Padrões de Projeto

Prof. Adilson Vahldick

Departamento de Engenharia de Software

Udesc Ibirama

Objetivos da aula

- Conhecer e aplicar o padrão
  - Command

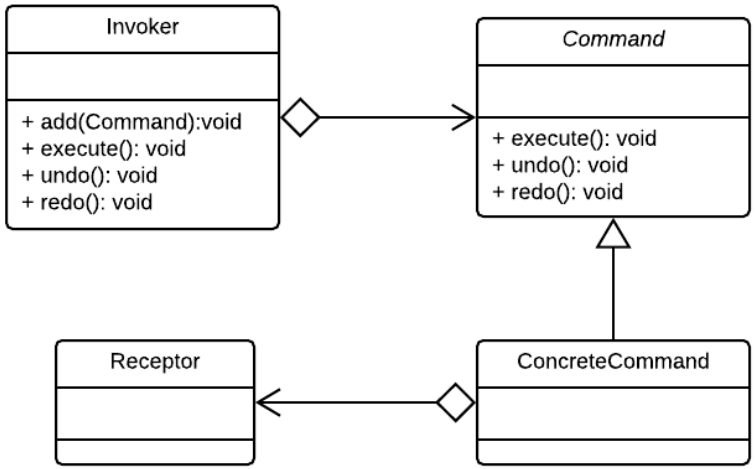
Problema

- As operações (adição, subtração, etc) possam ser desfeitas

- Registrar todas as operações feitas

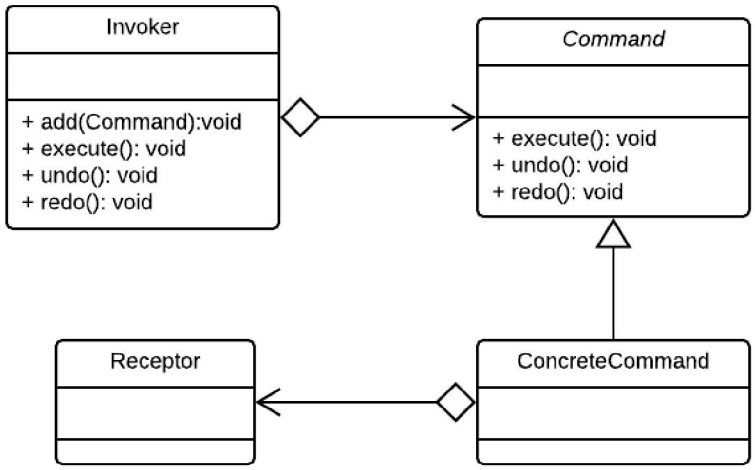
Solução (1)

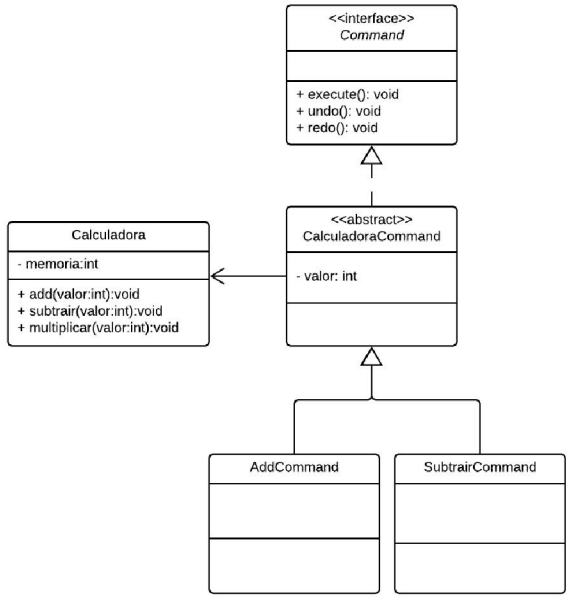
- **Command:** encapsular uma solicitação como um objeto, desta forma permitindo parametrizar clientes com diferentes solicitações, enfileirar ou fazer o registro (log) de solicitações e suportar operações que podem ser desfeitas



