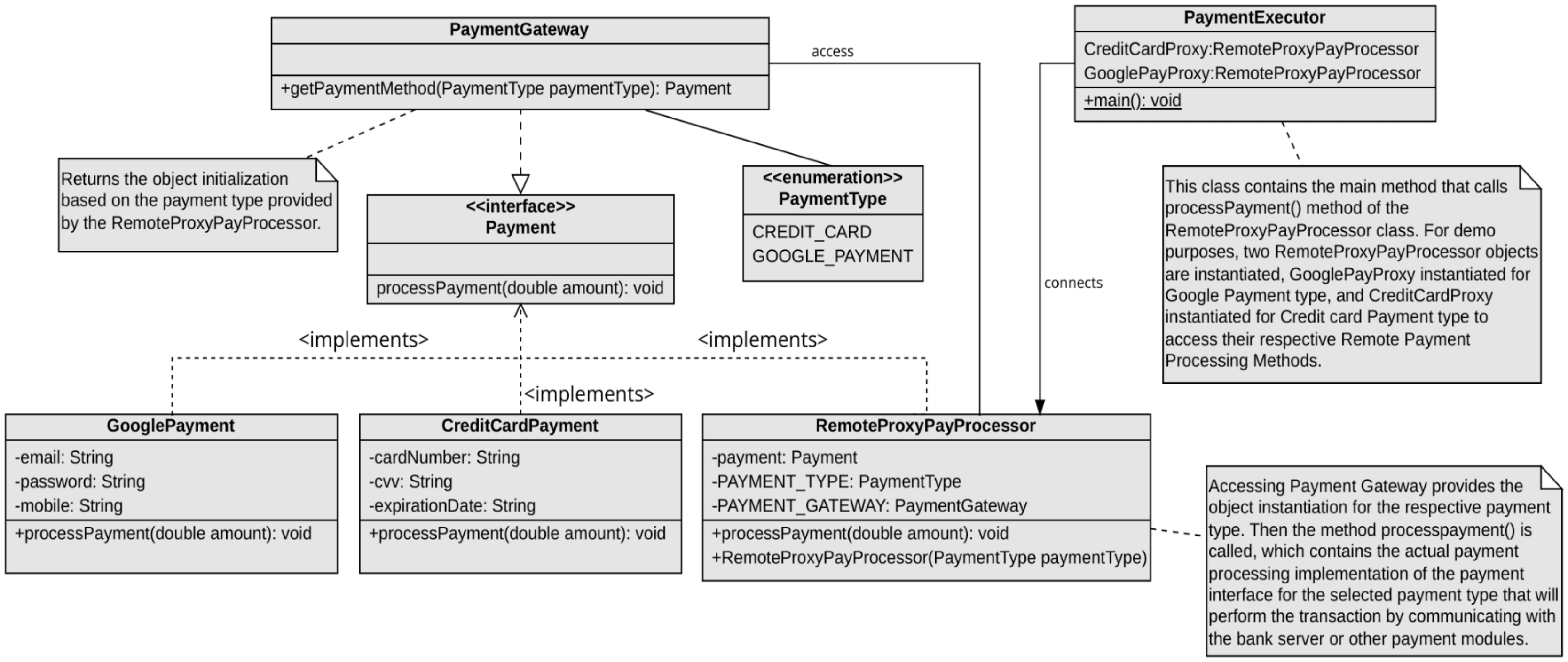
Name: Aravind Anand
Student ID: 030821269
Title: Assignment Unit 3 – Structural Design Pattern

Question 1: Find a compelling scenario where you can apply the Remote Proxy Design Pattern to the food delivery system. Draw the corresponding class diagram and sequence diagram and implement in either Java or Python (100 points)

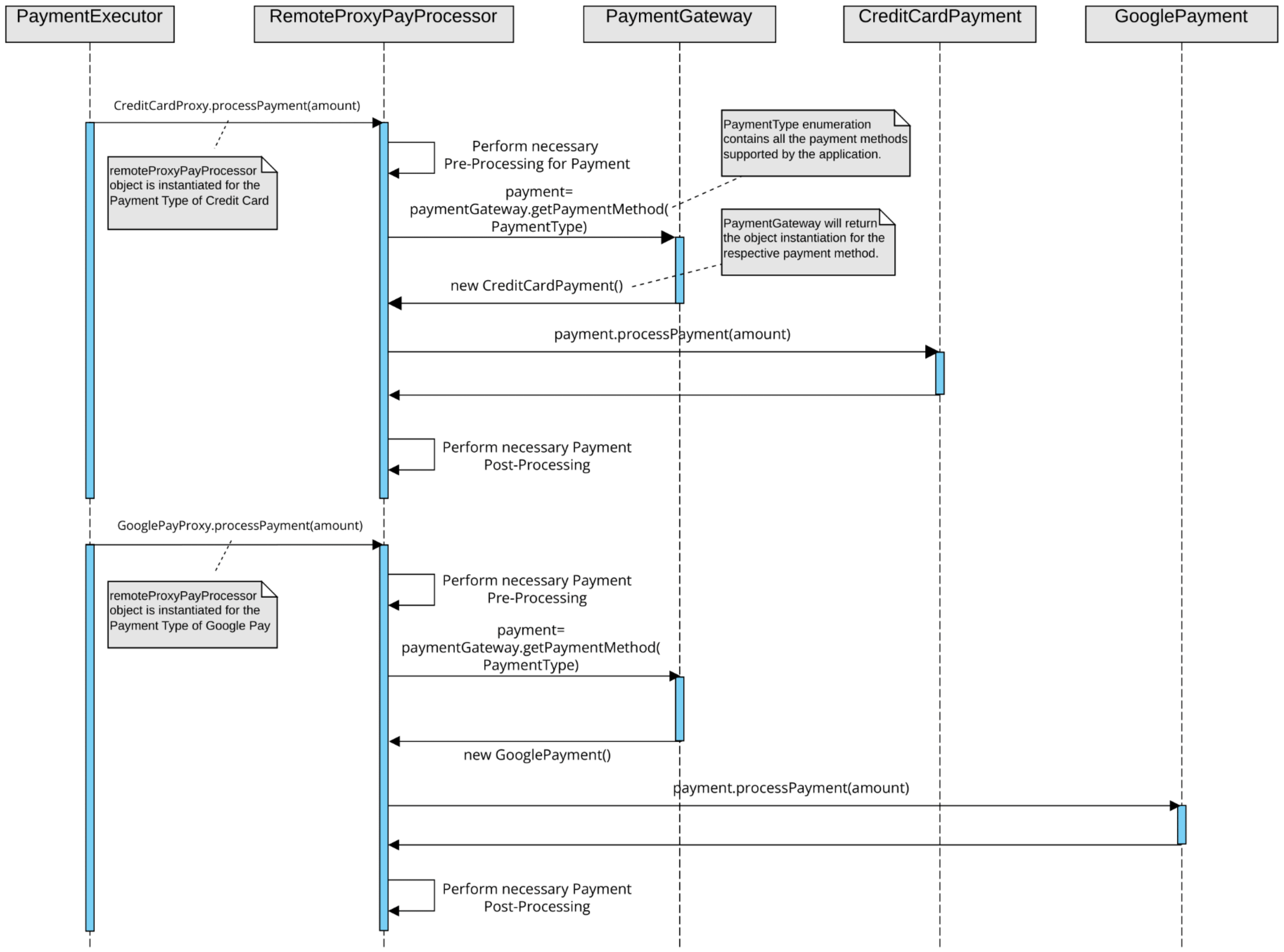
Report Proxy Design Pattern:

This Structural Design Pattern provides a remote object or placeholder for interaction purposes with the client, which acts like a local object. In other words, the client will be interacting with a representative object, which seems like the actual object and is used to make remote calls for the implementation of the desired operations. In the food delivery system, the payment processing can be implemented with a remote proxy design pattern, and we can also perform additional pre-processing and post-processing validations/operations. An implementation of payment processing using a Remote Proxy Design Pattern for a Food delivery system with their respective class diagram, sequence diagram, and code snippets are as follows:

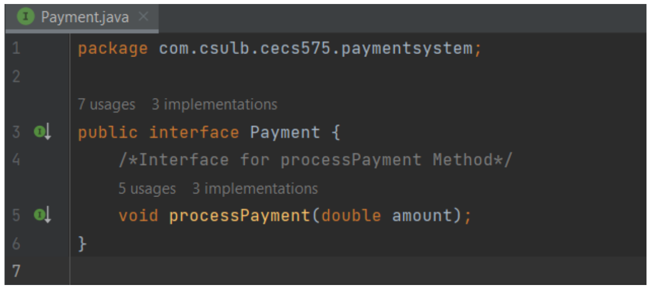
Class Diagram:


PaymentGateway
+getPaymentMethod(PaymentType paymentType): Payment
access
PaymentExecutor
CreditCardProxy:RemoteProxyPayProcessor
GooglePayProxy:RemoteProxyPayProcessor
+main(): void
Returns the object initialization based on the payment type provided by the RemoteProxyPayProcessor.
<<interface>> Payment
processPayment(double amount): void
<<enumeration>> PaymentType
CREDIT_CARD
GOOGLE_PAYMENT
connects
This class contains the main method that calls processPayment() method of the RemoteProxyPayProcessor class. For demo purposes, two RemoteProxyPayProcessor objects are instantiated, GooglePayProxy instantiated for Google Payment type, and CreditCardProxy instantiated for Credit card Payment type to access their respective Remote Payment Processing Methods.
<implements>
<implements>
<implements>
GooglePayment
-email: String
-password: String
-mobile: String
+processPayment(double amount): void
CreditCardPayment
-cardNumber: String
-cvv: String
-expirationDate: String
+processPayment(double amount): void
RemoteProxyPayProcessor
-payment: Payment
-PAYMENT_TYPE: PaymentType
-PAYMENT_GATEWAY: PaymentGateway
+processPayment(double amount): void
+RemoteProxyPayProcessor(PaymentType paymentType)
Accessing Payment Gateway provides the object instantiation for the respective payment type. Then the method processpayment() is called, which contains the actual payment processing implementation of the payment interface for the selected payment type that will perform the transaction by communicating with the bank server or other payment modules.

Sequence Diagram:


PaymentExecutor
RemoteProxyPayProcessor
PaymentGateway
CreditCardPayment
GooglePayment
CreditCardProxy.processPayment(amount)
remoteProxyPayProcessor object is instantiated for the Payment Type of Credit Card
Perform necessary Pre-Processing for Payment
PaymentType enumeration contains all the payment methods supported by the application.
payment= paymentGateway.getPaymentMethod(PaymentType)
PaymentGateway will return the object instantiation for the respective payment method.
new CreditCardPayment()
payment.processPayment(amount)
Perform necessary Payment Post-Processing
GooglePayProxy.processPayment(amount)
remoteProxyPayProcessor object is instantiated for the Payment Type of Google Pay
Perform necessary Payment Pre-Processing
payment= paymentGateway.getPaymentMethod(PaymentType)
new GooglePayment()
payment.processPayment(amount)
Perform necessary Payment Post-Processing

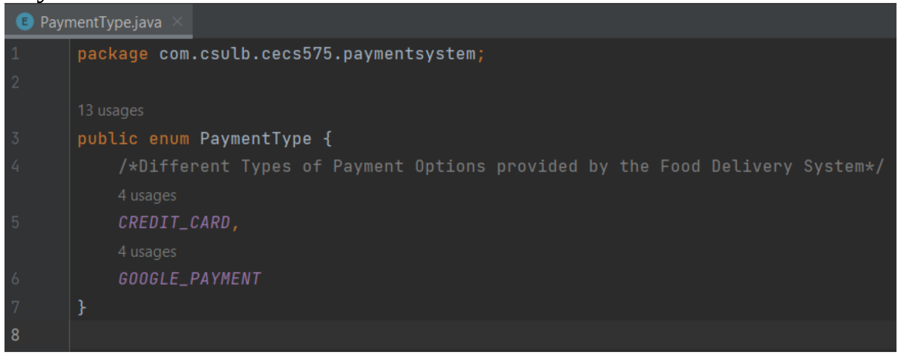
Remote Proxy for Payment Processing – Code Snippets:



Payment.java

```java
package com.csulb.cecs575.paymentsystem;

7 usages   3 implementations
public interface Payment {
    /*Interface for processPayment Method*/
    5 usages   3 implementations
    void processPayment(double amount);
}
```

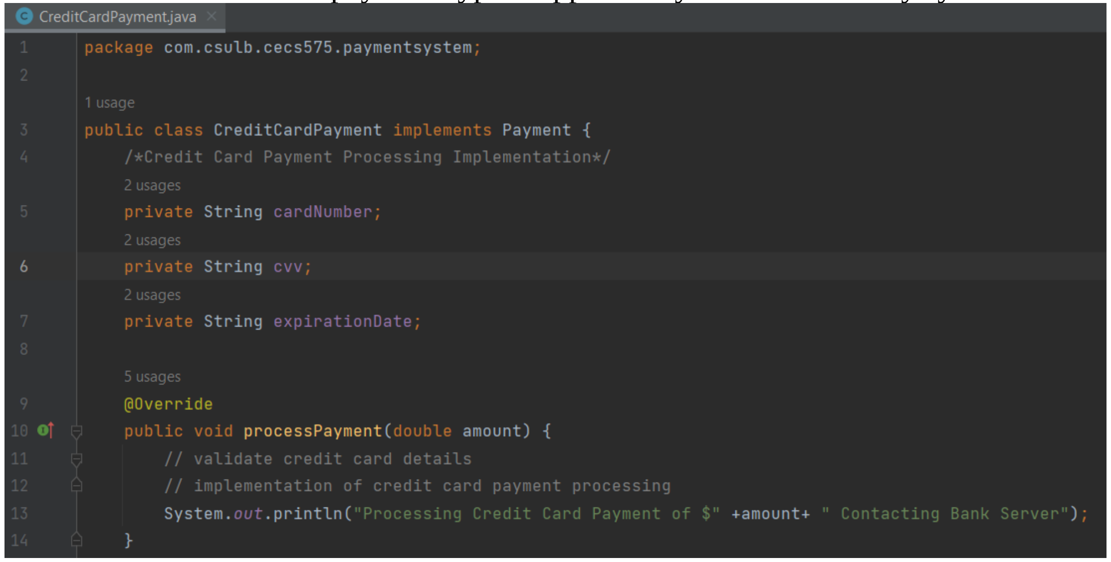
- Payment Interface.



PaymentType.java

```java
package com.csulb.cecs575.paymentsystem;

13 usages
public enum PaymentType {
    /*Different Types of Payment Options provided by the Food Delivery System*/
    4 usages
    CREDIT_CARD,
    4 usages
    GOOGLE_PAYMENT
}
```

