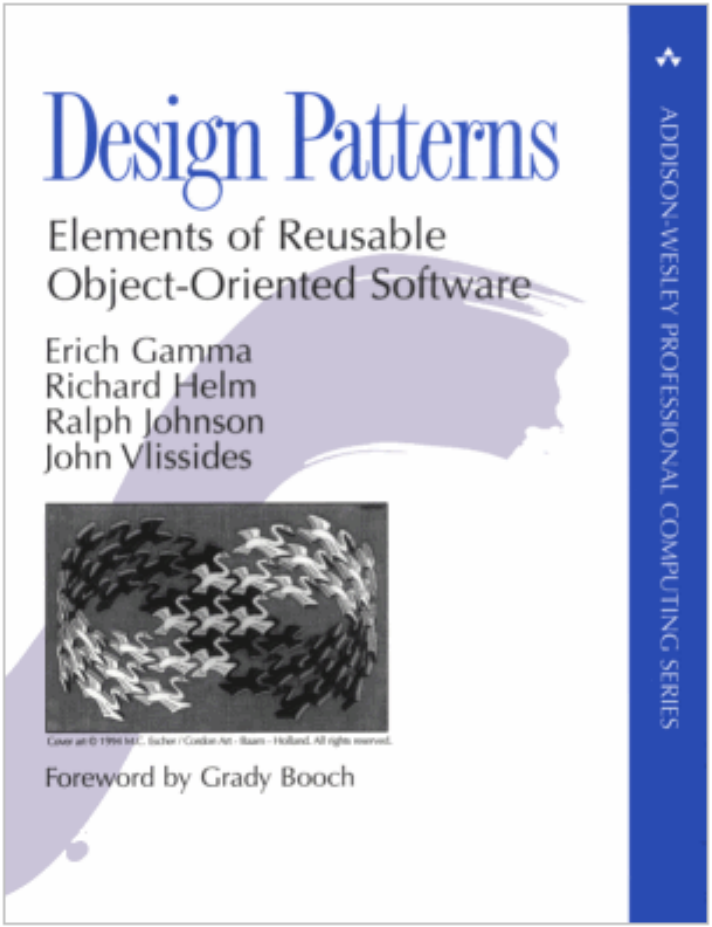
MSO
Design Patterns: Facade

Wouter Swierstra, Hans Philippi

September 11, 2018

Design Patterns



- The original Gang of Four book on Design Patterns is one of the best sold Computer Science books to date
- It introduces a structure to catalogue and describe design patterns in software
- It catalogues 23 such design patterns

Why study design patterns?

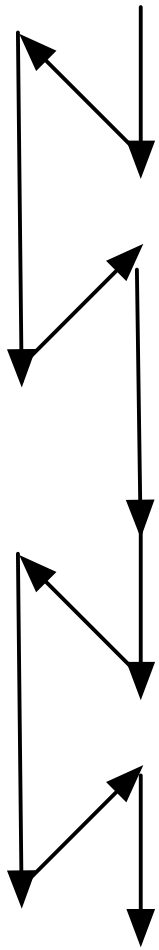
- Design patterns are *reusable solutions* to common problems: don't reinvent the wheel!
- It is good to learn from your failures, but it is better to learn from other people's failures
- Design patterns help establish a common terminology
- Design patterns give a higher perspective on analysis and design; using design patterns helps you abstract from irrelevant detail

Analogy: two carpenters designing a closet

Carpenter 1: How should we build the closet?

