Arquitetura de Software

Padrões de Projetos – Padrões Comportamentais

Nairon Neri Silva

Sumário

Padrões de Projeto Comportamentais

1. ***Chain of Responsibility***
2. ***Command***
3. *Interpreter*
4. *Iterator*
5. *Mediator*
6. *Memento*
7. *Observer*
8. *State*
9. *Strategy*
10. *Template Method*
11. *Visitor*

Introdução

- Se preocupam com algoritmos e a atribuição de responsabilidade entre objetos

- Descreve
  - o papel de classes e objetos
  - como se dará a comunicação entre os envolvidos

Chain of Responsibility

Cadeia de Responsabilidades

Intenção

- Evitar o acoplamento do remetente de uma solicitação ao seu receptor, ao dar a mais de um objeto a oportunidade de tratar a solicitação

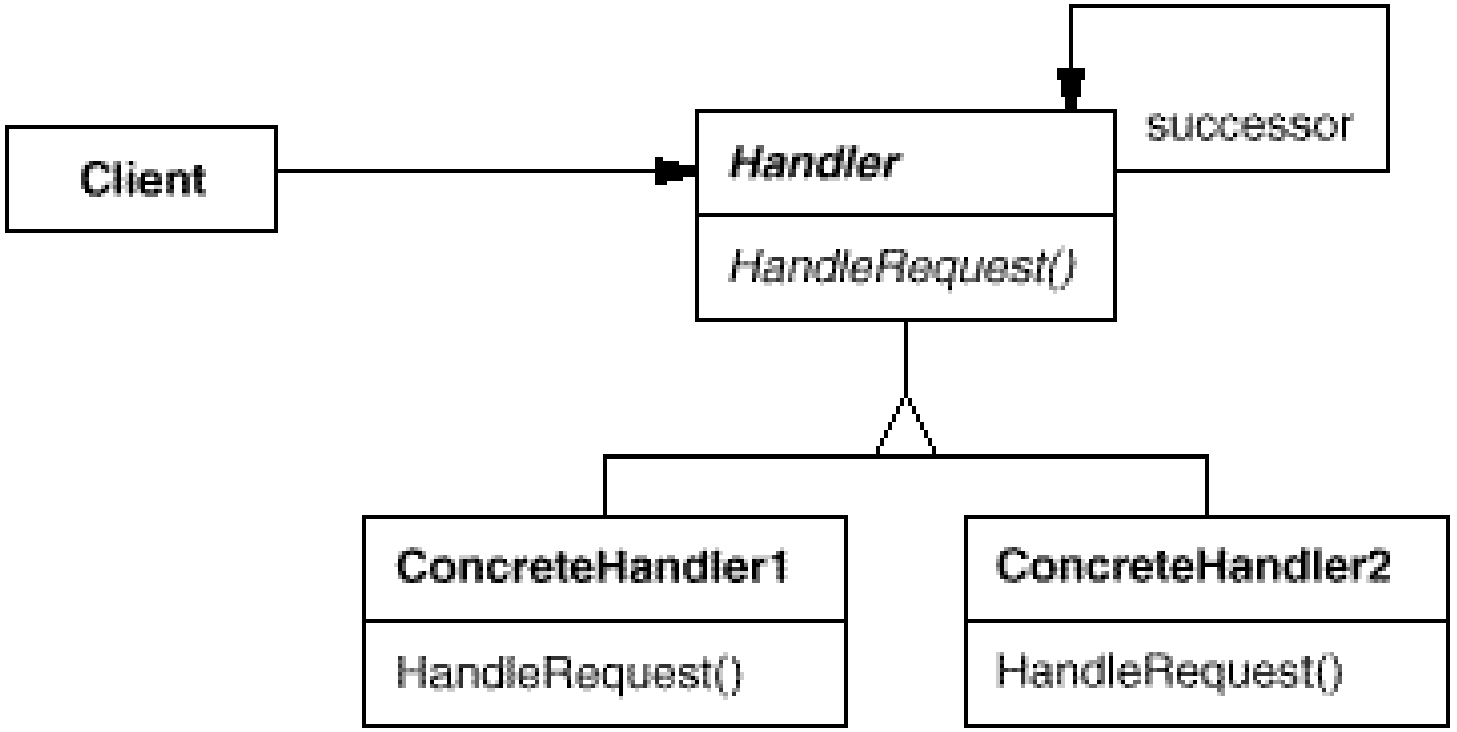
- Encadear os objetos receptores, passando a solicitação ao longo da cadeia até que um objeto a trate

