**Understanding the Strategy Pattern**

The Strategy Pattern is a design approach that allows you to define a family of algorithms, encapsulate each one, and make them interchangeable. This means you can switch between different algorithms without altering the clients that use them.

**Use Case**: Payment Processing System:

Imagine you run an online store that gives customers various payment options, such as credit cards, PayPal, or cryptocurrencies. Your payment processing system needs to easily switch between these methods based on the user’s choice.

Code Breakdown

**Interface**: PaymentMethod: This interface sets the standard for different payment methods.

Main Method: It includes a method executePayment responsible for processing payments.

**Concrete Payment Methods**

CreditCardMethod: This class implements the PaymentMethod interface and handles payments via credit card. It simply prints a message showing that the payment is being processed with a credit card.

PayPalMethod: Similar to the credit card method, this class also implements PaymentMethod but focuses on processing payments through PayPal. It displays a message indicating that a PayPal payment is in progress.

CryptoPaymentMethod: Another implementation of PaymentMethod, this class processes payments using cryptocurrency, providing feedback in the form of a printed message.

**Payment Handler**

PaymentHandler: This class serves as a bridge between the client (the main code) and the specific payment methods.

setPaymentMethod(PaymentMethod paymentMethod): This method allows the client to choose which payment method to use.

handlePayment(double amount): This method takes an amount and calls the appropriate executePayment method based on the currently selected payment method.

**Main Class**: PaymentMethodDemo

Purpose: This is where the program begins execution.

Functionality:

It creates an instance of `PaymentHandler`.

It demonstrates how to set different payment methods and process payments through the handlePayment() method.

**Summary**

This code is a practical implementation of the Strategy Pattern, as it neatly encapsulates different payment methods in their own classes. The PaymentHandler allows the program to dynamically choose the payment method based on user input, enhancing flexibility and maintainability.

Key Takeaways

The PaymentMethod interface ensures all payment methods follow a common contract.

Each payment method class contains specific logic for processing payments.

The PaymentHandler decouples the main program from the concrete payment implementations.

You can easily switch payment methods with the setPaymentMethod method.

The handlePayment method manages the actual payment process by delegating to the selected method.

This design not only promotes code reuse but also makes it easier to add new payment methods in the future without altering the existing code structure.