CPSC 310 – Software Engineering

Lecture 11

Design Patterns

Learning Goals

- Understand what are design patterns, their benefits and their drawbacks
- For at least the following design patterns: Singleton, Observer, Adapter, you will be able to describe them, know when to use them or not and give examples of situations where you could use them.

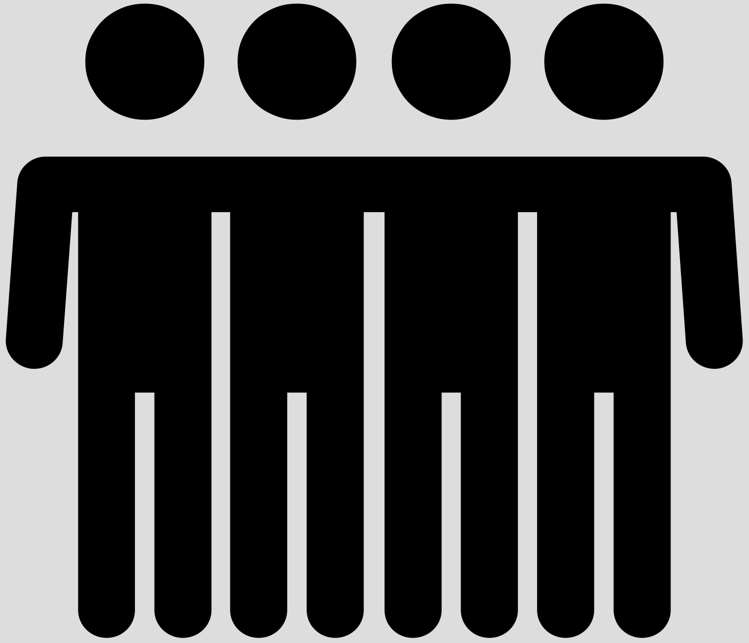
A bit of history

Christopher Alexander

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice"

"A pattern expresses a relation between a certain context, a problem and a solution"

A bit of history, continued



OOP World

The "Gang of Four" - GoF

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design patterns: elements of reusable object-oriented software

To be, or not to be

A Design Pattern IS:

- a way to benefit from the collective experience of skilled software developers
- an easy way to communicate about common problems

A Design Pattern IS NOT:

- the complete solution to your problem
- the only solution to your problem (but it's a proven one)
- something you should use if you do not understand it

Pattern Classification

Creational Patterns

- ℹ How an object can be created

Structural Patterns

- ℹ How objects can be composed

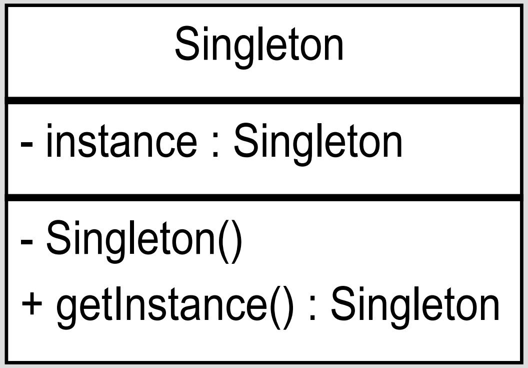
Behavioral Patterns

- ℹ How objects communicate

Singleton pattern

Intent: Make sure a class has only one instance, and provide a global point of access to it

Participants & Structure:



Singleton pattern

```java
public class Singleton {
    private static Singleton instance = null;
    private Singleton() { }
    public static synchronized Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
}
```

Singleton pattern

```
public class Singleton {
    private static Singleton instance = null;
    private Singleton() { }
    public static synchronized Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
}
```

Singleton pattern

To which category belongs this pattern ?

Creational Patterns

ℹ How an object can be created

Structural Patterns

ℹ How objects can be composed

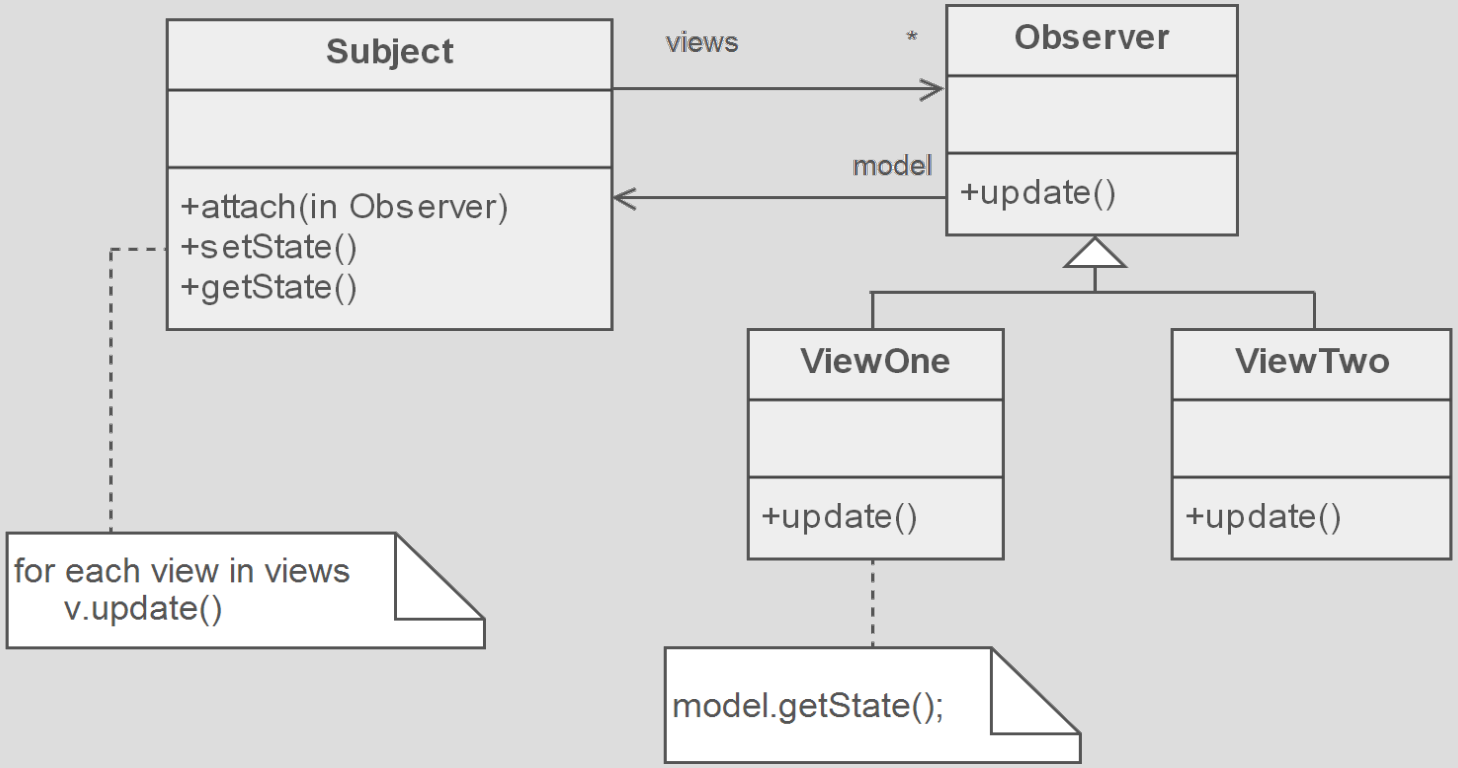
Behavioral Patterns

ℹ How objects communicate

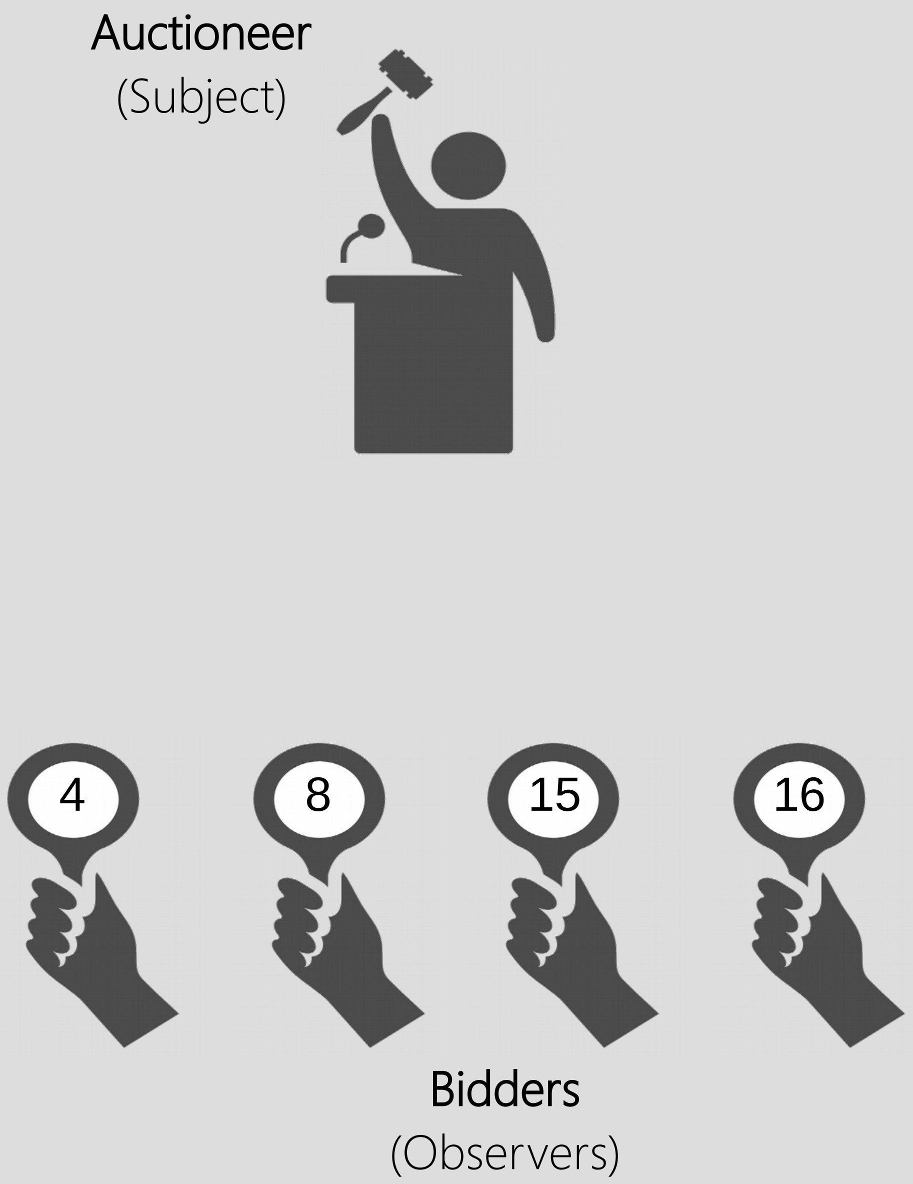
