

Facade Design Pattern

+ BOOKMARK

Search



Why read if you can watch?

[Watch design patterns video tutorial](#)

contents

Design Patterns

§ [Creational patterns](#)

§ [Structural patterns](#)

- [Structural patterns](#)
- [Adapter Design Pattern](#)
- [Bridge Design Pattern](#)
- [Composite Design Pattern](#)
- [Decorator Design Pattern](#)
- [Facade Design Pattern](#)
- [Flyweight Design Pattern](#)
- [Private Class Data](#)
- [Proxy Design Pattern](#)

Intent

- 01
- Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.
 - Wrap a complicated subsystem with a simpler interface.

Problem

- 02
- A segment of the client community needs a simplified interface to the overall functionality of a complex subsystem.

Discussion

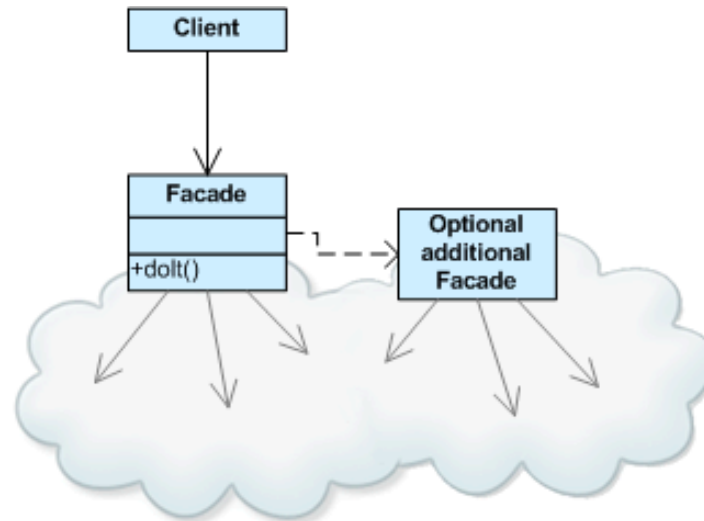
- 03
- Facade discusses encapsulating a complex subsystem within a single interface object.

This reduces the learning curve necessary to successfully leverage the subsystem. It also promotes decoupling the subsystem from its potentially many clients. On the other hand, if the Facade is the only access point for the subsystem, it will limit the features and flexibility that “power users” may need.

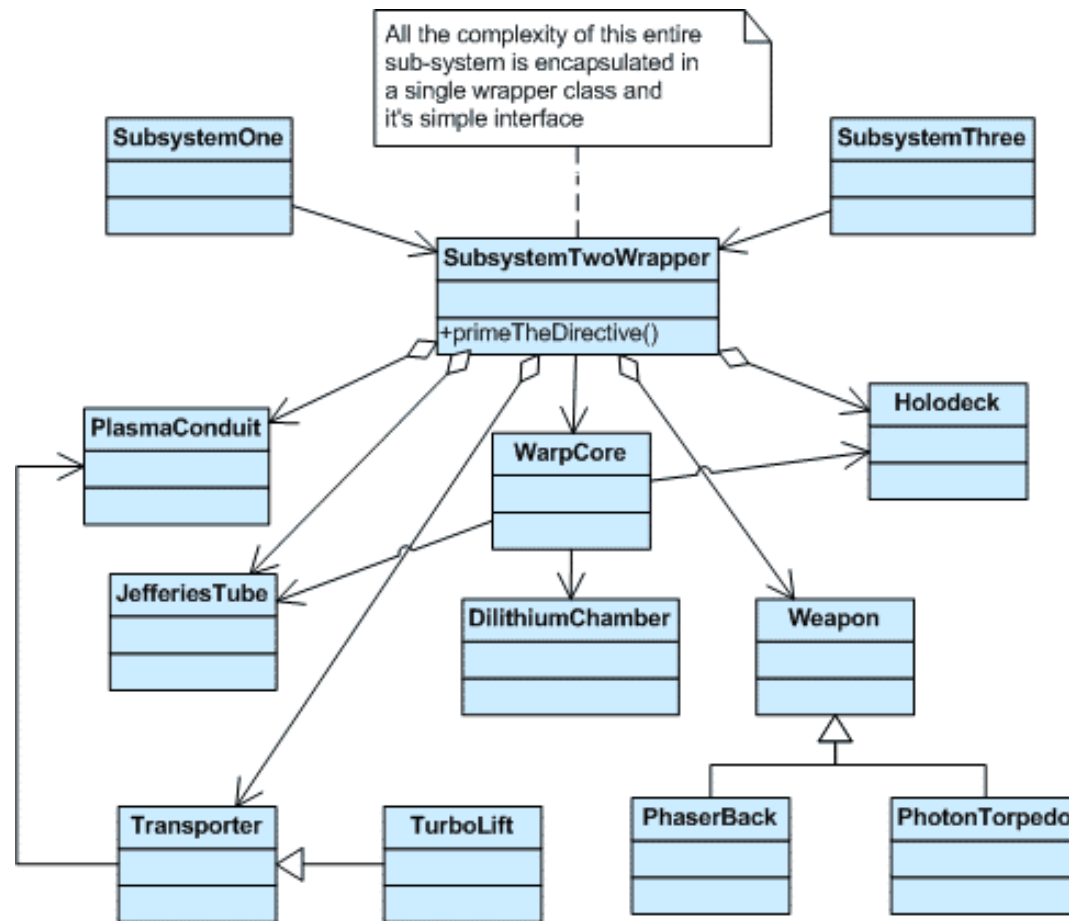
- 04 The Facade object should be a fairly simple advocate or facilitator. It should not become an all-knowing oracle or “god” object.