Motivação

- Considere um recurso de ajuda (*help*) sensível ao contexto para uma interface gráfica de usuário:
  - A ajuda está acessível em qualquer ponto da interface
  - A ajuda depende do ponto (contexto) onde foi acionada
  - O problema é que o objeto que fornece a ajuda não é conhecido explicitamente pelo solicitante
  - O que necessitamos é de uma maneira de desacoplar o botão que inicia a solicitação de ajuda dos objetos que podem fornecer as informações de ajuda

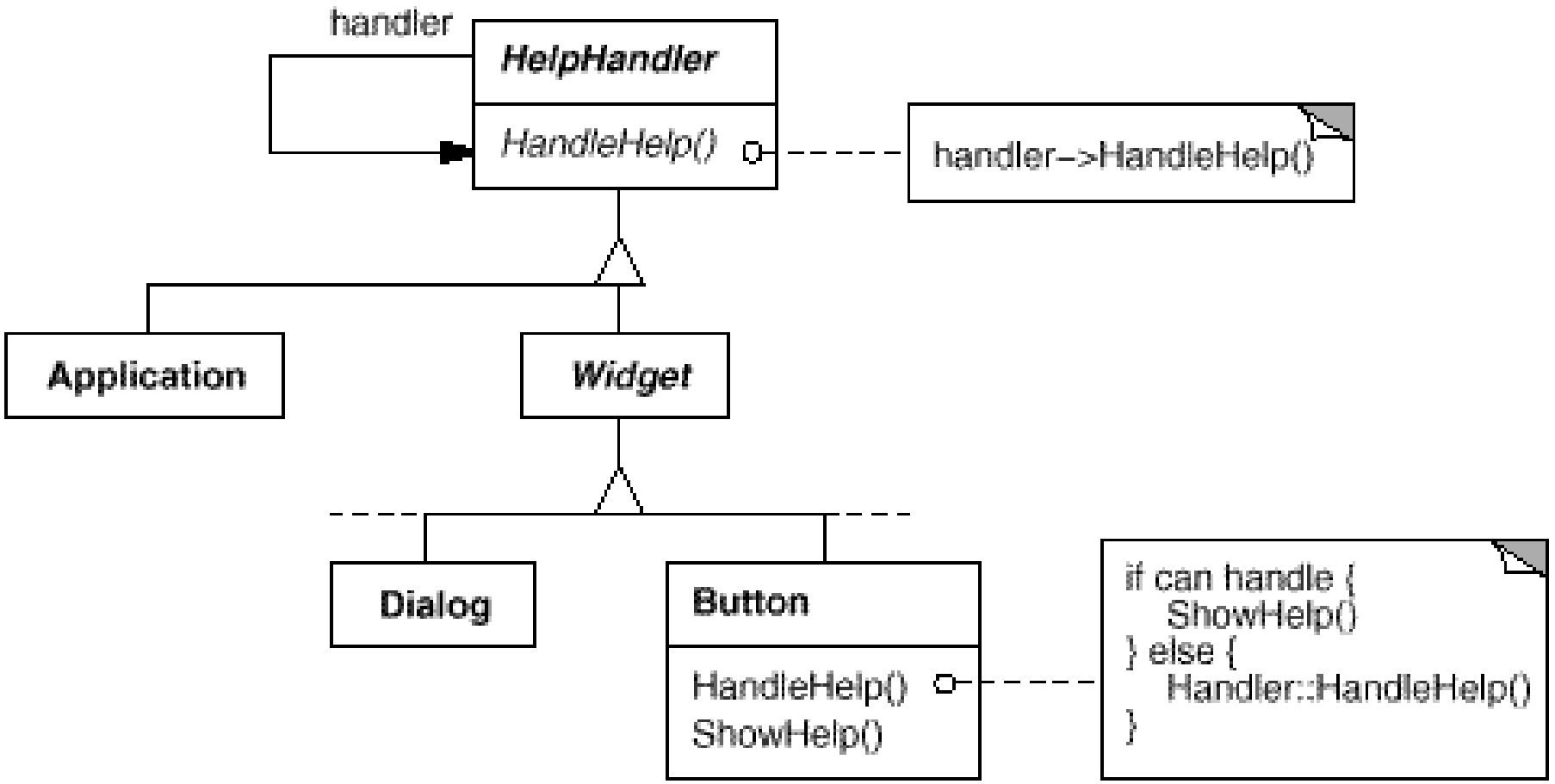
Aplicabilidade

- Utilizar quando:
  1. Mais de um objeto pode tratar uma requisição e o objeto que irá tratar a mesma não é conhecido a priori
  2. Desejamos fazer uma solicitação para um dentre vários objetos, sem especificar explicitamente o receptor
  3. O conjunto de objetos que pode tratar uma solicitação deveria ser especificado dinamicamente


Client
Handler
HandleRequest()
successor
ConcreteHandler1
HandleRequest()
ConcreteHandler2
HandleRequest()

Estrutura

- Uma típica estrutura de objetos se parece seria:


handler
HelpHandler
HandleHelp()
handler->HandleHelp()
Application
Widget
Dialog
Button
HandleHelp()
ShowHelp()
if can handle {
ShowHelp()
} else {
Handler::HandleHelp()
}

Exemplo / Participantes

1. ***Handler (HelpHandler)***
   - Define uma interface para tratar solicitações
   - (opcional) implementa um link para seu sucessor
2. ***ConcreteHandler (PrintButton, PrintDialog)***
   - Trata solicitações de sua responsabilidade
   - Pode acessar o seu sucessor
   - Se pode tratar, assim o faz, se não passa a solicitação para o seu sucessor