Solução (2)

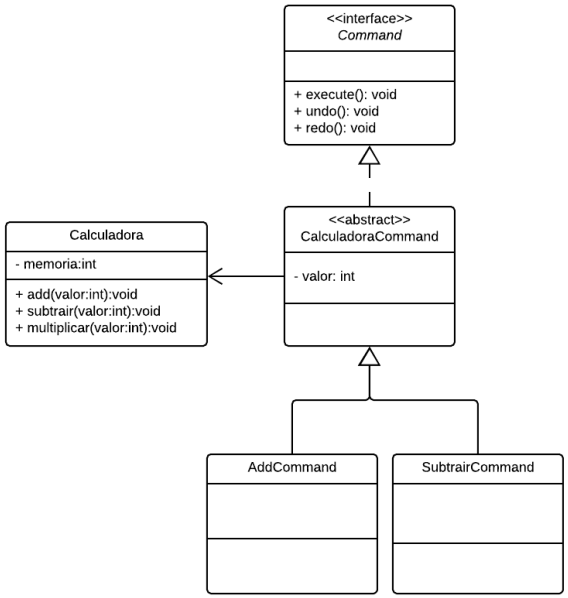
- **Command:** encapsular uma solicitação como um objeto, desta forma permitindo parametrizar clientes com diferentes solicitações, enfileirar ou fazer o registro (log) de solicitações e suportar operações que podem ser desfeitas





ick

Solução (3)

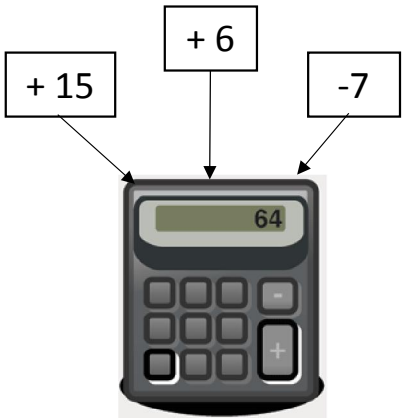


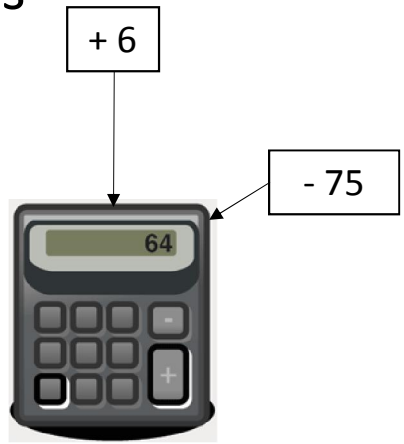

```
public abstract class CalculadoraCommand implements Command {

  protected Calculadora calc;
  protected int valor;

  public CalculadoraCommand(Calculadora calc, int valor) {
    this.calc = calc;
    this.valor = valor;
  }
}
```

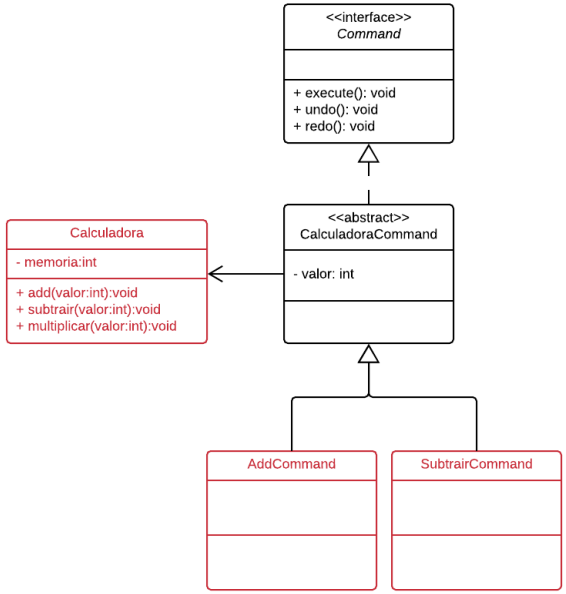
```
public class AddCommand extends CalculadoraCommand {

  public AddCommand(Calculadora calc, int valor) {
    super(calc, valor);
  }

  public void execute() { calc.add(valor); }

  public void undo() { calc.subtrair(valor); }

  public void redo() { calc.add(valor); }

  public String toString() { return " + " + valor; }

}
```

