# Behavioral Pattern: Chain of Responsibility

**Kevin Dockx**
Architect

@KevinDockx https://www.kevindockx.com

## Coming Up

**Describing the chain of responsibility pattern**

- Document validation and approval chain

**Structure of the chain of responsibility pattern**

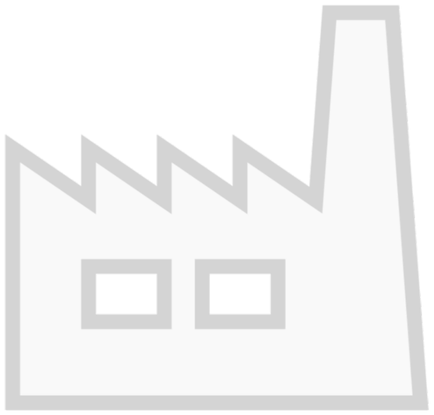
## Coming Up

Use cases for this pattern

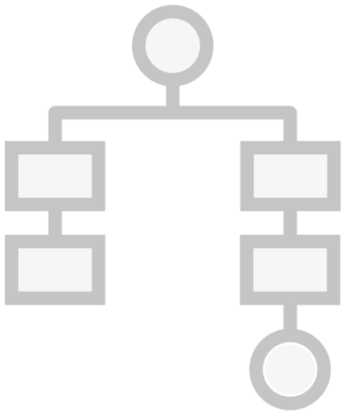
Pattern consequences

Related patterns

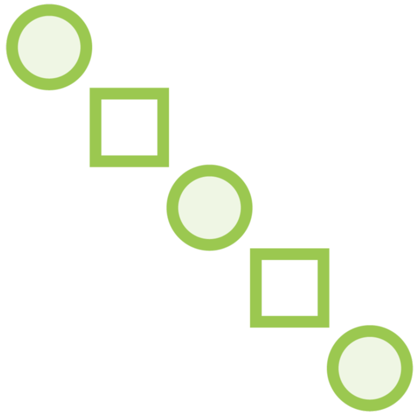
## Describing the Chain of Responsibility Pattern



Creational



Structural



**Behavioral**

## Chain of Responsibility

**The intent of this pattern is to avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. It does that by chaining the receiving objects and passing the request along the chain until an object handles it.**

```
public bool Validate() {
     if (document.Title == string.Emtpy)
     { return false; }

     if (document.LastModified < DateTime.UtcNow.AddDays(-30))
     { return false; }

     if (!document.ApprovedByLitigation)
     { return false; }

     if (!document.ApprovedByManagement)
     { return false; }

     return true; }
```

## Describing the Chain of Responsibility Pattern

```
public bool Validate() {
     if (document.Title == string.Emtpy)
     { return false; }

     if (document.LastModified < DateTime.UtcNow.AddDays(-30))
     { return false; }

     if (!document.ApprovedByLitigation)
     { return false; }

     if (!document.ApprovedByManagement)
     { return false; }

     return true; }
```

## Describing the Chain of Responsibility Pattern

```
public bool Validate() {
     if (document.Title == string.Emtpy)
     { return false; }

     if (document.LastModified < DateTime.UtcNow.AddDays(-30))
     { return false; }

     if (!document.ApprovedByLitigation)
     { return false; }

     if (!document.ApprovedByManagement)
     { return false; }

     return true; }
```

## Describing the Chain of Responsibility Pattern

```
public bool Validate() {
     if (document.Title == string.Emtpy)
     { return false; }

     if (document.LastModified < DateTime.UtcNow.AddDays(-30))
     { return false; }

     if (!document.ApprovedByLitigation)
     { return false; }

     if (!document.ApprovedByManagement)
     { return false; }

     return true; }
```

