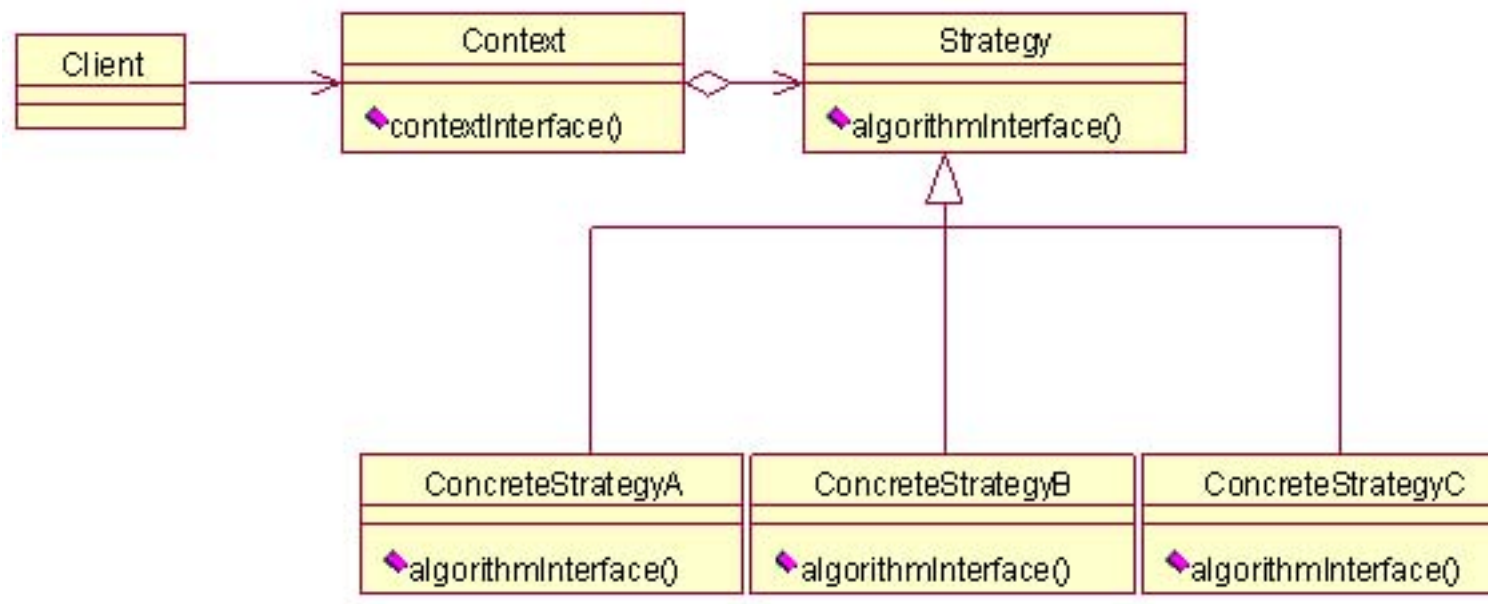
عنوان: **الگوی استراتژی - Strategy Pattern**
نویسنده: مجتبی شاطری
تاریخ: ۱۳۹۲/۰۶/۲۰ ۰:۰
آدرس: www.dotnettips.info
برچسب‌ها: Design patterns, Dependency Injection, SOLID Principals, Inheritance Strategies, SOLID, Dependency Inversion

الگوی استراتژی (Strategy) اجازه می-دهد که یک الگوریتم در یک کلاس بسته بندی شود و در زمان اجرا برای تغییر رفتار یک شیئ تعویض شود.
برای مثال فرض کنید که ما در حال طراحی یک برنامه مسیریابی برای یک شبکه هستیم. همانطوریکه می‌دانیم برای مسیر یابی الگوریتم‌های مختلفی وجود دارد که هر کدام دارای مزایا و معایبی هستند. و با توجه به وضعیت موجود شبکه یا عملی که قرار است انجام پذیرد باید الگوریتمی را که دارای بالاترین کارائی است انتخاب کنیم. همچنین این برنامه باید امکانی را به کاربر بدهد که کارائی الگوریتم‌های مختلف را در یک شبکه فرضی بررسی کنید. حالا طراحی پیشنهادی شما برای این مسئله چیست؟
دوباره فرض کنید که در مثال بالا در بعضی از الگوریتم‌ها نیاز داریم که گره‌های شبکه را بر اساس فاصله‌ی آنها از گره مبداء مرتب کنیم. دوباره برای مرتب سازی الگوریتم‌های مختلف وجود دارد و هر کدام در شرایط خاص، کارائی بهتری نسبت به الگوریتم‌های دیگر دارد. مسئله دقیقا شبیه مسئله بالا است و این مسله می‌توانند دارای طراحی شبیه مسله بالا باشد. پس اگر ما بتوانیم یک طراحی خوب برای این مسئله ارائه دهیم می‌توانیم این طراحی را برای مسائل مشابه به کار ببریم.

