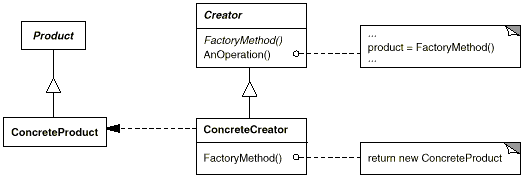
[*http://architects.dzone.com/articles/factory-method-vs-abstract*](http://architects.dzone.com/articles/factory-method-vs-abstract)

*Hello Friends,*

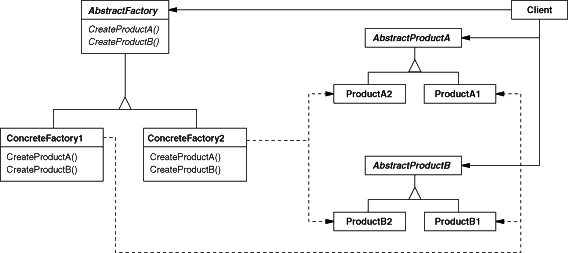
*Recently I was asked by one of my friends what is the difference between Factory Method and Abstract Factory design patterns, but I didn't seem to convince him easily. Of course I’d already read Head First, GoF, Pattern Hatching, Refactoring to Patterns, but not deeply all of them and it was not recent. So I decided to explore more and tried to simply the differences, and here ‘s my attempt.*

I assume that you are already familiar with both the patterns, so I’ll focus here on where the most people have confusion with the differences. Let’s revisit the definitions and their structure first:

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

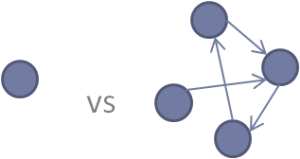


**Abstract Factory**: Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

[](http://javacurious.files.wordpress.com/2013/03/af_dp.png)

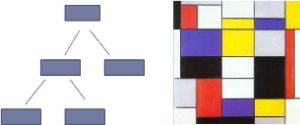
**Differences**

**One product vs Set of many**

[](http://javacurious.files.wordpress.com/2013/03/picture1.png)

This is perhaps the easiest one but important certainly. Factory Method is used to*create one product* only but Abstract Factory is about*creating families of related or dependent products*.

**Inheritance vs Composition**

[](http://javacurious.files.wordpress.com/2013/03/picture2.png)

This is perhaps the most confusing one (as both seems to be using inheritance). Factory Method depends on *inheritance* to decide which product to be created. In Creator class (in structure diagram), it has *other methods also* (implemented, to manipulate the product) which use the *only abstract method* *FactoryMethod() to create* the product and it *can only be implemented/changed by subclasses*.

Here the Creator class is also acting like *Client* which *depends only on a single method* to create a product. We have to create a *subclass of whole Creator/Client* class to create a new different Product. There’s no separate and dedicated class for creation of a Product, had it been the case we could have used it with composition where we can pass an object of factory to client and client can use it without getting into inheritance hierarchy.

On the other side, in Abstract Factory, there’s a *separate class dedicated to create* a family of related/dependent Products and its (any concrete subclass factory) *object* can be passed to the client which uses it (*composition*). Here the *Client* gets a *different* object (concrete factory) to create the Products, *instead of creating itself* (e.g. using factoryMethod() and forcing inheritance), and thus uses *composition*.

If we think of just a*product creation facility* and the *client*that uses it, it is clear that in Factory Method, we are restricted to use *inheritance*(class based) and in Abstract Factory we have the flexibility of *composition*(object based) to create specific Products.

//Factory Method

class Client {

public void anOperation() {

Product p = factoryMethod();

p.doSomething();

}

protected Product factoryMethod() {//or it can be abstract as well

return new DefaultProduct();

}

}

class NewClient extends Client {

protected Product factoryMethod() {//overriding

return new SpecificProduct();

}

}

//Abstract Factory

class Client {

private Factory factory;

public Client (Factory factory) {

this.factory = factory;

}

public void anOperation() {

ProductA p = factory.createProductA();

p.doSomething();//other products and operations as well

}

}

interface Factory {

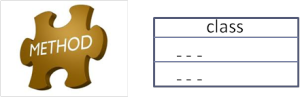
ProductA createProductA();

ProductB createProductB();

}

//concrete factories also, implementing Factory interface

**Method vs full class (object)**

[](http://javacurious.files.wordpress.com/2013/03/picture3.png)

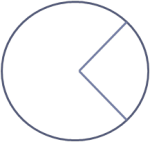
Factory Method is*just a method* while Abstract Factory is an *object*. The *purpose* of a Class having factory method is*not just create* objects, it does other work also, *only a method is responsible* for *creating* object. In Abstract Factory, the *whole purpose* of the class is to *create family of objects*.

**Level of abstraction**

[](http://javacurious.files.wordpress.com/2013/03/level-of-abstraction-640x390.jpg)

Abstract Factory is one level *higher in abstraction* than Factory Method. Factory Method *abstracts the way objects are created*, while Abstract Factory also *abstracts the way factories are created* which in turn abstracts the way objects are created.

**One inside another**

[](http://javacurious.files.wordpress.com/2013/03/picture4.png)

As Abstract Factory is at higher level in abstraction, it *often uses* Factory Method to create the products in factories.

*I also believe that we should not be obsessed with design patterns, these are all built on good basic design principles, and often mixed while using in real world.*

*I hope it helps somebody else also. Let me also know if it can be improved.*

*Happy patterns, bye.*