Factory Patterns - Factory Method Pattern

Snesh Prajapati

2 Oct 2016 CPOL

In this article we will understand Factory Method Pattern. This article is second part of Factory Patterns article series and is the continuation of Part 1: Simple Factory Pattern.

Download Source Code - 9 KB

Introduction

In this article we will understand Factory Method Pattern. This article is second part of Factory Patterns article series and is the continuation of Part1: Simple Factory Pattern. In first part we have learned Simple Factory Pattern. If you haven't gone through the first part, I will suggest you to do so then continue with this article.

In first part we have seen how Simple Factory Pattern helps us in object creation and found some issues with Simple Factory Pattern. In this part we will learn how Factory Method Pattern resolves those issues and in which situation we should use Factory Method Pattern.

Outlines

Part 1 - Simple Factory Pattern

Part 2 - Factory Method Pattern

- What is Factory Method Pattern
- Understand Definition with Example
- When to use Factory Method Pattern
- Advantages of Factory Method Pattern

Part 3 - Abstract Factory Pattern

What is Factory Method Pattern

Factory Method Pattern is introduced in Gang Of Four (GoF) book and falls in the category of Creational Patterns. Following is the definition of Factory Method Pattern is taken from the same book:

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

In order to be clear about this definition, we will understand this definition by breaking it into parts, in the sections of this article.

Understand Definition with Example

We will continue with the same example what we took in first part so you can understand when same application grows what type of problems comes and when we should use Factory Method Pattern (since each pattern has their own place, advantages and constraints).

Lets start with same code. We will use same interface called IFan which were having two methods as shown below.

Copy Code

```
interface IFan
{
    void SwitchOn();
    void SwitchOff();
}
```

Initially our company intented to create only three types of fans called TableFan, CeilingFan and ExhaustFan. All fans will have SwitchOn() and SwitchOff() methods and their implementation may differ. Lets use same classes and implement IFan interfaces.

Copy Code

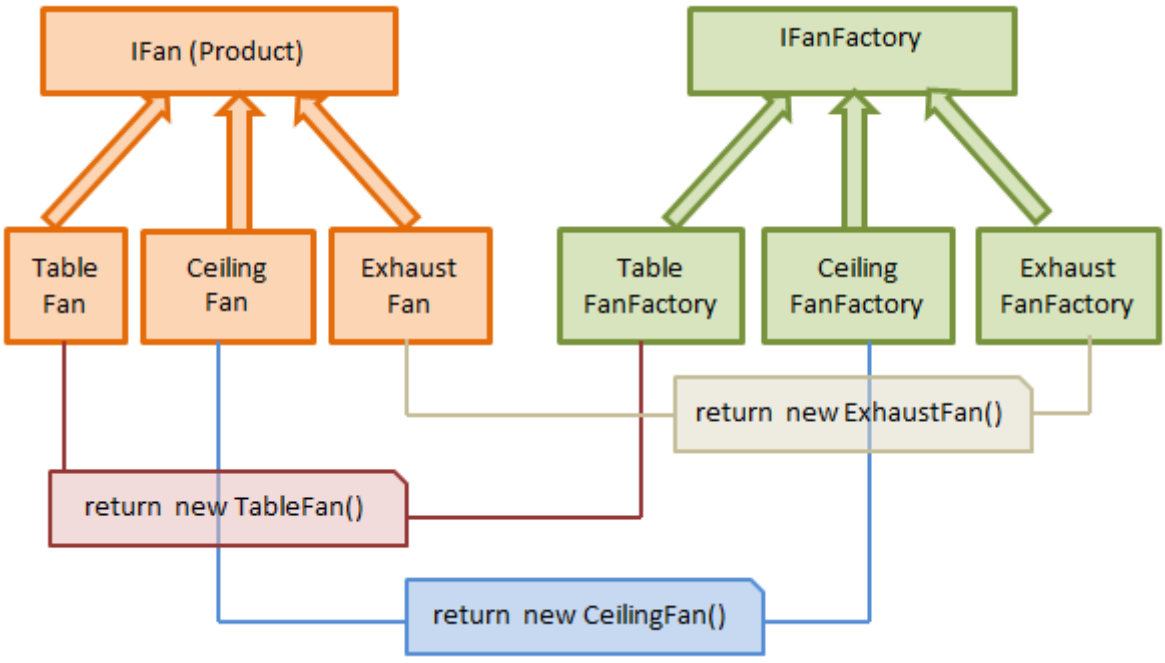
```
class TableFan : IFan {.... }

class CeilingFan : IFan {.... }

class ExhaustFan : IFan {..... }
```

So far what code we have written here, is same as we have written in “Simple Factory Pattern” example.

When we implements Factory Method Pattern, we have to follow its design as shown in below diagram:



So further we will follow its design and explain the standard definition of Factory Method Pattern.

According to definition ***"Define an interface for creating an object..."***so let’s define the interface which will have a method to create Fans. Implementation of this method in subclasses will have logic to create concrete object.

Note: we are using IFanFactory interface but instead of interface, an abstract class also can be used. Abstract class need to have at least one unimplemented abstarct method which will be implemented in subclass to create the object.

Copy Code

```
interface IFanFactory
{
    IFan CreateFan();
}
```

Since design pattern are just a particular way of design and they are language agnostic so we may also use some another approach to define base contact for object creation that would be to "Define an interface for creating an object". For example in JavaScript we do not have concept of abstract class or interface but still we can use design patterns. For more please have a look at this Rob Dodson's blog post.

