**DESIGN AND PATTERN PRINCIPLES**

**Exercise 1: Implementing the Singleton Pattern**

Logger.java

public class Logger {

    private static Logger instance;

    private Logger() {

        System.out.println("Logger Initialized");

    }

    public static Logger getInstance() {

        if (instance == null) {

            instance = new Logger();

        }

        return instance;

    }

    public void log(String message) {

        System.out.println("Log: " + message);

    }

}

SingletonTest.java

public class SingletonTest {

    public static void main(String[] args) {

        Logger logger1 = Logger.getInstance();

        logger1.log("First log");

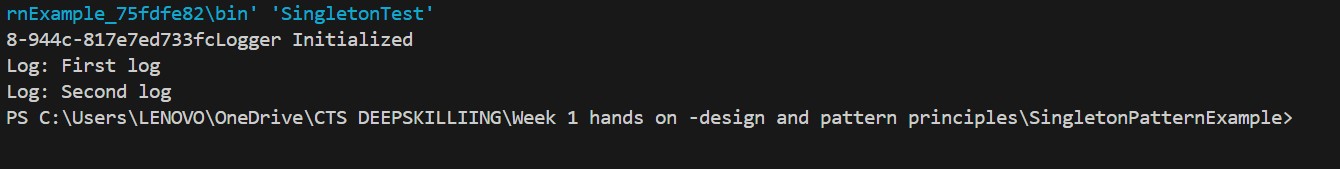
        Logger logger2 = Logger.getInstance();

        logger2.log("Second log");

    }

}

**OUTPUT**



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

Document.java

interface Document {

    void open();

}

WordDocument.java

class WordDocument implements Document {

    public void open() {

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

    }

}

PdfDocument.java

class PdfDocument implements Document {

    public void open() {

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

    }

}

ExcelDocument.java

class ExcelDocument implements Document {

    public void open() {

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

    }

}

DocumentFactory.java

abstract class DocumentFactory {

    abstract Document createDocument();

}

WordFactory.java

class WordFactory extends DocumentFactory {

    public Document createDocument() {

        return new WordDocument();

    }

}

PdfFactory.java

class PdfFactory extends DocumentFactory {

    public Document createDocument() {

        return new PdfDocument();

    }

}

ExcelFactory.java

class ExcelFactory extends DocumentFactory {

    public Document createDocument() {

        return new ExcelDocument();

    }

}

FactoryTest.java

public class FactoryTest {

    public static void main(String[] args) {

        DocumentFactory factory = new WordFactory();

        Document doc1 = factory.createDocument();

        doc1.open();

        factory = new PdfFactory();

        Document doc2 = factory.createDocument();

        doc2.open();

        factory = new ExcelFactory();

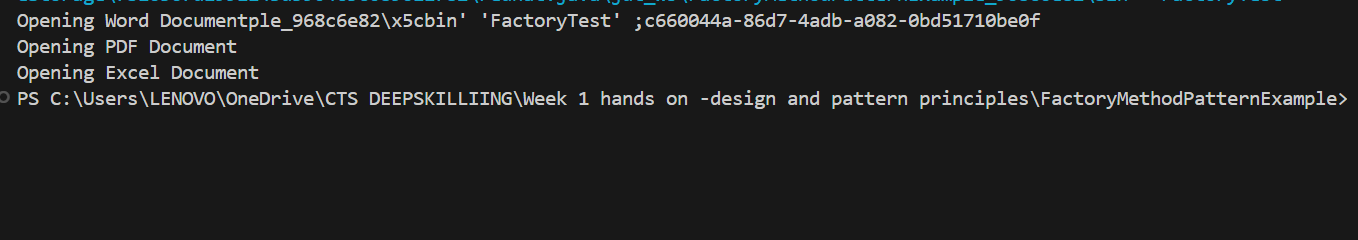
        Document doc3 = factory.createDocument();

        doc3.open();

    }

}

**OUTPUT**



**Exercise 3: Implementing the Builder Pattern**

BuilderTest.java

public class BuilderTest {

    public static void main(String[] args) {

        Computer comp1 = new Computer.Builder()

                .setCPU("Intel i5")

                .setRAM("8GB")

                .setStorage("512GB SSD")

                .build();

        comp1.showConfig();

    }

}

**Computer.java**

class Computer {

    private String CPU;

    private String RAM;

    private String storage;

    private Computer(Builder builder) {

        this.CPU = builder.CPU;

        this.RAM = builder.RAM;

        this.storage = builder.storage;

    }

    public static class Builder {

        private String CPU;

        private String RAM;

        private String storage;

        public Builder setCPU(String CPU) {

            this.CPU = CPU;

            return this;

        }

        public Builder setRAM(String RAM) {

            this.RAM = RAM;

            return this;

        }

        public Builder setStorage(String storage) {

            this.storage = storage;

            return this;

