Instituto de Computação

Técnicas de Programação Avançada

TCC-00.174

Prof.: Anselmo Montenegro

www.ic.uff.br/~anselmo

anselmo@ic.uff.br

Conteúdo: Padrão Command

Documento baseado no material preparado pelo Prof. Luiz André (http://www.ic.uff.br/~lapaesleme/)

Padrões de Projeto

Introdução

Como encapsular uma invocação de um método

Como fazer com que o objeto que invoca uma computação não se preocupe com os detalhes de como o processo é realizado

Como salvar as invocações dos métodos para implementar *do* e *undos* em nossa aplicação

Padrões de Projeto

Exemplo de problema

Considere a implementação de uma API para um Controlador Universal com 7 slots programáveis, cada um como botões on e off

Requisitos funcionais:

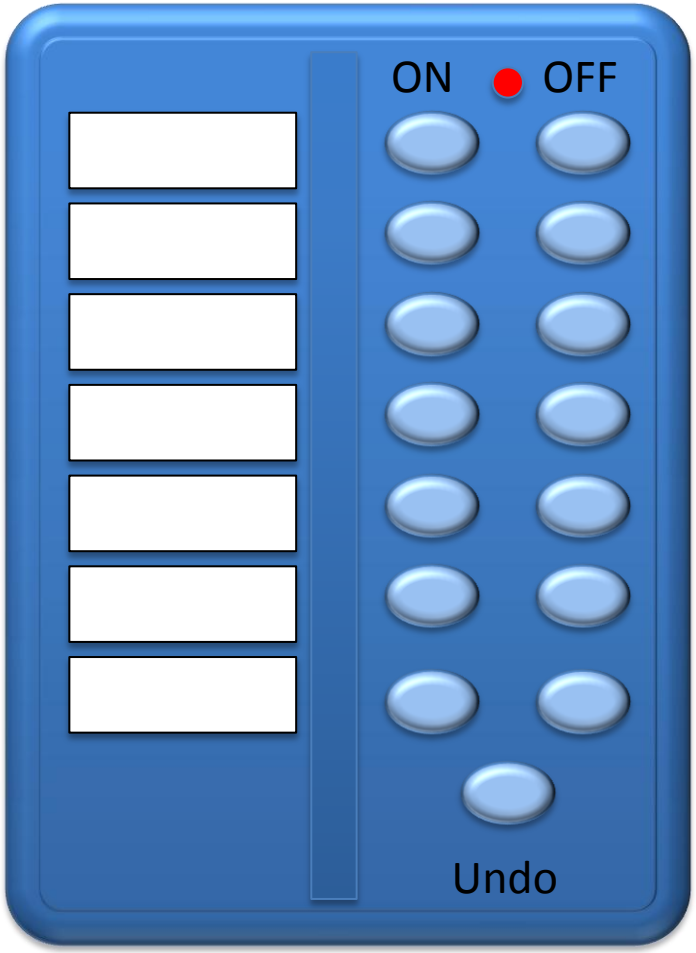
RF1 - O controlador deve permitir a associação de diferentes dispositivos a cada slot.

RF2 - O controlador deve ser capaz de inicialmente controlar um conjunto pré-definido de dispositivos como luzes, ventiladores, equipamento de áudio e outros que estão especificados.

RF3 – O controlador deve ser extensível, isto é poder controlar novos dispositivos que os distribuidores disponibilizarem.