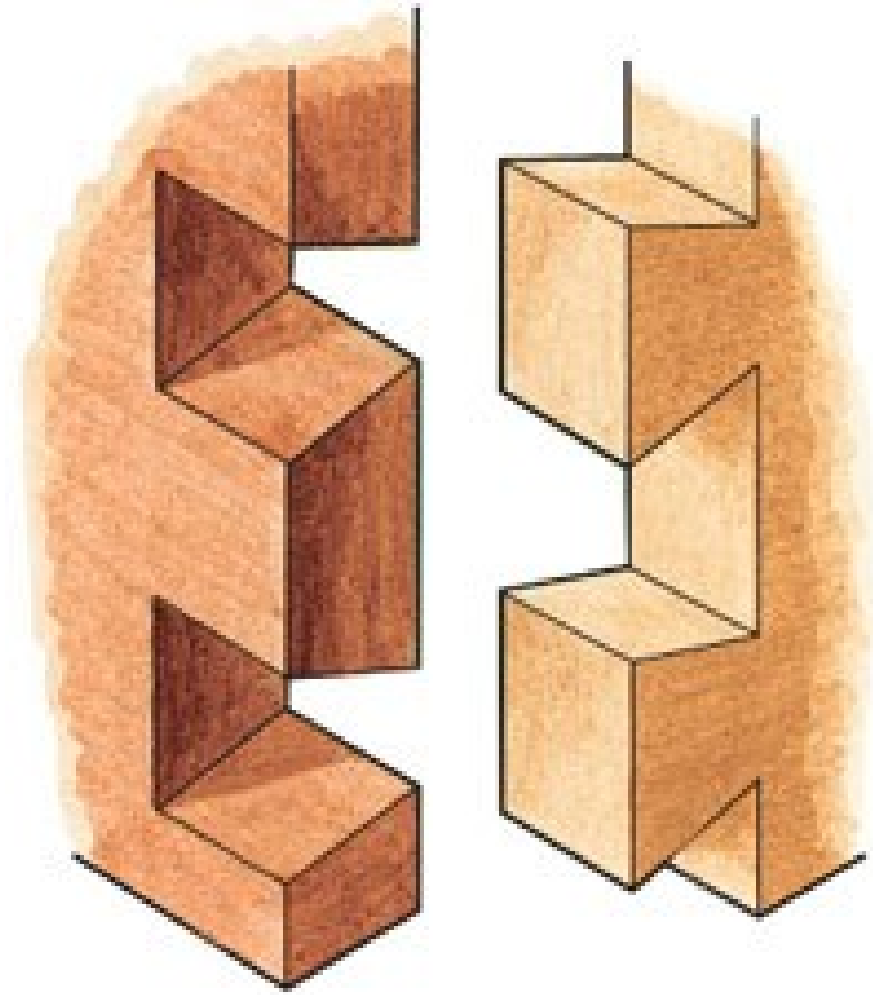
Carpenter 2: We should make the joint by cutting straight down into the wood, and then cut back up at 45 degrees, then down again, and the back up at -45 degrees, and then repeat this . . .

Dovetail joints



Carpenter 1: Oh, you mean a *dovetail joint*!

Dovetail joints



Think about software. . .

Designer A: I think we should implement this class with two methods, where the first method delegates some responsibility to another class, and the other method iterates over the aggregated. . .

Designer B: Should we use the Bridge or Strategy pattern?

Design patterns make communication easier

Why study design patterns?

- Most design patterns aim to make software easier to modify and maintain
- By *studying* and *understanding* design patterns, you also learn about good object-oriented design
- By studying design patterns you will learn to apply principles of object oriented design:
  - Design to interfaces
  - Find what varies and encapsulate it
  - Favour aggregation over inheritance

Learning about design patterns will make you a better designer!

Design patterns: structure

1. **Name** – each pattern has a unique name
2. **Intent** – the purpose of the pattern
3. **Problem** – the problem the pattern solves
4. **Solution** – the proposed solution
5. **Participants** – the entities involved in a pattern
6. **Consequences** – the consequences of applying the pattern; this typically investigates the forces that are at play
7. **Implementation** – how the pattern can be implemented
8. **Generic structure** – a standard (UML) diagram that shows the typical structure of the pattern

The Facade pattern

Finally, our first *design pattern*:

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

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

That sounds a bit abstract ...

