Facade

Definition

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

Frequency of use (*in JavaScript*): 1 2 3 4 5 high

Summary

The Façade pattern provides an interface which shields clients from complex functionality in one or more subsystems. It is a simple pattern that may seem trivial but it is powerful and extremely useful. It is often present in systems that are built around a multi-layer architecture.

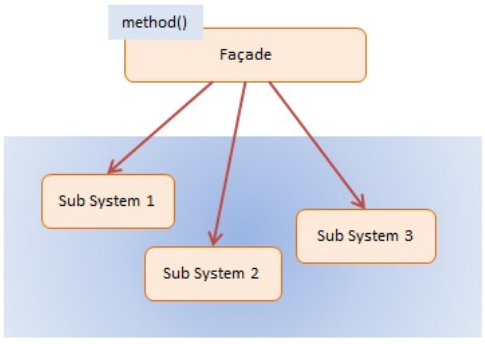
The intent of the Façade is to provide a high-level interface (properties and methods) that makes a subsystem or toolkit easy to use for the client.

On the server, in a multi-layer web application you frequently have a presentation layer which is a client to a service layer. Communication between these two layers takes place via a well-defined API. This API, or façade, hides the complexities of the business objects and their interactions from the presentation layer.

Another area where Façades are used is in refactoring. Suppose you have a confusing or messy set of legacy objects that the client should not be concerned about. You can hide this code behind a Façade. The Façade exposes only what is necessary and presents a cleaner and easy-to-use interface.

Façades are frequently combined with other design patterns. Facades themselves are often implemented as singleton factories.

Diagram



Participants

The objects participating in this pattern are:

- **Facade** -- In sample code: **Mortgage**
  - knows which subsystems are responsible for a request
  - delegates client requests to appropriate subsystem objects
- **Sub Systems** -- In sample code: **Bank, Credit, Background**

- implements and performs specialized subsystem functionality
- have no knowledge of or reference to the facade

Sample code in JavaScript

The Mortgage object is the Facade in the sample code. It presents a simple interface to the client with only a single method: applyFor. Eut underneath this simple API lies considerable complexity.

The applicant's name is passed into the Mortgage constructor function. Then the applyFor method is called with the requested loan amount. Internally, this method uses services from 3 separate subsystems that are complex and possibly take some time to process; they are Bank, Credit, and Background.

Based on several criteria (bank statements, credit reports, and criminal background) the applicant is either accepted or denied for the requested loan.

```
var Mortgage = function(name) {
    this.name = name;
}

Mortgage.prototype = {

    applyFor: function(amount) {
        // access multiple subsystems...
        var result = "approved";
        if (!new Bank().verify(this.name, amount)) {
            result = "denied";
        } else if (!new Credit().get(this.name)) {
            result = "denied";
        } else if (!new Background().check(this.name)) {
            result = "denied";
        }
        return this.name + " has been " + result +
               " for a " + amount + " mortgage";
    }
}

var Bank = function() {
    this.verify = function(name, amount) {
        // complex logic ...
        return true;
    }
}

var Credit = function() {
    this.get = function(name) {
        // complex logic ...
        return true;
    }
}

var Background = function() {
    this.check = function(name) {
        // complex logic ...
        return true;
    }
}

function run() {
    var mortgage = new Mortgage("Joan Templeton");
    var result = mortgage.applyFor("$100,000");

    alert(result);
}

```

Run

JavaScript Optimized Code

The example given is not optimized for JavaScript. Significant improvements can be obtained by applying advanced JavaScript techniques resulting in more effective, robust, and maintainable apps.

To learn how, check our comprehensive **JavaScript + jQuery Design Pattern Framework (/products/javascript-jquery-design-pattern-framework)**.

This unique package includes optimized JavaScript for all GoF patterns using more advanced features, such as namespaces, prototypes, modules, function objects, closures, anonymous functions, and ohters.

If you require the latest tools and techniques on JavaScript Patterns, jQuery Patterns, and Pattern Architectures then the **JavaScript + jQuery Design Pattern Framework (/products/javascript-jquery-design-pattern-framework)** is for you!. This package contains valuable, up-to-date information for JavaScript developers. Here is what is included:

(/products/javascript-jquery-design-pattern-framework)

- JavaScript-optimized GoF patterns
- Modern JavaScript Design Patterns
- Model-View Design Patterns
- jQuery Design Patterns
- Architecture patterns
- JavaScript Idioms (great fun!)
- Sample Apps (MVC, SPA, etc.)

Get it now. Start writing beautiful JavaScript!

Learn More (/products/javascript-jquery-design-pattern-framework)

Company

- About Us (/about)
- Our Story (/story)
- Services (/services)
- Training (/training)
- Contact Us (/contact)

- Privacy (/privacy)
- Terms (/terms)
- Licensing (/licensing)

Customers

- Our Customers (/customers)
- Customer Stories (/customers/stories)

Community

- Questions (/topic/search.aspx)
- Ask Question (/topic/add.aspx)
- Explore (/topic/topics.aspx)
- Tags (/tag/tags.aspx)
- Users (/user/users.aspx)

Reference Guides

- .NET Design Patterns (/net/design-patterns)
- JavaScript Design Patterns (/javascript/design-patterns)

- JavaScript Tutorial (/tutorial/javascript)
- SQL Tutorial (/sql/tutorial)

- Connection Strings (/reference/connection-strings)
- Visual Studio Shortcuts (/reference/visual-studio-shortcuts)
- C# Coding Standards (/reference/csharp-coding-standards)
- HTML Colors (/reference/html-color-codes)

Our Products

- .NET Design Pattern Framework (/products/net-design-pattern-framework) TM
- PRO .NET Design Pattern Framework (/products/pro-net-design-pattern-framework) TM

- JavaScript + jQuery Pattern Framework (/products/javascript-jquery-design-pattern-framework) TM
- SQL + Database Pattern Framework (/products/sql-database-design-pattern-framework) TM

- Products and Pricing (/products)