Payson Blackwell

1/23/19

Special Projects CSD 399

Factory Method

Let’s imagine that you are assigned to make a loan calculator for a bank that must change what the rate is for each of the different types of customers (regular, business, corporation, and more). You may make a class for each type of customer and have the different rates for each apply when you use that class. To help with some confusion, you may create a base customer that all other customer types are derived from. That way would work, but if you aren’t careful it would be quite easy to have the size of your calling method to grow for each different type of customer since you have to initialize and call the correct method for that specific class. Some people would call it good there, others might wonder if there is a better way to implement it, especially if it seems likely that you’ll have to add different type of customers in the future which would only lengthen the already hideous to read code you’ve written. It turns out, there is a better way to write it, and it doesn’t require that much extra effort to implement.

The factory method is a creational pattern for object-oriented design, which means that it is a convention that you should follow if you get a chance to since it will make your code easier to read, reusable in the future, and will make your life a lot simpler if you have add more subclasses in the future. The factory method is best to use when you need to create an object of a subclass, but not sure which type of subclass until you initialize it. That basics of it is that you have a base-class with all the shared methods you want your subclasses to have, and when you want to initialize one of your subclasses, you make a base-class object equal to the return of a different class (usually has the name creator in it) that returns the correct type of subclass object that you need based on a parameter. You can then call the needed method from your base-class object and that’s it.

It is tough to follow sometimes what is written in a paragraph instead of a code example (feel free to look at a small example that I made using the factory method) but let’s imagine you understand the general concept of it now on focus on the benefits of following it. Following the factory method makes it easier to make new subclasses since you don’t have to change any code in either the base-class nor the main method (or wherever you have it in your code). The only change you would have to do, is to change the creator class method to return your new subclass and create the subclass itself. Overall it makes the main method have less code, easier to follow, and you don’t have to change it for a new subclass. This is a lot better solution then just front loading the logic as in the example solution.

Parts of the Factory Method:

* Product (base-class): defines the interface of objects the factory method creates
* Concrete Product (subclasses): implements the product interface
* Creator (the creator class): declares the factory method, which returns an object of type Product
* ConcreteCreator (instance of the creator class): overrides the factory method to return an instance of a concreteProduct.

Factory Method Definition: Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses.