Padrões de Projeto

Prof. Adilson Vahldick

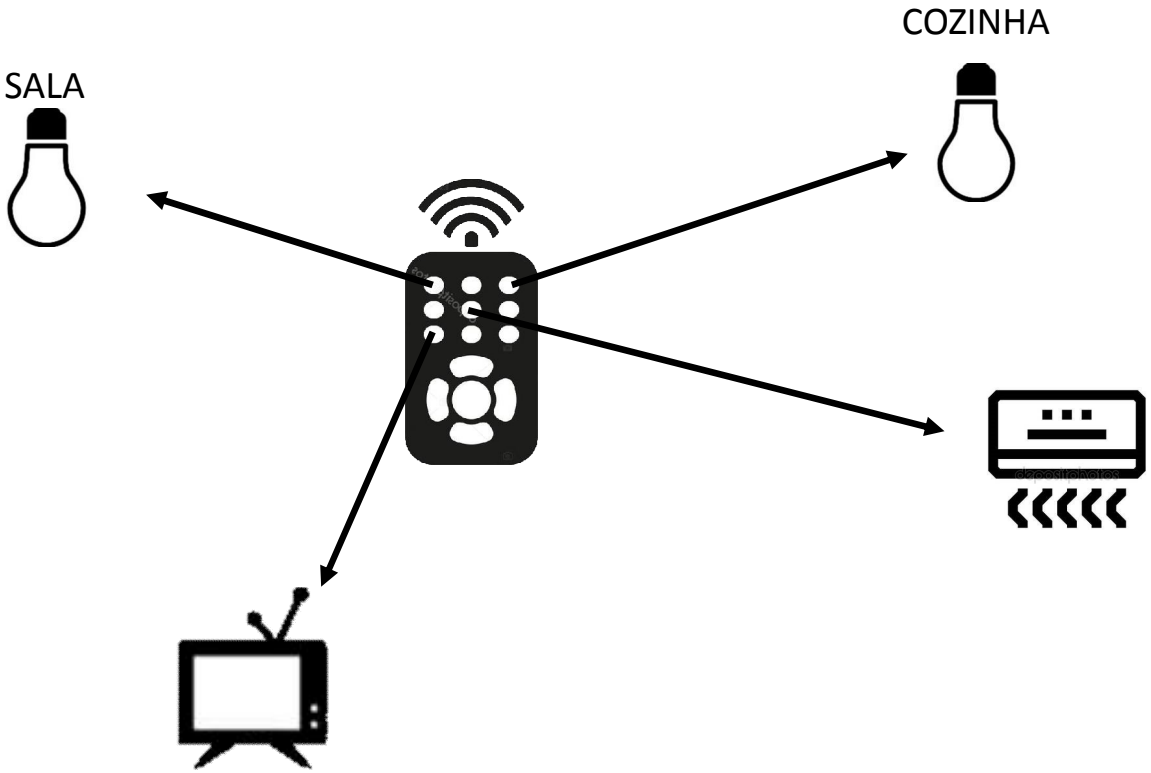
Departamento de Engenharia de Software

Udesc Ibirama

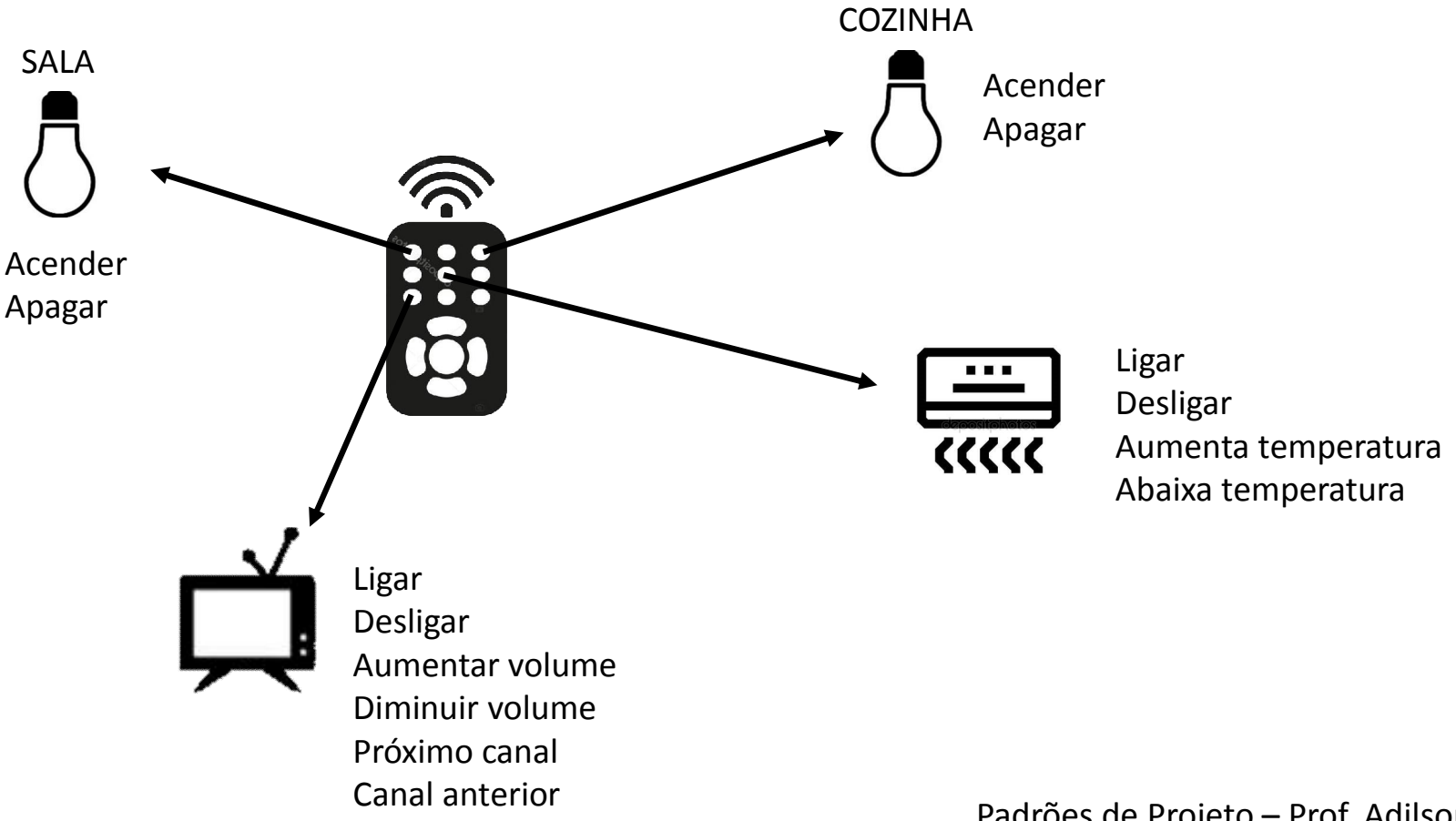
Objetivos da aula

- Conhecer e aplicar o padrão
  - Command

Problema (1)



Problema (2)

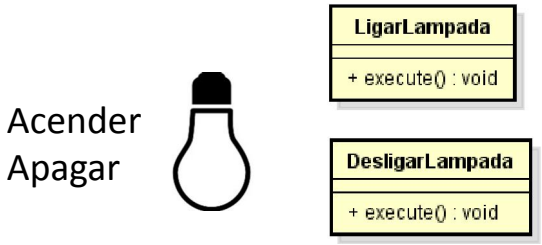

