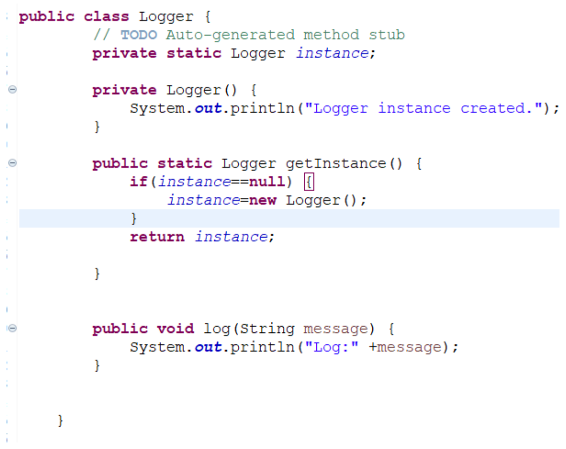
DESIGN PATTERNS AND PRINCIPLES(WEEK1)

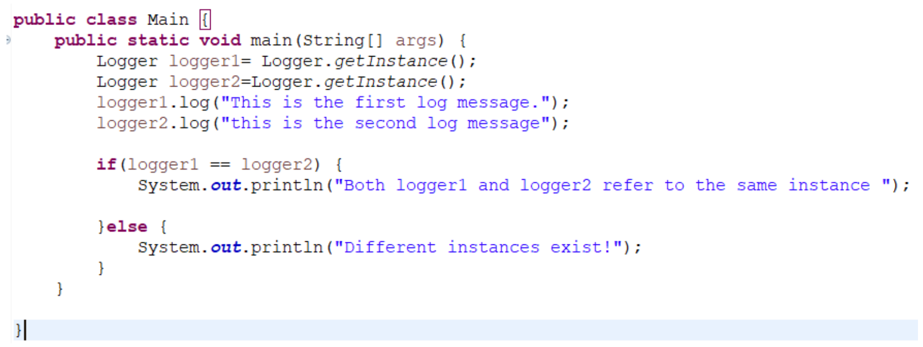
Exercise 1: Implementing the Singleton Pattern

Logger.java



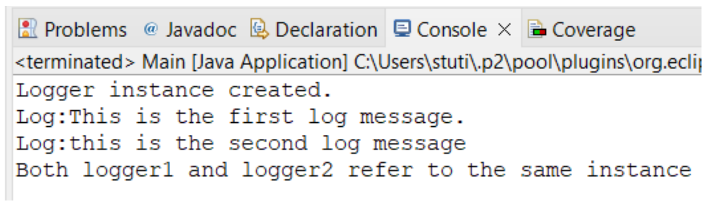
```java
public class Logger {
        // TODO Auto-generated method stub
        private static Logger instance;

        private Logger() {
            System.out.println("Logger instance created.");
        }

        public static Logger getInstance() {
            if(instance==null) {
                instance=new Logger();
            }
            return instance;

        }


        public void log(String message) {
            System.out.println("Log:" +message);
        }


    }
```

Main.java



```java
public class Main {
    public static void main(String[] args) {
        Logger logger1= Logger.getInstance();
        Logger logger2=Logger.getInstance();
        logger1.log("This is the first log message.");
        logger2.log("this is the second log message");

        if(logger1 == logger2) {
            System.out.println("Both logger1 and logger2 refer to the same instance ");

        }else {
            System.out.println("Different instances exist!");
        }
    }

}
```

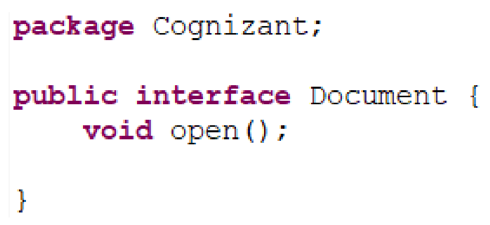
Output:



```
<terminated> Main [Java Application] C:\Users\stuti\.p2\pool\plugins\org.ecli
Logger instance created.
Log:This is the first log message.
Log:this is the second log message
Both logger1 and logger2 refer to the same instance
```

