Exercise 4: Implementing the Adapter Pattern

Objective

Integrate multiple third-party payment gateways into a single interface using the Adapter Pattern, enabling uniform access to diverse APIs.

Complete Java Code (Single File for Simplicity)

// File: AdapterPatternExample.java

// Step 1: Target Interface interface PaymentProcessor {

void processPayment(double amount);

}

// Step 2: Adaptee - PayPal gateway class PayPal {

public void sendPayment(double amountInDollars) {

System.out.println("Payment of $" + amountInDollars + " processed via PayPal.");

}

}

// Step 3: Adaptee - Stripe gateway class Stripe {

public void makeStripePayment(double amount) { System.out.println("Stripe charged $" + amount + " to your

card.");

}

}

// Step 4: Adapter for PayPal

class PayPalAdapter implements PaymentProcessor { private PayPal payPal;

public PayPalAdapter(PayPal payPal) { this.payPal = payPal;

}

@Override

public void processPayment(double amount) { payPal.sendPayment(amount);

}

}

// Step 5: Adapter for Stripe

class StripeAdapter implements PaymentProcessor { private Stripe stripe;

public StripeAdapter(Stripe stripe) { this.stripe = stripe;

}

@Override

public void processPayment(double amount) { stripe.makeStripePayment(amount);

}

}

// Step 6: Test Class

public class AdapterPatternExample {

public static void main(String[] args) {

// Using PayPal through adapter

PaymentProcessor paypalProcessor = new PayPalAdapter(new PayPal());

paypalProcessor.processPayment(500.00);

// Using Stripe through adapter

PaymentProcessor stripeProcessor = new StripeAdapter(new Stripe());

stripeProcessor.processPayment(1200.00);

}

}

Sample Output

Payment of $500.0 processed via PayPal. Stripe charged $1200.0 to your card.

Advantages of Adapter Pattern

Allows incompatible interfaces to work together.

Adds new functionality without changing existing code. Ideal for integrating third-party libraries or APIs.