Adapter

Design Pattern

Branga Andrei - 10 June 2018

Intent

- 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.

Adapter pattern works as a bridge between two incompatible interfaces. This type of design pattern comes under structural pattern as this pattern combines the capability of two independent interfaces.
This pattern involves a single class which is responsible to join functionalities of independent or incompatible interfaces. A real life example could be a case of card reader which acts as an adapter between memory card and a laptop. You plugin the memory card into card reader and card reader into the laptop so that memory card can be read via laptop.

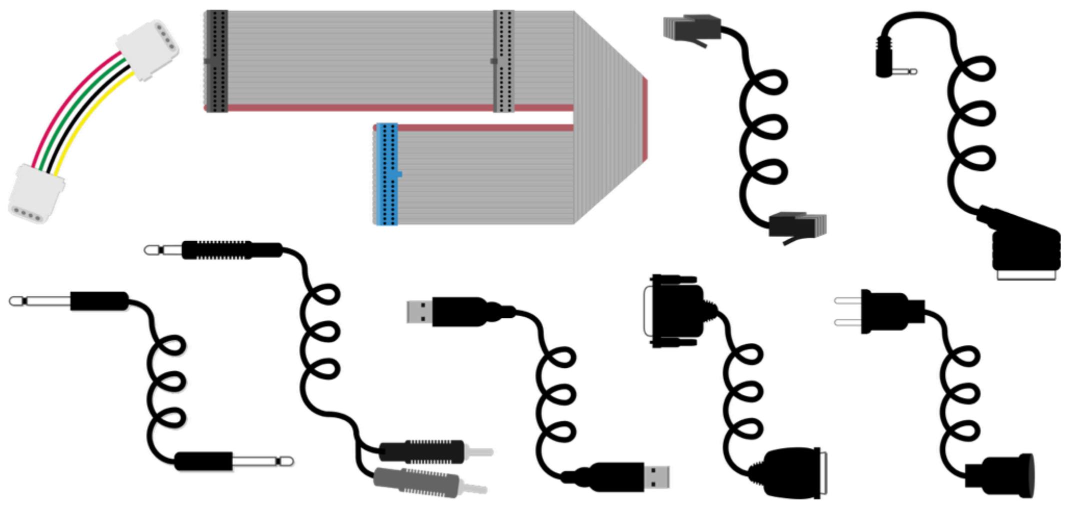
Explanation

Problem

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.