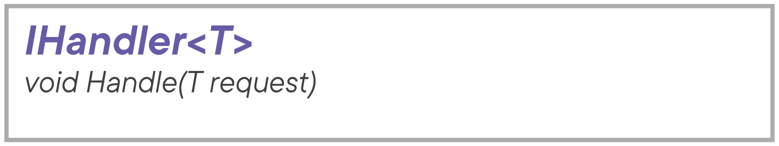
## Describing the Chain of Responsibility Pattern

```
public bool Validate() {
     if (document.Title == string.Emtpy)
     { return false; }

     if (document.LastModified < DateTime.UtcNow.AddDays(-30))
     { return false; }

     if (!document.ApprovedByLitigation)
     { return false; }

     if (!document.ApprovedByManagement)
     { return false; }

     return true; }
```

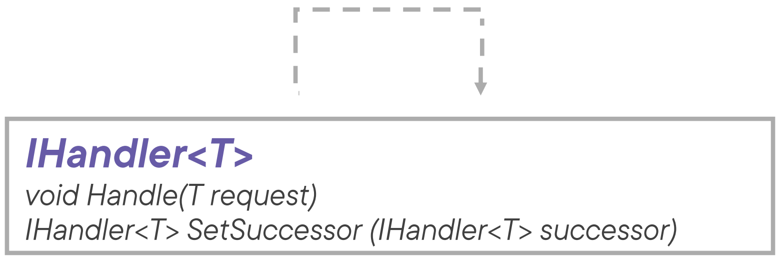
## Describing the Chain of Responsibility Pattern

**Too many conditional statements**
**Validation method becomes bloated**
**Cannot easily reuse this code**

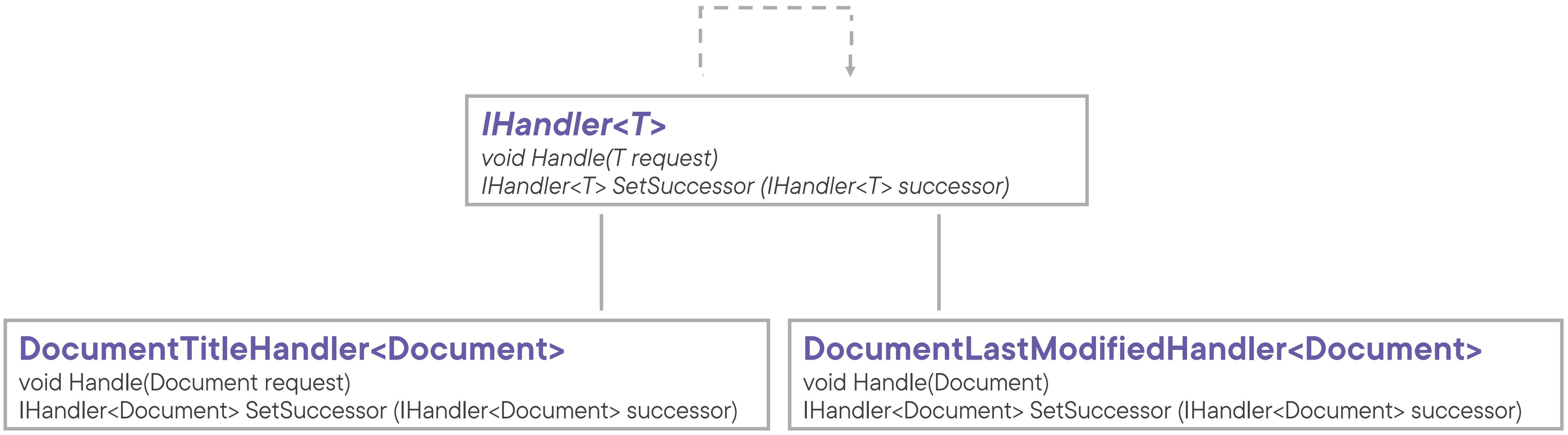
# Describing the Chain of Responsibility Pattern