هر کدام از ما می‌توانیم نسبت به درک خود از مسئله و سلیقه کاری، طراح‌های مختلفی برای این مسئله ارائه دهیم. اما یک طراحی که می‌تواند یک جواب خوب و عالی باشد، الگوی استراتژی است که توانسته است بارها و بارها به این مسئله پاسخ بدهد.

الگوی استراتژی گزینه مناسبی برای مسائلی است که می‌توانند از چندین الگوریتم مختلف به مقصود خود برسند.

نمودار UML الگوی استراتژی به صورت زیر است :




اجازه بدهید، شیوه کار این الگو را با مثال مربوط به مرتب سازی بررسی کنیم. فرض کنید که ما تصمیم گرفتیم که از سه الگویتم زیر برای مرتب سازی استفاده کنیم.

1 - الگوریتم مرتب سازی Shell Sort 2 - الگوریتم مرتب سازی Quick Sort
3 - الگوریتم مرتب سازی Merge Sort

ما برای مرتب سازی در این برنامه دارای سه استراتژی هستیم. که هر کدام را به عنوان یک کلاس جداگانه در نظر می‌گیریم (همان کلاس‌های ConcreteStrategy ). برای اینکه کلاس Client بتواند به سادگی یک از استراتژی‌ها را انتخاب کنید بهتر است که تمام کلاس‌های استراتژی دارای اینترفیس مشترک باشند. برای این کار می‌توانیم یک کلاس abstract تعریف کنیم و ویژگیهای مشترک کلاس‌های استراتژی را در آن قرار دهیم و کلاس‌های استراتژی آنها را به ارث ببرند(همان کلاس Strategy ) و پیاده سازی کنند.

در زیل کلاس Abstract که کل کلاس‌های استراتژی از آن ارث می‌برند را مشاهده می‌کنید :

```
abstract class SortStrategy
{
        public abstract void Sort(ArrayList list);
}
```

کلاس مربوط به QuickSort

```
class QuickSort : SortStrategy
{
        public override void Sort(ArrayList list)
        {
          // الگوریتم مربوطه
        }


}
```

کلاس مربوط به ShellSort

```
class ShellSort : SortStrategy
{
        public override void Sort(ArrayList list)
        {
          // الگوریتم مربوطه
        }
}
```

کلاس مربوط به MergeSort

```
class MergeSort : SortStrategy
{
        public override void Sort(ArrayList list)
        {
          // الگوریتم مربوطه
        }
}
```

و در آخر کلاس Context که یکی از استراتژی‌ها را برای مرتب کردن به کار می‌برد :

```
class SortedList
{
        private ArrayList list = new ArrayList();
        private SortStrategy sortstrategy;

        public void SetSortStrategy(SortStrategy sortstrategy)
        {
            this.sortstrategy = sortstrategy;
        }
        public void Add(string name)
```

```
        {
            list.Add(name);
        }
        public void Sort()
        {
            sortstrategy.Sort(list);
        }
}
```

## نظرات خوانندگان

نویسنده: علي
تاریخ: ۱۳۹۲/۰۶/۲۰ ۱۲:۴۶

با سلام؛ لطفا کلاس آخری را بیشتر توضیح دهید.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۶/۲۰ ۱۲:۵۵

کلاس آخری با یک پیاده سازی عمومی کار می‌کنه. دیگه نمی‌دونه نحوه مرتب سازی چطور پیاده سازی شده. فقط می‌دونه یک متد Sort هست که دراختیارش قرار داده شده. حالا شما راحت می‌تونی الگوریتم مورد استفاده رو عوض کنی، بدون اینکه نیاز داشته باشی کلاس آخری رو تغییر بدی. باز هست برای توسعه. بسته است برای تغییر. به این نوع طراحی رعایت open closed principle هم می‌گن.

نویسنده: SB
تاریخ: ۱۳۹۲/۰۶/۲۰ ۱۴:۲۳

بنظر شما متد Sort کلاس اولیه، نباید از نوع Virtual باشد؟

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۶/۲۰ ۱۴:۴۸

نوع کلاسش abstract هست.

نویسنده: مجتبی شاطری
تاریخ: ۱۳۹۲/۰۶/۲۰ ۱۶:۴۷

در صورتی از virtual استفاده می‌کنیم که یک پیاده سازی از متد Sort در SortStrategy داشته باشیم ، اما در اینجا طبق فرموده دوستمون کلاس ما فقط انتزاعی (Abstract) هست.

