Design principles and patterns

Mandatory Hands-on:

**Exercise 1: Implementing the Singleton Pattern**

**Program:**

**1.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);**

**}**

**}**

**2.LoggerTest.java**

**public class LoggerTest {**

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

**Logger logger1 = Logger.getInstance();**

**Logger logger2 = Logger.getInstance();**

**logger1.log("Started application.");**

**logger2.log("Loaded modules.");**

**if (logger1 == logger2) {**

**System.out.println("Same instance confirmed.");**

**} else {**

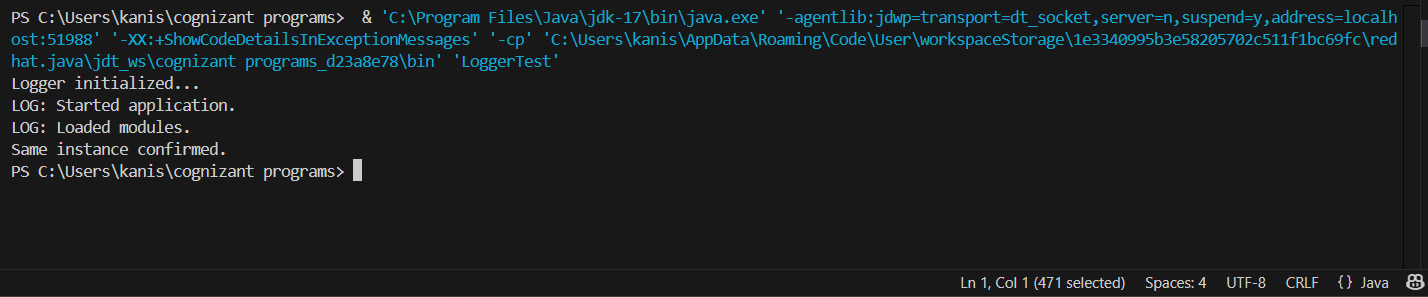
**System.out.println("Different instances detected.");**

