عنوان: آشنایی با الگوی طراحی Decorator
نویسنده: وحید فرهمندیان
تاریخ: ۱۳۹۳/۱۲/۰۴ ۱۴:۵
آدرس: www.dotnettips.info
گروه‌ها: C#, OOP, DesignPatterns

این بار مثال را با شیرینی و کیک پیش می‌بریم.
فرض کنید شما قصد پخت کیک و نان را دارید. طبیعی است که برای اینکار یک واسط را تعریف کرده و عمل «پختن» را در آن اعلام می‌کنید تا هر کلاسی که قصد پیاده سازی این واسط را داشت، «پختن» را انجام دهد. در ادامه یک کلاس بنام کیک ایجاد خواهید کرد و شروع به پخت آن می‌کنید.
خوب احتمالا الان کیک آماده‌است و می‌توانید آن‌را میل کنید! ولی یک سؤال. تکلیف شخصی که کیک با روکش کاکائو دوست دارد و شمایی که کیک با روکش میوه‌ای دوست دارید چیست؟ این را چطور در پخت اعمال کنیم؟ یا منی که نان کنجدی می‌خواهم و شمایی که نان برشته‌ی غیر کنجدی می‌خواهید چطور؟
احتمالا می‌خواهید سراغ ارث بری رفته و سناریوهای این چنینی را پیاده سازی کنید. ولی در مورد ارث بری، اگر کلاس sealed ((NotInheritable باشه چطور؟
احتمالا همین دو تا سؤال کافی‌است تا در پاسخ بگوئیم، گره‌ی کار، با الگوی Decorator باز می‌شود و همین دو تا سؤال کافی‌است تا اعلام کنیم که این الگو، از جمله الگوهای بسیار مهم و پرکاربرد است.
در ادامه سناریوی خود را با کد ذیل جلو می‌بریم:

```
public interface IBakery
    {
        string Bake();
        double GetPrice();
    }
    public class Cake: IBakery
    {
        public string Bake() { return "Cake baked"; }
        public double GetPrice() { return 2000; }
    }
    public class Bread: IBakery
    {
        public string Bake() { return "Bread baked"; }
        public double GetPrice() { return 100; }
    }
```

در کد فوق فرض کرده‌ام که شما می‌خواهید محصول خودتان را بفروشید و برای آن یک متد GetPrice نیز گذاشته‌ام. خوب در ابتدا واسطی تعریف شده و متدهای Bake و GetPrice اعلام شده‌اند. سپس کلاس‌های Cake و Bread پیاده سازی‌های خودشان را انجام دادند.
در ادامه باید مخلفاتی را که روی کیک و نان می‌توان اضافه کرد، پیاده نمود.

```
public abstract class Decorator : IBakery
    {
        private readonly IBakery _bakery;
        protected string bake = "N/A";
        protected double price = -1;

        protected Decorator(IBakery bakery) { _bakery= bakery; }
        public virtual string Bake() { return _bakery.Bake() + "/" + bake; }
        public double GetPrice() { return _bakery.GetPrice() + price; }
    }
    public class Type1 : Decorator
    {
        public Type1(IBakery bakery) : base(bakery) { bake= "Type 1"; price = 1; }
    }
    public class Type2 : Decorator
    {
        private const string bakeType = "special baked";
        public Type2(IBakery bakery) : base(bakery) { name = "Type 2"; price = 2; }
        public override string Bake() { return base.Bake() + bakeType ; }
    }
```

در کد فوق یک کلاس انتزاعی ایجاد و متدهای پختن و قیمت را پیاده سازی کردیم؛ همچنین کلاس‌های Type1 و Type2 را که من

فرض کردم کلاس‌هایی هستند برای اضافه کردن مخلفات به کیک و نان. در این کلاس‌ها در متد سازنده، یک شیء از نوع IBakery می‌گیریم که در واقع این شیء یا از نوع Cake هست یا از نوع Bread و مشخص می‌کند روی کیک می‌خواهیم مخلفاتی را اضافه کنیم یا بر روی نان. کلاس Type1 روش پخت و قیمت را از کلاس انتزاعی پیروی می‌کند، ولی کلاس Type2 روش پخت خودش را دارد.
با بررسی اجمالی در کدهای فوق مشخص می‌شود که هرگاه بخواهیم، می‌توانیم رفتارها و الحاقات جدیدی را به کلاس‌های Cake و Bread، اضافه کنیم؛ بدون آنکه کلاس اصلی آنها تغییر کند. حال شما شاید در پیاده سازی این الگو از کلاس انتزاعی Decorator هم استفاده نکنید.
با این حال شیوه‌ی استفاده از این کدها هم بصورت زیر خواهد بود:

```
Cake cc1 = new Cake();
 Console.WriteLine(cc1.Bake() + " ," + cc1.GetPrice());

 Type1 cd1 = new Type1 (cc1);
 Console.WriteLine(cd1.Bake() + " ," + cd1.GetPrice());

 Type2 cd2 = new Type2(cc1);
 Console.WriteLine(cd2.Bake() + " ," + cd2.GetPrice());
```

ابتدا یک کیک را پختیم در ادامه Type1 را به آن اضافه کردیم که این باعث می‌شود قیمتش هم زیاد شود و در نهایت Type2 را هم به کیک اضافه کردیم و حالا کیک ما آماده است.

