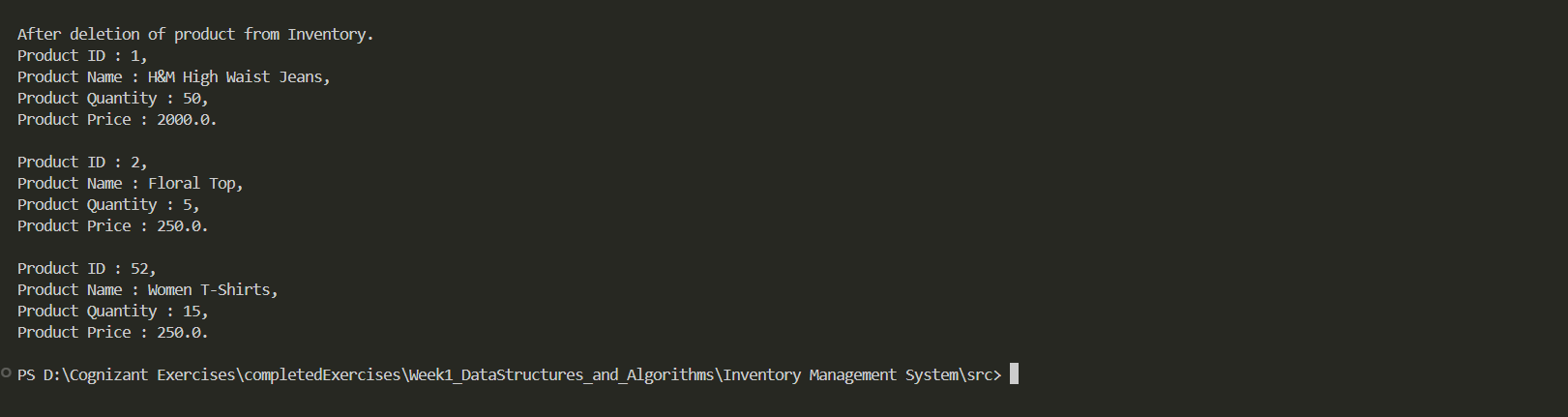
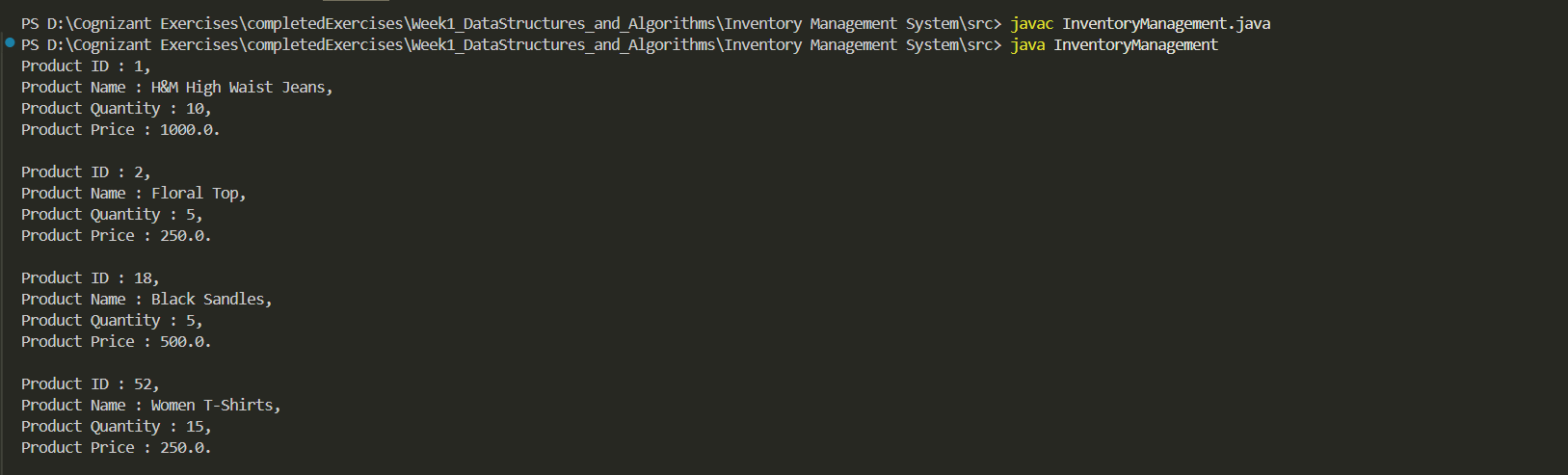
        System.out.println("After deletion of product from Inventory.");

        inventory.getAvailableProducts();

