Factory Method Design Pattern in C#

Back to: Design Patterns in C# With Real-Time Examples

Factory Method Design Pattern in C# with Real-time Example

In this article, I am going to discuss the **Factory Method Design Pattern in C#** with an example. Please read our previous article where we discussed the **Factory Design Pattern in C#** with one real-time example. The Factory Method Design Pattern belongs to the Creational Pattern Category and is one of the most frequently used design patterns in real-time applications. As part of this article, we are going to discuss the following pointers.

1. **What is Factory Method Design Pattern?**
2. **Understanding the Factory Method Design Pattern with Real-time Example**
3. **Implementing the Factory Method Design Pattern in C#**
4. **When to use the Factory Method Design Pattern?**

Note: The most important point that you need to remember is the Factory Method Design pattern is not the exact same as the simple Factory Design Pattern that we discussed in our previous article. Most people think that both are the same and thus they use the terms Factory and Factory method interchangeably which is not right.

What is Factory Method Design Pattern?

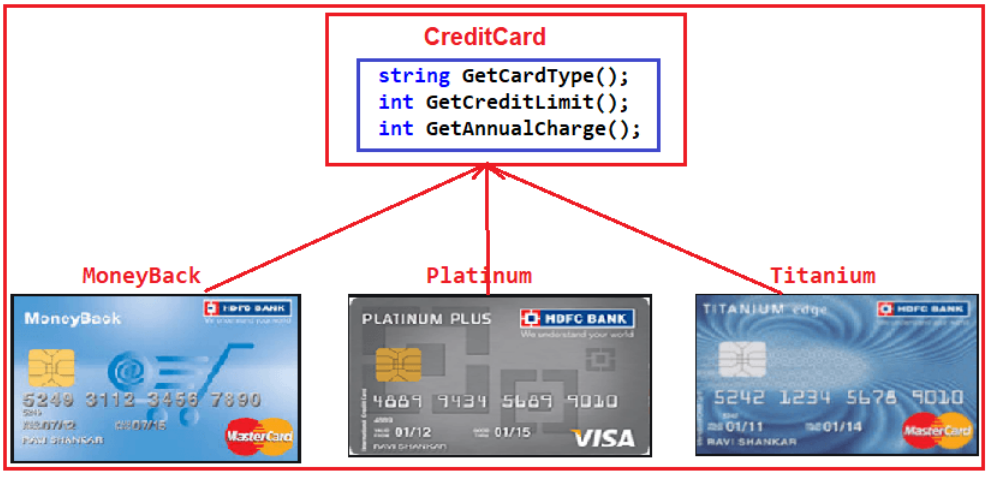
According to Gang of Four Definition "**Define an interface for creating an object, but let the subclasses decide which class to instantiate. The Factory method lets a class defer instantiation it uses to subclasses**".

Let us simplify the above definition. The Factory Method Design Pattern is used, when we need to create the object (i.e. instance of the Product class) without exposing the object creation logic to the client. To achieve this, in the factory method design pattern we will create an abstract class as the Factory class which will create and return the instance of the product, but it will let the subclasses decide which class to instantiate. If this is not clear at the moment then don't worry, I will explain this with one real-time example.

Understanding the Factory Method Design Pattern with Real-time Example:

Let us understand the Factory Method Design Pattern with one real-time example. We are going to develop an application for showing credit card information.

Please have a look at the following image. As you can see in the below diagram, we have three credit cards i.e. **MoneyBack, Titanium,** and **Platinum**. These credit cards are nothing but our Product classes. Again these three Credit Card classes are the subclasses of the **CreditCard** super interface. The CreditCard super interface defines the operations (i.e. **GetCardType, GetCreditLimit, and GetAnnualCharge**) which need to be implemented by the subclasses (i.e. **MoneyBack, Titanium, and Platinum**).



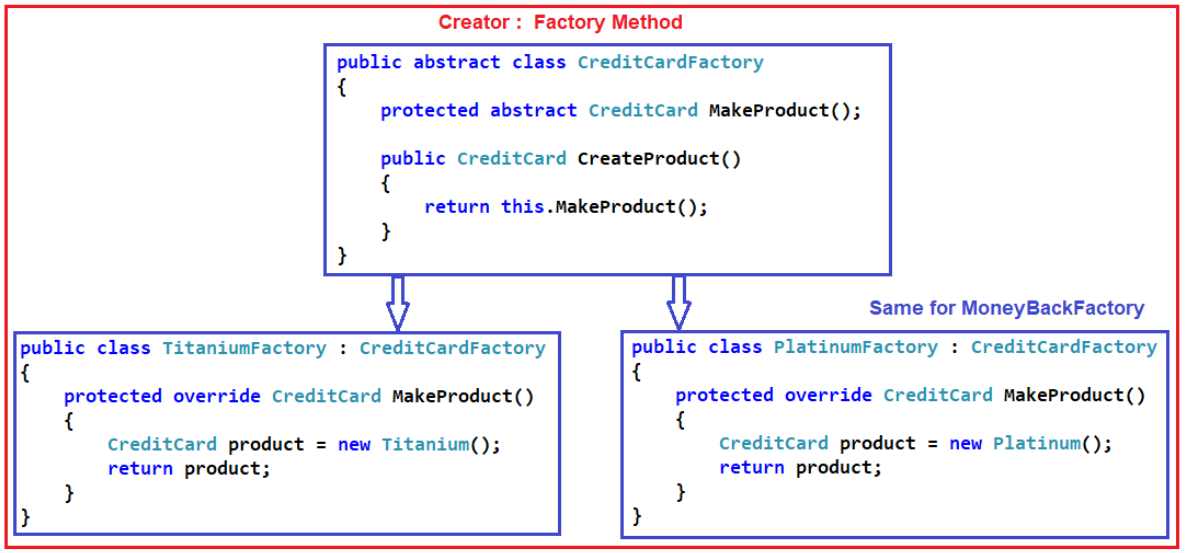
As per the definition of the Factory Method Design Pattern, we need to create an abstract class or interface for creating the object. Please have a look at the following diagram. This is going to be our Creator class that declares the factory method, which will return an object of type Product (i.e. CreditCard).

