**Exercise 4: Implementing the Adapter Pattern**

**Scenario:**

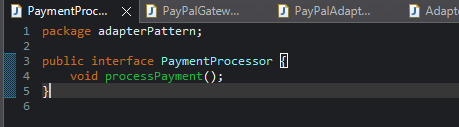
You are developing a payment processing system that needs to integrate with multiple third-party payment gateways with different interfaces. Use the Adapter Pattern to achieve this.

**Steps:**

1. **Create a New Java Project:**
   * Create a new Java project named **AdapterPatternExample**.
2. **Define Target Interface:**
   * Create an interface **PaymentProcessor** with methods like **processPayment()**.
3. **Implement Adaptee Classes:**
   * Create classes for different payment gateways with their own methods.
4. **Implement the Adapter Class:**
   * Create an adapter class for each payment gateway that implements PaymentProcessor and translates the calls to the gateway-specific methods.
5. **Test the Adapter Implementation:**
   * Create a test class to demonstrate the use of different payment gateways through the adapter.

**Solution:**

**PaymentProcessor.java**



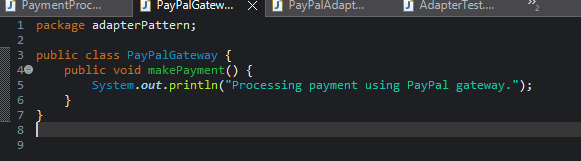
This is the interface expected by your application.

It defines a standard method (processPayment()) that every payment system must implement.

Your application code will use only this interface — not the actual payment gateway classes.

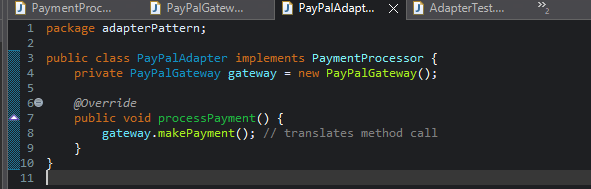
**PayPalGateway.java**

* This is the existing or third-party payment class (e.g., a PayPal SDK).
* It has its own method (makePayment()), which is not compatible with your PaymentProcessor interface.
* You cannot change this class (e.g., if it's from a third-party library), so you need to adapt it.



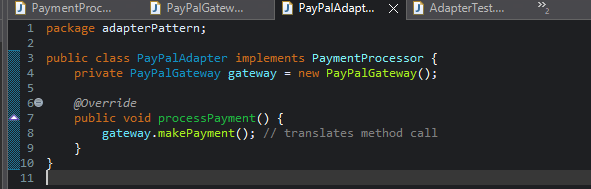
**PaypalAdapter.java**

* This class bridges the PaymentProcessor interface and PayPalGateway class.
* It implements PaymentProcessor, so it can be used by client code.
* Inside processPayment(), it delegates the call to the adaptee’s makePayment() method.
* It translates the method name and logic from the third-party class to fit your system's expectations.

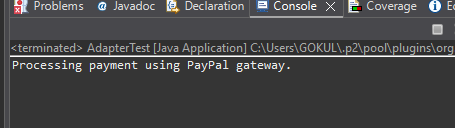
****

**AdapterTest.java**

* This is the main application code that uses the adapter.
* It works with the PaymentProcessor interface and is unaware of the internal PayPalGateway implementation.
* This makes the code flexible and easily swappable with other adapters (e.g., StripeAdapter, RazorpayAdapter) without any changes in the logic.

****

**Output:**

**.**