    }

}

**Output :**

****

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

public class Product {

*int* productID;

*String* name, category;

    public Product(*int* *productID*, *String* *name*, *String* *category*) {

        this.productID = productID;

        this.name = name;

        this.category = category;

    }

    public *void* getAvailableProducts(*Product* *product*) {

        System.out.println("Product ID : " + product.productID + "\nProduct Name : " + product.name + "\nProduct Category : " + product.category + ".\n");

    }

}

public class LinearSearchImpl {

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

        System.out.println("Searching for products Linear Search.");

        for(*Product* product : products) {

            if(product.name.equalsIgnoreCase(name)) {

                return product;

            }

        }

        return null;

    }

}

import java.util.Arrays;

import java.util.Comparator;

public class BinarySearchImpl {

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

        System.out.println("Searching for products using Binary Search.");

        sortProducts(products);

*int* left = 0, right = products.length - 1;

        while(left <= right) {

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

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

            if(cmp == 0) {

                return products[mid];

            } else if(cmp < 0) {

                left = mid + 1;

            } else {

                right = mid - 1;

            }

        }

        return null;

    }

    public *void* sortProducts(*Product*[] *products*) {

        Arrays.sort(products, Comparator.comparing(p *->* p.name.toLowerCase()));

    }

}

public class ProductSearching {

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

        Product[] products = { new Product(101, "Laptop", "Electronics"),

                                new Product(102, "Shoes", "Fashion"),

                                new Product(103, "Mobile", "Electronics"),

                                new Product(104, "Watch", "Accessories")

                            };

        // Searching for products which are present

        LinearSearchImpl linear = new LinearSearchImpl();

        Product linear\_found = linear.linearSearch(products, "Watch");

        System.out.println((checkIfFound(linear\_found)));

        BinarySearchImpl binary = new BinarySearchImpl();

        Product binary\_found = binary.binarySearch(products, "Watch");

        System.out.println((checkIfFound(binary\_found)));

        // Searching for products which are not present

        Product linear\_notfound = linear.linearSearch(products, "Gaming Laptop");

        System.out.println((checkIfFound(linear\_notfound)));

        Product binary\_notfound = binary.binarySearch(products, "Sandles");

        System.out.println((checkIfFound(binary\_notfound)));

    }

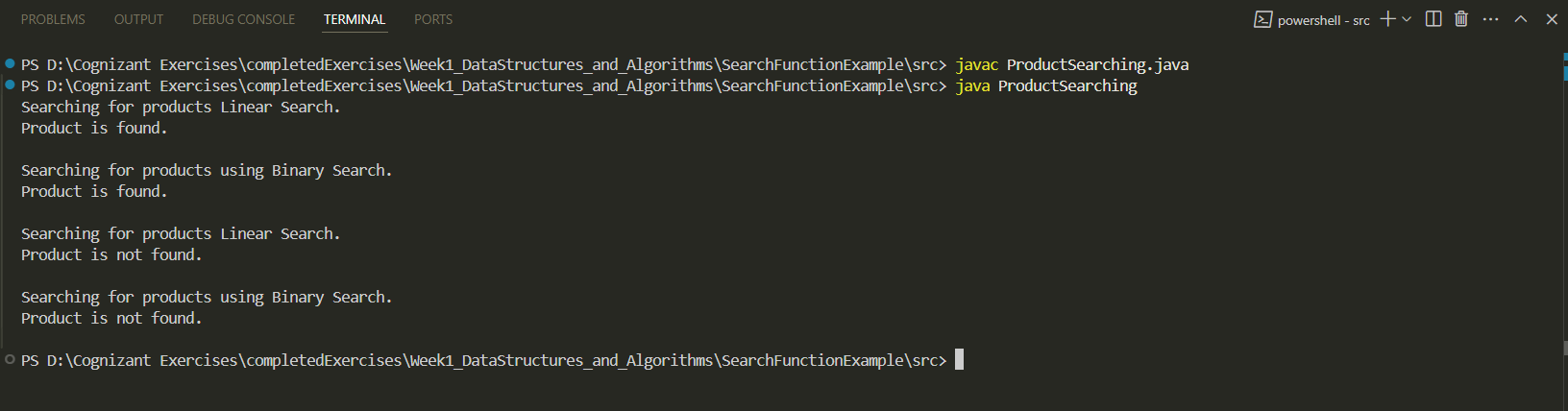
    public static String checkIfFound(Product *found*) {

        return (*found* != null)? "Product is found.\n" : "Product is not found.\n";

