COGNIZANT

Digital Nurture 4.0

Deep Skilling - Java FSE

WEEK-1 ADDITIONAL EXERCISE

By Kaviya P

DESIGN PATTERNS AND PRINCIPLES

**Exercise 3: Implementing the Builder Pattern**

**//CODE**

**//Builder class - step by step building**

**import java.util.\*;**

**import java.lang.\*;**

**class Computer {**

**private String CPU;**

**private String RAM;**

**private String Storage;**

**private String GraphicsCard;**

**private String OperatingSystem;**

**// Private constructor that takes the Builder as a parameter**

**private Computer(Builder builder) {**

**this.CPU = builder.CPU;**

**this.RAM = builder.RAM;**

**this.Storage = builder.Storage;**

**this.GraphicsCard = builder.GraphicsCard;**

**this.OperatingSystem = builder.OperatingSystem;**

**}**

**public String getCPU() {**

**return CPU;**

**}**

**public String getRAM() {**

**return RAM;**

**}**

**public String getStorage() {**

**return Storage;**

**}**

**public String getGraphicsCard() {**

**return GraphicsCard;**

**}**

**public String getOperatingSystem() {**

**return OperatingSystem;**

**}**

**@Override**

**public String toString() {**

**return "Computer [CPU=" + CPU + ", RAM=" + RAM + ", Storage=" + Storage +**

**", GraphicsCard=" + GraphicsCard + ", OperatingSystem=" + OperatingSystem + "]";**

**}**

**// Static nested Builder class**

**public static class Builder {**

**private String CPU;**

**private String RAM;**

**private String Storage;**

**private String GraphicsCard;**

**private String OperatingSystem;**

**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 Builder setGraphicsCard(String GraphicsCard) {**

**this.GraphicsCard = GraphicsCard;**

**return this;**

**}**

**public Builder setOperatingSystem(String OperatingSystem) {**

**this.OperatingSystem = OperatingSystem;**

**return this;**

**}**

**public Computer build() {**

**return new Computer(this);**

**}**

**}**

**}**

**public class Test {**

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

**Computer basicComputer = new Computer.Builder()**

**.setCPU("Intel i5")**

**.setRAM("8GB")**

**.setStorage("256GB SSD")**

**.build();**

**System.out.println("Basic Computer: " + basicComputer);**

**Computer gamingComputer = new Computer.Builder()**

**.setCPU("Intel i9")**

**.setRAM("32GB")**

**.setStorage("1TB NVMe SSD")**

**.setGraphicsCard("NVIDIA RTX 4080")**

**.setOperatingSystem("Windows 11 Pro")**

**.build();**

**System.out.println("Gaming Computer: " + gamingComputer);**

**Computer serverComputer = new Computer.Builder()**

**.setCPU("AMD EPYC")**

**.setRAM("128GB")**

**.setStorage("4TB HDD + 500GB SSD")**

**.setOperatingSystem("Linux Server")**

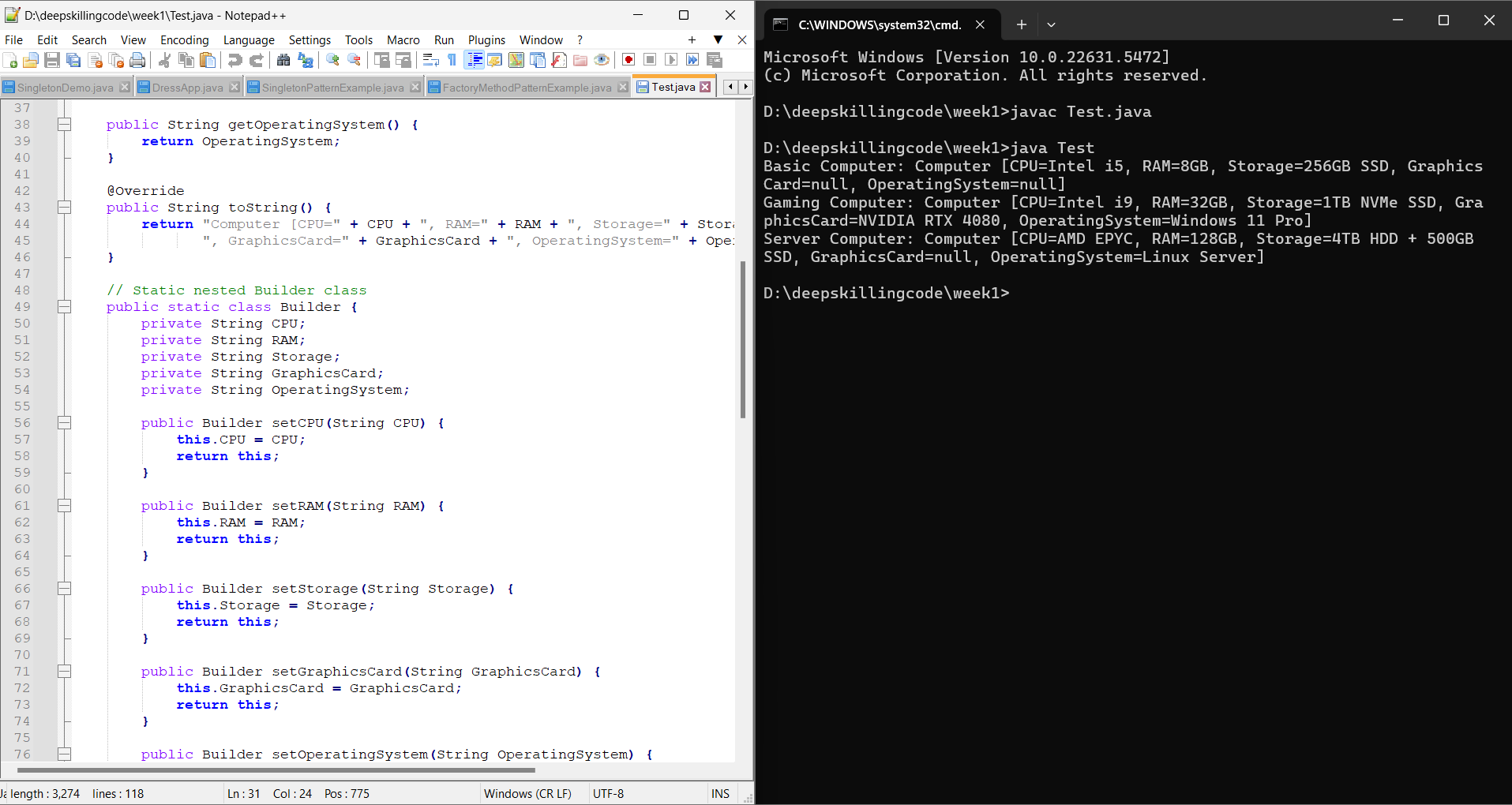
**.build();**

**System.out.println("Server Computer: " + serverComputer);**

**}**

**}**

**OUTPUT**

****

**---------------------------------------------------------------------------------------------------------**

**Exercise 4: Implementing the Adapter Pattern**

**//CODE**

**//ADAPTER PATTERN-TO build incompatible interface**

**// PaymentProcessor.java**

**// This is the Target interface that our system expects.**

**interface PaymentProcessor {**

**void processPayment(double amount);**

**}**

**// PayPalGateway.java**

**// This is an Adaptee class with its own specific interface.**

**class PayPalGateway {**

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

**System.out.println("Processing payment of $" + amount + " through PayPal Gateway.");**

**}**

**}**

**// StripeGateway.java**

**// This is another Adaptee class with its own specific interface.**

**class StripeGateway {**

**public void charge(double amount) {**

**System.out.println("Charging $" + amount + " using Stripe Gateway.");**

**}**

**}**

**// PayPalAdapter.java**

**// This Adapter makes PayPalGateway compatible with PaymentProcessor.**

**class PayPalAdapter implements PaymentProcessor {**

**private PayPalGateway payPalGateway;**

**public PayPalAdapter(PayPalGateway payPalGateway) {**

**this.payPalGateway = payPalGateway;**

**}**

**@Override**

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

