Design Defects & Restructuring

Week 9: 05 Nov 2022

Rahim Hasnani

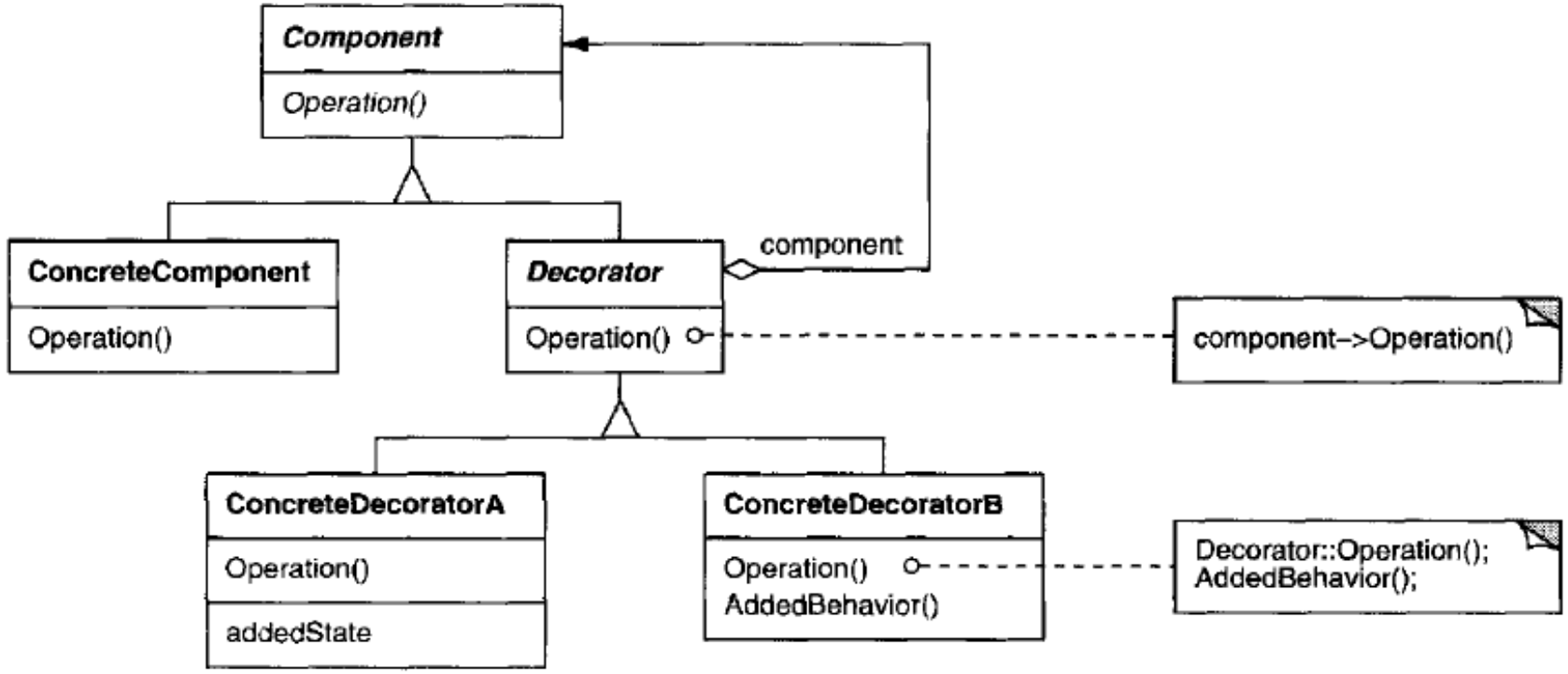
Project

- Decide on a topic NOW
- Make a git-hub repository for the project NOW and make me a member/collaborator (rhasnani@yahoo.com)

Decorator Pattern - Recap

Add responsibility to an individual object and not to an entire class

Why both aggregation and inheritance below?



Read the known uses section

Command Pattern

Encapsulating Method Invocation

Greetings!

I recently received a demo and briefing from Johnny Hurricane, CEO of Weather-O-Rama, on their new expandable weather station. I have to say, I was so impressed with the software architecture that I'd like to ask you to design the API for our new Home Automation Remote Control. In return for your services we'd be happy to handsomely reward you with stock options in Home Automation or Bust, Inc.

