Factory Method Pattern vs. Abstract Factory Pattern

Marla Sukesh
28 Jan 2014 CPOL

In this article we will learn difference between Factory Method Pattern and Abstract Factory Pattern.

Download Simple Factory Sample
Download Factory Method Sample
Download Abstract Factory Sample

Introduction

Design patterns are reusable and documented solutions for commonly occurring problems in software programming or development.

In one of my previous article about Factory Pattern I spoke about what are different flavors of Factory pattern and how to choose between them. But I think there are some confusion with regards to Factory Method Pattern and Abstract Factory. So we will work on it.

Who should read this article?

If you have confusion understanding the difference between Factory Method Pattern and Abstract Factory Pattern you are at right place.

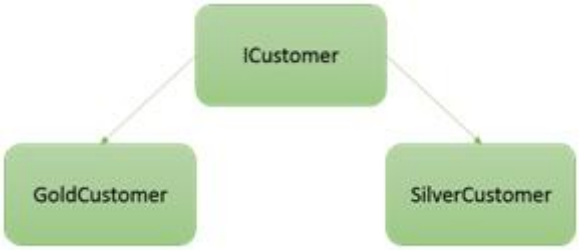
What these two are?

- First of all both of them falls under Creational category and it means it will solves the problem pertaining to object creation.
- Factory Method and abstract factory pattern all are about creating objects.

In this article I would like to emphasize on one more terminology that is Simple Factory.

1. Simple Factory and Factory Method

For our discussion let's have a small problem statement.



Class structure

Shrink ▲ Copy Code

```
public interface ICustomer
{
    void AddPoints();
    void AddDiscount();
}

public class GoldCustomer : ICustomer
{
    public void AddPoints()
    {
        Console.WriteLine("Gold Customer - Points Added");
    }

    public void AddDiscount()
    {
        Console.WriteLine("Gold Customer - Discount Added");
    }

    public void GoldOperation()
    {
        Console.WriteLine("Operation specific to Gold Customer");
    }
}

public class SilverCustomer : ICustomer
{
    public void AddPoints()
    {
        Console.WriteLine("Silver Customer -  Points Added");
    }

    public void AddDiscount()
    {
        Console.WriteLine("Silver Customer - Discount Added");
```

```
    }

    public void SilverOperation()
    {
        Console.WriteLine("Operation specific to Silver Customer");
    }
}
```

Problem Statement

Client want to create Customer Object (either Gold or Silverbased on requirement).

Simple Factory

This is one of the pattern not born from GOF and most of the peopleconsiders this as the default Factory method pattern.

Here, we will just take out object creation process out of the client code and put into some other class. Look at the code demonstration.

Copy Code

```
class CustomerFactory
{
    public static ICustomer GetCustomer(int i)
    {
        switch (i)
        {
            case 1:
                GoldCustomer goldCustomer = new GoldCustomer();
                goldCustomer.GoldOperation();
                goldCustomer.AddPoints();
                goldCustomer.AddDiscount();
                return goldCustomer;
            case 2:
                SilverCustomer silverCustomer = new SilverCustomer();
                silverCustomer.SilverOperation();
                silverCustomer.AddPoints();
                silverCustomer.AddDiscount();
                return silverCustomer;
            default: return null;
        }
    }
}

//Client Code
ICustomer c = CustomerFactory.GetCustomer(someIntegerValue);
```

Factory Method Pattern

In this pattern we define an interface which will expose a **method** which will create objects for us. Return type of that method is never be a concrete type rather it will be some interface (or may be an abstract class)

Shrink ▲ Copy Code

```
public abstract class BaseCustomerFactory
{
    public ICustomer GetCustomer()
    {
        ICustomer myCust = this.CreateCustomer();
        myCust.AddPoints();
        myCust.AddDiscount();
        return myCust;
    }
    public abstract ICustomer CreateCustomer();
}
```

```
public class GoldCustomerFactory : BaseCustomerFactory
{
    public override ICustomer CreateCustomer()
    {
        GoldCustomer objCust = new GoldCustomer();
        objCust.GoldOperation();
        return objCust;
    }
}
public class SilverCustomerFactory : BaseCustomerFactory
{
    public override ICustomer CreateCustomer()
    {
        SilverCustomer objCust = new SilverCustomer();
        objCust.SilverOperation();
        return objCust;
    }
}
//Client Code
BaseCustomerFactory c = new GoldCustomerFactory();// Or new SilverCustomerFactory();
ICustomer objCust = c.GetCustomer();
```

Note:- To understand when to use Simple Factory and when to use Factory Method Pattern click here.