3. ***Client***
   - Inicia uma solicitação para um objeto da cadeia

Consequências

1. Acoplamento reduzido

2. Flexibilidade adicional na atribuição de responsabilidades a objetos

3. A recepção não é garantida

Exemplo Prático

Exemplo presente no site: https://refactoring.guru/pt-br/design-patterns/chain-of-responsibility/java/example

1. Neste exemplo vamos mostrar como uma solicitação que contém dados do usuário passa por uma cadeia sequencial de manipuladores que executam várias tarefas, como autenticação, autorização, e validação.

Saiba mais...

- https://www.youtube.com/watch?v=AdzLq9FVTXs
- https://refactoring.guru/pt-br/design-patterns/chain-of-responsibility
- https://www.baeldung.com/chain-of-responsibility-pattern

Acesse os endereços e veja mais detalhes sobre o padrão Chain of Responsability.

Command

Também conhecido como ***Action*** ou ***Transaction***

Intenção

- Encapsular uma solicitação como um objeto, desta forma permitindo parametrizar clientes com diferentes solicitações, enfileirar ou fazer o registro (*log*) de solicitações e suportar operações que podem ser desfeitas

Motivação

- Algumas vezes é necessário emitir solicitações para objetos nada sabendo sobre a operação que está sendo solicitada ou sobre o receptor da mesma

- Exemplo: no contexto de *frameworks* para desenvolver interfaces gráficas com o usuário
  - Incluem objetos como botões de menu que executam uma solicitação em resposta a uma entrada do usuário
  - Mas, só as classes de domínio sabem o que precisa ser feito em resposta a uma solicitação do usuário