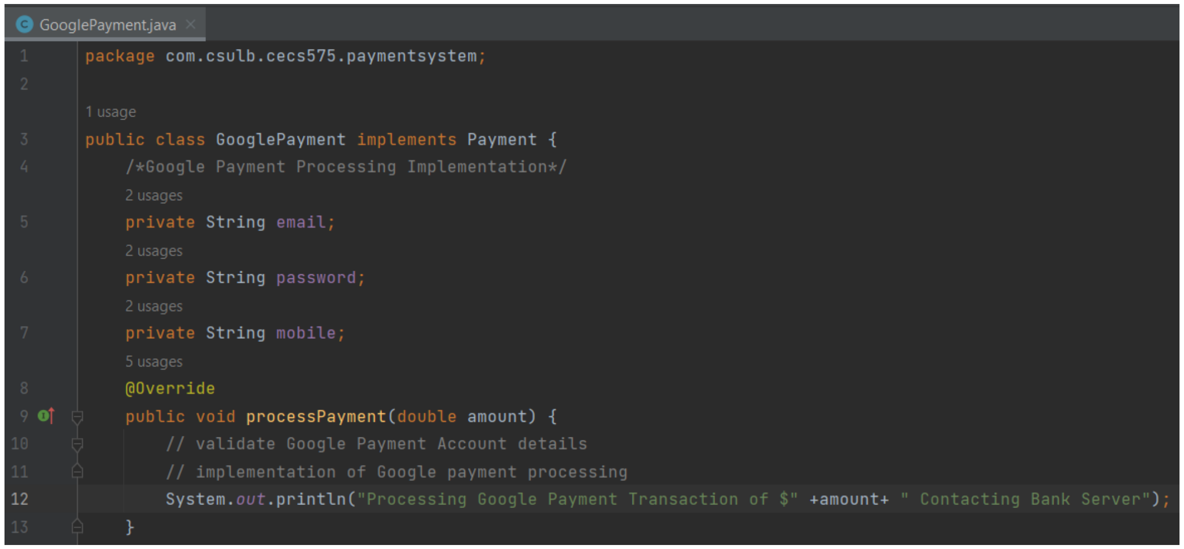
- Enumeration of different payment types supported by the food delivery system.



CreditCardPayment.java

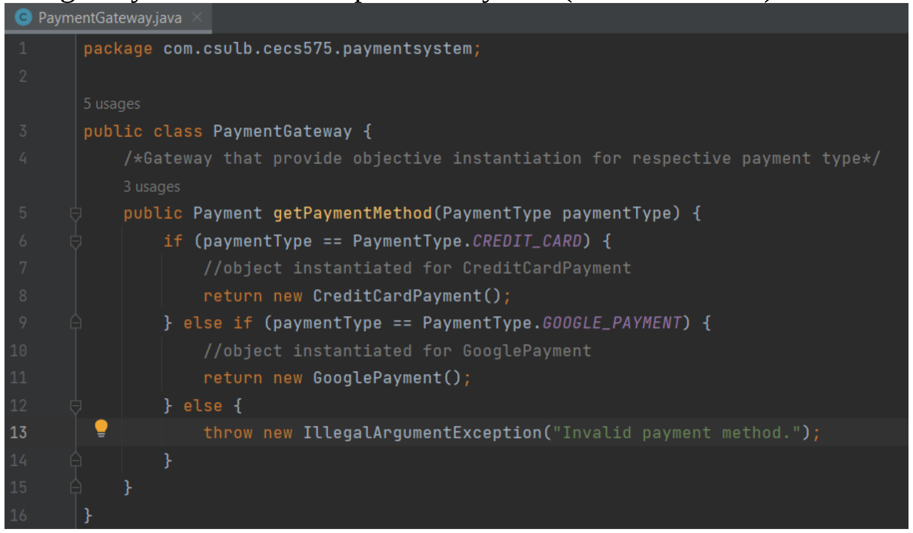
```java
package com.csulb.cecs575.paymentsystem;

1 usage
public class CreditCardPayment implements Payment {
    /*Credit Card Payment Processing Implementation*/
    2 usages
    private String cardNumber;
    2 usages
    private String cvv;
    2 usages
    private String expirationDate;

    5 usages
    @Override
    public void processPayment(double amount) {
        // validate credit card details
        // implementation of credit card payment processing
        System.out.println("Processing Credit Card Payment of $" +amount+ " Contacting Bank Server");
    }
```

- CreditCardPayment class with processPayment(double amount) method implementation.



