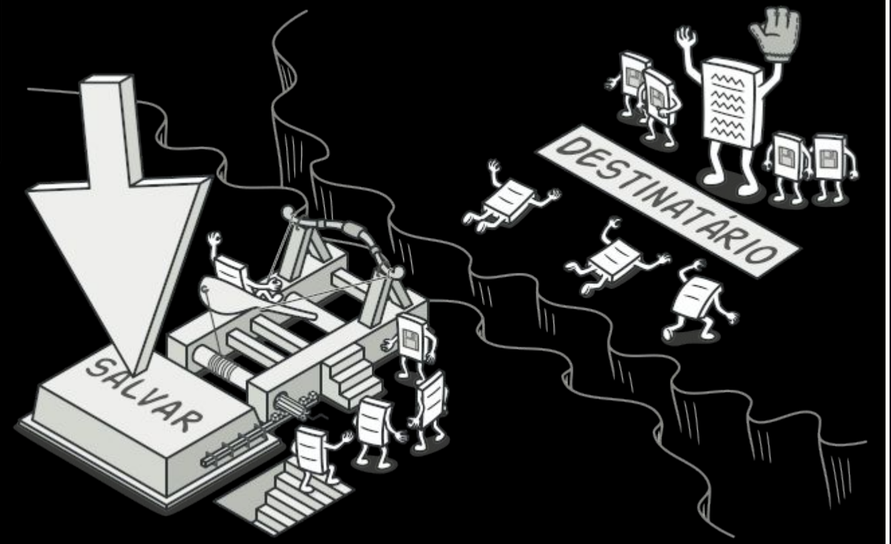
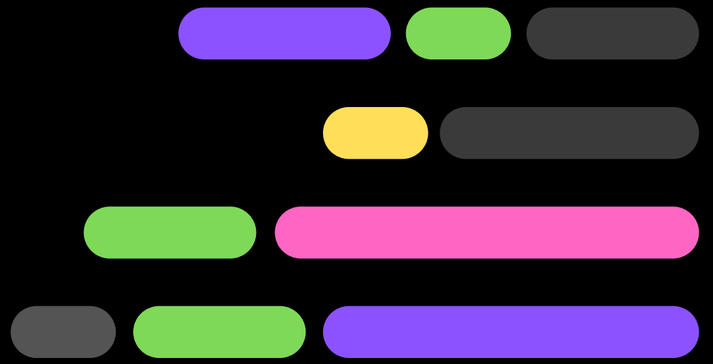
{COMMAND PATTERN}

Carla Cavalcante
Juliana Guarloth



01

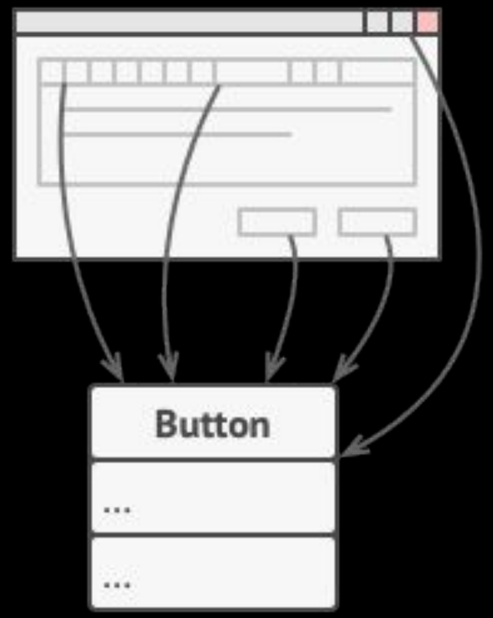
INTRODUÇÃO



"Encapsular uma requisição como um objeto, permitindo que clientes parametrizem diferentes requisições, filas ou requisições de log, e suportar operações reversíveis." [GoF]

