

INSTITUTO FEDERAL
Rio Grande do Sul

Campus
Farroupilha

Engenharia de Software II

Padrões de Projeto **COMMAND**

Professor

Rogério Xavier

Estudantes

Júnior Ceccato e Matheus Vosgnach

Padrões GoF - Gang of Four

I Padrões Criacionais

Autores do livro “**Design Patterns: Elements of Reusable Object-Oriented Software**”, de 1994

23 padrões catalogados

II Padrões Estruturais

III Padrões Comportamentais

Estão relacionados à forma com que objetos **colaboram** entre si, com a finalidade de oferecer uma determinada funcionalidade ao sistema

Memento

Observer

Chain of Responsibility

Iterator

Strategy

Template Method

Mediator

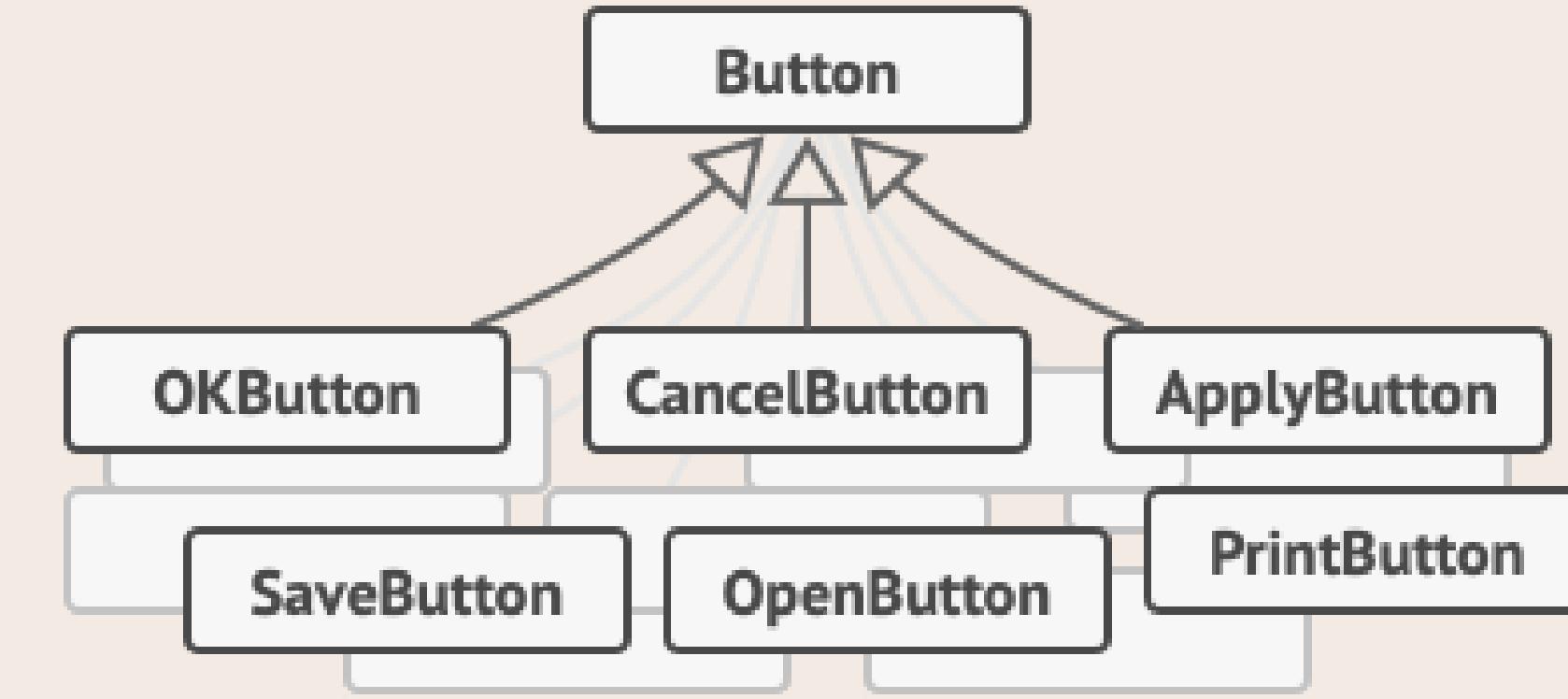
Visitor

State

----- Command -----

Este padrão cria uma classe intermediária que coordena a parametrização e delegação de tarefas à classes de comando concretas que possuem tarefas mais específicas

PRINCIPAIS PROBLEMAS



----- Quais os Problemas? -----

Problema 01: **código GUI dependente do código da Lógica do Negócio, que é altamente volátil**

Problema 02: possibilidade de remetentes distintos utilizarem o mesmo código de comando (duplicado)

PRINCIPAIS PROBLEMAS



SaveButton

Código

SaveMenuItem

Código

SaveShortcut

Código

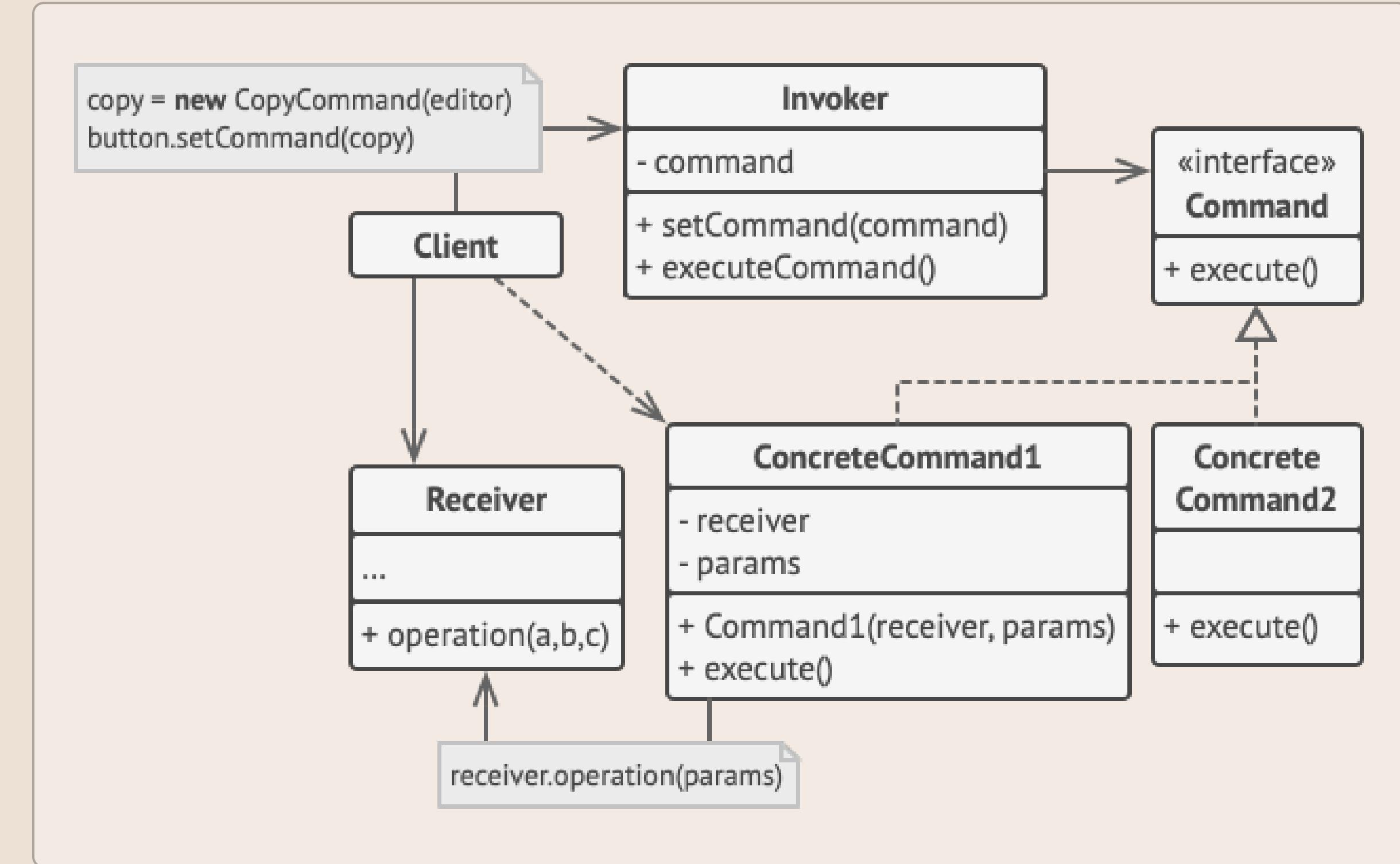
----- Quais os Problemas? -----

Problema 01: código GUI dependente do código da Lógica do Negócio, que é altamente volátil

Problema 02: **possibilidade de alguns remetentes utilizarem o mesmo código de comando (duplicado)**

MODELO DE SOLUÇÃO

- **Client**: cria 'comandos concretos', determina os parâmetros do pedido e o relaciona com um destinatário (receiver)
- **Invoker** (remetente): armazena uma referência para um objeto comando e chama um método que o **executa** (indiretamente)



- **ConcreteCommands**: implementam a interface command, guardam uma referência a um objeto destinatário, bem como os parâmetros que este requer para execução - a nível da lógica do negócio

- **Receiver** (destinatário): objetos referenciados por um Comando Concreto, recebem os parâmetros necessários para a execução de uma operação através do mesmo.