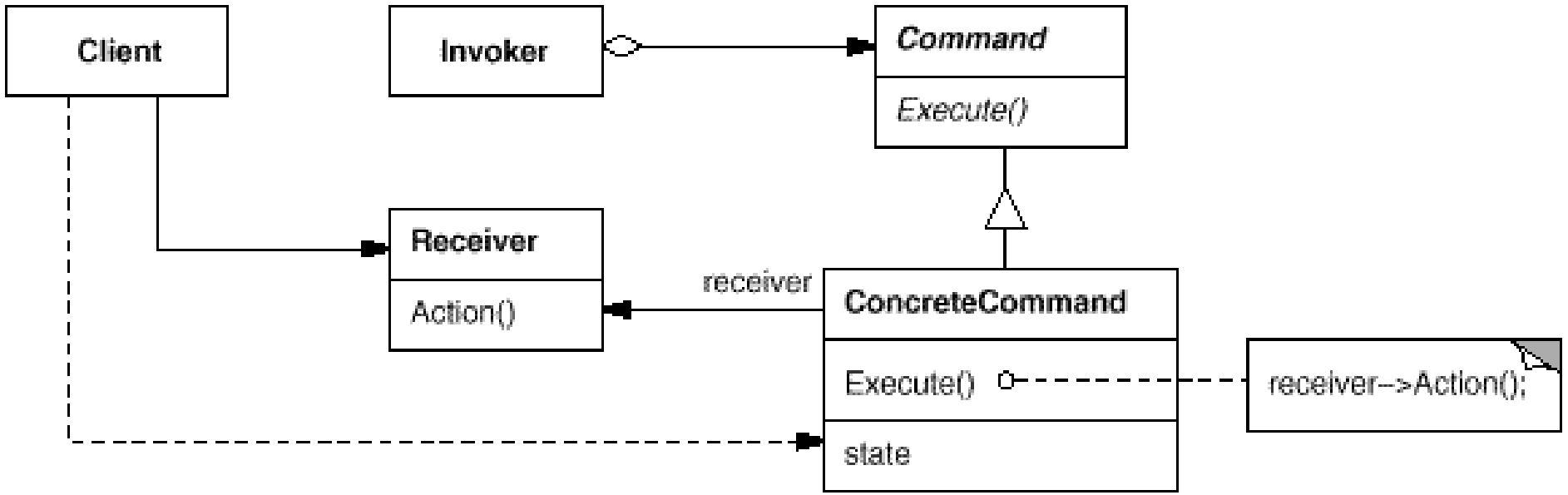
Motivação

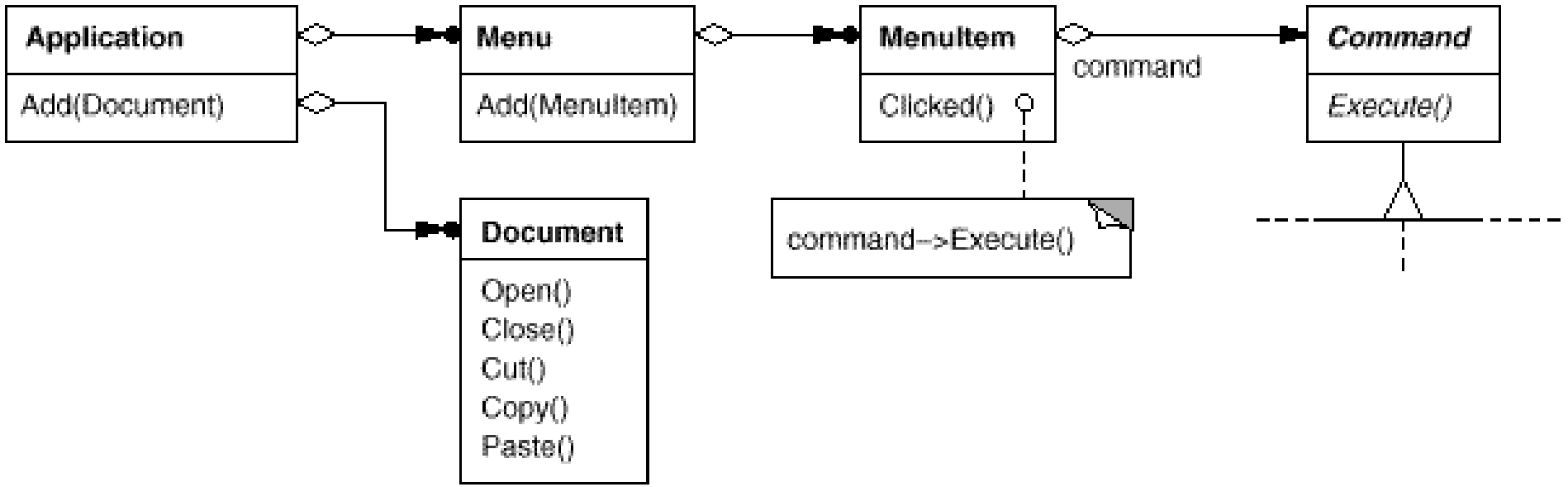
- Continuando...
  - O padrão *Command* permite que objetos do *framework* possam fazer solicitações de objetos não especificados, transformando a própria solicitação em um objeto
  - A chave desse padrão é a classe abstrata *Command*, a qual define uma interface para execução de operações
  - Na sua forma mais simples, essa interface inclui uma operação abstrata execute()

