Facade Method Design Pattern

Last Updated : 03 Jan, 2025

Facade Method Design Pattern is a part of the Gang of Four design patterns and it is categorized under Structural design patterns. Before we go into the details, visualize a structure. The house is the facade, it is visible to the outside world, but beneath it is a working system of pipes, cables, and other components that allow the building to run. It provides an easy-to-use interface so that users may interact with the system.

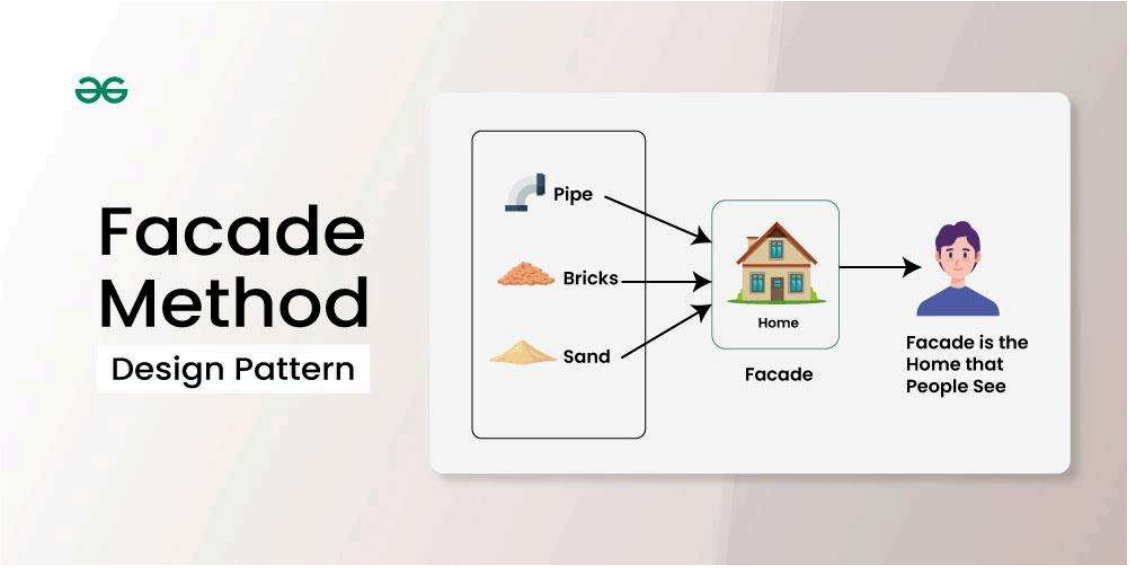
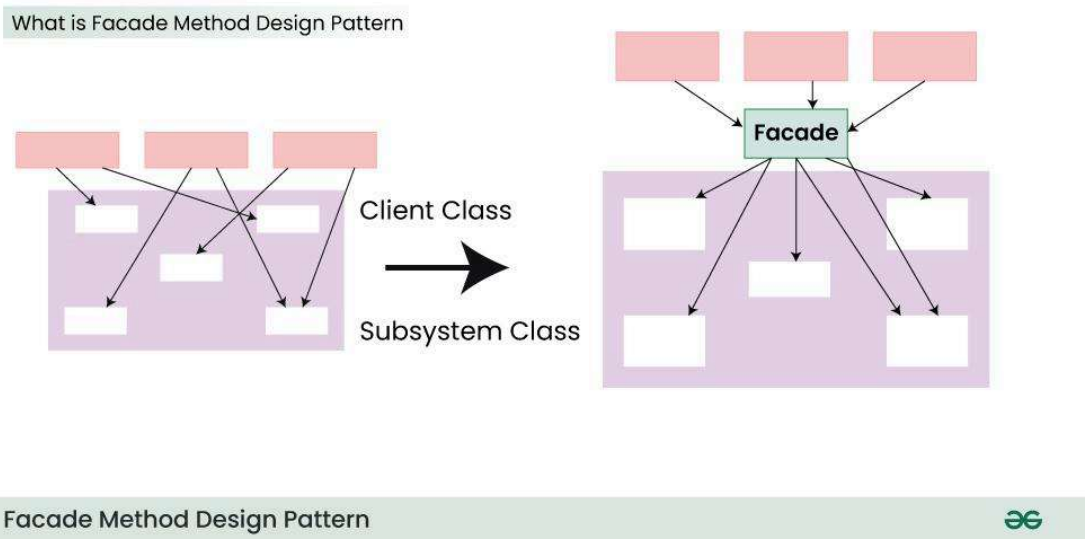