Solução (4)

- **Command:** 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



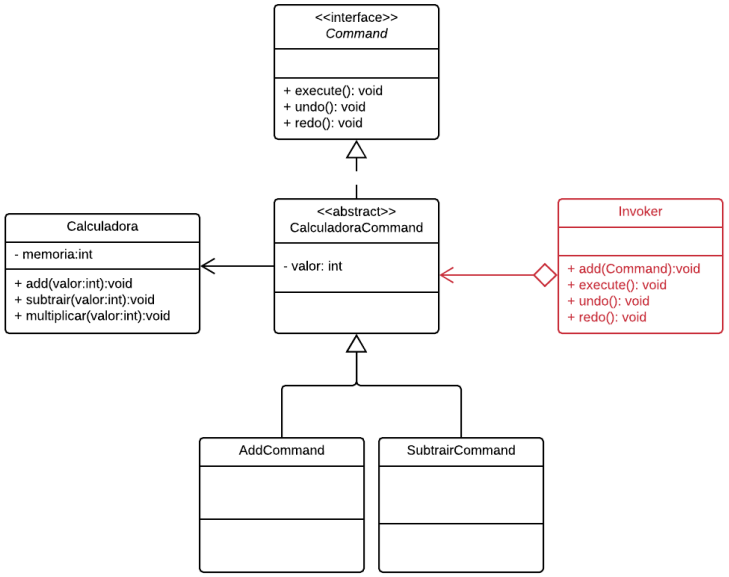




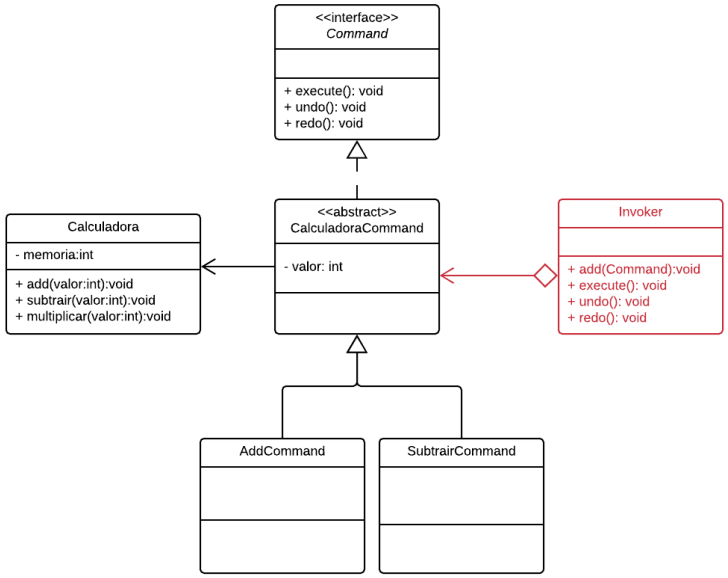
Pa k

Solução (5)

- **Command:** 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



Solução (6)




```
public class Invoker {

  private List<Command> imediatos = new ArrayList<>();
  private List<Command> todos = new ArrayList<>();

  public void add(Command comm) {
    imediatos.add(comm);
  }

  public void execute() {
    for (Command comm:imediatos) {
      comm.execute();
      todos.add(comm);
    }
    imediatos.clear();
  }

  public void imprimir() {
    System.out.println("Log :");
    for (Command comm:todos) {
      System.out.println(" " + comm);
    }
  }
}
```