**payPalGateway.makePayment(amount); // Translates to PayPal's specific method**

**}**

**}**

**// StripeAdapter.java**

**// This Adapter makes StripeGateway compatible with PaymentProcessor.**

**class StripeAdapter implements PaymentProcessor {**

**private StripeGateway stripeGateway;**

**public StripeAdapter(StripeGateway stripeGateway) {**

**this.stripeGateway = stripeGateway;**

**}**

**@Override**

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

**stripeGateway.charge(amount); // Translates to Stripe's specific method**

**}**

**}**

**//Test.java**

**class Test {**

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

**// Integrate with PayPal using the adapter**

**PayPalGateway payPalGateway = new PayPalGateway();**

**PaymentProcessor payPalProcessor = new PayPalAdapter(payPalGateway);**

**payPalProcessor.processPayment(100.50);**

**// Integrate with Stripe using the adapter**

**StripeGateway stripeGateway = new StripeGateway();**

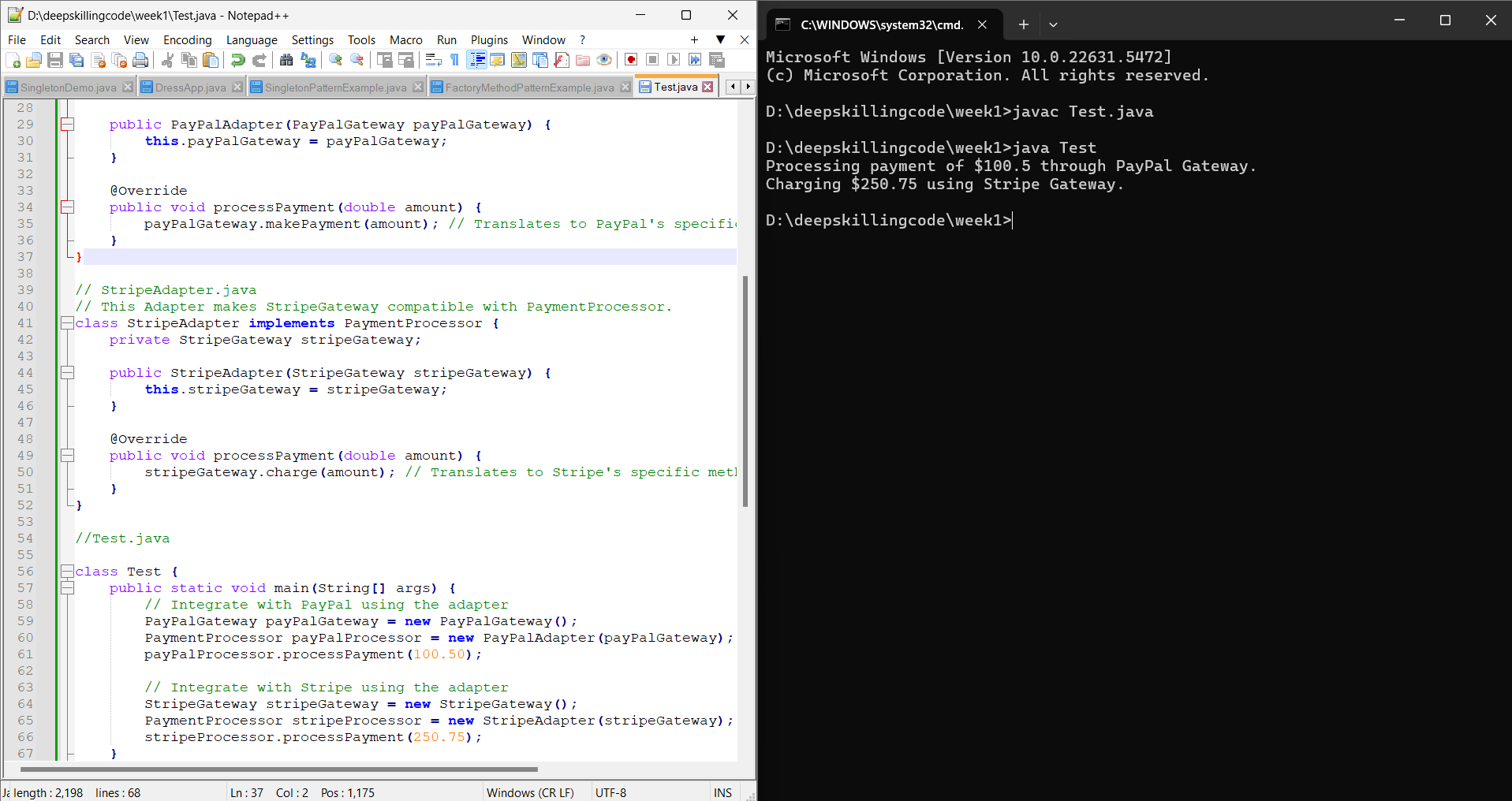
**PaymentProcessor stripeProcessor = new StripeAdapter(stripeGateway);**

**stripeProcessor.processPayment(250.75);**

**}**

**}**

**OUTPUT**

****

**---------------------------------------------------------------------------------------------------------**

**Exercise 5: Implementing the Decorator Pattern**

**//CODE**

**// Notifier.java**

**// This is the Component interface.**

**interface Notifier {**

**void send(String message);**

**}**

**// EmailNotifier.java**

**// This is the Concrete Component.**

**class EmailNotifier implements Notifier {**

**@Override**

**public void send(String message) {**

**System.out.println("Sending Email with message: " + message);**

**}**

**}**

**// NotifierDecorator.java**

**// This is the Abstract Decorator.**

**abstract class NotifierDecorator implements Notifier {**

**protected Notifier wrappedNotifier;**

**public NotifierDecorator(Notifier notifier) {**

**this.wrappedNotifier = notifier;**

**}**

**@Override**

**public void send(String message) {**

**wrappedNotifier.send(message); // Delegates to the wrapped component**

**}**

**}**

**// SMSNotifierDecorator.java**

**// This is a Concrete Decorator adding SMS functionality.**

**class SMSNotifierDecorator extends NotifierDecorator {**

**public SMSNotifierDecorator(Notifier notifier) {**

**super(notifier);**

**}**

**@Override**

**public void send(String message) {**

**super.send(message); // Call the wrapped notifier's send method**

**sendSMS(message);    // Add SMS functionality**

**}**

**private void sendSMS(String message) {**

**System.out.println("Sending SMS with message: " + message);**

**}**

**}**

**// SlackNotifierDecorator.java**

**// This is another Concrete Decorator adding Slack functionality.**

**class SlackNotifierDecorator extends NotifierDecorator {**

**public SlackNotifierDecorator(Notifier notifier) {**

**super(notifier);**

**}**

**@Override**

**public void send(String message) {**

**super.send(message);**

**sendSlackMessage(message);**

**}**

**private void sendSlackMessage(String message) {**

**System.out.println("Sending Slack message with message: " + message);**

**}**

**}**

**// Test.java**

**class Test {**

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

**// Send a basic email notification**

**Notifier emailNotifier = new EmailNotifier();**

**System.out.println("Basic Email Notification:");**

**emailNotifier.send("Hello from basic email!");**

**System.out.println();**

**// Send an email and SMS notification**

**Notifier emailAndSmsNotifier = new SMSNotifierDecorator(new EmailNotifier());**

**System.out.println("Email and SMS Notification:");**

**emailAndSmsNotifier.send("Important update!");**

**System.out.println();**

**// Send an email, SMS, and Slack notification (stacking decorators)**

**Notifier allChannelsNotifier = new SlackNotifierDecorator(new SMSNotifierDecorator(new EmailNotifier()));**

**System.out.println("Email, SMS, and Slack Notification:");**

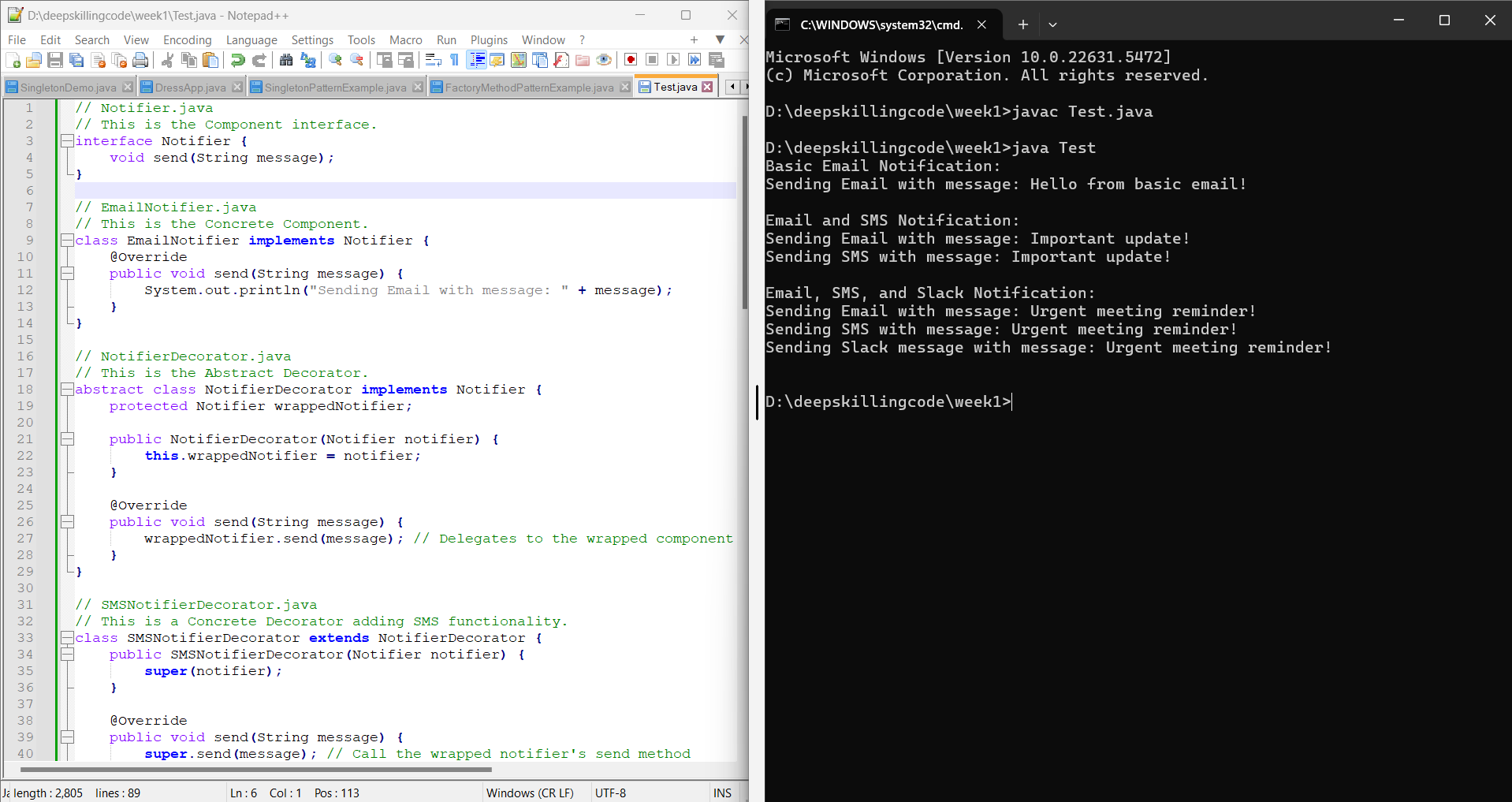
**allChannelsNotifier.send("Urgent meeting reminder!");**

