SOLID PRINCIPLES

CONTENT

SINGLE RESPONSIBILITY

Single Responsibility Principle (SRP):

A class should have only one reason to change, meaning that a class should have only one responsibility or job

Responsibility: Storing Payment Information
The primary responsibility of the **PaymentDetails** class is to store information related to a payment.

```
public class PaymentDetails {
 private String paymentToken;
 private double amount;
 public PaymentDetails() {
 super();
 this.paymentToken = paymentToken;
 this.amount = amount;
 }
 public String getPaymentToken() {
 return paymentToken;
 }
 public void setPaymentToken(String paymentToken) {
 this.paymentToken = paymentToken;
 }
 public double getAmount() {
 return amount;
 }
 public void setAmount(double amount) {
 this.amount = amount;
 }}
```

OPEN/CLOSED PRINCIPLE

Open/Closed Principle (OCP):
Software entities (classes, modules, functions, etc.) should be open for extension but closed for modification.

BasePayment Class:
The **BasePayment** class is declared as an abstract class and implements the **PaymentProcessor** interface.
This class is open for extension because you can create subclasses that inherit from it.

```
public abstract class BasePayment implements PaymentProcessor{
 public boolean processPayment(String paymentToken, double amount) {
   System.out.println("Basic payment logic for all payment method");
 return true;
 }
 }
```

OPEN/CLOSED PRINCIPLE

APIPayment Class:

- The **APIPayment** class extends **BasePayment** and overrides the **processPayment** method. It introduces new behavior specific to API payments without modifying the existing logic in the base class.

```
public class APIPayment extends BasePayment{
 private static final double DISCOUNT = 50; //
   public boolean processPayment(String paymentToken, double amount) {
 System.out.println("API Payment");
 double totalAmount = calculateTotalAmount(amount);
 System.out.println("Original Amount: " + amount);
 System.out.println("Total Amount with Discount: " + totalAmount);
 return true; // Return success or failure
 }
 public double calculateTotalAmount(double originalAmount) {
 return (originalAmount-DISCOUNT);
 }
 @Override
 public boolean processPayment(PaymentDetails paymentDetails) {
                double totalAmount = calculateTotalAmount(paymentDetails.getAmount());
 System.out.println("API Payment");
 System.out.println("Original Amount: " + paymentDetails.getAmount());
 System.out.println("Total Amount with Discount: " + totalAmount);
 return true;
 }
}
```

OPEN/CLOSED PRINCIPLE

CreditCardPayment Class:

- Similar to **APIPayment**, the **CreditCardPayment** class extends **BasePayment** and introduces new behavior specific to credit card payments without modifying the existing logic in the base class. It overrides the **processPayment** method and adds a new method, **calculateTotalAmount**, to calculate the total amount with a 5% tax rate.

```
public class CreditCardPayment extends BasePayment {
 private static final double TAX_RATE = 0.05; // 5% tax rate
 @Override
   public boolean processPayment(String paymentToken, double amount) {
 double totalAmount = calculateTotalAmount(amount);
 System.out.println("Card Payment");
 System.out.println("Original Amount: " + amount);
 System.out.println("Total Amount with 5% Tax: " + totalAmount);
 return true; // Return success or failure
 }
 @Override
 public boolean processPayment(PaymentDetails paymentDetails) {
                double totalAmount = calculateTotalAmount(paymentDetails.getAmount());
 System.out.println("Card Payment");
 System.out.println("Original Amount: " + paymentDetails.getAmount());
 System.out.println("Total Amount with 5% Tax: " + totalAmount);
 return true;
 }
 public double calculateTotalAmount(double originalAmount) {
 return originalAmount + (originalAmount * TAX_RATE);
 }
}
```

LISKOV SUBSTITUTION PRINCIPLE

Liskov substituton Principle:

The Liskov Substitution Principle (LSP) is about ensuring that objects of a base class can be replaced with objects of a derived class without affecting the correctness of the program.