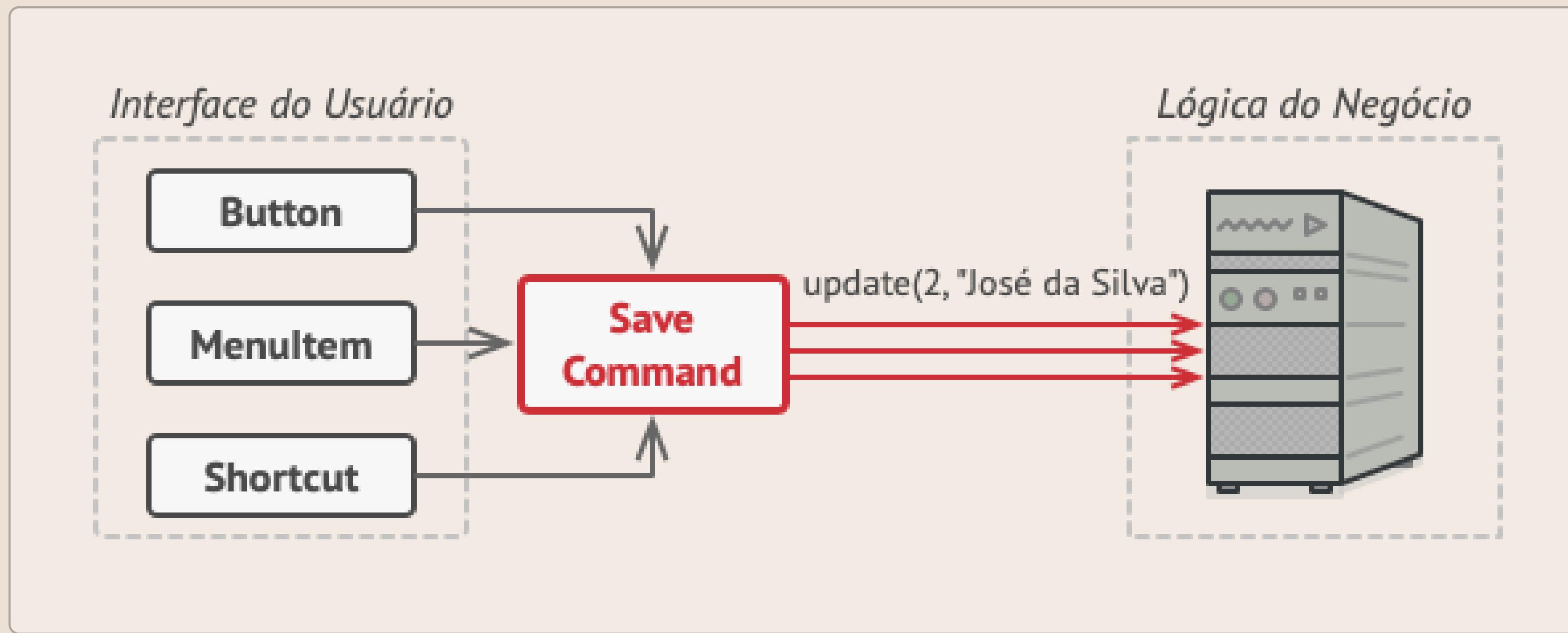
BENEFÍCIOS E APLICABILIDADE



- Auxilia na **PARAMETRIZAÇÃO** de pedidos, criando objetos independentes e diminuindo o acoplamento e dependência entre as classes
- Em conjunto com o padrão *Memento*, que possibilita a cópia de atributos privados dos objetos de forma segura, pode fornecer uma alternativa bastante interessante para a implementação de operações **REVERSAS**
- Possibilita, através da **SERIALIZAÇÃO** (conversão para bytes), manipular a ordem de execução dos comandos, programar atrasos e controlá-los remotamente

RELACIONANDO AO PRINCÍPIO DE **LOW COUPLING**

O princípio do **Baixo Acoplamento** preza pela independência e responsabilidade única das classes. Além de prover um algoritmo elegante e de alta manutenibilidade, também contribui para uma alta coesão no código



EXEMPLOS DE UTILIZAÇÃO

➤ Em *Mortal Kombat 11* (NetherRealm Studios, 2019) o encapsulamento de comandos também auxilia na criação de **combos e ataques sequenciais**, acessando os comandos recentes por meio de uma coleção: lista ou pilha.

➤ Essa modularização poderia ter a utilidade de **adaptar controles** à múltiplas plataformas, variando os botões de entrada, inclusive

INPUT COMMANDS		MOVE DATA	
		DAMAGE	BLOCK DAMAGE
Straight Punch	1	2.00	0.300
Gut Slice	← + 1		MOVE TYPE
Low Jab	↓ + 1		High
Specter Strike	2	N/A	PROPERTIES
Rising Cut	← + 2		DESCRIPTION
Rising Spear	↓ + 2		N/A
Hinge Kick	3		
Flip Kick	← + 3		FRAME DATA
Flick Kick	→ + 3		
Side Strike	↓ + 3		
Step Kick	4		START-UP
Scorpion Sting	← + 4	7	ACTIVE
Shin Strike	→ + 4	2	CANCEL
		11	
			HIT ADVANTAGE
		4	BLOCK ADVANTAGE
		-2	

EXEMPLOS DE UTILIZAÇÃO

➤ Em *Dead Cells*, jogo da Motion Twin lançado em 2018 para PC e consoles, uma vasta variedade de armas e habilidades podem ser combinadas através de **comandos básicos** para a execução de ataques, magias e desvios de batalha

O armazenamento das informações essenciais dos controles básicos em comandos torna versátil a parametrização para cada um dos itens e atributos



EXEMPLOS DE UTILIZAÇÃO

- Com um comando para cada um dos *slots* de cura, ataque e habilidades, torna-se possível, além de personalizar controles facilmente, **parametrizar itens e ferramentas** com diferentes animações, taxas de disparo, dano causado, vulnerabilidade, e outra série de variáveis que estão envolvidas nas particularidades de cada arma e atributo.