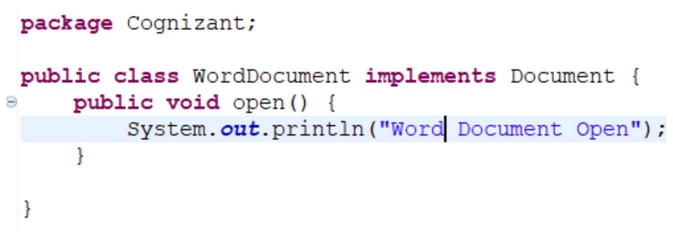
Exercise 2: Implementing the Factory Method Pattern

Document.java



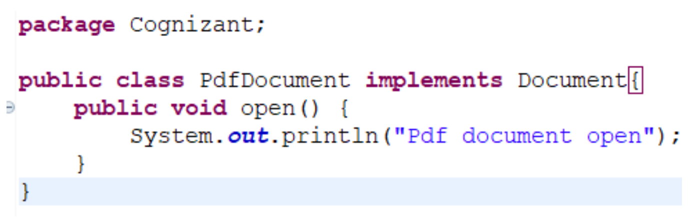
```java
package Cognizant;

public interface Document {
    void open();

}
```

WordDocument.java



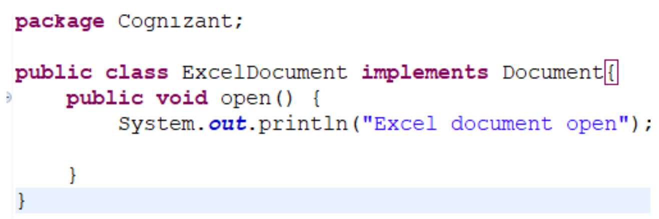
```java
package Cognizant;

public class WordDocument implements Document {
    public void open() {
        System.out.println("Word Document Open");
    }

}
```

PdfDocument.java



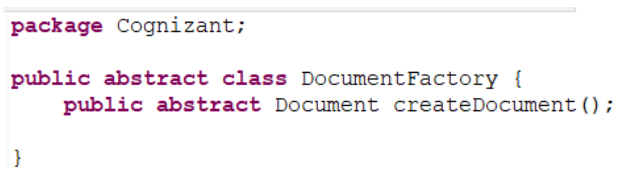
```java
package Cognizant;

public class PdfDocument implements Document{
    public void open()  {
        System.out.println("Pdf document open");
    }
}
```

ExcelDocument.java



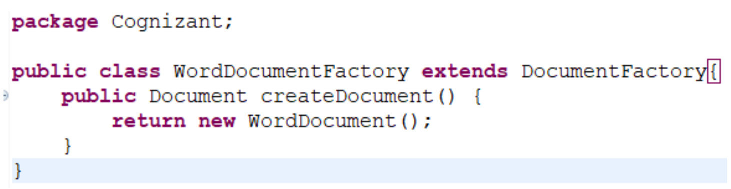
```java
package Cognizant;

public class ExcelDocument implements Document{
    public void open() {
        System.out.println("Excel document open");

    }
}
```

DocumentFactory.java



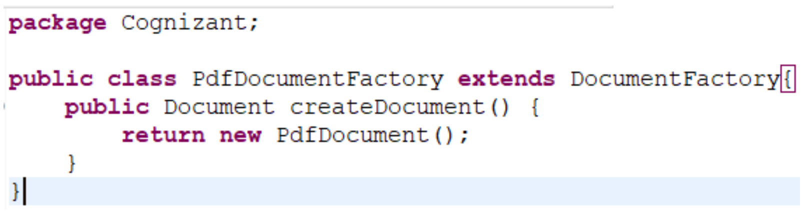
```java
package Cognizant;

public abstract class DocumentFactory {
    public abstract Document createDocument();

}
```

