عنوان: **آشنایی با الگوی طراحی Abstract Factory**
نویسنده: وحید فرهمندیان
تاریخ: ۱۳۹۳/۱۲/۰۵ ۱۹:۱۰
آدرس: www.dotnettips.info
گروه‌ها: C#, OOP, DesignPatterns

قبل از مطالعه‌ی این مطلب، حتما الگوی طراحی Factory Method را مطالعه نمایید.
همانطور که در الگوی طراحی Factory Method مشاهده شد، این الگو یک عیب دارد، آن هم این است که از کدام Creator باید استفاده شود و مستقیما در کد بایستی ذکر شود.

```
class ConcreteCreator : Creator
{
     public override IProduct FactoryMethod(string type)
    {
            switch (type)
           {
                case "A": return  new ConcreteProductA();
                case "B": return  new ConcreteProductB();
                default: throw new ArgumentException("Invalid type", "type");
           }
     }
}
```

برای حل این مشکل می‌توانیم سراغ الگوی طراحی دیگری برویم که Abstract Factory نام دارد. این الگوی طراحی 4 بخش اصلی دارد که هر کدام از این بخش‌ها را طی مثالی توضیح می‌دهم:
1. Abstract Factory: در کشور، صنعت خودروسازی داریم که خودروها را در دو دسته‌ی دیزلی و سواری تولید می‌کنند :

```
public interface IVehicleFactory     {
        IDiesel GetDiesel();
        IMotorCar GetMotorCar();
    }
```

2. Concrete Factory: دو کارخانه‌ی تولید خودرو داریم که در صنعت خودرو سازی فعالیت دارند و عبارتند از ایران خودرو و سایپا که هر کدام خودروهای خود را تولید می‌کنند. ولی هر خودرویی که تولید می‌کنند یا دیزلی است یا سواری. شرکت ایران خودرو، خودروی آرنا را بعنوان دیزلی تولید می‌کند و پژو 206 را بعنوان سواری. همچنین شرکت سایپا خودروی فوتون را بعنوان خودروی دیزلی تولید می‌کند و خودروی پراید را بعنوان خودروی سواری.

```
public class IranKhodro : IVehicleFactory
    {
        public IDiesel GetDiesel() { return new Arena(); }
        public IMotorCar GetMotorCar() { return new Peugeot206(); }
    }
    public class Saipa : IVehicleFactory
    {
        public IDiesel GetDiesel() { return new Foton(); }
        public IMotorCar GetMotorCar() { return new Peride(); }
    }
```

3. Abstract Product: خودروهای تولیدی همانطور که گفته شد یا دیزلی هستند یا سواری که هر کدام از این خودروها ویژگی‌های خاص خود را دارند (در این مثال هر دو دسته خودرو برای خود نام دارند)

```
public interface IDiesel { string GetName();}
    public interface IMotorCar { string GetName();}
```

4. Concrete Product: در بین این خودروها، خودروی پژو 206 و پراید یک خودروی سواری هستند و خودروی فوتون و آرنا، خودروهای دیزلی.

```
public class Foton : IDiesel { public string GetName() { return "This is Foton"; } }
    public class Arena : IDiesel { public string GetName() { return "This is Arena"; } }
    public class Peugeot206 : IMotorCar { public string GetName() { return "This is Peugeot206"; } }
```

```
    public class Peride : IMotorCar { public string GetName() { return "This is Peride"; } }
```

حال که 4 دسته اصلی این الگوی طراحی را آموختیم می‌توان از آن بصورت زیر استفاده نمود:

```
IVehicleFactory factory = new IranKhodro();
 Console.WriteLine("***" + factory.GetType().Name + "***");
 IDiesel diesel = factory.GetDiesel();
 Console.WriteLine(diesel.GetName());
 IMotorCar motorCar = factory.GetMotorCar();
 Console.WriteLine(motorCar.GetName());

 factory = new Saipa();
 Console.WriteLine("***" + factory.GetType().Name + "***");
 diesel = factory.GetDiesel();
 Console.WriteLine(diesel.GetName());
 motorCar = factory.GetMotorCar();
 Console.WriteLine(motorCar.GetName());
```

همانطور که در کد فوق مشاهده میشود، ایراد موجود در الگوی Factory Method اینجا از بین رفته است و برای ساخت آبجکت‌های مختلف از Innterfaceها یا Abstract Classها استفاده می‌کنیم.

کلا Abstract Factory مزایای زیر را دارد:

پیاده سازی و نامگذاری Product در Factory مربوطه متمرکز می‌شود و بدین ترتیب Client به نام و نحوه پیاده سازی Typeهای مختلف Product وابستگی نخواهد داشت.

به راحتی می‌توان Concrete Factory مورد استفاده در برنامه را تغییر داد، بدون اینکه تاثیری در عملکرد سایر بخش‌ها داشته باشد.