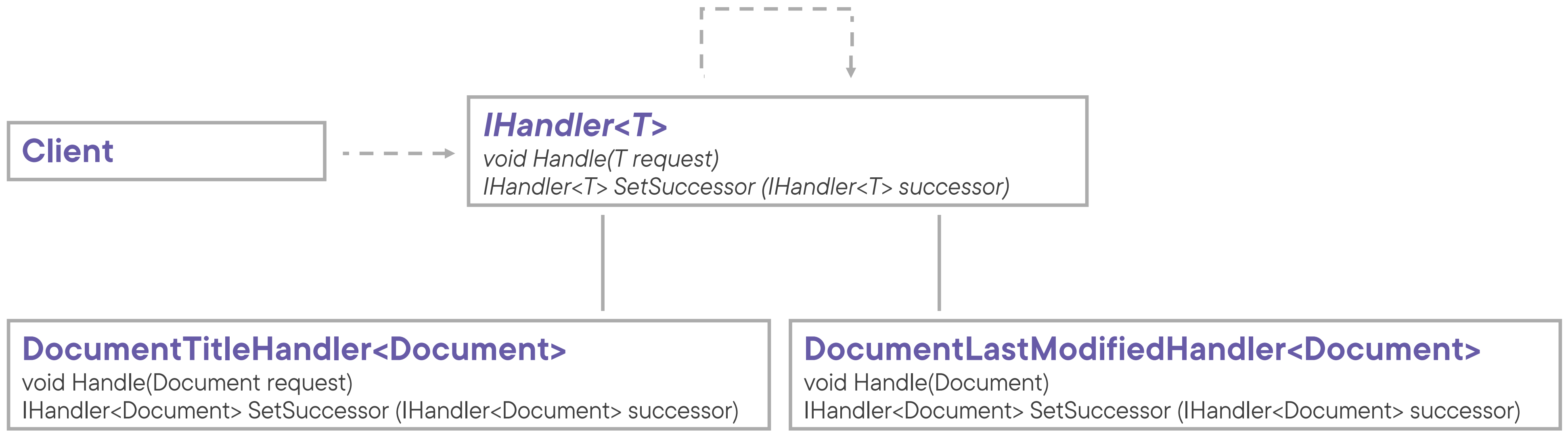
## Describing the Chain of Responsibility Pattern



## Describing the Chain of Responsibility Pattern


*IHandler<T>*
*void Handle(T request)*
*IHandler<T> SetSuccessor (IHandler<T> successor)*
DocumentTitleHandler<Document>
void Handle(Document request)
IHandler<Document> SetSuccessor (IHandler<Document> successor)
DocumentLastModifiedHandler<Document>
void Handle(Document)
IHandler<Document> SetSuccessor (IHandler<Document> successor)