Observer pattern

Intent: Ensure that, when an object changes his state, all its dependents are notified and updated automatically

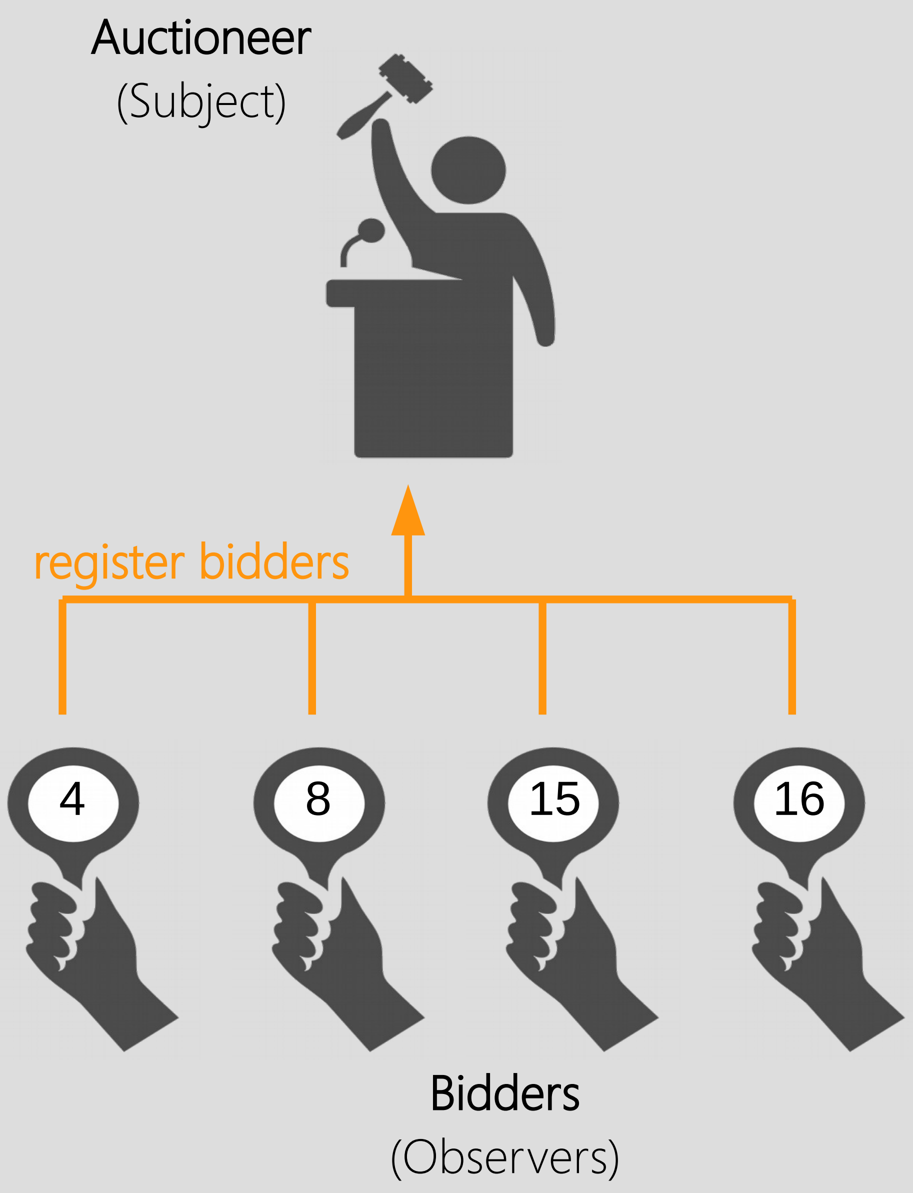
Participants & Structure:

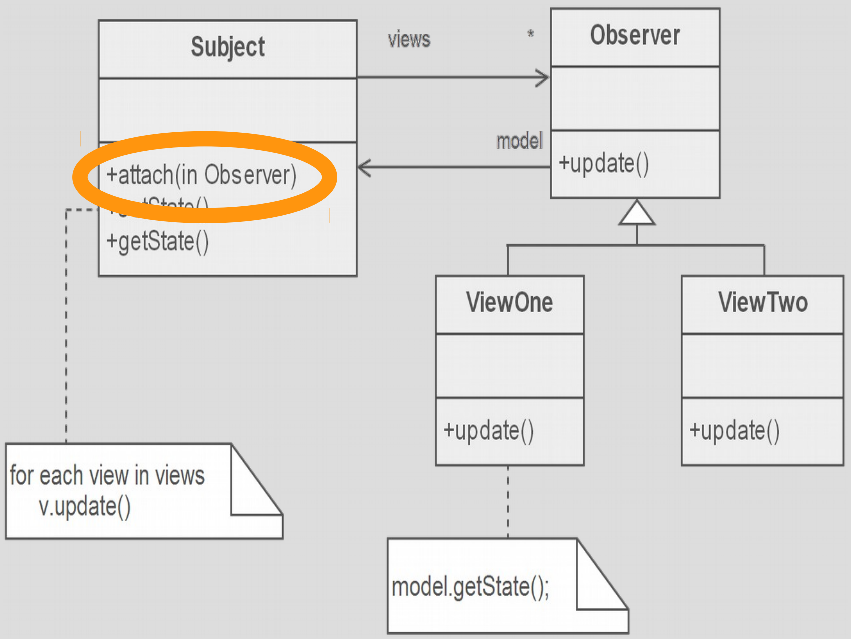


Observer pattern [example]