در مواردی که بیش از یک محصول برای هر خانواده وجود داشته باشد، استفاده از Abstract Factory تضمین می‌کند که Productهای هر خانواده همه در کنار هم قرار دارند و با هم فعال و غیر فعال می‌شوند. (یا همه، یا هیچکدام)

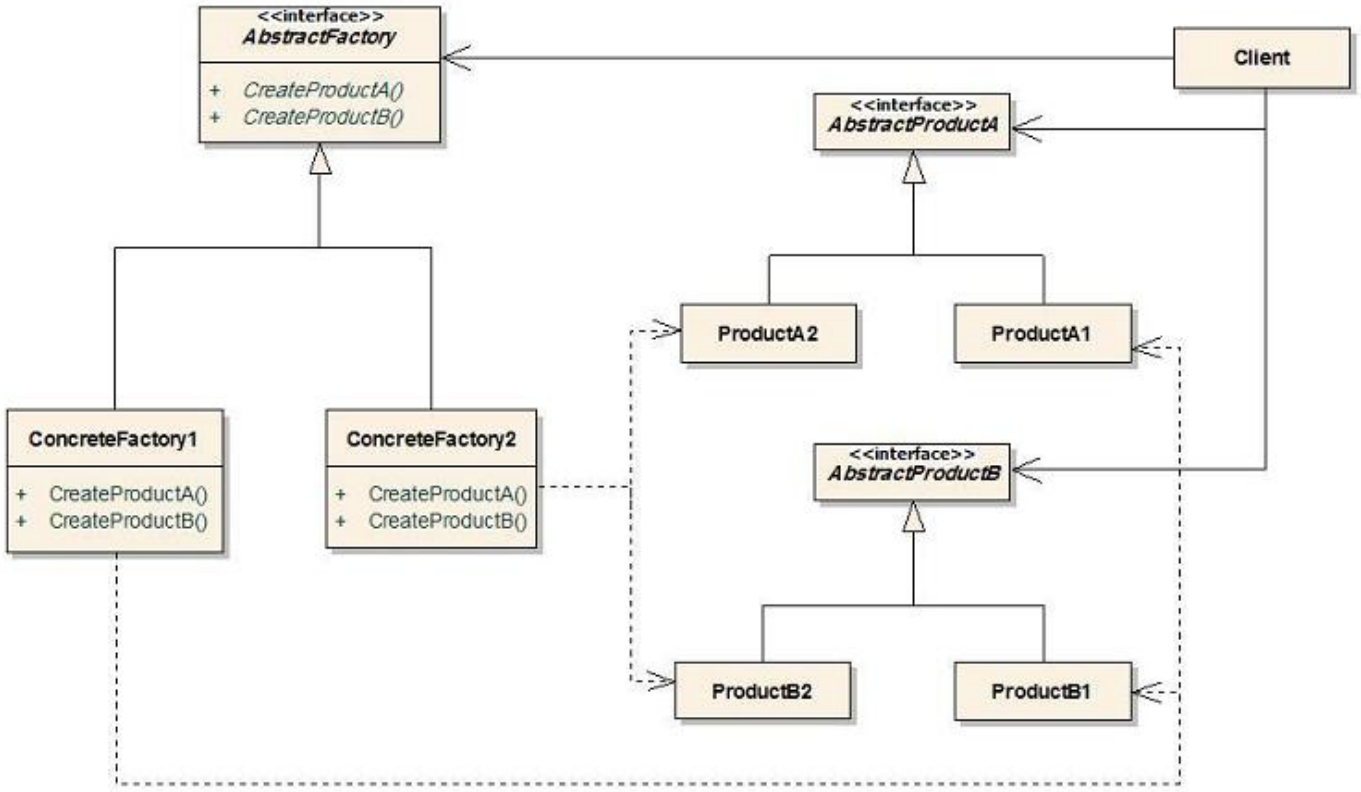
بزرگترین عیبی که این الگوی طراحی دارد این است که با اضافه شدن فقط یک Product تازه، Abstract Factory باید تغییر کند که این مساله منجر به تغییر همه Concrete Factoryها می‌شود.

نهایتا اینکه در استفاده از این الگوی طراحی به این تکنیک‌ها توجه داشته باشید:

Factoryها معمولا Singleton هستند. زیرا هر Application **بطور معمول** فقط به یک instance از هر Concrete Factory نیاز دارد.

انتخاب Concrete Factory مناسب معمولا توسط پارامترهایی انجام می‌شود.

نمودار کلاسی این الگو نیز بصورت زیر میباشد:



و کلام آخر در مورد این الگو:

Abstract Factory یک interface یا کلاس abstract است که signature متدهای ساخت Objectها در آن تعریف شده است و Concrete Factoryها آن‌ها را implement می‌نمایند.

در Abstract Factory Pattern همه Productهای هم خانواده در Concrete Factory مربوط به آن خانواده پیاده سازی و مجتمع می‌گردند.

در کدهای برنامه تنها با Abstract Factory و Abstract Productها سر و کار داریم و به هیچ وجه درگیر این مساله که کدام یک از Concrete Classها در برنامه مورد استفاده قرار می‌گیرند، نمی‌شویم.