EXEMPLOS DE UTILIZAÇÃO

- No *Chess.com* é possível acessar estados anteriores do tabuleiro para realizar uma análise detalhada da partida
- O armazenamento de cada comando do jogo em uma coleção permite o acesso às versões anteriores dos objetos. Ações como o “desfazer” podem ser criadas através do padrão **command** também

Report

Key Moments

Analysis

+9.61

?

is a mistake

g7 is best

8. e5 $\mathbb{Q}xe5$ (8... h5) (8... $\mathbb{W}d8$)

6... $\mathbb{Q}b4$ 7. $\mathbb{K}c1$

(7. $\mathbb{W}d2$) (7. $\mathbb{Q}xc4$ $\mathbb{Q}xc3+$)

7... 0-0 8. a3 $\mathbb{Q}a5$ 9. b4 $\mathbb{Q}b6$

(9... cxb3 10. $\mathbb{W}xb3$ h5 (10... $\mathbb{Q}d5$)

(10... $\mathbb{Q}xe4$ 11. f3))

10. e5 $\mathbb{Q}h7$ 11. $\mathbb{Q}f3$ $\mathbb{M}e8$ 12. a4 a6

13. a5 $\mathbb{Q}a7$ 14. $\mathbb{Q}xc4$ g5 15. $\mathbb{Q}e3$ f5

16. d5

(16. exf6 $\mathbb{Q}xf6$ 17. $\mathbb{Q}e5$)

16... f4 17. $\mathbb{Q}d4$ exd5 18. $\mathbb{Q}xd5+$

(18. $\mathbb{Q}xd5$ $\mathbb{Q}xd4$ 19. $\mathbb{Q}f6+$)