**System.out.println();**

**}**

**}**

**OUTPUT**

****

**---------------------------------------------------------------------------------------------------------**

**Exercise 6: Implementing the Proxy Pattern**

**//CODE**

**import java.util.HashMap;**

**import java.util.Map;**

**// Image.java**

**// This is the Subject interface.**

**interface Image {**

**void display();**

**}**

**// RealImage.java**

**// This is the Real Subject class, which is resource-intensive.**

**class RealImage implements Image {**

**private String filename;**

**public RealImage(String filename) {**

**this.filename = filename;**

**loadFromServer(); // Simulates heavy operation**

**}**

**private void loadFromServer() {**

**System.out.println("Loading image: " + filename + " from remote server...");**

**try {**

**Thread.sleep(2000); // Simulate network delay**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

**}**

**System.out.println("Image " + filename + " loaded.");**

**}**

**@Override**

**public void display() {**

**System.out.println("Displaying image: " + filename);**

**}**

**}**

**// ProxyImage.java**

**// This is the Proxy class, controlling access and adding lazy loading/caching.**

**class ProxyImage implements Image {**

**private String filename;**

**private RealImage realImage;**

**private static Map<String, RealImage> imageCache = new HashMap<>();**

**public ProxyImage(String filename) {**

**this.filename = filename;**

**}**

**@Override**

**public void display() {**

**if (realImage == null) {**

**// Check cache first**

**if (imageCache.containsKey(filename)) {**

**System.out.println("Retrieving " + filename + " from cache.");**

**realImage = imageCache.get(filename);**

**} else {**

**realImage = new RealImage(filename);**

**imageCache.put(filename, realImage);**

**}**

**}**

**realImage.display();**

**}**

**}**

**// ProxyPatternTest.java**

**// This file demonstrates the usage of the Proxy Pattern.**

**class Test {**

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

**System.out.println("First attempt to display image1:");**

**Image image1 = new ProxyImage("photo\_1.jpg");**

**image1.display();**

**System.out.println("\nSecond attempt to display image1:");**

**image1.display();**

**System.out.println("\nFirst attempt to display image2:");**

**Image image2 = new ProxyImage("document\_2.png");**

**image2.display();**

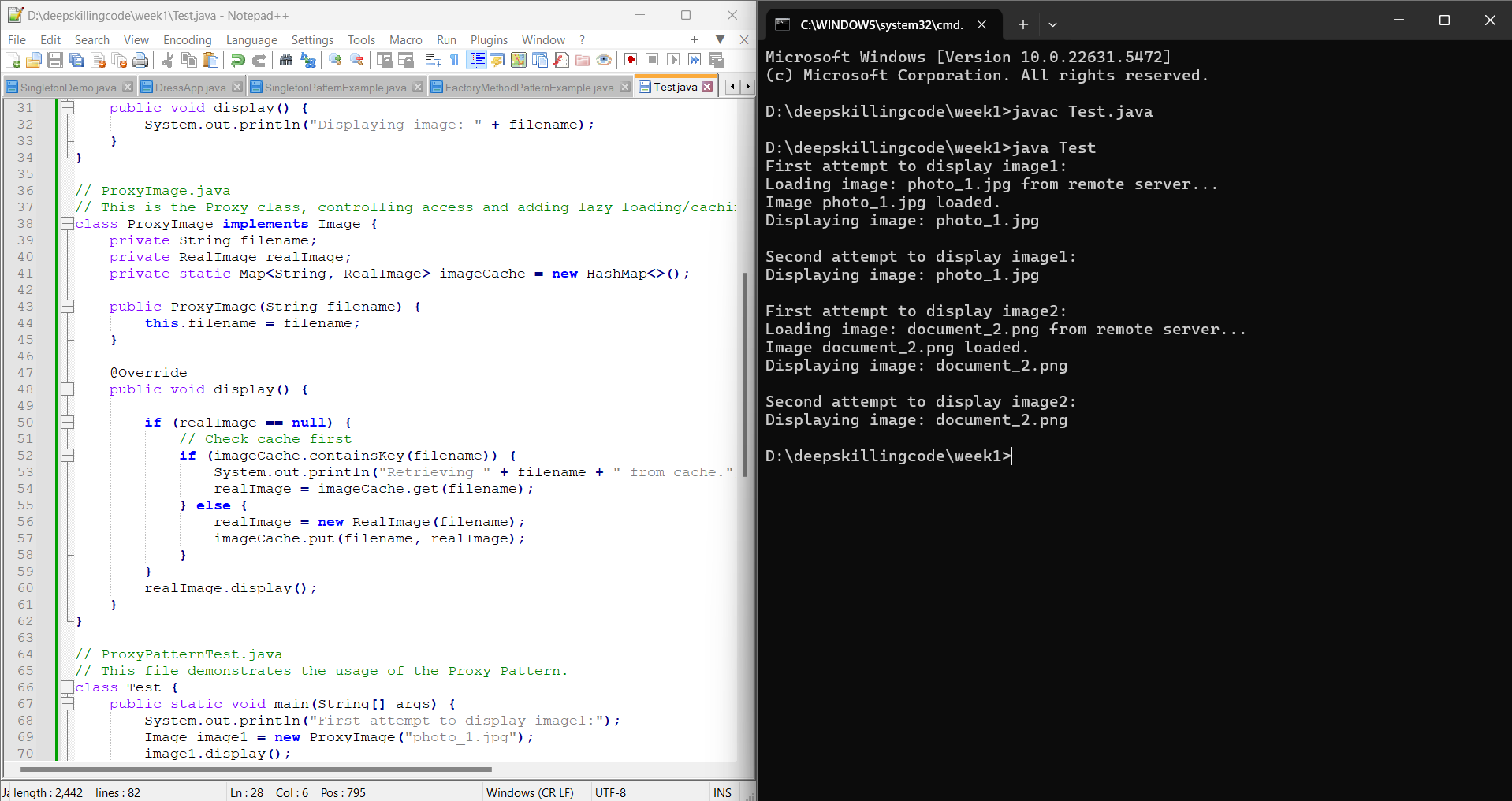
**System.out.println("\nSecond attempt to display image2:");**

