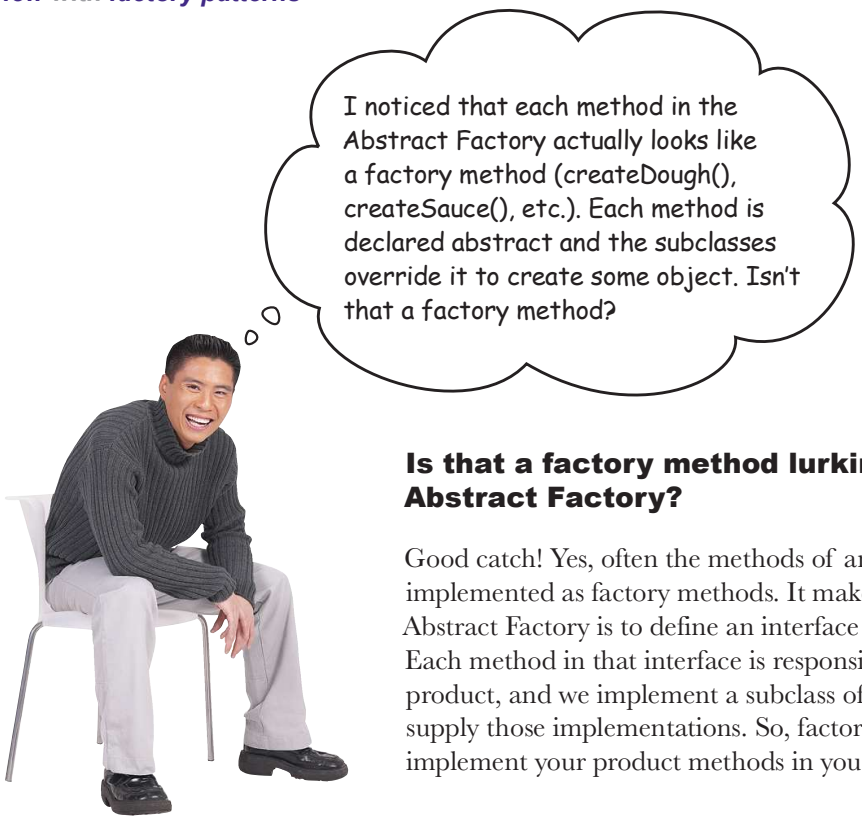

Is that a factory method lurking inside the Abstract Factory?

Good catch! Yes, often the methods of an Abstract Factory are implemented as factory methods. It makes sense, right? The job of an Abstract Factory is to define an interface for creating a set of products. Each method in that interface is responsible for creating a concrete product, and we implement a subclass of the Abstract Factory to supply those implementations. So, factory methods are a natural way to implement your product methods in your abstract factories.

Patterns Exposed

This week's interview:
Factory Method and Abstract Factory, on each other

HeadFirst: Wow, an interview with two patterns at once! This is a first for us.

Factory Method: Yeah, I'm not so sure I like being lumped in with Abstract Factory, you know. Just because we're both factory patterns doesn't mean we shouldn't get our own interviews.

HeadFirst: Don't be miffed, we wanted to interview you together so we could help clear up any confusion about who's who for the readers. You do have similarities, and I've heard that people sometimes get you confused.

Abstract Factory: It's true, there have been times I've been mistaken for Factory Method, and I know you've had similar issues, Factory Method. We're both really good at decoupling applications from specific implementations; we just do it in different ways. So I can see why people might sometimes get us confused.

Factory Method: Well, it still ticks me off. After all, I use classes to create and you use objects; that's totally different!

HeadFirst: Can you explain more about that, Factory Method?

Factory Method: Sure. Both Abstract Factory and I create objects—that's our job. But I do it through inheritance...

Abstract Factory: ...and I do it through object composition.

Factory Method: Right. So that means, to create objects using Factory Method, you need to extend a class and provide an implementation for a factory method.

HeadFirst: And that factory method does what?

Factory Method: It creates objects, of course! I mean, the whole point of the Factory Method Pattern is that you're using a subclass to do your creation for you. In that way, clients only need to know the abstract type they are using; the subclass worries about the concrete type. So, in other words, I keep clients decoupled from the concrete types.

Abstract Factory: And I do too, only I do it in a different way.

HeadFirst: Go on, Abstract Factory...you said something about object composition?

Abstract Factory: I provide an abstract type for creating a family of products. Subclasses of this type define how those products are produced. To use the factory, you instantiate one and pass it into some code that is written against the abstract type. So, like Factory Method, my clients are decoupled from the actual concrete products they use.

HeadFirst: Oh, I see, so another advantage is that you group together a set of related products.

Abstract Factory: That's right.

HeadFirst: What happens if you need to extend that set of related products to, say, add another one? Doesn't that require changing your interface?

Abstract Factory: That's true; my interface has to change if new products are added, which I know people don't like to do....

Factory Method: <snicker>

Abstract Factory: What are you snickering at, Factory Method?

Factory Method: Oh, come on, that's a big deal! Changing your interface means you have to go in and change the interface of every subclass! That sounds like a lot of work.

Abstract Factory: Yeah, but I need a big interface because I am used to creating entire families of products. You're only creating one product, so you don't really need a big interface, you just need one method.

HeadFirst: Abstract Factory, I heard that you often use factory methods to implement your concrete factories?

Abstract Factory: Yes, I'll admit it, my concrete factories often implement a factory method to create their products. In my case, they are used purely to create products...

Factory Method: ...while in my case I usually implement code in the abstract creator that makes use of the concrete types the subclasses create.

HeadFirst: It sounds like you both are good at what you do. I'm sure people like having a choice; after all, factories are so useful, they'll want to use them in all kinds of different situations. You both encapsulate object creation to keep applications loosely coupled and less dependent on implementations, which is really great, whether you're using Factory Method or Abstract Factory. May I allow you each a parting word?

Abstract Factory: Thanks. Remember me, Abstract Factory, and use me whenever you have families of products you need to create and you want to make sure your clients create products that belong together.

Factory Method: And I'm Factory Method; use me to decouple your client code from the concrete classes you need to instantiate, or if you don't know ahead of time all the concrete classes you are going to need. To use me, just subclass me and implement my factory method!