# Describing the Chain of Responsibility Pattern



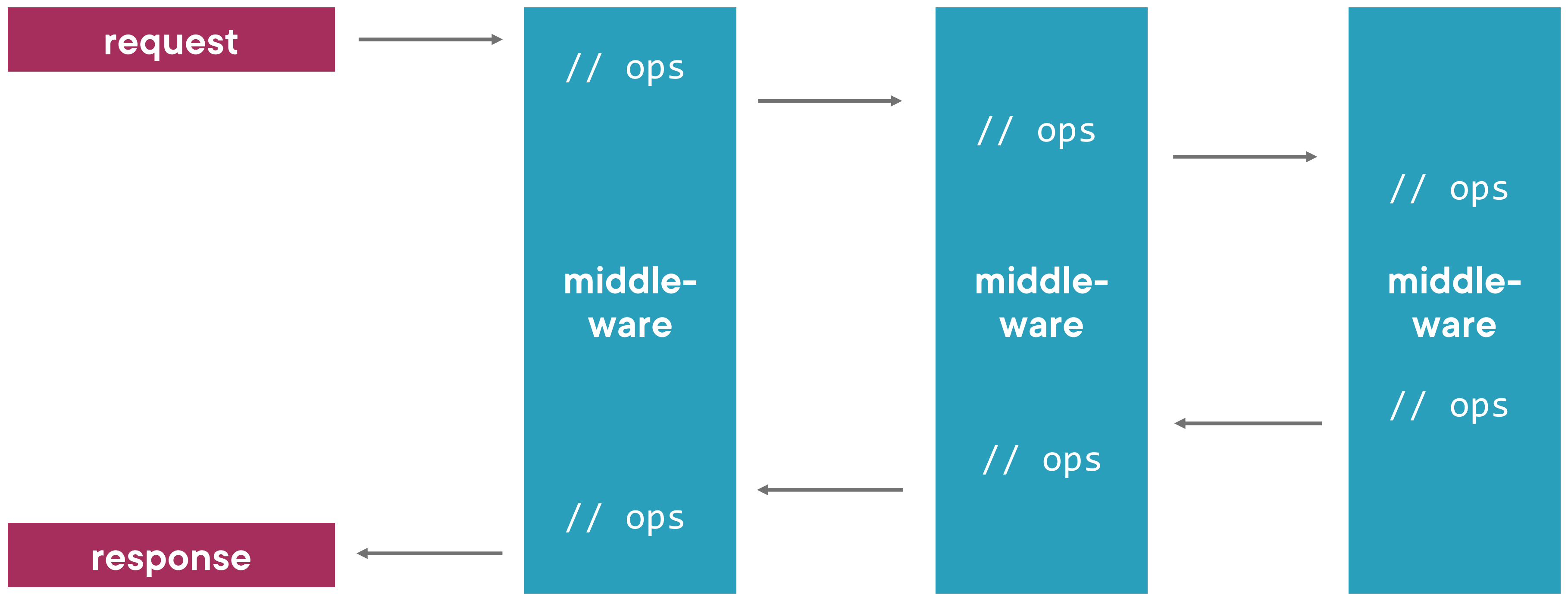
Client
IHandler<T>
void Handle(T request)
IHandler<T> SetSuccessor (IHandler<T> successor)
DocumentTitleHandler<Document>
void Handle(Document request)
IHandler<Document> SetSuccessor (IHandler<Document> successor)
DocumentLastModifiedHandler<Document>
void Handle(Document)
IHandler<Document> SetSuccessor (IHandler<Document> successor)

## Describing the Chain of Responsibility Pattern

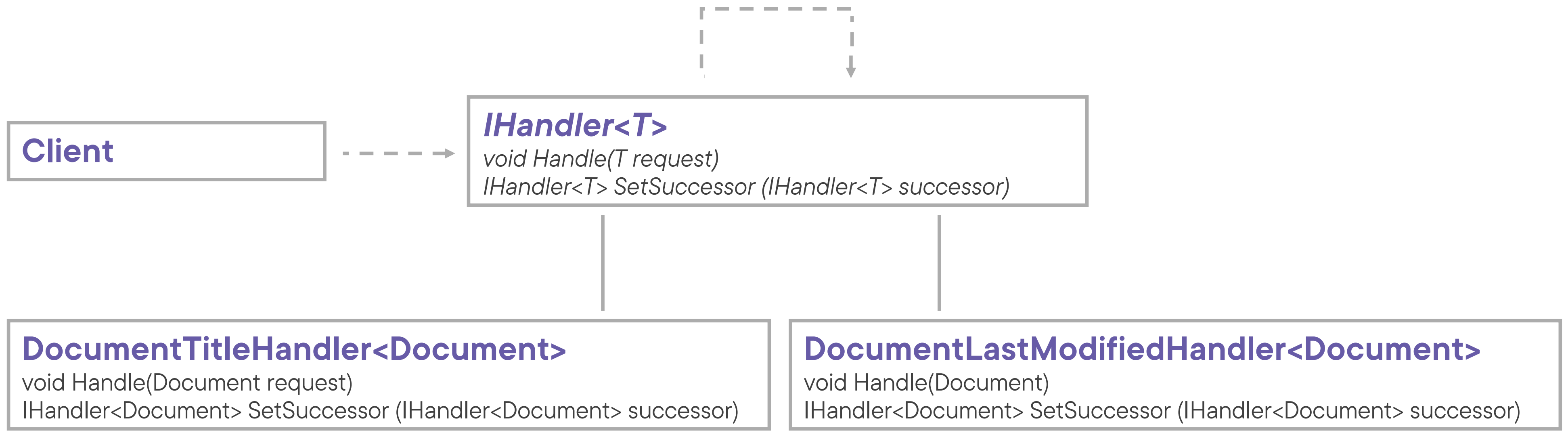
**The original GoF template is more strict**

