CHAPTER 6

The Facade Pattern

Overview

In this chapter

I start the study of design patterns with a pattern that you have probably implemented in the past but may not have had a name for: the Facade pattern.

This chapter

- Explains what the Facade pattern is and where it is used.
- Presents the key features of the pattern.
- Presents some variations on the Facade pattern.
- Relates the Facade pattern to the CAD/CAM problem.

Introducing the Facade Pattern

Intent: A unified, high-level interface

According to the Gang of Four, the intent of the Facade pattern is to

> *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.*[1]

Basically, this is saying that we need to interact with a system that is easier than the current method, or we need to use the system in a particular way (such as using a 3D drawing program in a 2D way). We can build such a method of interaction because we only need to use a subset of the system in question.

1. Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software,* Boston: Addison-Wesley, 1995, p. 185.

Learning the Facade Pattern

A motivating example: Learn how to use our complex system!

Once, I worked as a contractor for a large engineering and manufacturing company. My first day on the job, the technical lead of the project was not in. Now, this client did not want to pay me by the hour and not have anything for me to do. They wanted me to be doing something, even if it was not useful! Haven't you had days like this?

So, one of the project members found something for me to do. She said, "You are going to have to learn the CAD/CAM system we use some time, so you might as well start now. Start with these manuals over here." Then she took me to the set of documentation. I am not making this up: There were *8 feet* of manuals for me to read—each page 8.5 × 11 inches and in small print! This was one complex system!

Figure 6-1 Eight feet of manuals = one complex system!

I want to be insulated from this

Now, if you and I and say another four or five people were on a project that needed to use this system, what approach would we take? Would we all learn the system? Or would we draw straws and the *loser* would have to write routines that the rest of us would use to interface with the system?

This person would determine how I and others on our team were going to use the system and what *application programming interface* (API) would be best for our particular needs. She would then create a new class or classes that had the interface we required. Then I and the rest of the programming community could use this new interface without having to learn the entire complicated system (see Figure 6-2).

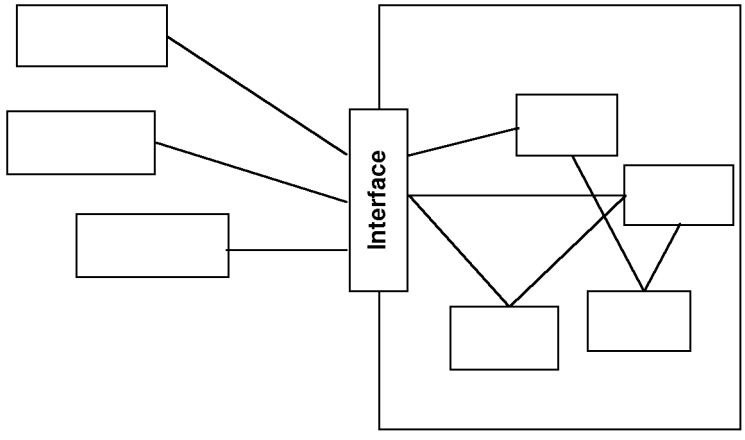

Figure 6-2 Insulating clients from the subsystem.

Works with subsets

This approach works only when using a subset of the system's capabilities or when interacting with it in a particular way. If everything in the system needs to be used, the only way to improve the design would be if it were poor in the first place.

This is the Facade pattern

This is the Facade pattern. It enables us to use a complex system more easily, either to use just a subset of the system or use the system in a particular way. We have a complicated system of which we need to use only a part. We end up with a simpler, easier-to-use system or one that is customized to our needs.

Most of the work still needs to be done by the underlying system. The Facade provides a collection of easier-to-understand methods. These methods use the underlying system to implement the newly defined functions.

The Facade Pattern: Key Features

Intent	You want to simplify how to use an existing system. You need to define your own interface.
Problem	You need to use only a subset of a complex system. Or you need to interact with the system in a particular way.
Solution	The Facade presents a new interface for the client of the existing system to use.
Participants and collaborators	It presents a simplified interface to the client that makes it easier to use.
Consequences	The Facade simplifies the use of the required subsystem. However, because the Facade is not complete, certain functionality may be unavailable to the client.
Implementation	Define a new class (or classes) that has the required interface. Have this new class use the existing system.

