

Command Design Pattern

Command design pattern is used to encapsulate a request as an object and pass to an invoker, wherein the invoker does not know how to service the request but uses the encapsulated command to perform an action.

To understand command design pattern we should understand the associated key terms like client, command, command implementation, invoker, receiver.

- Command is an interface with execute method. It is the core of contract.
- A client creates an instance of a command implementation and associates it with a receiver.
- An invoker instructs the command to perform an action.
- A Command implementation's instance creates a binding between the receiver and an action.
- Receiver is the object that knows the actual steps to perform the action.

Command Pattern Discussion

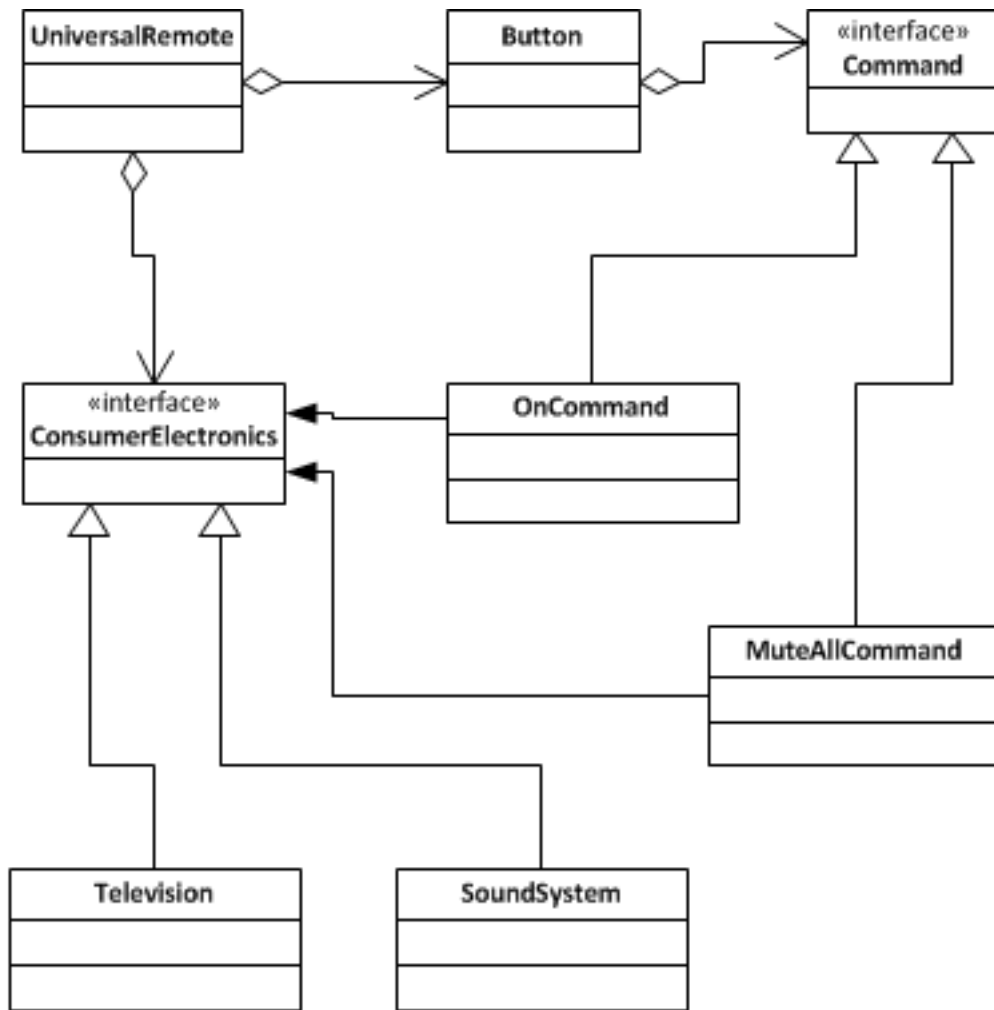
It will be easier to understand all the above with an example. Let us think of developing a universal remote. Our UniversalRemote will have only two buttons, one is to power on and another is to mute all associated ConsumerElectronics.

For ConsumerElectronics we will have couple of implementations Television and SoundSystem. Button is a class that is the invoker of an action.

OnCommand is used to switch on a ConsumerElectronics. While instantaiting OnCommand client needs to set the receiver. Here receiver is either Television or SoundSystem. UniversalRemote class will maintain all the receivers and will provide a method to identify the currently active receiver.

When an instance of a invoker is created, a concrete command is passed to it. Invoker does not know how to perform an action. All it does is honoring the agreement with the Command. Invoker calls the execute() method of the set concrete command.