WordDocumentFactory.java



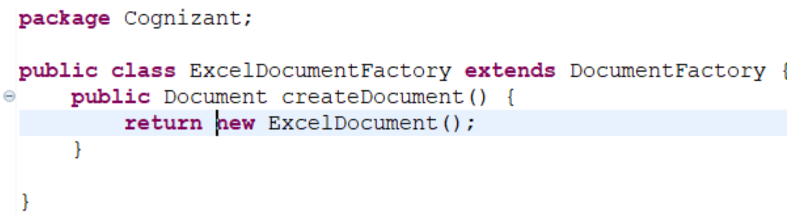
```java
package Cognizant;

public class WordDocumentFactory extends DocumentFactory{
    public Document createDocument() {
        return new WordDocument();
    }
}
```

PdfDocumentFactory.java



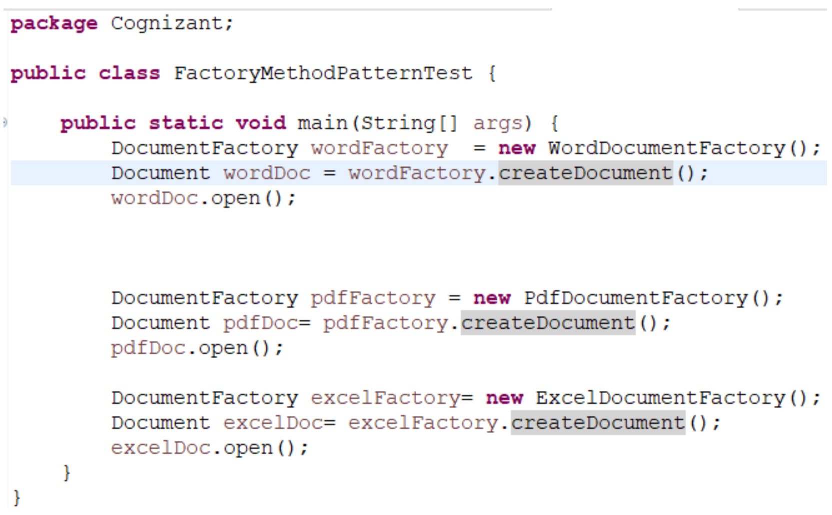
```java
package Cognizant;

public class PdfDocumentFactory extends DocumentFactory{
    public Document createDocument() {
        return new PdfDocument();
    }
}
```

ExcelDocumentFactory.java



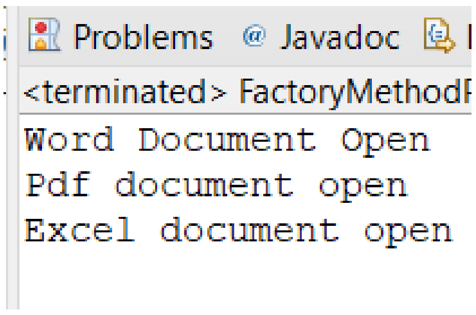
```java
package Cognizant;

public class ExcelDocumentFactory extends DocumentFactory {
    public Document createDocument() {
        return new ExcelDocument();
    }

}
```

FactoryMethodPatternTest.java



```java
package Cognizant;

public class FactoryMethodPatternTest {

    public static void main(String[] args) {
        DocumentFactory wordFactory  = new WordDocumentFactory();
        Document wordDoc = wordFactory.createDocument();
        wordDoc.open();



        DocumentFactory pdfFactory = new PdfDocumentFactory();
        Document pdfDoc= pdfFactory.createDocument();
        pdfDoc.open();

        DocumentFactory excelFactory= new ExcelDocumentFactory();
        Document excelDoc= excelFactory.createDocument();
        excelDoc.open();
    }
}
```