Gravar num arquivo texto o comando, por exemplo,
Log.gravar(comm.toString());

Solução (7)

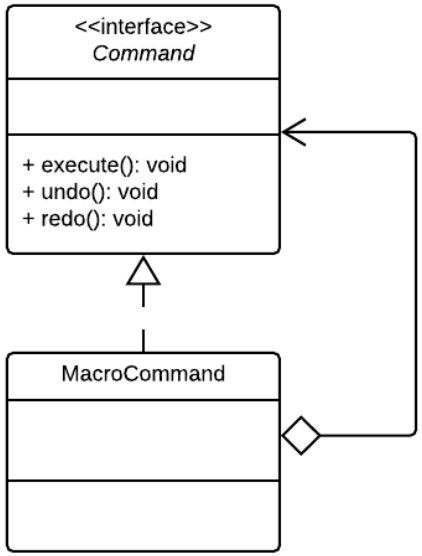
- **Command:** encapsular uma solicitação como um objeto, desta forma permitindo parametrizar clientes com diferentes solicitações, enfileirar ou fazer o registro (log) de solicitações e suportar operações que podem ser desfeitas

```
public class Invoker {

  private List<Command> todos = new ArrayList<>();
  private List<Command> undo = new ArrayList<>();

...

  public void undo() {
    Command comm = todos.remove(todos.size()-1);
    comm.undo();
    undo.add(comm);
  }

  public void redo() {
    Command comm = undo.remove(undo.size()-1);
    comm.redo();
    todos.add(comm);
  }

 }
```

```
public class AddCommand extends CalculadoraCommand {

  public AddCommand(Calculadora calc, int valor) {
    super(calc, valor);
  }

  public void execute() { calc.add(valor); }

  public void undo() { calc.subtrair(valor); }

  public void redo() { calc.add(valor); }

  public String toString() { return " + " + valor; }

}
```

Solução (8)

- **Macro de comandos**




```
public class MacroCommand implements Command {

  private List<Command> comms = new ArrayList<>();

  public void add(Command comm) { comms.add(comm); }

  public void execute() {
    for (Command comm:comms) {
      comm.execute();
    }
  }

  public void undo() {
    for (int i=comms.size()-1; i>=0; i--) {
      comms.get(i).undo();
    }
  }

  public void redo() {
    for (Command comm:comms) {
      comm.redo();
    }
  }

  public String toString() {
    String ret = "Macro : ";
    for (Command comm:comms) {
      ret += "\n " + comm;
    }
    return ret;
  }
}
```

```
Calculadora calc2 = new Calculadora();
MacroCommand macro = new MacroCommand();
macro.add(new AddCommand(calc2, 50));
macro.add(new SubtrairCommand(calc2, 20));
macro.add(new AddCommand(calc2, 70));

inv.execute(macro);

inv.undo();

inv.redo();
```

Exercício 1 – command1

- **Calculadora**

- Crie um novo Command para multiplicação

- Altere a classe Sistema adicionando um novo objeto desse novo Command na execução

Exercício 2 – command2

- **Controle de estoque**

- Falta implementar o comando para tirar do estoque (RemoverEstoque)
- SistemaV1:
  - Adicione as referências para usar esse Command
- SistemaV2:
  - Descomente na classe Cliente

Exercício 3 – command3

- Cadastro de Pessoas

- Já está implementado um dos comandos para listar todas as pessoas
- Implementar um comando para incluir, excluir e obter uma pessoa

- Quando necessário, incluir métodos no Observer

Exercício 4 – command4

- Refatorar a classe PaintFixo
  - Aplicar padrão Command
  - Os objetos de comando é que devem ser os responsáveis por desenhar
  - Implementartambém Undo e Redo