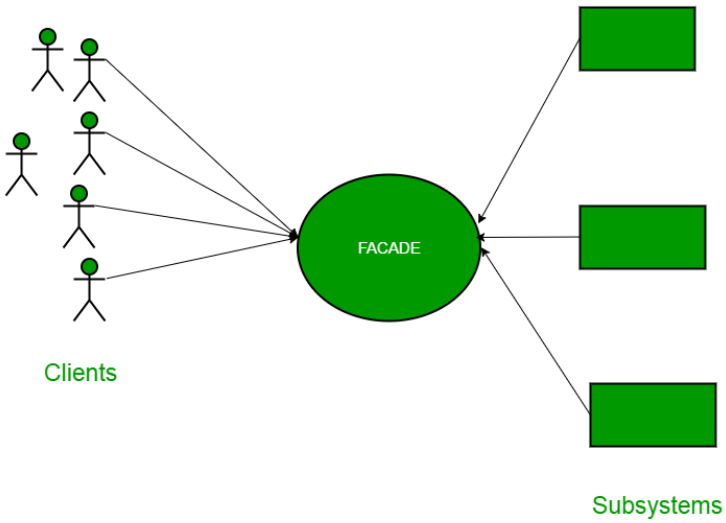
Facade Design Pattern | Introduction

Read Discuss

Facade is a part of the Gang of Four design patterns and it is categorized under Structural design patterns. Before we dig into the details of it, let us discuss some examples which will be solved by this particular Pattern. So, As the name suggests, it means the face of the building. The people walking past the road can only see the glass face of the building. They do not know anything about it, the wiring, the pipes, and other complexities. It hides all the complexities of the building and displays a friendly face.

More examples

In Java, the interface JDBC can be called a facade because we as users or clients create connections using the “java.sql.Connection” interface, the implementation of which we are not concerned about. The implementation is left to the vendor of the driver. Another good example can be the startup of a computer. When a computer starts up, it involves the work of CPU, memory, hard drive, etc. To make it easy to use for users, we can add a facade that wraps the complexity of the task, and provide one simple interface instead. The same goes for the **Facade Design Pattern**. It hides the complexities of the system and provides an interface to the client from where the client can access the system.



Facade Design Pattern Diagram

Now Let's try and understand the facade pattern better using a simple example. Let's consider a hotel. This hotel has a hotel keeper. There are a lot of restaurants inside the hotel e.g. Veg restaurants, Non-Veg restaurants, and Veg/Non Both restaurants. You, as a client want access to different menus of different restaurants. You do not know what are the different menus they have. You just have access to a hotel keeper who knows his hotel well. Whichever menu you want, you tell the hotel keeper and he takes it out of the respective restaurants and hands it over to you. Here, the hotel keeper acts as the **facade**, as he hides the complexities of the system hotel. Let's see how it works:

Interface of Hotel

Java

```
package structural.facade;

public interface Hotel {
```

The hotel interface only returns Menus. Similarly, the Restaurant are of three types and can implement the hotel interface. Let's have a look at the code for one of the Restaurants.

NonVegRestaurant.java

Java

```
package structural.facade;

public class NonVegRestaurant implements Hotel {

    public Menus getMenus()
    {
        NonVegMenu nv = new NonVegMenu();
        return nv;
    }
}
```

VegRestaurant.java

Java

```
package structural.facade;

public class VegRestaurant implements Hotel {

    public Menus getMenus()
    {
        VegMenu v = new VegMenu();
        return v;
    }
}
```

VegNonBothRestaurant.java

```
package structural.facade;

public class VegNonBothRestaurant implements Hotel {

    public Menus getMenus()
    {
        Both b = new Both();
        return b;
    }
}
```

Now let's consider the facade,

HotelKeeper.java

Java

```
/*package whatever //do not write package name here */

package structural.facade;

public interface HotelKeeper {


  public VegMenu getVegMenu();
  public NonVegMenu getNonVegMenu();
  public Both getVegNonMenu();

}
```