**}**

**}**

**}**

**3.Output:**

****

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

**Program**

**1.Document.java**

**public interface Document {**

**void open();**

**}**

**2.DocumentFactory.java**

**public abstract class DocumentFactory {**

**public abstract Document createDocument();**

**}**

**3.DocumentTest.java**

**public class DocumentTest {**

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

**DocumentFactory wordFactory = new WordFactory();**

**Document wordDoc = wordFactory.createDocument();**

**wordDoc.open();**

**DocumentFactory pdfFactory = new PdfFactory();**

**Document pdfDoc = pdfFactory.createDocument();**

**pdfDoc.open();**

**DocumentFactory excelFactory = new ExcelFactory();**

**Document excelDoc = excelFactory.createDocument();**

**excelDoc.open();**

**}**

**}**

**4.ExcelDocument.java**

**public class ExcelDocument implements Document {**

**@Override**

**public void open() {**

**System.out.println("Opening Excel document...");**

**}**

**}**

**5.PdfDocument.java**

**public class ExcelFactory extends DocumentFactory {**

**@Override**

**public Document createDocument() {**

**return new ExcelDocument();**

**}**

**}**

**6.PdfFactory.java**

**public class PdfFactory extends DocumentFactory {**

**@Override**

**public Document createDocument() {**

**return new PdfDocument();**

**}**

**}**

**7.WordDocument.java**

**public class WordDocument implements Document {**

**@Override**

**public void open() {**

**System.out.println("Opening Word document...");**

**}**

**}**

**8.WordFactory.java**

**public class WordFactory extends DocumentFactory {**

**@Override**

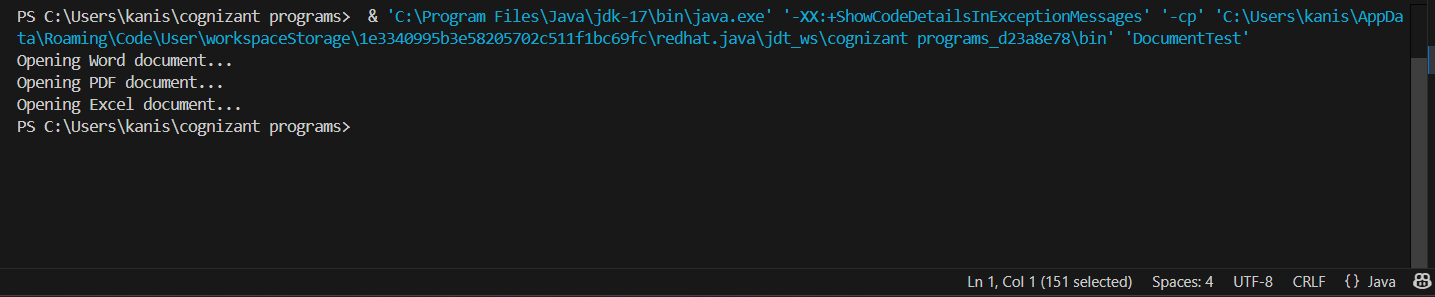
**public Document createDocument() {**

**return new WordDocument();**

**}**

**}**

**9.Output:**

****

**Additional Hands-On:**

**Exercise 5: Implementing the Decorator Pattern**

**Program:**

**1.EmailNotifier.java**

**public class EmailNotifier implements Notifier {**

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

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

**}**

**}**

**2.NotificationTest.java**

**public class NotificationTest {**

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

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

**Notifier sms = new SMSNotifierDecorator(basic);**

**Notifier slack = new SlackNotifierDecorator(sms);**

**slack.send("Server down alert!");**

**}**

**}**

**3.Notifier.java**

**public interface Notifier {**

**void send(String message);**

**}**

**4.NotifierDecorator.java**

**public abstract class NotifierDecorator implements Notifier {**

**protected Notifier wrapped;**

**public NotifierDecorator(Notifier notifier) {**

**this.wrapped = notifier;**

**}**

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

**wrapped.send(message);**

**}**

**}**

**5.SlackNotifierDecorator.java**

**public 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);**

**}**

**}**

**6.SMSNotifierDecorator.java**

**public class SMSNotifierDecorator extends NotifierDecorator {**

**public SMSNotifierDecorator(Notifier notifier) {**

**super(notifier);**

**}**

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

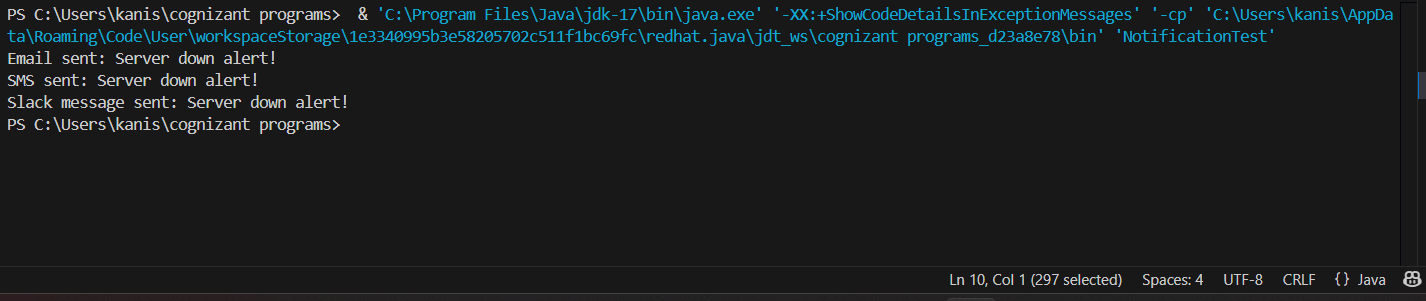
**super.send(message);**

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

**}**

**}**

**7.Output:**

****

**Exercise 8: Implementing the Strategy Pattern**

**Program:**

**1.CreditCardPayment.java**

**public class CreditCardPayment implements PaymentStrategy {**

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

**System.out.println("Paid Ruppees:" + amount + " via Credit Card.");**

**}**

**}**

**2.PaymentContext.java**

**public class PaymentContext {**

**private PaymentStrategy strategy;**

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

**this.strategy = strategy;**

**}**

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

**if (strategy == null) {**

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

**} else {**

**strategy.pay(amount);**

**}**

**}**

**}**

**3.PaymentStrategy.java**

**public interface PaymentStrategy {**

**void pay(double amount);**

**}**

**4.PaymentTest.java**

**public class PaymentTest {**

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

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

**context.setStrategy(new CreditCardPayment());**

**context.payBill(2500.00);**

**context.setStrategy(new PayPalPayment());**

**context.payBill(780.50);**

**}**

**}**

**5.PayPalPayment.java**

**public class PaymentTest {**

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

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

**context.setStrategy(new CreditCardPayment());**

**context.payBill(2500.00);**

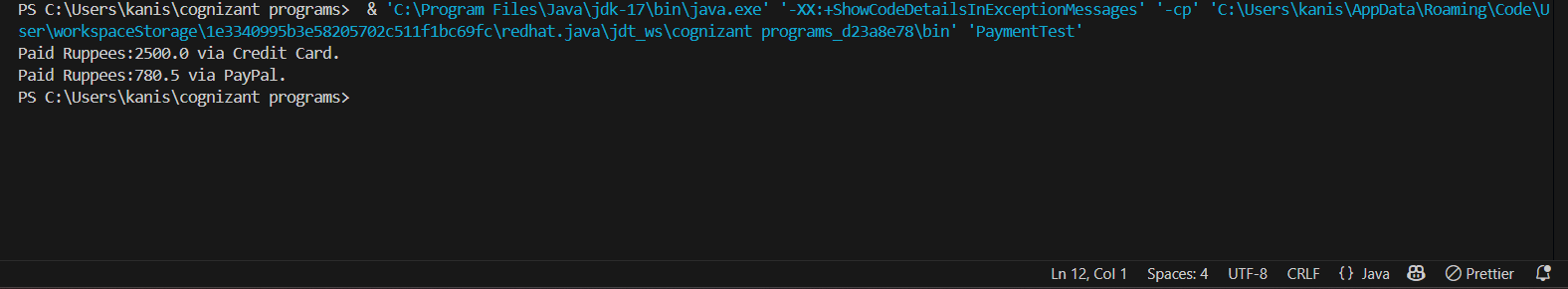
**context.setStrategy(new PayPalPayment());**

**context.payBill(780.50);**

**}**

**}**

**6.Ouput:**

****