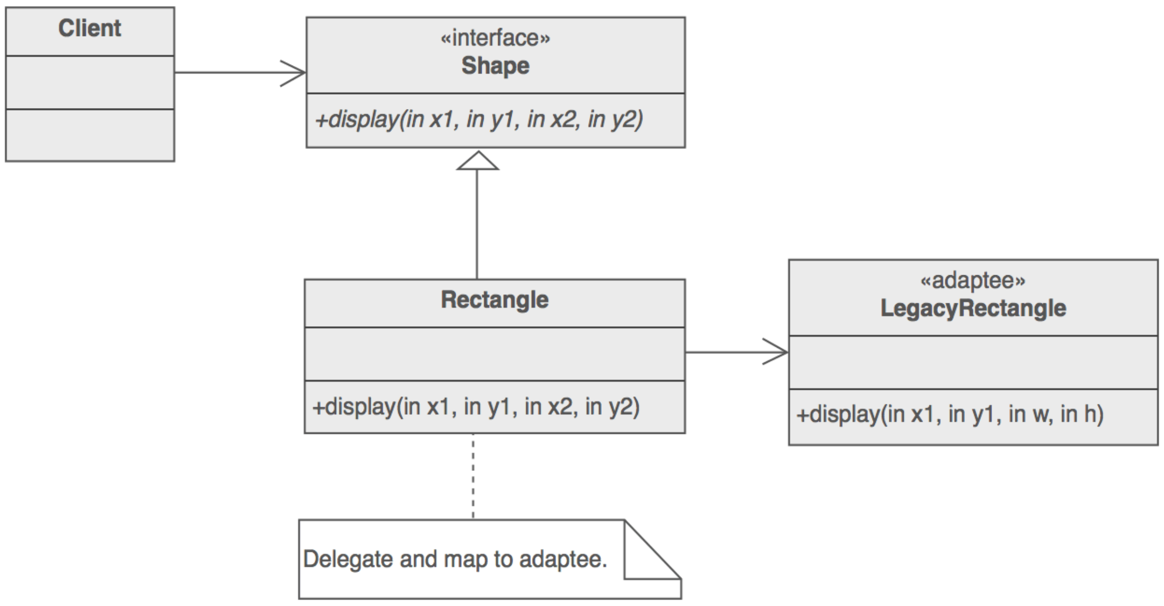
Solving / Discussion

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 synchronisation.
It may be unfortunate assumptions or competing standards.
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.

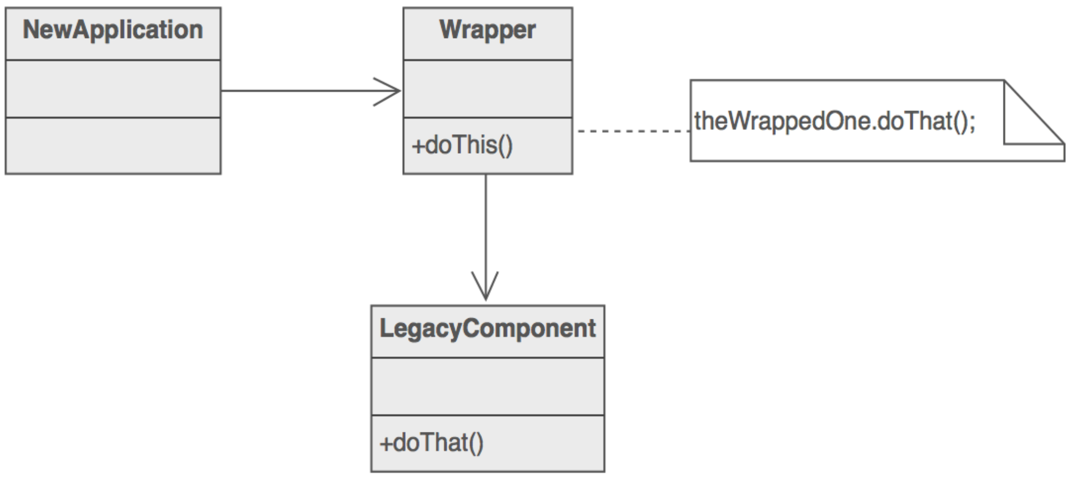


Adapter is about creating an intermediary abstraction that translates, or maps, the old component to the new system. Clients call methods on the Adapter object which redirects them into calls to the legacy component. This strategy can be implemented either with inheritance or with aggregation.
Adapter functions as a wrapper or modifier of an existing class. It provides a different or translated view of that class.

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
+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.

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



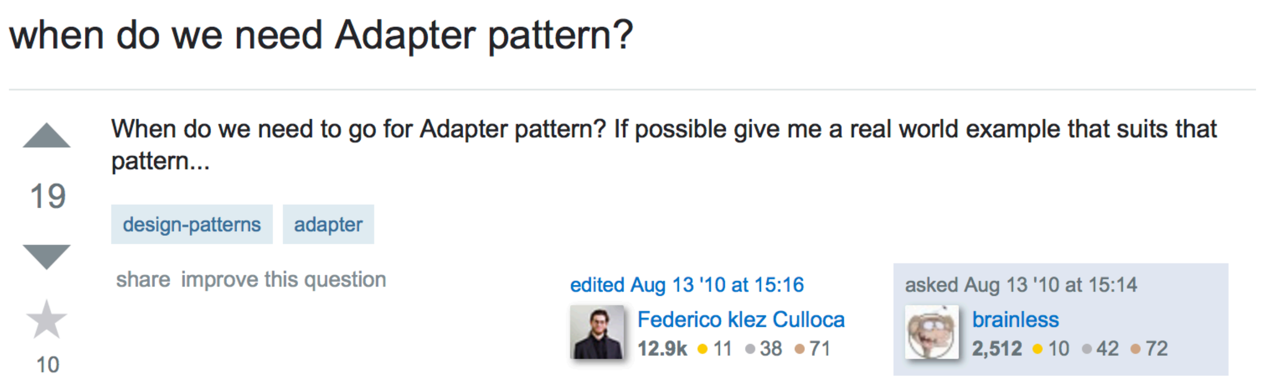
https://sourcemaking.com/design_patterns/adapter - SourceMaking

