Payment Processing System with Adapter Pattern

public class AdapterPatternExample {

public static void main(String[] args) {

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

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

PaymentProcessor bankProcessor = new BankAdapter(new BankGateway());

System.out.println("=== Processing Payments ===");

processPayment(stripeProcessor, 99.99, "USD", "cust\_12345");

processPayment(paypalProcessor, 49.95, "EUR", "user\_789");

processPayment(bankProcessor, 199.00, "GBP", "acc\_456");

System.out.println("\n=== Processing Refunds ===");

processRefund(stripeProcessor, 25.00, "USD", "cust\_12345");

processRefund(paypalProcessor, 10.50, "EUR", "user\_789");

processRefund(bankProcessor, 50.00, "GBP", "acc\_456");

}

private static void processPayment(PaymentProcessor processor, double amount,

String currency, String customerId) {

System.out.printf("\nProcessing %.2f %s payment for %s using %s%n",

amount, currency, customerId, processor.getClass().getSimpleName());

processor.processPayment(amount, currency, customerId);

}

private static void processRefund(PaymentProcessor processor, double amount,

String currency, String customerId) {

System.out.printf("\nProcessing %.2f %s refund for %s using %s%n",

amount, currency, customerId, processor.getClass().getSimpleName());

processor.processRefund(amount, currency, customerId);

}

interface PaymentProcessor {

void processPayment(double amount, String currency, String customerId);

void processRefund(double amount, String currency, String customerId);

}

//stripe gateway

static class StripeGateway {

public void makeStripePayment(int amountInCents, String currencyCode, String stripeCustomerId) {

System.out.printf("Stripe: Charging %d %s to customer %s%n",

amountInCents, currencyCode, stripeCustomerId);

}

public void issueStripeRefund(int amountInCents, String currencyCode, String stripeCustomerId) {

System.out.printf("Stripe: Refunding %d %s to customer %s%n",

amountInCents, currencyCode, stripeCustomerId);

}

}

// PayPal Gateway

static class PayPalGateway {

public void sendPayPalPayment(double amount, String currencyType, String payPalEmail) {

System.out.printf("PayPal: Sending %.2f %s to %s%n",

amount, currencyType, payPalEmail);

}

public void sendPayPalRefund(double amount, String currencyType, String payPalEmail) {

System.out.printf("PayPal: Refunding %.2f %s to %s%n",

amount, currencyType, payPalEmail);

}

}

// Bank Gateway

static class BankGateway {

public void initiateBankTransfer(double amount, String currency, String accountNumber) {

System.out.printf("Bank: Transferring %.2f %s to account %s%n",

amount, currency, accountNumber);

}

public void reverseBankTransfer(double amount, String currency, String accountNumber) {

System.out.printf("Bank: Reversing %.2f %s from account %s%n",

amount, currency, accountNumber);

}

}

// Adapter for Stripe

static class StripeAdapter implements PaymentProcessor {

private final StripeGateway stripeGateway;

public StripeAdapter(StripeGateway stripeGateway) {

this.stripeGateway = stripeGateway;

}

@Override

public void processPayment(double amount, String currency, String customerId) {

int amountInCents = (int) Math.round(amount \* 100);

stripeGateway.makeStripePayment(amountInCents, currency, customerId);

}

@Override

public void processRefund(double amount, String currency, String customerId) {

int amountInCents = (int) Math.round(amount \* 100);

stripeGateway.issueStripeRefund(amountInCents, currency, customerId);

}

}

// Adapter for PayPal

static class PayPalAdapter implements PaymentProcessor {

private final PayPalGateway payPalGateway;

public PayPalAdapter(PayPalGateway payPalGateway) {

this.payPalGateway = payPalGateway;

}

@Override

public void processPayment(double amount, String currency, String customerId) {

payPalGateway.sendPayPalPayment(amount, currency, customerId);

}

@Override

public void processRefund(double amount, String currency, String customerId) {

payPalGateway.sendPayPalRefund(amount, currency, customerId);

}

}

// Adapter for Bank

static class BankAdapter implements PaymentProcessor {

private final BankGateway bankGateway;

public BankAdapter(BankGateway bankGateway) {

this.bankGateway = bankGateway;

}

@Override

public void processPayment(double amount, String currency, String customerId) {

bankGateway.initiateBankTransfer(amount, currency, customerId);

}

@Override

public void processRefund(double amount, String currency, String customerId) {

bankGateway.reverseBankTransfer(amount, currency, customerId);

}

}

}

Key Features:

1. **Unified Interface**: PaymentProcessor provides consistent methods for all gateways
2. **Gateway Adapters**: Each adapter translates the standard interface to specific gateway APIs
3. **Payment and Refund**: Demonstrates both payment and refund operations
4. **Currency Handling**: Shows different currency representations (dollars vs cents)
5. **Clean Client Code**: Payment processing code doesn't need to know about gateway specifics

Adapter Pattern Benefits:

1. **Integration**: Lets incompatible interfaces work together
2. **Reusability**: Existing gateway implementations can be reused without modification
3. **Single Responsibility**: Each adapter handles one gateway's translation
4. **Open/Closed Principle**: New gateways can be added without changing existing code
5. **Simplified Client Code**: Clients work with one consistent interface\

Output