نویسنده: سید ایوب کوکبي
تاریخ: ۱۳۹۲/۰۶/۳۱ ۱۱:۲۲

چرا استراتژی توسط Abstract پیاده سازی شده و از اینترفیس استفاده نشده؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۶/۳۱ ۱۲:۵۱

تفاوت مهمی نداره ؛ فقط اینترفیس ورژن پذیر نیست. یعنی اگر در این بین متدی رو به تعاریف اینترفیس خودتون اضافه کردید، تمام استفاده کننده‌ها مجبور هستند اون رو پیاده سازی کنند. اما کلاس Abstract می‌تونه شامل یک پیاده سازی پیش فرض متد خاصی هم باشه و به همین جهت ورژن پذیری بهتری داره.
بنابراین کلاس Abstact یک اینترفیس است که می‌تواند پیاده سازی هم داشته باشد.
همین مساله خاص نگارش پذیری، در طراحی ASP.NET MVC به کار گرفته شده: ( ^ )
برای من نوعی شاید این مساله اهمیتی نداشته باشه. اگر من قرارداد اینترفیس کتابخانه خودم را تغییر دادم، بالاخره شما با یک حداقل نق زدن مجبور به به روز رسانی کار خودتان خواهید شد. اما اگر مایکروسافت چنین کاری را انجام دهد، هزاران نفر شروع خواهند کرد به بد گفتن از نحوه مدیریت پروژه تیم‌های مایکروسافت و اینکه چرا پروژه جدید آن‌ها با یک نگارش جدید MVC کامپایل نمی‌شود. بنابراین انتخاب بین این دو بستگی دارد به تعداد کاربر پروژه شما و استراتژی ورژن پذیری قرار دادهای کتابخانه‌ای که ارائه می‌دهید.

نویسنده: سید ایوب کوکبی
تاریخ: ۱۳۹۲/۰۶/۳۱ ۱۳:۲۷

اطلاعات خوبی بود، ممنون، ولی با توجه به تجربه تون، در پروژه‌های متن باز فعلی تحت بستر دات نت بیشتر از کدام مورد استفاده میشه؟ اینترفیس روحیه نظامی خاصی به کلاس‌های مصرف کننده اش میده، یه همین دلیل من زیاد رقبت به استفاده از اون ندارم، آیا مواردی هست که چاره ای نباشه حتما از یکی از این دو نوع استفاده بشه؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۶/۳۱ ۱۳:۴۹

- اگر پروژه خودتون هست، از اینترفیس استفاده کنید. تغییرات آن و نگارش‌های بعدی آن تحت کنترل خودتان است و build دیگران را تحت تاثیر قرار نمی‌دهد.
- در پروژه‌های سورس باز دات نت، عموما از ترکیب این دو استفاده می‌شود. مواردی که قرار است در اختیار عموم باشند حتی دو لایه هم می‌شوند. مثلا در MVC یک اینترفیس IController هست و بعد یک کلاس Abstract به نام Controller، که این اینترفیس را پیاده سازی کرده برای ورژن پذیری بعدی و کنترلرهای پروژه‌های عمومی MVC از این کلاس Abstract مشتق می‌شوند یا در پروژه RavenDB از کلاس‌های Abstract زیاد استفاده شده، مانند AbstractIndexCreationTask و AbstractMultiMapIndexCreationTask و غیره.

نویسنده: جمشیدی فر
تاریخ: ۱۳۹۲/۰۷/۰۱ ۱۶:۱۱

توابع abstract بطور ضمنی virtual هستند.

نویسنده: جمشیدی فر
تاریخ: ۱۳۹۲/۰۷/۰۱ ۱۸:۳۸

در کلاس abstract نیز می‌توان از پیاده سازی پیشفرض استفاده کرد . یکی از تفاوتهای کلاس abstract با Interface همین ویژگی است که سبب ورژن پذیری آن شده است.

نویسنده: جمشیدی فر
تاریخ: ۱۳۹۲/۰۸/۲۱ ۹:۱۵

بهتر نیست در کلاس SortedList برای مشخص کردن استراتژی مرتب سازی، از روش تزریق وابستگی - Dependency Injection - استفاده بشه؟

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۸/۲۱ ۹:۲۵

خوب، الان هم وابستگی کلاس یاد شده از طریق سازنده آن در اختیار آن قرار گرفته و داخل خود کلاس وهله سازی نشده. (در این مطلب طراحی بیشتر مدنظر هست تا اینکه حالا این وابستگی به چه صورتی و کجا قرار هست وهله سازی بشه و در اختیار کلاس قرار بگیره؛ این مساله ثانویه است)

نویسنده: جمشیدی فر
تاریخ: ۱۳۹۲/۰۸/۲۱ ۱۱:۴۳

از طریق سازنده کلاس SortedList؟ بنظر نمیاداز طریق سازنده انجام شده باشه. ولی ظاهرا این امکان هست که کلاس بالادستی که می‌خواهد از SortedList استفاده کند، بتواند از طریق تابع SetSortStrategy کلاس مورد نظر رادر اختیار SortedList قراردهد. به نظر شبیه Setter Injection می‌شود.