I'm enclosing a prototype of our ground-breaking remote control for your perusal. The remote control features seven programmable slots (each can be assigned to a different household device) along with corresponding on/off buttons for each. The remote also has a global undo button.

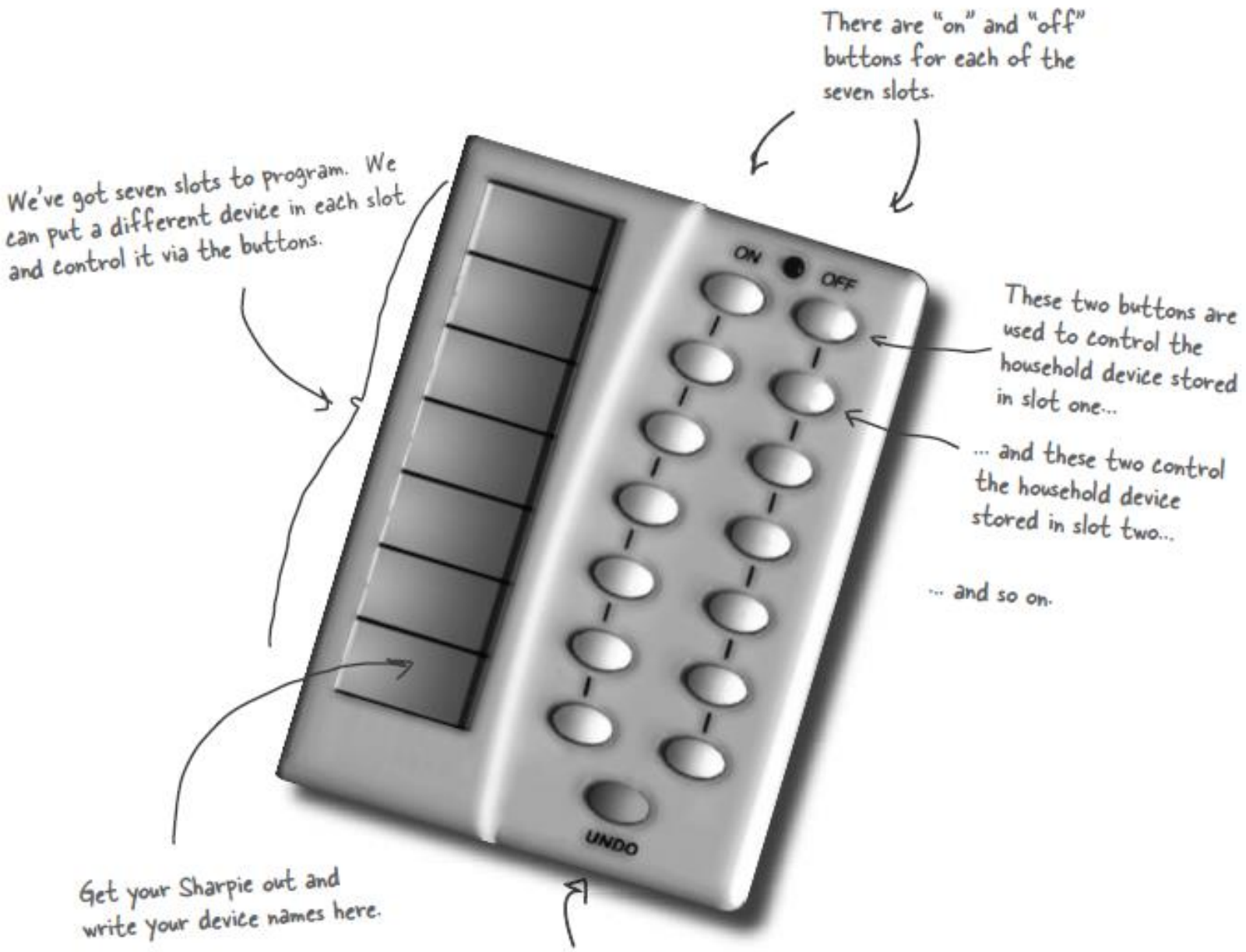
I'm also enclosing a set of Java classes on CD-R that were created by various vendors to control home automation devices such as lights, fans, hot tubs, audio equipment, and other similar controllable appliances.

We'd like you to create an API for programming the remote so that each slot can be assigned to control a device or set of devices. Note that it is important that we be able to control the current devices on the disc, and also any future devices that the vendors may supply.