Continue with definition further: ***....., let subclasses decide which class to instantiate.....***: Let's create subclasses which will decide which concrete class will be used to create instance.

To create three type of object namely TableFan, CeilingFan and ExhaustFan, we need to create three types of factories or subclasses of IFanFactory. These three factories will implement IFanFactory interface.

Copy Code

```
class TableFanFactory : IFanFactory {....}

class CeilingFanFactory : IFanFactory {....}

class ExhaustFanFactory : IFanFactory {....}
```

Again continue with definition further: ***...Factory Method lets a class defer instantiation to subclasses.***": Here `IFan CreateFan();` is a Factory Method. Inside CreateFan method (which is Factory Method) in concrete Factories (TableFanFactory etc.) we specify which exact class will be instantiated to create Fan object. So IFanFactory is deferring decision making for concrete class to its subclasses, namely to TableFanFactory, CeilingFanFactory and ExhaustFanFactory.

Now client implementation is given below:

Copy Code

```
//The client code is as follows:
static void Main(string[] args)
{
     IFanFactory fanFactory = new PropellerFanFactory();

     // Creation of a Fan using Factory Method Pattern
     IFan fan = fanFactory.CreateFan();

     // Use created object
     fan.SwitchOn();

     Console.ReadLine();
 }
```

When to use Factory Method Pattern

In the above example to create three types of objects (TableFan, CeilingFan and ExhaustFan), we have created three factory classes but while learning "Simple Factory Pattern", we created only one factory. It's because here we are following rules of Factory Method Pattern. Now question comes why we need to use Factory Method Pattern as already we can achieve same this with "Simple Factory Pattern". *Assume Company is going to launch new Fan called "PropellerFan". This is new requirement. Using Factory Method Pattern, we don't add new condition in switch case as we were doing in "Simple Factory Pattern". Simply we will create a new class called "PropellerFan" and inheret this class with IFan interface as we have done for other fans.*

Copy Code

```
class PropellerFan : IFan {..... }
```

Then will create a new factory called PropellerFanFactory and inferet it from IFanFactory. Whenever client wants to create instance of PropellerFan class. Simple by creating the instant of PropellerFanFactory, clients will get the instance of PropellerFan.

Advantages of Factory Method Pattern

- In the above section, we have seen Factory Method Pattern follows Open Close Principle. When new requirement came, we did not make changes in existing code but need to create an additonal Factory.
- Writing unit test cases is easy with Factory Method Pattern in comparison to Simple Factory Pattern since switch case (or long if else blocks) is not used.
- To support additional products, we do not modify existing code but just add one new Factory class, so no need to re-run existing old unit tests.
- Client calls CreateFan (Factory Method) without knowing how and what actual type of the object was created.
- If we are using Abstract class like BaseFanFactory (instead of IFanFactory), we can provide implementation of common methods in BaseFanFactory abstract class, only declare CreateFan method as abstract. Based on requirements, we can have more abstract methods in BaseFanFactory.

Conclusion

In this article, we had a walkthrough to learn Factory Method Pattern and its use. We understood the context of Factory Method Pattern and how it is different from Simple Factory Pattern. But it can create only one kind of products (which all implements IFan in our example), so next article we will see how we can have create a set of related products with Abstract Factory Pattern. Thanks for reading. Your comments and suggestions for improvement are most welcome.

References

1. Discussion on Motivation for Simple Factory
2. Corey Broderick's Blog
3. Coing Geek Blog
4. Article on Factory Patterns

License

Share

About the Author

Snesh Prajapati
Software Developer
India

I am a Software Developer working on Microsoft technologies. My interest is exploring and sharing the awesomeness of emerging technologies.

Comments and Discussions

You must Sign In to use this message board.

Search Comments

First Prev Next

My vote of 5
iSahilSharma 4-Mar-18 4:56

Multi simple factory
zg l 2-Apr-17 4:21

Fantastic and Good Job
Ramachandran Murugan 13-Jan-17 8:33

Re: Fantastic and Good Job
Snesh Prajapati 13-Jan-17 15:12

A little bit different implementation...
vladi7 7-Nov-16 22:11

Very clear, easy to read article with a good style.
Member 8999948 3-Oct-16 23:58

Re: Very clear, easy to read article with a good style.
Snesh Prajapati 5-Oct-16 7:13

Missing a big piece...
PureNsanity 3-Oct-16 6:58

Re: Missing a big piece...
Snesh Prajapati 3-Oct-16 16:46

Re: Missing a big piece...
katibotar@hotmail.com 15-Jul-20 12:58

Re: Missing a big piece...

PureNsanity 27-Jul-20 8:55

Factory Method Pattern
DeyChandan **3-Oct-16 0:59**

Re: Factory Method Pattern
Snesh Prajapati 3-Oct-16 16:26

please provide some info regarding abstract factory
Tridip Bhattacharjee **2-Oct-16 21:53**

Re: please provide some info regarding abstract factory
Snesh Prajapati 3-Oct-16 16:25

Generics
Afterlife99 **2-Oct-16 17:05**

Re: Generics
Snesh Prajapati 2-Oct-16 17:43

Re: Generics
PureNsanity 3-Oct-16 6:40

Re: Generics
dmjm-h 3-Oct-16 11:17

Thanks for both your posts. I had thought maybe I was missing something but no, the pattern just doesn't make sense as presented.

Sign In · View Thread

Re: Generics
Snesh Prajapati 3-Oct-16 16:56

Refresh 1

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.