SALA
Acender
Apagar
COZINHA
Acender
Apagar
Ligar
Desligar
Aumenta temperatura
Abaixa temperatura
Ligar
Desligar
Aumentar volume
Diminuir volume
Próximo canal
Canal anterior

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

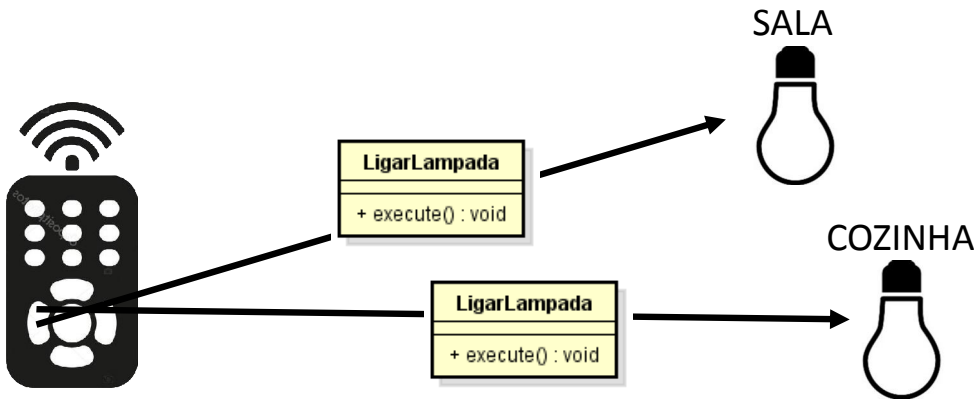
Solução (2)