**image2.display();**

**}**

**}**

**OUTPUT**

****

**---------------------------------------------------------------------------------------------------------**

**//CODE**

**Exercise 7: Implementing the Observer Pattern**

**import java.util.ArrayList;**

**import java.util.List;**

**// Stock.java**

**interface Stock {**

**void registerObserver(Observer o);**

**void deregisterObserver(Observer o);**

**void notifyObservers();**

**}**

**// Observer.java**

**interface Observer {**

**void update(String stockName, double price);**

**}**

**// StockMarket.java**

**class StockMarket implements Stock {**

**private List<Observer> observers;**

**private String stockName;**

**private double price;**

**public StockMarket(String stockName) {**

**this.stockName = stockName;**

**this.observers = new ArrayList<>();**

**}**

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

**System.out.println("\n--- " + stockName + " Price Change ---");**

**this.price = price;**

**notifyObservers();**

**}**

**public String getStockName() {**

**return stockName;**

**}**

**public double getPrice() {**

**return price;**

**}**

**@Override**

**public void registerObserver(Observer o) {**

**observers.add(o);**

**System.out.println(o.getClass().getSimpleName() + " registered for " + stockName);**

**}**

**@Override**

**public void deregisterObserver(Observer o) {**

**observers.remove(o);**

**System.out.println(o.getClass().getSimpleName() + " deregistered from " + stockName);**

**}**

**@Override**

**public void notifyObservers() {**

**for (Observer observer : observers) {**

**observer.update(stockName, price);**

**}**

**}**

**}**

**// MobileApp.java**

**class MobileApp implements Observer {**

**private String appName;**

**public MobileApp(String appName) {**

**this.appName = appName;**

**}**

**@Override**

**public void update(String stockName, double price) {**

**System.out.println(appName + " (Mobile App): " + stockName + " new price is $" + price);**

**}**

**}**

**// WebApp.java**

**class WebApp implements Observer {**

**private String appName;**

**public WebApp(String appName) {**

**this.appName = appName;**

**}**

**@Override**

**public void update(String stockName, double price) {**

**System.out.println(appName + " (Web App): " + stockName + " new price is $" + price);**

**}**

**}**

**//Test.java**

**class Test {**

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

**StockMarket googleStock = new StockMarket("GOOGL");**

**StockMarket appleStock = new StockMarket("AAPL");**

**MobileApp mobileApp1 = new MobileApp("MyStockTracker");**

**WebApp webApp1 = new WebApp("StockDashboard");**

**MobileApp mobileApp2 = new MobileApp("TraderPro");**

**googleStock.registerObserver(mobileApp1);**

**googleStock.registerObserver(webApp1);**

**appleStock.registerObserver(webApp1);**

**appleStock.registerObserver(mobileApp2);**

**googleStock.setPrice(1500.75);**

**appleStock.setPrice(175.20);**

**googleStock.deregisterObserver(mobileApp1);**

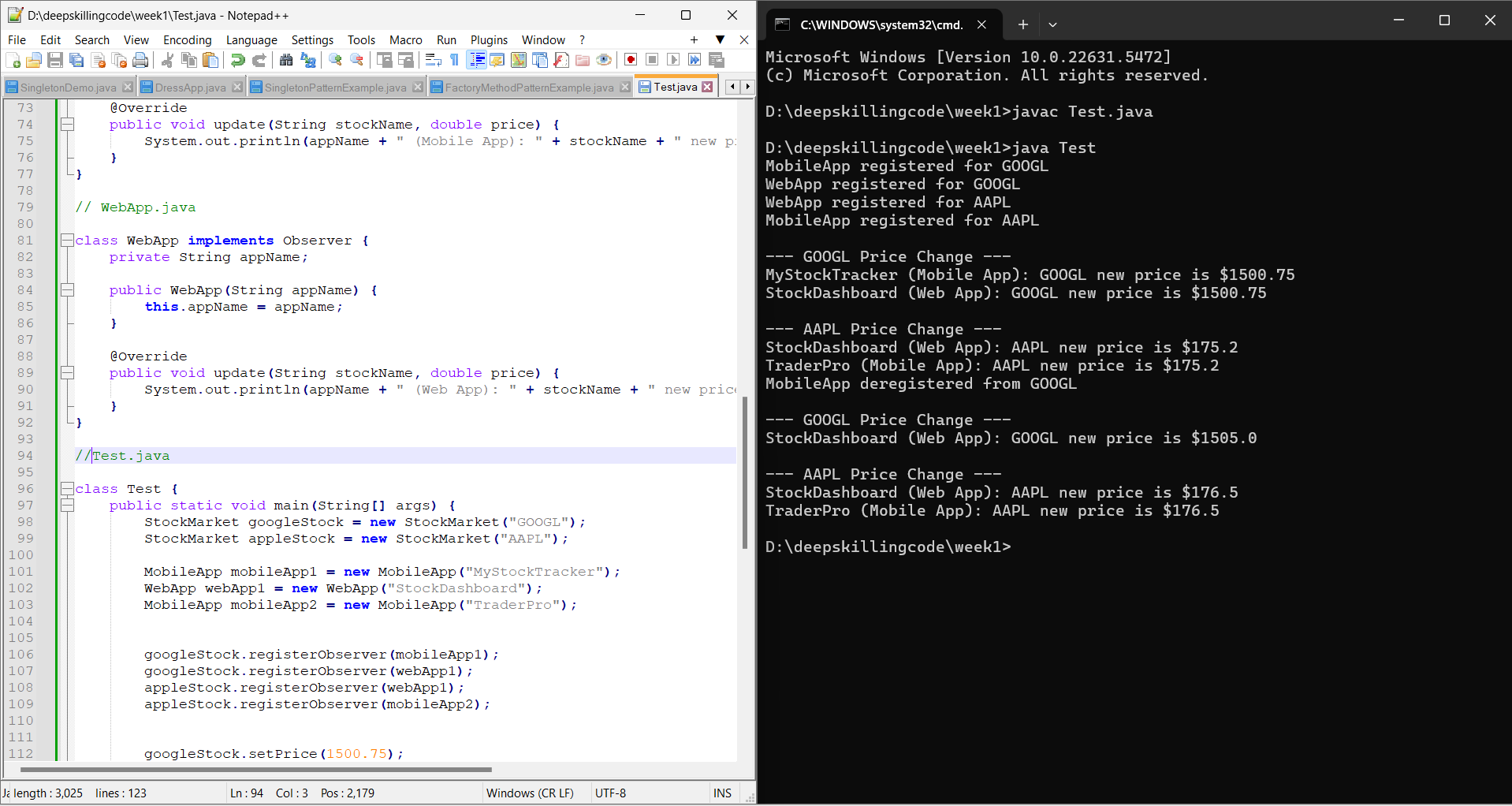
**googleStock.setPrice(1505.00);**

**appleStock.setPrice(176.50);**

**}**

**}**

**OUTPUT**

****

**---------------------------------------------------------------------------------------------------------**

**//CODE**

**Exercise 8: Implementing the Strategy Pattern**

**// PaymentStrategy.java**

**interface PaymentStrategy {**

**void pay(double amount);**

**}**

**// CreditCardPayment.java**

**class CreditCardPayment implements PaymentStrategy {**

**private String cardNumber;**

**private String nameOnCard;**

**public CreditCardPayment(String cardNumber, String nameOnCard) {**

**this.cardNumber = cardNumber;**

**this.nameOnCard = nameOnCard;**

**}**

**@Override**

**public void pay(double amount) {**

**System.out.println("Paying $" + amount + " using Credit Card (" + cardNumber + ")");**

**// Simulate credit card processing logic**

**}**

**}**

**// PayPalPayment.java**

**class PayPalPayment implements PaymentStrategy {**

**private String email;**

**public PayPalPayment(String email) {**

**this.email = email;**

**}**

**@Override**

**public void pay(double amount) {**

**System.out.println("Paying $" + amount + " using PayPal (Email: " + email + ")");**

**}**

**}**

**// PaymentContext.java**

**// This is the Context class, which uses a PaymentStrategy.**

**class PaymentContext {**

**private PaymentStrategy paymentStrategy;**

**public void setPaymentStrategy(PaymentStrategy paymentStrategy) {**

**this.paymentStrategy = paymentStrategy;**

**}**

**public void executePayment(double amount) {**

**if (paymentStrategy == null) {**

**System.out.println("No payment strategy selected.");**

**return;**

**}**

**paymentStrategy.pay(amount);**

**}**

**}**

**class Test {**

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

**PaymentContext context = new PaymentContext();**

**context.setPaymentStrategy(new CreditCardPayment("1234-5678-9012-3456", "John Doe"));**

**context.executePayment(150.75);**

**context.setPaymentStrategy(new PayPalPayment("john.doe@example.com"));**

**context.executePayment(75.20);**

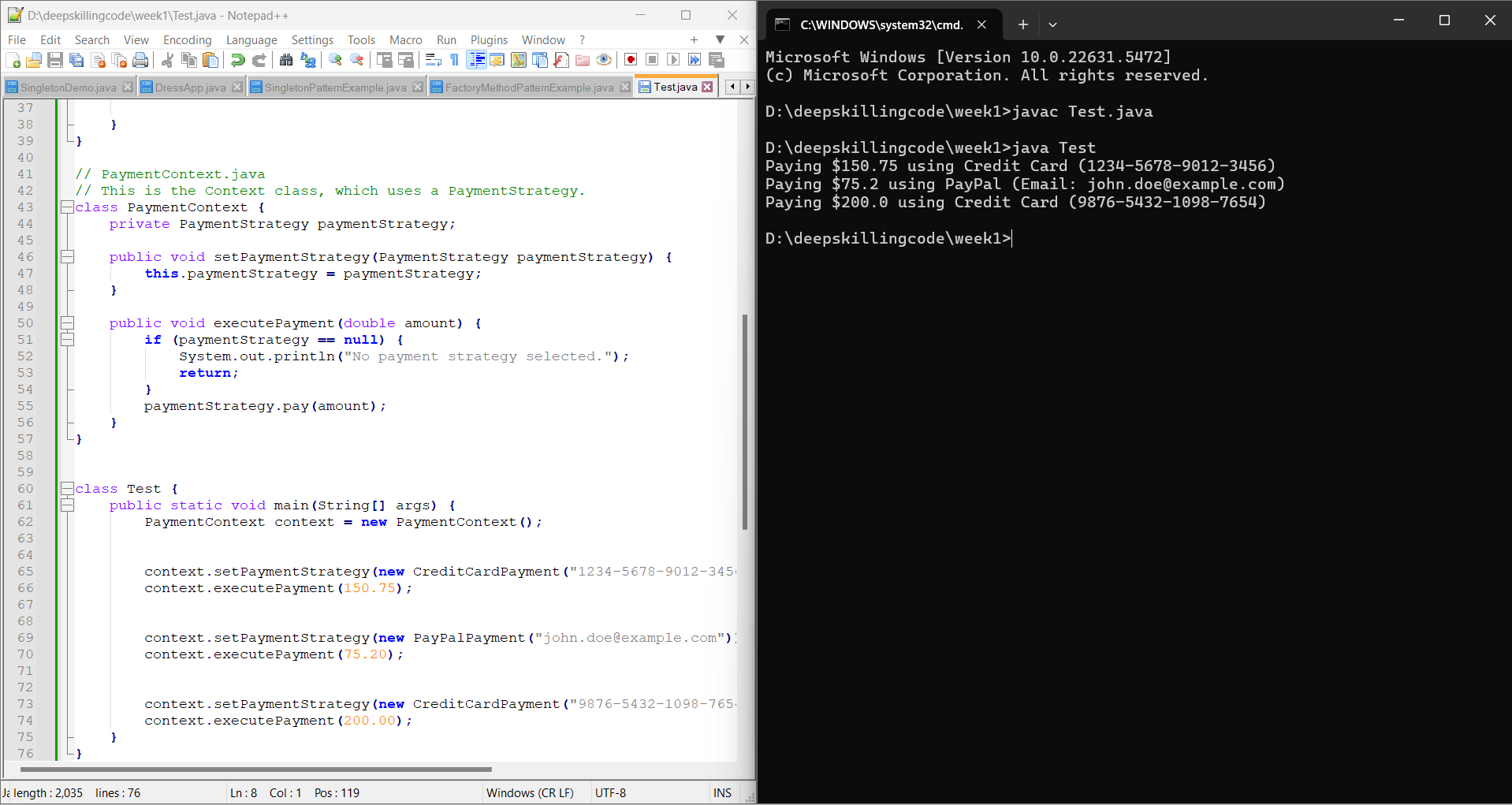
**context.setPaymentStrategy(new CreditCardPayment("9876-5432-1098-7654", "Jane Smith"));**

