Padrões de Projeto de Software Orientados a Objetos

Tecnologia em Análise e Desenvolvimento de Sistemas

Paulo Mauricio Gonçalves Júnior

Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco

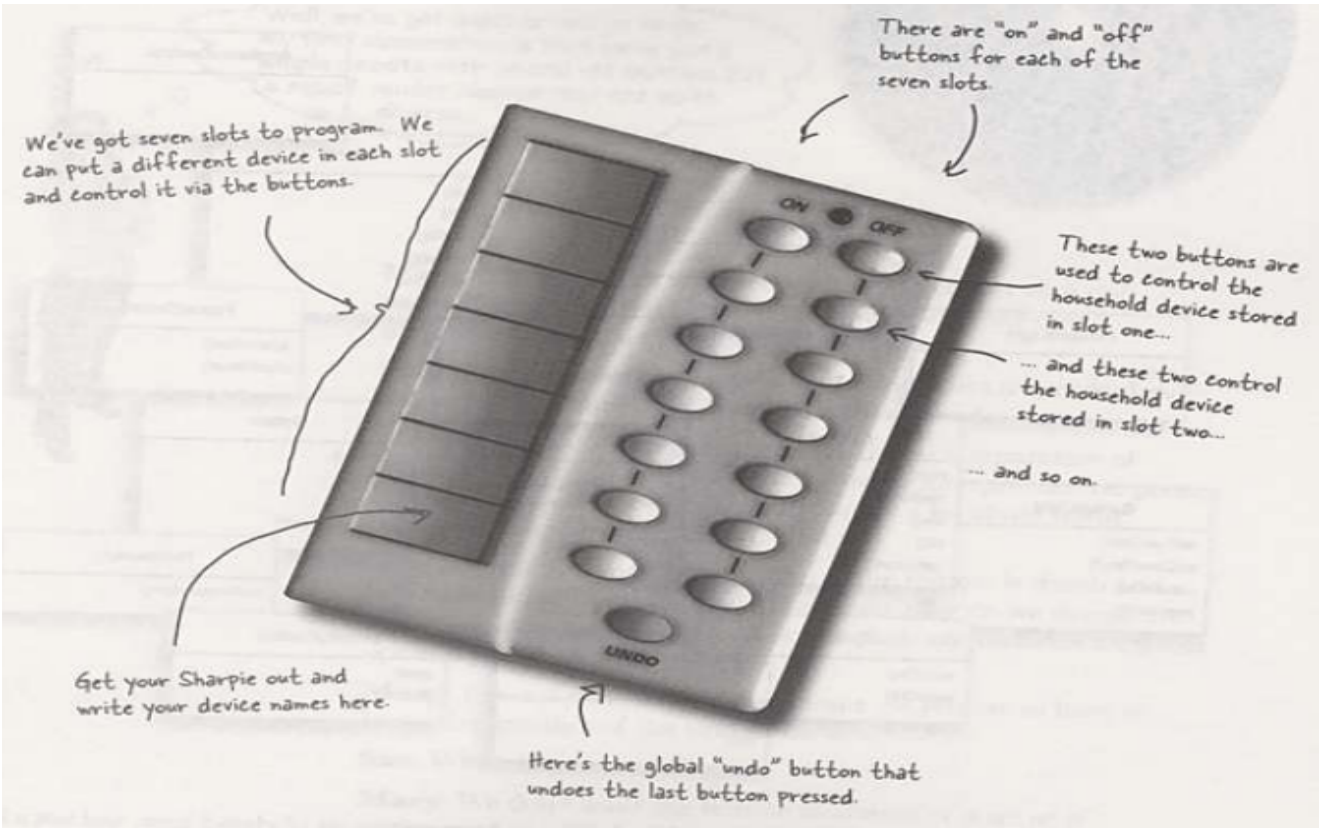
