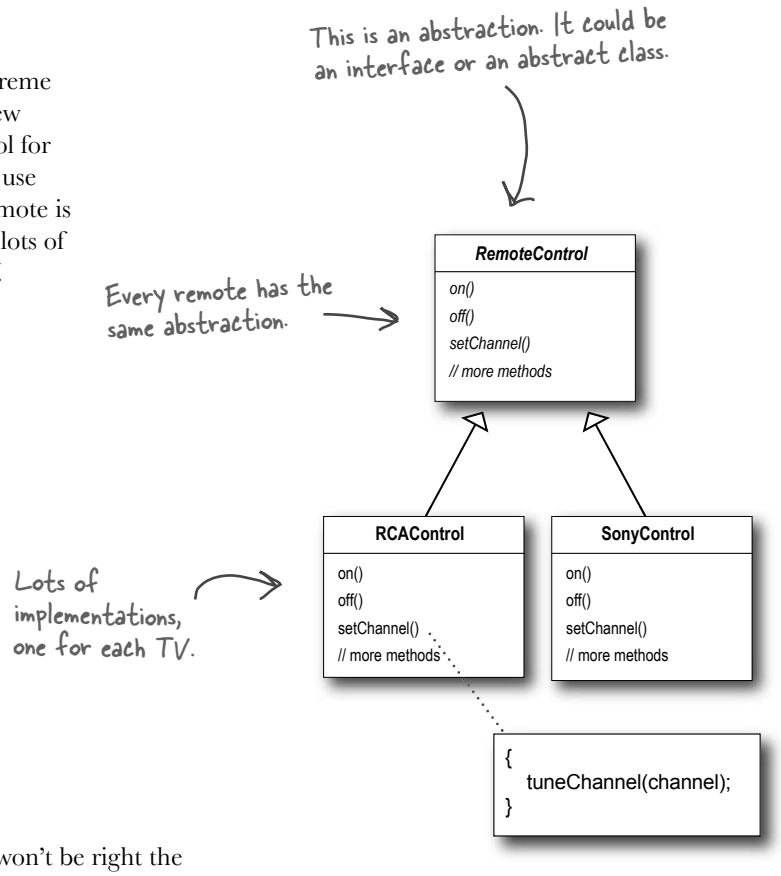
Bridge

Use the Bridge Pattern to vary not only your implementations, but also your abstractions.

A scenario

Imagine you're going to revolutionize "extreme lounging." You're writing the code for a new ergonomic and user-friendly remote control for TVs. You already know that you've got to use good OO techniques because while the remote is based on the same *abstraction*, there will be lots of *implementations* – one for each model of TV.



Your dilemma

You know that the remote's user interface won't be right the first time. In fact, you expect that the product will be refined many times as usability data is collected on the remote control.

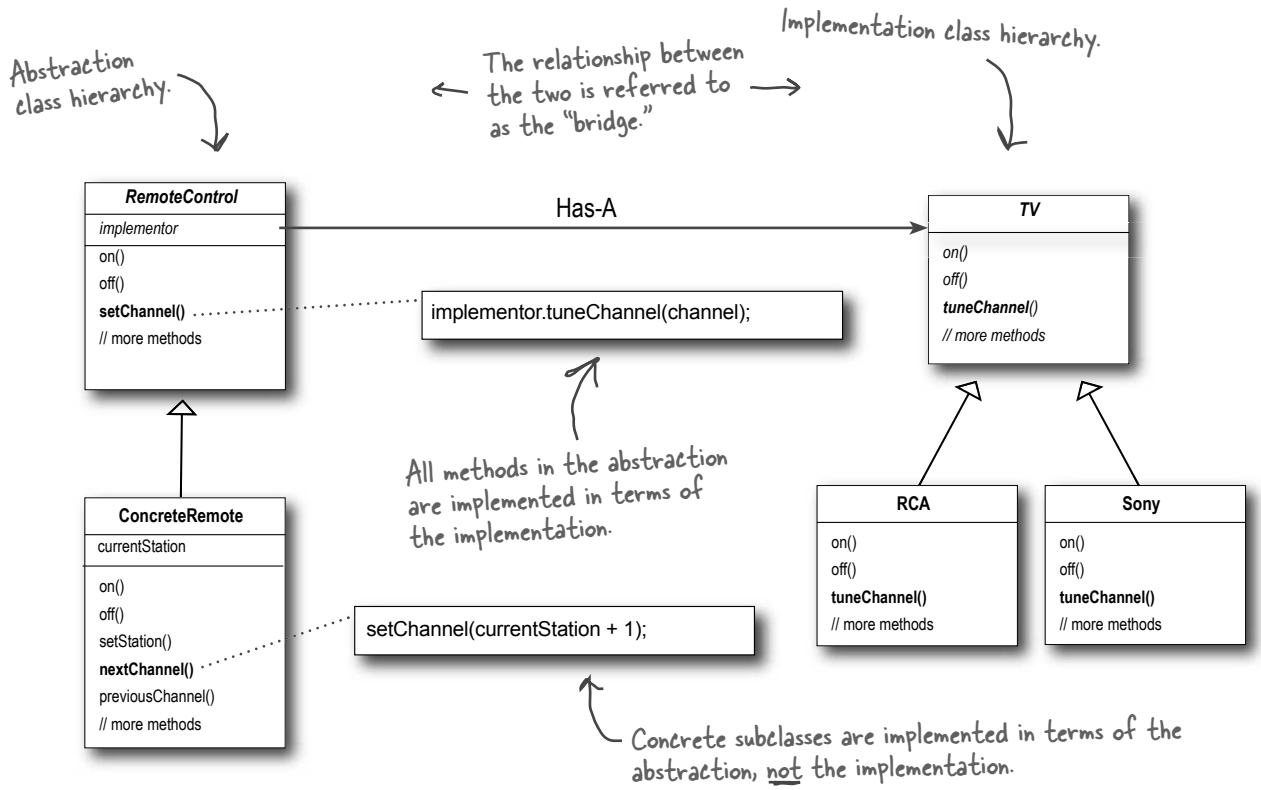
So your dilemma is that the remotes are going to change and the TVs are going to change. You've already *abstracted* the user interface so that you can vary the *implementation* over the many TVs your customers will own. But you are also going to need to *vary the abstraction* because it is going to change over time as the remote is improved based on the user feedback.

So how are you going to create an OO design that allows you to vary the implementation *and* the abstraction?

Using this design we can vary only the TV implementation, not the user interface.

Why use the Bridge Pattern?

The Bridge Pattern allows you to vary the implementation *and* the abstraction by placing the two in separate class hierarchies.



Now you have two hierarchies, one for the remotes and a separate one for platform specific TV implementations. The bridge allows you to vary either side of the two hierarchies independently.

Bridge Benefits

- Decouples an implementation so that it is not bound permanently to an interface.
- Abstraction and implementation can be extended independently.
- Changes to the concrete abstraction classes don't affect the client.

Bridge Uses and Drawbacks

- Useful in graphic and windowing systems that need to run over multiple platforms.
- Useful any time you need to vary an interface and an implementation in different ways.
- Increases complexity.