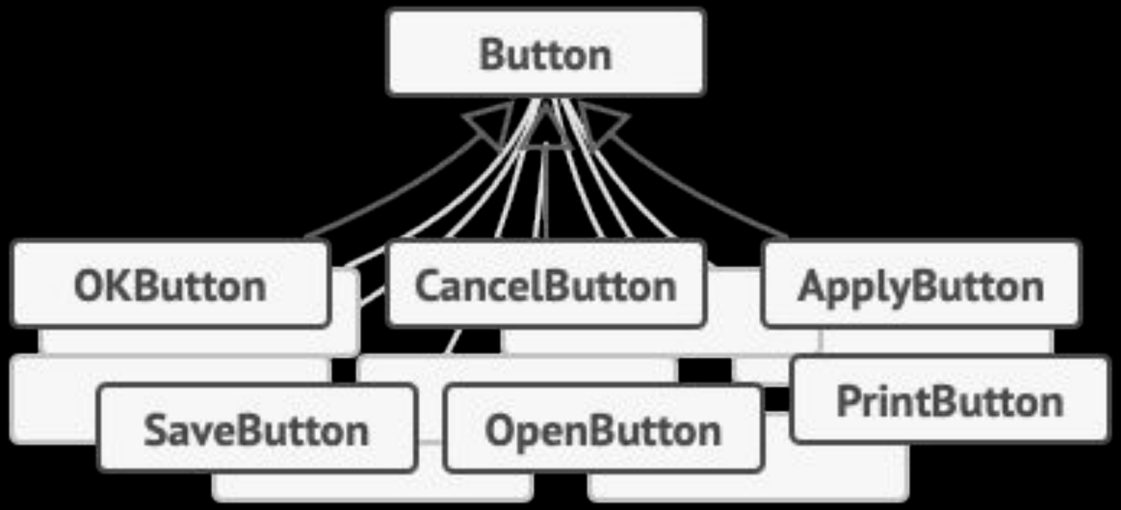
PROBLEMA ☹️

- Em um editor de texto todos os botões são derivações da mesma classe.



PROBLEMA ☹️

- Mesmo os botões sendo similares, possuem funções diferentes, apesar de parecerem similares, todos fazem coisas diferentes, sendo necessário criar várias subclasses.

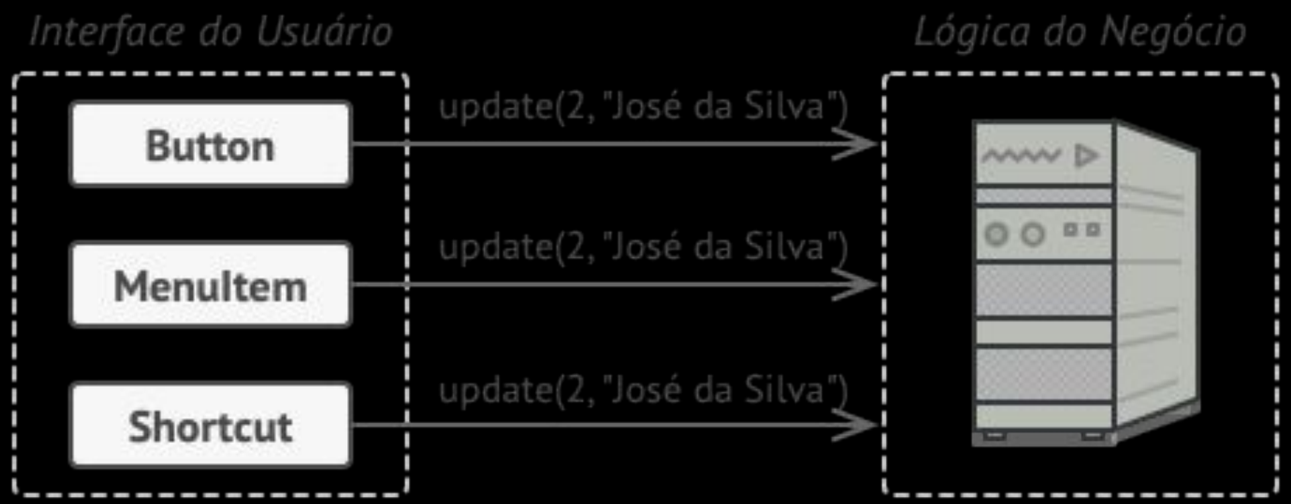


PROBLEMA ☹️

- Isso gera um problema de alta dependência
- Manutenção difícil, baixa flexibilidade, e propensão a quebras.