Table of Content

What is the Facade Method Design Pattern?

Facade Method Design Pattern provides a unified interface to a set of interfaces in a subsystem. Facade defines a high-level interface that makes the subsystem easier to use.



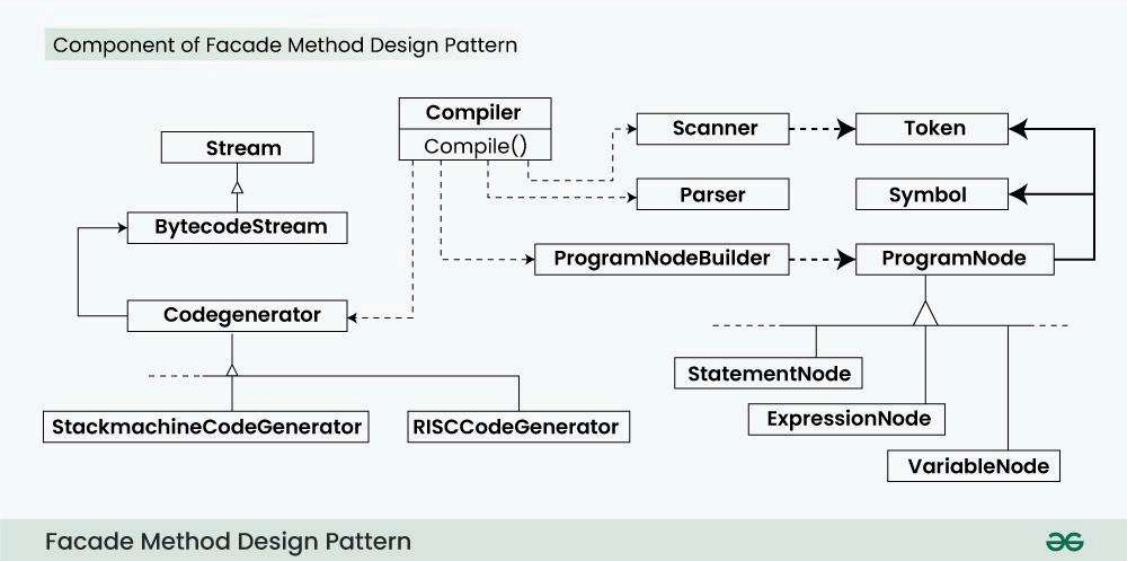
In the above diagram,

- Structuring a system into subsystems helps reduce complexity.
- A common design goal is to minimize the communication and dependencies between subsystems.
- One way to achieve this goal is to introduce a Facade object that provides a single simplified interface to the more general facilities of a subsystem.

When to use Facade Method Design Pattern

- A Facade provide a simple default view of the subsystem that is good enough for most clients.
- There are many dependencies between clients and the implementation classes of an abstraction.
- A Facade to decouple the subsystem from clients and other subsystems, thereby promoting subsystem independence and portability.
- Facade define an entry point to each subsystem level. If subsystem are dependent, then you can simplify the dependencies between them by making them communicate with each other through their facades.

Key Components of Facade Method Design Pattern



In the above diagram, Consider for example a programming environment that gives applications access to its compiler subsystem.

- This subsystem contains classes such as Scanner,Parser, ProgramNode, BytecodeStream, and ProgramNodeBuilder that implement the compiler.
- Compiler class acts as a facade: It offers clients a single, simple interface to the compilersubsystem.
- It glues together the classes that implement compilerfunctionality without hiding themcompletely.
- The compiler facade makes life easier for most programmers without hiding the lower-level functionality from the few that need it.

