Factory Method Pattern vs. Abstract Factory Pattern

codeproject.com/Articles/716413/Factory-Method-Pattern-vs-Abstract-Factory-Pattern

Marla Sukesh

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?

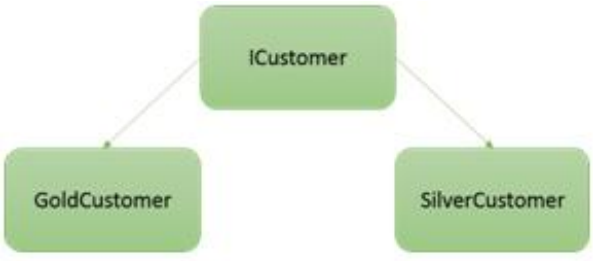
- First of all both of them falls under Creational category and it means it will solves the problem pertaining to object creation.
- Factory Method and abstract factory pattern all are about creating objects.

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



```
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.

```
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)

```
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,

```
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

```
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();
   }
}

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();
   }
}

//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