14 Structural: Adapter Pattern for a Payment Gateway Integration

**Goal** Integrate a new third-party payment SDK whose API doesn’t match your system’s PaymentProcessor interface.  
 You will implement **both** Object Adapter and Class Adapter solutions, compare them, write unit tests, and reflect on the trade-offs.

#### **Tasks**

1. **Problem analysis** (analysis/adapter\_problems.md)  
   * Describe why direct use of ThirdPartyPaymentGateway violates the Liskov Substitution Principle with respect to PaymentProcessor.
   * List two risks of hard-coding third-party APIs into business services.
2. **Object Adapter implementation** (src/main/java/objectadapter/)  
   * PaymentProcessor (target interface)
   * ThirdPartyPaymentGateway (adaptee—leave unchanged)
   * PaymentAdapter (wraps a gateway instance and implements PaymentProcessor)
   * Driver class ObjectAdapterDemo that processes a payment.
3. **Class Adapter implementation** (src/main/java/classadapter/)  
   * ClassPaymentAdapter extends ThirdPartyPaymentGateway & implements PaymentProcessor.
   * Driver class ClassAdapterDemo.
4. **Unit tests** (src/test/java/)  
   * ObjectAdapterPaysTest – assert correct console output for PaymentAdapter.
   * ClassAdapterPaysTest – assert correct console output for ClassPaymentAdapter.
   * ObjectVsClassFlexibilityTest – prove object adapter can adapt two different gateways (create a stub second gateway) while class adapter cannot (single inheritance).
5. **Reflection** (reflection.md)  
   * Compare object vs. class adapter on flexibility, testability, and coupling.
   * When would you choose one over the other in production?
   * How would the design change if the gateway suddenly required an auth token on every call?

#### **Deliverables**

analysis/adapter\_problems.md

src/main/java/objectadapter/PaymentProcessor.java

src/main/java/objectadapter/ThirdPartyPaymentGateway.java

src/main/java/objectadapter/PaymentAdapter.java

src/main/java/objectadapter/ObjectAdapterDemo.java

src/main/java/classadapter/ClassPaymentAdapter.java

src/main/java/classadapter/ClassAdapterDemo.java

src/test/java/ObjectAdapterPaysTest.java

src/test/java/ClassAdapterPaysTest.java

src/test/java/ObjectVsClassFlexibilityTest.java

reflection.md

README.md

## **Detailed Solution**

### **1 Target Interface & Adaptee**

/\* objectadapter/PaymentProcessor.java \*/

package objectadapter;

public interface PaymentProcessor {

void processPayment(double amount);

}

/\* objectadapter/ThirdPartyPaymentGateway.java \*/

package objectadapter;

/\*\* Incompatible API we cannot modify. \*/

public class ThirdPartyPaymentGateway {

public void makePayment(String currency,double amount){

System.out.printf("Processing payment of %.2f %s through ThirdPartyPaymentGateway.%n",

amount,currency);

}

}

### **2 Object Adapter**

/\* objectadapter/PaymentAdapter.java \*/

package objectadapter;

public class PaymentAdapter implements PaymentProcessor {

private final ThirdPartyPaymentGateway gateway;

public PaymentAdapter(ThirdPartyPaymentGateway gateway){

this.gateway = gateway;

}

@Override public void processPayment(double amount){

gateway.makePayment("USD",amount); // adaptation

}

}

/\* objectadapter/ObjectAdapterDemo.java \*/

package objectadapter;

public class ObjectAdapterDemo {

public static void main(String[] args){

PaymentProcessor proc = new PaymentAdapter(new ThirdPartyPaymentGateway());

proc.processPayment(100.00);

}

}

**Console**

Processing payment of 100.00 USD through ThirdPartyPaymentGateway.

### **3 Class Adapter**

/\* classadapter/ClassPaymentAdapter.java \*/

package classadapter;

import objectadapter.\*;

/\*\* Uses inheritance to adapt. \*/

public class ClassPaymentAdapter extends ThirdPartyPaymentGateway

implements PaymentProcessor {

@Override public void processPayment(double amount){

makePayment("USD",amount);

}

}

/\* classadapter/ClassAdapterDemo.java \*/

package classadapter;

import objectadapter.PaymentProcessor;

public class ClassAdapterDemo {

public static void main(String[] args){

PaymentProcessor proc = new ClassPaymentAdapter();

proc.processPayment(150.00);

}

}

**Console**

Processing payment of 150.00 USD through ThirdPartyPaymentGateway.

### **4 Key Unit-Test Snippets**

/\* ObjectAdapterPaysTest \*/

PaymentProcessor p = new PaymentAdapter(new ThirdPartyPaymentGateway());

assertDoesNotThrow(() -> p.processPayment(10));

/\* ObjectVsClassFlexibilityTest \*/

class SecondGateway extends ThirdPartyPaymentGateway{}

PaymentProcessor p1 = new PaymentAdapter(new ThirdPartyPaymentGateway());

PaymentProcessor p2 = new PaymentAdapter(new SecondGateway()); // OK

PaymentProcessor p3 = new ClassPaymentAdapter(); // only 1 type

### **5 Reflection (summary)**

| **Criterion** | **Object Adapter** | **Class Adapter** |
| --- | --- | --- |
| **Implementation** | composition | single inheritance |
| **Can adapt multiple gateways?** | ✔ Yes | ✖ No |
| **Requires new subclass when gateway signature changes?** | No | Yes |
| **Testability** | Mock the gateway easily | Hard-wired inheritance |

*If an auth token became mandatory you’d add it to PaymentAdapter constructor, leaving PaymentProcessor users untouched. With a class adapter you’d instead modify the subclass or create a new one, risking ripple changes.*

By completing this assignment you show mastery in applying the Adapter pattern with both composition and inheritance, and in choosing the right variant for flexibility versus simplicity.