GeeksforGeeks
A computer science portal for geeks

Custom Search

Courses Login

Suggest an Article

Command Pattern

Like previous articles, let us take up a design problem to understand command pattern. Suppose you are building a home automation system. There is a programmable remote which can be used to turn on and off various items in your home like lights, stereo, AC etc. It looks something like this.

You can do it with simple if-else statements like


```
if (buttonPressed == button1)
     lights.on()
```

But we need to keep in mind that turning on some devices like stereo comprises of many steps like setting cd, volume etc. Also we can reassign a button to do something else. By using simple if-else we are coding to implementation rather than interface. Also there is tight coupling.

So what we want to achieve is a design that provides loose coupling and remote control should not have much information about a particular device. The command pattern helps us do that.

Definition: The **command pattern** encapsulates a request as an object, thereby letting us parameterize other objects with different requests, queue or log requests, and support undoable operations.

The definition is a bit confusing at first but let's step through it. In analogy to our problem above remote control is the client and stereo, lights etc. are the receivers. In command pattern there is a Command object that *encapsulates a request* by binding together a set of actions on a specific receiver. It does so by exposing just one method execute() that causes some actions to be invoked on the receiver.

Parameterizing other objects with different requests in our analogy means that the button used to turn on the lights can later be used to turn on stereo or maybe open the garage door.

queue or log requests, and support undoable operations means that Command's Execute operation can store state for reversing its effects in the Command itself. The Command may have an added unExecute operation that reverses the effects of a previous call to execute.It may also support logging changes so that they can be reapplied in case of a system crash.

Below is the Java implementation of above mentioned remote control example:

```
// A simple Java program to demonstrate
// implementation of Command Pattern using
// a remote control example.

// An interface for command
interface Command
{
    public void execute();
}

// Light class and its corresponding command
// classes
class Light
{
```

```
        public void on()
        {
            System.out.println("Light is on");
        }
        public void off()
        {
            System.out.println("Light is off");
        }
    }
    class LightOnCommand implements Command
    {
        Light light;

        // The constructor is passed the light it
        // is going to control.
        public LightOnCommand(Light light)
        {
           this.light = light;
        }
        public void execute()
        {
           light.on();
        }
    }
    class LightOffCommand implements Command
    {
        Light light;
        public LightOffCommand(Light light)
        {
            this.light = light;
        }
        public void execute()
        {
             light.off();
        }
    }

    // Stereo and its command classes
    class Stereo
    {
        public void on()
        {
```

```
            System.out.println("Stereo is on");
        }
        public void off()
        {
            System.out.println("Stereo is off");
        }
        public void setCD()
        {
            System.out.println("Stereo is set " +
                               "for CD input");
        }
        public void setDVD()
        {
            System.out.println("Stereo is set"+
                              " for DVD input");
        }
        public void setRadio()
        {
            System.out.println("Stereo is set" +
                               " for Radio");
        }
        public void setVolume(int volume)
        {
           // code to set the volume
           System.out.println("Stereo volume set"
                              + " to " + volume);
        }
    }
    class StereoOffCommand implements Command
    {
        Stereo stereo;
        public StereoOffCommand(Stereo stereo)
        {
            this.stereo = stereo;
        }
        public void execute()
        {
           stereo.off();
        }
    }
    class StereoOnWithCDCommand implements Command
    {
```

```
        Stereo stereo;
        public StereoOnWithCDCommand(Stereo stereo)
        {
            this.stereo = stereo;
        }
        public void execute()
        {
            stereo.on();
            stereo.setCD();
            stereo.setVolume(11);
        }
    }

    // A Simple remote control with one button
    class SimpleRemoteControl
    {
        Command slot;  // only one button

        public SimpleRemoteControl()
        {
        }

        public void setCommand(Command command)
        {
            // set the command the remote will
            // execute
            slot = command;
        }

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

    // Driver class
    class RemoteControlTest
    {
        public static void main(String[] args)
        {
            SimpleRemoteControl remote =
                      new SimpleRemoteControl();
```

```
        Light light = new Light();
        Stereo stereo = new Stereo();

        // we can change command dynamically
        remote.setCommand(new
                    LightOnCommand(light));
        remote.buttonWasPressed();
        remote.setCommand(new
                StereoOnWithCDCommand(stereo));
        remote.buttonWasPressed();
        remote.setCommand(new
                   StereoOffCommand(stereo));
        remote.buttonWasPressed();
    }
}
```

Output:

```
Light is on
Stereo is on
Stereo is set for CD input
Stereo volume set to 11
Stereo is off
```

Notice that the remote control doesn't know anything about turning on the stereo. That information is contained in a separate command object. This reduces the coupling between them.

Advantages:

- Makes our code extensible as we can add new commands without changing existing code.
- Reduces coupling the invoker and receiver of a command.

Disadvantages:

- Increase in the number of classes for each individual command

References:

- Head First Design Patterns (book)
- https://github.com/bethrobson/Head-First-Design-Patterns/tree/master/src/headfirst/designpatterns/command

If This article is contributed by **Sulabh Kumar**. you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Recommended Posts:

Observer Pattern | Set 1 (Introduction)

Observer Pattern | Set 2 (Implementation)

Singleton Design Pattern | Implementation

Decorator Pattern | Set 1 (Background)

The Decorator Pattern | Set 2 (Introduction and Design)

Decorator Pattern | Set 3 (Coding the Design)

Strategy Pattern | Set 1 (Introduction)

Strategy Pattern | Set 2 (Implementation)

Adapter Pattern

Iterator Pattern

Curiously recurring template pattern (CRTP)

Flyweight Design Pattern

Implementing Iterator pattern of a single Linked List

Singleton Design Pattern | Introduction

Java Singleton Design Pattern Practices with Examples

Article Tags : Design Pattern

2

To-do Done

2.6

Based on **10** vote(s)

Feedback/ Suggest Improvement | Add Notes | Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments | Share this post!

A computer science portal for geeks

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org

COMPANY

About Us
Careers
Privacy Policy
Contact Us

LEARN

Algorithms
Data Structures
Languages
CS Subjects
Video Tutorials

PRACTICE

Company-wise
Topic-wise
Contests
Subjective Questions

CONTRIBUTE

Write an Article
Write Interview Experience
Internships
Videos

@geeksforgeeks, Some rights reserved