**context.executePayment(200.00);**

**}**

**}**

**OUTPUT**

****

**---------------------------------------------------------------------------------------------------------**

**Exercise 9: Implementing the Command Pattern**

**//CODE**

**// Command.java**

**interface Command {**

**void execute();**

**}**

**// Light.java**

**class Light {**

**private String location;**

**public Light(String location) {**

**this.location = location;**

**}**

**public void turnOn() {**

**System.out.println(location + " light is ON");**

**}**

**public void turnOff() {**

**System.out.println(location + " light is OFF");**

**}**

**}**

**// LightOnCommand.java**

**class LightOnCommand implements Command {**

**private Light light; // Reference to the Receiver**

**public LightOnCommand(Light light) {**

**this.light = light;**

**}**

**@Override**

**public void execute() {**

**light.turnOn(); // Calls the action on the Receiver**

**}**

**}**

**// LightOffCommand.java**

**class LightOffCommand implements Command {**

**private Light light; // Reference to the Receiver**

**public LightOffCommand(Light light) {**

**this.light = light;**

**}**

**@Override**

**public void execute() {**

**light.turnOff();**

**}**

**}**

**// RemoteControl.java**

**class RemoteControl {**

**private Command command;**

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

**this.command = command;**

**}**

**public void pressButton() {**

**if (command != null) {**

**command.execute();**

**} else {**

**System.out.println("No command set for this button.");**

**}**

**}**

**}**

**// Test.java**

**class Test {**

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

**Light livingRoomLight = new Light("Living Room");**

**Light kitchenLight = new Light("Kitchen");**

**Command livingRoomLightOn = new LightOnCommand(livingRoomLight);**

**Command livingRoomLightOff = new LightOffCommand(livingRoomLight);**

**Command kitchenLightOn = new LightOnCommand(kitchenLight);**

**Command kitchenLightOff = new LightOffCommand(kitchenLight);**

**RemoteControl remote = new RemoteControl();**

**System.out.println("--- Living Room Control ---");**

**remote.setCommand(livingRoomLightOn);**

**remote.pressButton();**

**remote.setCommand(livingRoomLightOff);**

**remote.pressButton();**

**System.out.println("\n--- Kitchen Control ---");**

**remote.setCommand(kitchenLightOn);**

**remote.pressButton();**

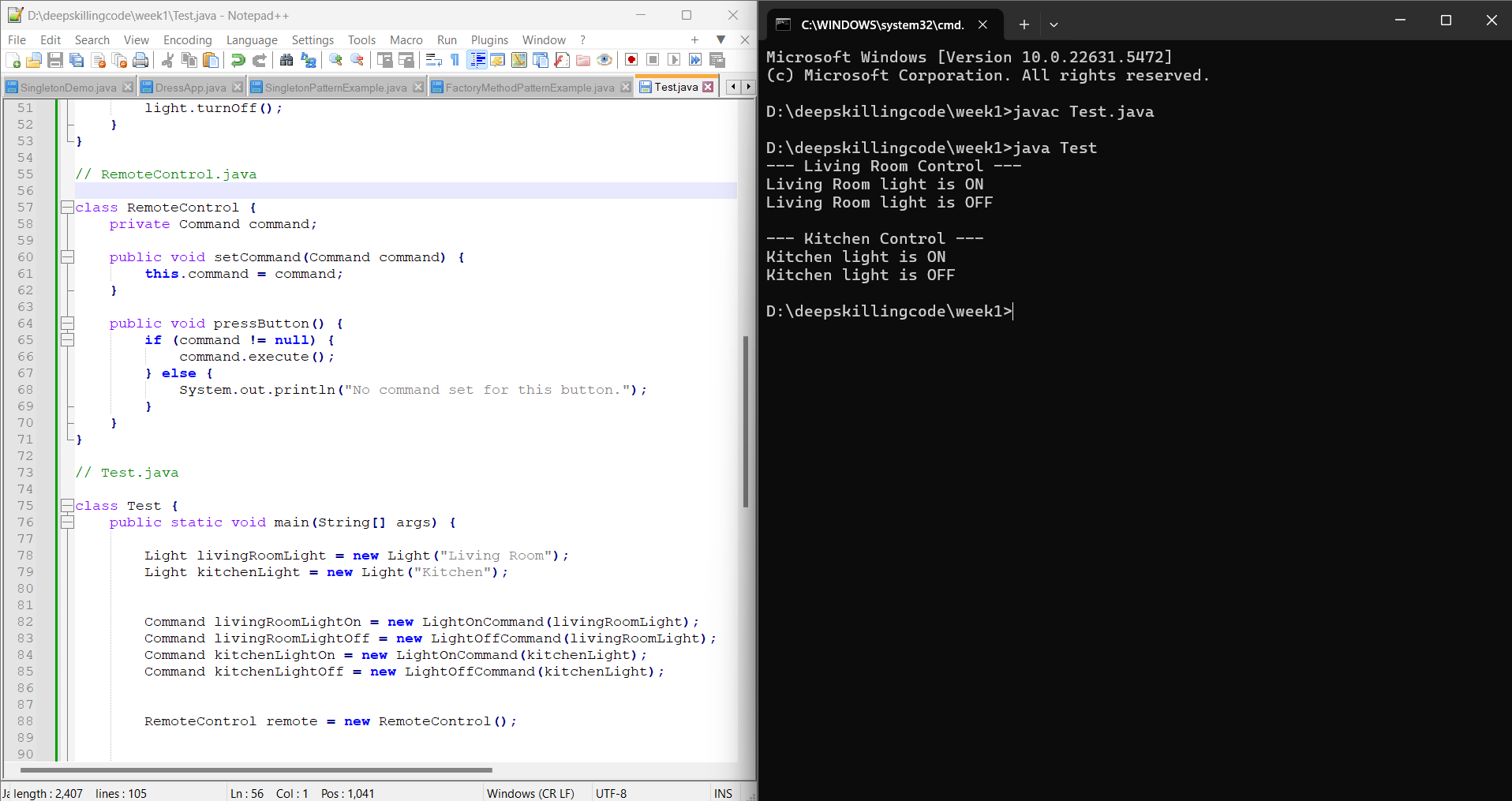
**remote.setCommand(kitchenLightOff);**

**remote.pressButton();**

**}**

**}**

**OUTPUT**

****

**---------------------------------------------------------------------------------------------------------**

**Exercise 10: Implementing the MVC Pattern**

**//CODE**

**// Student.java**

**// This is the Model class, representing the data.**

**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 void setId(String id) {**

**this.id = id;**

**}**

**public String getGrade() {**

**return grade;**

**}**

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

**this.grade = grade;**

**}**

**}**

**// StudentView.java**

**// This is the View class, responsible for displaying data.**

**class StudentView {**

**public void displayStudentDetails(String studentName, String studentId, String studentGrade) {**

**System.out.println("--- Student Details ---");**

**System.out.println("Name: " + studentName);**

**System.out.println("ID: " + studentId);**

**System.out.println("Grade: " + studentGrade);**

**}**

**}**

**// StudentController.java**

**// This is the Controller class, handling logic and mediating between Model and View.**

**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 String getStudentName() {**

**return model.getName();**

**}**

**public void setStudentId(String id) {**

**model.setId(id);**

**}**

**public String getStudentId() {**

**return model.getId();**

**}**

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

**model.setGrade(grade);**

**}**

**public String getStudentGrade() {**

**return model.getGrade();**

**}**

**// Method to update the view based on the model's current state**

**public void updateView() {**

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

**}**

**}**

**//test.java**

**// This file is the main entry point to demonstrate the MVC pattern.**

**public class Test {**

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

**// 1. Create a Student model (data)**

**Student student = new Student("Alice Johnson", "S101", "A");**

**// 2. Create a StudentView (presentation)**

**StudentView view = new StudentView();**

**// 3. Create a StudentController, linking Model and View**

**StudentController controller = new StudentController(student, view);**

**// Display initial student details through the controller**

**System.out.println("Initial Student Details:");**

**controller.updateView();**

**// Update student details via the controller (which modifies the model)**

**controller.setStudentName("Bob Williams");**

**controller.setStudentGrade("B+");**

**// Display updated student details through the controller (which refreshes the view)**

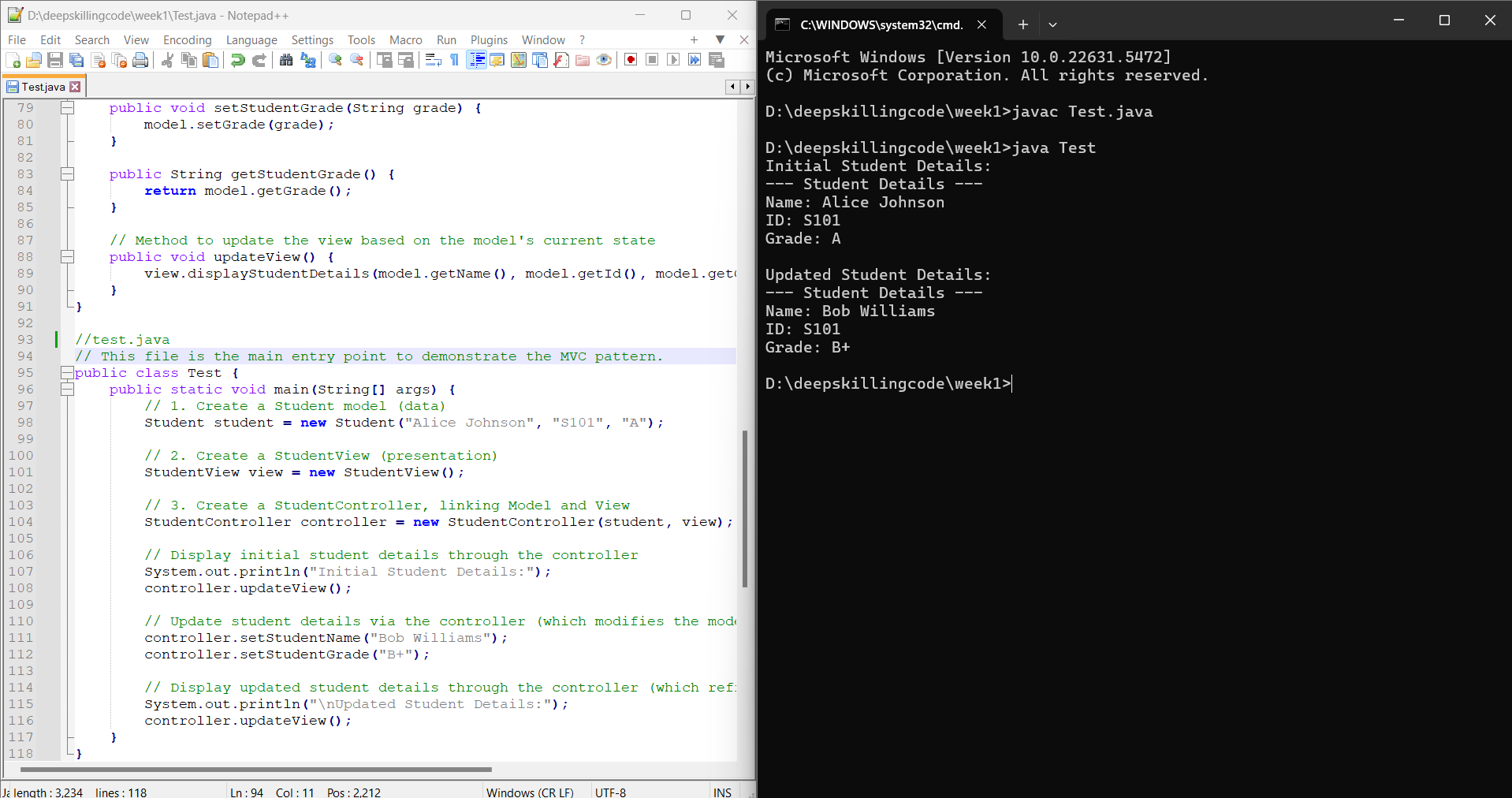
**System.out.println("\nUpdated Student Details:");**

