FACTORY DESIGN PATTERN

The Factory design pattern is an object-oriented Creational design pattern. It implements the code to allow creation of objects the way products are created in factories.

The essence of this pattern is to "*Define an interface for creating an object, but let the classes that implement the interface decide which class to instantiate. Hence, deferring (postponing) the instantiation to subclasses.*" A super-class specifies standards and generic behavior (using pure virtual "placeholders" for creation steps), and then delegates the creation details to sub-classes that are supplied by the client.

Factory Method is to creating objects as Template Method is to implementing an algorithm.

Factory design pattern can come in multiple flavors. But there are two basic elements

- The Factory
- The Products

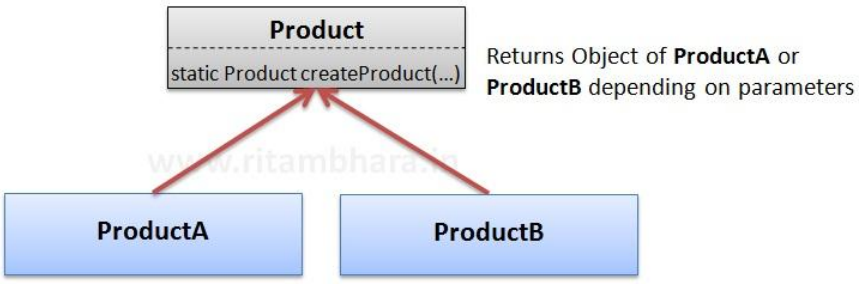
Factory is responsible for creating products and Products are created in a factory.

Factory may be a simple function in a class or we can have a separate factory class altogether (Sometimes we can even have multiple factory classes, one for each Product).

Lets discuss the common implementations of Factory Pattern:

1. static function returning object of same class type

In this case we actually don't have a factory class but just a static function inside the product class which allows creation of the Product. something like what is shown below:



The actual object being returned by the factory method (createProduct) is the object of either of the subclasses. The Product class may be an abstract class in this case.

For example: If Product is Colour and ProductA & ProductB are RGB & CMYK, then the implementations may be:

```
1   class Color{
2     public:
3      static Color makeColor(int colorType)
4      {
5         switch(type)
6         {
7            case 1: return new RGB();
8            break;
9            case 2: return new CMYK();
10           break;
11        }
12     }
13  };
14  class RGB : public Color
15  {
16  };
17  class CMYK : public Color
18  {
19  };
```

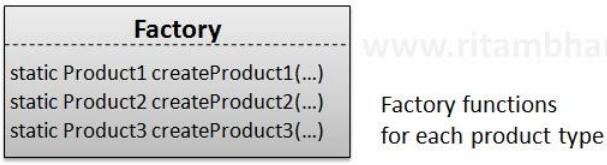
2. Static function returning object of another class

In the above implementation, we can have multiple static functions, each returning object of specific product type. For example, in theColor class above we can have two static functions

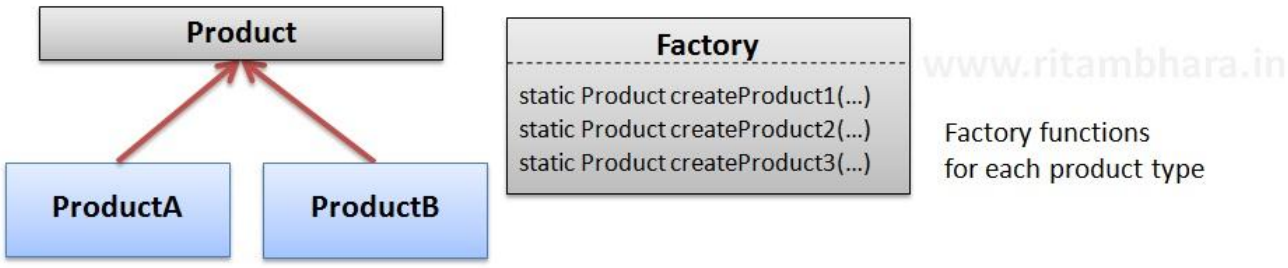
1 static RGB createRGBColor();

2 static CMYK createCMYKColor();

In this case the concrete products (RGB and CMYK) may not necessarily be the child of the same class (or may not even have a common parent class.



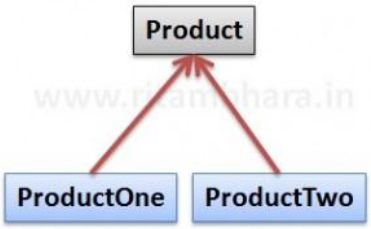
Or all the Products have a common parent class (More common), and each static function returns the same type.



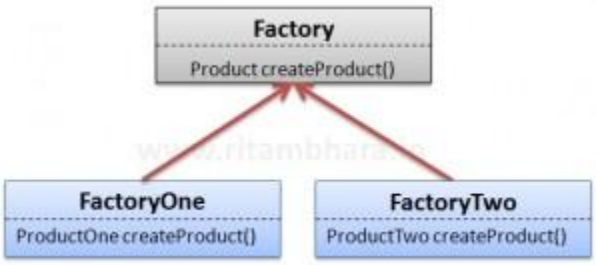
3. Multiple Factory Multiple Products

In this case each concrete product has its dedicated Factory. But since, object creation interface should be independent of the type of the object (the basic requirement of Factory Pattern). All the factories will be implementing the common interface to create the product (defined in the parent class of all factories).

Let ProductOne and ProductTwo be two concrete products (ex. RGB and CMYK).



FactoryOne and FactoryTwo will create objects of ProductOne and ProductTwo respectively.



In this case it looks more like the “Abstract Factory” design pattern but there is a small difference.

This is nothing but an extension of factory pattern called: “Abstract Factory Design Pattern”.

Important Points:

1. Since we are creating the objects using factory methods, Default Constructor, Copy constructor and overloaded assignment operator should be declared as private to avoid direct object creation. Read more about this in Singleton design pattern post.
2. If you have an inheritance hierarchy that exercises polymorphism, consider adding a polymorphic creation capability by defining a static factory method in the base class.
3. Factory method's argument should be related to some characteristic of product.

Source: http://www.ritambhara.in/factory-design-pattern/