Padrões de Projeto

Exemplo de problema



Padrões de Projeto

Exemplo de problema

Stereo
on() off() setCd() setDvd() setVolume()

ApplianceControl
on() off()

FaucetControl
openValve() closeValve()

GarageDoor
up() down() stop() lightOn() lightOff()

Light
on() off()

OutdoorLight
on() off()

HotTube
circulate() jetsOn() jetsOff() setTemperature()

CellingLight
on() off() dim()

Thermostat
setTemperature( )

TV
on() off() setInputChannel() setVolume()

SecurityControl
arm( ) disarm()

GardenLight
setDuskTime() setDawnTime() low() off() getSpeed()

CeilingFan
high() medium() low() off() getSpeed()

Sprinkler
waterOn() waterOff()

Padrões de Projeto

Exemplo de problema

Problemas:

Cada dispositivo tem sua própria interface

As interfaces variam muito de dispositivo para dispositivo

Impossível predeterminar as interfaces dos novos dispositivos que os vendedores podem introduzir

Objetivo:

Desacoplar quem requisita uma ação (o controle) de quem de fato executa a ação (o receptor, no caso, cada dispositivo)

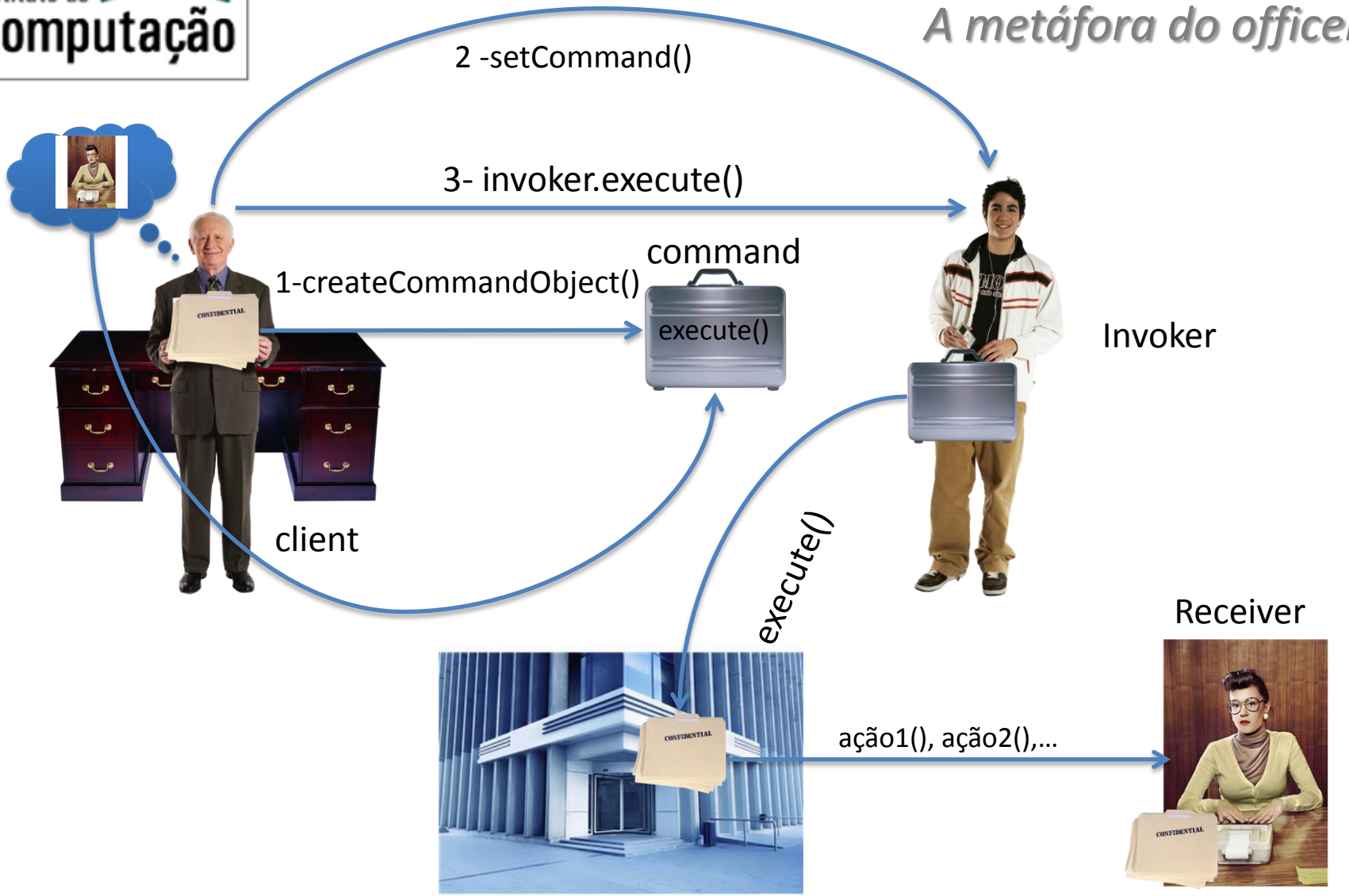
Como:

Encapsular as operações e o receptor na ideia abstrata de um comando com uma interface execute()

O controlador passa a ser apenas um invocador do comando; não sabe como fazer, nem quem de fato o faz

Padrões de Projeto

A metáfora do officeboy



Padrões de Projeto

Exemplo de problema

Refinamento da abordagem:

Encapsular as operações que controlam (ligar e desligar) um dispositivo em um comando (command) que conhece quem sabe executar (receiver)

Tornar o controlador apenas um invocador (invoker) de comandos (ele passa a não conhecer como e quem faz o comando)

O cliente cria o comando, o configura e passa ao invocador (invoker)

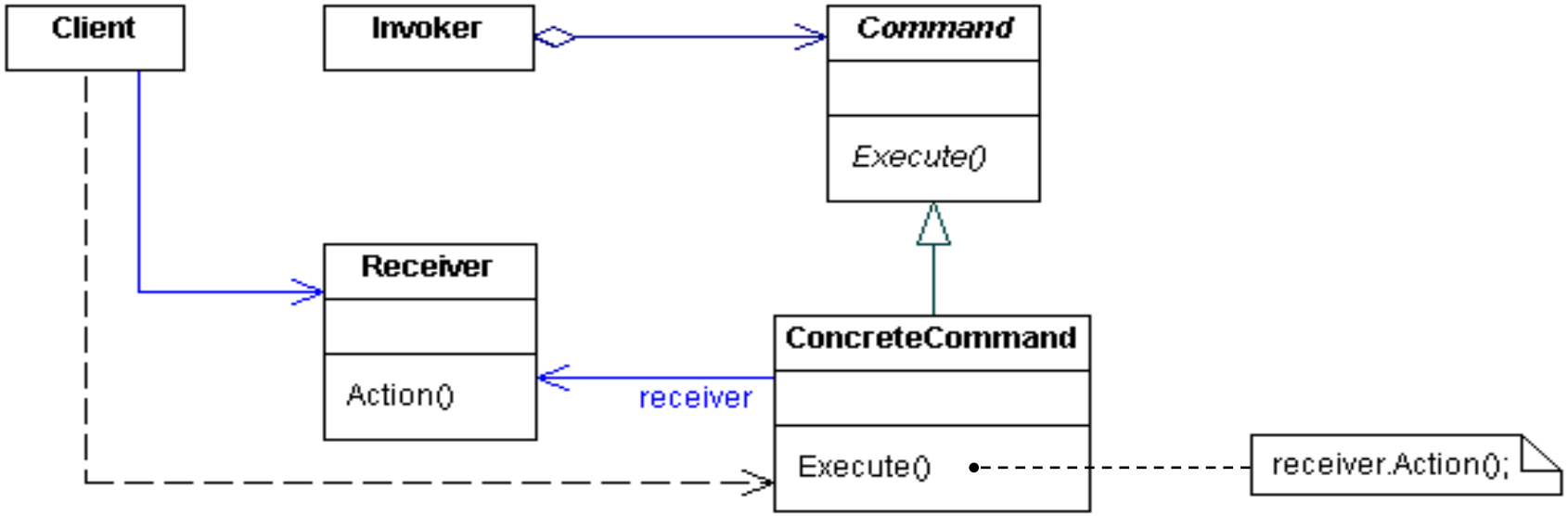
Quando acionado, o invocador dispara (execute()) o comando que é de fato executado em detalhes pelo receiver, efetuando as operações descritas no comando.

