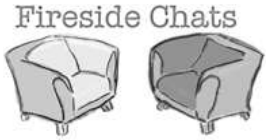
fireside chats: *decorator and adapter*



Tonight's talk: **The Decorator Pattern and the Adapter Pattern discuss their differences.**

Decorator

I'm important. My job is all about *responsibility* – you know that when a Decorator is involved there's going to be some new responsibilities or behaviors added to your design.

That may be true, but don't think we don't work hard. When we have to decorate a big interface, whoa, that can take a lot of code.

Cute. Don't think we get all the glory; sometimes I'm just one decorator that is being wrapped by who knows how many other decorators. When a method call gets delegated to you, you have no idea how many other decorators have already dealt with it and you don't know that you'll ever get noticed for your efforts servicing the request.

Adapter

You guys want all the glory while us adapters are down in the trenches doing the dirty work: converting interfaces. Our jobs may not be glamorous, but our clients sure do appreciate us making their lives simpler.

Try being an adapter when you've got to bring several classes together to provide the interface your client is expecting. Now that's tough. But we have a saying: "an uncoupled client is a happy client."

Hey, if adapters are doing their job, our clients never even know we're there. It can be a thankless job.

Head First Design Patterns By Eric Freeman, Elisabeth Freeman, Bert Bates, Kathy Sierra
ISBN: 0596007124 Publisher: O'Reilly
Print Publication Date: 2004/10/25

Decorator

Well us decorators do that as well, only we allow *new behavior* to be added to classes without altering existing code. I still say that adapters are just fancy decorators – I mean, just like us, you wrap an object.

Uh, no. Our job in life is to extend the behaviors or responsibilities of the objects we wrap, we aren't a *simple pass through*.

Maybe we should agree to disagree. We seem to look somewhat similar on paper, but clearly we are *miles* away in our *intent*.

Adapter

But, the great thing about us adapters is that we allow clients to make use of new libraries and subsets without changing *any* code, they just rely on us to do the conversion for them. Hey, it's a niche, but we're good at it.

No, no, no, not at all. We *always* convert the interface of what we wrap, you *never* do. I'd say a decorator is like an adapter; it is just that you don't change the interface!

Hey, who are you calling a simple pass through? Come on down and we'll see how long *you* last converting a few interfaces!

Oh yeah, I'm with you there.

Head First Design Patterns By Eric Freeman, Elisabeth Freeman, Bert Bates, Kathy Sierra
ISBN: 0596007124 Publisher: O'Reilly
Print Publication Date: 2004/10/25

who does what?

And now for something different...

There's another pattern in this chapter.

You've seen how the Adapter Pattern converts the interface of a class into one that a client is expecting. You also know we achieve this in Java by wrapping the object that has an incompatible interface with an object that implements the correct one.

We're going to look at a pattern now that alters an interface, but for a different reason: to simplify the interface. It's aptly named the Facade Pattern because this pattern hides all the complexity of one or more classes behind a clean, well-lit facade.

WHO DOES WHAT?

Match each pattern with its intent:

Pattern	Intent
Decorator	Converts one interface to another
Adapter	Doesn't alter the interface, but adds responsibility
Facade	Makes an interface simpler

Head First Design Patterns By Eric Freeman, Elisabeth Freeman, Bert Bates, Kathy Sierra
ISBN: 0596007124 Publisher: O'Reilly
Print Publication Date: 2004/10/25