18... $\mathbb{Q}h8$ 19. $e6+$ $\mathbb{Q}f6$ 20. $\mathbb{Q}xf6+$

$\mathbb{Q}xf6$ 21. $\mathbb{Q}e4$ $\mathbb{Q}e7$ 22. 0-0

Self

Threats

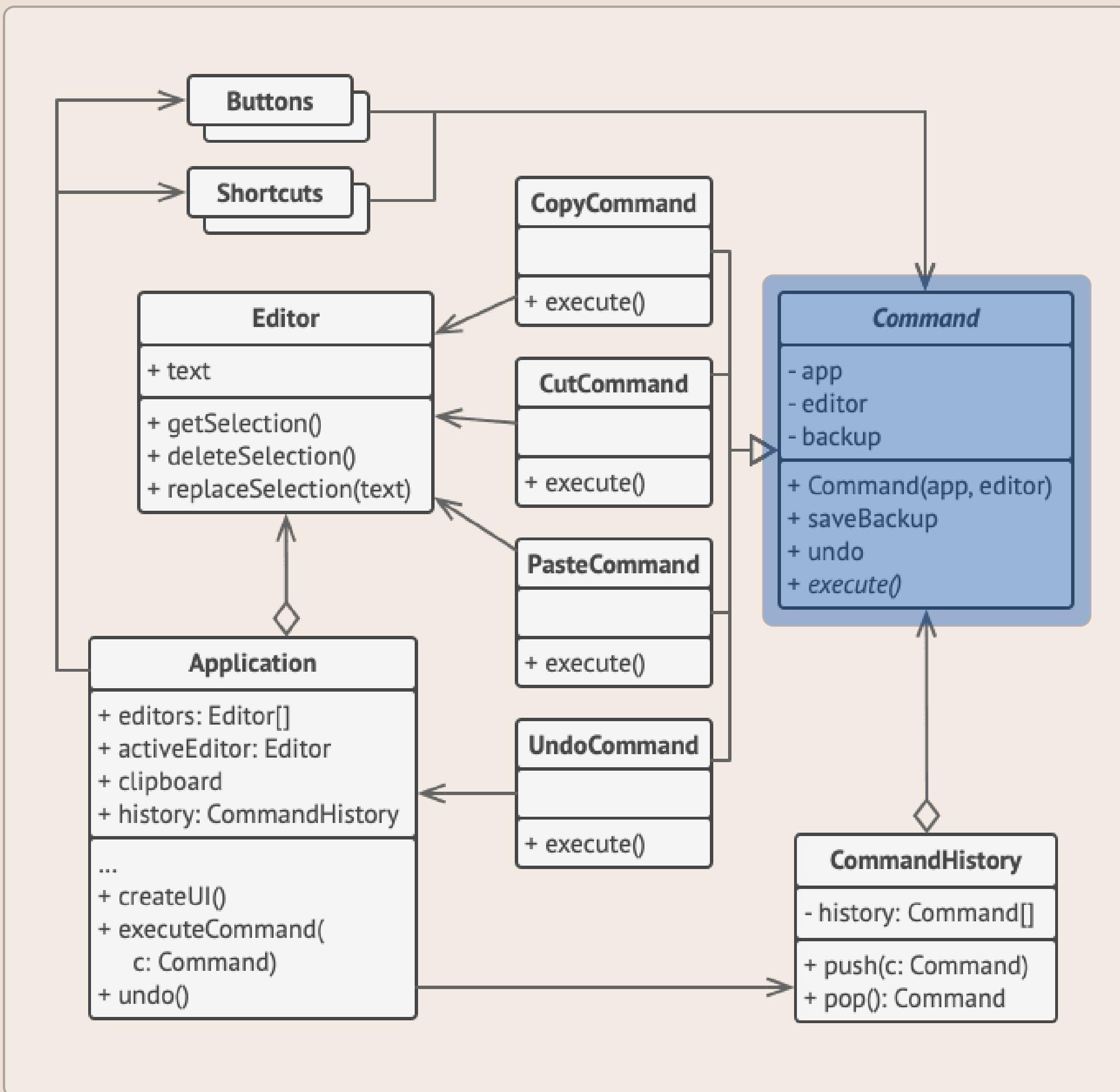
Mistake

Back

Forward

IMPLEMENTAÇÃO PSEUDOCÓDIGO

- Exemplo de Diagrama de Classes que apresenta um **editor de texto**
- A classe **Command** armazena o nome da referida aplicação, do editor utilizado e um *backup* do texto, além do comando abstrato **execute()**
- * Os comandos concretos herdam o comando *execute()* da classe abstrata *Command*, direcionando as ações de **copiar, colar, recortar e desfazer** ao seu respectivo destinatário (*receiver*)



IMPLEMENTAÇÃO - EDITOR DE TEXTO PSEUDOCÓDIGO



```
// The base command class defines the common interface for all
// concrete commands.

abstract class Command is

    protected field app: Application
    protected field editor: Editor
    protected field backup: text

    constructor Command(app: Application, editor: Editor) is
        this.app = app
        this.editor = editor

    // Make a backup of the editor's state.
    method saveBackup() is
        backup = editor.text

    // Restore the editor's state.
    method undo() is
        editor.text = backup

    // The execution method is declared abstract to force all
    // concrete commands to provide their own implementations.
    // The method must return true or false depending on whether
    // the command changes the editor's state.

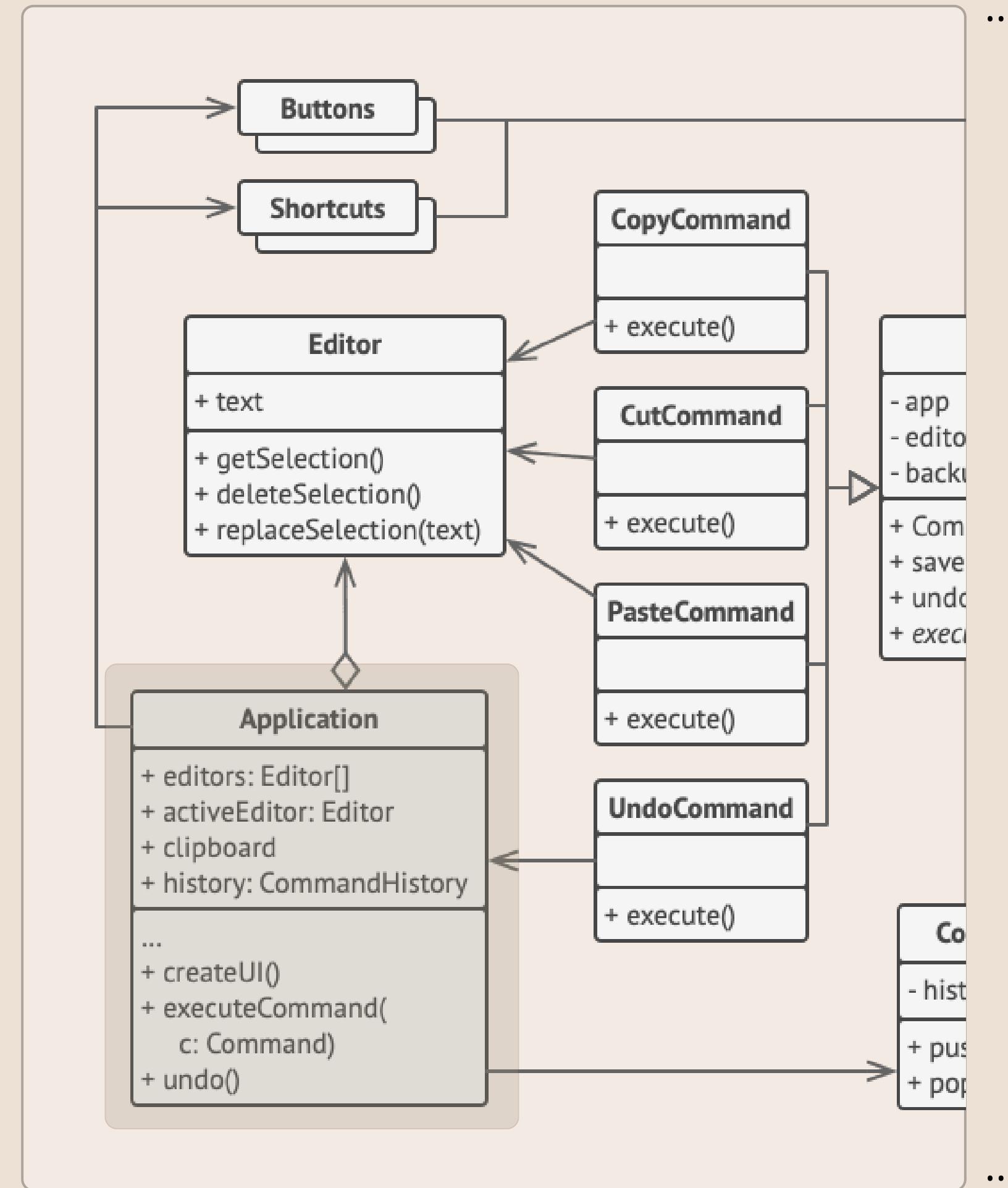
    abstract method execute()
```

