Chain of responsibility 🔗

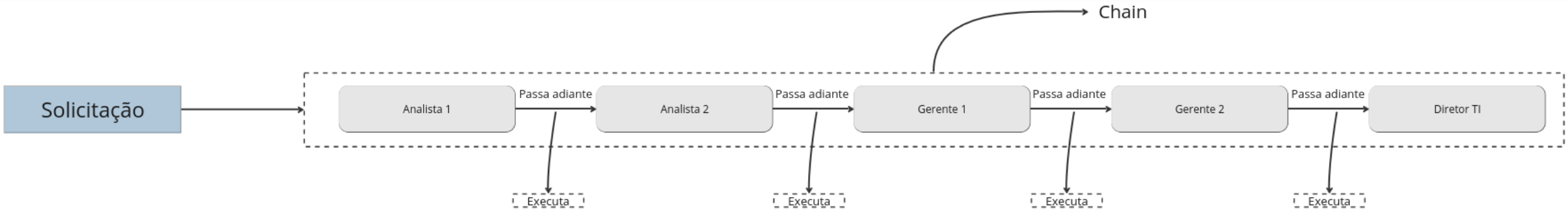
Análise e Desenvolvimento de Sistemas - 6º Período

Padrões comportamentais

- Padrão comportamental
- Os padrões comportamentais se preocupam com a interação e responsabilidade entre objetos e classes
- Aborda como os objetos se comunicam e colaboram para realizar tarefas e determinados comportamentos

Cenário

- Imagine um cenário de abertura de chamados em uma grande empresa
- Na maioria dos casos, esses chamados devem ser passados por diferentes **aprovadores**
- Esses aprovadores geralmente são gerentes ou diretores que decidem o andamento das solicitações
- Esses chamados formam uma **cadeia de aprovadores**


Chain
Solicitação
Analista 1
Passa adiante
Analista 2
Passa adiante
Gerente 1
Passa adiante
Gerente 2
Passa adiante
Diretor TI
Executa
Executa
Executa
Executa

Cenários

- Cada aprovador precisa **decidir se ele é responsável ou não pela aprovação e passar adiante**
- Uma solicitação pode ter um **único aprovador ou vários**, dependendo do tipo de chamado
- Implementar esse cenário pode ser desafiador em orientação a objetos se não possuirmos um padrão para seguir

Chain of responsibility

- Cria uma cadeia de objetos, onde cada objeto pode ou não processar uma solicitação
- Cada objeto dessa **cadeia (ou corrente) de processamento** é chamado de **handler**
- A decisão de processar ou não é implementada internamente no handler