**controller.updateView();**

**}**

**}**

**OUTPUT**

****

**---------------------------------------------------------------------------------------------------------**

**Exercise 11: Implementing Dependency Injection**

**//CODE**

**// CustomerRepository.java**

**// This is the Repository Interface, defining the contract for data access.**

**interface CustomerRepository {**

**String findCustomerById(String id);**

**}**

**// CustomerRepositoryImpl.java**

**// This is the Concrete Repository implementation.**

**class CustomerRepositoryImpl implements CustomerRepository {**

**@Override**

**public String findCustomerById(String id) {**

**// In a real application, this would interact with a database**

**if ("123".equals(id)) {**

**return "John Doe (ID: 123)";**

**} else if ("456".equals(id)) {**

**return "Jane Smith (ID: 456)";**

**} else {**

**return "Customer not found for ID: " + id;**

**}**

**}**

**}**

**// CustomerService.java**

**// This is the Service Class, which depends on CustomerRepository.**

**class CustomerService {**

**private CustomerRepository customerRepository;**

**public CustomerService(CustomerRepository customerRepository) {**

**this.customerRepository = customerRepository;**

**System.out.println("CustomerService created with " + customerRepository.getClass().getSimpleName());**

**}**

**public String getCustomerDetails(String id) {**

**System.out.println("CustomerService: Fetching customer details for ID " + id);**

**return customerRepository.findCustomerById(id);**

**}**

**}**

**// Test.java**

**// This file demonstrates the usage of Dependency Injection.**

**class Test {**

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

**// 1. Create an instance of the concrete dependency**

**CustomerRepository customerRepository = new CustomerRepositoryImpl();**

**// 2. Inject the dependency into the CustomerService during its creation.**

**CustomerService customerService = new CustomerService(customerRepository);**

**// 3. Use the service, which internally uses the injected repository.**

**System.out.println(customerService.getCustomerDetails("123"));**

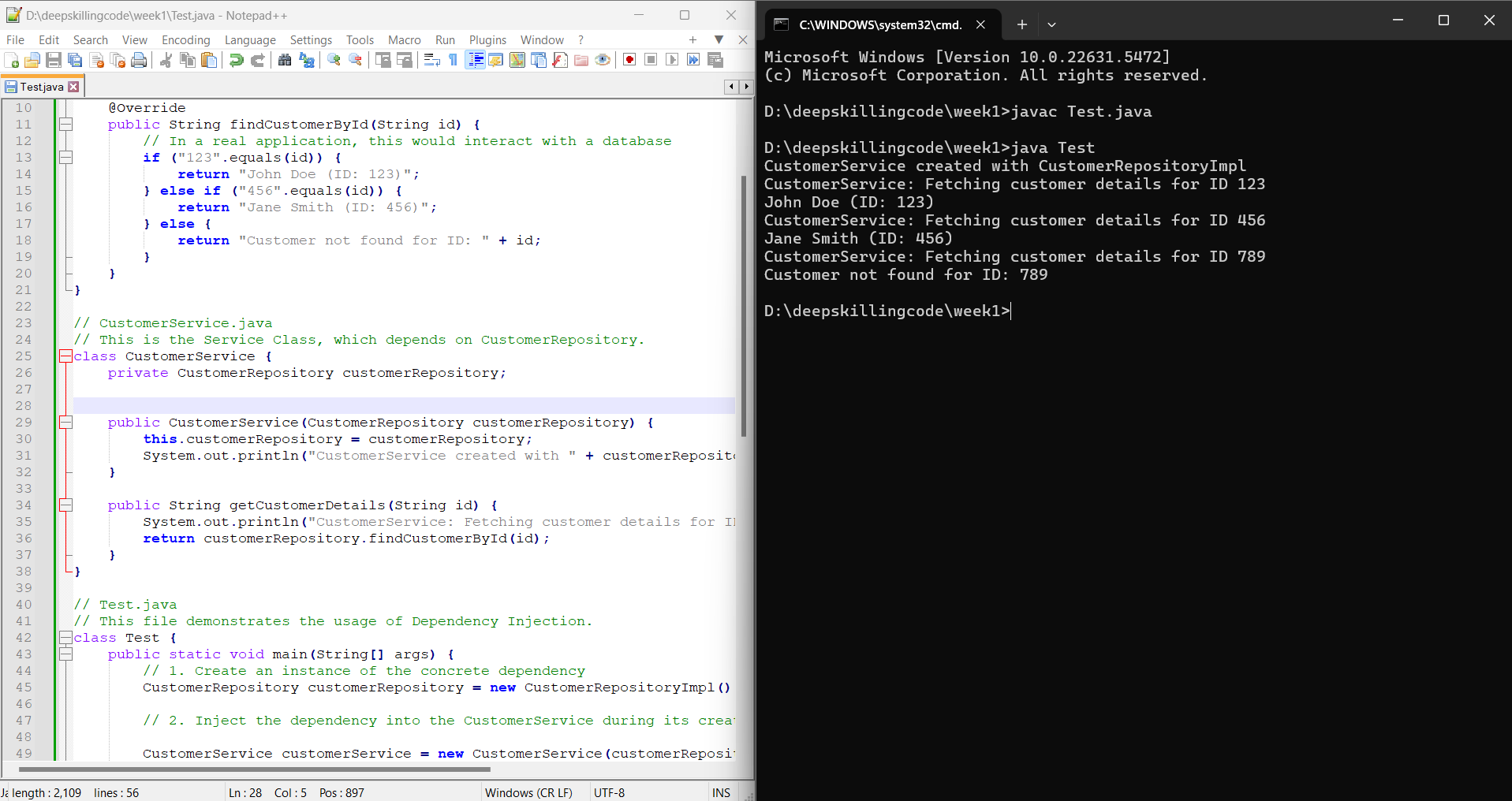
**System.out.println(customerService.getCustomerDetails("456"));**

**System.out.println(customerService.getCustomerDetails("789"));**

**}**

**}**

**OUTPUT**

****

**---------------------------------------------------------------------------------------------------------**

ALOGORITHM AND DATA STRUCTURES

**Exercise 1: Inventory Management System**

**//CODE**

**import java.util.\*;**

**class Product {**

**int productId;**

**String productName;**

**String quality;**

**double price;**

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

**this.productId = productId;**

**this.productName = productName;**

**this.quality = quality;**

**this.price = price;**

**}**

**public String toString() {**

**return "ProductID: " + productId + ", Name: " + productName +**

**", Quality: " + quality + ", Price: " + price;**

**}**

**}**

**class InventoryManagement {**

**Map<Integer, Product> m = new HashMap<>();**

**public void add(int productId, Product p) {**

**m.put(productId, p);**

**}**

**public void update(int productId, String name, String quality, double price) {**

**if (m.containsKey(productId)) {**

**Product p = m.get(productId);**

**p.productName = name;**

**p.quality = quality;**

**p.price = price;**

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

**} else {**

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

**}**

**}**

**public void delete(int productId) {**

**if (m.containsKey(productId)) {**

**m.remove(productId);**

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

**} else {**

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

**}**

**}**

**public void display() {**

**if (m.isEmpty()) {**

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

**} else {**

**for (Map.Entry<Integer, Product> entry : m.entrySet()) {**

**System.out.println(entry.getValue());**

**}**

**}**

**}**

**}**

**public class Main {**

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

**InventoryManagement inventory = new InventoryManagement();**

**// Add**

**inventory.add(1, new Product(1, "Dell Laptop", "High", 50000));**

**inventory.add(2, new Product(2, "HP Laptop", "Medium", 40000));**

**// Display**

**System.out.println("Initial Inventory:");**

**inventory.display();**

**// Update**

**inventory.update(1, "Dell Laptop Pro", "Premium", 60000);**

**// Display after update**

**System.out.println("\nAfter Update:");**

**inventory.display();**

**// Delete**

**inventory.delete(2);**

**// Display after delete**

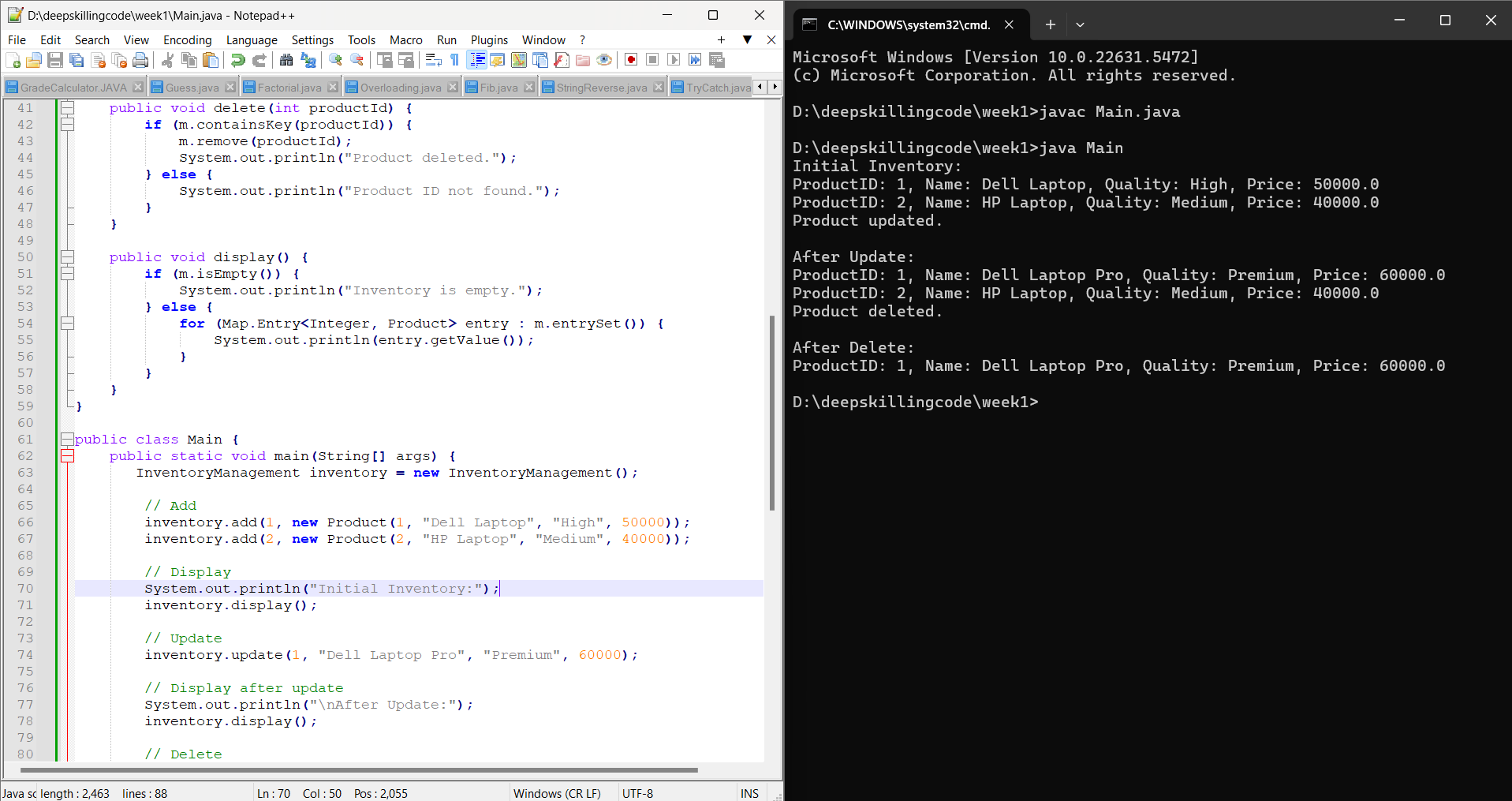
**System.out.println("\nAfter Delete:");**