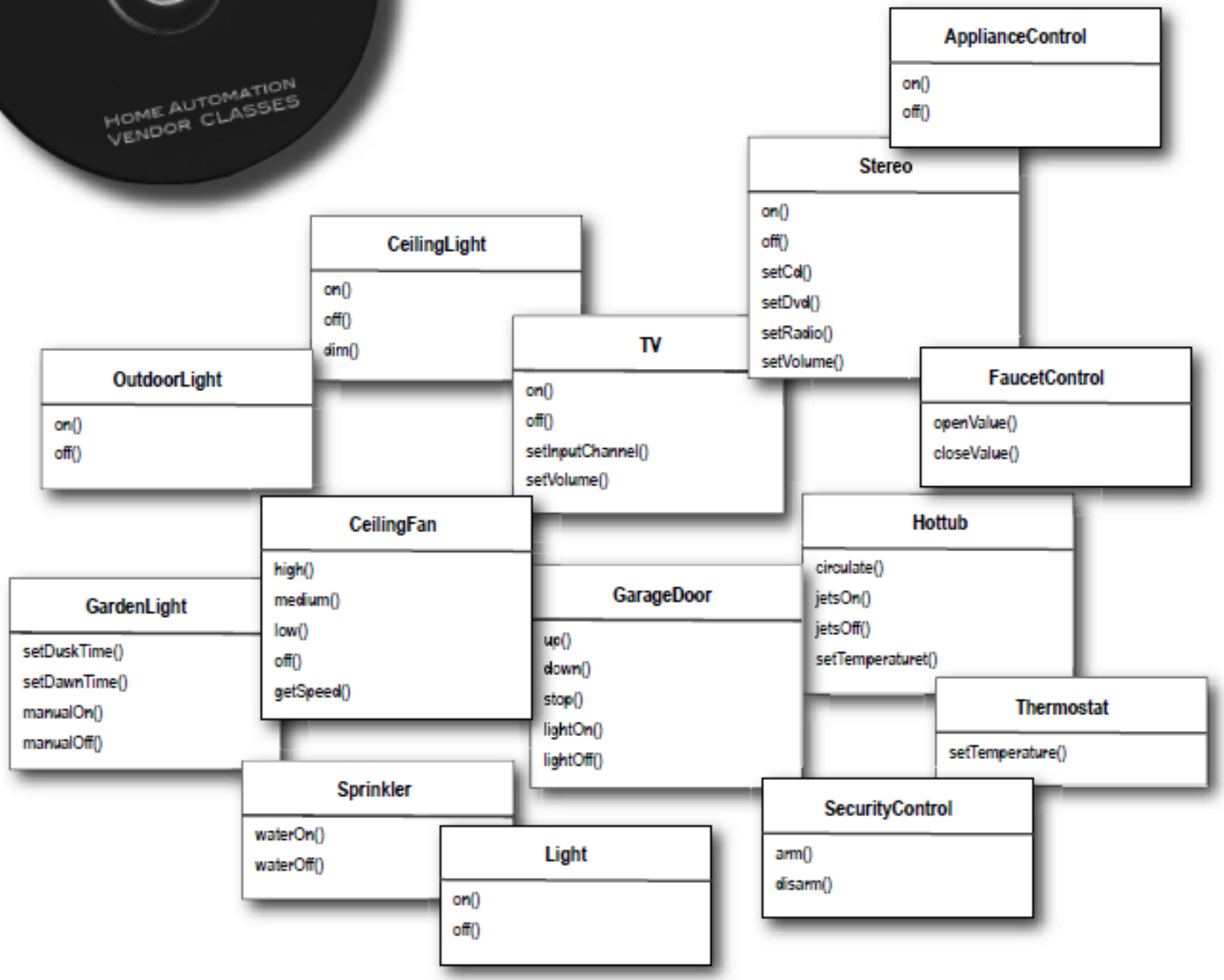
Given the work you did on the Weather-O-Rama weather station, we know you'll do a great job on our remote control!

We look forward to seeing your design.

Sincerely,


There are "on" and "off" buttons for each of the seven slots.
We've got seven slots to program. We can put a different device in each slot and control it via the buttons.
ON
OFF
These two buttons are used to control the household device stored in slot one...
... and these two control the household device stored in slot two...
... and so on.
UNDO
Get your Sharpie out and write your device names here.
Here's the global "undo" button that undoes the last button pressed.