        }

        public Computer build() {

            return new Computer(this);

        }

    }

    public void showConfig() {

        System.out.println("CPU: " + CPU);

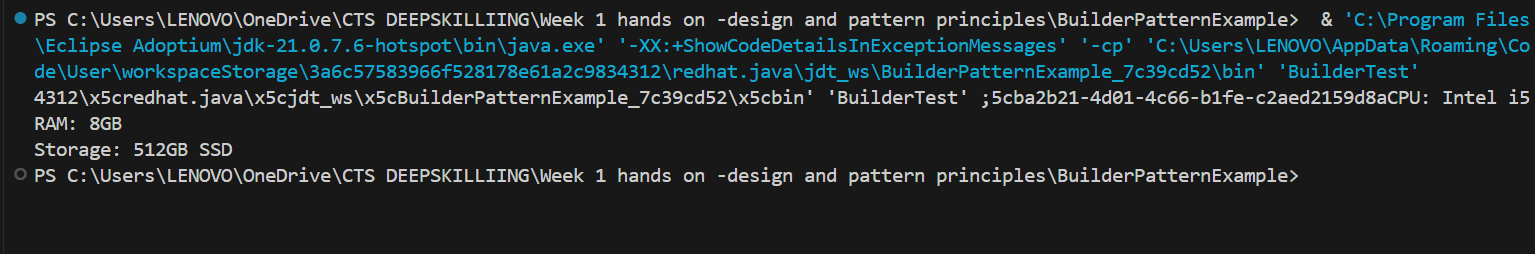
        System.out.println("RAM: " + RAM);

        System.out.println("Storage: " + storage);

    }

}

**OUTPUT**



**Exercise 4: Implementing the Adapter Pattern**

AdapterTest.java

public class AdapterTest {

    public static void main(String[] args) {

        OldGateway oldGateway = new OldGateway();

        PaymentProcessor adapter = new NewGatewayAdapter(oldGateway);

        adapter.processPayment(500.0);

    }

}

NewGatewayAdapter.java

class NewGatewayAdapter implements PaymentProcessor {

    private OldGateway oldGateway;

    public NewGatewayAdapter(OldGateway oldGateway) {

        this.oldGateway = oldGateway;

    }

    public void processPayment(double amount) {

        oldGateway.makePayment(amount);

    }

}

OldGateway.java

class OldGateway {

    public void makePayment(double amount) {

        System.out.println("Payment of Rs." + amount + " processed by OldGateway");

    }

}

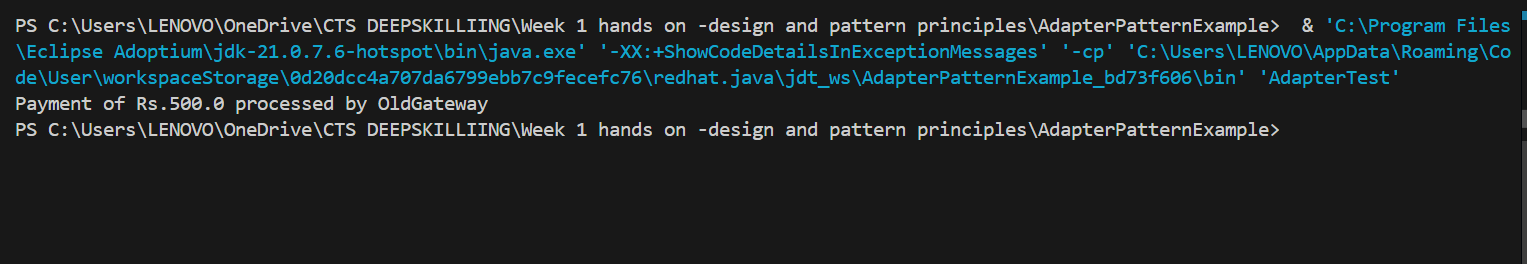
PaymaentProcessor.java

interface PaymentProcessor {

    void processPayment(double amount);

}

**OUTPUT**



**Exercise 5: Implementing the Decorator Pattern**

DecoratorTest.java

public class DecoratorTest {

    public static void main(String[] args) {

        Notifier notifier = new EmailNotifier();

        Notifier smsNotifier = new SMSNotifierDecorator(notifier);

        Notifier slackNotifier = new SlackNotifierDecorator(smsNotifier);

        slackNotifier.send("System Alert!");

    }

}

EmailNotifier.java

class EmailNotifier implements Notifier {

    public void send(String message) {

        System.out.println("Email sent: " + message);

    }

}

Notifier.java

interface Notifier {

    void send(String message);

}

NotifierDecorator.java

abstract class NotifierDecorator implements Notifier {

    protected Notifier notifier;

    public NotifierDecorator(Notifier notifier) {

        this.notifier = notifier;

    }

    public void send(String message) {

        notifier.send(message);