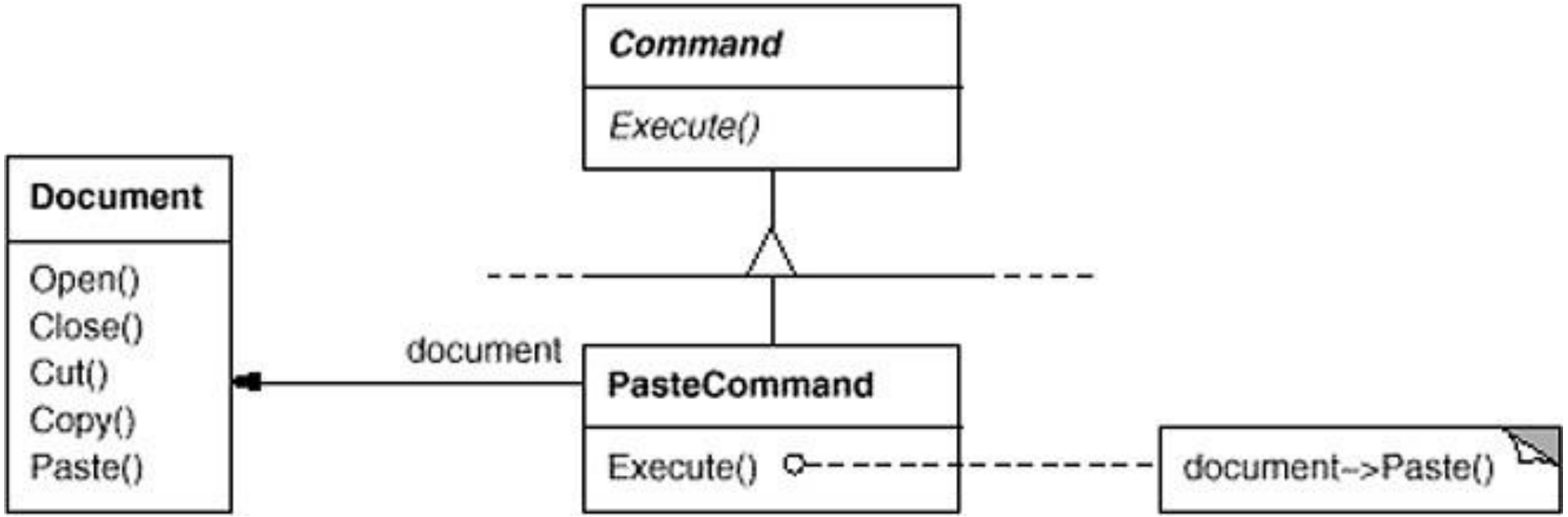
Aplicabilidade

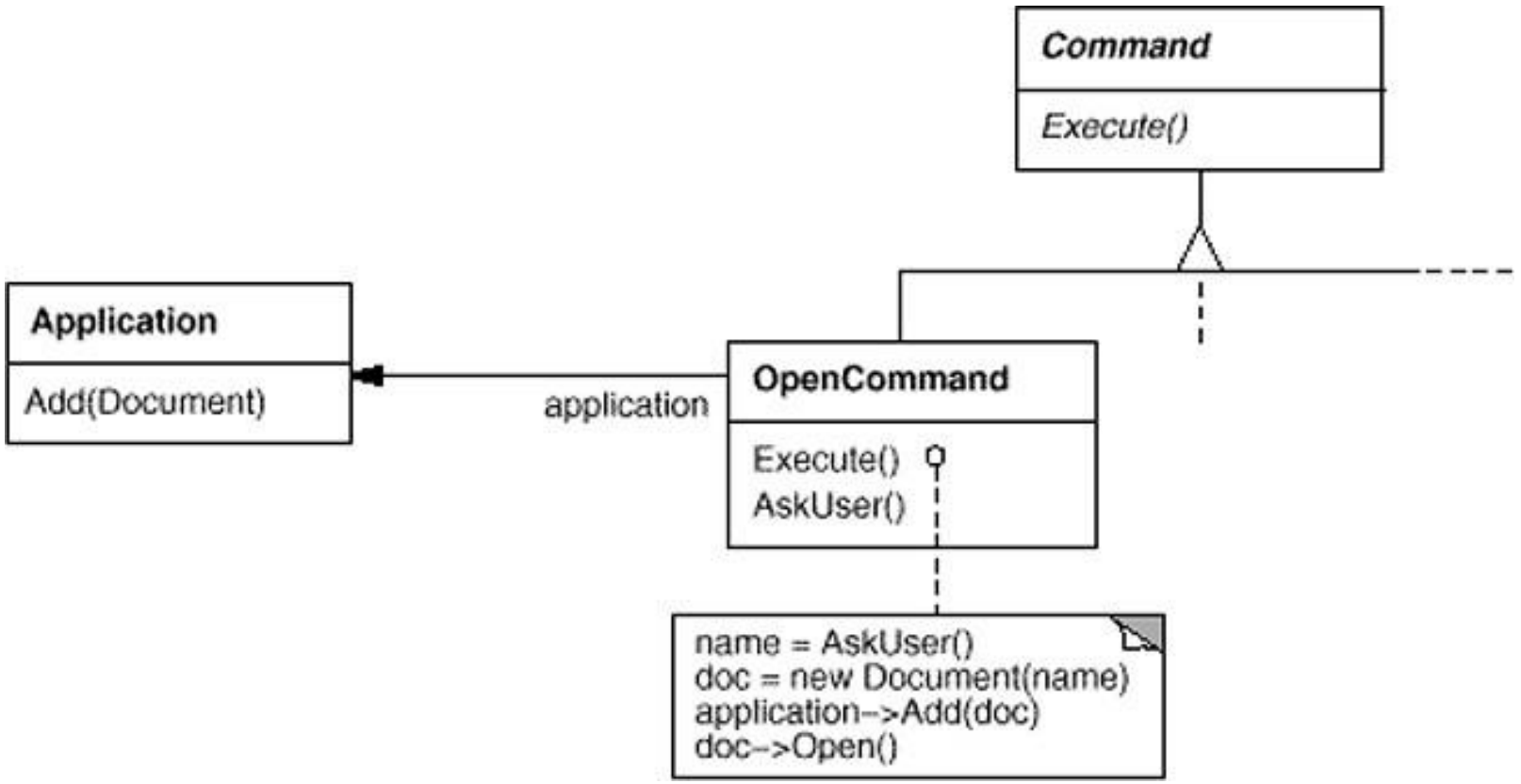
Utilizar o padrão Command quando necessitamos:

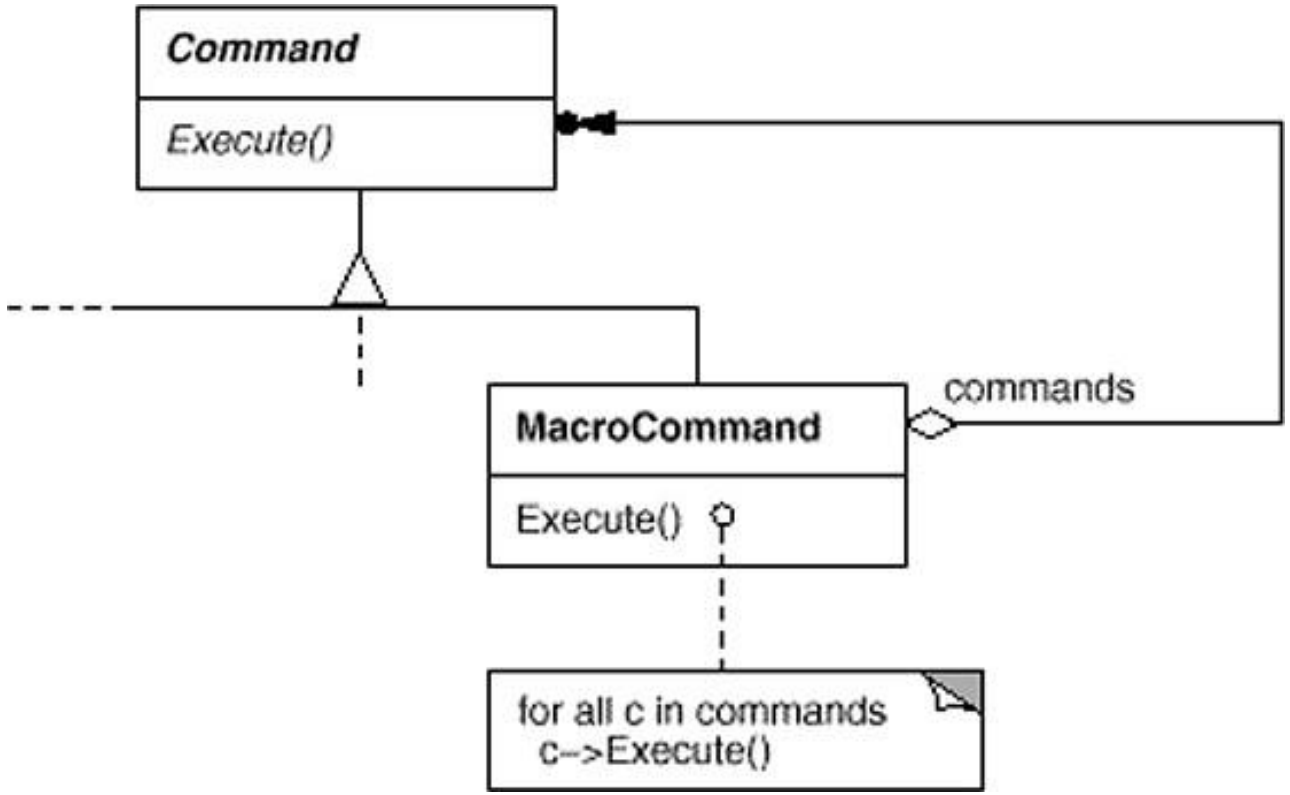
1. parametrizar objetos por uma ação a ser executada
2. especificar, enfieirar e executar solicitações em tempos diferentes
3. suportar ”desfazer” operações
   - O objeto Command podem armazenar estado, tornando possível a reversão dos afeitos da sua ação
   - Operação unexecute()
   - Os comandos executados podem ser armazenados em uma lista histórica – tornando possível desfazer uma série de comandos em sequência
4. suportar o registro (*logging*) de mudanças de maneira que possam ser replicadas no caso de uma queda no sistema
5. estruturar um sistema em torno de operações de alto nível construídas sobre operações primitivas
   - Comum em sistemas que suportam transações