    }

}

**Output :**

****

**Exercise 3: Sorting Customer Orders**

public class Order {

    String orderid;

    String customerName;

*double* totalPrice;

    public Order(String *orderid*, String *customerName*, *double* *totalPrice*) {

        this.orderid = *orderid*;

        this.customerName = *customerName*;

        this.totalPrice = *totalPrice*;

    }

    public String getDetails(){

        return "Order ID : " + this.orderid + "\n"

                + "Customer Name : " + this.customerName + "\n"

                + "Total Price : " + this.totalPrice + "\n";

    }

}

public class BubbleSortImpl {

    public *void* bubbleSort(Order[] *orders*) {

*int* N = *orders*.length;

*boolean* swapped;

        for(*int* i = 0; i < N; i++) {

            swapped = false;

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

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

                    // swapping

                    Order temp = *orders*[j];

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

*orders*[j+1] = temp;

                    swapped = true;

                }

            }

            if(!swapped) {

                break;

            }

        }

    }

}

public class QuickSortImpl {

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

        if(*low* < *high*) {

*int* pivotIndex = partition(*orders*, *low*, *high*);

            quickSort(*orders*, *low*, pivotIndex - 1);

            quickSort(*orders*, pivotIndex + 1, *high*);

        }

    }

    public *int* partition(Order[] *orders*, *int* *low*, *int* *high*) {

        Order pivot = *orders*[*high*];

*int* i = *low* - 1;

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

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

                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 class SortingImplementation {

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

        // Creating some orders first

        Order[] bubble\_orders = {

                            new Order("ORD826315", "Devi Saranya", 60000.0),

                            new Order("ORD103472", "Akshaya", 2500.0),

                            new Order("ORD592380", "Harshitha", 1000.0),

                            new Order("ORD774911", "Tejaswi", 750.0),

                            new Order("ORD449027", "Deepika", 2050.0)

                        };

        System.out.println("Bubble Sort Algorithm...");

        BubbleSortImpl bubble\_sort = new BubbleSortImpl();

        bubble\_sort.bubbleSort(bubble\_orders);

        printSortedArray(bubble\_orders);

        Order[] quick\_orders = {

                            new Order("ORD501637", "Devi Saranya", 60000.0),

                            new Order("ORD680153", "Akshaya", 2500.0),

                            new Order("ORD218934", "Harshitha", 1000.0),

                            new Order("ORD375840", "Tejaswi", 750.0),

                            new Order("ORD960721", "Deepika", 2050.0)

                        };

        System.out.println("Quick Sort Algorithm...");

        QuickSortImpl quick\_sort = new QuickSortImpl();

        quick\_sort.quickSort(quick\_orders, 0, quick\_orders.length-1);

        printSortedArray(quick\_orders);