8 de abril de 2018

Parte I

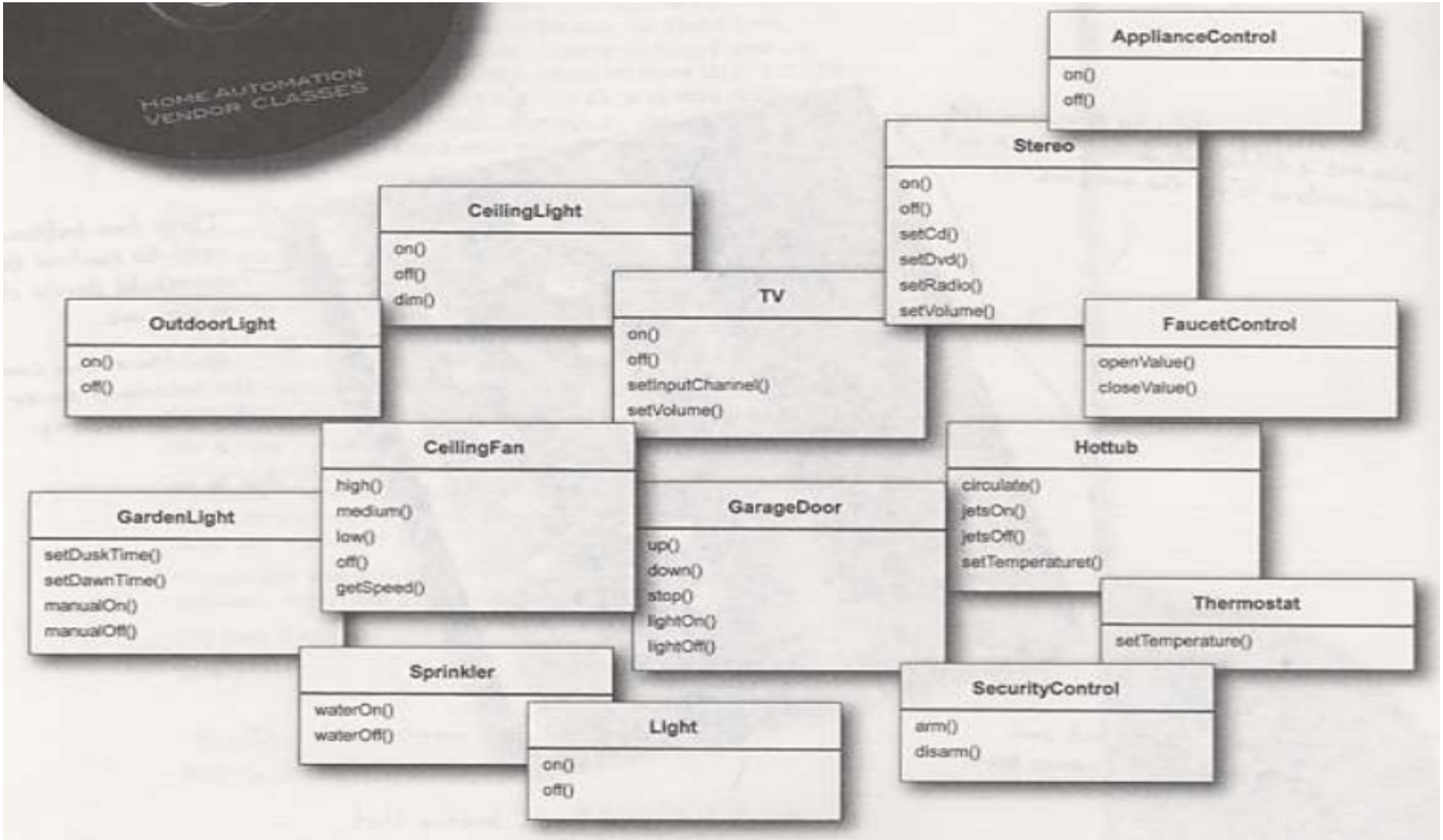
Command

Introdução I

- Vamos implementar um controle remoto que gerencia dispositivos residenciais a partir de classes pré-concebidas.