UML Diagram for Command Pattern



Command Pattern Example Implementation

Command.java

```
package com.javapapers.designpattern.command;

public interface Command {
```

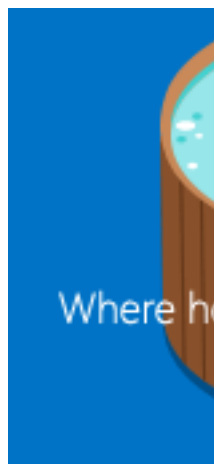
```
public abstract void execute();  
}
```

OnCommand.java

```
package com.javapapers.designpattern.command;  
  
public class OnCommand implements Command {  
  
    private ConsumerElectronics ce;  
  
    public OnCommand(ConsumerElectronics ce) {  
        this.ce = ce;  
    }  
  
    public void execute() {  
        ce.on();  
    }  
}
```

Ads by Google





MuteAllCommand.java

```
package com.javapapers.designpattern.command;

import java.util.List;

public class MuteAllCommand implements Command {
    List ceList;

    public MuteAllCommand(List ceList) {
        this.ceList = ceList;
    }

    @Override
    public void execute() {
```

```
        for (ConsumerElectronics ce : ceList) {  
            ce.mute();  
        }  
    }  
}
```

ConsumerElectronics.java

```
package com.javapapers.designpattern.command;  
  
public interface ConsumerElectronics {  
    public abstract void on();  
    public abstract void mute();  
}
```

Television.java

```
package com.javapapers.designpattern.command;  
  
public class Television implements ConsumerElectronics {  
  
    public void on() {  
        System.out.println("Television is on!");  
    }  
}
```

```
@Override
public void mute() {
    System.out.println("Television is muted!");
}
}
```

SoundSystem.java

```
package com.javapapers.designpattern.command;

public class SoundSystem implements ConsumerElectronics {

    public void on() {
        System.out.println("Sound system is on!");
    }

    @Override
    public void mute() {
        System.out.println("Sound system is muted!");
    }
}
```

Button.java

```
package com.javapapers.designpattern.command;

public class Button {
    Command c;

    public Button(Command c) {
        this.c = c;
    }

    public void click() {
        c.execute();
    }
}
```

UniversalRemote.java

```
package com.javapapers.designpattern.command;

public class UniversalRemote {

    public static ConsumerElectronics getActiveDevice() {
        // here we will have a complex electronic circuit :- )
        // that will maintain current device
        Television tv = new Television();
        return tv;
    }
}
```



```
}
```

DemoCommandPattern.java

```
package com.javapapers.designpattern.command;

import java.util.ArrayList;
import java.util.List;

public class DemoCommandPattern {
    public static void main(String args[]) {

        // OnCommand is instantiated based on active device supplied
        by Remote

        ConsumerElectronics ce = UniversalRemote.getActiveDevice();
        OnCommand onCommand = new OnCommand(ce);
        Button onButton = new Button(onCommand);
        onButton.click();

        Television tv = new Television();
        SoundSystem ss = new SoundSystem();
        List all = new ArrayList();
        all.add(tv);
        all.add(ss);
        MuteAllCommand muteAll = new MuteAllCommand(all);
        Button muteAllButton = new Button(muteAll);
        muteAllButton.click();
    }
}
```

```
}  
}
```

Download Command Pattern Java Source Code

Important Points on Command Pattern

- Command pattern helps to decouple the invoker and the receiver. Receiver is the one which knows how to perform an action.
- Command helps to implement call back in java.
- Helps in terms of extensibility as we can add new command without changing existing code.
- Command defines the binding between receiver and action.
- A command should be able to implement undo and redo operations. That is restoring the state of the receiver. It can be done from the support of receiver.

Command Pattern in JDK

Implementations of `java.lang.Runnable` and `javax.swing.Action` follows command design pattern.

This Behavioral Design Pattern tutorial was added on 02/09/2012.

App and Game Source

Code

 chupamobile.com

Buy and sell Mobile App Start
Making Money in No Time!



« [Top 10 Java Debugging Tips with Eclipse](#)

[Get User Input in Android](#) »

Comments on “Command Design Pattern”

Archana

03/09/2012 at 12:50 pm

Nice Explanation Please explain MVP Design Pattern too..

Reply

ernie

18/11/2013 at 3:38 am

Isn't MVC pattern part of the observer design pattern?

Reply

Satheesh.C

03/09/2012 at 12:54 pm

nice

Reply

Ragu

03/09/2012 at 1:24 pm

Interesting explanation. Please provide your way explanation to Session Facade. I have read in many articles but not satisfied.

Reply

Praveen Kumar Jayaram

03/09/2012 at 2:00 pm

Nice explanation.

Reply

Satyprasad

04/09/2012 at 10:18 am

Good effort. I appreciate the way you present the content. Well organized.