    }

    public static *void* printSortedArray(Order[] *orders*) {

        for(Order order : *orders*) {

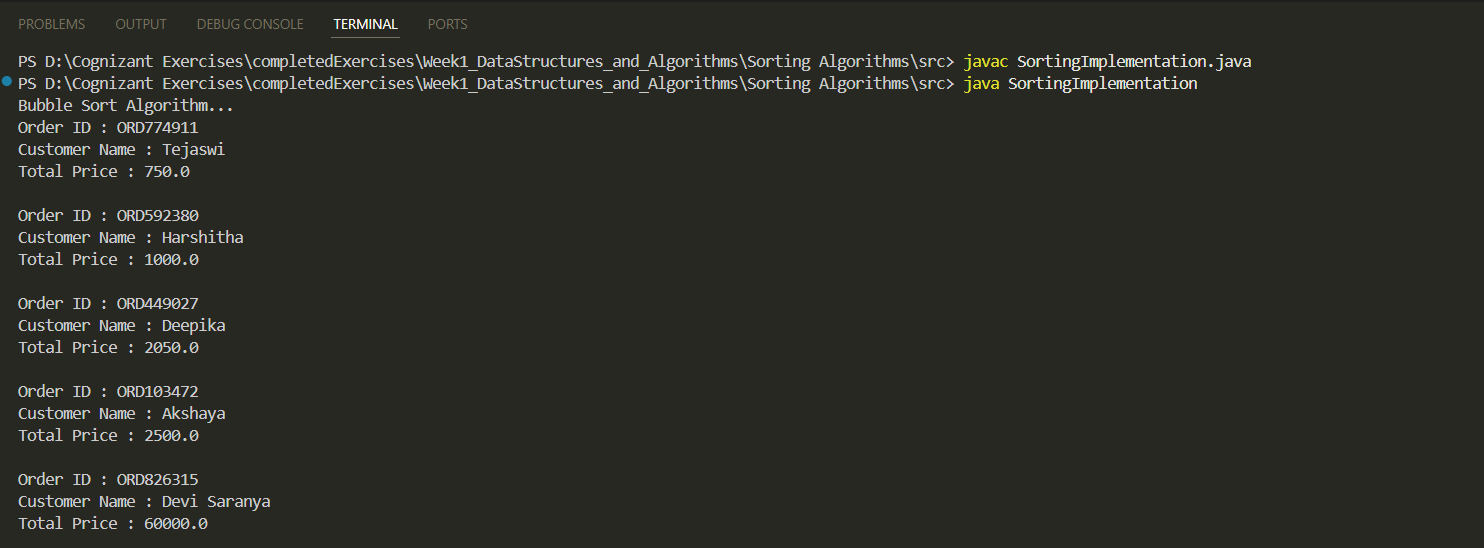
            System.out.println(order.getDetails());

        }

    }

}

**Output :**

****

**Exercise 4: Employee Management System**

public class Employee {

    String employeeID;

    String name;

    String position;

*double* salary;

    public Employee(String *employeeID*, String *name*, String *position*, *double* *salary*) {

        this.employeeID = *employeeID*;

        this.name = *name*;

        this.position = *position*;

        this.salary = *salary*;

    }

    public String getEmployeeDetails() {

        return "Employee ID : " + ((this != null)? this.employeeID : "NULL") +

                "\n Employee Name : " + ((this != null)? this.name : "NULL") +

                "\n Employee Position : " + ((this != null)? this.position : "NULL") +

                "\n Employee Salary : " + ((this != null)? this.salary : "NULL") + "\n";

    }

}

public class EmployeeManagement {

    private Employee[] employees;

*int* count = 0;

    public EmployeeManagement(*int* *size*) {

        employees = new Employee[*size*];

    }

    public *void* addEmployee(Employee *employee*) {

        if(count < employees.length) {

            employees[count++] = *employee*;

        } else {

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

        }

    }

    public *boolean* searchEmployee(String *empID*) {

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

            if(employees[i].employeeID == *empID*) {

                return true;

            }

        }

        return false;

    }

    public *void* deleteEmployee(String *empID*) {

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

            if(employees[i].employeeID == *empID*) {

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

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

                }

                employees[count - 1] = null;

                count--;

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

                return;

            }

        }

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

    }

    public *void* getAllEmployees() {

        for(Employee employee : employees) {

            if(employee != null)

                System.out.println(employee.getEmployeeDetails());

        }

    }

}

public class UnderstandArrays {

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

        EmployeeManagement emp = new EmployeeManagement(5);

        // Adding new employees

        emp.addEmployee(new Employee("EMP1023", "Alice", "Manager", 75000.0));

        emp.addEmployee(new Employee("EMP4871", "Bob", "Developer", 60000.0));

        emp.addEmployee(new Employee("EMP2509", "Charlie", "Designer", 55000.0));

        // Showing all employees

        emp.getAllEmployees();

        // Deleting an employee

        System.out.println("Deleting Employee EMP4871");

        emp.deleteEmployee("EMP4871");