    }

}

SlackNotifierDecorator.java

class SlackNotifierDecorator extends NotifierDecorator {

    public SlackNotifierDecorator(Notifier notifier) {

        super(notifier);

    }

    public void send(String message) {

        super.send(message);

        System.out.println("Slack message sent: " + message);

    }

}

SMSNotifierDecorator.java

class SMSNotifierDecorator extends NotifierDecorator {

    public SMSNotifierDecorator(Notifier notifier) {

        super(notifier);

    }

    public void send(String message) {

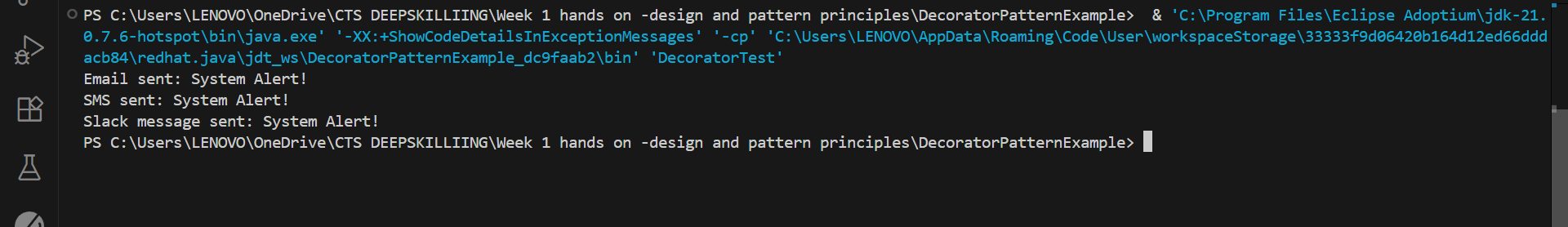
        super.send(message);

        System.out.println("SMS sent: " + message);

    }

}

**OUTPUT**



**Exercise 6: Implementing the Proxy Pattern**

Image.java

interface Image {

    void display();

}

ProxyImage.java

class ProxyImage implements Image {

    private RealImage realImage;

    private String filename;

    public ProxyImage(String filename) {

        this.filename = filename;

    }

    public void display() {

        if (realImage == null) {

            realImage = new RealImage(filename);

        }

        realImage.display();

    }

}

Proxytest.java

public class ProxyTest {

    public static void main(String[] args) {

        Image image = new ProxyImage("dog.jpeg");

        image.display();

        image.display();

    }

}

RealImage.java

class RealImage implements Image {

    private String filename;

    public RealImage(String filename) {

        this.filename = filename;

        loadFromDisk();

    }

    private void loadFromDisk() {

        System.out.println("Loading " + filename);

    }

    public void display() {

        System.out.println("Displaying " + filename);

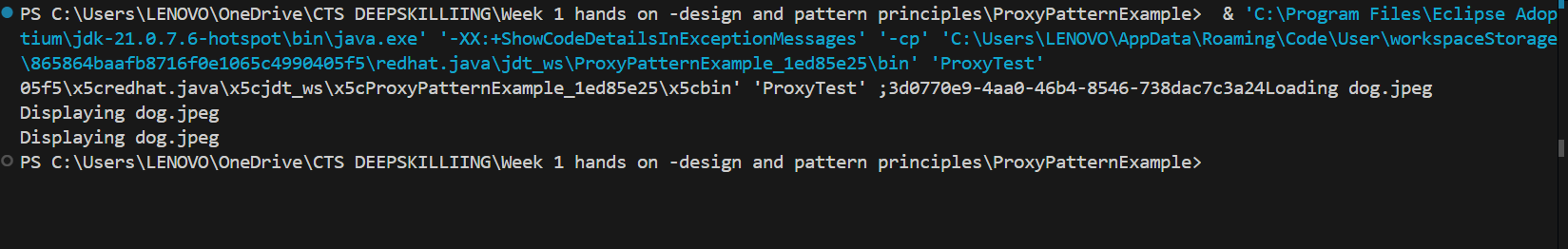
    }

}

dog.jpeg



**OUTPUT**



**Exercise 7: Implementing the Observer Pattern**

MobileApp.java

class MobileApp implements Observer {

    public void update(float price) {

        System.out.println("Mobile App: New stock price = Rs." + price);

    }

}

Observer.java

interface Observer {

    void update(float price);

}

ObserverTest.java

public class ObserverTest {

    public static void main(String[] args) {

        StockMarket market = new StockMarket();

        Observer mobile = new MobileApp();

        Observer web = new WebApp();

        market.register(mobile);

        market.register(web);

        market.setPrice(101.5f);

    }

}

Stock.java

interface Stock {

    void register(Observer o);

    void unregister(Observer o);

    void notifyObservers();

}

StockMarket.java

import java.util.\*;

class StockMarket implements Stock {

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

    private float price;

    public void register(Observer o) {

        observers.add(o);