IMPLEMENTAÇÃO DO PSEUDOCÓDIGO

Neste caso, o remetente é ***Application***, que cria comandos através do método *createCommand()*, mantém as coleções *Editor[]* e histórico de comandos, determina o editor ativo, uma área de transferência (*clipboard*) e direciona a execução de um ***Command***

```
method createUI() is
    // ...
    copy = function() { executeCommand(
        new CopyCommand(this, activeEditor)) }
    copyButton.setCommand(copy)
    shortcuts.onKeyPress("Ctrl+C", copy)
```

➤ *undo()*: realiza um *pop* da pilha *CommandHistory*



```
class Application is
    field clipboard: string
    field editors: array of Editors
    field activeEditor: Editor
    field history: CommandHistory
```



IMPLEMENTAÇÃO DO PSEUDOCÓDIGO

```
// Execute a command and check whether it has to be added to
// the history.

method executeCommand(command) is
    if (command.execute())
        history.push(command)

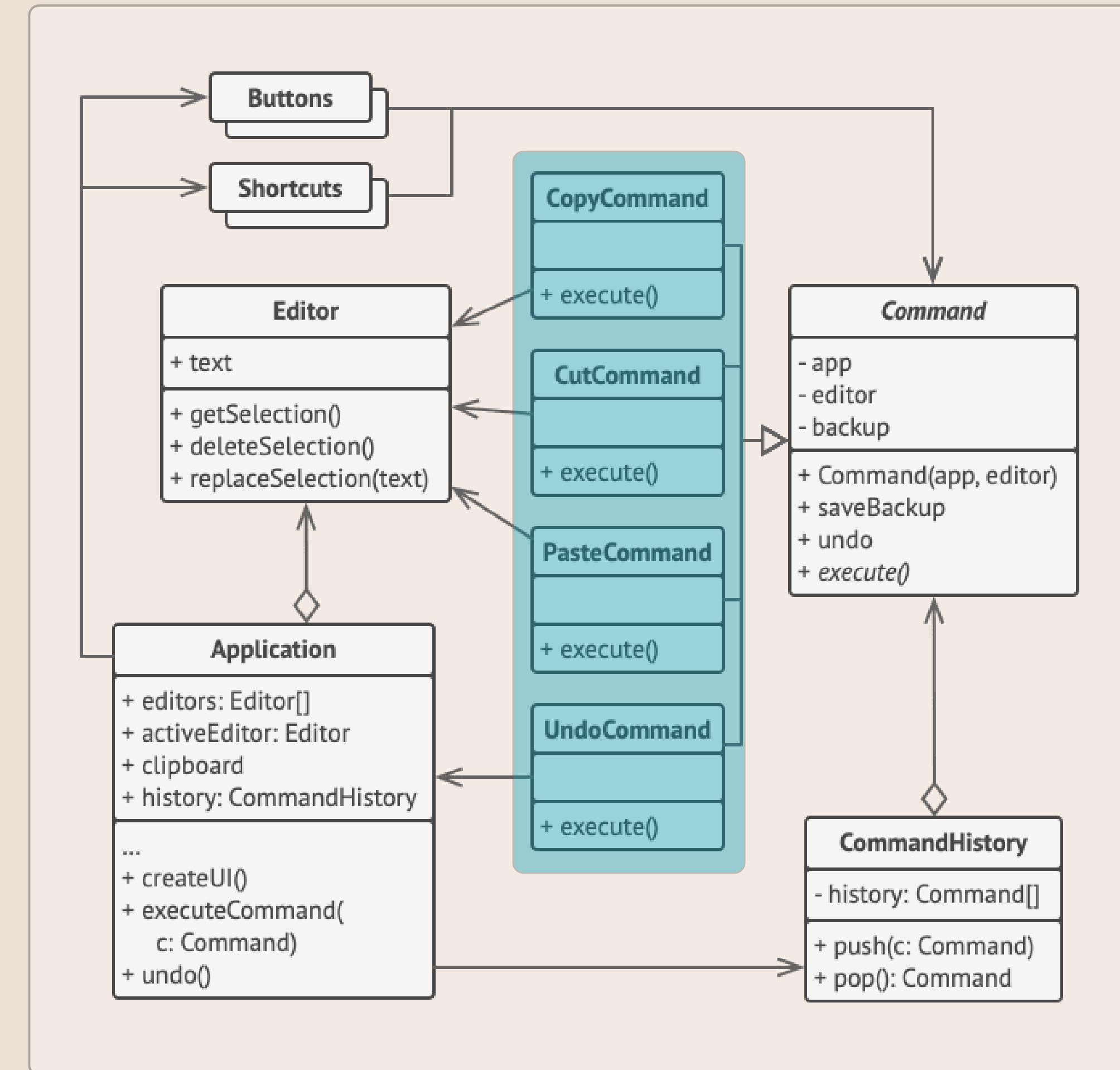
// Take the most recent command from the history and run its
// undo method. Note that we don't know the class of that
// command. But we don't have to, since the command knows
// how to undo its own action.

method undo() is
    command = history.pop()
    if (command != null)
        command.undo()
```