- **PaymentProcessor** is the base interface representing the common behavior of all payment processors.
- **BasePayment** is an abstract class implementing common logic shared by all payment processors. It serves as a base class for specific payment processors.

```
public interface PaymentProcessor {

    boolean processPayment(PaymentDetails paymentDetails);
}
```

```
public abstract class BasePayment implements PaymentProcessor{
  public boolean processPayment(String paymentToken, double amount) {
 System.out.println("Basic payment logic for all payment method");
 return true;
 }
 }
```

LISKOV SUBSTITUTION PRINCIPLE

Liskov substituton Principle:

- **CreditCardPaymentProcessor**, and API**PaymentProcessor** are concrete classes that inherit from **BasePayment** and provide specific implementations for their respective payment methods.

```
public class APIPayment extends BasePayment{
 private static final double DISCOUNT = 50; //
 public boolean processPayment(String paymentToken, double amount)
{
 System.out.println("API Payment");
 double totalAmount = calculateTotalAmount(amount);
 System.out.println("Original Amount: " + amount);
 System.out.println("Total Amount with Discount: " + totalAmount);
 return true; // Return success or failurE
 }
 public double calculateTotalAmount(double originalAmount)
 return (originalAmount-DISCOUNT);
 }
 @Override
 public boolean processPayment(PaymentDetails paymentDetails) {
 double totalAmount =
calculateTotalAmount(paymentDetails.getAmount());
 System.out.println("API Payment");
 System.out.println("Original Amount: " + paymentDetails.getAmount());
 System.out.println("Total Amount with Discount: " + totalAmount);
 return true;
 }
}
```

LISKOV SUBSTITUTION PRINCIPLE

```
public class CreditCardPayment extends BasePayment {
private static final double TAX_RATE = 0.05; // 5% tax rate
 @Override
 public boolean processPayment(String paymentToken, double amount) {
 double totalAmount = calculateTotalAmount(amount);
 System.out.println("Card Payment");
 System.out.println("Original Amount: " + amount);
 System.out.println("Total Amount with 5% Tax: " + totalAmount);
 return true; // Return success or failure
 }
 @Override
 public boolean processPayment(PaymentDetails paymentDetails) {
 double totalAmount = calculateTotalAmount(paymentDetails.getAmount());
 System.out.println("Card Payment");
 System.out.println("Original Amount: " + paymentDetails.getAmount());
 System.out.println("Total Amount with 5% Tax: " + totalAmount);
 return true;
 }
 public double calculateTotalAmount(double originalAmount) {
 return originalAmount + (originalAmount * TAX_RATE);
 }
}
```

INTERFACE SEGREGATION

Interface Segregation Principle (ISP):

- A class should not be forced to implement interfaces it does not use.

```
public interface PaymentProcessor {
 boolean processPayment(PaymentDetails paymentDetails);
 }
public abstract class BasicPaymentinterface {
  boolean processBasicPayment(String paymentToken, double amount);
}

public abstract class BasePayment implements PaymentProcessor{
 public boolean processPayment(String paymentToken, double amount)
{
 System.out.println("Basic payment logic for all payment method");
 return true;
 }
 }
```

DEPENDENCY INVERSION PRINCIPLE

Dependency Inversion Principle (DIP):
High-level modules should not depend on low-level modules. Both should depend on abstractions

- **IMdbProMembership** now takes a **PaymentProcessor** instead of a specific implementation (**creditCardProcessor**). This adheres to DIP by depending on the abstraction (**PaymentProcessor** interface) rather than a concrete implementation.

```
public interface PaymentHandler {
 void handlePayment(PaymentDetails paymentDetails);
}

public class IMdbProMembership implements PaymentHandler {

 public IMdbProMembership(PaymentProcessor creditCardProcessor) {
 System.out.println(“IMDB PRO UPGRADED”);
 }

 public void handlePayment(PaymentDetails paymentDetails) {
 System.out.println(“IMDB PRO UPGRADED”);
 }

}
```