- Each handler only checks whether it can handle the request or not
  - If it can't, the request is passed on
  - If it can, the request is handled and no longer passed on

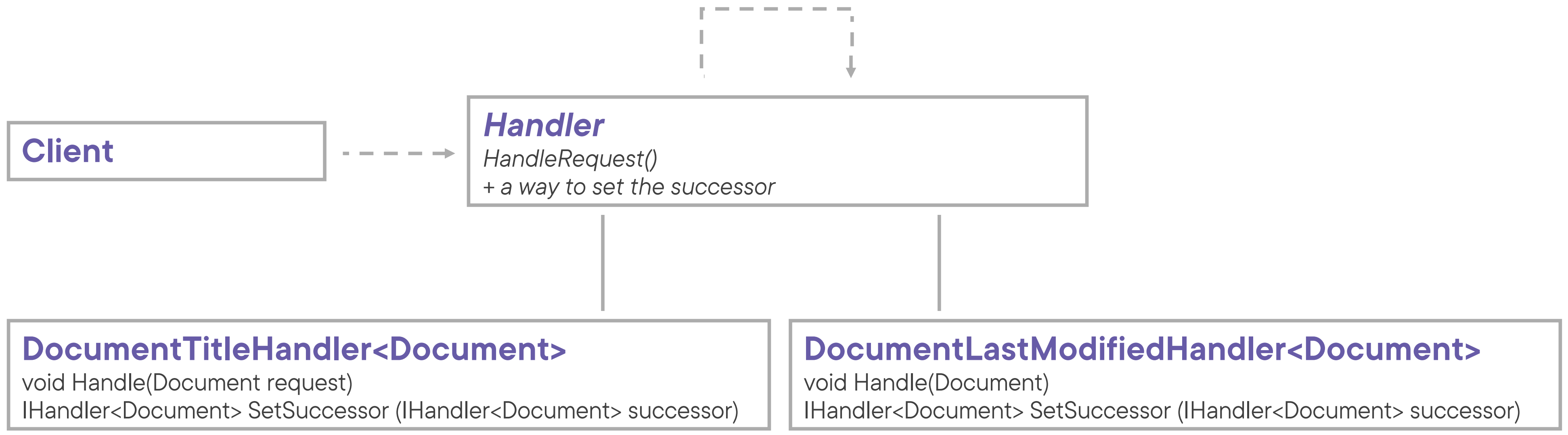
# Describing the Chain of Responsibility Pattern