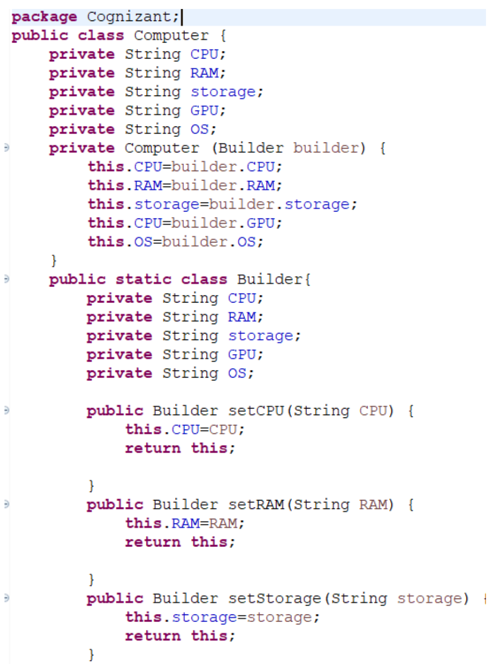
Output:



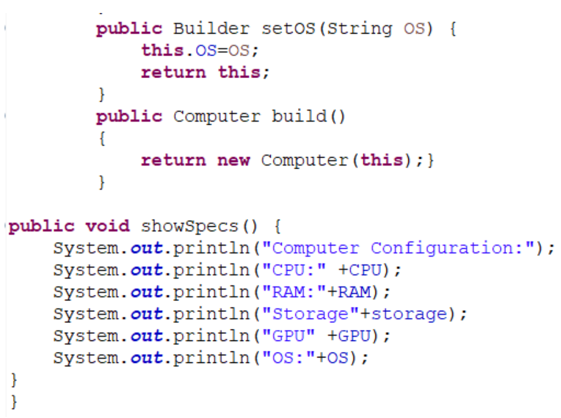
```
<terminated> FactoryMethod
Word Document Open
Pdf document open
Excel document open
```

Exercise 3: Implementing the Builder Pattern

Computer.java

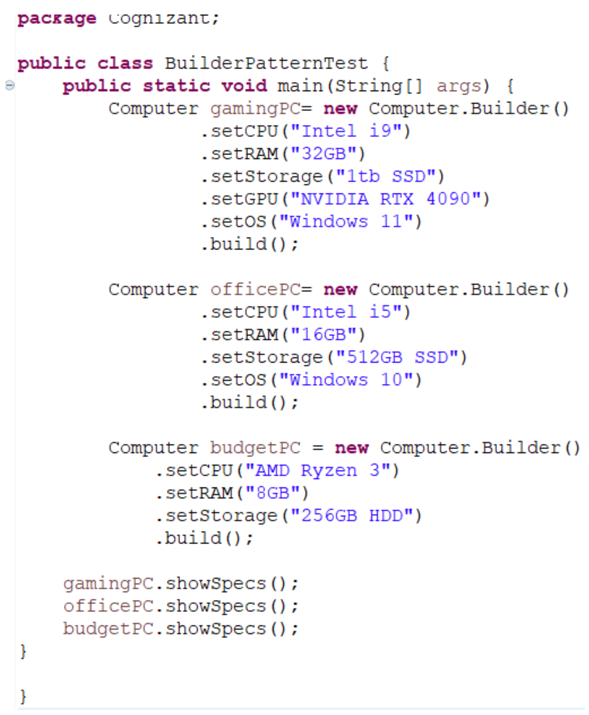


```
package Cognizant;
public class Computer {
    private String CPU;
    private String RAM;
    private String storage;
    private String GPU;
    private String OS;
    private Computer (Builder builder) {
        this.CPU=builder.CPU;
        this.RAM=builder.RAM;
        this.storage=builder.storage;
        this.CPU=builder.GPU;
        this.OS=builder.OS;
    }
    public static class Builder{
        private String CPU;
        private String RAM;
        private String storage;
        private String GPU;
        private String OS;

        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;
        }
```