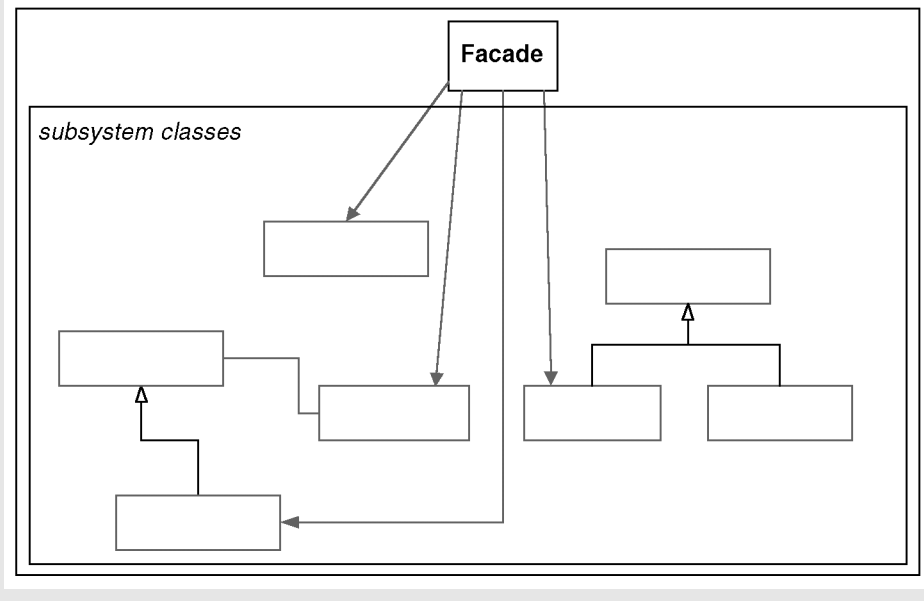

Figure 6-3 Generic structure of the Facade pattern.

Field Notes: The Facade Pattern

Variations on Facade: Reduce the number of objects a client must work with

Facades can be used not only to create a simpler interface in terms of method calls, but also to reduce the number of objects that a client object must deal with. For example, suppose I have a **Client** object that must deal with **Database**s, **Model**s, and **Element**s. The **Client** must first open the **Database** and get a **Model**. Then it queries the **Model** to get an **Element**. Finally, it asks the **Element** for information. It might be a lot easier to create a **DatabaseFacade** that could be queried by the **Client** (see Figure 6-4).

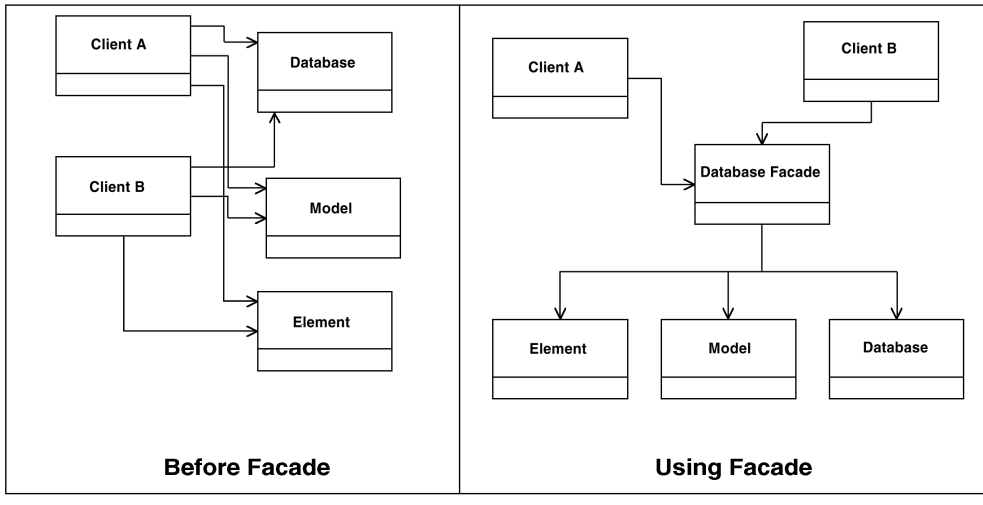

Figure 6-4 Facade reduces the number of objects for the client.