- **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



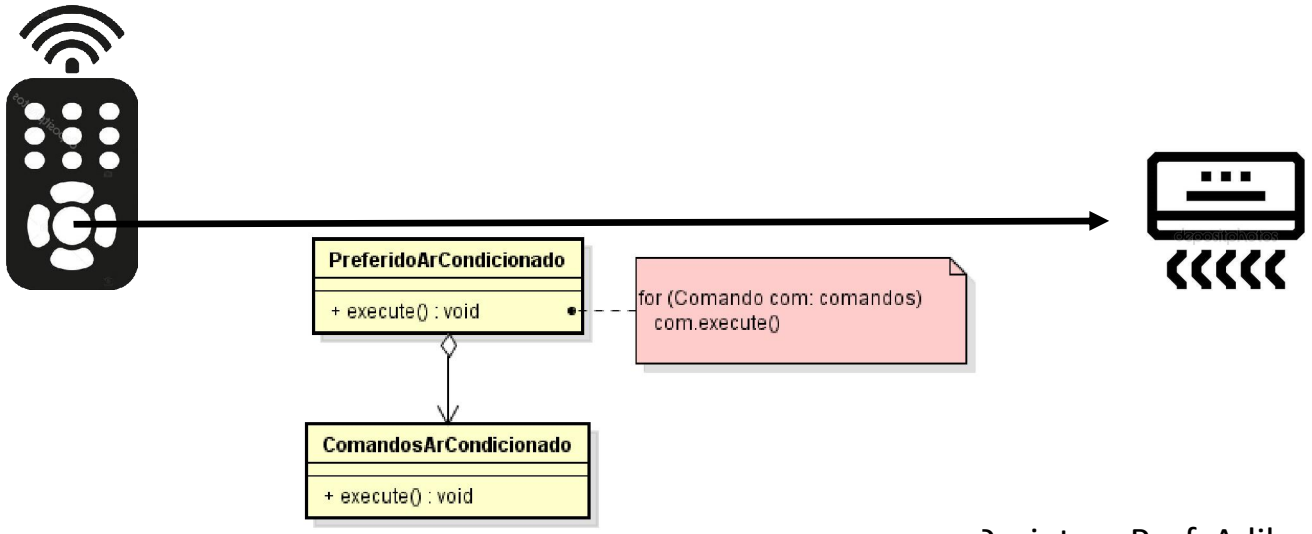
Solução (3)

- **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



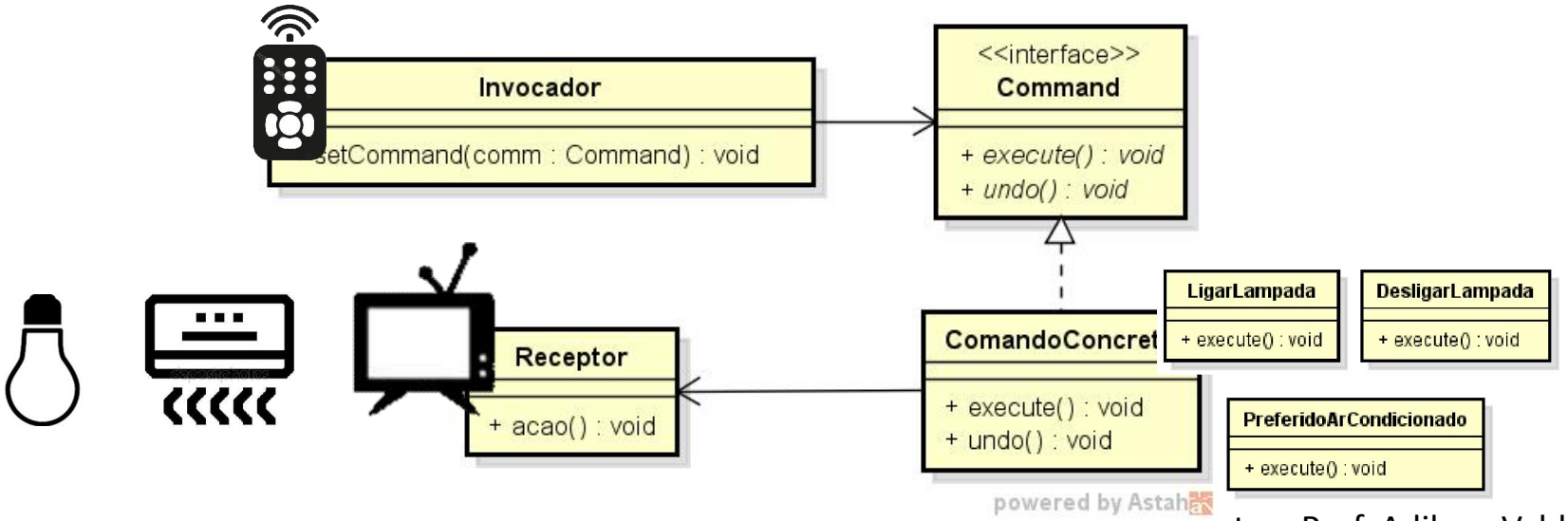
Solução (4)