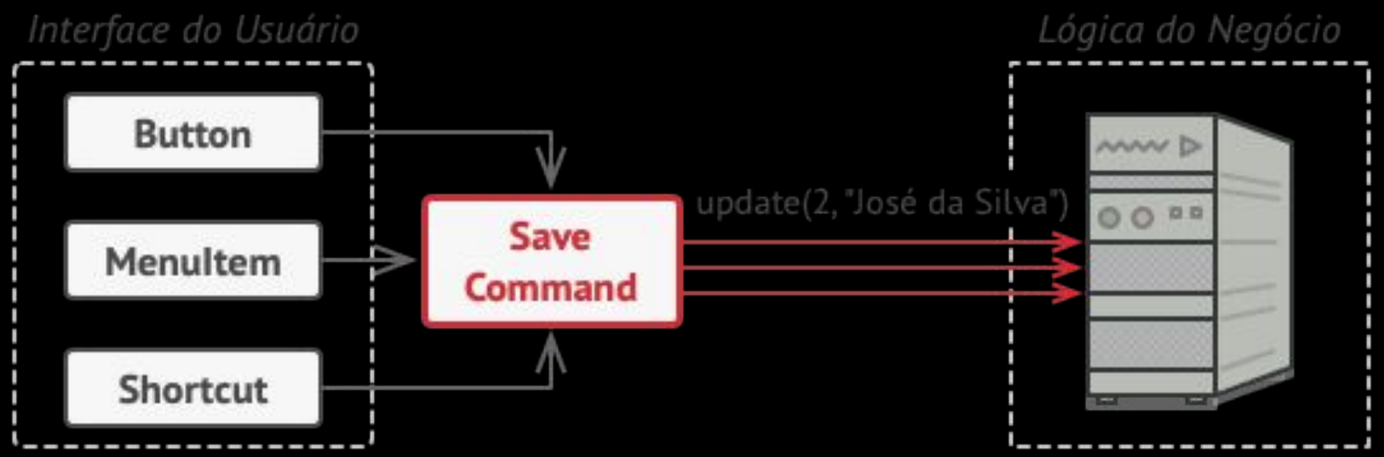
SOLUÇÃO 😁

- Princípio da Separação de Interesses
- Os objetos GUI podem acessar os objetos da lógica do negócio diretamente.



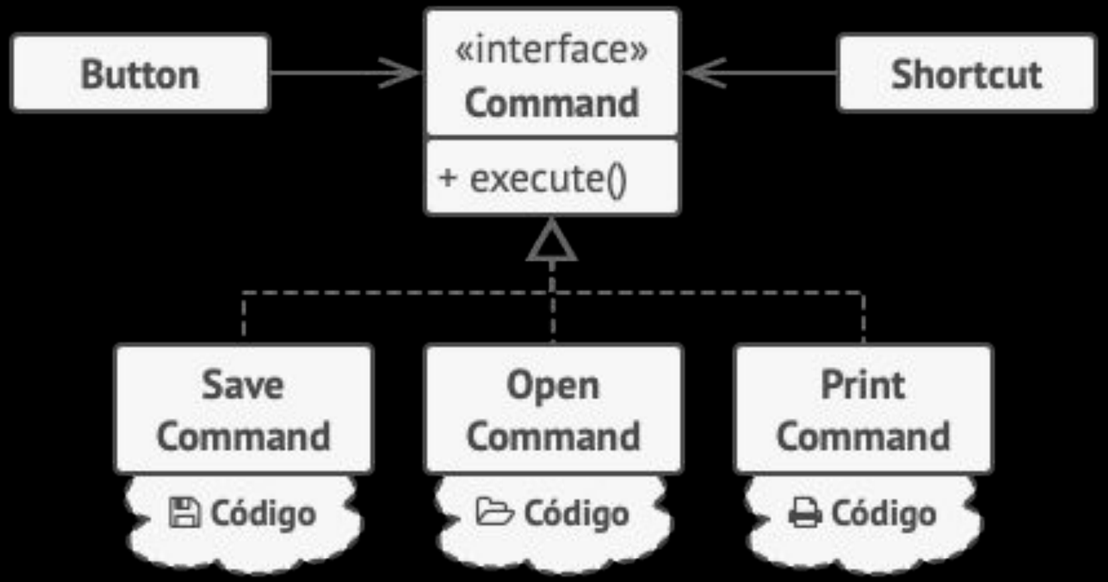
SOLUÇÃO 😁

- Acessa a lógica de negócio através do comando, desacoplando o emissor da solicitação do seu processamento.