HotelKeeperImplementation.java

Java

```
package structural.facade;
```

```
    {
        VegRestaurant v = new VegRestaurant();
        VegMenu vegMenu = (VegMenu)v.getMenus();
        return vegMenu;
    }
```



```
    {
        NonVegRestaurant v = new NonVegRestaurant();
        NonVegMenu NonvegMenu = (NonVegMenu)v.getMenus();
        return NonvegMenu;
    }

    public Both getVegNonMenu()
    {
        VegNonBothRestaurant v = new VegNonBothRestaurant();
        Both bothMenu = (Both)v.getMenus();
        return bothMenu;
    }
}
```

From this, It is clear that the complex implementation will be done by HotelKeeper himself. The client will just access the HotelKeeper and ask for either Veg, NonVeg or VegNon Both Restaurant menu.

How will the client program access this façade?

Java

```
package structural.facade;

public class Client
{
    public static void main (String[] args)
    {
        HotelKeeper keeper = new HotelKeeperImplementation();

        VegMenu v = keeper.getVegMenu();
        NonVegMenu nv = keeper.getNonVegMenu();
        Both = keeper.getVegNonMenu();
```

In this way, the implementation is sent to the façade. The client is given just one interface and can access only that. This hides all the complexities.

When Should this pattern be used?

The facade pattern is appropriate when you have a **complex system** that you want to expose to clients in a simplified way, or you want to make an external communication layer over an existing system that is incompatible with the system. Facade deals with interfaces, not implementation. Its purpose is to hide internal complexity behind a single interface that appears simple on the outside.

Further Read: *Facade Method in Python*

This article is contributed by **Saket Kumar**. If you like GeeksforGeeks and would like to contribute, you can also write an article using write.geeksforgeeks.org or mail your article to review-team@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.

Last Updated : 06 Feb, 2023 31

Similar Reads

1. Facade Method - Python Design Patterns
2. The Decorator Pattern | Set 2 (Introduction and Design)

4. Singleton Design Pattern | Implementation
5. Decorator Pattern | Set 3 (Coding the Design)
6. Flyweight Design Pattern
7. Java Singleton Design Pattern Practices with Examples
8. Proxy Design Pattern
9. Composite Design Pattern
10. Prototype Design Pattern

Previous Next

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Improved By : gust7hgd, quest, mirceag2tqt

Article Tags : Design Pattern

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Greedy

Dynamic Programming

Pattern Searching

Recursion

Backtracking

JavaScript

Bootstrap

ReactJS

AngularJS

NodeJS

Computer Science

GATE CS Notes

Operating Systems

Computer Network

Database Management System

Software Engineering

Digital Logic Design

Engineering Maths

Python

Python Programming Examples

Django Tutorial

Python Projects

Python Tkinter

OpenCV Python Tutorial

Python Interview Question

Data Science & ML

Data Science With Python

Data Science For Beginner

Machine Learning Tutorial

Maths For Machine Learning

Pandas Tutorial

NumPy Tutorial

NLP Tutorial

Deep Learning Tutorial

DevOps

Git

AWS

Docker

Kubernetes

Azure

GCP

Competitive Programming

Top DSA for CP

Top 50 Tree Problems

System Design

What is System Design

Monolithic and Distributed SD

Top 50 DP Problems

Top 15 Websites for CP

Low Level Design or LLD

Top SD Interview Questions

Interview Corner

Company Preparation

Preparation for SDE

Company Interview Corner

Experienced Interview

Internship Interview

Competitive Programming

Aptitude

GfG School

CBSE Notes for Class 8

CBSE Notes for Class 9

CBSE Notes for Class 10

CBSE Notes for Class 11

CBSE Notes for Class 12

English Grammar

Commerce

Accountancy

Business Studies

Microeconomics

Macroeconomics

Statistics for Economics

Indian Economic Development

UPSC

Polity Notes

Geography Notes

History Notes

Science and Technology Notes

Economics Notes

Important Topics in Ethics

UPSC Previous Year Papers

SSC/ BANKING

SSC CGL Syllabus

SBI PO Syllabus

SBI Clerk Syllabus

IBPS PO Syllabus

IBPS Clerk Syllabus

Aptitude Questions

Write & Earn

Write an Article

Improve an Article

Pick Topics to Write

Write Interview Experience

Internships

Video Internship