Object-Oriented Methodology HW 11

2024 Fall Semester

21 CST H3Art

Design Problem: Adapter Pattern in E-commerce Development

Problem Statement:

Imagine you are working on a project where you need to integrate a new payment gateway system into an existing e-commerce platform. The existing platform expects a specific interface with methods like `processPayment()`, `authorizeTransaction()`, and `refundPayment()`. However, the new payment gateway provides a different interface with methods named `charge()`, `confirmAuthorization()`, and `issueRefund()`.

How would you use the Adapter Pattern to bridge this interface mismatch and allow the existing platform to interact seamlessly with the new payment gateway?

Interface expected by the existing platform:

```java
public interface OldPayment {
  void processPayment(double amount);
  void authorizeTransaction(String transactionId);
  void refundPayment(String transactionId, double amount);
}
```

Interface provided by the new payment gateway:

```java
public interface NewPayment {
  void charge(double amount);
  void confirmAuthorization(String transactionId);
  void issueRefund(String transactionId, double amount);
}
```

New payment gateway that implements NewPayment interface:

```java
public class NewPaymentGateway implements NewPayment {
  public void charge(double amount) {
    System.out.println("Charging: $" + amount);
  }
  public void confirmAuthorization(String transactionId) {
    System.out.println("Confirming authorization: " + transactionId);
  }
  public void issueRefund(String transactionId, double amount) {
    System.out.println("Issuing refund: " + transactionId + " Amount: $" + amount);
  }
}
```

1. Write a simple Java code snippet demonstrating the Adapter class and how it would be used;

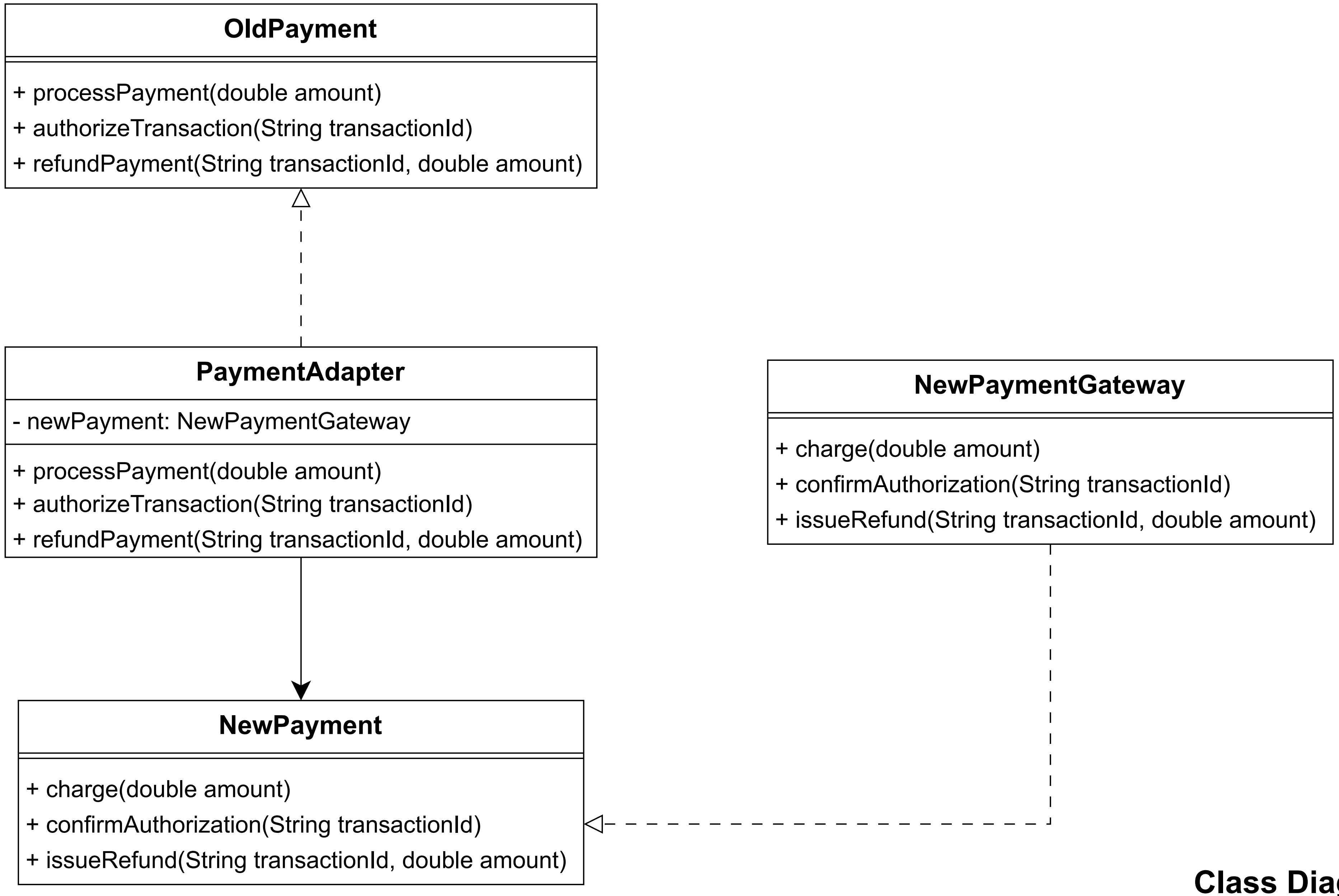
The Adapter class is as follows:

```java
public class PaymentAdapter implements OldPayment {
    private NewPayment newPaymentGateway;

    public PaymentAdapter(NewPayment newPaymentGateway) {
        this.newPaymentGateway = newPaymentGateway;
    }

    @Override
    public void processPayment(double amount) {
        newPaymentGateway.charge(amount);
    }

    @Override
    public void authorizeTransaction(String transactionId) {
        newPaymentGateway.confirmAuthorization(transactionId);
    }

    @Override
    public void refundPayment(String transactionId, double amount) {
        newPaymentGateway.issueRefund(transactionId, amount);
    }
}
```

To demonstrate its usage, I wrote the following simple Java code snippet:

```java
public class ECommercePlatform {
    public static void main(String[] args) {
        NewPayment newPaymentGateway = new NewPaymentGateway();
        OldPayment paymentAdapter = new PaymentAdapter(newPaymentGateway);

        // Use the adapter as if it's the old interface
        paymentAdapter.processPayment(100.0);
        paymentAdapter.authorizeTransaction("H3Art12345");
        paymentAdapter.refundPayment("H3Art12345", 50.0);
    }
}
```

2. Draw the class diagram.


OldPayment
+ processPayment(double amount)
+ authorizeTransaction(String transactionId)
+ refundPayment(String transactionId, double amount)
PaymentAdapter
- newPayment: NewPaymentGateway
+ processPayment(double amount)
+ authorizeTransaction(String transactionId)
+ refundPayment(String transactionId, double amount)
NewPaymentGateway
+ charge(double amount)
+ confirmAuthorization(String transactionId)
+ issueRefund(String transactionId, double amount)
NewPayment
+ charge(double amount)
+ confirmAuthorization(String transactionId)
+ issueRefund(String transactionId, double amount)

Class Diagram
21 CST H3Art