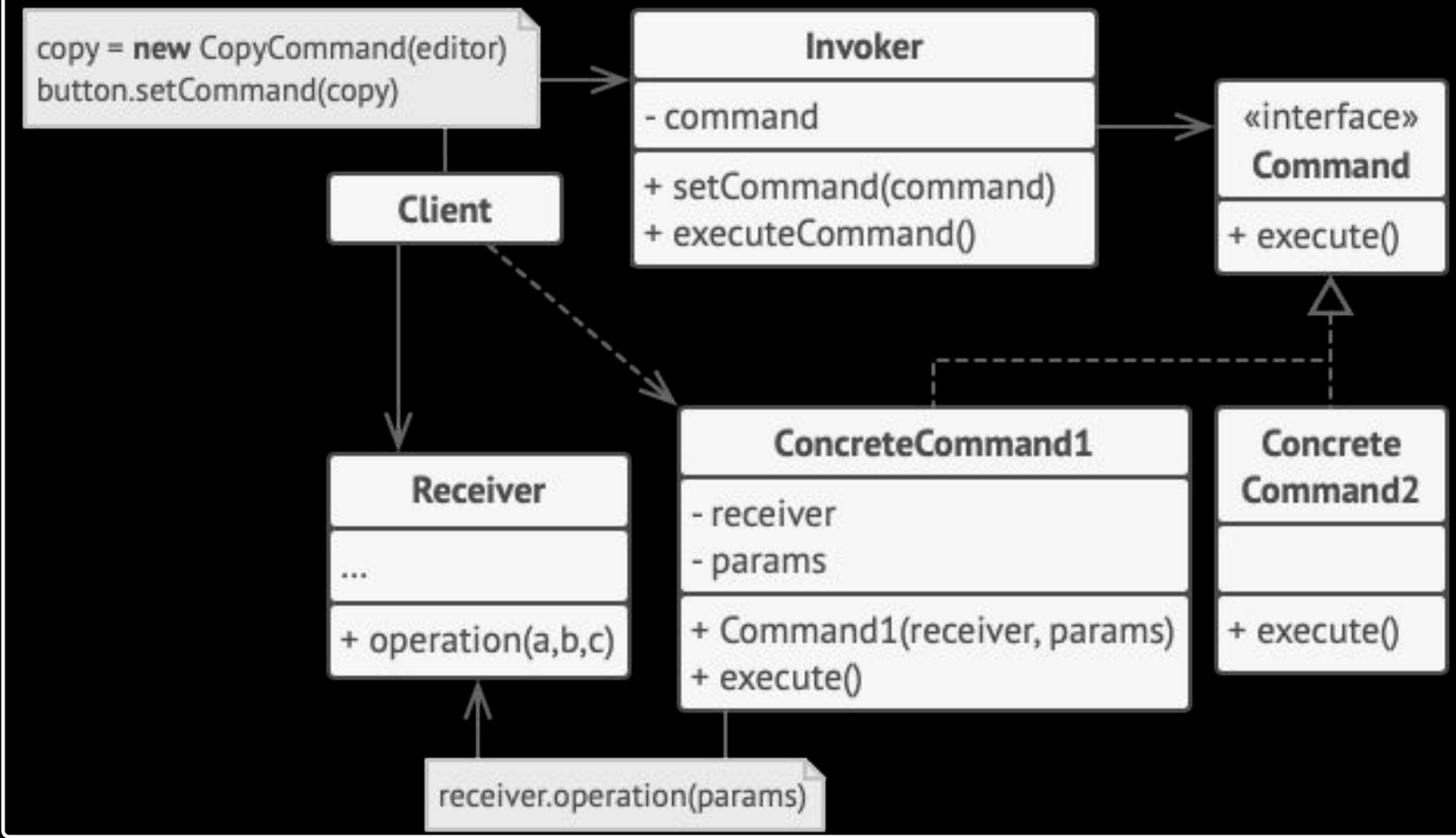


SOLUÇÃO 😁

- Implementação da interface comando



ESTRUTURA



- Invoker
- Comando (interface)
- Comando Concreto
- Receiver
- Cliente


Buttons
Shortcuts
CopyCommand
+ execute()
Editor
+ text
+ getSelection()
+ deleteSelection()
+ replaceSelection(text)
CutCommand
+ execute()
Command
- app
- editor
- backup
+ Command(app, editor)
+ saveBackup
+ undo
+ execute()
PasteCommand
+ execute()
Application
+ editors: Editor[]
+ activeEditor: Editor
+ clipboard
+ history: CommandHistory
...
+ createUI()
+ executeCommand(
c: Command)
+ undo()
UndoCommand
+ execute()
CommandHistory
- history: Command[]
+ push(c: Command)
+ pop(): Command

INTERFACE COMMAND

Interface que define um único método "execute()" que é implementado por classes concretas para realizar operações específicas.

```
public interface TextFileOperation {
    String execute();
}
```

COMANDO CONCRETO

Os comandos concretos implementam a interface command com pedidos específico.

```
public class OpenTextFileOperation
implements TextFileOperation {
    private TextFile textFile;
    @Override
    public String execute() {
        return textFile.open();
    }
}
```

COMANDO CONCRETO

Os comandos concretos implementam a interface command com pedidos específico.

```
public class SaveTextFileOperation implements
TextFileOperation {
     private TextFile textFile;
    public SaveTextFileOperation(TextFile
textFile) {
        this.textFile = textFile;
    }
    public String execute() {
        return textFile.save();
    } }
```

RECEIVER

Define as operações reais de abrir e salvar um arquivo. É o componente que executa a ação real quando o método execute() do comando é chamado.

```
public class TextFile {
    private String name;
    public TextFile(String name) {
        this.name = name;
    }
    public String open() {
        return "Opening file " + name; }
    public String save() {
        return "Saving file " + name; }
}
```

INVOKER

Um invoker é um objeto que sabe como executar um dado comando, mas não sabe como o comando foi implementado. Ele só conhece a interface do comando.

```
public class TextFileOperationExecutor {


