The Adapter and Facade Patterns

Relevant Links

- OODesign on Adapter[1]
- GoF Book on Adapter[2], page 139
- GoF Book on Facade[3], page 185

Notes

We discuss together two somewhat related patterns, the Adapter and Facade.

Adapter

We start with the Adapter.

Classification

The Adapter is a Structural Pattern.

Intent

The **Adapter** is used to convert the interface of a class to another interface that clients expect. It allows classes to work together that otherwise could not because of incompatible interfaces.

It is also often called **Wrapper**.

Motivation

The idea is that we have a class whose objects already achieve a certain goal and have a given interface. We want to use those objects in a different part of the application that uses a slightly different interface. For instance maybe some methods are omitted, or they have different names, or take different arguments, and so on.

The Adapter provides this "transformation".

Similar to the Proxy, the Adapter sits in front of the actual object, and it relays calls to the actual object. But unlike the Proxy, the Adapter *changes the interface*. While

[1] http://www.oodesign.com/adapter-pattern.html
[2] http://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612/
[3] http://www.amazon.com/Design-Patterns-Elements-Reusable-Object-Oriented/dp/0201633612/

the Proxy was meant to provide the exact same interface as the object it was proxying, the Adapter is meant to provide a different interface, to match what the client wants.

The Adapter is useful when you have to use some piece of code that behaves in a certain way, but you need to use it in a different way.

Participants / Implementation

Target This is the Interface that the client wants to use/see.
Client Expect to work with objects with Target interface.
Adaptee An existing interface provided by e.g. an external class.
Adapter Adapts the interface of the Adaptee to the Target.

Clients will call the Adapter, which appears to them via the Target interface. The Adapter turns around and relays requests to the Adaptee as appropriate.

Facade

We now look at the Facade Pattern.

Classification

The Facade is a Structural Pattern.

Intent

The Facade Pattern provides a unified interface to a set of interfaces in a subsystem. It essentially defines a higher-level interface that makes the subsystem easier to use.

Motivation

Use a Facade when you want to offer your clients a simpler interface, while still allowing them more fine-grained control when they choose to.

For instance if we are writing a compiler, then we have all sorts of different components: A lexxer that breaks the program into "tokens", a parser that turns those tokens into syntax structures, a class to hold the different syntax structures, a semantic analyzer, a code generator and so on and so forth. For somes uses clients might need to peek into all these parts, but for the most part they just want to get their programs compiled. This is where the Facade comes in, to expose to the client a simplified interface with a "compile" method that under the hood calls all these other components.

Similarly in a Linear Algebra application, there is typically a "solve" method that solves linear systems. Behind the scenes there are many different specialized types

of solvers, depending on what type of system of equations we are solving. While specialized clients might want to pick the specific subsystem, most just want a "solve" method.

Participants / Implementation

Facade Knows the various subsystem classes and what it needs to use to handle a request. It delegates all client requests to appropriate subystem objects.

Subsystem Classes The subsystem classes implement their functionality as normal, and they handle the work assigned to them by the Facade object.

They have no knowledge of the fact that they are called by the Facade rather than the client directly.