1. Facade (Compiler)

- Facade knows which subsystem classes are responsible for a request.
- It delegate client requests to appropriate subsystem objects.

2. Subsystem classes (Scanner, Parser, ProgramNode, etc.)

- It implement subsystem functionality.
- It handle work assigned by the Facade object.
- It have no knowledge of the facade; that is, they keep no references to it.

3. Interface

- The Interface in the Facade Design Pattern refers to the set of simplified methods that the facade exposes to the client.
- It hides the complexities of the subsystem, ensuring that clients interact only with high-level operations, without dealing with the underlying details of the system.

Facade Method Design Pattern collaborate in different way

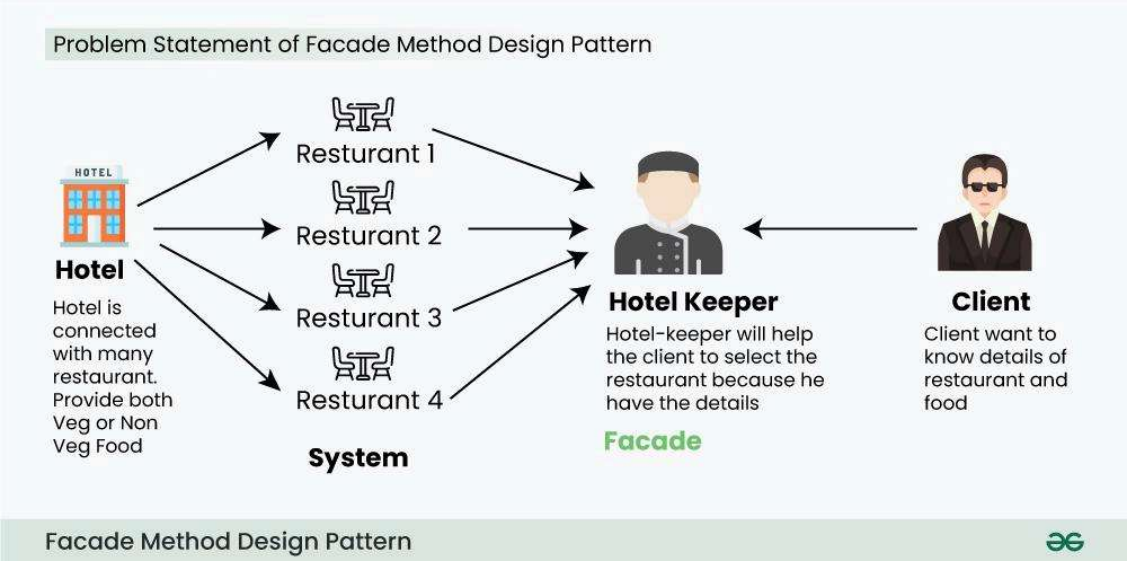
- Client communicate with the subsystem by sending requests to Facade, which forwards them to the appropriate subsystem objects.
- The Facade may have to do work of its own to translate it inheritance to subsystem interface.
- Clients that use the Facade don't have to access its subsystem objects directly.

Steps to implement Facade Design Pattern

Below are the simple steps to implement the Facade Design Pattern:

- **Step 1:** First, determine the complex subsystems or components that the client needs to interact with.
- **Step 2:** Build a facade class that serves as the middle layer between the client and the subsystems. This class should offer simplified methods that wrap the interactions with the subsystems.
- **Step 3:** Expose a clear, high-level interface in the facade class. This interface will include the methods that the client will use, hiding the internal complexity.
- **Step 4:** Inside the facade methods, delegate the requests to the appropriate subsystem classes to perform the actual operations.
- **Step 5:** The client now interacts only with the facade, which manages the underlying calls to the subsystems, simplifying the client's experience.