Facade analogy

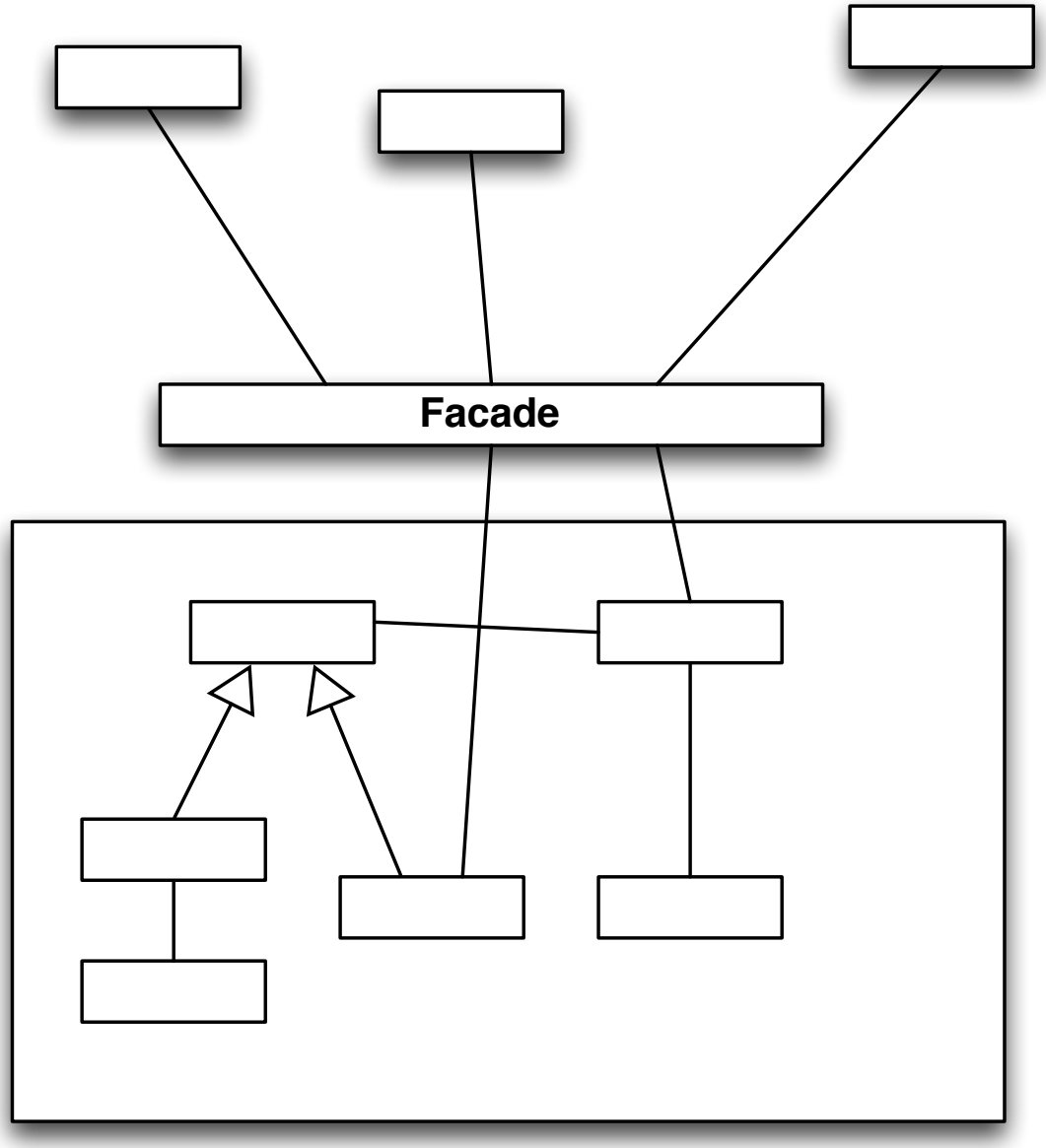
I bought a new smart TV with endless possibilities.
What do I do?