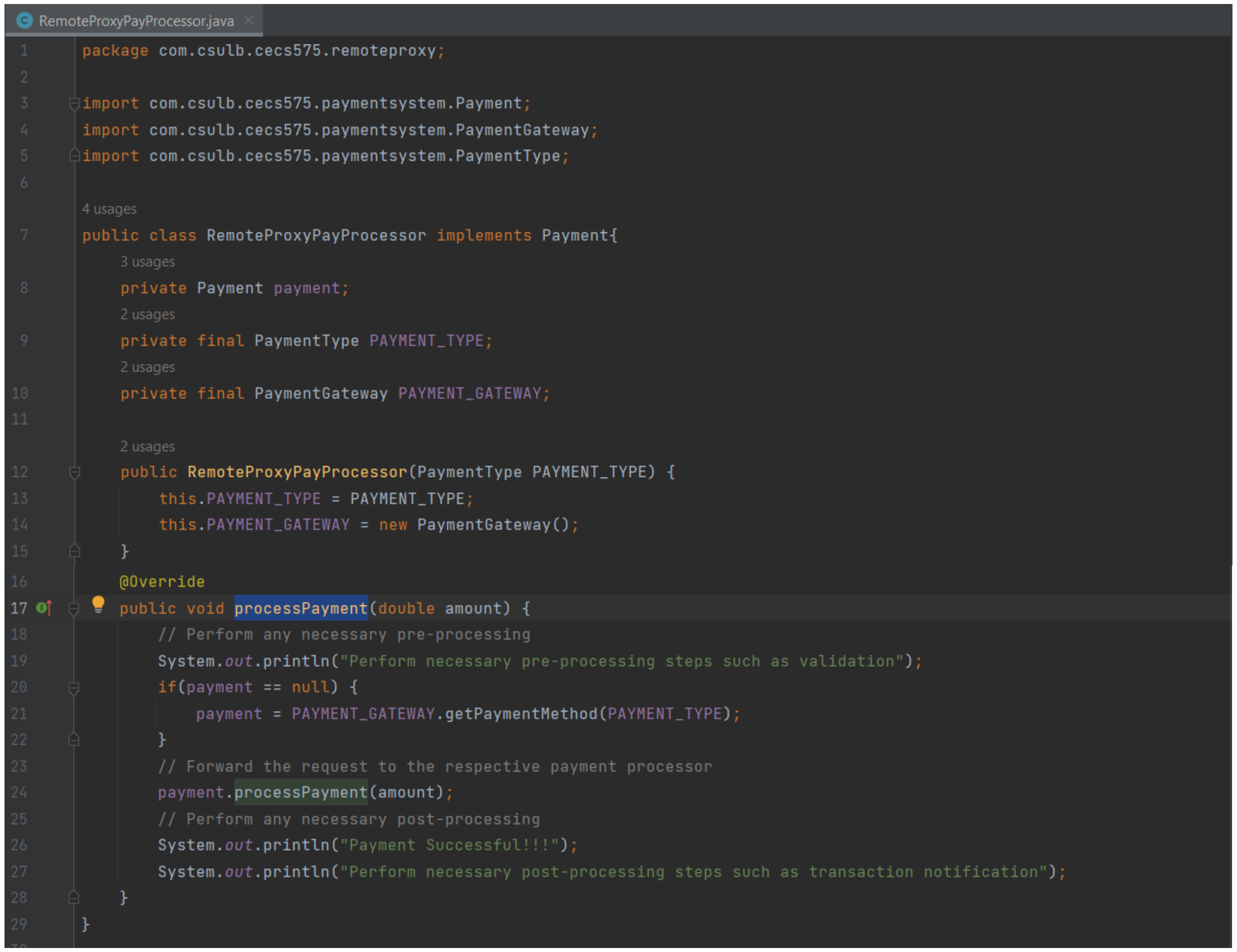
```
GooglePayment.java
package com.csulb.cecs575.paymentsystem;

1 usage
public class GooglePayment implements Payment {
    /*Google Payment Processing Implementation*/
    2 usages
    private String email;
    2 usages
    private String password;
    2 usages
    private String mobile;
    5 usages
    @Override
    public void processPayment(double amount) {
        // validate Google Payment Account details
        // implementation of Google payment processing
        System.out.println("Processing Google Payment Transaction of $" +amount+ " Contacting Bank Server");
    }
```

