Facade Design Pattern in Java

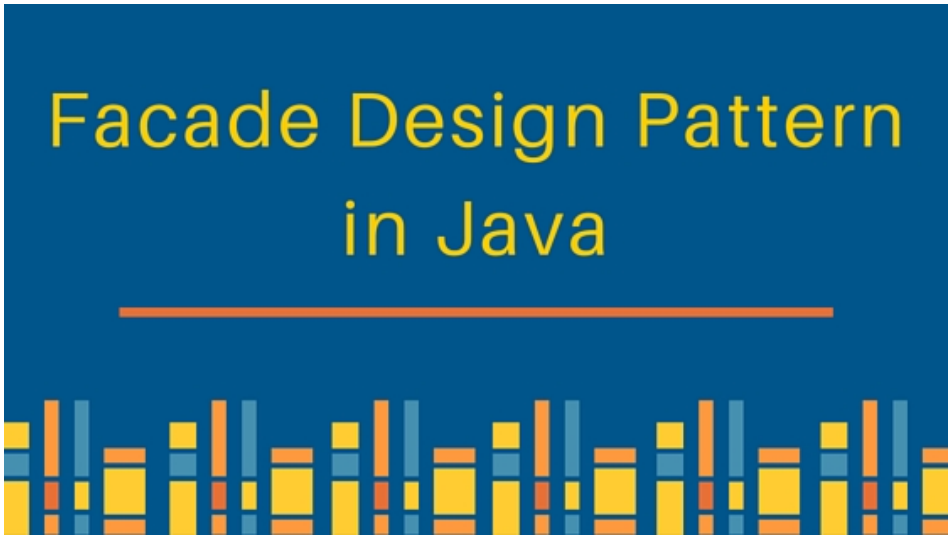
APRIL 2, 2018 BY PANKAJ — 15 COMMENTS

Facade Design Pattern is one of the **Structural design patterns** (such as Adapter pattern and Decorator pattern). Facade design pattern is used to help client applications to easily interact with the system.

Table of Contents [hide]

1 Facade Design Pattern
- 1.1 Facade Design Pattern – Set of Interfaces
- 1.2 Facade Design Pattern Interface
- 1.3 Facade Design Pattern Client Program
- 1.4 Facade Design Pattern Important Points

Facade Design Pattern



According to GoF Facade design pattern is:

> Provide a unified interface to a set of interfaces in a subsystem. Facade Pattern defines a higher-level interface that makes the subsystem easier to use.

Suppose we have an application with set of interfaces to use MySql/Oracle database and to generate different types of reports, such as HTML report, PDF report etc.

So we will have different set of interfaces to work with different types of database. Now a client application can use these interfaces to get the required database connection and generate reports.

But when the complexity increases or the interface behavior names are confusing, client application will find it difficult to manage it.

So we can apply Facade design pattern here and provide a wrapper interface on top of the existing interface to help client application.

Facade Design Pattern – Set of Interfaces

We can have two helper interfaces, namely `MySqlHelper` and `OracleHelper`.

```
package com.journaldev.design.facade;

import java.sql.Connection;

public class MySqlHelper {

	public static Connection getMySqlDBConnection(){
		//get MySql DB connection using connection parameters
		return null;
```

```
        }

        public void generateMySqlPDFReport(String tableName, Connection con){
                //get data from table and generate pdf report
        }

        public void generateMySqlHTMLReport(String tableName, Connection con){
                //get data from table and generate pdf report
        }
}


package com.journaldev.design.facade;

import java.sql.Connection;

public class OracleHelper {

        public static Connection getOracleDBConnection(){
                //get Oracle DB connection using connection parameters
                return null;
        }

        public void generateOraclePDFReport(String tableName, Connection con){
                //get data from table and generate pdf report
        }

        public void generateOracleHTMLReport(String tableName, Connection con){
                //get data from table and generate pdf report
        }

}
```

Facade Design Pattern Interface

We can create a Facade pattern interface like below. Notice the use of Java Enum for type safety.

```
package com.journaldev.design.facade;

import java.sql.Connection;

public class HelperFacade {
```

```
        public static void generateReport(DBTypes dbType, ReportTypes reportType,
String tableName){
                Connection con = null;
                switch (dbType){
                case MYSQL:
                        con = MySqlHelper.getMySqlDBConnection();
                        MySqlHelper mySqlHelper = new MySqlHelper();
                        switch(reportType){
                        case HTML:
                                mySqlHelper.generateMySqlHTMLReport(tableName, con);
                                break;
                        case PDF:
                                mySqlHelper.generateMySqlPDFReport(tableName, con);
                                break;
                        }
                        break;
```

Facade Design Pattern Client Program

Now lets see client code without using Facade pattern and using Facade pattern interface.

