

Lights, Camera, Facade!

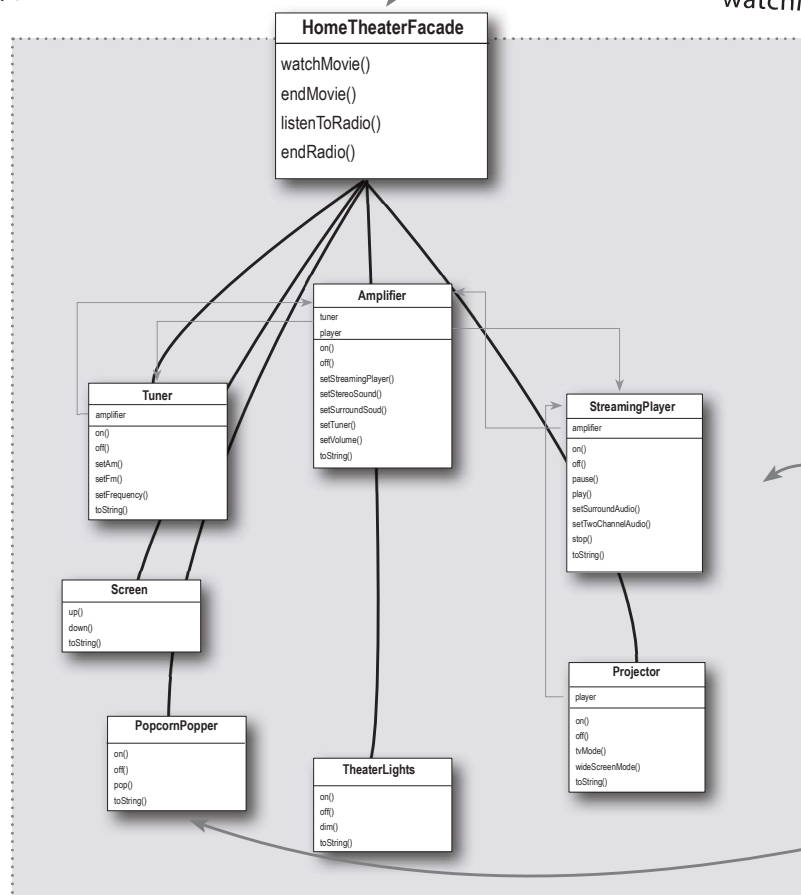
A Facade is just what you need: with the Facade Pattern you can take a complex subsystem and make it easier to use by implementing a Facade class that provides one, more reasonable interface. Don't worry; if you need the power of the complex subsystem, it's still there for you to use, but if all you need is a straightforward interface, the Facade is there for you.

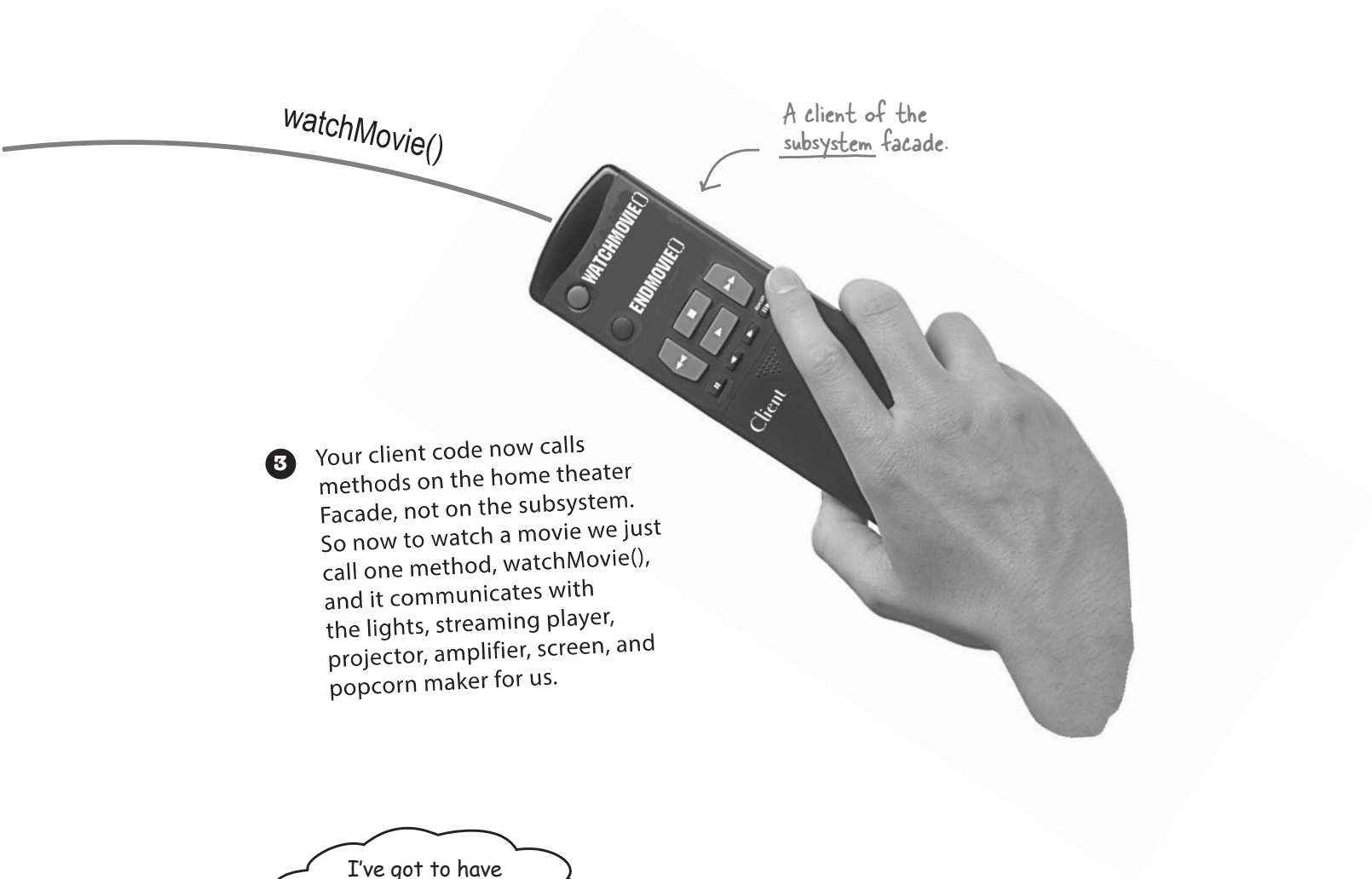
Let's take a look at how the Facade operates:

- 1 Okay, time to create a Facade for the home theater system. To do this we create a new class `HomeTheaterFacade`, which exposes a few simple methods such as `watchMovie()`.

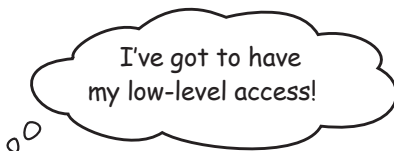
- 2 The Facade class treats the home theater components as a subsystem, and calls on the subsystem to implement its `watchMovie()` method.

The subsystem the Facade is simplifying.





- 3** Your client code now calls methods on the home theater Facade, not on the subsystem. So now to watch a movie we just call one method, `watchMovie()`, and it communicates with the lights, streaming player, projector, amplifier, screen, and popcorn maker for us.



Former president of the Rushmore High School A/V Science Club.

- 4** The Facade still leaves the subsystem accessible so it can be used directly. If you need the advanced functionality of the subsystem classes, they are available for your use.

there are no Dumb Questions

Q: If the facade encapsulates the subsystem classes, how does a client that needs lower-level functionality gain access to them?

A: Facades don't "encapsulate" the subsystem classes; they merely provide a simplified interface to their functionality. The subsystem classes still remain available for direct use by clients that need to use more specific interfaces. This is a nice property of the Facade Pattern: it provides a simplified interface while still exposing the full functionality of the system to those who may need it.

Q: Does the facade add any functionality or does it just pass through each request to the subsystem?

A: A facade is free to add its own "smarts" in addition to making use of the subsystem. For instance, while our home theater facade doesn't implement any new behavior, it is smart enough to know that the popcorn popper has to be turned on before it can pop (as well as the details of how to turn on and stage a movie showing).

Q: Does each subsystem have only one facade?

A: Not necessarily. The pattern certainly allows for any number of facades to be created for a given subsystem.

Q: What is the benefit of the facade other than the fact that I now have a simpler interface?

A: The Facade Pattern also allows you to decouple your client implementation from any one subsystem. Let's say that you get a big raise and decide to upgrade your home theater to all new components that have different interfaces. Well, if you coded your client to the facade rather than the subsystem, your client code doesn't need to change, just the facade (and hopefully the manufacturer is supplying that!).

Q: So the way to tell the difference between the Adapter Pattern and the Facade Pattern is that the adapter wraps one class and the facade may represent many classes?

A: No! Remember, the Adapter Pattern changes the interface of one or more classes into one interface that a client is expecting. While most textbook examples show the adapter adapting one class, you may need to adapt many classes to provide the interface a client is coded to. Likewise, a Facade may provide a simplified interface to a single class with a very complex interface.

The difference between the two is not in terms of how many classes they "wrap," it is in their intent. The intent of the Adapter Pattern is to alter an interface so that it matches one a client is expecting. The intent of the Facade Pattern is to provide a simplified interface to a subsystem.

A facade not only simplifies an interface, it decouples a client from a subsystem of components.

Facades and adapters may wrap multiple classes, but a facade's intent is to simplify, while an adapter's is to convert the interface to something different.