Exmaple for the Facade Method Design Pattern (with implementation)



Problem Statement:

> *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.*

So here, **Hotel-Keeper** is Facade and respective **Restaurants** is system.Below is the step-by-step Implementation of above problem

1. Interface of Hotel

```
package structural.facade;

public interface Hotel {
    public Menus getMenus();
}
```

> *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.

2. NonVegRestaurant

```
package structural.facade;

public class NonVegRestaurant implements Hotel {

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

3. VegRestaurant

```
package structural.facade;

public class VegRestaurant implements Hotel {

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

4. VegNonBothRestaurant

```
package structural.facade;

public class VegNonBothRestaurant implements Hotel {

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

Now let's consider the facade,

1. HotelKeeper.java

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

package structural.facade;

public interface HotelKeeper {


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

}
```

2. HotelKeeperImplementation.java

```
package structural.facade;

public class HotelKeeperImplementation implements HotelKeeper {

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

    public NonVegMenu getNonVegMenu()
    {
        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 facade?

```
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.

Use Cases of Facade Method Design Pattern

- **Simplifying Complex External Systems:**
    - A facade encapsulates database connection, query execution, and result processing, offering a clean interface to the application.
    - A facade simplifies the usage of external APIs by hiding the complexities of authentication, request formatting, and response parsing.
- **Layering Subsystems:**
    - Facades define clear boundaries between subsystems, reducing dependencies and promoting modularity.

    - Facades offer simplified interfaces to lower-level subsystems, making them easier to understand and use.
- **Providing a Unified Interface to Diverse Systems:**
    - A facade can combine multiple APIs into a single interface, streamlining interactions and reducing code duplication.
    - A facade can create a modern interface for older, less accessible systems, facilitating their integration with newer components.
- **Protecting Clients from Unstable Systems:**
    - Facades minimize the impact of changes to underlying systems by maintaining a stable interface.
    - Facades can protect clients from changes or issues in external libraries or services.

Advantages of Facade Method Design Pattern

- **Simplified Interface:**
    - Simplifies the use and understanding of a complex system by offering a clear and concise interface.
    - Hides the internal details of the system, reducing cognitive load for clients.
- **Reduced Coupling:**
    - Clients become less reliant on the internal workings of the underlying system when they are disconnected from it.
    - Encourages the reusability and modularity of code components.
    - Allows for the independent testing and development of various system components.
- **Encapsulation:**
    - Encapsulates the complex interactions within a subsystem, protecting clients from changes in its implementation.
    - Allows for changes to the subsystem without affecting clients, as long as the facade interface remains stable.
- **Improved Maintainability:**
    - Easier to change or extend the underlying system without affecting clients, as long as the facade interface remains consistent.

- Allows for refactoring and optimization of the subsystem without impacting client code.

Disadvantages of Facade Method Design Pattern

- **Increased Complexity:**
  - Adding the facade layer in the system increases the level of abstraction.
  - Because of this, the code may be more difficult to understand and debug
- **Reduced Flexibility:**
  - The facade acts as a single point of access to the underlying system.
  - This can limit the flexibility for clients who need to bypass the facade or access specific functionalities hidden within the subsystem.
- **Overengineering:**
  - Applying the facade pattern to very simple systems can be overkill, adding unnecessary complexity where it's not needed.
  - Consider the cost-benefit trade-off before implementing a facade for every situation.
- **Potential Performance Overhead:**
  - Adding an extra layer of indirection through the facade can introduce a slight performance overhead, especially for frequently used operations.
  - This may not be significant for most applications, but it's worth considering in performance-critical scenarios.

Conclusion

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.

Other Articles:

- Software Design Pattern Tutorial
- Java Design Patterns Tutorial
- Modern C++ Design Patterns Tutorial
- JavaScript Design Patterns Tutorial
- Python Design Patterns Tutorial

Next Article

Flyweight Design Pattern

Similar Reads

Software Design Patterns Tutorial

Software design patterns are important tools developers, providing proven solutions to common problems encountered during software development. This article will act as tutorial to help you understand the concep...