- **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



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) – command1

- 9 botões para indicar que equipamento será usado

```
// vamos começar a controlar o equipamento do botão 1
controle.pressBotao(ControleRemoto.BOTAO_1);
// vamos começar a controlar o equipamento do botão 2
controle.pressBotao(ControleRemoto.BOTAO_2);
```

Solução (7) – command1

- 5 botões para executar uma ação do equipamento

```
controle.pressBotao(ControleRemoto.BOTAO_1);

// vamos interagir com o equipamento 1
controle.pressBotao(ControleRemoto.BOTAO_CIMA);
controle.pressBotao(ControleRemoto.BOTAO_BAIXO);
```

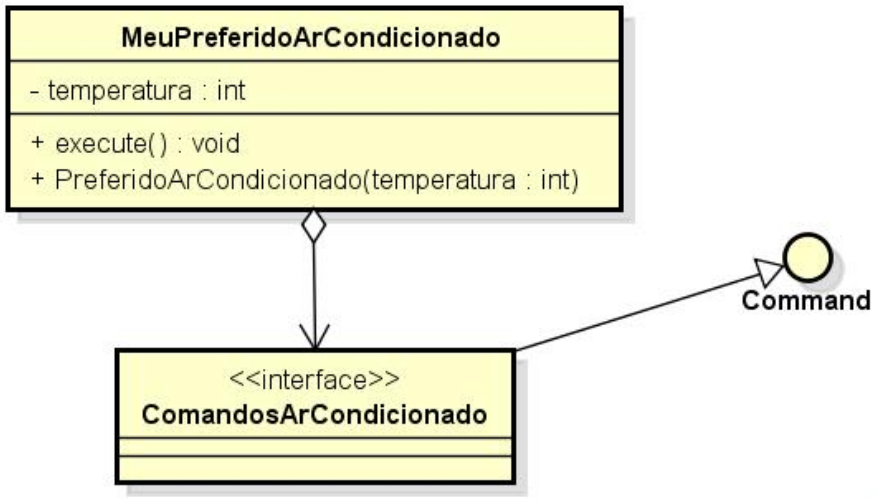
Solução (8) – command1


```
public class ControleRemoto {

    private int equip = 0;

    public void configurar(int botao1_9, int botaoTeclasEspeciais, Command comando) {
        botoes[botao1_9][botaoTeclasEspeciais-10] = comando;
    }

    public void pressBotao(int botao) {
        if (botao <= 9) {
          equip = botao - 1;
        } else {
          Command comm = botoes[equip][botao-10];
          comm.execute();
        }
    }

}
```

```
controle.pressBotao(ControleRemoto.BOTAO_1);

// vamos interagir com o equipamento 1
controle.pressBotao(ControleRemoto.BOTAO_CIMA);
controle.pressBotao(ControleRemoto.BOTAO_BAIXO);
```

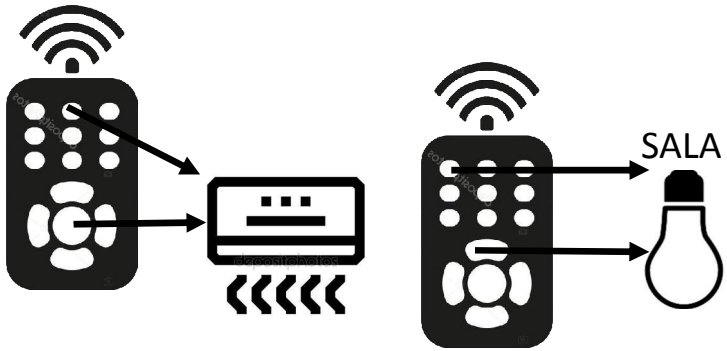
Solução (9) – command1

- **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 (10) – command1

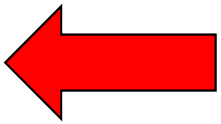
- **Command:** encapsular uma solicitação como um objeto, desta forma permitindo parametrizar clientes com diferentes solicitações, enfileirar ou fazer o registro (log) de solicitações e suportar operações que podem ser desfeitas