**inventory.display();**

**}**

**}**

**OUTPUT**



**Exercise 3: Sorting Customer Orders**

**//CODE**

**import java.util.\*;**

**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 String toString() {**

**return "Order ID: " + orderId + ", Customer: " + customerName + ", Total Price: " + totalPrice;**

**}**

**}**

**class Main {**

**//QUICK SORT**

**public static void quickSort(List<Order> list, int low, int high) {**

**if (low < high) {**

**int pi = partition(list, low, high);**

**quickSort(list, low, pi - 1);  // left side**

**quickSort(list, pi + 1, high); // right side**

**}**

**}**

**public static int partition(List<Order> list, int low, int high) {**

**double pivot = list.get(high).totalPrice;**

**int i = low - 1;**

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

**if (list.get(j).totalPrice <= pivot) {**

**i++;**

**// swap i and j**

**Order temp = list.get(i);**

**list.set(i, list.get(j));**

**list.set(j, temp);**

**}**

**}**

**// swap pivot with i+1**

**Order temp = list.get(i + 1);**

**list.set(i + 1, list.get(high));**

**list.set(high, temp);**

**return i + 1;**

**}**

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

**List<Order> o = new ArrayList<>();**

**o.add( new Order(101, "Alice", 1500.0));**

**o.add( new Order(102, "Bob", 2000.0));**

**o.add( new Order(101, "Alice", 1500.0));**

**o.add( new Order(103, "Charlie", 3000.0));**

**//BUBBLE SORT**

**for(int i=o.size()-1;i>=1;i--)**

**{**

**for(int j=i-1;j>=0;j--)**

**{**

**if(o.get(i).totalPrice<o.get(j).totalPrice)**

**{**

**Order temp = o.get(i);**

**o.set(i, o.get(j ));**

**o.set(j , temp);**

**}**

**}**

**}**

**quickSort(o, 0, o.size() - 1);**

**// Display orders**

**for (Order o1 : o) {**

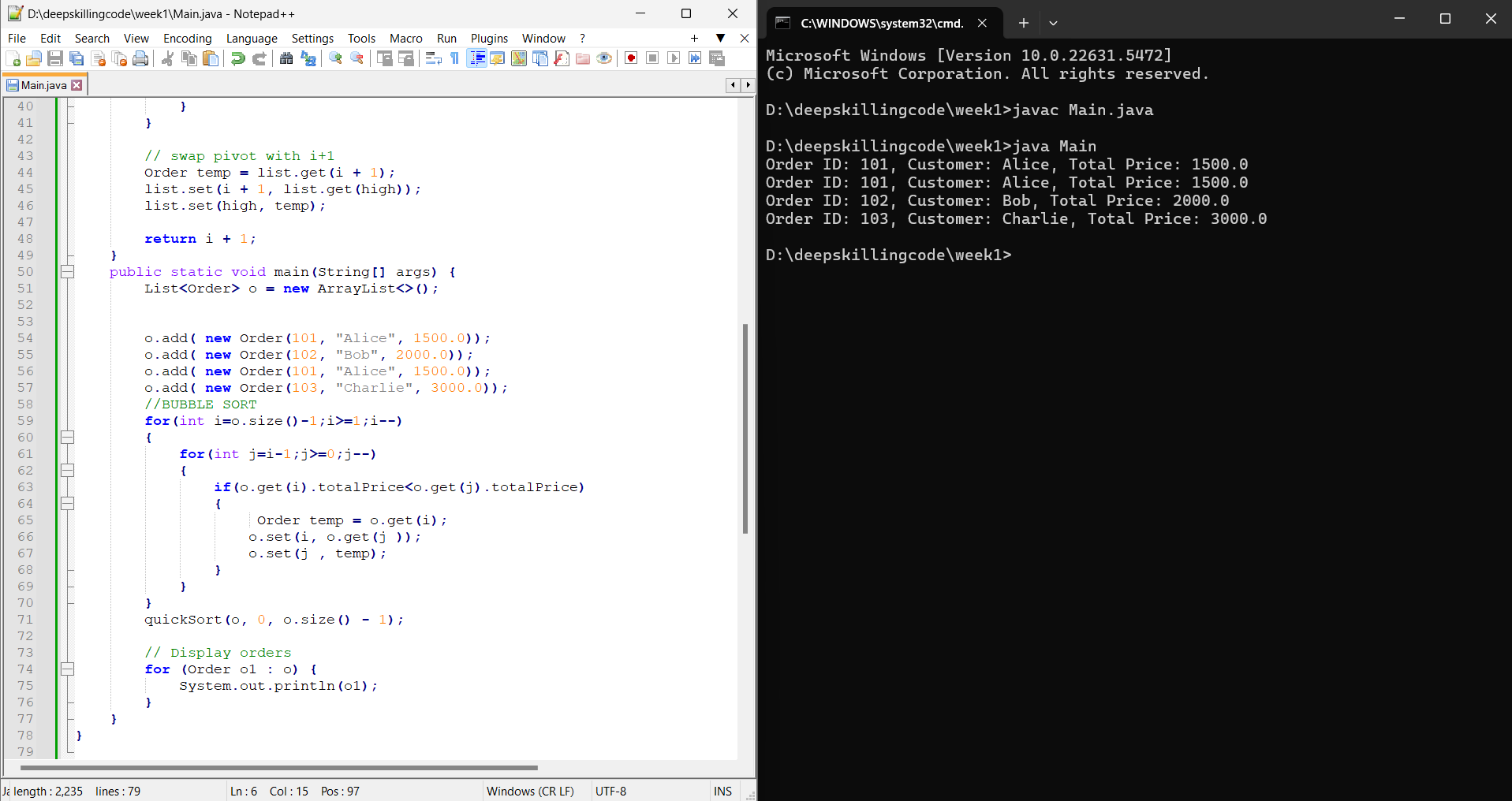
**System.out.println(o1);**

**}**

**}**

**}**

**OUTPUT**



**Exercise 4: Employee Management System**

**//CODE**

**import java.util.\*;**

**class Employee {**

**int employeeId;**

**String name;**

**int position;**

**double salary;**

**public Employee(int employeeId, String name, int 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;**

**}**

**}**

**public class Main {**

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

**Scanner in = new Scanner(System.in);**

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

**int count = 0;**

**// Add new Employee**

**System.out.print("Enter ID: ");**

**int addId = in.nextInt();**

**in.nextLine(); // consume newline**

**System.out.print("Enter Name: ");**

**String name = in.nextLine();**

**System.out.print("Enter Position: ");**

**int pos = in.nextInt();**

**System.out.print("Enter Salary: ");**

**double sal = in.nextDouble();**

**employees[count++] = new Employee(addId, name, pos, sal);**

**// Search**

**System.out.print("Enter ID to search: ");**

**int searchId = in.nextInt();**

**boolean found = false;**

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

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

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

**found = true;**

**break;**

**}**

**}**

**if (!found) {**

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

**}**

**// Print**

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

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

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

**}**

**// Delete**

**System.out.print("\nEnter ID to delete: ");**

**int deleteId = in.nextInt();**

**boolean deleted = false;**

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

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

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

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

**}**

**employees[--count] = null;**

**deleted = true;**

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

**break;**

**}**

**}**

**if (!deleted) {**

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

**}**

**// Print after delete**

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