In my view this article is missing one point that is, its Real Time usage in Projects along with general problems we face while implement it. If it was included in it, then it would be perfect.

I would be more happy to look forward such articles with Real Time situations (when we will choose this pattern) along with general problems.

Reply

Joe

04/09/2012 at 9:43 pm

@Satyprasad,

Thanks.

Sure, your point noted.

Reply

Animesh Das

06/09/2012 at 9:52 am

Nice Description..

Reply

Dinesh

08/09/2012 at 8:57 am

very nice, but does it require to keep the instance variable of the commands same? So that DemoCommandPattern need not write code specific to a command. Please correct me if I am wrong.

Reply

Sharil

13/09/2012 at 1:07 pm

Can you explain how Struts Controller(ActionServlet, RequestProcessor) uses the Command Design Pattern

Reply

pradyut sharma

22/09/2012 at 11:36 am

very good description

Reply

R Manoj

26/09/2012 at 12:43 pm

Thanks a lot, Joseph. You are real guru helping ppl in a most lucid way.

Reply

Amit

30/09/2012 at 2:27 am

Great article!!

One doubt..Don't you think that Button should be part of IniversalRemote class?

Reply

Anonymous

16/10/2012 at 9:01 am

Your site has really been a great help.. thanks ...

Reply

sahil

27/10/2012 at 2:39 pm

how we can create?

Reply

Anonymous

28/10/2012 at 5:06 am

Your Article are too much Abstract !

Reply

Kasun

01/11/2012 at 12:10 pm

I think in the diagram UniversalRemote does not need to have an association for Button.
right?

Rest is excellent. Thanks for sharing

Reply

Niranjana

03/11/2012 at 9:01 pm

Nice Article

Reply

chandan kumar

04/11/2012 at 3:00 am

hi,

all

I want to maintain the state of checkboxes across pages.

What happens is when i check one checkbox and then go to say 2nd page then check one checkbox there and when i return to previous page the checked box is again unchecked.

Means the status of the checkbox is not maintained.If it is maintained then it should show checked.

I am using jsps and display tag in jsp for pagination.
Any kind of help would be greatly appreciated.

please help me.....

Reply

Anonymous

18/11/2012 at 7:30 pm

Chandan,

You need pass on flag which pass on the state of the check box when you navigate from one page to another page and vise versa if there are any change in the check box status then flag has to be marked accordingly .Hope this clue will help you

Reply

Anonymous

12/01/2013 at 10:55 pm

use a filter.and one map which will keep stock of pages in terms of mapping or path info and check box status.

if any changes happen modify the entryset of map retrieving it from session.

instantiate the map on session listener's activate session.and put that in session.

Reply

Gurpreet Singh

18/11/2012 at 7:58 pm

thanks for notes

please sir tell me about the j components

Reply

lakshmi

30/11/2012 at 11:55 am

thanks u sir for this note

Reply

heispa1@gmail.com

07/12/2012 at 11:39 am

Hi Joe,

Your articals are really just awesome.

I really am able to call back Einsteins Quote:"Everything should be made as simple as possible, but no simpler."

The way you explain was really cool.

I am still in between reading your blogs on design patterns..

Can u please give a clarification on how to create dynamic objects along with new instace of a class?

If it is explained in any of the design patterns please ping me that link..i'll go through it.

What am expecting is something like this.

```
for(;;){  
Pavan p=new Pavan();//Pavan is a class  
}
```

At the end what i want to acheive is to find the number of instances of that particular class.

Thank you,

Pavan Karasala

Reply

Is there a way to find objects present in one JVM from another JVM?

Via RMI/JMS/Web service we need to either send the object from other vm or something else...but we cannot jst browse through the objects in another JVM..Please correct me about what i am trying to saying is meaningful...

Regards,
Pavan Karasala

Reply

arvind

17/12/2012 at 7:04 pm

Nice article....

Reply

shilpa

18/12/2012 at 3:35 pm

Nice Blog

Reply

Mahendra

17/01/2013 at 12:31 pm

really learning a lot from you

Reply

Samuel

18/01/2013 at 10:48 pm

Hello Joe,How to read any number of lines from the console in java?

It is easy in c/c++

Regards

Samuel

Reply

vikrant

19/01/2013 at 7:35 pm

The article was good, but it keeps some questions unanswered like,

Why do we really create this complex structure ? When UniversalRemote already has access to ConsumerElectronics, why not directly invoke ConsumerElectronics.on() and ConsumerElectronics.muteAll() methods, when the appropriate button is clicked, what is the advantage of creating the intermediate classes like Command ?

Reply

Rakesh Vende

29/01/2013 at 11:38 am

If aim is to keep the receiver and invoker of command decoupled from each other, intermediary interface is needed.