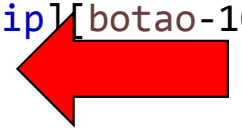



```
Selecionado Equip 2
Executado Ligar
Executado Aumentar Temp
Executado Ligar

Selecionado Equip 1
Executado Ligar
```

Solução (11) – command1

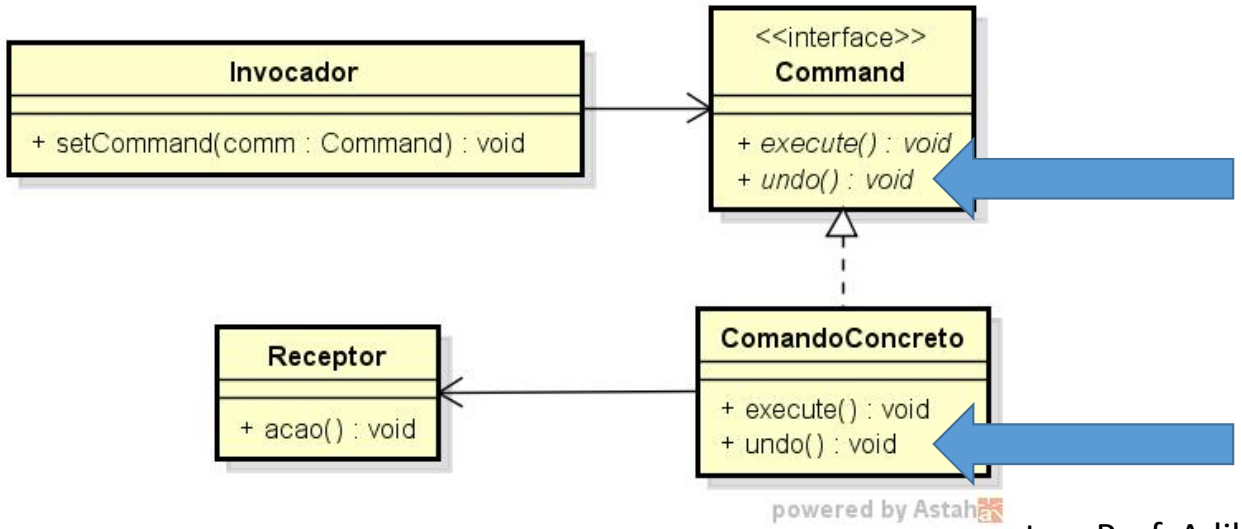
```
public void pressBotao(int botao) {
    if (botao <= 9) {
        equip  = botao - 1;
        System.out.println("Selecionado Equip " + botao);
    } else {
        Command comm = botoes[equip][botao-10];
        System.out.println(comm);
        comm.execute();
    }
}
```





Solução (12) – command1

- **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 (13) – command1

- **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 interface Command {
  void execute();
  void undo();
}
```

```
public class LigarLampada implements Command {

     public void undo() {
       lampada.apagar();
     }
...
}
```

```
public class ControleRemoto {

  public void desfazerTudo() {
    for (int i = comandosUndo.size()-1; i >= 0; i--) {
      comandosUndo.get(i).undo();
      comandosRedo.add(0, comandosUndo.get(i));
    }

    comandosUndo.clear();
  }
...
}
```

Exercício 1 – command2

- Controle de estoque

- Falta implementar o comando para tirar do estoque
- Adicione a funcionalidade na classe Cliente
- Na classe CommandInvoker, implemente as funcionalidades undo e redo para um comando só

Exercício 1++

- Elimine o switch na classe Cliente

Exercício 2

- Refatore a aplicação do exercício Builder3 para uso do Command, objetivando substituir o código abaixo. Não é necessário ter o método undo().

```
BuilderContato builder = null;
if (jrInternet.isSelected()) {
  builder = new BuilderInternet();
} else {
  if (jrTelefone.isSelected()) {
    builder = new BuilderTelefone();
  } else {
    builder = new BuilderCompleto();
  }
}
```

Exercício 3 – command3

- Refatorar a classe PaintFixo
  - Aplicar padrão Command
  - Command deve ser o responsável por desenhar
  - Implementar Undo
  - Implementar Redo