    }

    public void unregister(Observer o) {

        observers.remove(o);

    }

    public void setPrice(float price) {

        this.price = price;

        notifyObservers();

    }

    public void notifyObservers() {

        for (Observer o : observers) {

            o.update(price);

        }

    }

}

WebApp.java

class WebApp implements Observer {

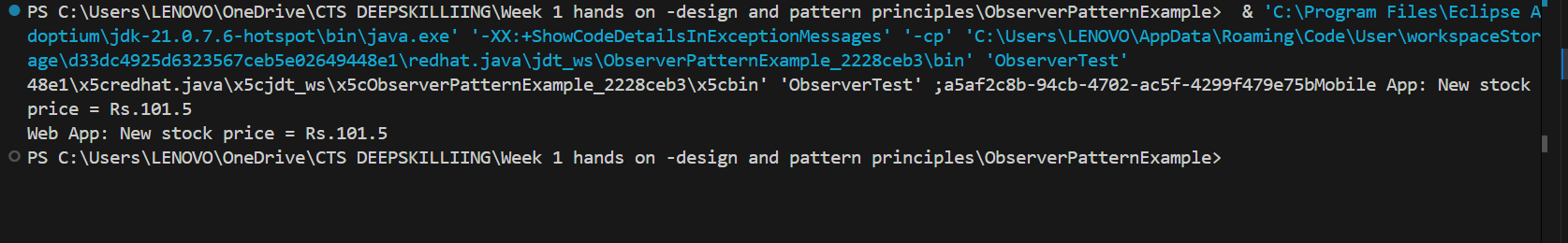
    public void update(float price) {

        System.out.println("Web App: New stock price = Rs." + price);

    }

}

**OUTPUT**



**Exercise 8: Implementing the Strategy Pattern**

CreditCardPayment.java

class CreditCardPayment implements PaymentStrategy {

    public void pay(double amount) {

        System.out.println("Paid Rs." + amount + " using Credit Card");

    }

}

PaymentContext.java

class PaymentContext {

    private PaymentStrategy strategy;

    public void setStrategy(PaymentStrategy strategy) {

        this.strategy = strategy;

    }

    public void pay(double amount) {

        strategy.pay(amount);

    }

}

PaymentStrategy.java

interface PaymentStrategy {

    void pay(double amount);

}

PayPalPayment.java

class PayPalPayment implements PaymentStrategy {

    public void pay(double amount) {

        System.out.println("Paid Rs." + amount + " using PayPal");

    }

}

StrategyTest.java

public class StrategyTest {

    public static void main(String[] args) {

        PaymentContext context = new PaymentContext();

        context.setStrategy(new CreditCardPayment());

        context.pay(300);

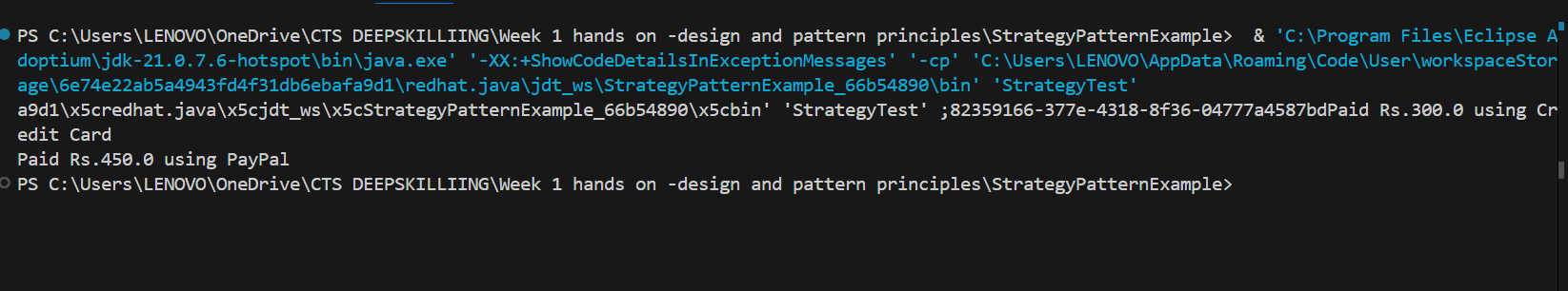
        context.setStrategy(new PayPalPayment());

        context.pay(450);

    }

}

**OUTPUT**



**Exercise 9: Implementing the Command Pattern**

Command.java

interface Command {

    void execute();

}

CommandTest.java

public class CommandTest {

    public static void main(String[] args) {

        Light light = new Light();

        Command onCommand = new LightOnCommand(light);

        Command offCommand = new LightOffCommand(light);

        RemoteControl remote = new RemoteControl();

        remote.setCommand(onCommand);

        remote.pressButton();

        remote.setCommand(offCommand);

        remote.pressButton();