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

import java.sql.Connection;

import com.journaldev.design.facade.HelperFacade;
import com.journaldev.design.facade.MySqlHelper;
import com.journaldev.design.facade.OracleHelper;

public class FacadePatternTest {

        public static void main(String[] args) {
                String tableName="Employee";

                //generating MySql HTML report and Oracle PDF report without using
Facade
                Connection con = MySqlHelper.getMySqlDBConnection();
                MySqlHelper mySqlHelper = new MySqlHelper();
                mySqlHelper.generateMySqlHTMLReport(tableName, con);
```

```
Connection con1 = OracleHelper.getOracleDBConnection();
OracleHelper oracleHelper = new OracleHelper();
```

As you can see that using Facade pattern interface is a lot easier and cleaner way to avoid having a lot of logic at client side. JDBC Driver Manager class to get the database connection is a wonderful example of facade design pattern.

Facade Design Pattern Important Points

- Facade design pattern is more like a helper for client applications, it doesn't hide subsystem interfaces from the client. Whether to use Facade or not is completely dependent on client code.
- Facade design pattern can be applied at any point of development, usually when the number of interfaces grow and system gets complex.
- Subsystem interfaces are not aware of Facade and they shouldn't have any reference of the Facade interface.
- Facade design pattern should be applied for similar kind of interfaces, its purpose is to provide a single interface rather than multiple interfaces that does the similar kind of jobs.
- We can use Factory pattern with Facade to provide better interface to client systems.

Thats all for Facade design pattern, stay tuned for more design pattern articles. □

« PREVIOUS
Decorator Design Pattern in Java Example

NEXT »
Flyweight 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

Prasanti says
MARCH 26, 2018 AT 12:12 AM
I have been reading all your design patterns. Thanks for the explanation. It indeed helped me.
Reply

SUPRATIM BHATTACHARYA says
JANUARY 27, 2018 AT 10:21 AM
Superbly described.. Thanks Pankaj.. Please post some more Java Tricky interview questions in future..
Reply

day says
SEPTEMBER 4, 2017 AT 12:13 PM
thank you, really clear and easy to understand
Reply

DEBARATI MAJUMDER says
AUGUST 18, 2017 AT 5:00 AM
Looks like Facade and Factory design patterns are somewhat similar.
Reply

Abhirag Kesarwani says
JULY 1, 2017 AT 11:55 PM
very useful article
Reply

Nikhil says
APRIL 30, 2016 AT 5:14 PM
Why do you return null, return a connection obj, will look better
Reply

Hafeez says
DECEMBER 28, 2015 AT 1:09 AM
simple and clear explanations. thanks.
Reply

Ronald says
JUNE 8, 2015 AT 8:16 PM
Hi Pankaj,
Indeed a very interesting article, pardon me for asking, since im lack of skill in the design pattern, but i notice your HelperFacade, is kinda acting like a Factory Pattern that calls the Facades(MySql,Oracle,etc). So in this article, you were explaining both design pattern in one post, fantastic Am i correct?
Reply

Sachin Tailor says
MARCH 12, 2015 AT 11:05 PM
You have not used any interface here. Is the definition satisfied ?
“Facade Pattern defines a higher-level interface that makes the subsystem easier to use” —
Reply

Ramandeep says
AUGUST 4, 2015 AT 6:18 AM
Hi Sachin.
I think in definition it doesnt mean Java interfaces, it means simple client interface. In Facade pattern we usually use class with static methods.

Reply

Muhammad Gelbana says
FEBRUARY 14, 2015 AT 11:15 AM
The first time I understand this pattern Thank you.
Also please fix the comment for the Oracle helper class. It looks like you first created the MySQL helper class and copied it with a new name. What I'm trying to say is that the "getOracleDBConnection" method has a misleading comment.
Reply

Pankaj says
FEBRUARY 14, 2015 AT 8:34 PM
Thanks, corrected the typo.
Reply

You Know says
DECEMBER 18, 2014 AT 2:56 PM
"Facade pattern is more like a helper for client applications, it doesn't hide subsystem interfaces from the client. Whether to use Facade or not is completely dependent on client code".
That's a very good reason to use a facade sometimes: to hide subsystems interfaces.
Reply

Gopinath says
JUNE 4, 2014 AT 12:57 AM
excellent article. simple and subtle description of the design pattern. Thank you.
Reply

Swetha says
MAY 20, 2014 AT 6:48 AM
Very informative. Thank you..
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

DESIGN PATTERNS TUTORIAL

Java Design Patterns

Creational Design Patterns

- Singleton
- Factory
- Abstract Factory
- Builder
- Prototype

Structural 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