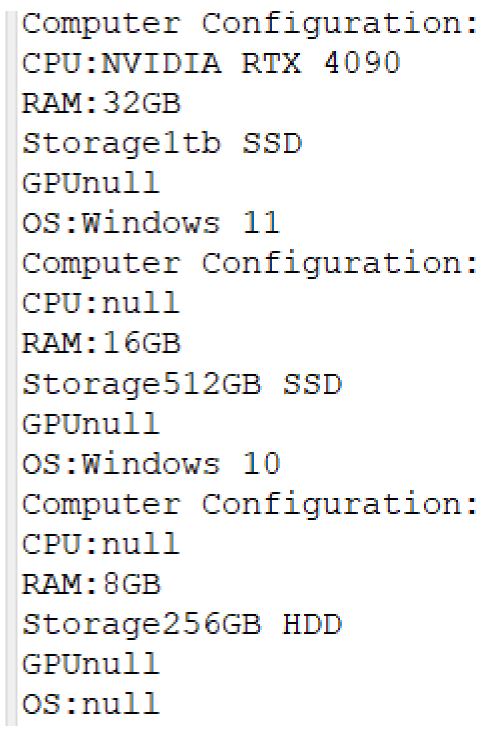
```
        public Builder setOS(String OS) {
            this.OS=OS;
            return this;
        }
        public Computer build()
        {
            return new Computer(this);}
        }

public void showSpecs() {
    System.out.println("Computer Configuration:");
    System.out.println("CPU:" +CPU);
    System.out.println("RAM:"+RAM);
    System.out.println("Storage"+storage);
    System.out.println("GPU" +GPU);
    System.out.println("OS:"+OS);
}
}
```

BuilderPatternTest.java



```java
package cognizant;

public class BuilderPatternTest {
    public static void main(String[] args) {
        Computer gamingPC= new Computer.Builder()
                .setCPU("Intel i9")
                .setRAM("32GB")
                .setStorage("1tb SSD")
                .setGPU("NVIDIA RTX 4090")
                .setOS("Windows 11")
                .build();

        Computer officePC= new Computer.Builder()
                .setCPU("Intel i5")
                .setRAM("16GB")
                .setStorage("512GB SSD")
                .setOS("Windows 10")
                .build();

        Computer budgetPC = new Computer.Builder()
            .setCPU("AMD Ryzen 3")
            .setRAM("8GB")
            .setStorage("256GB HDD")
            .build();

    gamingPC.showSpecs();
    officePC.showSpecs();
    budgetPC.showSpecs();
}

}
```

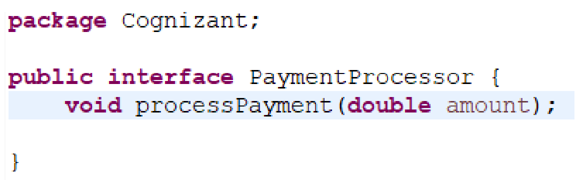
Output:



```
Computer Configuration:
CPU:NVIDIA RTX 4090
RAM:32GB
Storage1tb SSD
GPUnull
OS:Windows 11
Computer Configuration:
CPU:null
RAM:16GB
Storage512GB SSD
GPUnull
OS:Windows 10
Computer Configuration:
CPU:null
RAM:8GB
Storage256GB HDD
GPUnull
OS:null
```

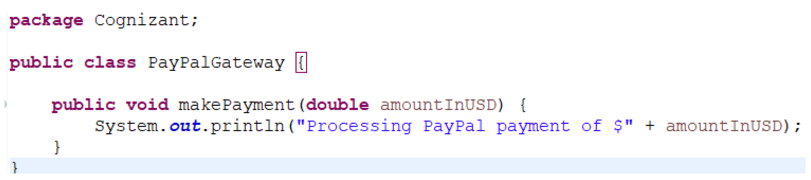
Exercise 4: Implementing the Adapter Pattern