    }

}

Light.java

class Light {

    public void on() {

        System.out.println("Light is ON");

    }

    public void off() {

        System.out.println("Light is OFF");

    }

}

LightOffCommand.java

class LightOffCommand implements Command {

    private Light light;

    public LightOffCommand(Light light) {

        this.light = light;

    }

    public void execute() {

        light.off();

    }

}

LightOnCommand.java

class LightOnCommand implements Command {

    private Light light;

    public LightOnCommand(Light light) {

        this.light = light;

    }

    public void execute() {

        light.on();

    }

}

RemoteControl.java

class RemoteControl {

    private Command command;

    public void setCommand(Command command) {

        this.command = command;

    }

    public void pressButton() {

        command.execute();

    }

}

**OUTPUT**



**Exercise 10: Implementing the MVC Pattern**

MVCTest.java

public class MVCTest {

    public static void main(String[] args) {

        Student student = new Student("Anu", "101", "B");

        StudentView view = new StudentView();

        StudentController controller = new StudentController(student, view);

        controller.updateView();

        controller.setStudentGrade("A+");

        controller.updateView();

    }

}

Student.java

class Student {

    private String name;

    private String id;

    private String grade;

    public Student(String name, String id, String grade) {

        this.name = name;

        this.id = id;

        this.grade = grade;

    }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public String getId() { return id; }

    public String getGrade() { return grade; }

    public void setGrade(String grade) { this.grade = grade; }

}

StudentController.java

class StudentController {

    private Student model;

    private StudentView view;

    public StudentController(Student model, StudentView view) {

        this.model = model;

        this.view = view;

    }

    public void updateView() {

        view.displayStudentDetails(model.getName(), model.getId(), model.getGrade());

    }

    public void setStudentGrade(String grade) {

        model.setGrade(grade);

    }

}

StudentView.java

class StudentView {

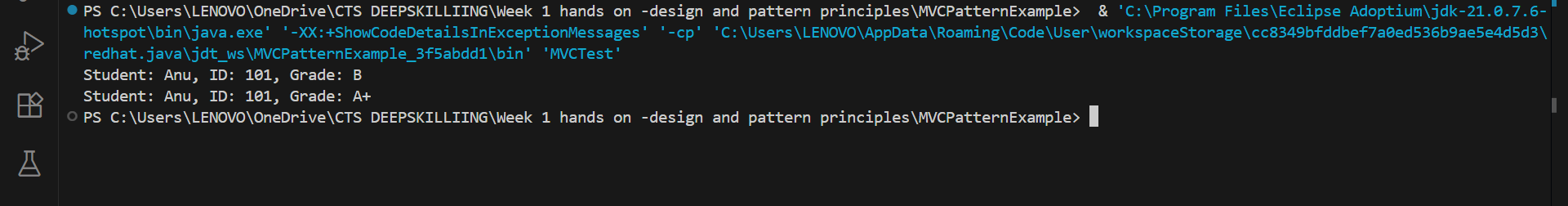
    public void displayStudentDetails(String name, String id, String grade) {

        System.out.println("Student: " + name + ", ID: " + id + ", Grade: " + grade);

    }

}

**OUTPUT**



**Exercise 11: Implementing Dependency Injection**

CustomerRespository.java

interface CustomerRepository {

    String findCustomerById(String id);

}

CustomerRespositoryImpl.java

class CustomerRepositoryImpl implements CustomerRepository {

    public String findCustomerById(String id) {

        return "Customer[" + id + "]";

    }

}

CustomerService.java

class CustomerService {

    private CustomerRepository repository;

    public CustomerService(CustomerRepository repository) {

        this.repository = repository;

    }

    public void printCustomer(String id) {

        String customer = repository.findCustomerById(id);

        System.out.println("Found: " + customer);

    }

}

DIApp.java

public class DIApp {

    public static void main(String[] args) {

        CustomerRepository repo = new CustomerRepositoryImpl();

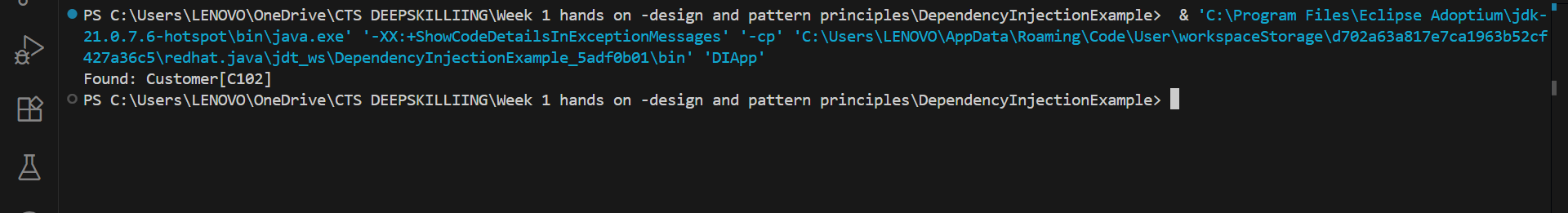
        CustomerService service = new CustomerService(repo);

        service.printCustomer("C102");