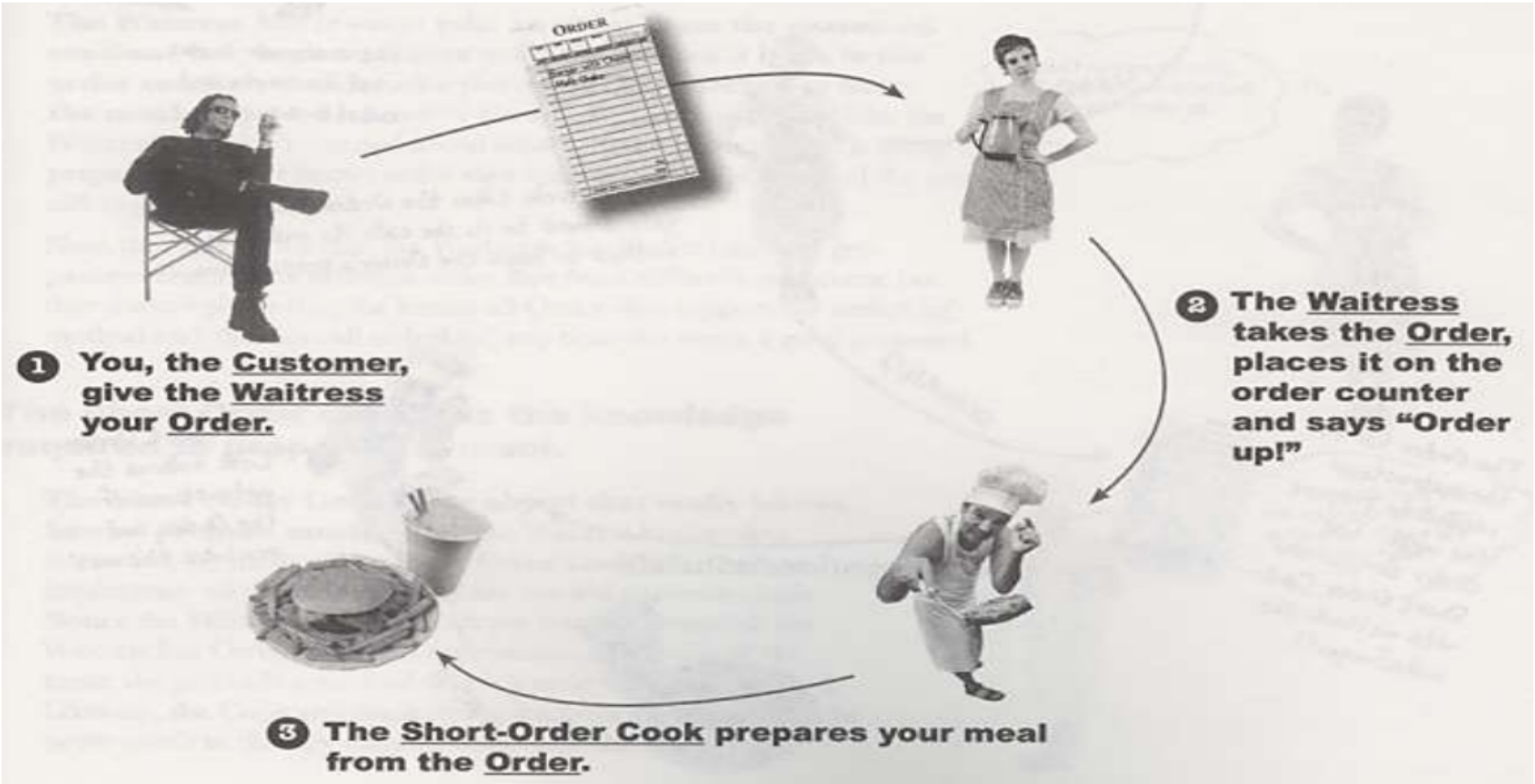


Introdução II

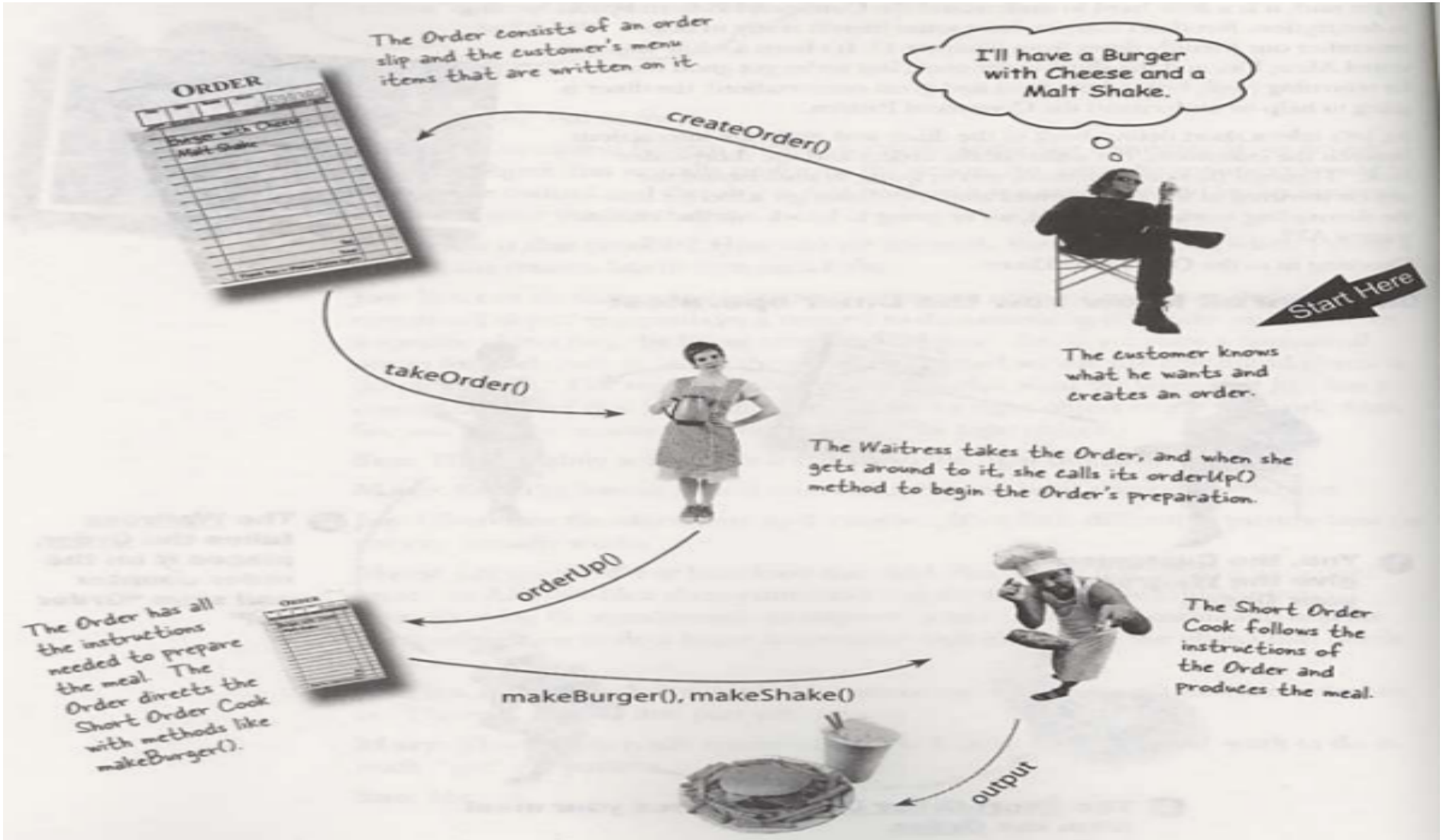


Introdução III

- Como um pedido em um restaurante funciona?