- GooglePayment class with processPayment(double amount) method implementation.



```
PaymentGateway.java
package com.csulb.cecs575.paymentsystem;

5 usages
public class PaymentGateway {
    /*Gateway that provide objective instantiation for respective payment type*/
    3 usages
    public Payment getPaymentMethod(PaymentType paymentType) {
        if (paymentType == PaymentType.CREDIT_CARD) {
            //object instantiated for CreditCardPayment
            return new CreditCardPayment();
        } else if (paymentType == PaymentType.GOOGLE_PAYMENT) {
            //object instantiated for GooglePayment
            return new GooglePayment();
        } else {
            throw new IllegalArgumentException("Invalid payment method.");
        }
    }
}
```

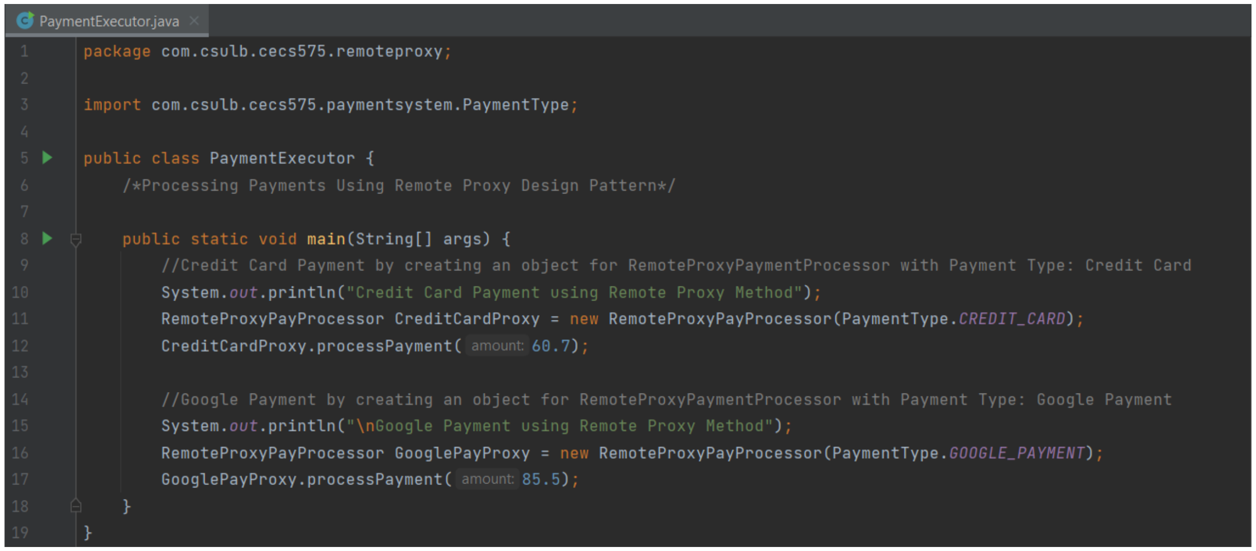
- PaymentGateway class with getPaymentMethods(PaymentType paymentType) which will return the instantiation of the appropriate payment type.



RemoteProxyPayProcessor.java

```java
package com.csulb.cecs575.remoteproxy;

import com.csulb.cecs575.paymentsystem.Payment;
import com.csulb.cecs575.paymentsystem.PaymentGateway;
import com.csulb.cecs575.paymentsystem.PaymentType;

4 usages
public class RemoteProxyPayProcessor implements Payment{
    3 usages
    private Payment payment;
    2 usages
    private final PaymentType PAYMENT_TYPE;
    2 usages
    private final PaymentGateway PAYMENT_GATEWAY;

    2 usages
    public RemoteProxyPayProcessor(PaymentType PAYMENT_TYPE) {
        this.PAYMENT_TYPE = PAYMENT_TYPE;
        this.PAYMENT_GATEWAY = new PaymentGateway();
    }
    @Override
    public void processPayment(double amount) {
        // Perform any necessary pre-processing
        System.out.println("Perform necessary pre-processing steps such as validation");
        if(payment == null) {
            payment = PAYMENT_GATEWAY.getPaymentMethod(PAYMENT_TYPE);
        }
        // Forward the request to the respective payment processor
        payment.processPayment(amount);
        // Perform any necessary post-processing
        System.out.println("Payment Successful!!!");
        System.out.println("Perform necessary post-processing steps such as transaction notification");
    }
}
```