Client
Invoker
Command
Execute()
Receiver
Action()
receiver
ConcreteCommand
Execute()
state
receiver->Action();


Application
Add(Document)
Menu
Add(MenuItem)
MenuItem
Clicked()
command
Command
Execute()
Document
Open()
Close()
Cut()
Copy()
Paste()
command->Execute()


Command
Execute()
Document
Open()
Close()
Cut()
Copy()
Paste()
document
PasteCommand
Execute()
document->Paste()


Command
Execute()
Application
Add(Document)
application
OpenCommand
Execute()
AskUser()
name = AskUser()
doc = new Document(name)
application->Add(doc)
doc->Open()


Command
Execute()
commands
MacroCommand
Execute()
for all c in commands
c->Execute()

Exemplo/Participantes

1. ***Command***
   - Define uma interface para a execução de uma operação
2. ***ConcreteCommand (PasteCommand, OpenCommand)***
   - Implementa a operação execute() correspondente
3. ***Client (Application)***
   - Criar um objeto ConcreteCommand e envia a seu receptor
4. ***Invoker (MenuItem)***
   - Solicita ao Command a execução da solicitação
5. ***Receiver (Document, Application)***
   - Sabe como executar a operação associada a uma solicitação

Consequências

1. Desacopla o objeto que invoca a operação daquele que sabe como executá-la
2. Commands podem ser manipulados e estendidos como qualquer outro objeto
3. Você pode montar comandos para formar um comando composto (MacroCommand)
4. É fácil acrescentar novos comandos, pois não é necessário modificar as classes existentes

Exemplo Prático

Exemplo presente no site: https://refactoring.guru/pt-br/design-patterns/command/java/example

1. Neste exemplo vamos criar um editor de texto que cria novos objetos comando sempre que um usuário interage com ele. Após executar suas ações, um comando é enviado para a pilha do histórico.

2. Agora, para executar a operação desfazer, o aplicativo obtém o último comando executado do histórico e executa uma ação inversa ou restaura o estado passado do editor, salvo por esse comando.

Saiba mais...

- https://www.youtube.com/watch?v=WwDnYXr7jqk
- https://refactoring.guru/pt-br/design-patterns/command
- https://www.baeldung.com/java-command-pattern

Acesse os endereços e veja mais detalhes sobre o padrão Command.