Introdução IV



Introdução V

- O pedido contém as ações a serem realizadas. Ele possui uma referência para o objeto que realiza as ações (o cozinheiro).
- A garçonete não precisa saber sobre o conteúdo nem como preparar o pedido. Ela só precisa indicar ao cozinheiro para preparar o pedido.

Command I

Controle Remoto Simples

```
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;
```

Command II

Controle Remoto Simples

```
  public SimpleRemoteControl() {}

  public void setCommand(Command command) {
    slot = command;
  }

  public void buttonWasPressed() {
    slot.execute();
  }
}

public class RemoteControlTest {
  public static void main(String[] args) {
    SimpleRemoteControl remote = new SimpleRemoteControl();
    Light light = new Light();
    GarageDoor garageDoor = new GarageDoor();
    LightOnCommand lightOn = new LightOnCommand(light);
    GarageDoorOpenCommand garageOpen = new GarageDoorOpenCommand(
        garageDoor);
```

Command III

Controle Remoto Simples

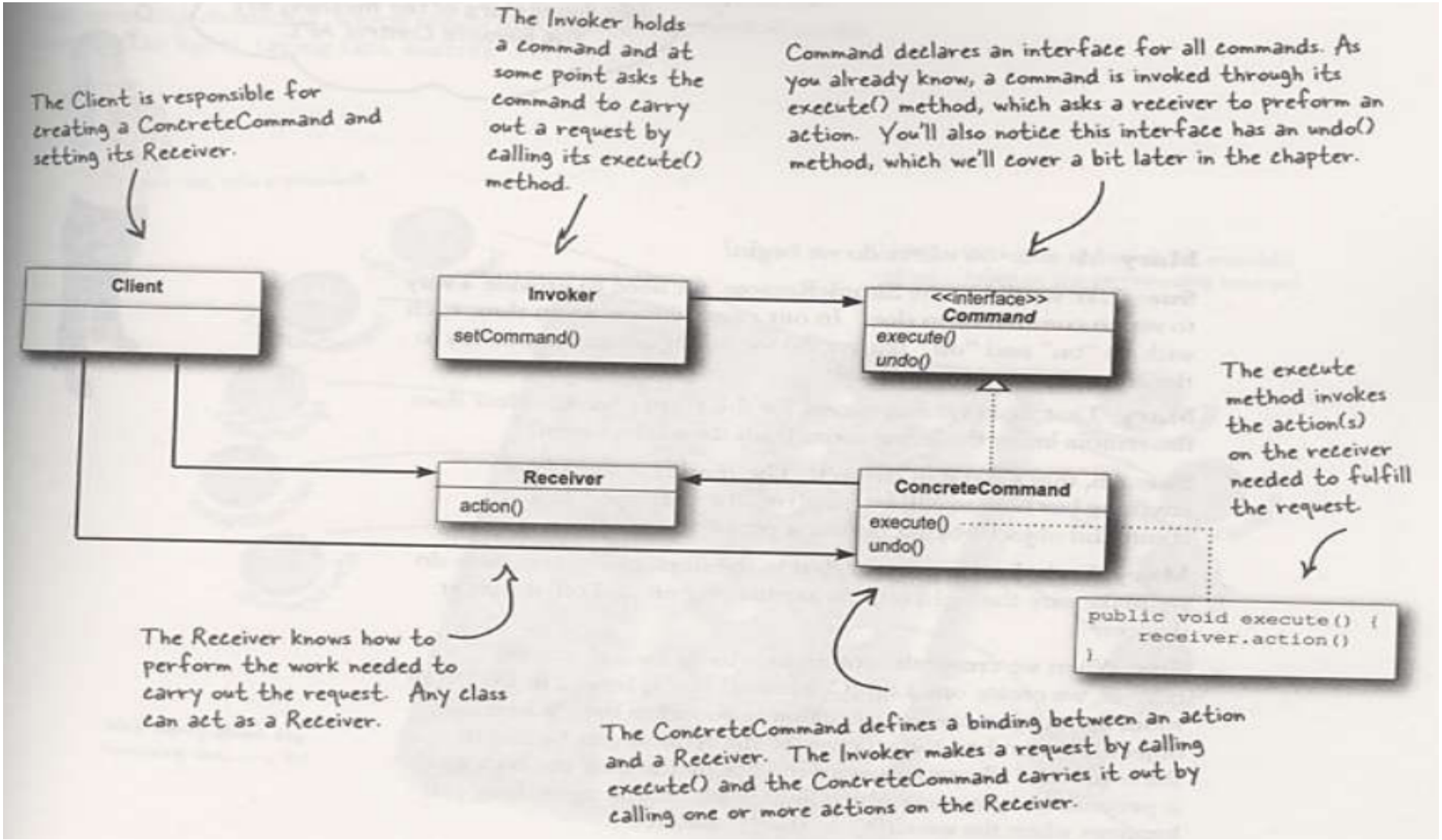
```
        remote.setCommand(lightOn);
        remote.buttonWasPressed();
        remote.setCommand(garageOpen);
        remote.buttonWasPressed();
        }
}
```