        // Employees after removing EMP4871

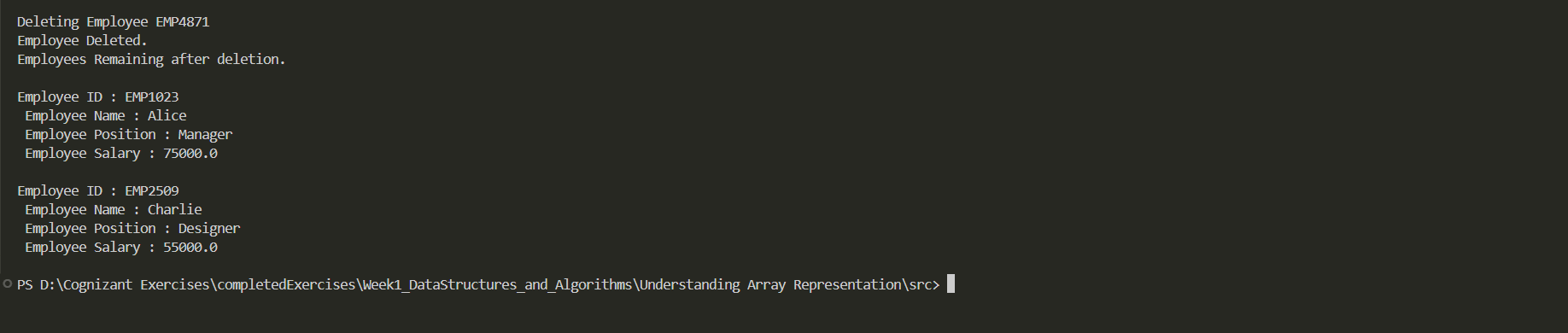
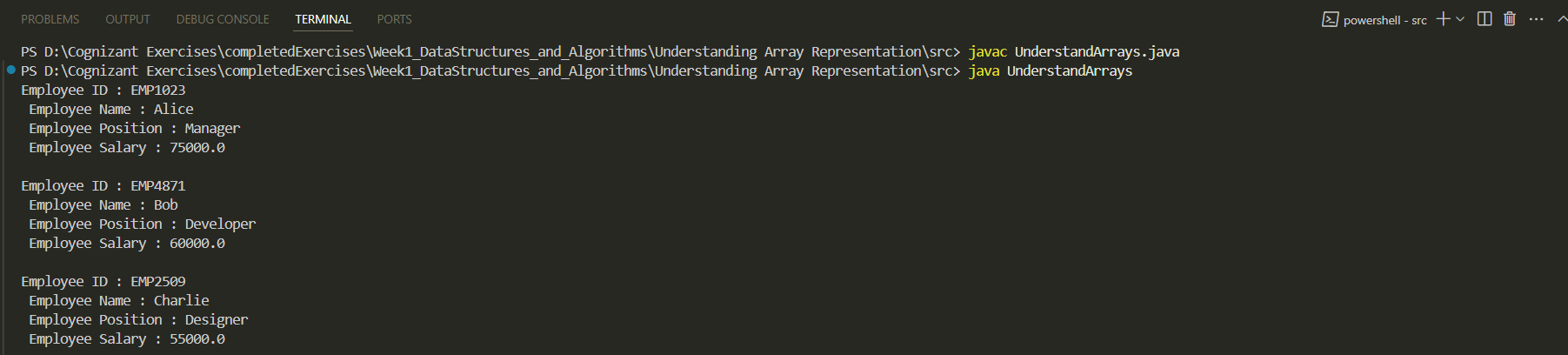
        System.out.println("Employees Remaining after deletion.\n");

        emp.getAllEmployees();

    }

}

**Output :**

****

**Exercise 5: Task Management System**

public class Task {

*int* taskID;

    String taskName, status;

    public Task(*int* *taskID*, String *taskName*, String *status*) {

        this.taskID = *taskID*;

        this.taskName = *taskName*;

        this.status = *status*;

    }

    public String getTaskDetails() {

        return "Task ID : " + this.taskID +

                "\n Task Name : " + this.taskName +

                "\n Task Status : " + this.status + "\n";

    }

}

public class LinkedList {

    Node head = null, dummy = head;

*int* length = 0;

    class Node {

        Task data;

        Node next;

        Node(Task *data*) {

            this.data = *data*;

            this.next = null;

        }

    }

    public *void* addTask(*int* *taskID*, String *taskName*, String *status*) {

        Task task = new Task(*taskID*, *taskName*, *status*);

        Node node = new Node(task);

        if(head == null) {

            head = node;

            dummy = head;

        } else {

            dummy.next = node;

            dummy = dummy.next;

        }

        length++;

    }

    public *void* deleteTask(*int* *taskID*) {

        Node temp = head, prev = null;

        while(temp != null) {

            if(temp.data.taskID == *taskID*) {

                // delete it

                if(prev == null) {

                    head = temp;

                } else {

                    prev.next = temp.next;

                }

                // If it is deleted then break out

                break;

            }

            // traverse the other nodes

            prev = temp;

            temp = temp.next;

            length--;

