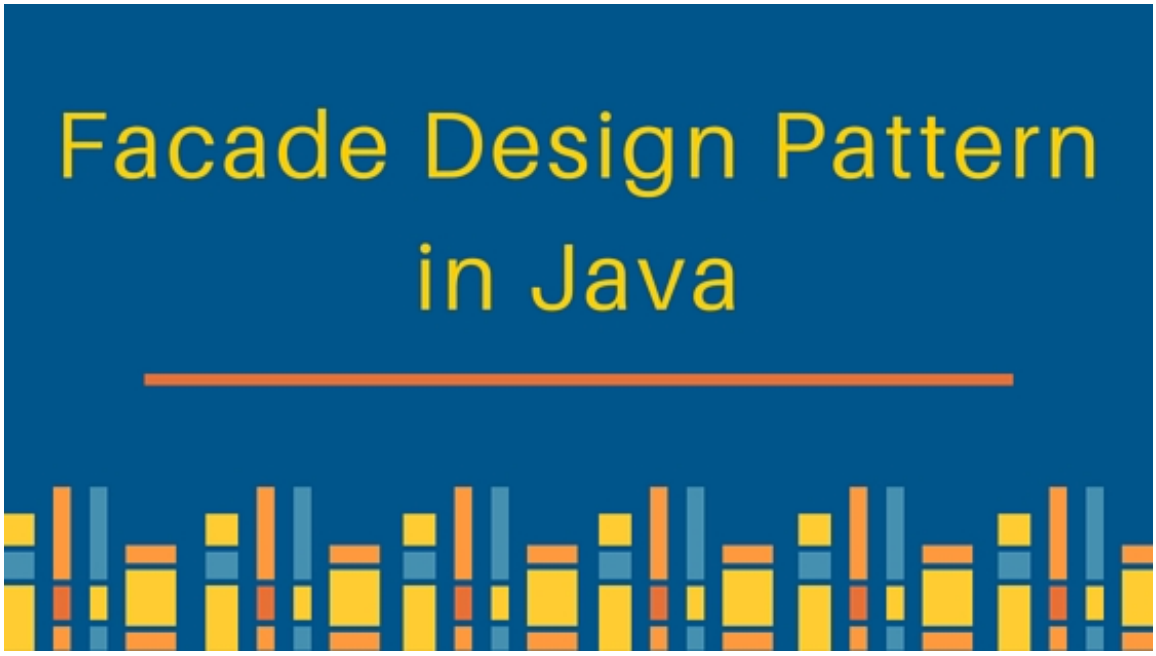
Facade Design Pattern in Java

journaldev.com/1557/facade-design-pattern-in-java

July 7, 2013

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.

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;
  case ORACLE:
   con = OracleHelper.getOracleDBConnection();
   OracleHelper oracleHelper = new OracleHelper();
   switch(reportType){
   case HTML:
    oracleHelper.generateOracleHTMLReport(tableName, con);
    break;
   case PDF:
    oracleHelper.generateOraclePDFReport(tableName, con);
    break;
   }
   break;
  }

 }

 public static enum DBTypes{
  MYSQL,ORACLE;
 }

 public static enum ReportTypes{
  HTML,PDF;
 }
}
```

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();
  oracleHelper.generateOraclePDFReport(tableName, con1);

  //generating MySql HTML report and Oracle PDF report using Facade
  HelperFacade.generateReport(HelperFacade.DBTypes.MYSQL,
HelperFacade.ReportTypes.HTML, tableName);
  HelperFacade.generateReport(HelperFacade.DBTypes.ORACLE,
HelperFacade.ReportTypes.PDF, tableName);
 }

}
```

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. 🙂