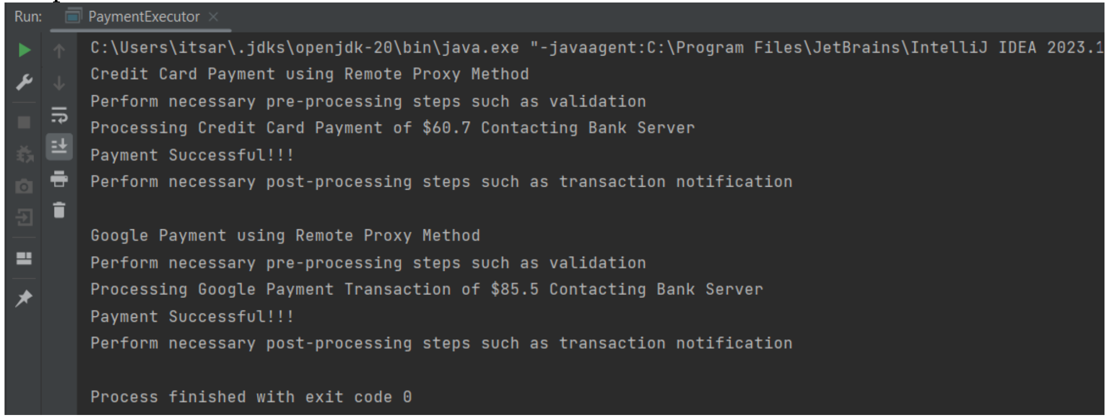
- RemoteProxyPayProcessor class with the actual references for the PaymentInterface, PaymentType, and Payment Gateway. It has the implementation of the processPayment(double amount) which will call the actual processPayment(double amount) method of the respective payment type chosen by the client.



PaymentExecutor.java

```java
package com.csulb.cecs575.remoteproxy;

import com.csulb.cecs575.paymentsystem.PaymentType;

public class PaymentExecutor {
    /*Processing Payments Using Remote Proxy Design Pattern*/

    public static void main(String[] args) {
        //Credit Card Payment by creating an object for RemoteProxyPaymentProcessor with Payment Type: Credit Card
        System.out.println("Credit Card Payment using Remote Proxy Method");
        RemoteProxyPayProcessor CreditCardProxy = new RemoteProxyPayProcessor(PaymentType.CREDIT_CARD);
        CreditCardProxy.processPayment( amount: 60.7);

        //Google Payment by creating an object for RemoteProxyPaymentProcessor with Payment Type: Google Payment
        System.out.println("\nGoogle Payment using Remote Proxy Method");
        RemoteProxyPayProcessor GooglePayProxy = new RemoteProxyPayProcessor(PaymentType.GOOGLE_PAYMENT);
        GooglePayProxy.processPayment( amount: 85.5);
    }
}
```

- ProxyPayment class contains the main method which has the representative objects for calling processPayment(amount) method of the RemoteProxyPayProcessor, which will navigate to the RemoteProxyPayProcessor class and interact with the respective payment processing methods implemented using the Payment Interface. CreditCardProxy links to the credit card payment type and GooglePayProxy links to the google payment type.
- Output:



Run: PaymentExecutor

```
C:\Users\itsar\.jdks\openjdk-20\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.1
Credit Card Payment using Remote Proxy Method
Perform necessary pre-processing steps such as validation
Processing Credit Card Payment of $60.7 Contacting Bank Server
Payment Successful!!!
Perform necessary post-processing steps such as transaction notification

Google Payment using Remote Proxy Method
Perform necessary pre-processing steps such as validation
Processing Google Payment Transaction of $85.5 Contacting Bank Server
Payment Successful!!!
Perform necessary post-processing steps such as transaction notification

Process finished with exit code 0
```