https://www.tutorialspoint.com/design_pattern/adapter_pattern.htm - TutorialsPoint

Related patterns

- 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.

Common situations of use



when do we need Adapter pattern?

19

When do we need to go for Adapter pattern? If possible give me a real world example that suits that pattern...

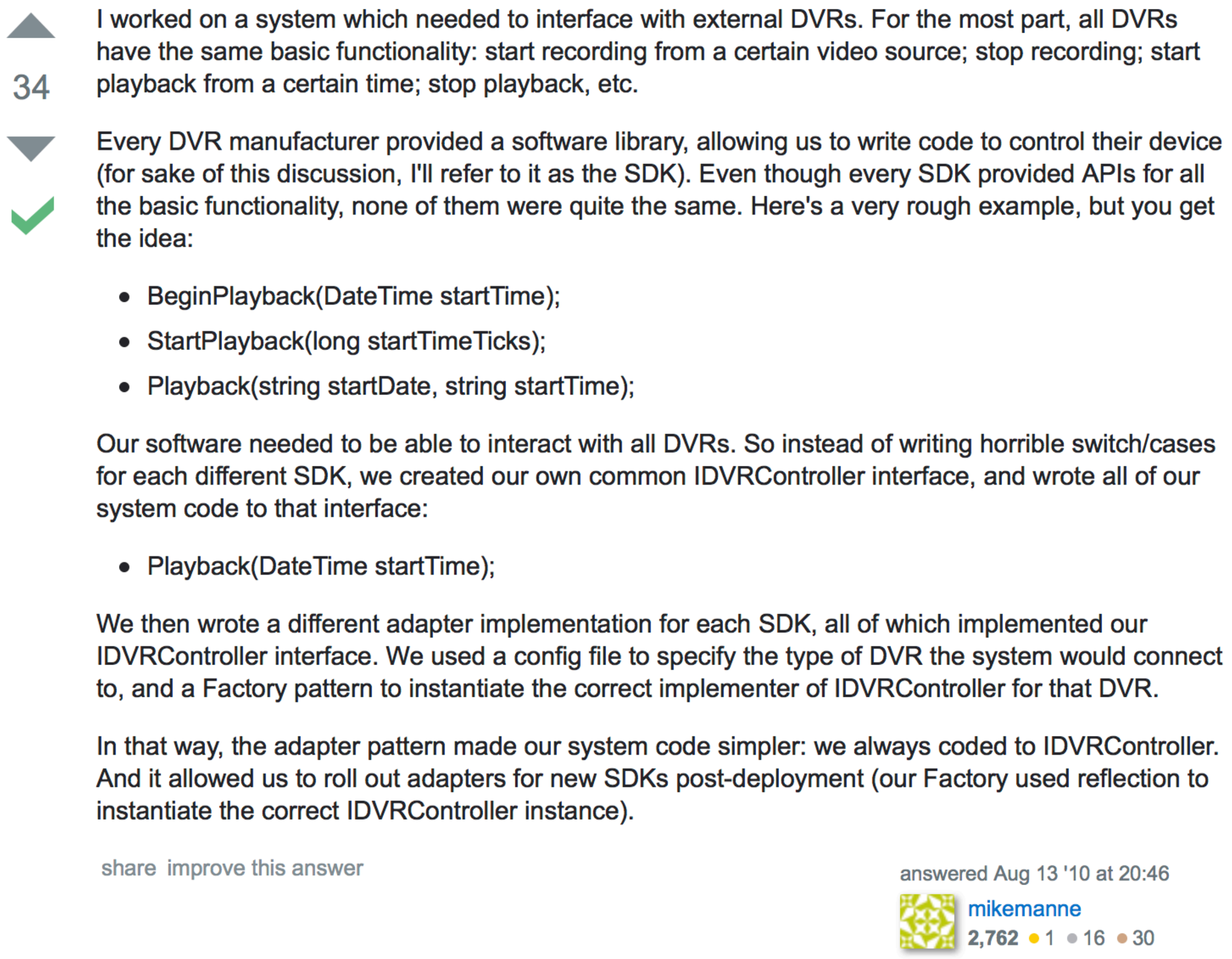
design-patterns adapter

share improve this question

10

edited Aug 13 '10 at 15:16
Federico klez Culloca
12.9k ● 11 ● 38 ● 71

asked Aug 13 '10 at 15:14
brainless
2,512 ● 10 ● 42 ● 72



34

I worked on a system which needed to interface with external DVRs. For the most part, all DVRs have the same basic functionality: start recording from a certain video source; stop recording; start playback from a certain time; stop playback, etc.

Every DVR manufacturer provided a software library, allowing us to write code to control their device (for sake of this discussion, I'll refer to it as the SDK). Even though every SDK provided APIs for all the basic functionality, none of them were quite the same. Here's a very rough example, but you get the idea:

