there are no Dumb Questions

Q: What's the advantage of the Factory Method Pattern when you only have one ConcreteCreator?

A: The Factory Method Pattern is useful if you've only got one concrete creator because you are decoupling the implementation of the product from its use. If you add additional products or change a product's implementation, it will not affect your Creator (because the Creator is not tightly coupled to any ConcreteProduct).

Q: Would it be correct to say that our NY and Chicago stores are implemented using Simple Factory? They look just like it.

A: They're similar, but used in different ways. Even though the implementation of each concrete store looks a lot like the SimplePizzaFactory, remember that the concrete stores are extending a class that has defined createPizza() as an abstract method. It is up to each store to define the behavior of the createPizza() method. In Simple Factory, the factory is another object that is composed with the PizzaStore.

Q: Are the factory method and the Creator class always abstract?

A: No, you can define a default factory method to produce some concrete product. Then you always have a means of creating products even if there are no subclasses of the Creator class.

Q: Each store can make four different kinds of pizzas based on the type passed in. Do all concrete creators make multiple products, or do they sometimes just make one?

A: We implemented what is known as the parameterized factory method. It can make more than one object based on a parameter passed in, as you noticed. Often, however, a factory just produces one object and is not parameterized. Both are valid forms of the pattern.

Q: Your parameterized types don't seem "type-safe." I'm just passing in a String! What if I asked for a "CalmPizza"?

A: You are certainly correct, and that would cause what we call in the business a "runtime error." There are several other more sophisticated techniques that can be used to make parameters more "type safe"—in other words, to ensure errors in parameters can be caught at compile time. For instance, you can create objects that represent the parameter types, use static constants, or use enums.

Q: I'm still a bit confused about the difference between Simple Factory and Factory Method. They look very similar, except that in Factory Method, the class that returns the pizza is a subclass. Can you explain?

A: You're right that the subclasses do look a lot like Simple Factory; however, think of Simple Factory as a one-shot deal, while with Factory Method you are creating a framework that lets the subclasses decide which implementation will be used. For example, the orderPizza() method in the Factory Method Pattern provides a general framework for creating pizzas that relies on a factory method to actually create the concrete classes that go into making a pizza. By subclassing the PizzaStore class, you decide what concrete products go into making the pizza that orderPizza() returns. Compare that with Simple Factory, which gives you a way to encapsulate object creation, but doesn't give you the flexibility of Factory Method because there is no way to vary the products you're creating.