Log in   Contact

Home › Design Patterns › Structural patterns

Adapter Design Pattern

[+] Feedback

BOOKMARK

Search ✕

Why read if you can watch?

[Watch design patterns video tutorial](#)

contents
Design Patterns

Intent

01 ○ Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

○ Wrap an existing class with a new interface.

○ Impedance match an old component to a new system

Problem

02 An "off the shelf" component offers compelling functionality that you would like to reuse, but its "view of the world" is not compatible with the philosophy and architecture of the system currently being developed.

Discussion

03   Reuse has always been painful and elusive. One reason has been the tribulation of designing something new, while reusing something old. There is always something not quite right between the old and the new. It may be physical dimensions or misalignment. It may be timing or synchronization. It may be unfortunate assumptions or competing standards.

04   It is like the problem of inserting a new three-prong electrical plug in an old two-prong wall outlet – some kind of adapter or intermediary is necessary.

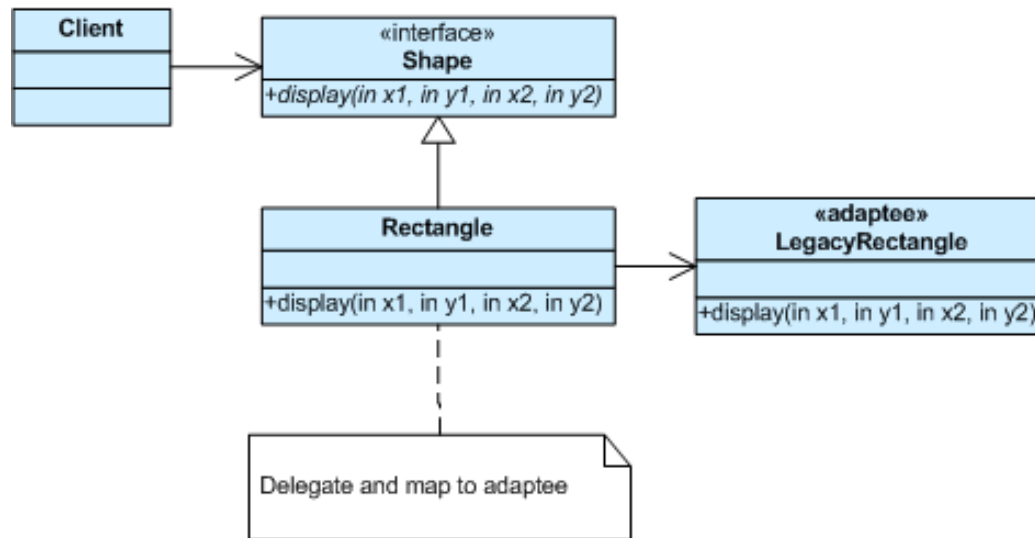05   Adapter is about creating an intermediary abstraction that translates, or maps, the old component to the new system. Clients call methods on the Adapter object which redirects them into calls to the legacy component. This strategy can be implemented either with inheritance or with aggregation.

06   Adapter functions as a wrapper or modifier of an existing class. It provides a different or translated view of that class.
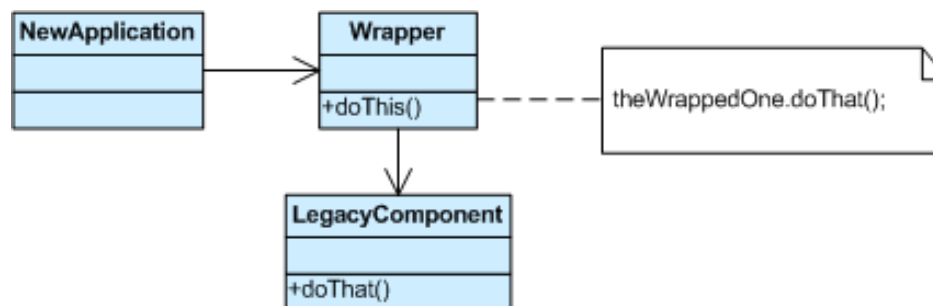
Structure