IMPLEMENTAÇÃO PSEUDOCÓDIGO

As classes **concretas** implementam a classe abstrata *Command*, de forma que cada uma delas orienta e delega a execução das ações aos **receivers** (*Editor* e *Application*), para copiar, recortar, colar e desfazer alterações no texto

```
class CutCommand extends Command is
    // The cut command does change the editor
    // it must be saved to the history. And it
    // long as the method returns true.
    method execute() is
        saveBackup()
        app.clipboard = editor.getSelection()
        editor.deleteSelection()
        return true
```



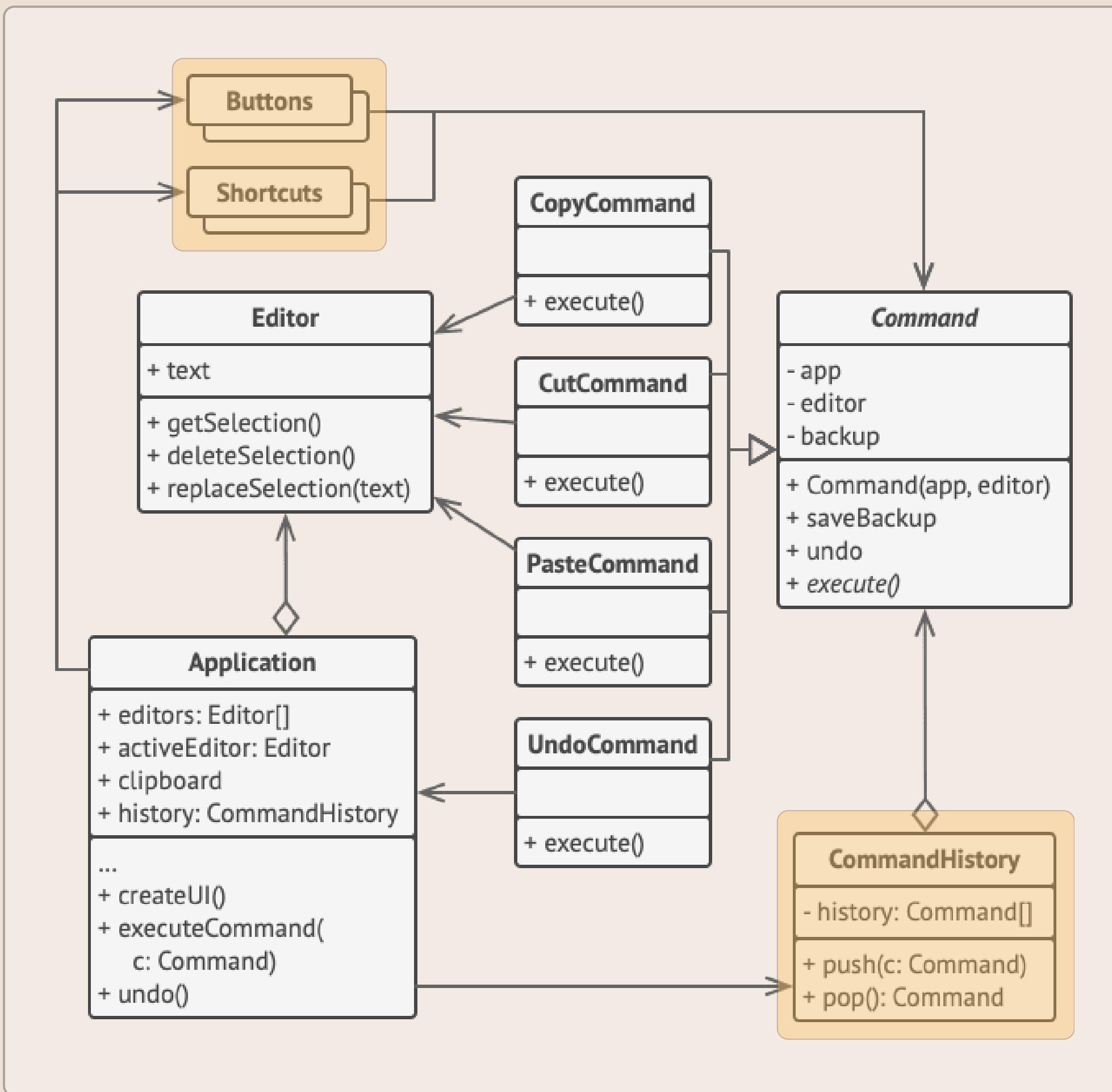
IMPLEMENTAÇÃO PSEUDOCÓDIGO

- O código do cliente (interface e históricos) está ligado à interface dos comandos pois, desta forma, admite a inserção de novos comandos sem que haja fragmentação do código original

```
class CommandHistory is
    private field history: array of Command

    // Last in...
    method push(c: Command) is
        // Push the command to the end of the history

    // ...first out
    method pop():Command is
        // Get the most recent command from the history
```