Creator : Factory Method

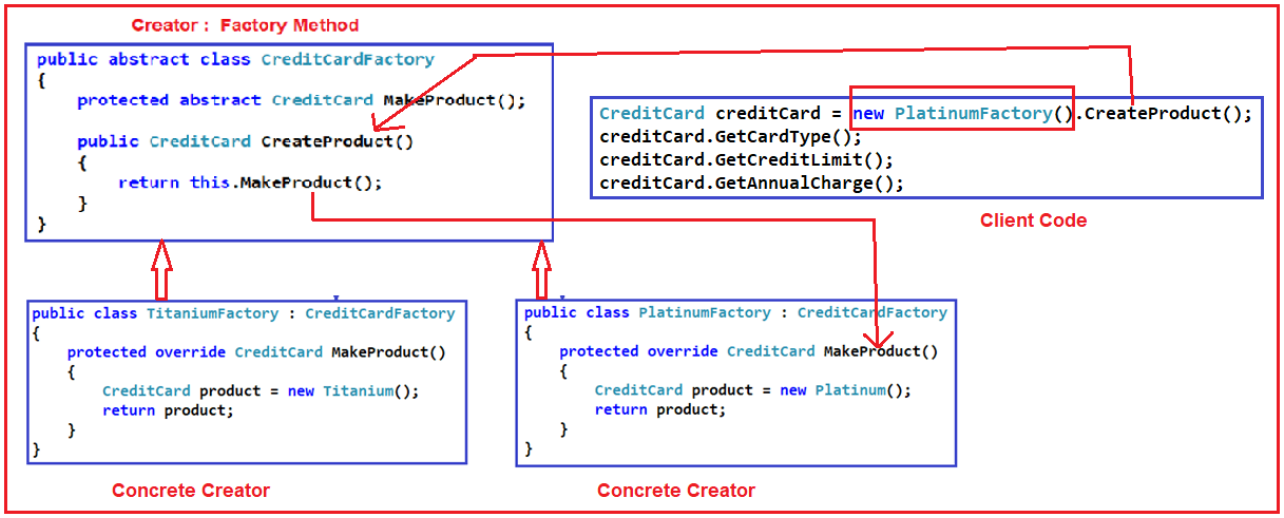
```
public abstract class CreditCardFactory
{
    protected abstract CreditCard MakeProduct();

    public CreditCard CreateProduct()
    {
        return this.MakeProduct();
    }
}
```

As you can see, the above abstract class (i.e. CreditCardFactory) contains two methods, one abstract method i.e. **MakeProduct()** and one concrete method i.e. **CreateProduct()**. The **CreateProduct()** method internally calls the **MakeProduct()** method of the subclass which will create the product instance and return that instance.

Please have a look at the following diagram. As we have three credit cards (i.e. MoneyBack, Platinum, and Titanium), so here we created three subclasses (i.e. PlatinumFactory, TitaniumFactory, and MoneyBackFactory) of the Abstract CreditCradFactory class and implement the MakeProduct method. This method is going to return the actual product object i.e. (MoneyBack, Platinum, and Titanium).



Now let see how the client is going to consume the above CreditCardFactory to create an object. Please have a look at the following diagram.



As shown in the above image, the client wants to create the Platinum object. In order to do so, the clients call the CreateProduct method on the PlatinumFactory instance which will return a concrete product.