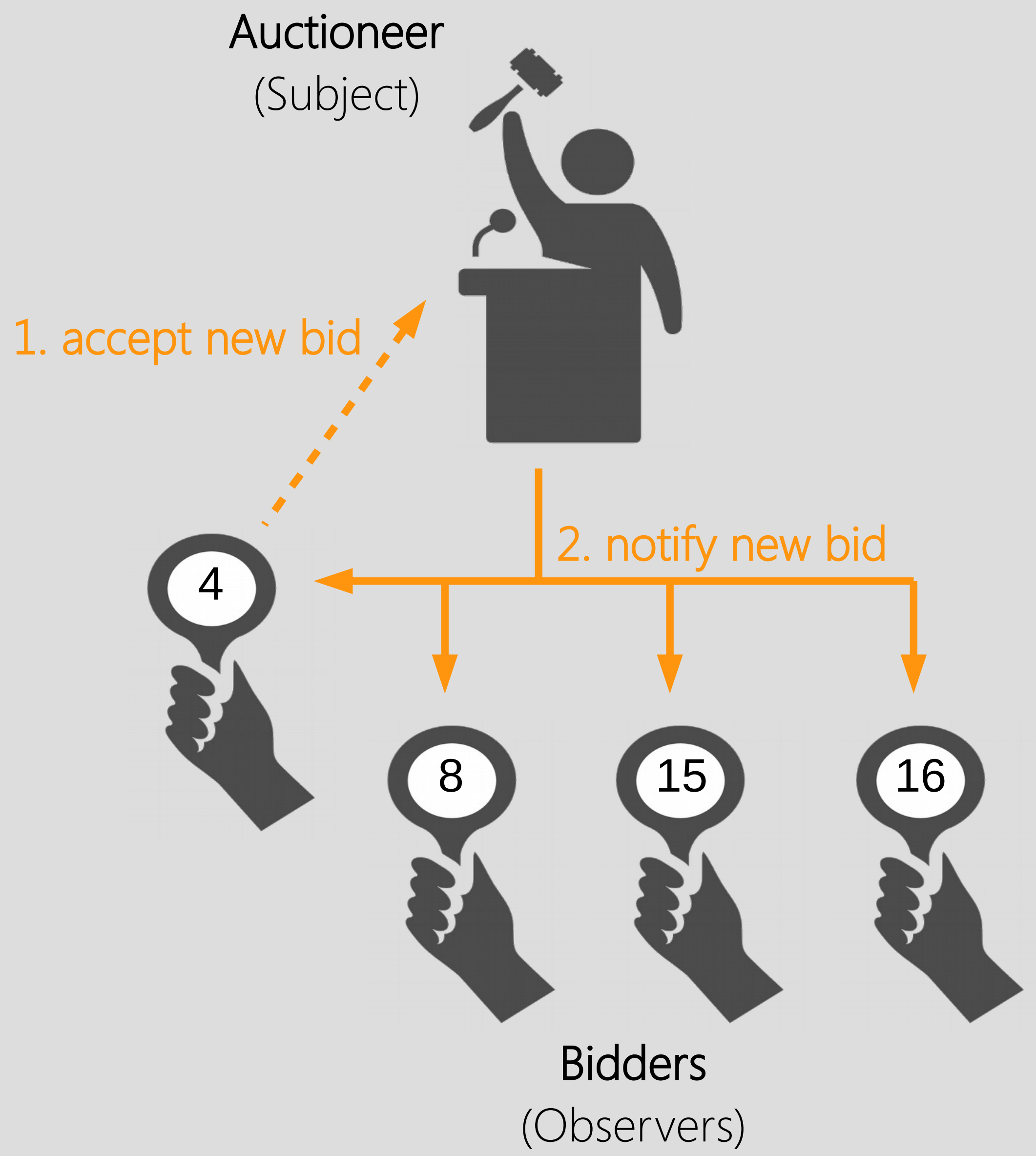


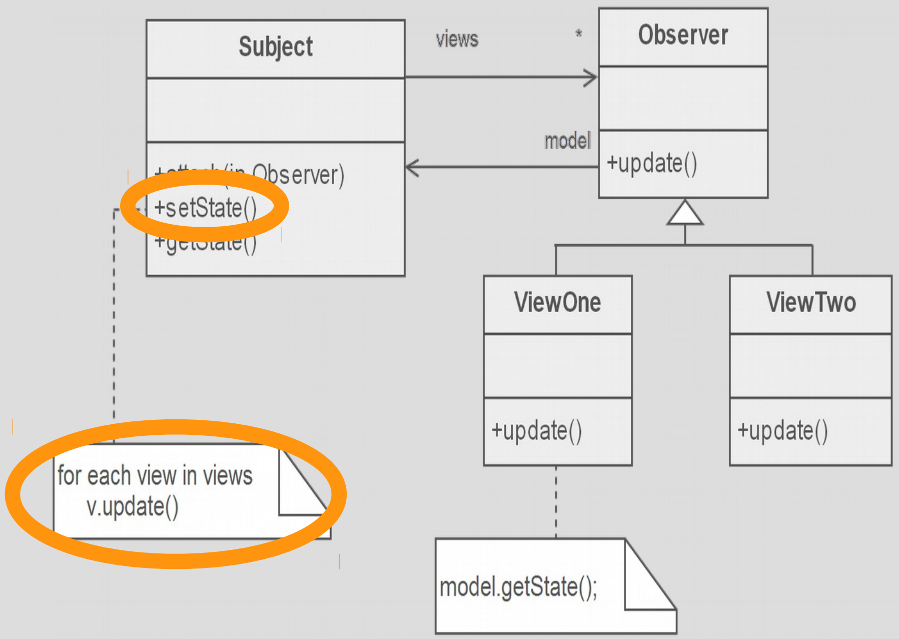