Definição

O padrão Command encapsula um pedido como um objeto, permitindo a você para metrizar outros objetos com diferentes pedidos, empilhar ou logar pedidos, e suportando operações de desfazer.

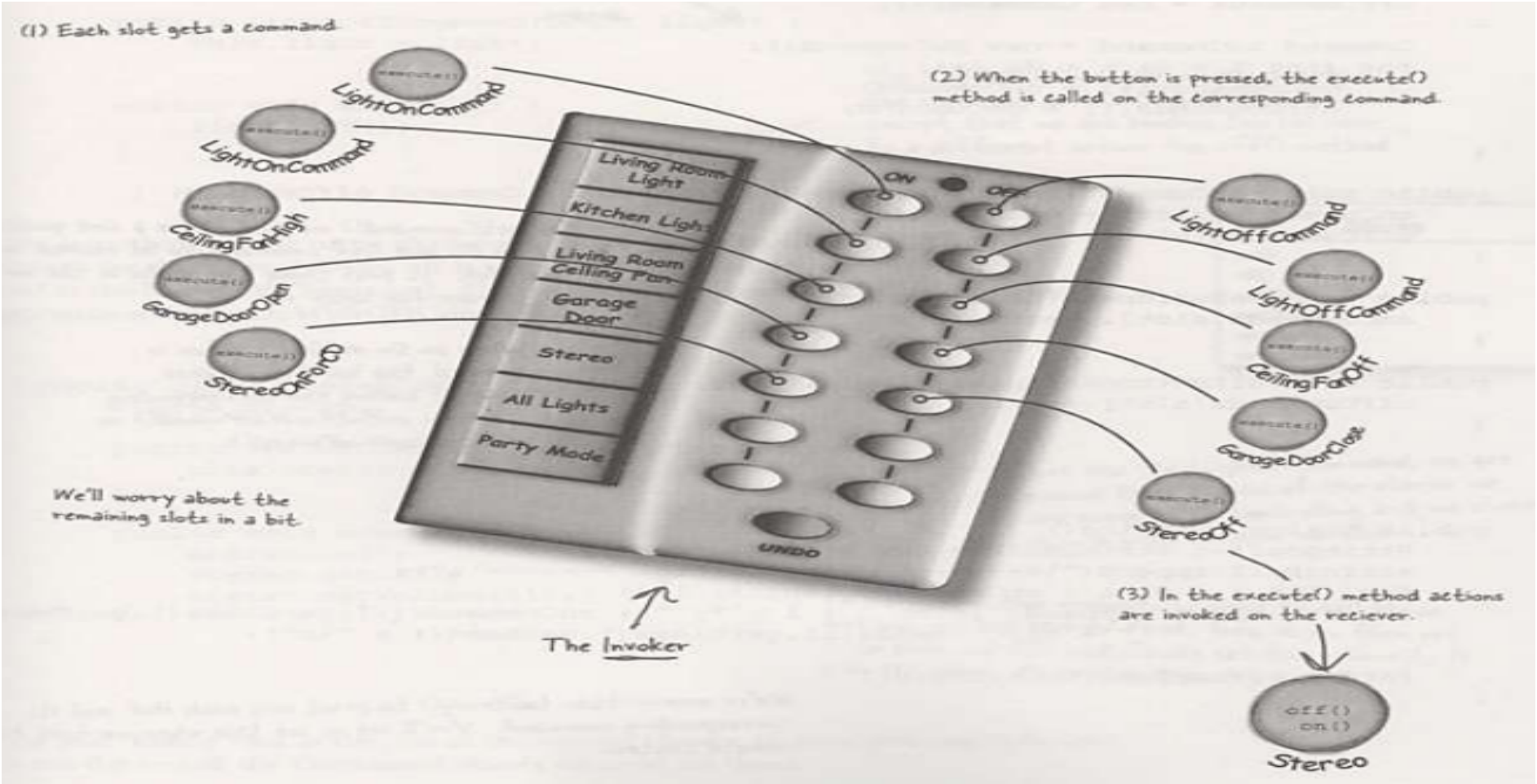
Command IV

Controle Remoto Simples



Command I

Controle Remoto



Command II

Controle Remoto

```
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;
  }
```

Command III

Controle Remoto

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

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

public class RemoteLoader {
  public static void main(String[] args) {
    RemoteControl remoteControl = new RemoteControl();
    Light livingRoomLight = new Light("Living Room");
    Light kitchenLight = new Light("Kitchen");
    CeilingFan ceilingFan= new CeilingFan("Living Room");
    GarageDoor garageDoor = new GarageDoor("");
    Stereo stereo = new Stereo("Living Room");
```

Command IV

Controle Remoto

```
LightOnCommand livingRoomLightOn = new LightOnCommand(
    livingRoomLight);
LightOffCommand livingRoomLightOff = new LightOffCommand(
    livingRoomLight);
LightOnCommand kitchenLightOn = new LightOnCommand(kitchenLight
    );
LightOffCommand kitchenLightOff = new LightOffCommand(
    kitchenLight);

CeilingFanOnCommand ceilingFanOn = new CeilingFanOnCommand(
    ceilingFan);
CeilingFanOffCommand ceilingFanOff = new CeilingFanOffCommand(
    ceilingFan);
GarageDoorUpCommand garageDoorUp = new GarageDoorUpCommand(