# Structure of the Chain of Responsibility Pattern

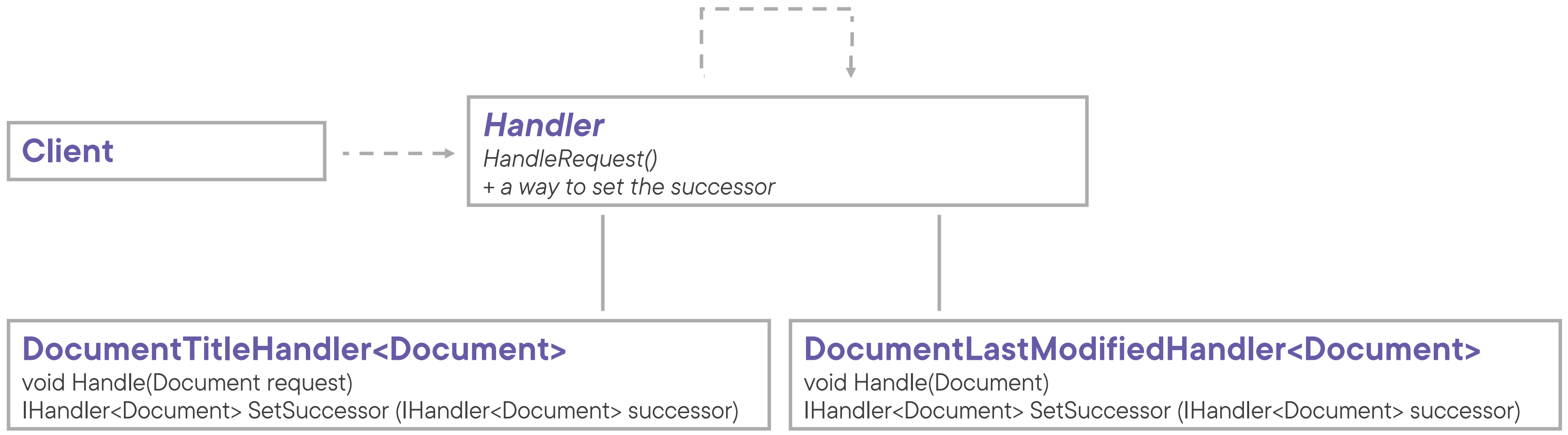

Client
IHandler<T>
void Handle(T request)
IHandler<T> SetSuccessor (IHandler<T> successor)
DocumentTitleHandler<Document>
void Handle(Document request)
IHandler<Document> SetSuccessor (IHandler<Document> successor)
DocumentLastModifiedHandler<Document>
void Handle(Document)
IHandler<Document> SetSuccessor (IHandler<Document> successor)

## Structure of the Chain of Responsibility Pattern

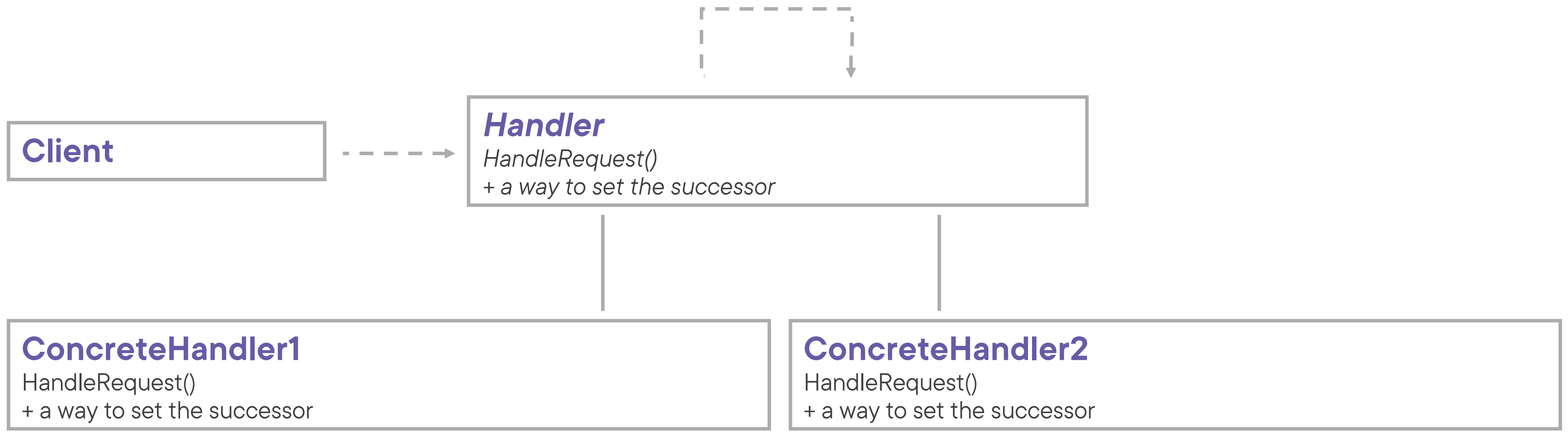

Client
Handler
HandleRequest()
+ a way to set the successor
DocumentTitleHandler<Document>
void Handle(Document request)
IHandler<Document> SetSuccessor (IHandler<Document> successor)
DocumentLastModifiedHandler<Document>
void Handle(Document)
IHandler<Document> SetSuccessor (IHandler<Document> successor)