        }

    }

    public *void* findTask(*int* *taskID*) {

        Node temp = head;

        while(temp != null) {

            if(temp.data.taskID == *taskID*) {

                System.out.println("Task with task ID " + *taskID* + " is found.\n");

                break;

            }

            temp = temp.next;

        }

        System.out.println("Task with task ID " + *taskID* + " is not found.\n");

    }

    public *void* viewAllTasks() {

        Node temp = head;

        while(temp != null) {

            System.out.println(temp.data.getTaskDetails());

            temp = temp.next;

        }

    }

}

public class TaskManagement {

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

        LinkedList linkedlist = new LinkedList();

        // Add tasks to the list

        linkedlist.addTask(101, "Design UI", "Pending");

        linkedlist.addTask(102, "Develop Backend", "In Progress");

        linkedlist.addTask(103, "Write Tests", "Pending");

        linkedlist.addTask(104, "Deploy App", "Not Started");

        linkedlist.addTask(105, "Review Code", "Completed");

        // View all the tasks

        linkedlist.viewAllTasks();

        // delete task 104

        linkedlist.deleteTask(104);

        // view all tasks after deletion

        System.out.println("Tasks after deletion.\n");

        linkedlist.viewAllTasks();

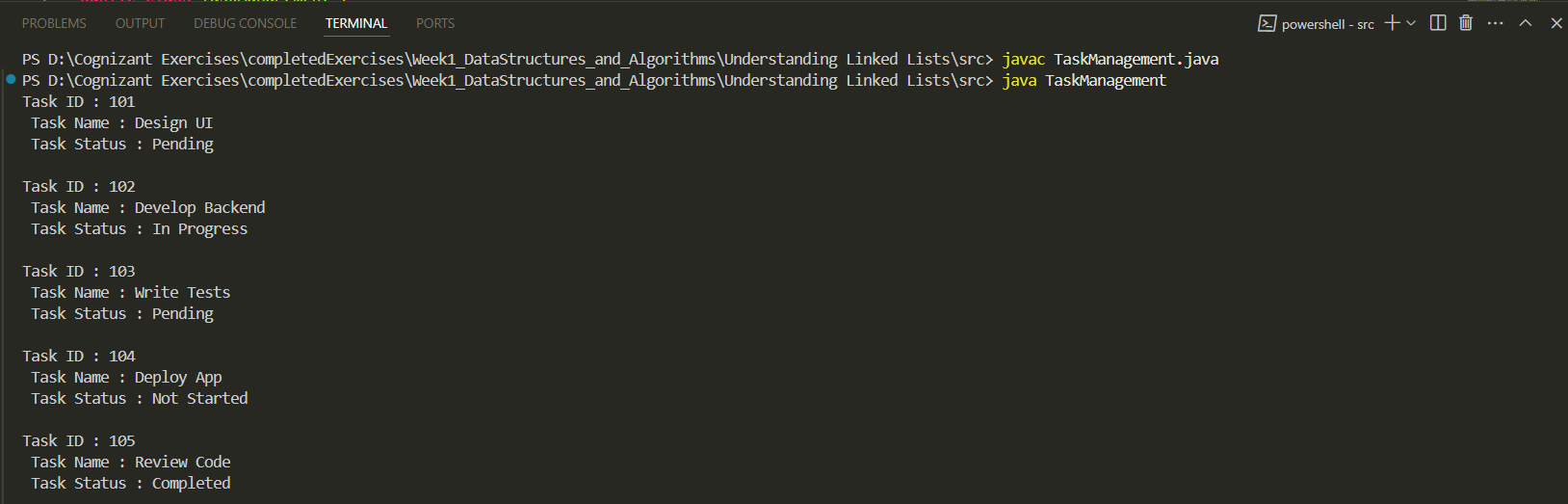
        // search for task 101

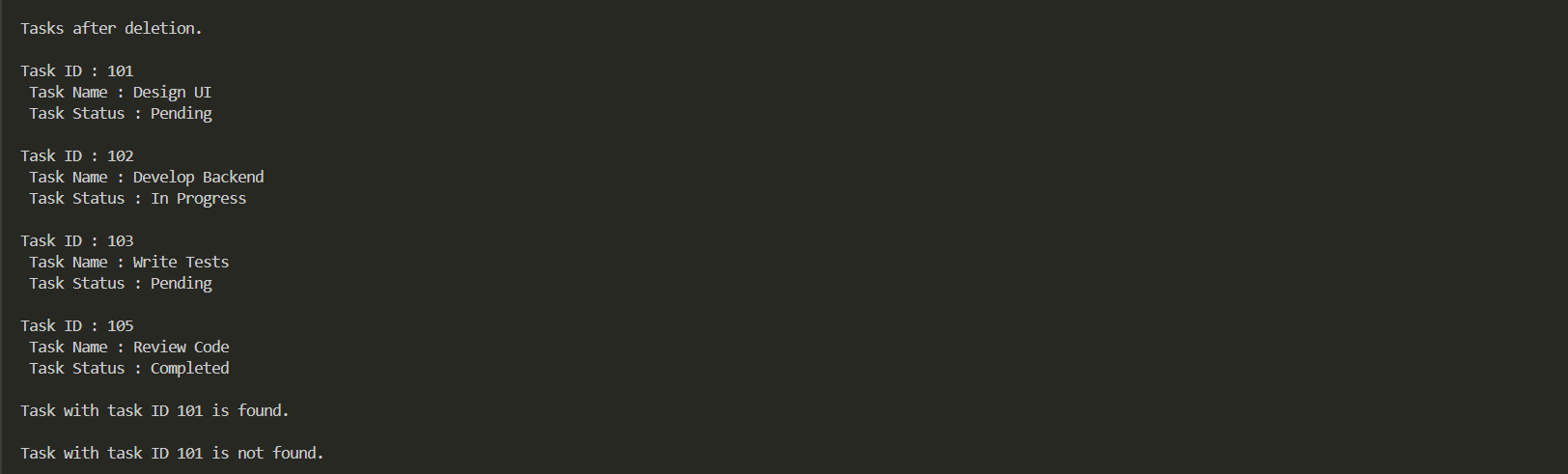
        linkedlist.findTask(101);

    }

}

**Output :**

****

****

**Exercise 6: Library Management System**

public class Book {

    String bookID, title, author;

    public Book(String *bookID*, String *title*, String *author*) {

        this.bookID = *bookID*;

        this.title = *title*;

        this.author = *author*;

    }

    public String getBookDetails() {

        return "Book ID : " + this.bookID +

                "\n Book Title : " + this.title +