07 Below, a legacy Rectangle component's `display()` method expects to receive "x, y, w, h" parameters. But the client wants to pass "upper left x and y" and "lower right x and y". This incongruity can be reconciled by adding an additional level of indirection – i.e. an Adapter object.

```
┌─────────────┐        ┌─────────────────────────────┐
│   Client    │        │        «interface»          │
├─────────────┤───────▷│          Shape              │
├─────────────┤        ├─────────────────────────────┤
└─────────────┘        │+display(in x1, in y1, in x2, in y2)│
                       └─────────────────────────────┘
                                      △
                                      │
                 ┌──────────────────────────┐        ┌──────────────────────────────┐
                 │        Rectangle         │        │         «adaptee»            │
                 ├──────────────────────────┤───────▷│        LegacyRectangle       │
                 ├──────────────────────────┤        ├──────────────────────────────┤
                 │+display(in x1, in y1, in x2, in y2)│  │+display(in x1, in y1, in x2, in y2)│
                 └──────────────────────────┘        └──────────────────────────────┘
                               ┆
                 ┌──────────────────────────┐
                 │ Delegate and map to adaptee │
                 └──────────────────────────┘
```

08 The Adapter could also be thought of as a "wrapper".

```
┌─────────────┐     ┌─────────────┐
│NewApplication│    │   Wrapper   │
├─────────────┤────▷├─────────────┤─ ─ ─ ─ ┌──────────────────────────┐
├─────────────┤     │+doThis()    │        │ theWrappedOne.doThat();  │
└─────────────┘     └─────────────┘        └──────────────────────────┘
                          │
                          ▽
                   ┌─────────────┐
                   │LegacyComponent│
                   ├─────────────┤
                   │+doThat()    │
                   └─────────────┘
```

Example

The Adapter pattern allows otherwise incompatible classes to work together by converting the interface of one class into an interface expected by the clients. Socket wrenches provide an example of the Adapter. A socket attaches to a ratchet, provided that the size of the drive is the same. Typical drive sizes in the United States are 1/2" and 1/4". Obviously, a 1/2" drive ratchet will not fit into a 1/4" drive socket unless an adapter is used. A 1/2" to 1/4" adapter has a 1/2" female connection to fit on the 1/2" drive ratchet, and a 1/4" male connection to fit in the 1/4" drive socket.
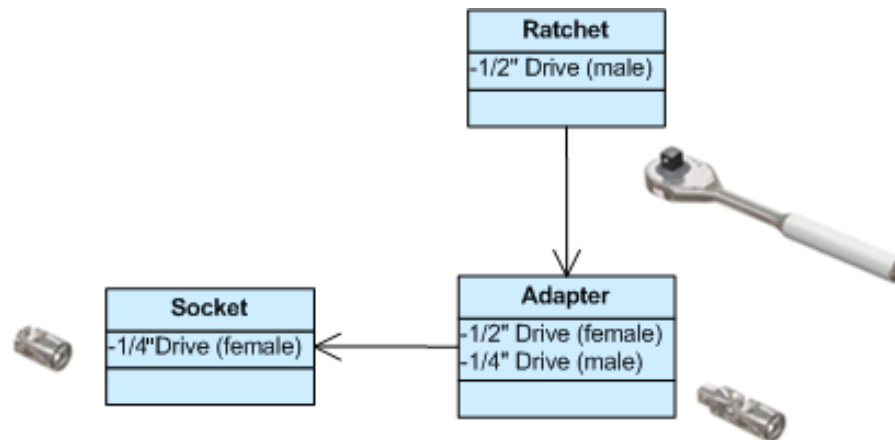
Check list

1. Identify the players: the component(s) that want to be accommodated (i.e. the

client), and the component that needs to adapt (i.e. the adaptee).

2. Identify the interface that the client requires.

3. Design a "wrapper" class that can "impedance match" the adaptee to the client.

4. The adapter/wrapper class "has a" instance of the adaptee class.

5. The adapter/wrapper class "maps" the client interface to the adaptee interface.

6. The client uses (is coupled to) the new interface

Rules of thumb

- Adapter makes things work after they're designed; Bridge makes them work before they are.

- Bridge is designed up-front to let the abstraction and the implementation vary independently. Adapter is retrofitted to make unrelated classes work together.

- Adapter provides a different interface to its subject. Proxy provides the same interface. Decorator provides an enhanced interface.

- Adapter is meant to change the interface of an existing object. Decorator enhances another object without changing its interface. Decorator is thus more transparent to the application than an adapter is. As a consequence, Decorator supports recursive composition, which isn't possible with pure Adapters.

- Facade defines a new interface, whereas Adapter reuses an old interface. Remember that Adapter makes two existing interfaces work together as opposed to defining an entirely new one.

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

Adapter code examples

C# examples

- Adapter in C#

C++ examples

- Adapter in C++
- Adapter in C++: External Polymorphism

Delphi examples

- Adapter in Delphi

Java examples

- Adapter in Java: Before and after
- Adapter in Java

PHP examples

- Adapter in PHP

‹ Structural patterns   Bridge Design Pattern ›