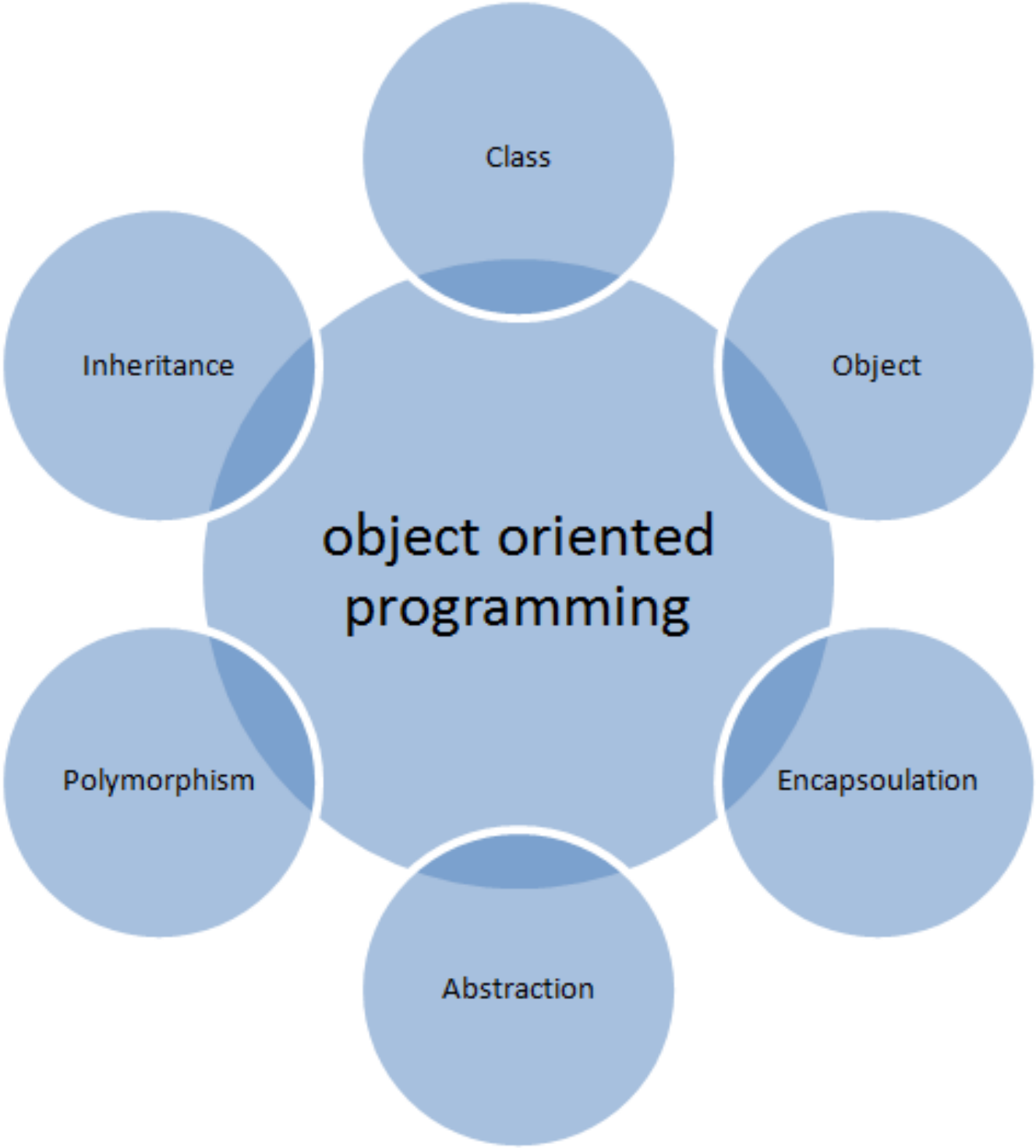
عنوان: **اصول طراحی شی گرا SOLID - #بخش اول اصل SRP**
نویسنده: ناصر طاهری
تاریخ: ۱۳۹۲/۰۷/۰۶ ۲۰:۳۰
آدرس: www.dotnettips.info
گروه‌ها: OOP, SOLID Principals, Object Oriented Design, Single Responsibility

مخاطب چه کسی است؟

این مقاله برای کسانی در نظر گرفته شده است که حداقل پیش زمینه ای در مورد برنامه نویسی شی گرا داشته باشند.کسانی که تفاوت بین کلاس‌ها و اشیاء را میدانند و میتوانند در مورد ارکان پایه ای برنامه نویسی شی گرایی نظیر : کپسوله سازی (Encapsulation) ، کلاس‌های انتزاعی (Abstraction) ، چند ریختی (Polymorphism) ، ارث بری (Inheritance) و... صحبت کنند.