                "\n Book Author : " + this.author + "\n";

    }

}

public class LinearSearch {

    public String linearSearchAlgorithm(Book[] *books*, String *title*) {

        for(Book book : *books*) {

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

                return *title* + " is found. \n";

            }

        }

        return *title* + " is not found. \n";

    }

}

import java.util.Arrays;

import java.util.Comparator;

public class BinarySearch {

    public String binarySearchAlgorithm(Book[] *books*, String *title*) {

        Arrays.sort(*books*, Comparator.comparing(*book* *->* *book*.title.toLowerCase()));

*int* low = 0, high = *books*.length - 1;

        while(low <= high) {

*int* mid = low + (high - low) / 2;

*int* compare = *books*[mid].title.compareToIgnoreCase(*title*);

            if(compare == 0) {

                return *title* + " is found. \n";

            } else if(compare < 0) {

                low = mid + 1;

            } else {

                high = mid - 1;

            }

        }

        return *title* + " is not found. \n";

    }

}

public class LibraryManagement {

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

        // Creating a list of books

        Book[] books = {

                        new Book("BK1023", "Clean Code", "Robert C. Martin"),

                        new Book("BK5408", "Design Patterns", "Erich Gamma"),

                        new Book("BK3876", "Effective Java", "Joshua Bloch"),

                        new Book("BK7642", "Introduction to Algorithms", "Thomas H. Cormen"),

                        new Book("BK2194", "The Pragmatic Programmer", "Andy Hunt")

        };

        // Using Linear Search

        LinearSearch linear\_search = new LinearSearch();

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

        System.out.println(linear\_search.linearSearchAlgorithm(books, "Effective Java"));

        // Using Binary Search

        BinarySearch binary\_search = new BinarySearch();