Structure

- 05 Facade takes a “riddle wrapped in an enigma shrouded in mystery”, and interjects a wrapper that tames the amorphous and inscrutable mass of software.



- 06 `SubsystemOne` and `SubsystemThree` do not interact with the internal components of `SubsystemTwo`. They use the `SubsystemTwoWrapper` “facade” (i.e. the higher level abstraction).

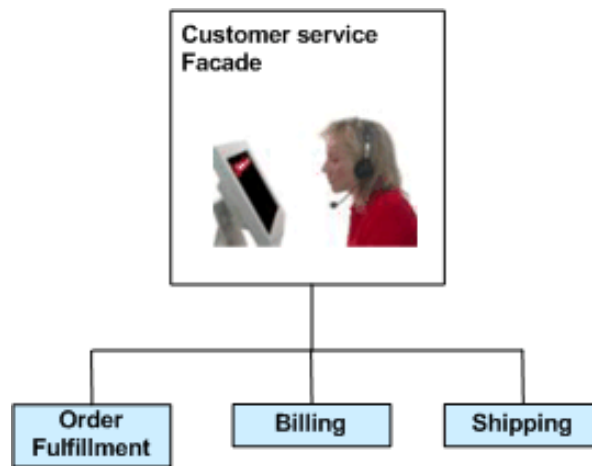


[+] Feedback

Example

07

The Facade defines a unified, higher level interface to a subsystem that makes it easier to use. Consumers encounter a Facade when ordering from a catalog. The consumer calls one number and speaks with a customer service representative. The customer service representative acts as a Facade, providing an interface to the order fulfillment department, the billing department, and the shipping department.



Check list

08

1. Identify a simpler, unified interface for the subsystem or component.
2. Design a 'wrapper' class that encapsulates the subsystem.
3. The facade/wrapper captures the complexity and collaborations of the component, and delegates to the appropriate methods.
4. The client uses (is coupled to) the Facade only.
5. Consider whether additional Facades would add value.

Rules of thumb

09

- Facade defines a new interface, whereas Adapter uses an old interface. Remember that Adapter makes two existing interfaces work together as opposed to defining an entirely new one.
- Whereas Flyweight shows how to make lots of little objects, Facade shows how to make a single object represent an entire subsystem.
- Mediator is similar to Facade in that it abstracts functionality of existing classes.

Mediator abstracts/centralizes arbitrary communications between colleague objects. It routinely “adds value”, and it is known/referenced by the colleague objects. In contrast, Facade defines a simpler interface to a subsystem, it doesn’t add new functionality, and it is not known by the subsystem classes.

- Abstract Factory can be used as an alternative to Facade to hide platform-specific classes.
- Facade objects are often Singletons because only one Facade object is required.
- Adapter and Facade are both wrappers; but they are different kinds of wrappers. The intent of Facade is to produce a simpler interface, and the intent of Adapter is to design to an existing interface. While Facade routinely wraps multiple objects and Adapter wraps a single object; Facade could front-end a single complex object and Adapter could wrap several legacy objects.

10

Question: So the way to tell the difference between the Adapter pattern and the Facade pattern is that the Adapter wraps one class and the Facade may represent many classes?

11

Answer: No! Remember, the Adapter pattern changes the interface of one or more classes into one interface that a client is expecting. While most textbook examples show the adapter adapting one class, you may need to adapt many classes to provide the interface a client is coded to. Likewise, a Facade may provide a simplified interface to a single class with a very complex interface. The difference between the two is not in terms of how many classes they “wrap”, it is in their intent.

Read next



This article is taken from the book **Design Patterns - Simply**.



ALL design patterns are compiled there. The book is written in a clear and simple language that makes it easy to read and understand (just like this article).

It is a part of our Design Patterns Course. We distribute it in PDF format, so it will be available for downloading in 10 seconds!

[Learn more...](#)

12

Facade code examples

C# examples

- [Facade in C#](#)

C++ examples

- [Facade in C++](#)

Delphi examples

- [Facade in Delphi](#)

Java examples

- [Facade in Java](#)

PHP examples

- [Facade in PHP](#)



This work is licensed under a [Creative Commons Attribution-NonCommercial-NoDerivatives 3.0 Unported License](#)