    }

}

**OUTPUT**



**DATA STRUCTURES AND ALGORITHM**

**Exercise 1 Inventory Management System**

import java.util.\*;

class Product {

    int productId;

    String productName;

    int quantity;

    double price;

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

        this.productId = productId;

        this.productName = productName;

        this.quantity = quantity;

        this.price = price;

    }

}

public class InventorySystem {

    static Map<Integer, Product> inventory = new HashMap<>();

    public static void addProduct(Product p) {

        inventory.put(p.productId, p);

    }

    public static void updateProduct(int id, int quantity, double price) {

        if (inventory.containsKey(id)) {

            Product p = inventory.get(id);

            p.quantity = quantity;

            p.price = price;

        }

    }

    public static void deleteProduct(int id) {

        inventory.remove(id);

    }

    public static void main(String[] args) {

        addProduct(new Product(1, "Laptop", 10, 55000));

        addProduct(new Product(2, "Mouse", 25, 500));

        updateProduct(2, 30, 520);

        deleteProduct(1);

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

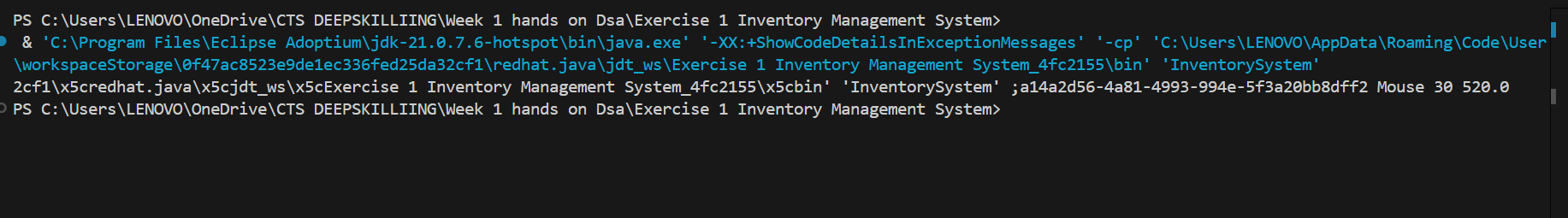
            System.out.println(p.productId + " " + p.productName + " " + p.quantity + " " + p.price);

        }

    }

}

**OUTPUT**

****

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

SearchFunction.java

import java.util.\*;

class ProductSearch {

    int productId;

    String productName;

    String category;

    public ProductSearch(int productId, String productName, String category) {

        this.productId = productId;

        this.productName = productName;

        this.category = category;

    }

}

public class SearchFunction {

    static ProductSearch[] products = {

        new ProductSearch(1, "Laptop", "Electronics"),

        new ProductSearch(2, "Keyboard", "Electronics"),

        new ProductSearch(3, "Book", "Stationery")

    };

    public static int linearSearch(String name) {

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

            if (products[i].productName.equalsIgnoreCase(name)) return i;

        }

        return -1;

    }

    public static int binarySearch(String name) {

        Arrays.sort(products, (a, b) -> a.productName.compareToIgnoreCase(b.productName));

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

        while (low <= high) {

            int mid = (low + high) / 2;

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

            if (cmp == 0) return mid;

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

            else high = mid - 1;

        }

        return -1;

    }

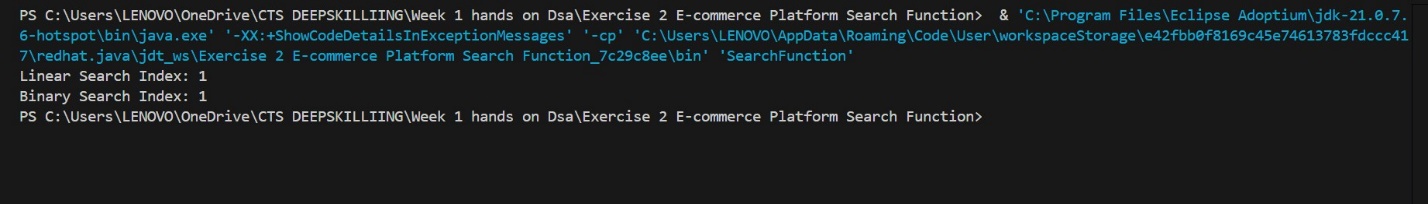
    public static void main(String[] args) {

        System.out.println("Linear Search Index: " + linearSearch("Keyboard"));

        System.out.println("Binary Search Index: " + binarySearch("Keyboard"));

    }

}

**OUTPUT**

**Exercise 3 Sorting Customer Orders**

class Order {

    int orderId;