Padrões de Projeto

O Padrão Command

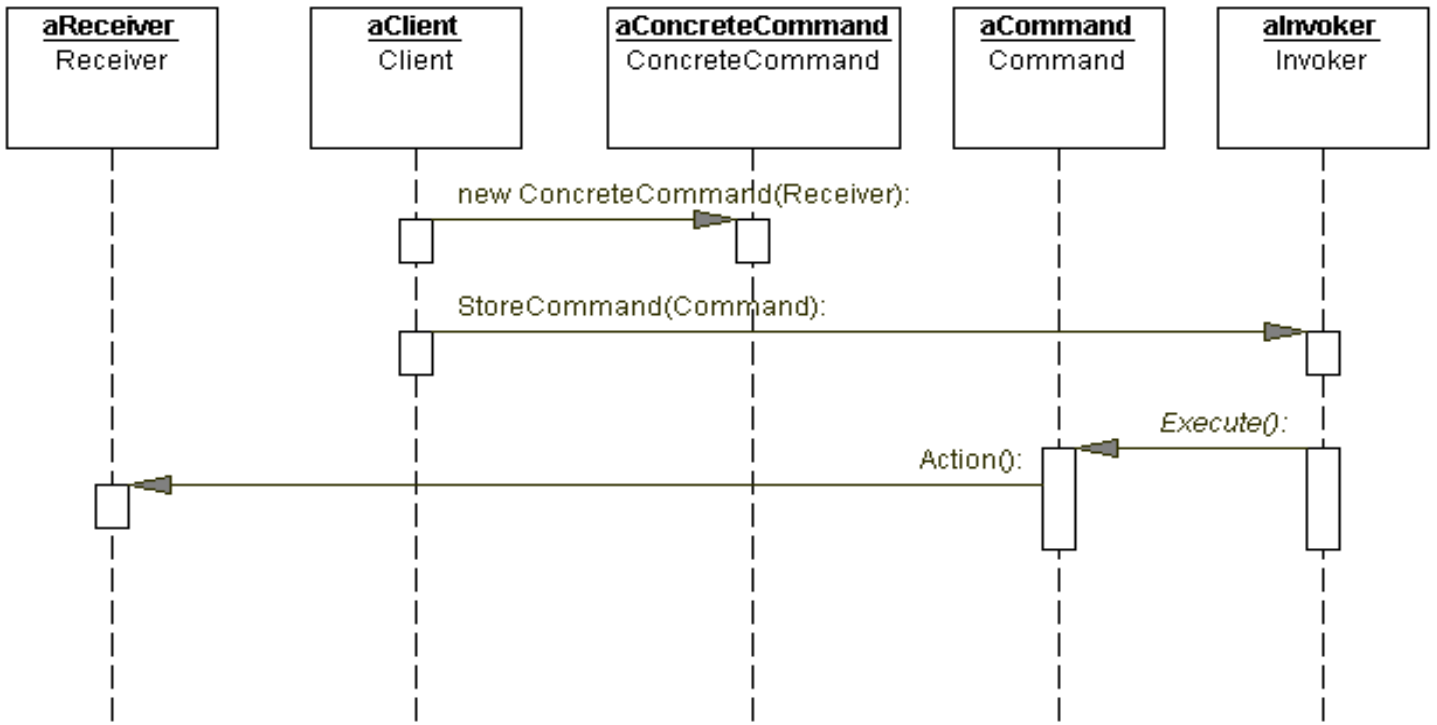
O Padrão Command encapsula uma requisição como um objeto, permitindo parametrizar outros objetos com diferentes requisições, filas ou log de requisições, dando suporte a operações que possam ser desfeitas.

O Padrão Command – Diagrama de Classes



Padrões de Projeto

O Padrão Command – Diagrama de Colaboração



Padrões de Projeto

O Padrão Command

Command

Define a interface para a execução de uma operação

ConcreteCommand

Define uma vinculação entre o um objeto Receiver e uma ação

Implementa Execute através da invocação da(s) correspondente(s) operação(ões) no Receiver.