Chain of responsibility

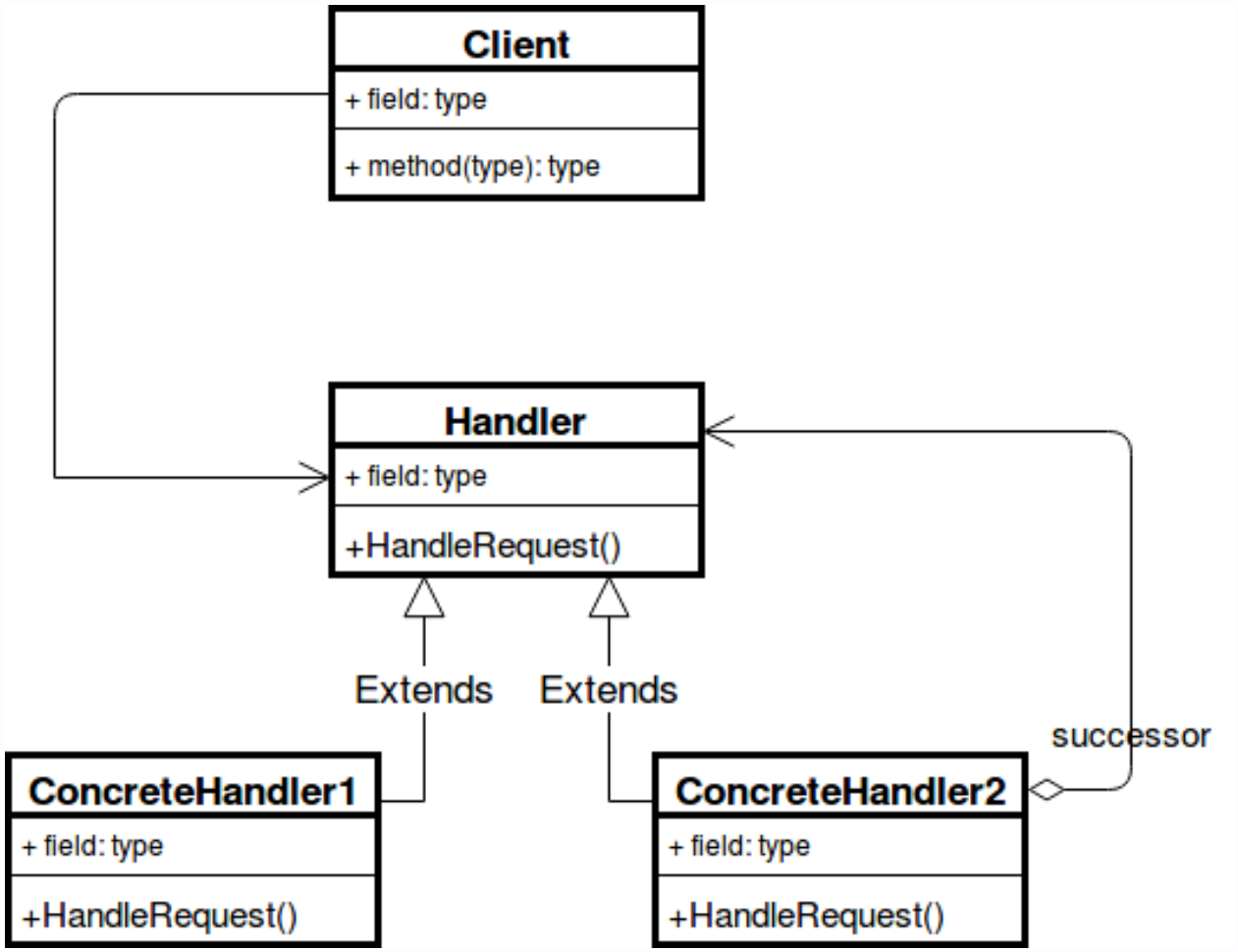
- Cada handler possui uma **referência ao próximo handler** na cadeia de processamento
- Portanto, além de processar, **ele passa adiante a solicitação na corrente**
- Mas o handler pode simplesmente **decidir não passar a solicitação e finalizar a execução da corrente**
- Isso nos permite criar um **comportamento de checagem de informações**

Chain of responsibility

- Portanto, podemos ter dois tipos de comportamentos no processamento da cadeia:
  1. Onde apenas um handler processa a solicitação;
  2. Mais de um handler processa (Processa mas passa adiante)

Chain of responsibility

- Todos os handlers devem **implementar a mesma interface**.
- Cada handler deve se preocupar se o próximo na corrente implementa um método para executar a solicitação. Se todos implementam a mesma interface, conseguimos garantir isso


Client
+ field: type
+ method(type): type
Handler
+ field: type
+HandleRequest()
Extends
Extends
successor
ConcreteHandler1
+ field: type
+HandleRequest()
ConcreteHandler2
+ field: type
+HandleRequest()

Chain of responsibility

- **Utilizamos quando**: 1) Processamos diferentes tipos de pedidos de diferentes maneiras. 2) Queremos executar handlers em ordem específica
- **Vantagens**: Controlamos a ordem de execução; Princípio de responsabilidade única (separa o código que invoca do código que usa); Princípio open/close (conseguimos introduzir novos handlers sem quebrar o código)