Check out the vendor classes on the CD-R. These should give you some idea of the interfaces of the objects we need to control from the remote.



Command Pattern…

```
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();
    }
}
```

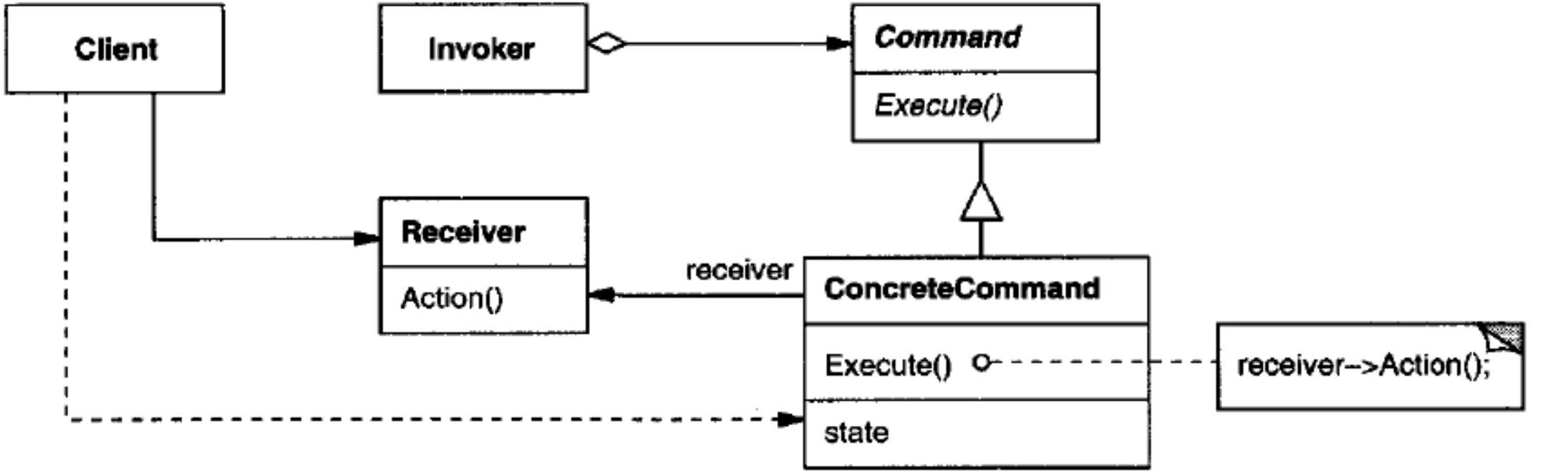
```
public class RemoteControlTest {
    public static void main(String[] args) {
        SimpleRemoteControl remote = new SimpleRemoteControl();
        Light light = new Light();
        LightOnCommand lightOn = new LightOnCommand(light);

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

```
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);


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

Command Pattern - Structure



Adapter Pattern – Let's start from Applicability…

Applicability

Use the Adapter pattern when

- you want to use an existing class, and its interface does not match the one you need.
- you want to create a reusable class that cooperates with unrelated or unforeseen classes, that is, classes that don't necessarily have compatible interfaces.
- *(object adapter only)* you need to use several existing subclasses, but it's impractical to adapt their interface by subclassing every one. An object adapter can adapt the interface of its parent class.

Refactoring Methods: Change Function Declaration

- Change Function Declaration
  - Name of function
  - Name & type of parameters
  - Add/Remove parameters
- Simple mechanics vs Migration mechanics
- Simple Mechanics Example: Rename Function
  - Change the method declaration to the desired declaration.
  - Find all references to the old method declaration, update them to the new one.
  - Test.

```
function circum(radius) {
  return 2 * Math.PI * radius;
}
```

```
function circumference(radius) {
  return 2 * Math.PI * radius;
}
```

Refactoring Methods:

- Migration Mechanics Example: Rename function
  - Use Extract Function on the function body to create the new function.
  - Test.
  - Apply Inline Function to the old functions.
  - Test.

```
function circum(radius) {
  return 2 * Math.PI * radius;
}
```

```
function circum(radius) {
  return circumference(radius);
}
function circumference(radius) {
  return 2 * Math.PI * radius;
}
```