    private final List<TextFileOperation>
textFileOperations
     = new ArrayList<>();


    public String executeOperation(TextFileOperation
textFileOperation) {
        textFileOperations.add(textFileOperation);
        return textFileOperation.execute();
    }
}
```

CLIENTE

Um cliente é um objeto que controla o processo de execução de comandos especificando quais comandos executar e em quais estágios do processo executá-los.

```
public static void main(String[] args) {
    TextFileOperationExecutor textFileOperationExecutor
      = new TextFileOperationExecutor();
    textFileOperationExecutor.executeOperation(
      new OpenTextFileOperation(new
TextFile("file1.txt"))));
    textFileOperationExecutor.executeOperation(
      new SaveTextFileOperation(new
TextFile("file2.txt"))));
}
```

APLICABILIDADE

Utilize o padrão Command quando você quer colocar operações em fila, agendar sua execução, ou executá-las remotamente.

```
public class CommandInvoker {
    private List<TextFileOperation> commandQueue = new
ArrayList<>();
    public void scheduleCommand(TextFileOperation command)
{         commandQueue.add(command);}
    public void executeScheduledCommands() {
        for (TextFileOperation command : commandQueue){
            command.execute();}
        commandQueue.clear();    }
    public void executeCommand(TextFileOperation command)
{
       command.execute();    }}
```

APLICABILIDADE

Utilize o padrão Command quando você quer implementar operações reversíveis.

```
public class CutTextFileOperation implements
TextFileOperation {
    private TextFile textFile;
    private String backup;
    public CutTextFileOperation(TextFile textFile) {
        this.textFile = textFile; }
    @Override
    public String execute() {
        backup = textFile.getText();
        return textFile.cut();}
    public void undo() {
        textFile.setText(backup); } }
```

Prós e contras

Prós

- Princípio de responsabilidade única.
- Princípio aberto/fechado.
- Desfazer/refazer
- Possibilidade de adiar/agendar as operações

Contras

- O código pode ficar mais complicado uma vez que você está introduzindo uma nova camada entre remetentes e destinatários.