2. Abstract Factory Pattern

In Abstract Factory we define an interface which will create families of related or dependent objects. In simple words, interface will expose multiple methods each of which will create some object. Again, here method return types will be generic interfaces. All this objects will together become the part of some important functionality.

Question – *If every factory is going to create multiple objects and all those objects will be related to each other (means they will use each other) how this relating happens and who does that?*

Answer –

- There will be an intermediary class which will have composition relationship with our interface.
- This class will do all the work, using all the objects got from interface methods.
- This will be the class with which client will interact.

Let's talk about a scenario.
We want to build desktop machine. Let see what will be the best design for that,

Shrink ▲ Copy Code

```
public interface IProcessor
{
    void PerformOperation();
}
public interface IHardDisk { void StoreData(); }
public interface IMonitor { void DisplayPicture();}

public class ExpensiveProcessor : IProcessor
{
    public void PerformOperation()
    {
        Console.WriteLine("Operation will perform quickly");
    }
}
public class CheapProcessor : IProcessor
{
    public void PerformOperation()
    {
        Console.WriteLine("Operation will perform Slowly");
    }
}

public class ExpensiveHDD : IHardDisk
```

```
{
    public void StoreData()
    {
        Console.WriteLine("Data will take less time to store");
    }
}
public class CheapHDD : IHardDisk
{
    public void StoreData()
    {
        Console.WriteLine("Data will take more time to store");
    }
}

public class HighResolutionMonitor : IMonitor
{
    public void DisplayPicture()
    {
        Console.WriteLine("Picture quality is Best");
    }
}
public class LowResolutionMonitor : IMonitor
{
    public void DisplayPicture()
    {
        Console.WriteLine("Picture quality is Average");
    }
}
```

Factory Code will be as follows

Shrink ▲ Copy Code

```
public interface IMachineFactory
{
    IProcessor GetRam();
    IHardDisk GetHardDisk();
    IMonitor GetMonitor();
}

public class HighBudgetMachine : IMachineFactory
{
    public IProcessor GetRam() { return new ExpensiveProcessor(); }
    public IHardDisk GetHardDisk() { return new ExpensiveHDD(); }
    public IMonitor GetMonitor() { return new HighResolutionMonitor(); }
}
public class LowBudgetMachine : IMachineFactory
{
    public IProcessor GetRam() { return new CheapProcessor(); }
    public IHardDisk GetHardDisk() { return new CheapHDD(); }
    public IMonitor GetMonitor() { return new LowResolutionMonitor(); }
}
//Let's say in future...Ram in the LowBudgetMachine is decided to upgrade then
//first make GetRam in LowBudgetMachine Virtual and create new class as follows

public class AverageBudgetMachine : LowBudgetMachine
{
    public override IProcessor GetRam()
    {
        return new ExpensiveProcessor();
    }
}
```

Copy Code

```
public class ComputerShop
{
    IMachineFactory category;
```

```
    public ComputerShop(IMachineFactory _category)
    {
        category = _category;
    }
    public void AssembleMachine()
    {
        IProcessor processor = category.GetRam();
        IHardDisk hdd = category.GetHardDisk();
        IMonitor monitor = category.GetMonitor();
        //use all three and create machine

        processor.PerformOperation();
        hdd.StoreData();
        monitor.DisplayPicture();
    }
}
```

Copy Code

```
//Client Code
IMachineFactory factory = new HighBudgetMachine();// Or new LowBudgetMachine();
ComputerShop shop = new ComputerShop(factory);
shop.AssembleMachine();
```

Conclusion

I think now you know what is the difference between Factory Method Pattern and Abstract Factory

Hope all of you enjoyed reading this article. Thank you for the patience.

For technical training related to various topics including ASP.NET, Design Patterns, WCF and MVC contact SukeshMarla@Gmail.com or at www.sukesh-marla.com

For more stuff like this click here. Subscribe to article updates or follow at twitter @SukeshMarla

License

Share

About the Author

Marla Sukesh

Founder Just Compile
India

Learning is fun but teaching is awesome.

Who I am? Trainer + consultant + Developer/Architect + Director of Just Compile

My Company - Just Compile

I can be seen in, @sukeshmarla or Facebook

Comments and Discussions

You must Sign In to use this message board.