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

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

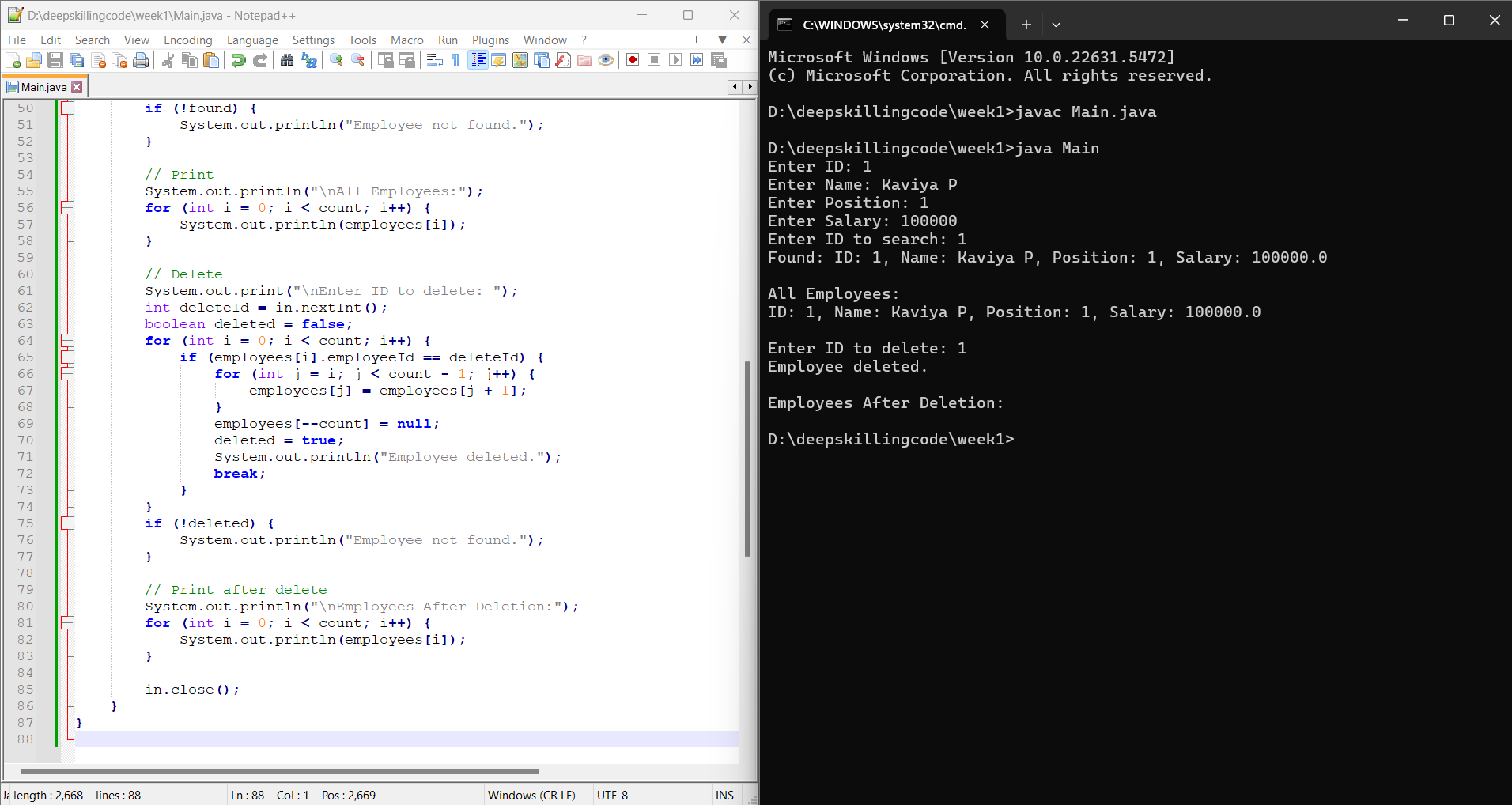
**}**

**in.close();**

**}**

**}**

**OUTPUT**



**Exercise 5: Task Management System**

**import java.util.\*;**

**class Task {**

**int taskId;**

**String taskName;**

**String status;**

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

**this.taskId = taskId;**

**this.taskName = taskName;**

**this.status = status;**

**}**

**public String toString() {**

**return "Task ID: " + taskId + ", Name: " + taskName + ", Status: " + status;**

**}**

**}**

**public class Main {**

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

**Scanner in = new Scanner(System.in);**

**List<Task> taskList = new LinkedList<>();**

**// Add tasks**

**taskList.add(new Task(1, "Machine Testing", "Progress"));**

**taskList.add(new Task(2, "Debug Module", "Pending"));**

**taskList.add(new Task(3, "Deploy Project", "Completed"));**

**// Traverse**

**System.out.println("All Tasks:");**

**for (Task t : taskList) {**

**System.out.println(t);**

**}**

**// Delete a task by ID**

**System.out.print("\nEnter task ID to delete: ");**

**int deleteId = in.nextInt();**

**boolean found = false;**

**Iterator<Task> iterator = taskList.iterator();**

**while (iterator.hasNext()) {**

**Task t = iterator.next();**

**if (t.taskId == deleteId) {**

**iterator.remove();**

**found = true;**

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

**break;**

**}**

**}**

**if (!found) {**

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

**}**

**// Display after deletion**

**System.out.println("\nTasks after deletion:");**

**for (Task t : taskList) {**

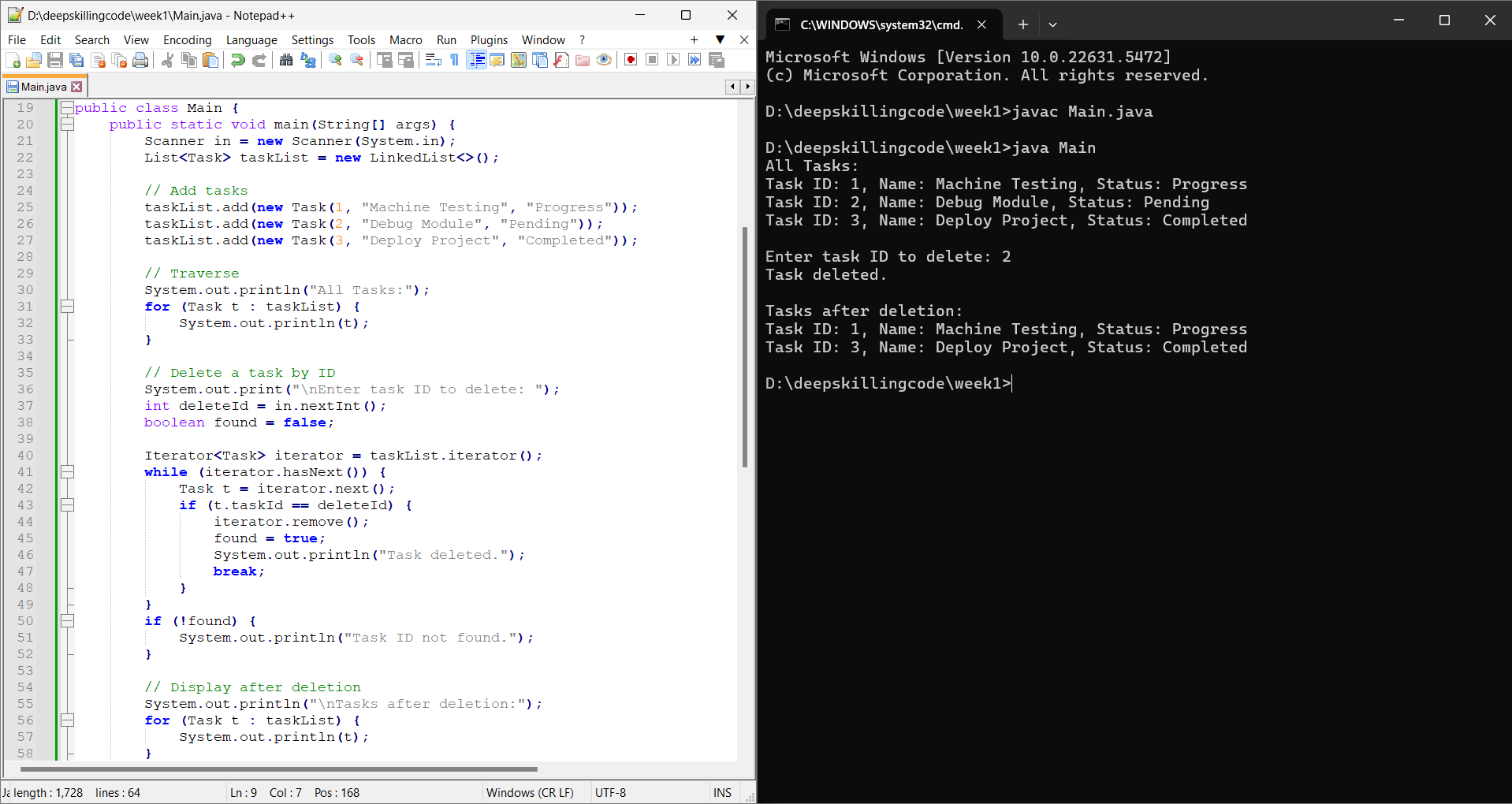
**System.out.println(t);**

**}**

**}**

**}**

**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 String toString() {**

**return "Book ID: " + bookId + ", Title: " + title + ", Author: " + author;**

**}**

**}**

**public class Main {**

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

**Scanner in = new Scanner(System.in);**

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

**books.add(new Book(1, "Java Basics", "James Gosling"));**

**books.add(new Book(2, "Python Crash Course", "Eric Matthes"));**

**books.add(new Book(3, "Effective Java", "Joshua Bloch"));**

**books.add(new Book(4, "Clean Code", "Robert C. Martin"));**

**books.add(new Book(5, "Head First Java", "Kathy Sierra"));**

**books.add(new Book(6, "Introduction to Algorithms", "Thomas Cormen"));**

**books.add(new Book(7, "The Pragmatic Programmer", "Andrew Hunt"));**

**books.add(new Book(8, "Design Patterns", "Erich Gamma"));**

**books.add(new Book(9, "Artificial Intelligence", "Stuart Russell"));**

**books.add(new Book(10, "Data Structures in Java", "Narasimha Karumanchi"));**

**System.out.print("Enter the book title to search: ");**

**String searchTitle = in.nextLine();**

**// Linear Search**

**System.out.println("\nLinear Search Result:");**

**boolean foundLinear = false;**

**for (Book b : books) {**

**if (b.title.equalsIgnoreCase(searchTitle)) {**

**System.out.println(b);**

**foundLinear = true;**

**break;**

**}**

**}**

**if (!foundLinear) {**

**System.out.println("Book not found using linear search.");**

**}**

**// Binary Search**

**System.out.println("\nBinary Search Result:");**

**Collections.sort(books, Comparator.comparing(b -> b.title.toLowerCase()));**

**int low = 0;**

**int high = books.size() - 1;**

**boolean foundBinary = false;**

**while (low <= high) {**

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

**int cmp = books.get(mid).title.compareToIgnoreCase(searchTitle);**

**if (cmp == 0) {**

**System.out.println(books.get(mid));**

**foundBinary = true;**

**break;**

**} else if (cmp < 0) {**

**low = mid + 1;**

**} else {**

**high = mid - 1;**

**}**

**}**

**if (!foundBinary) {**

**System.out.println("Book not found using binary search.");**

**}**

**in.close();**

**}**

**}**

**OUTPUT**