PaymentProcessor.java



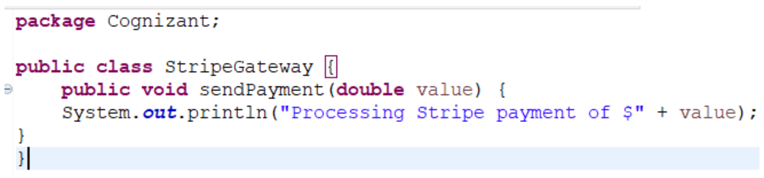
```java
package Cognizant;

public interface PaymentProcessor {
    void processPayment(double amount);

}
```

PayPalGateway.java



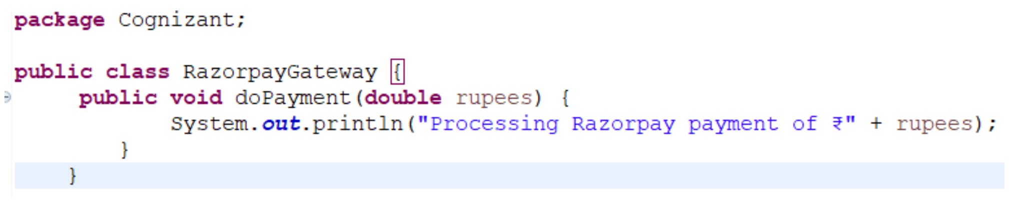
```java
package Cognizant;

public class PayPalGateway {

    public void makePayment(double amountInUSD) {
        System.out.println("Processing PayPal payment of $" + amountInUSD);
    }
}
```

StripeGateway.java



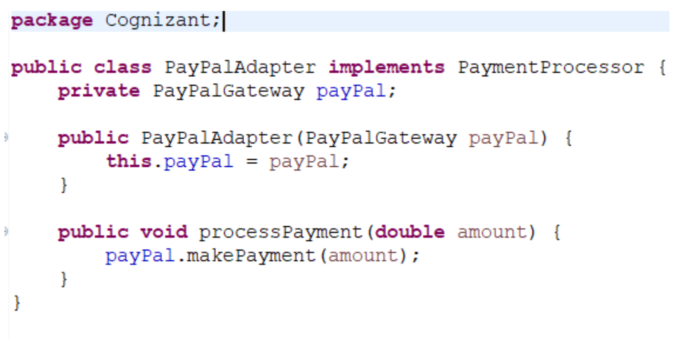
```java
package Cognizant;

public class StripeGateway {
    public void sendPayment(double value) {
    System.out.println("Processing Stripe payment of $" + value);
}
}
```

RazorpayGateway.java



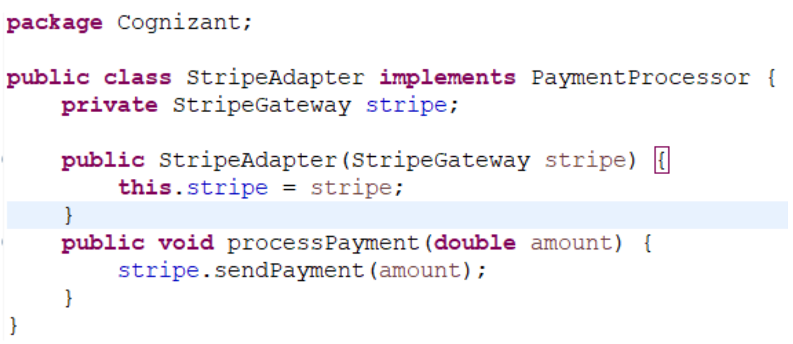
```java
package Cognizant;

public class RazorpayGateway {
    public void doPayment(double rupees) {
            System.out.println("Processing Razorpay payment of ₹" + rupees);
        }
    }
```

