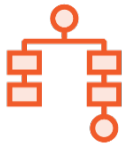
C# Design Patterns: Chain of Responsibility

CHAIN OF RESPONSIBILITY PATTERN

Filip Ekberg
PRINCIPAL CONSULTANT & CEO

@fekberg fekberg.com

Course Overview



Understanding and implementing the chain of responsibility

Identifying and leveraging existing implementations

Understanding the benefits and tradeoffs

Chain of Responsibility Pattern Characteristics

Sender

Invokes the Handler

Handler

Runs through the chain of receivers

Receiver

Handles the given command

Chain of Responsibility Pattern Characteristics

Sender

Calls Logger.Log()

Handler

Logger has a chain of ILoggers

Receiver

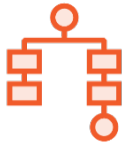
Console Logger
File Logger
Database Logger

One or many handlers can process the request

Demo

Chain of Responsibility: First Look

What Did We Achieve?



A more extensible, object oriented and dynamic implementation

Easily re-arrange in what order the handlers operate

Cleaner approach with single responsibility in mind

Decouple your code and achieve a cleaner, more extensible code base

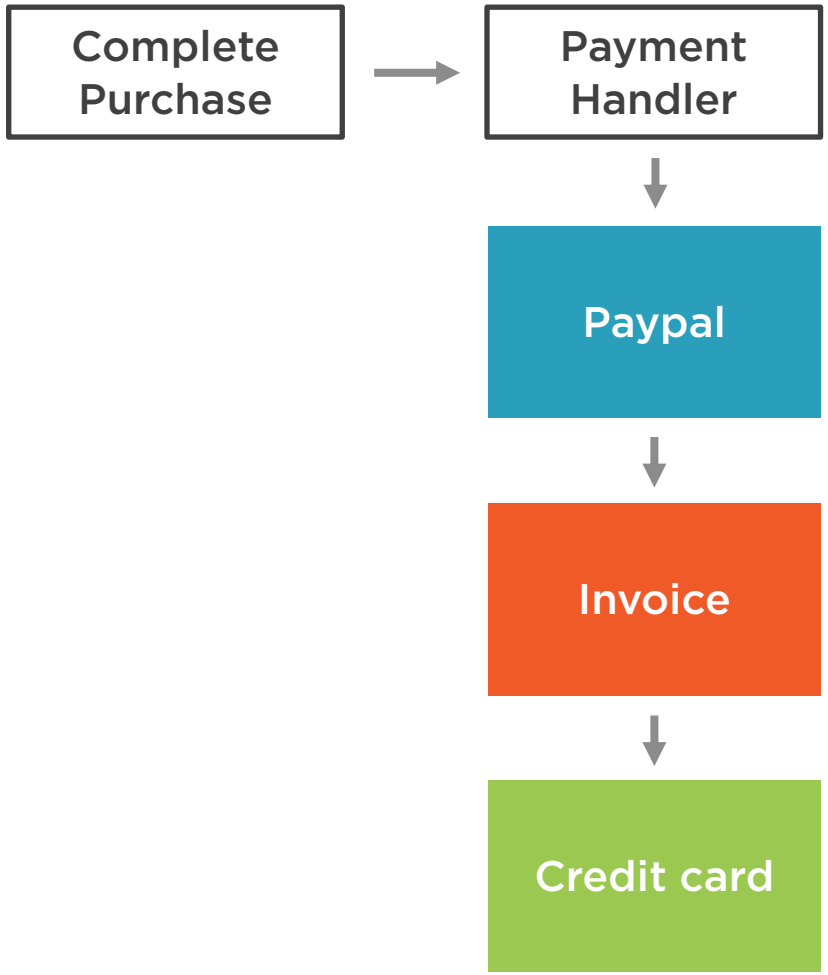
Demo

Example: Payment processing

Example: Payment processing

```
foreach (var payment in order.SelectedPayments)
{
    if (payment.PaymentProvider == PaymentProvider.Paypal)
    { }
    else if (payment.PaymentProvider == PaymentProvider.Invoice)
    { }
    else if (payment.PaymentProvider == PaymentProvider.CreditCard)
    { }
    else { }
}
```

Example: Payment processing



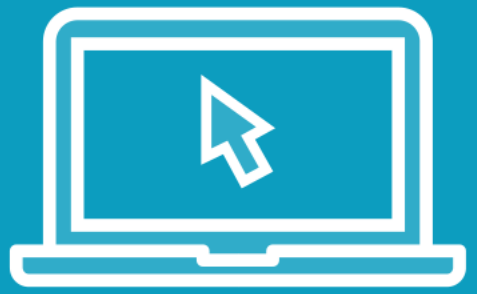
Demo

Example: Improving the payment processing

Demo

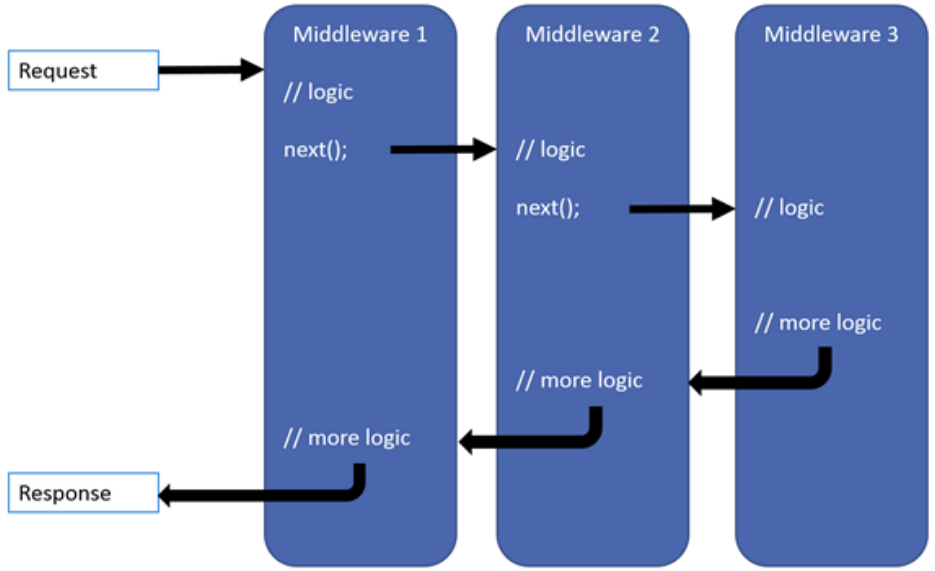
Existing implementations: ILogger

Demo



Existing implementations: ASP.NET Core processing chain

ASP.NET Core Middelwares



https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/

Decouples the sender and receiver

One or many handlers can act on a given request

Allows for clean and single responsibility handlers

An object oriented way to express a chain of “if”, “else if” and “else” statements

Easily extend a chain to add additional handlers