Most of the design pattern based on this design principal. "Classes should be open for extension, but closed for modification".

If I have to add ConsumerElectricals to be switched on from the same Button class, then the scenario that have been suggested by you ask me to modify the existing button implementation which violates above rule. However with the Command patters design would be like this.

```
interface ConsumerElectrical{
```

```
    public void switchOn();
```

```
}
```

```
class CeilingFan implements ConsumerElectricals{
```

```
public void switchOn{
```

```
System.out.println("Fan is switched on");
```

```
}
```

```
}
```

```
class ElectricalsSwitchOnCommand implements Command{
```

```
ConsumerElectrical CE;
```

```
ElectricalsSwitchOnCommand(ConsumerElectrical CE){
```

```
this.CE = CE;
```

```
}
```

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

```
CE.swithcon();
```

```
}
```

```
}
```

```
Public class demo{
```



```
public static void main(String [] args){  
  
    ConsumerElectronics ce = UniversalRemote().getActiveDevice();  
  
    Command C = new onCOMmand(cs);  
    Button b = new Button(c);  
    b.click();  
  
    ConsumerElectricals cel = getActiveDevice();  
  
    Command cl = new ElectricalsSwitchOnCommand();  
    Button b = newbutton(cl)  
    b.click();  
  
    }  
}
```

Reply

Arun Abraham

30/01/2013 at 4:14 pm

Your tutorials are just amazing. Relying on your tutorials rather than my text book :)

Reply

Joe

02/02/2013 at 9:48 pm

Welcom Arun. Enjoy :-)

Reply

Sadhu Bala Bhaskar

21/02/2013 at 12:41 am

Hi Joe,

This article is really very useful for the beginners in Java. I really appreciate the effort you put in explaining in more diagrammatic way.

I got a question... why `@Override` annotation is used only for one method (`mute()`).

Is it a typo or any other reason ?

```
public class Television implements ConsumerElectronics {
```

```
public void on() {  
    System.out.println("Sound system is on!");  
}  
  
@Override  
public void mute() {  
    System.out.println("Sound system is muted!");  
}
```

Thanks & Regards,
Sadhu

Reply

Nilesh

28/05/2013 at 3:52 pm

yes it should be as both method are from
ConsumerElectronics interface.

```
public class Television implements ConsumerElectronics {
```

```
@Override
```

```
public void on() {  
    System.out.println("Sound system is on!");  
}  
  
@Override  
public void mute() {  
    System.out.println("Sound system is muted!");  
}  
  
Reply
```

Anonymous

09/07/2013 at 6:57 am

nice example... easy to understand.. thanks

Reply

Sriram

12/07/2013 at 11:21 pm

Wonderful articles Joe!

Rakesh added extra value with his comments giving some coded explanation.

Both your work is appreciated

Joe, you deserve a Hip, Hip, Hurray!

Reply

vinay

10/09/2013 at 10:33 am

By default all the methods in an interface is abstract. Then why in command.java you used

```
public void abstract execute();
```

?

Reply

vinay

10/09/2013 at 10:34 am

sorry its

```
public abstract void execute();
```

Reply

Triguna

20/12/2013 at 4:49 pm

This pattern is too complex or confusing... The examples given here are very nice and could understand but I was not able to correlate exactly on how it is implemented in JDK – Runnable and Action implementations.

Reply

fabian kessler

23/02/2014 at 6:31 am

Here's an open source command framework for Java: <https://github.com/optimaize/command4j> (disclaimer: I'm co-author)

Reply

Ankur

30/03/2014 at 4:04 pm

Very Nice description Joe. Your site helps me a lot.

Reply

Zaheer

29/05/2014 at 2:29 pm

nice complex circuit... :-)

Reply

anup junagade

11/06/2014 at 6:23 pm

all your articles for design pattern are very good and useful. I really appreciate your good work .Thanks a lot for your help

Reply

Dinoop p

19/07/2014 at 11:00 pm

Dear Joe, please include some the use cases of this pattern in real programming problems.

Reply

Your Comment

Name

Email

Post Comment

TUTORIAL MENU



Java



Android



Design Patterns



Creational Design Patterns

 Structural Design Patterns

 Behavioral Design Patterns

 Chain Of Responsibility Design Pattern

 **Command Design Pattern**

 Interpreter Design Pattern

 Iterator Design Pattern

 Mediator Design Pattern

 Memento Design Pattern

 Observer Design Pattern

 State Design Pattern

 Template Method Design Pattern


 Spring

 Web Services

 Servlet




 Java

 Android

 Design Patterns

 Spring

 Web Services

Site Map

© 2008-2014 *javapapers.com*. The design and content are copyrighted to Joe and may not be reproduced in any form.