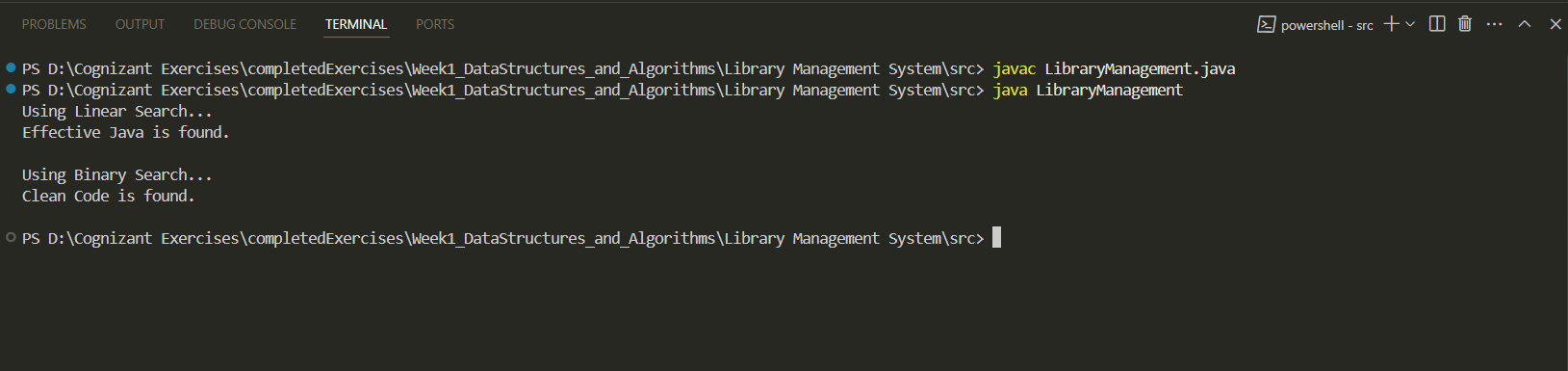
        System.out.println("Using Binary Search...");

        System.out.println(binary\_search.binarySearchAlgorithm(books, "Clean Code"));

    }

}

**Output :**

****

**Exercise 7: Financial Forecasting**

import java.util.Scanner;

public class FinancialForecasting {

    public static *double* predictFutureValue(*double* *present\_value*, *double* *rate*, *double* *years*) {

        if(*years* == 0) {

            return *present\_value*;

        } else {

            return predictFutureValue(*present\_value* \* (1 + *rate*), *rate*, *years* - 1);

        }

    }

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

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter your present value : ");

*double* present\_value = sc.nextDouble();

        System.out.println("Enter your previous value : ");

*double* rate = sc.nextDouble();

        System.out.println("Enter your time period : ");

*double* years = sc.nextDouble();

*double* future\_value = predictFutureValue(present\_value, rate, years);

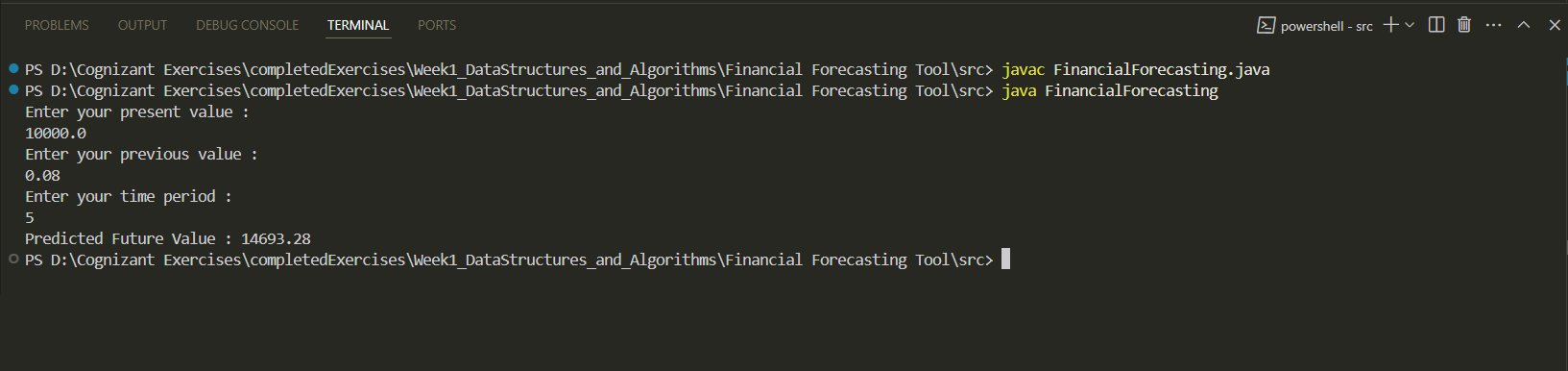
        System.out.format("Predicted Future Value : %.2f\n", future\_value);

        sc.close();

    }

}

**Output :**

****