    garageDoor);
GarageDoorDownCommand garageDoorDown = new
    GarageDoorDownCommand(garageDoor);

StereoOnWithCDCommand stereoOnWithCD = new
    StereoOnWithCDCommand(stereo);
```

Command V

Controle Remoto

```
        StereoOffCommand  stereoOff = new StereoOffCommand(stereo);

        remoteControl.setCommand(0, livingRoomLightOn,
            livingRoomLightOff);
        remoteControl.setCommand(1, kitchenLightOn, kitchenLightOff);
        remoteControl.setCommand(2, ceilingFanOn, ceilingFanOff);
        remoteControl.setCommand(3, stereoOnWithCD, stereoOff);

        remoteControl.onButtonWasPushed(0);
        remoteControl.offButtonWasPushed(0);
        remoteControl.onButtonWasPushed(1);
        remoteControl.offButtonWasPushed(1);
        remoteControl.onButtonWasPushed(2);
        remoteControl.offButtonWasPushed(2);
        remoteControl.onButtonWasPushed(3);
        remoteControl.offButtonWasPushed(3);
    }
}

public class NoCommand implements Command {
```

Command VI

Controle Remoto

```
    public void execute() { }
}
```

Command I

Controle Remoto com desfazer

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

public class LightOnCommand implements Command {
  Light light;
  int level;
  public LightOnCommand(Light light) {
    this.light = light;
  }

  public void execute() {
        level = light.getLevel();
    light.on();
  }

  public void undo() {
    light.dim(level);
```