IMPLEMENTAÇÃO PSEUDOCÓDIGO

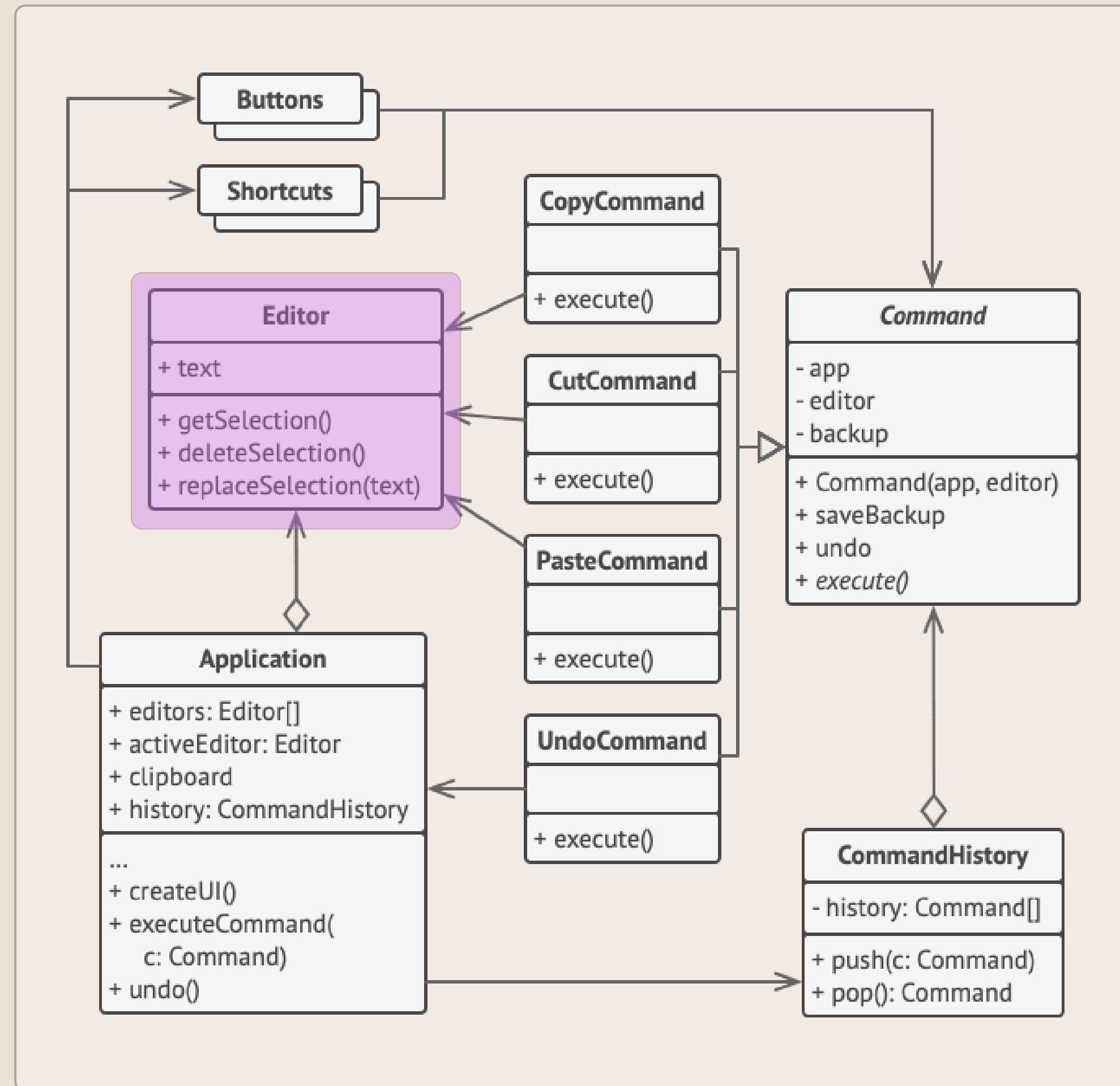
- Os comandos abstratos delegam os pedidos aos métodos do **Editor** que, neste caso, realiza o papel de classe destinatária (*receiver*)

```
class Editor is
    field text: string

    method getSelection() is
        // Return selected text.

    method deleteSelection() is
        // Delete selected text.

    method replaceSelection(text) is
        // Insert the clipboard's content
        // at the selection's position.
```



FONTES E REFERÊNCIAS



The Big Mortal Kombat Gameplay Write Up Pt. 1 – Why it Rules.
Disponível em: <<https://themsfightingwordsblog.com/2022/02/24/the-big-mortal-kombat-gameplay-write-up-pt-1-why-it-rules/>>. Acesso em: 5 nov. 2024.

Steam Community :: Guide :: [How to] Dead Cells. Disponível em: <<https://steamcommunity.com/sharedfiles/filedetails/?id=973253526>>. Acesso em: 5 nov. 2024.

Play Chess Online - Chess.com. Disponível em: <<https://www.chess.com/play>>. Acesso em: 24 out. 2024.

SHVETS, A. Command. Disponível em: <<https://refactoring.guru/pt-br/design-patterns/command>>. Acesso em: 28 out. 2024.

OBRIGADO PELA
ATENÇÃO!