- I am going to study the 384 pages Complete User Manual and Reference to be sure that I will not miss any of its features now or in the future, or ...
- I will use the 3 pages Quick Start Guide and have fun within a few minutes

Facade: idea

- Big systems can be complicated
- But sometimes, your program only needs to use a subset of the functionality. Or it might only need to use the system in a limited way.
- **Idea:** Create your own *application programming interface* (API) that meets your specific needs
- In this way, you can hide the original system's complexity – your simple API is a *facade*

Facade: in pictures



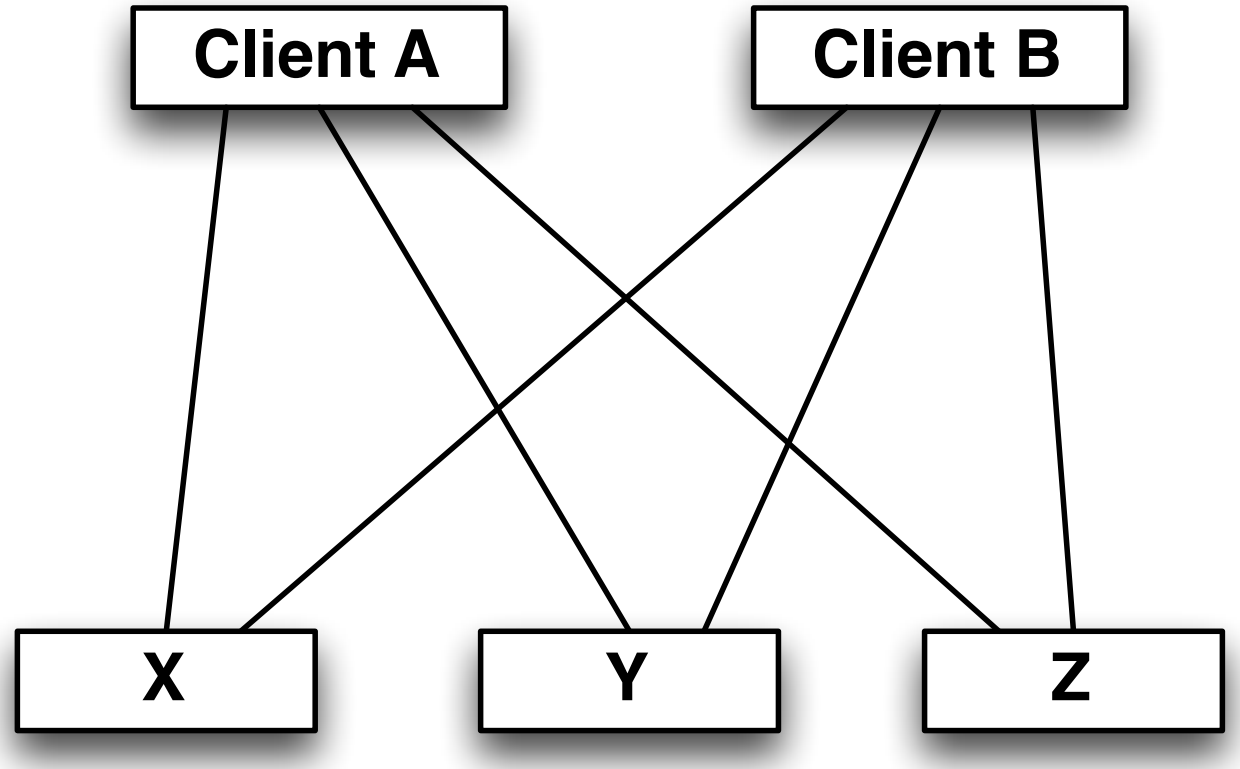
Facade pattern

- **Intent:** You want to simplify how to use an existing system. Or you need to define your own interface.
- **Problem:** You only need to use 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 clients of the existing system to use.
- **Consequences:** The Facade simplifies the use of the subsystem. As a result, it may lack certain functionality of the original system.
- **Implementation:** Define a new class (or set of classes) that has the required interface. Have this class use the existing system, but adapt all other classes to only communicate with the system through the new Facade.