Client

Cria um objeto ConcreteCommand e estabelece o seu receptor (Receiver).

Invoker

Solicita ao Command a execução da solicitação

Receiver

Sabe como executar as operações associadas a uma solicitação. Qualquer classe pode funcionar como um receiver.

Padrões de Projeto

Solução para o controle remoto usando o Padrão Command

```
public interface Command {
  public void execute();
}
```

```
public class LightOnCommand
implements Command {
  Light light;

  public LightOnCommand(Light light) {
    this.light = light;
  }

  public void execute() {
    light.on();
  }
}
```

```
public class SimpleRemoteControl {

  Command slot;

  public SimpleRemoteControl() {}

  public void setCommand(Command command)
  {
    slot = command;
  }
 public void buttonWasPressed() {
   slot.execute();
 }
}
```

Padrões de Projeto

Solução para o controle remoto usando o Padrão Command

```
public class RemoteControlTest {
  public static void main(String[] args) {
    SimpleRemoteControl remote = new SimpleRemoteControl();
    Light light = new Light();
    LightOnCommand lightOn = new LightOnCommand(light);
    remote.setCommand(lightOn);
    remote.buttonWasPressed();
}
```

Padrões de Projeto

Solução para o controle remoto usando o Padrão Command – vários slots

```
public class RemoteControl {

  Command[] onCommands;
  Command[] offCommands;

  public RemoteControl() {
    onCommands = new Command[7];
    offCommands = new Command[7];

    Command noCommand = new NoCommand();

    for (int i = 0; i < 7; i++) {
      onCommands[i] = noCommand;
      offCommands[i] = noCommand;
    }
```

```
  public void setCommand(int slot, Command onCommand,
    Command offCommand) {
      onCommands[slot] = onCommand;
      offCommands[slot] = offCommand;
  }
  public void onButtonWasPushed(int slot) {
    onCommands[slot].execute();
  }
  public void offButtonWasPushed(int slot) {
    offCommands[slot].execute();
  }
  public String toString() {
    StringBuffer stringBuff = new StringBuffer();
    stringBuff.append("\n------ Remote Control -------\n");

    for (int i = 0; i < onCommands.length; i++) {
      stringBuff.append("[slot " + i + "] " +
      onCommands[i].getClass  ().getName() + "                " +
      offCommands[i].getClass().getName() + "\n");
    }
    return stringBuff.toString();
  }
}
```

Padrões de Projeto

Solução para o controle remoto usando o Padrão Command – vários slots

```
public class LightOffCommand
implements Command {
  Light light;

  public LightOffCommand(Light light) {
    this.light = light;
  }

  public void execute() {
    light.off();
  }
}
```

```
public class StereoOnWithCDCommand implements
Command {
  Stereo stereo;
  public StereoOnWithCDCommand(Stereo   stereo){
     this.stereo = stereo;
  }

  public void execute() {
    stereo.on();
    stereo.setCD();
    stereo.setVolume(11);
  }
}
```

Padrões de Projeto

Como fazer um undo

```
public interface Command {
   public void execute();
   public void undo();
}
```

```
public class LightOffCommand
implements Command {
  Light light;

  public LightOffCommand(Light light) {
    this.light = light;
  }

  public void execute() {
    light.off();
  }

 public void undo() {
    light.on();
 }
}
```

```
public class LightOnCommand
implements Command {
  Light light;

  public LightOnCommand(Light light) {
    this.light = light;
  }

  public void execute() {
    light.on();
  }

 public void undo() {
    light.off();
 }
}
```