Observer pattern [example]





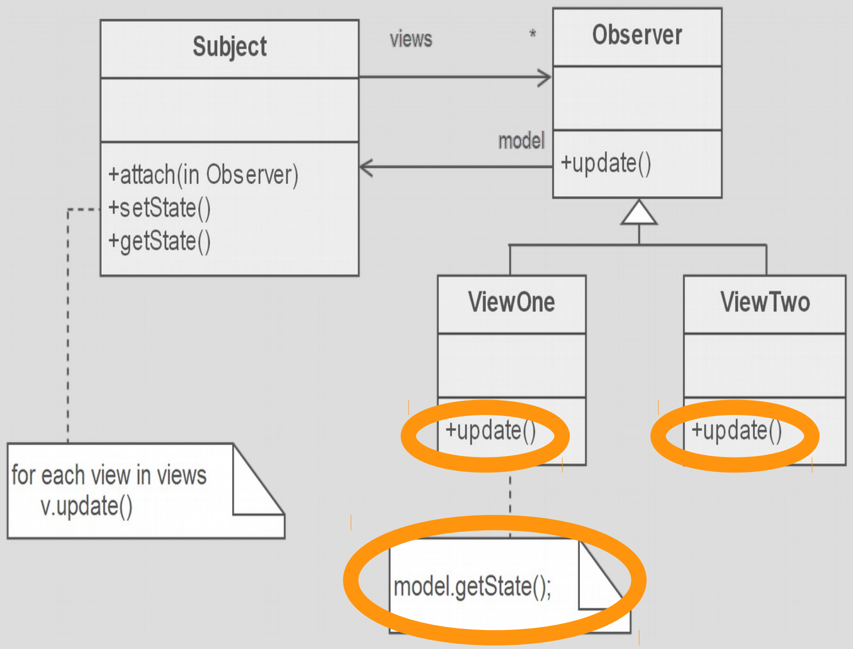
Observer pattern [example]