Facade: variations

A Facade may not only reduce the complexity of the system, but it may also reduce the number of objects with which clients must interact.

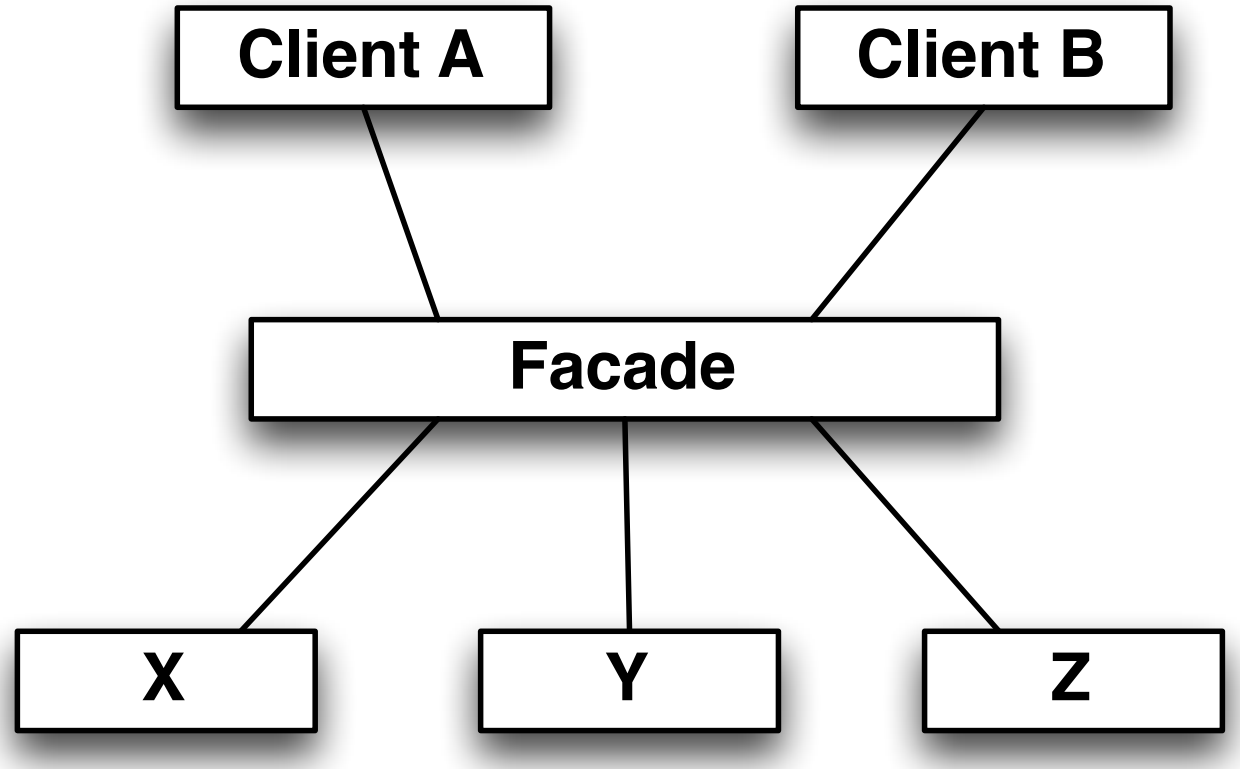
Before:



Facade: variations

A Facade may not only reduce the complexity of the system, but it may also reduce the number of objects with which clients must interact.

After:



Facade: variations

- A Facade may also introduce new functionality, capturing common communication patterns with the original system
- A Facade also *encapsulates* the original system
  - You can track system usage in the Facade
  - You can limit the interface exposed
- You can swap out systems, keeping the interface the same. All the changes required happen in one place: the Facade class.

Material covered and references

- Design Patterns Explained: chapters 3–8.
- Click here for Facade pattern on DoFactory