- BeginPlayback(DateTime startTime);
- StartPlayback(long startTimeTicks);
- Playback(string startDate, string startTime);

Our software needed to be able to interact with all DVRs. So instead of writing horrible switch/cases for each different SDK, we created our own common IDVRController interface, and wrote all of our system code to that interface:

- Playback(DateTime startTime);

We then wrote a different adapter implementation for each SDK, all of which implemented our IDVRController interface. We used a config file to specify the type of DVR the system would connect to, and a Factory pattern to instantiate the correct implementer of IDVRController for that DVR.

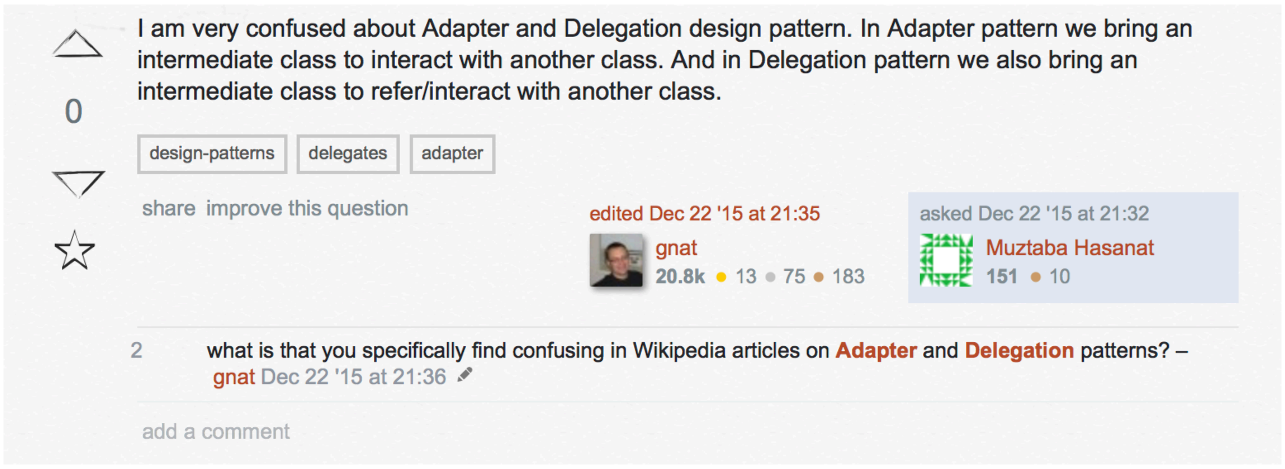
In that way, the adapter pattern made our system code simpler: we always coded to IDVRController. And it allowed us to roll out adapters for new SDKs post-deployment (our Factory used reflection to instantiate the correct IDVRController instance).