Padrões de Projeto

Como fazer um undo

```
public class RemoteControlWithUndo {
  Command[] onCommands;
  Command[] offCommands;
  Command undoCommand;

  public RemoteControlWithUndo() {
    onCommands = new Command[7];
    offCommands = new Command[7];
    Command noCommand = new NoCommand();
    for(int i=0;i<7;i++) {
       onCommands[i] = noCommand;
       offCommands[i] = noCommand;
    }
  }
```

```
  public void setCommand(int slot, Command
onCommand, Command offCommand) {
    onCommands[slot] = onCommand;
    offCommands[slot] = offCommand;
  }

  public void onButtonWasPushed(int slot) {
    onCommands[slot].execute();
    undoCommand = onCommands[slot];
  }

  public void offButtonWasPushed(int slot) {
    offCommands    [slot].execute();
    undoCommand = onCommands[slot];
}

  public void undoButtonWasPushed(){
    undoCommand.execute();
  }

  public String toString() {
   // toString code here...
  }
}
```

Padrões de Projeto

Como fazer um undo para objetos com estados

```
public class CeilingFan {
   public static final int HIGH = 3;
   public static final int MEDIUM = 2;
   public static final int LOW = 1;
   public static final int OFF = 0;
   String location; int speed;

   public CeilingFan(String location) {
      this.location = location;
      speed = OFF;
   }
   public void high() {
     speed = HIGH;
   }
   public void medium() {
    speed = MEDIUM;
   }
   public void low() {
    speed = LOW;
   }
   public void off() {
     speed = OFF;
  }
  public int getSpeed() {
    return speed;
  }
}
```

Padrões de Projeto

Como fazer um undo para objetos com estados

```
public class CeilingFanHighCommand implements Command {
  CeilingFan ceilingFan;
  int prevSpeed;

  public CeilingFanHighCommand(CeilingFan ceilingFan) {
    this.ceilingFan = ceilingFan;
  }

  public void execute() {
    prevSpeed = ceilingFan.getSpeed();
    ceilingFan.high();
  }

  public void undo() {
    if (prevSpeed == CeilingFan.HIGH) {
      ceilingFan.high();
    } else if (prevSpeed == CeilingFan.MEDIUM) {
     ceilingFan.medium();
    } else if (prevSpeed == CeilingFan.LOW) {
     ceilingFan.low();
    } else if (prevSpeed == CeilingFan.OFF) {
     ceilingFan.off();
    }
  }
}
```

Padrões de Projeto

Criando uma Macro

```
public class MacroCommand implements Command {
  Command[] commands;

  public MacroCommand(Command[] commands) {
     this.commands = commands;
  }

  public void execute() {
     for (int i = 0; i < commands.length; i++) {
        commands[i].execute();
     }
  }
}
```

Padrões de Projeto

Exemplo de código para o Padrão Command

```
public interface Command {
   public Object execute(Object arg);
}
```

```
public class Server {
  private Database db = ...;
  private HashMap cmds = new HashMap();

  public Server() {
    initCommands();
  }

  private void initCommands() {
    cmds.put("new", new NewCommand(db));
    cmds.put("del",
                 new DeleteCommand(db));
    ...
  }

  public void service(String cmd,
                      Object data) {
    ...
    Command c = (Command)cmds.get(cmd);
    ...
    Object result = c.execute(data);
    ...
  }
}
```

```
public interface NewCommand implements Command {

  public NewCommand(Database db) {
    this.db = db;
  }

  public Object execute(Object arg) {
    Data d = (Data)arg;
    int id = d.getArg(0);
    String nome = d.getArg(1);
    db.insert(new Member(id, nome));
  }
}
```

```
public class DeleteCommand implements Command {

  public DeleteCommand(Database db) {
    this.db = db;
  }

  public Object execute(Object arg) {
    Data d = (Data)arg;
    int id = d.getArg(0);
    db.delete(id);
  }
}
```

Padrões de Projeto

Bibliografia

- Use a Cabeça ! Padrões de Projetos (design Patterns) - 2ª Ed. Elisabeth Freeman e Eric Freeman. Editora: Alta Books

- Padroes de Projeto – Soluções reutilizáveis de software orientado a objetos. Erich Gamma, Richard Helm, Ralph Johnson. Editora Bookman