9 min read

Complete Guide to Design Patterns

Design patterns help in addressing the recurring issues in software design and provide a shared vocabulary for developers to communicate and collaborate effectively. They have been documented and refined over tim...

11 min read

Types of Software Design Patterns

Designing object-oriented software is hard, and designing reusable object-oriented software is even harder. Christopher Alexander says, "Each pattern describes a problem which occurs over and over again in our...

9 min read

1. Creational Design Patterns

Creational Design Patterns

Creational Design Patterns focus on the process of object creation or problems related to object creation. They help in making a system independent of how its objects are created, composed, and represented. Creational...

4 min read

Types of Creational Patterns

2. Structural Design Patterns

Structural Design Patterns

Structural Design Patterns are solutions in software design that focus on how classes and objects are organized to form larger, functional structures. These patterns help developers simplify relationships betwee...

7 min read

Types of Structural Patterns

Adapter Design Pattern

One structural design pattern that enables the usage of an existing class's interface as an additional interface is the adapter design pattern. To make two incompatible interfaces function together, it serves as a bridge....

8 min read

Bridge Design Pattern

The Bridge design pattern allows you to separate the abstraction from the implementation. It is a structural design pattern. There are 2 parts in Bridge design pattern : AbstractionImplementationThis is a design...

4 min read

Composite Method | Software Design Pattern

Composite Pattern is a structural design pattern that allows you to compose objects into tree structures to represent part-whole hierarchies. The main idea behind the Composite Pattern is to build a tree structure of...

9 min read

Decorator Design Pattern

The Decorator Design Pattern is a structural design pattern that allows behavior to be added to individual objects dynamically, without affecting the behavior of other objects from the same class. It involves creating...

9 min read

Facade Method Design Pattern

Facade Method Design Pattern is a part of the Gang of Four design patterns and it is categorized under Structural design patterns. Before we go into the details, visualize a structure. The house is the facade, it is...

8 min read

Flyweight Design Pattern

The Flyweight design pattern is a structural pattern that optimizes memory usage by sharing a common state among multiple objects. It aims to reduce the number of objects created and to decrease memory...

10 min read

Proxy Design Pattern

The Proxy Design Pattern a structural design pattern is a way to use a placeholder object to control access to another object. Instead of interacting directly with the main object, the client talks to the proxy, which then...

9 min read

3. Behvioural Design Patterns

Behavioral Design Patterns

Behavioral design patterns are a category of design patterns that focus on the interactions and communication between objects. They help define how objects collaborate and distribute responsibility among them, makin...

5 min read

3. Types of Behvioural Patterns

Top Design Patterns Interview Questions [2024]

A design pattern is basically a reusable and generalized solution to a common problem that arises during software design and development. Design patterns are not specific to a particular programming language or...

9 min read

Software Design Pattern in Different Programming Languages

Java Design Patterns Tutorial

Design patterns in Java refer to structured approaches involving objects and classes that aim to solve recurring design issues within specific contexts. These patterns offer reusable, general solutions to common problems...

8 min read

Python Design Patterns Tutorial

Design patterns in Python are communicating objects and classes that are customized to solve a general design problem in a particular context. Software design patterns are general, reusable solutions to common...

8 min read

Modern C++ Design Patterns Tutorial

Design patterns in C++ help developers create maintainable, flexible, and understandable code. They encapsulate the expertise and experience of seasoned software architects and developers, making it easier f...

7 min read

JavaScript Design Patterns Tutorial

Design patterns in Javascipt are communicating objects and classes that are customized to solve a general design problem in a particular context. Software design patterns are general, reusable solutions to common...

8 min read

Software Design Pattern Books

Software Design Pattern in Development

Some other Popular Design Patterns

Design Patterns in Different Languages

Interview Questions

School Subjects

Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar
Commerce
World GK

GeeksforGeeks Videos

DSA
Python
Java
C++
Web Development
Data Science
CS Subjects