Command II

Controle Remoto com desfazer

```
    }
}

public class LightOffCommand implements Command {
    Light light;
    int level;
    public LightOffCommand(Light light) {
        this.light = light;
    }

    public void execute() {
            level = light.getLevel();
        light.off();
    }

    public void undo() {
        light.dim(level);
    }
}
```

Command III

Controle Remoto com desfazer

```
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;
    }
    undoCommand = noCommand;
  }

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

Command IV

Controle Remoto com desfazer

```
    offCommands[slot] = offCommand;
  }

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

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

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

public class RemoteLoader {
```

Command V

Controle Remoto com desfazer

```
public static void main(String[] args) {
    RemoteControlWithUndo remoteControl = new RemoteControlWithUndo
        ();

    Light livingRoomLight = new Light("Living Room");

    LightOnCommand livingRoomLightOn =
        new LightOnCommand(livingRoomLight);
    LightOffCommand livingRoomLightOff =
        new LightOffCommand(livingRoomLight);

    remoteControl.setCommand(0, livingRoomLightOn,
        livingRoomLightOff);

    remoteControl.onButtonWasPushed(0);
    remoteControl.offButtonWasPushed(0);
    System.out.println(remoteControl);
    remoteControl.undoButtonWasPushed();
    remoteControl.offButtonWasPushed(0);
    remoteControl.onButtonWasPushed(0);
```

Command VI

Controle Remoto com desfazer

```
        System.out.println(remoteControl);
        remoteControl.undoButtonWasPushed();
    }
}
```

Command I

Controle Remoto com Macros

```
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();
    }
  }

    /* These commands have to be done backwards to ensure proper
         undo functionality */
  public void undo() {
    for (int i = commands.length -1; i >= 0; i--) {
      commands[i].undo();
    }
```

Command II

Controle Remoto com Macros

```
    }
}

public class RemoteLoader {

    public static void main(String[] args) {

        RemoteControl remoteControl = new RemoteControl();

        Light light = new Light("Living Room");
        TV tv = new TV("Living Room");
        Stereo stereo = new Stereo("Living Room");
        Hottub hottub = new Hottub();

        LightOnCommand lightOn = new LightOnCommand(light);
        StereoOnCommand stereoOn = new StereoOnCommand(stereo);
        TVOnCommand tvOn = new TVOnCommand(tv);
        HottubOnCommand hottubOn = new HottubOnCommand(hottub);
        LightOffCommand lightOff = new LightOffCommand(light);
        StereoOffCommand stereoOff = new StereoOffCommand(stereo);
```

Command III

Controle Remoto com Macros

```
        TVOffCommand tvOff = new TVOffCommand(tv);
        HottubOffCommand hottubOff = new HottubOffCommand(hottub);

        Command[] partyOn = { lightOn, stereoOn, tvOn, hottubOn};
        Command[] partyOff = { lightOff, stereoOff, tvOff, hottubOff};

        MacroCommand partyOnMacro = new MacroCommand(partyOn);
        MacroCommand partyOffMacro = new MacroCommand(partyOff);

        remoteControl.setCommand(0, partyOnMacro, partyOffMacro);

        System.out.println(remoteControl);
        System.out.println("--- Pushing Macro On---");
        remoteControl.onButtonWasPushed(0);
        System.out.println("--- Pushing Macro Off---");
        remoteControl.offButtonWasPushed(0);
    }
}
```

Mais usos I

- Empilhar pedidos: pool de threads.
- Pedidos de log: recuperação de crashes. Adicionar métodos `store()` e `load()`