نظرات خوانندگان

نویسنده: محمد اسکندری
تاریخ: ۱۳۹۳/۱۲/۰۵ ۱۰:۵۴

در استفاده از الگوی دکوراتور روش بهتر بهره گیری از آن بصورت سری است و نه ایجاد شیء جدید برای تایپ جدید.
مثلا:

```
Cake c = new Cake();

c = new Type1(c);
c = new SubType(c); //SubType derived from  Cake (e.g. CakeComponent like Cream)

//or: c = new Type1 (new SubType(c));

Console.WriteLine(c.Bake() + ", " + c.GetPrice());
```

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۱۲/۰۵ ۱۱:۴۴

بستگی به هدف نهایی دارد. اگر هدف تولید کیک با روکش کاکائویی و روکش میوه‌ای به صورت همزمان است، نحوه‌ی تزئین آن با کیکی که فقط قرار هست روکش کاکائویی داشته باشه، فرق می‌کنه.

نویسنده: محمد اسکندری
تاریخ: ۱۳۹۳/۱۲/۰۵ ۱۲:۰

فرقی نمیکنه. اگر قرار بود فرق میکرد و نیاز به ایجاد تغییرات در کد بود که این الگوها ارائه نمی‌شدند.

```
// ساخت کیک معمولی با روکش کاکائویی
Cake c = new CommonCake();
c = new Chocolate(c);

// ساخت کیک معمولی با روکش میوه‌ای
Cake c = new CommonCake();
c = new Fruity(c);

// ساخت کیک معمولی مخلوط با روکش کاکائویی و روکش میوه‌ای به صورت همزمان
Cake c = new CommonCake();
c = new Chocolate(c);
c = new Fruity(c);

// ساخت کیک مخصوص مخلوط با روکش کاکائویی و روکش میوه‌ای به صورت همزمان
Cake c = new SpecialCake();
c = new Chocolate(c);
c = new Fruity(c);
```

برای هر c میتوان متدهای اینترفیسش را اجرا کرد.

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۱۲/۰۵ ۱۲:۸

عنوان کردید «در استفاده از الگوی دکوراتور روش بهتر بهره گیری از آن بصورت **سری** است و نه ایجاد شیء **جدید** برای تایپ جدید»، بعد الان برای تهیه روکش فقط میوه‌ای از حالت سری استفاده نکردید و یک وهله جدید ایجاد شده. بحث بر سر سری بودن یا نبودن مراحل بود. بنابراین بسته به هدف، می‌تونه سری باشه یا نباشه و اگر نبود، مشکلی نداره، چون هدفش تولید یک روکش مخصوص بوده و نه ترکیبی.

نویسنده: محمد اسکندری
تاریخ: ۱۳۹۳/۱۲/۰۵ ۱۲:۲۳

فرض من این بود که کاربر نیازی به رفرنس گیری از هر آبجکت ندارد.
مثلا طبق مقاله:

```
// ساخت کیک مخصوص مخلوط با روکش کاکائویی و روکش میوه‌ای به صورت همزمان
Cake c = new SpecialCake();
Chocolate ch = new Chocolate(c);
Fruity f = new Fruity(ch);
```

همانطور که در مقاله گفته شده:

```
Cake cc1 = new Cake();
Type1 cd1 = new Type1 (cc1);
Type2 cd2 = new Type2(cc1);
```

کد فوق را میتوان اینگونه **هم** داشت:

```
// ساخت کیک مخصوص مخلوط با روکش کاکائویی و روکش میوه‌ای به صورت همزمان
Cake c = new SpecialCake();
c = new Chocolate(c);
c = new Fruity(c);
```

بدون اینکه *شیء جدید برای تایپ جدید* بسازیم.

نویسنده: محسن خان
تاریخ: ۱۳۹۳/۱۲/۰۵ ۱۲:۴۳

مهم این نیست که نام تمام متغیرها را c تعریف کردید، مهم این است که به ازای هر new یک شیء کاملا جدید ایجاد می‌شود که ریفرنس آن با ریفرنس قبلی یکی نیست.