آشنایی با الگوهای طراحی(Design Patterns)

يكي از مهمترين دلايل محققين علوم رايانه براي ايجاد الگوهاي طراحي ياDesign Patterns ، برآورده كردن نياز به كدهاي زيبا، ساده و قابل استفاده مجدد مي باشد. به طور خلاصه مي توان گفت :

•الگوهاي طراحي راهكارهايي هستند كه مي توانند براي حل مشكلات كدنويسي شما بارها مورد استفاده قرار گيرند

•الگوهاي طراحي مشخص كننده مفاهيم انتزاعي هستند كه در سطحي بالاتر از كلاسها، نمونه ها (instances)و يا كمپونتها قرار مي گيرند.

اما مي توان گفت كه الگوهاي طراحي تنها براي بيان چگونگي طراحي اشياء بيان نمي شوند، بلكه ارتباط و تعامل بين اشيا را نيز شرح مي دهند و حتي بيانگر راههايي براي چگونگي ارث بري و شرايط و محدوديت هاي آنها نيز هستند.

در حقيقت الگوهاي طراحي نوشته نمي شوند، بلكه كشف مي گردند و الگوهاي متداولي كه امروزه به وفور در مقالات و كتابها از آنها ياد مي شود، الگوهايي هستند كه در برنامه هاي متعدد و شناخته شده بارها مورد استفاده قرار گرفته اند و مي توانند بسياري از ديگر برنامه ها را نيز تحت پوشش خود قرار دهند.

به طور كلي الگوهاي طراحي به سه دسته تقسيم مي شوند:

1- الگوهاي آفرينشي:(Creational patterns)

اين الگوها، نمونه هاي اشياء را براي شما ايجاد مي كنند، بدون اينكه شما را در آن عمل دخيل نمايند.

2- الگوهاي ساختاري:(Structural patterns)

به شما كمك مي كنند كه گروههاي اشياء را در ساختارهاي بزرگتر پياده سازي كنيد يا به عبارت ديگر كلاسها و اشيا را با هم تركيب كنيد.

3- الگوهاي رفتاري :(Behavioral patterns)

به شما كمك مي كنندكه ارتباط بين اشياء را در سيستم خود تعريف كنيد و يا چگونگي تعامل بين كلاسها و نحوه توزيع مسئوليت بين آنها را مورد بحث قرار دهيد.

الگوهاي آفرينشي:

1- Abstract Factory : رابط (اينترفيسي) ايجاد مي كند كه يک مجموعه از اشياء مرتبط را ساخته و برمي گرداند.

2- Builder : ساخته شدن يک شي ء پيچيده را از نمايش آن جدا مي كند. بنابراين بسته به نيازهاي برنامه چندين نمايش متفاوت از شي ء مي تواند ايجاد گردد .

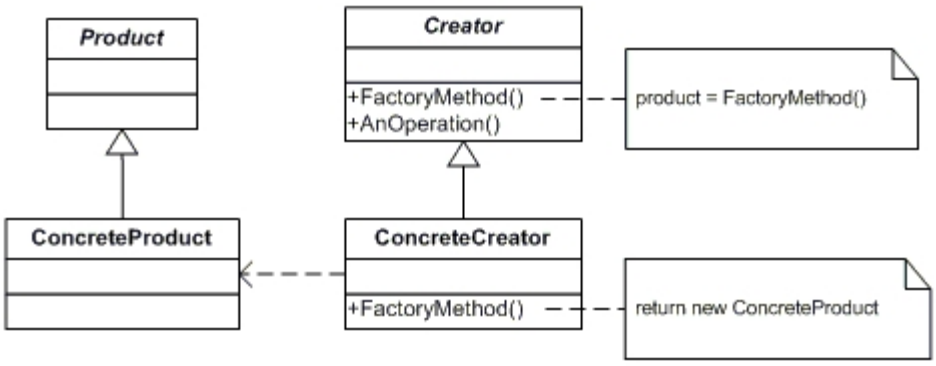
3- Prototype : با ايجاد يک كلاس نمونه شروع شده و براي ساختن ساير نمونه هاي جديد، شبيه سازي يا كلون (Clone) انجام مي شود. اين نمونه ها بعدا مي توانند توسط متدهاي عمومي خود بهم متصل شوند.