Search Comments

First Prev Next

Error in Your Code
whitepacific 5-Aug-17 20:48

My vote of 5
Vikas K Solegaonkar 5-Mar-17 23:14

Very good article, but...
Member 12176294 28-Oct-16 5:48

I came here directly from my first reading of GOF's *Design Patterns* because your title was exactly the question which I was struggling with. However, the article does the same thing as the GOF book - it lucidly described the two patterns but does not actually perform a direct **contrast**. You do an enlightening **comparison** in your introduction, but you do not directly tackle the 'versus' of your title.

The Factory Method section helps tremendously with how it differs from the Simple Factory, although I had to mentally provide my own justification for its deployment (avoid repeating code and allows flexibility). For me, this certainly was an improvement on the GOF book.

[**EDIT** (the next day): GOF, *Design Patterns* has the sentence

Shrink ▲ Copy Code

```
Creational class patterns defer some part of object creation to subclasses, while
Creational object patterns defer it to another object.
```

Had I spotted this yesterday, I may not have written the next paragraph.]

FYI, some of my confusion arose from the particular question I was trying to answer: Even though GOF defines the Factory Method as concerning the relationship between classes (class scope), why did they not define the Abstract Pattern as class scope, when the Abstract Pattern is explicitly intended to maintain links between classes when deployed? It still seemed to me that the two patterns offered the same thing at a different levels. I had a breakthrough, which may be of no use to others learner, when I noticed there was no need for an *abstract* method in the *Abstract* Factory method. It *is* a different level of abstraction, but the Factory Method helps to define differences between subclasses when they are created, while the Abstract Method *may* be concerned with deploying entirely unrelated (though similar) classes, as a whole. Thus I am

now happy with the GOF calling the Factory Method 'class scope' (as it farms out the deployment of possibly unique subclass traits to subFactories) and the way they call the Abstract Method 'object scope' though it works on a higher level of abstraction (since the classes may be deployed as whole entities without any regard for alteratives). If I have it right, and I may wake up tomorrow embarrassed by this comment, both involve deploying sibling concrete classes, but the Factory Method is directly concerned with the differences between these siblings, whereas the Abstract Method just cares if they work or not in the situation where they are deployed with other classes.

In case this sounds too negative, let me repeat: IMHO the first half of your article is much better than GOF, and the second is as good (and GOF are *very* good). I just feel you did not do justice to the 'vs.' of the title which brought me here.

Thank you,
Reg.

modified 29-Oct-16 8:23am.

Sign In · View Thread

Re: Very good article, but...
Vikas K Solegaonkar 6-Mar-17 1:01

My vote of 5
Member 12162078 1-Dec-15 17:26

Re: My vote of 5
Marla Sukesh 15-Apr-16 3:18

GetRam() -> GetCPU()
Mobile Developer 21-Oct-15 7:53

Re: GetRam() -> GetCPU()
Marla Sukesh 22-Oct-15 2:15

Very good article for beginners
Member 3748706 23-Sep-15 0:06

Re: Very good article for beginners
Marla Sukesh 22-Oct-15 2:16

My vote of 5
weligamagepriya 13-Dec-14 1:12

Re: My vote of 5
Marla Sukesh 13-Dec-14 3:54

Thanks.
shalim ahmed 26-Sep-14 4:25

My vote of 5
Humayun Kabir Mamun 10-Jul-14 1:02

Re: My vote of 5
Marla Sukesh 13-Dec-14 3:52

Nice
Member 4518976 18-Jun-14 2:39

Re: Nice
Marla Sukesh 18-Jun-14 3:46

Abstract Factory Pattern

sasson24 **18-Mar-14 3:07**

My vote of 5
Ștefan-Mihai MOGA **14-Feb-14 17:04**

Re: My vote of 5
Marla Sukesh 20-Feb-14 20:30

My vote of 5
manoj.jsm **13-Feb-14 22:06**

Re: My vote of 5
Marla Sukesh 20-Feb-14 20:30

My vote of 5
ThatsAlok **10-Feb-14 1:50**

Re: My vote of 5
Marla Sukesh 20-Feb-14 20:30

Abstract Factory
Member 10366887 **29-Jan-14 19:57**

Refresh 1 2 Next ▷

General News Suggestion Question Bug Answer Joke Praise Rant Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.