**Handler** defines an interface for handling requests, and optionally inmplements the successor link

# Structure of the Chain of Responsibility Pattern

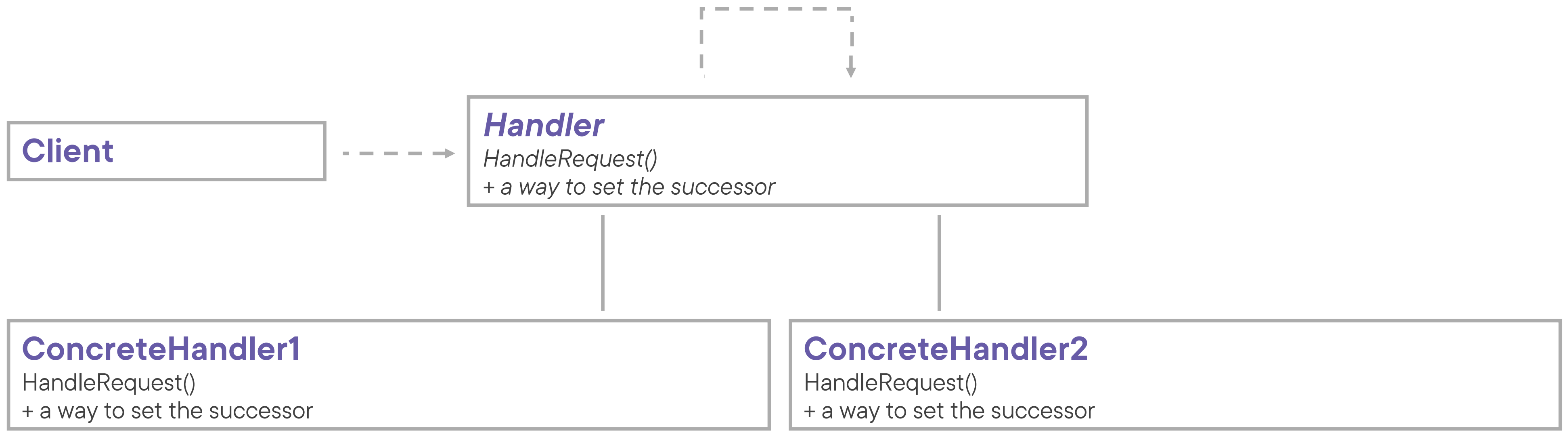

Client
Handler
HandleRequest()
+ a way to set the successor
DocumentTitleHandler<Document>
void Handle(Document request)
IHandler<Document> SetSuccessor (IHandler<Document> successor)
DocumentLastModifiedHandler<Document>
void Handle(Document)
IHandler<Document> SetSuccessor (IHandler<Document> successor)

# Structure of the Chain of Responsibility Pattern



**ConcreteHandler** handles requests it’s responsible for.  It can access the successor and potentially pass the request on.

# Structure of the Chain of Responsibility Pattern



**Client** initiates the request to a **ConcreteHandler** object on the chain

Demo

Implementing the chain of responsibility pattern

## Use Cases for the Chain of Responsibility Pattern

**When more than one object may handle a request and the handler isn't known beforehand**

**When you want to issue a request to one of several objects (handlers) without specifying the receiver explicitly**

**When the set of objects that handle a request should be specified dynamically**

## Pattern Consequences

It enables reduced coupling & works towards a single responsibility per class

It adds flexibility in regards to assigning responsibilities to objects

It does not guarantee receipt of the request

## Related Patterns

**Composite**
The parent of a leaf can act as the successor

**Command**
Chain of responsibility handlers can be implemented as commands

## Patterns that Connect Senders and Receivers

**Chain of Responsibility**
Passes a request along a chain of receivers

**Command**
Connects senders with receivers unidirectionally

**Mediator**
Eliminates direct connections altogether

**Observer**
Allows receivers of requests to (un)subscribe at runtime

## Summary

**Intent of the chain of responsibility pattern:**

- To avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request

## Summary

**Implementation:**

- Provide an easy way to set the next handler
- Return the successor when setting the next handler to enable a fluent interface
- Use generics to make the handler more generic

Up Next:

Behavioral Pattern: Observer