Having one Facade work for many objects

If a Facade can be made to be stateless (that is, no state is stored in it), one Facade object can be used by several other objects. Later, in Chapter 21, I show you how to do this, using the Singleton pattern and the Double-Checked Locking pattern.

Variations on Facade: Supplement existing functions with new routines

Suppose that in addition to using functions that are in the system, I also need to provide some new functionality—say, record all calls to specific routines. In this case, I am going beyond a simple subset of the system.

In this case, the methods I write for the Facade class may be supplemented by new routines for the new functionality. This is still the Facade pattern, but expanded with new functionality. I consider the primary goal one of simplification because I don't want to have to force the client routine to know that it needs to call the extra routines—the Facade does that.

The Facade pattern sets the general approach; it got me started. The Facade part of the pattern is the fact that I am creating a new interface for the client to use instead of the existing system's interface. I can do this because the Client object does not need to use all of the functions in my original system.

Patterns Set a General Approach

A pattern just sets the general approach. Whether or not to add new functionality depends upon the situation at hand. Patterns are blueprints to get you started; they are not carved in stone.

Variations on Facade: An "encapsulating" layer

The Facade can also be used to hide, or encapsulate, the system. The Facade could contain the system as private members of the Facade class. In this case, the original system would be linked in with the Facade class, but not presented to users of the Facade class.

There are a number of reasons to encapsulate the system, including the following:

- **Track system usage**—By forcing all accesses to the system to go through the Facade, I can easily monitor system usage.
- **Swap out systems**—I may need to switch systems in the future. By making the original system a private member of the Facade class, I can swap out the system for a new one with minimal effort. There may still be a significant amount of effort required, but at least I will only have to change the code in one place (the Facade class).

Relating the Facade Pattern to the CAD/CAM Problem

Encapsulate the V1 system

Think of the example above. The Facade pattern could be useful to help **V1Slot**s, **V1Hole**s, and so on use **V1System**. I will do just that in the solution in Chapter 13, "Solving the CAD/CAM Problem with Patterns."

Summary

In this chapter

The Facade pattern is so named because it puts up a new interface (a facade) in front of the original system.

The Facade pattern applies when

- You do not need to use all the functionality of a complex system and can create a new class that contains all the rules for accessing that system. If this is a subset of the original system, as it usually is, the API that you create for the new class should be much simpler than the original system's API.
- You want to encapsulate or hide the original system.
- You want to use the functionality of the original system and want to add some new functionality as well.
- The cost of writing this new class is less than the cost of everybody learning how to use the original system or is less than you would spend on maintenance in the future.

Review Questions

Observations

1. Define *Facade*.
2. What is the intent of the Facade pattern?
3. What are the consequences of the Facade pattern? Give an example.

4. In the Facade pattern, how do clients work with subsystems?
5. Does the Facade pattern usually give you access to the entire system?

Interpretations

1. The Gang of Four says that the intent of the Facade pattern is to "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."
   - What does this mean?
   - Give an example.
2. Here is an example of a Facade that comes from outside of software: Pumps at gasoline stations in the United States can be very complex. There are many options on them: how to pay, the type of gas to use, watch an advertisement. One way to get a unified interface to the gas pump is to use a human gas attendant. Some states even require this.
   - What is another example from real life that illustrates a Facade?

Opinions and Applications

1. If you need to add functionality beyond what the system provides, can you still use the Facade pattern?
2. What is a reason for encapsulating an entire system using the Facade pattern?
3. Is there a case for writing a new system rather than encapsulating the old system with Facade? What is it?
4. Why do you think the Gang of Four call this pattern Facade? Is it an appropriate name for what it is doing? Why or why not?