PayPalAdapter.java



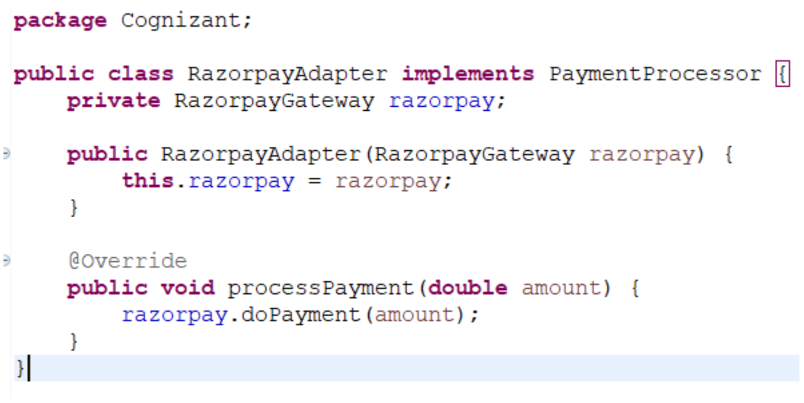
```java
package Cognizant;

public class PayPalAdapter implements PaymentProcessor {
    private PayPalGateway payPal;

    public PayPalAdapter(PayPalGateway payPal) {
        this.payPal = payPal;
    }

    public void processPayment(double amount) {
        payPal.makePayment(amount);
    }
}
```

StripeAdapter.java



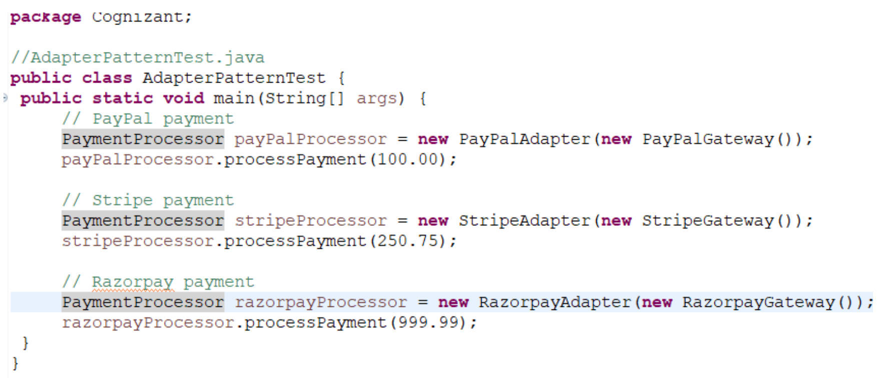
```java
package Cognizant;

public class StripeAdapter implements PaymentProcessor {
    private StripeGateway stripe;

    public StripeAdapter(StripeGateway stripe) {
        this.stripe = stripe;
    }
    public void processPayment(double amount) {
        stripe.sendPayment(amount);
    }
}
```

RazorpayAdapter.java



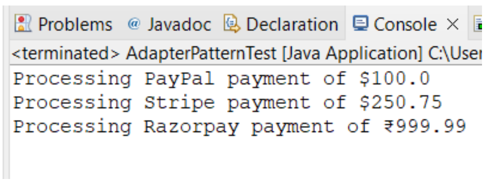
```java
package Cognizant;

public class RazorpayAdapter implements PaymentProcessor {
    private RazorpayGateway razorpay;

    public RazorpayAdapter(RazorpayGateway razorpay) {
        this.razorpay = razorpay;
    }

    @Override
    public void processPayment(double amount) {
        razorpay.doPayment(amount);
    }
}
```

AdapterPatternTest



```java
package Cognizant;

//AdapterPatternTest.java
public class AdapterPatternTest {
 public static void main(String[] args) {
     // PayPal payment
     PaymentProcessor payPalProcessor = new PayPalAdapter(new PayPalGateway());
     payPalProcessor.processPayment(100.00);

     // Stripe payment
     PaymentProcessor stripeProcessor = new StripeAdapter(new StripeGateway());
     stripeProcessor.processPayment(250.75);

     // Razorpay payment
     PaymentProcessor razorpayProcessor = new RazorpayAdapter(new RazorpayGateway());
     razorpayProcessor.processPayment(999.99);
 }
}
```

Output:



```
<terminated> AdapterPatternTest [Java Application] C:\User
Processing PayPal payment of $100.0
Processing Stripe payment of $250.75
Processing Razorpay payment of ₹999.99
```