Observer pattern [example]

Auctioneer
(Subject)
retrieve
new price
4
8
15
16
Bidders
(Observers)



Observer pattern

To which category belongs this pattern ?

Creational Patterns

- How an object can be created

Structural Patterns

- How objects can be composed

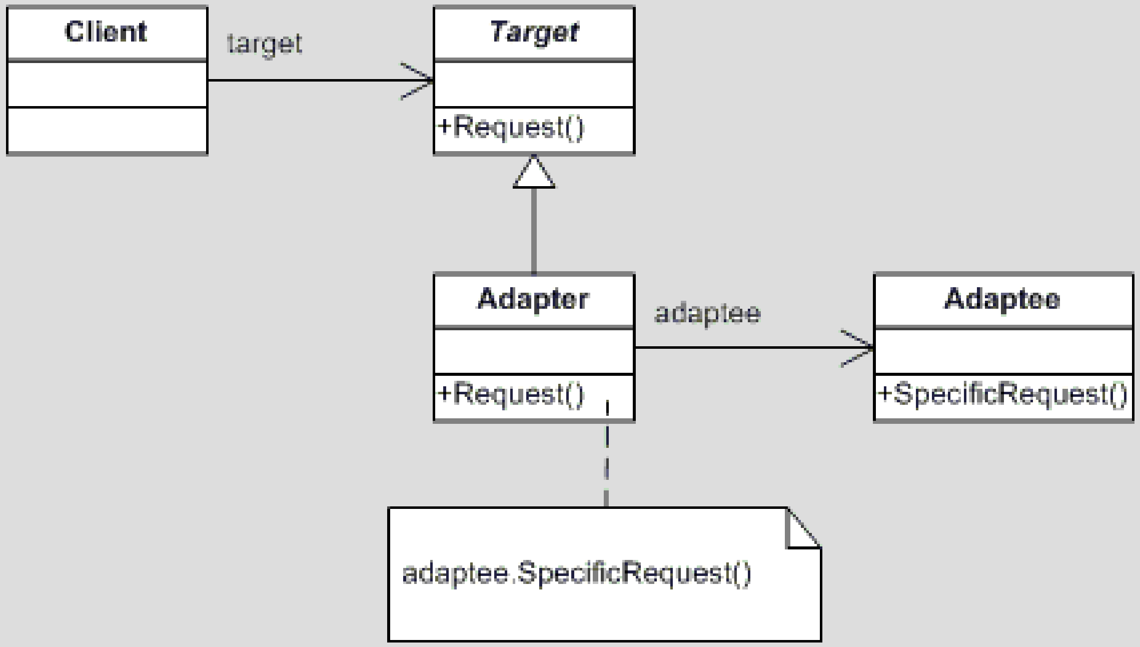
Behavioral Patterns

- How objects communicate

Adapter pattern

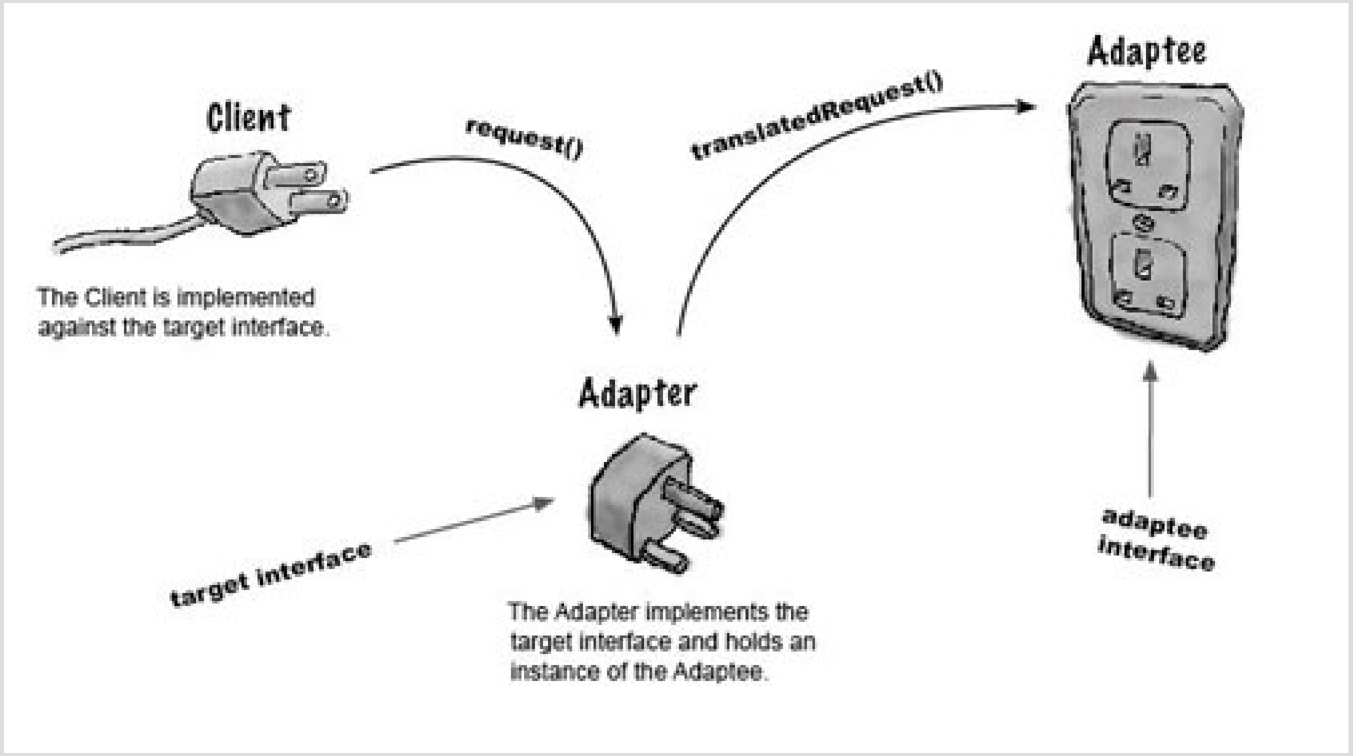
Intent: Convert the interface of a class into another interface that clients expect.

Participants & Structure:



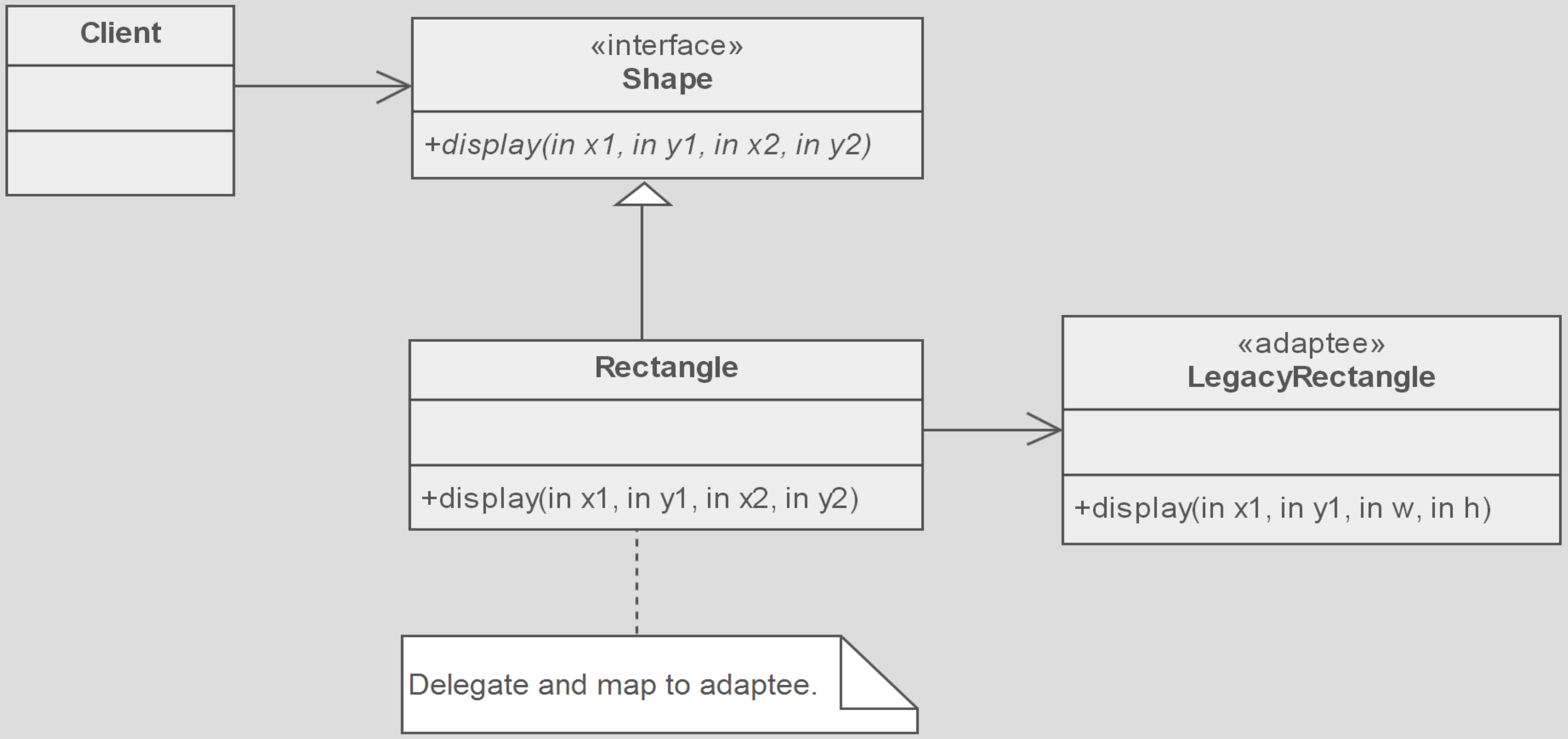
Adapter pattern

Power supply adapter analogy


Client
request()
translatedRequest()
Adaptee
The Client is implemented against the target interface.
Adapter
target interface
adaptee interface
The Adapter implements the target interface and holds an instance of the Adaptee.

Adapter pattern

Concrete example


Client
«interface»
Shape
+display(in x1, in y1, in x2, in y2)
Rectangle
+display(in x1, in y1, in x2, in y2)
«adaptee»
LegacyRectangle
+display(in x1, in y1, in w, in h)
Delegate and map to adaptee.

Adapter pattern

To which category belongs this pattern ?

Creational Patterns

ℹ How an object can be created

Structural Patterns

ℹ How objects can be composed

Behavioral Patterns

ℹ How objects communicate

Design Pattern Collection

Factory
Singleton
Decorator
Proxy
Template
Composite
Adapter
Observer

http://sourcemaking.com/design_patterns

Anti Pattern

- A bad solution to a recurring problem
- "Strong" code smell
- A good pattern in the wrong context can lead to an anti-pattern

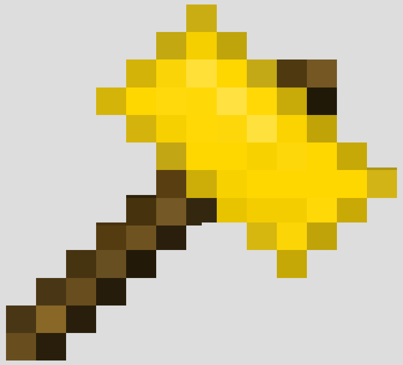
Anti Pattern

- Several categories
  - Development
  - Architecture
  - Management

http://c2.com/cgi/wiki?AntiPatternsCatalog

http://sourcemaking.com/antipatterns

Eg: Golden Hammer



- Problem: You need to choose technologies for your development, and you are of the belief that you must choose exactly one technology to dominate the architecture.
- Context: You need to develop some new system or piece of software that doesn't fit very well the technology that the development team are familiar with.
- Forces:
  - The development team are committed to the technology they know
  - The development team are not familiar with other technologies
  - Other, unfamiliar, technologies are seen as risky
  - It is easy to plan and estimate for development in the familiar technology
- **Supposed Solution**: Use the familiar technology anyway. The technology (or concept) is applied obsessively to many problems, including where it is clearly inappropriate.
- **Refactored Solution**: Expanding the knowledge of developers through education, training, and book study groups exposes developers to new solutions.

http://sourcemaking.com/antipatterns/golden-hammer

Eg. Singleton Overuse

- Problems
  - violate information hiding since dependencies are hidden in the code and not expressed in the interface
  - can cause high coupling
- You must have a good damn reason to use it
  - The fact that you know it is not enough

```
public void someMethod() ...
Profile.getInstance().getUserLevel()
```

```
public void someMethod(Profile profile) ...
profile.getUserLevel()
```

Design Pattern Drawbacks

- Can make the design more complex if not needed
  - **Start simple** and then **refactor** by using a design pattern if it is justified
  - Do not try to anticipate future needs too much
- Can lead to bad design if not applied in the right context