    String customerName;

    double totalPrice;

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

        this.orderId = orderId;

        this.customerName = customerName;

        this.totalPrice = totalPrice;

    }

}

public class SortOrders {

    static Order[] orders = {

        new Order(1, "Alice", 2500),

        new Order(2, "Bob", 1500),

        new Order(3, "Charlie", 3500)

    };

    public static void quickSort(int low, int high) {

        if (low < high) {

            int pi = partition(low, high);

            quickSort(low, pi - 1);

            quickSort(pi + 1, high);

        }

    }

    public static int partition(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;

    }

    public static void printOrders() {

        for (Order o : orders)

            System.out.println(o.orderId + " " + o.customerName + " " + o.totalPrice);

    }

    public static void main(String[] args) {

        System.out.println("Before sorting:");

        printOrders();

        quickSort(0, orders.length - 1);

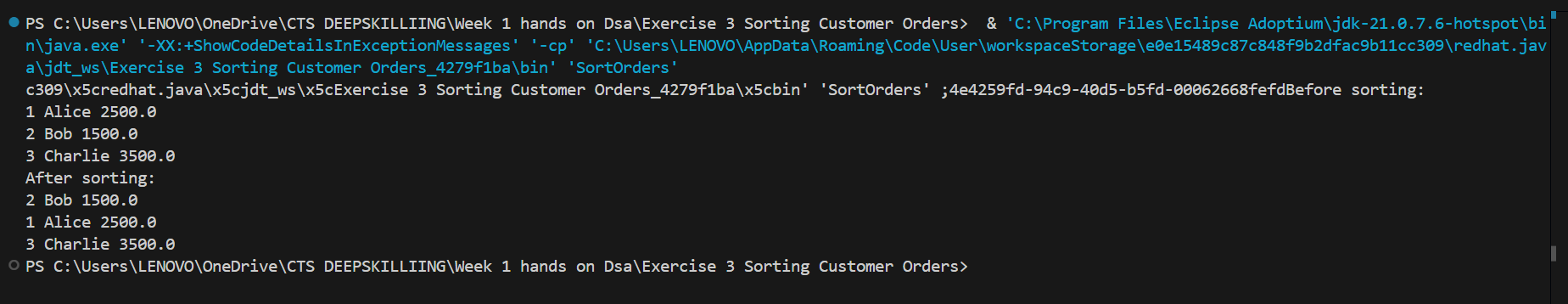
        System.out.println("After sorting:");

        printOrders();

    }

}

**OUTPUT**

****

**Exercise 4 Employee Management System**

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;

    }

}

public class EmployeeSystem {

    static Employee[] employees = new Employee[100];

    static int count = 0;

    public static void addEmployee(Employee e) {

        employees[count++] = e;

    }

    public static void searchEmployee(int id) {

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

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

                System.out.println("Found: " + employees[i].name);

                return;

            }

        }

        System.out.println("Not found");

    }

    public static 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];

                }

                count--;

                System.out.println("Deleted");

                return;

            }

        }

    }

    public static void traverseEmployees() {

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

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

        }

    }

    public static void main(String[] args) {

        addEmployee(new Employee(101, "John", "Manager", 50000));

        addEmployee(new Employee(102, "Jane", "Clerk", 30000));

        searchEmployee(101);

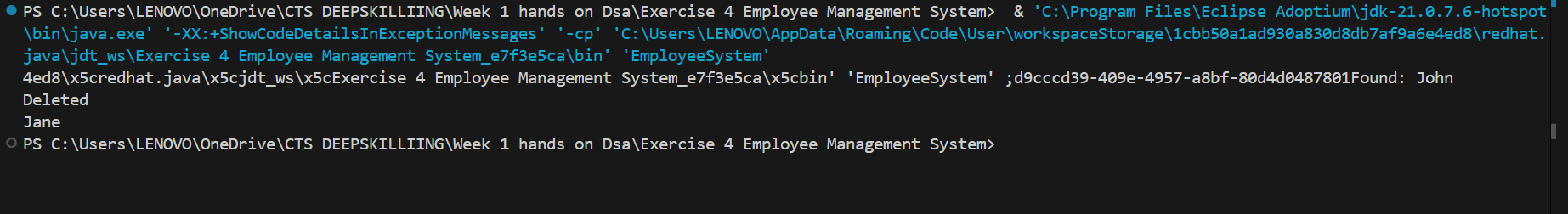
        deleteEmployee(101);

        traverseEmployees();

    }

}

**OUTPUT**

****

**Exercise 5 Task Management System**

class Task {

    int taskId;

    String taskName;

    String status;

    Task next;

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

        this.taskId = taskId;

        this.taskName = taskName;

        this.status = status;

        this.next = null;

    }

}

public class TaskManager {

    static Task head = null;

    public static void addTask(Task newTask) {

        newTask.next = head;

        head = newTask;

