- شماره مقاله : ۱
- سطح مقاله : مبتدي
- عنوان : آشنايي مقدماتي با مفهوم Polymorphism و پياده سازي آن در سي شارپ
- توسط: Mojgan110 http://mojgan110.wordpress.com
- تاريخ : دوم دي ماه ۱۳۸۵
- حرف امروز ! : تو با اون دعفا داري ، اون هم با تو ، من چيكاره بيدم ؟

همه زبانهای شی گرا ، از سه اصل پایه ای شی گرایی حمایت میکنند که عبارتند از :

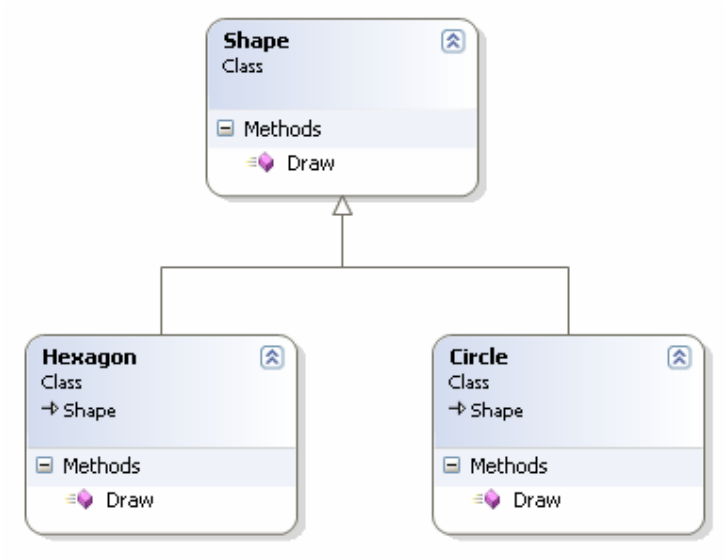
- Encapsulation : بحث درباره اینکه زبان مورد نظر ،چگونه پیاده سازی داخلی اشیا را از کاربر پنهان میکند.
- Inheritance : بحث درباره اینکه زبان موردنظر ، چگونه استفاده مجدد از کد را ممکن میکند .
- Polymorphism: بحث درباره اینکه زبان مورد نظر چگونه امکان میدهد که با آبجکت های مرتبط بهم ، رفتار مشابهی داشته باشید.

در این نوشته ، به سومین ستون از شی گرایی ، یهنی Polymorphism میپردازیم.

همانطور که گفته شد ، Polymorphism این امکان را فراهم میکند که با آبجکت های مرتبط بهم ، رفتار مشابهی داشته باشیم . به کمك پلی مرفیسم ، یك کلاس پایهbase class میتونه مجموعه ای از اعضا را برای همه فرزندانش که از اون ارث میبرند ، فراهم کنه که در اصطلاح به اون Polymorphic Interface میگویند و با هرتعداد اعضای abstract یا virtual میتوان آنرا ساخت.

در یك جمله ساده ، میتوان اینطور خلاصه کرد که یك عضو virtual **میتواند** توسط کلاسهایی که آنرا به ارث میبرند ، تغییر داشته شود و یا اصطلاحا override شود. در حالیکه یك عضو abstract ، **بایستی** توسط کلاسهای ارث برنده ، override شود. وقتی داریم اعضا را override میکنیم ، درواقع داریم نحوه پاسخگویی و رفتار آنها را به یك درخواست یکسان ، تعریف میکنیم.

برای مثال به شکل زیر نگاه کنید :



در اینجا یك کلاس پایه داریم بنام Shape که یك سری اعمالی را واسه اشکال تعریف میکنه ، برای سادگی فرض میکنیم که فقط یك کار بشه با کلاس Shape انجام داد و آن هم ترسیم شکل باشد و متدی بنام Draw برای آن ساخته باشیم.

بسیارخب ، کلاس Shape به نظر میرسه که زیادی کلی باشه ! خب شکلهای متفاوتی داریم ، مثلا دایره و چند ضلعی . پس این نیاز احساس میشود که بایستی کلاسهای جدیدی هم تعریف کنیم و

درعین حال، این کلاسها خصوصیتهای مشترك فراوانی دارند ، پس اینطور بنظر میرسد که بهتر هست کلاسهای جدید اشکال از کلاس پایه ای و اصلیه Shape ارث ببرند تا بتوانیم زحمت پیاده سازی مجدد خصوصیتها و رفتارهای یکسانی که این کلاسها با کلاس اصلی دارند را از دوش خودمان برداریم .

مثلا خصوصیتی مانند "رنگ شکل" خصوصیتی هست که برای پیاده سازی آن در کلاسهای زیرین ، کار خاصی نباید انجام دهیم و به سادگی میتوان آنرا از کلاس Shape به ارث برد
از سوی دیگر متدی مثل Draw را داریم که بدیهی هست که چون ترسیم هرشکلی به شیوه خاصی انجام میشود ، پس هر کلاسی باید شیوه خاص خودش را برای ترسیم تعریف کنه ، یعنی این نیاز هست که کلاسهایی که از Shape ارث برده اند، بتوانند مطابق نیازهایشان متد پایه ای Draw ی موروثی را **تغییر** دهند و همانطور که در ابتدای این نوشته عرض کردم ، این **تغییر** را در ادبیات شی گرا ، **override** مینامند پس کلاسهای زیرین متد Draw ئه موروثی را override خواهند کرد .
علاوه بر آن فراموش نکنید که درصورتیکه قصد override کردن داریم ، بایستی متد مورد نظر در کلاس پایه را با کلمه virtual تعریف کنیم تا متوجه شود که وارثانش قرار هست که در آن دخل و تصرف کنند و آنرا بقولی override کنند.

در زیر ، کد نمونه برای دیاگرام فوق و نحوه استفاده از آنرا مشاهده میکنید.

```
namespace PolyMorphiZm
{
    public class Shape
    {
        public virtual void Draw()
        {
            Console.WriteLine("A Shape is drawn");
        }
    }
}
```

```
namespace PolyMorphiZm
{
    public class Hexagon : Shape
    {
        public override void Draw()
        {
            Console.WriteLine("A Hexagon is Drawn");
        }
    }
}
```

```
namespace PolyMorphiZm
{
    public class Circle : Shape
    {
        public override void Draw()
        {
            Console.WriteLine("A Circle is drawn!");
        }
    }
}
```

```
namespace PolyMorphiZm
{
    class Program
    {
        static void Main()
        {
            Shape[] myShapes = new Shape[3];
            myShapes[0] = new Circle();
            myShapes[1] = new Hexagon();
            myShapes[2] = new Shape();

            foreach (Shape sh in myShapes)
            {
                sh.Draw();
            }
        }
    }
}
```