4- Singleton : كلاسي است كه نمي تواند بيش از يک نمونه شي ء داشته باشد. به اين ترتيب تنها يک نقطه دسترسي عمومي از نمونه شي ء ايجاد مي گردد.

در اين مقاله به منظور آشنايي مقدماتي با الگوي Abstract Factory با يک الگوي مقدماتي اما متداول به نام Factory Method آشنا مي شويم. در اين الگو هيچ كلاسي به تنهايي تصميم نمي گيرد كه چه زير كلاسي از آن نمونه گيري شود. بلكه تصميم گيري را به كلاسهاي پايين تر واگذار مي نمايد.

در اين الگو در واقع يک نقطه تصميم گيري براي نمونه گيري از كلاسها وجود ندارد و به جاي آن، برنامه هاي نوشته شده در اين الگو، يک كلاس انتزاعي (Abstract) تعريف مي كنند كه اشيا را خلق مي كند اما به زير كلاسها اجازه مي دهد تا تصميم بگيرند كداميک از اشيا ايجاد شوند .در واقع اين مسئله با تعريف يک متد جداگانه براي ايجاد اشيا حل مي گردد كه زيركلاسها مي توانند آنرا Override كنند و به اين وسيله نوع محصول مشتق شده را كه ايجاد خواهد شد، مشخص نمايند.

شكل زير نماينده كلاس دياگرام UML از اين الگو مي باشد:



كلاسها و يا اشيايي كه در ابن دياگرام وجود دارند عبارتند از:

1- Product : (به عنوان مثال صفحه) يک اينترفيس يا رابط از اشبايي كه Factory method ايجاد مي كند، تعريف مي نمايد.

2- ConcreteProduct : (به عنوان مثال صفحه مهارتها، صفحه آموزش و ...) شامل پياده سازي اينترفيس Product مي باشد.

3- Creator : (به عنوان مثال سند) معرف FactoryMethod است كه يک شي ء از نوع Product را برمي گرداند.همچنين Creator مي تواند يک پياده سازي از FactoryMethod را تعريف كند كه يک شي ء از نوع ConcreteProduct را برمي گرداند. همچنين ممكن است FactoryMethod را براي ايجادد شي ء از نوع Product صدا بزند.

4- ConcreteCreator : با بازنويسي (Override) متد Factory يک نمونه از ConcreteProduct را برمي گرداند.

نمونه كد ساده اي براي روشن شدن اين مفاهيم آوره شده است:

```
// Factory Method pattern -- Structural example
using System;
using System.Collections;
namespace DoFactory.GangOfFour.Factory.Structural
{
  // MainApp test application
  class MainApp
  {
    static void Main()
```

```
  {
    // An array of creators
    Creator[] creators = new Creator[2];
    creators[0] = new ConcreteCreatorA();
    creators[1] = new ConcreteCreatorB();
    // Iterate over creators and create products
    foreach(Creator creator in creators)
    {
      Product product = creator.FactoryMethod();
      Console.WriteLine("Created {0}",
        product.GetType().Name);
    }
    // Wait for user
    Console.Read();
  }
}
// "Product"
abstract class Product
{
}
// "ConcreteProductA"
class ConcreteProductA : Product
{
}
// "ConcreteProductB"
class ConcreteProductB : Product
{
}
// "Creator"
abstract class Creator
{
  public abstract Product FactoryMethod();
}
// "ConcreteCreator"
class ConcreteCreatorA : Creator
{
  public override Product FactoryMethod()
  {
    return new ConcreteProductA();
  }
}
// "ConcreteCreator"
class ConcreteCreatorB : Creator
{
  public override Product FactoryMethod()
  {
```

```
    return new ConcreteProductB();
  }
 }
}
```
//خروجي
Created ConcreteProductA
Created ConcreteProductB