share improve this answer

answered Aug 13 '10 at 20:46
mikemanne
2,762 ● 1 ● 16 ● 30

https://stackoverflow.com/questions/3478225/when-do-we-need-adapter-pattern - StackOverflow

Confusions



0

I am very confused about Adapter and Delegation design pattern. In Adapter pattern we bring an intermediate class to interact with another class. And in Delegation pattern we also bring an intermediate class to refer/interact with another class.

design-patterns delegates adapter

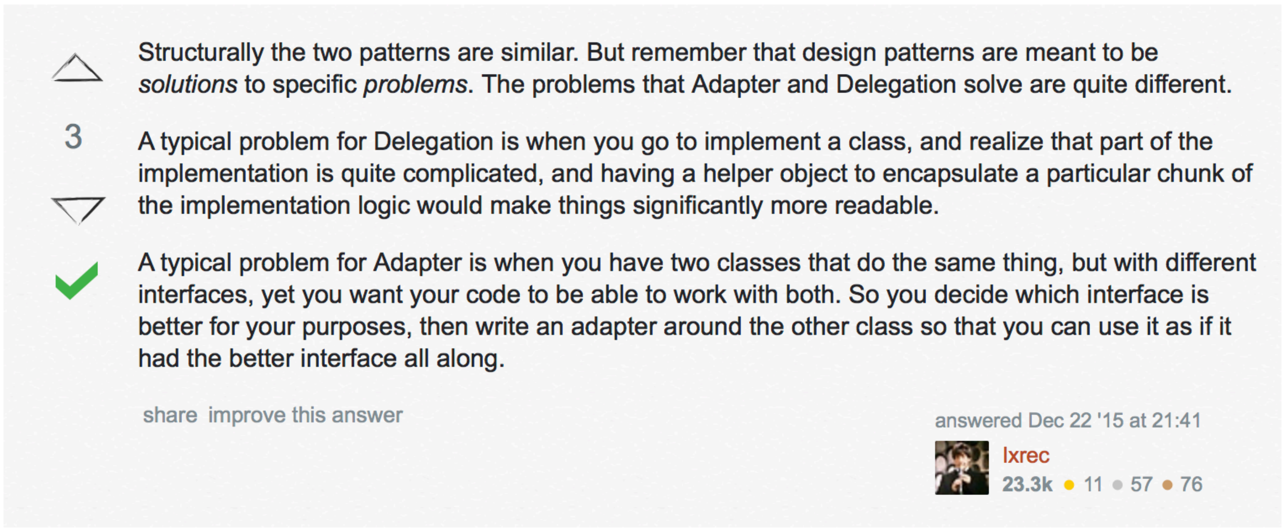
share improve this question

edited Dec 22 '15 at 21:35
gnat
20.8k ● 13 ● 75 ● 183

asked Dec 22 '15 at 21:32
Muztaba Hasanat
151 ● 10

2 what is that you specifically find confusing in Wikipedia articles on **Adapter** and **Delegation** patterns? – gnat Dec 22 '15 at 21:36

add a comment



3

Structurally the two patterns are similar. But remember that design patterns are meant to be *solutions* to specific *problems*. The problems that Adapter and Delegation solve are quite different.

A typical problem for Delegation is when you go to implement a class, and realize that part of the implementation is quite complicated, and having a helper object to encapsulate a particular chunk of the implementation logic would make things significantly more readable.

A typical problem for Adapter is when you have two classes that do the same thing, but with different interfaces, yet you want your code to be able to work with both. So you decide which interface is better for your purposes, then write an adapter around the other class so that you can use it as if it had the better interface all along.

share improve this answer

answered Dec 22 '15 at 21:41
Ixrec
23.3k ● 11 ● 57 ● 76

https://softwareengineering.stackexchange.com/questions/305676/what-is-the-difference-between-adapter-and-delegation-design-pattern - StackExchange