Implementing the Factory Method Design Pattern in C#:

Step1: Creating Product Interface

Create a class file with the name **CreditCard.cs** and then copy and paste the following code. This is going to be our Product interface which will provide the

signature of the common functionalities which should be implemented by the concrete product classes.

```
namespace FactoryMethodDesignPattern
{
    public interface CreditCard
    {
        string GetCardType();
        int GetCreditLimit();
        int GetAnnualCharge();
    }
}
```

Step2: Creating Concrete Products

This is a class implementing the Product interface. As we have three types of Credit cards in our example, so we are going to create three classes (**MoneyBack, Titanium,** and **Platinum**) by implementing the **CreditCard** interface as shown below.

```
namespace FactoryMethodDesignPattern
{
    public class Platinum : CreditCard
    {
        public string GetCardType()
        {
            return "Platinum Plus";
        }
        public int GetCreditLimit()
        {
            return 35000;
        }
        public int GetAnnualCharge()
        {
            return 2000;
        }
    }
    public class Titanium : CreditCard
    {
        public string GetCardType()
        {
            return "Titanium Edge";
        }
        public int GetCreditLimit()
        {
            return 25000;
        }
        public int GetAnnualCharge()
        {
            return 1500;
        }
    }
    public class MoneyBack : CreditCard
    {
        public string GetCardType()
        {
            return "MoneyBack";
        }

        public int GetCreditLimit()
        {
            return 15000;
        }

        public int GetAnnualCharge()
        {
            return 500;
        }
    }
}
```

Step3: Creating Abstract Creator

The Abstract **Creator** declares the factory method, which returns an object of type Product. As per the definition, we need to create an abstract class or interface for creating the object. So, let us create an abstract class that will be our factory class with a publicly exposed method. That method is nothing but the factory method which will return the instance of the product. So create a class file with the name **CreditCardFactory.cs** and then copy and paste the following code in it.

```
namespace FactoryMethodDesignPattern
{
    public abstract class CreditCardFactory
    {
        protected abstract CreditCard MakeProduct();

        public CreditCard CreateProduct()
        {
            return this.MakeProduct();
        }
    }
}
```

We created the above abstract class with one abstract method i.e. **MakeProduct()** and one concrete method i.e. **CreateProduct()**. The **CreateProduct()** method internally calls the **MakeProduct()** method of the subclass which will create the product instance and return that instance.

Step4: Creating Concrete Creator

The **Concrete Creator** object overrides the factory method to return an instance of a Concrete Product. As we have three types of Credit Card, so we are going to create three classes (**PlatinumFactory, MoneyBackFactory, and TitaniumFactory**) which will implement the abstract CreditCardFactory class. So, create a class file with the name **ConcreteCreator.cs** and then copy and paste the following code into it.

```
namespace FactoryMethodDesignPattern
{
    public class MoneyBackFactory : CreditCardFactory
    {
        protected override CreditCard MakeProduct()
        {
            CreditCard product = new MoneyBack();
            return product;
        }
    }

    public class PlatinumFactory : CreditCardFactory
    {
        protected override CreditCard MakeProduct()
        {
            CreditCard product = new Platinum();
            return product;
        }
    }

    public class TitaniumFactory : CreditCardFactory
    {
        protected override CreditCard MakeProduct()
        {
            CreditCard product = new Titanium();
            return product;
        }
    }
}
```

As you can see the above three classes implement the MakeProduct method of the CreditCardFactory class. That's it. We are done with our implementation. Let's compare the Gang of Four Definition with our example.

According to Gang of Four, we need to define an interface or abstract class for creating an object. In our example, it is an abstract class i.e. **CreditCardFactory** class. The second part of the definition saying that let the subclasses decide which class to instantiate. In our example, the subclasses are **PlatinumFactory**, **MoneyBackFactory**, and **TitaniumFactory**. So these subclasses will decide which class to instantiate, for example, **MoneyBack, Titanium, and Platinum**.

Step5: Consuming the factory Method in the Client Code:

If you want to create an instance of the Platinum CreditCard then call the CreateProduct method of the PlatinumFactory instance, similarly, if you want the instance of Titanium CreditCard, then call the CreateProduct method of the TitaniumFactory instance. So, modify the Main method as shown below.

```
using System;
namespace FactoryMethodDesignPattern
{
    class Program
    {
        static void Main(string[] args)
        {

            CreditCard creditCard = new PlatinumFactory().CreateProduct();
            if (creditCard != null)
            {
                Console.WriteLine("Card Type : " + creditCard.GetCardType());
                Console.WriteLine("Credit Limit : " + creditCard.GetCreditLimit());
                Console.WriteLine("Annual Charge :" + creditCard.GetAnnualCharge());
            }
            else
            {
                Console.Write("Invalid Card Type");
            }
            Console.WriteLine("--------------");
            creditCard = new MoneyBackFactory().CreateProduct();
            if (creditCard != null)
            {
                Console.WriteLine("Card Type : " + creditCard.GetCardType());
                Console.WriteLine("Credit Limit : " + creditCard.GetCreditLimit());
                Console.WriteLine("Annual Charge :" + creditCard.GetAnnualCharge());
            }
            else
            {
                Console.Write("Invalid Card Type");
            }
            Console.ReadLine();
        }
    }
}
```

When we run the application, it displays the output as expected as shown below.

```
Card Type : Platinum Plus
Credit Limit : 35000
Annual Charge :2000
--------------
Card Type : MoneyBack
Credit Limit : 15000
Annual Charge :500
```

In the next article, I am going to discuss the **Abstract Factory Design Pattern in C#** with an example. In this article, I try to explain the **Factory Method**

Design Pattern in C# step by step with an example. I hope this article will help you with your needs. I would like to have your feedback. Please post your feedback, question, or comments about this article.