Proxy Design Pattern

APRIL 2, 2018 BY PANKAJ — 11 COMMENTS

Proxy Design pattern is one of the Structural design pattern and in my opinion one of the simplest pattern to understand.

Table of Contents [hide]

Proxy Design Pattern

Proxy design pattern intent according to GoF is:

Provide a surrogate or placeholder for another object to control access to it.

The definition itself is very clear and proxy design pattern is used when we want to provide controlled access of a functionality.

Let's say we have a class that can run some command on the system. Now if we are using it, its fine but if we want to give this program to a client application, it can have severe issues because client program can issue command to delete some system files or change some settings that you don't want.

Here a proxy class can be created to provide controlled access of the program.

Proxy Design Pattern – Main Class

Since we code Java in terms of interfaces, here is our interface and its implementation class.

`CommandExecutor.java`

```java
package com.journaldev.design.proxy;

public interface CommandExecutor {

	public void runCommand(String cmd) throws Exception;
}
```

`CommandExecutorImpl.java`

```
package com.journaldev.design.proxy;

import java.io.IOException;

public class CommandExecutorImpl implements CommandExecutor {

        @Override
        public void runCommand(String cmd) throws IOException {
                //some heavy implementation
                Runtime.getRuntime().exec(cmd);
                System.out.println("'" + cmd + "' command executed.");
        }

}
```

Proxy Design Pattern – Proxy Class

Now we want to provide only admin users to have full access of above class, if the user is not admin then only limited commands will be allowed. Here is our very simple proxy class implementation.

`CommandExecutorProxy.java`

```
package com.journaldev.design.proxy;

public class CommandExecutorProxy implements CommandExecutor {

        private boolean isAdmin;
        private CommandExecutor executor;

        public CommandExecutorProxy(String user, String pwd){
                if("Pankaj".equals(user) && "J@urnalD$v".equals(pwd)) isAdmin=true;
                executor = new CommandExecutorImpl();
        }

        @Override
        public void runCommand(String cmd) throws Exception {
                if(isAdmin){
                        executor.runCommand(cmd);
                }else{
                        if(cmd.trim().startsWith("rm")){
                                throw new Exception("rm command is not allowed for
non-admin users.");
                        }else{
                                executor.runCommand(cmd);
```

Proxy Design Pattern Client Program

`ProxyPatternTest.java`

```
package com.journaldev.design.test;

import com.journaldev.design.proxy.CommandExecutor;
import com.journaldev.design.proxy.CommandExecutorProxy;

public class ProxyPatternTest {

        public static void main(String[] args){
                CommandExecutor executor = new CommandExecutorProxy("Pankaj",
"wrong_pwd");
                try {
                        executor.runCommand("ls -ltr");
                        executor.runCommand(" rm -rf abc.pdf");
                } catch (Exception e) {
                        System.out.println("Exception Message::"+e.getMessage());
                }

        }

}
```

Output of above proxy design pattern example program is:

```
'ls -ltr' command executed.
Exception Message::rm command is not allowed for non-admin users.
```

Proxy design pattern common uses are to control access or to provide a wrapper implementation for better performance.

Java RMI package uses proxy pattern. That's all for proxy design pattern in java.

« PREVIOUS
Flyweight Design Pattern in Java

NEXT »
Chain of Responsibility Design Pattern in Java

About Pankaj
If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: DESIGN PATTERNS

Comments

Yogesh says
MARCH 27, 2017 AT 7:42 AM
Client application can directly create instance of CommandExecutorImpl class and invoke runCommand.

How this can be avoided?

Reply

DEBARATI MAJUMDER says

AUGUST 17, 2017 AT 7:44 PM

Are you talking about overriding? We can declare the class or method as final to avoid overriding.

Reply

Nina says

AUGUST 22, 2017 AT 8:18 AM

Hi, I have the same doubt as Yogesh. CommandExecutorImpl can not be overriden by declaring it to be final, but how to avoid directly creating instances of it by new CommandExecutorImpl(); ? Thanks.

Reply

Pankaj says

AUGUST 22, 2017 AT 10:49 PM

Note that it's hidden from Client side. In normal scenarios, you won't even know how the executor is running the commands.

Reply

Shubham says

JULY 4, 2018 AT 6:58 PM

Make the constructor private. In this case, the client won't be able to get an instance of IMPL class.

Reply

Nimesh Nuwanjana says

NOVEMBER 19, 2017 AT 12:17 AM

Since both CommandExecutorImpl and CommandExecutorProxy are in the same package can't we make CommandExecutorImpl final and use a protected constructor? So the clients won't be able to create instance and still the CommandExecutorProxy could.

Reply

Vijay Raj R says
MARCH 2, 2017 AT 9:04 PM
Please explain all the 4 types of proxy pattern.
Reply

prasad says
JULY 9, 2016 AT 6:28 AM
there is a chance of throwing NUllPointer Exception in the above program in case of the user is not an admin and the command he is issuing is not containing rm……………
Reply

Paul says
JANUARY 24, 2017 AT 3:14 AM
Prasad I think you are wrong, there are no braces around the "if("Pankaj".equals(user) && "J@urnalD$v".equals(pwd))" and this is legal if the if statement only has a songle line.
So the command executor object is created every time, and the default for the boolean value is false. So you are incorrect.
Reply

aneesh says
SEPTEMBER 15, 2015 AT 1:37 PM
awsem one ….clearly understood the proxy pattern
Reply

Amit says
JULY 14, 2013 AT 9:56 AM
Thanks, great tutorial. I will use it in future to provide restricted access.
Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

☐

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...

DOWNLOAD ANDROID APP

GET IT ON Google Play

DESIGN PATTERNS TUTORIAL

Structual Design Patterns

- Adapter
- Composite
- Proxy
- Flyweight
- Facade
- Bridge
- Decorator

Behavioral Design Patterns

- Template Method
- Mediator
- Chain of Responsibility
- Observer
- Strategy
- Command
- State
- Visitor
- Interpreter
- Iterator
- Memento

Miscellaneous Design Patterns

- Dependency Injection
- Thread Safety in Java Singleton

RECOMMENDED TUTORIALS

Java Tutorials

- Java IO
- Java Regular Expressions
- Multithreading in Java
- Java Logging
- Java Annotations
- Java XML
- Collections in Java
- Java Generics
- Exception Handling in Java
- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial

- Apache Axis 2
- JAX-RS
- Memcached Tutorial

Java Course

Learn Programming Online

Free Programming Tutorials

10 Easy Baby Quilt Patterns

Free Quilt Pattern To Print

Patterns For Baby Quilts

Downloadable Quilt Patterns

Free Toy Knitting Patterns