    }

    public static void traverseTasks() {

        Task temp = head;

        while (temp != null) {

            System.out.println(temp.taskName + " - " + temp.status);

            temp = temp.next;

        }

    }

    public static void deleteTask(int id) {

        if (head == null) return;

        if (head.taskId == id) {

            head = head.next;

            return;

        }

        Task current = head;

        while (current.next != null && current.next.taskId != id) {

            current = current.next;

        }

        if (current.next != null) {

            current.next = current.next.next;

        }

    }

    public static void main(String[] args) {

        addTask(new Task(1, "Design", "Pending"));

        addTask(new Task(2, "Development", "In Progress"));

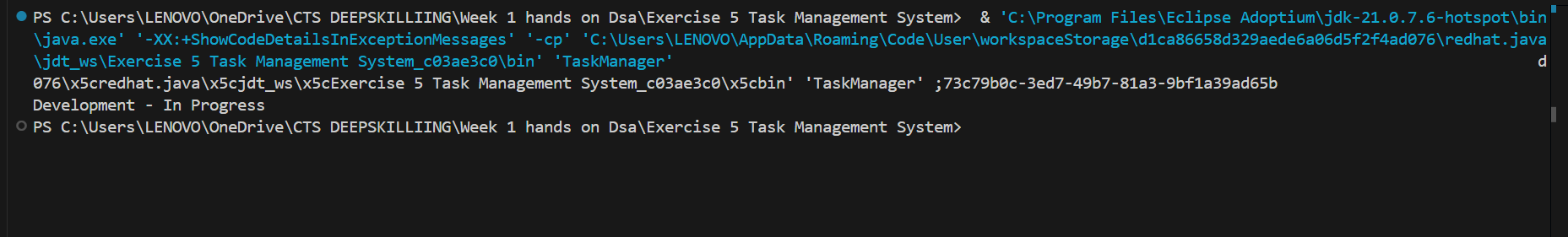
        deleteTask(1);

        traverseTasks();

    }

}

**OUTPUT**

****

**Exercise 6 Library Management System**

import java.util.\*;

class Book {

    int bookId;

    String title;

    String author;

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

        this.bookId = bookId;

        this.title = title;

        this.author = author;

    }

}

public class LibrarySystem {

    static Book[] books = {

        new Book(1, "Java Programming", "James"),

        new Book(2, "Data Structures", "Alice"),

        new Book(3, "Algorithms", "Bob")

    };

    public static int linearSearch(String title) {

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

            if (books[i].title.equalsIgnoreCase(title)) return i;

        }

        return -1;

    }

    public static int binarySearch(String title) {

        Arrays.sort(books, (a, b) -> a.title.compareToIgnoreCase(b.title));

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

        while (low <= high) {

            int mid = (low + high) / 2;

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

            if (cmp == 0) return mid;

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

            else high = mid - 1;

        }

        return -1;

    }

    public static void main(String[] args) {

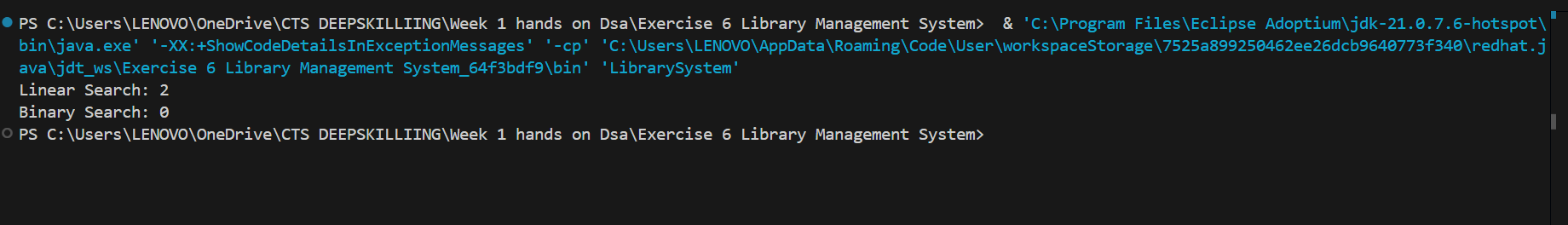
        System.out.println("Linear Search: " + linearSearch("Algorithms"));

        System.out.println("Binary Search: " + binarySearch("Algorithms"));

    }

}

**OUTPUT**

****

**Exercise 7: Financial Forecasting**

FinancialForecast.java

public class FinancialForecast {

    public static double forecastValue(double currentValue, double growthRate, int years) {

        if (years == 0) return currentValue;

        return forecastValue(currentValue \* (1 + growthRate), growthRate, years - 1);

    }

    public static void main(String[] args) {

        double futureValue = forecastValue(10000, 0.1, 5);

        System.out.println("Future Value: " + futureValue);

    }

}

OUTPUT