مقدمه :

در جهان شی گرا ما فقط اشیاء را میبینیم که با یکدیگر در ارتباط هستند.کلاس ها، شی ها، ارث بری، کپسوله سازی ، کلاس‌های انتزاعی و ... کلماتی هستند که ما هر روز در حرفه‌ی خودمان بارها آنها را میشنویم.

در دنیای مدرن نرم افزار، بدون شک هر توسعه دهنده‌ی نرم افزار ،یکی از انواع زبان‌های شی گرا را برای استفاده انتخاب میکند. اما آیا او واقعا میداند که برنامه نویسی شی گرا به چه معنی است؟ آیا او واقعا از قدرت شی گرایی استفاده میکند؟

در این مقاله تصمیم گرفته ایم پای خود را فراتر از ارکان پایه ای برنامه نویسی شی گرا قرار دهیم و بیشتر در مورد طراحی شی گرا صحبت کنیم. **طراحی شی گرا :**

طراحی شی گرا یک فرایند از برنامه ریزی یک سیستم نرم افزاری است که در آن اشیاء برای حل مشکلات خاص با یکدیگر در ارتباط هستند. در حقیقت یک طراحی شی گرای مناسب ، کار توسعه دهنده را آسان میکند و یک طراحی نامناسب تبدیل به یک

فاجعه برای او میشود. **هر کسی چگونه شروع میکند؟**

وقتی کسانی شروع به ایجاد معماری نرم افزار میکنند، نشان میدهند که اهداف خوبی در سر دارند.آنها سعی میکنند از تجارب خود برای ساخت یک طراحی زیبا و تمیز استفاده کنند.

اما با گذشت زمان، نرم افزار کارایی خود را از دست میدهد و بلااستفاده میشود. با هر درخواست ایجاد ویژگی جدید در نرم افزار ، به تدریج نرم افزار شکل خود را از دست میدهد و درنهایت ساده‌ترین تغییرات در نرم افزار موجب تلاش و دقت زیاد، زمان طولانی و مهمتر از همه بالا رفتن تعداد باگها در نرم افزار میشود.

چه کسی مقصر است؟

"تغییرات" یک قسمت جدایی ناپذیر از جهان نرم افزار هستند بنابراین ما نمتوانیم "تغییر" را مقصر بدانیم و در حقیقت این طراحی ما است که مشکل دارد.

یکی از بزرگترین دلایل مخرب کننده‌ی نرم افزار، تعریف وابستگی‌های ناخواسته و بیخود در قسمت‌های مختلف سیستم است. در این گونه طراحی‌ها ، هر قسمت از سیستم وابسته به چندین قسمت دیگر است ، بنابراین تغییر یک قسمت، بر روی قسمت‌های دیگر نیز تاثیر میگذارد و باعث این چنین مشکلاتی میشود. ولی در صورتی که ما قادر به مدیریت این وابستگی باشیم در آینده خواهیم توانست از این سیستم نرم افزاری به آسانی نگهداری کنیم.

مثال:

راه حل : اصول ، الگوهای طراحی و معماری نرم افزار معماری نرم افزار به عنوان مثال MVC, MVP, 3-Tire به ما میگویند که پروژه‌ها از از چه ساختاری استفاده میکنند. **الگوهای طراحی** یک سری راه حل‌های قابل استفاده‌ی مجدد را برای مسائلی که به طور معمول اتفاق می‌افتند، فراهم میکند. یا به عبارتی دیگر الگوهای طراحی راه کارهایی را به ما معرفی میکنند که میتوانند برای حل مشکلات کد نویسی بارها مورد استفاده قرار بگیرند. **اصول به ما میگوید** اینها را انجام بده تا به آن دست پیدا کنی واینکه چطور انجامش میدهی به خودت بستگی دارد. هر کس یک سری اصول را در زندگی خود تعریف میکند مانند : "من هرگز دروغ نمیگویم" یا "من هرگز سیگار نمیکشم" و از این قبیل. او با دنبال کردن این اصول زندگی آسانی را برای خودش ایجاد میکند. به همین شکل، طراحی شی گرا هم مملو است از اصولی که به ما اجازه میدهد تا با طراحی مناسب مشکلاتمان را مدیریت کنیم.

آقای رابرت مارتین(Robert Martin) این موارد را به صورت زیر طبقه بندی کرده است :

1- اصول طراحی کلاس‌ها که SOLID نامیده می‌شوند.

2- اصول انسجام بسته بندی

3- اصول اتصال بسته بندی

در این مقاله ما در مورد اصول SOLID به همراه مثال‌های کاربردی صحبت خواهیم کرد.

SOLID مخففی از 5 اصول معرفی شده توسط آقای مارتین است:

```
S -> Single responsibility Principle
```

```
O-> Open Close Principle
```

```
L-> Liskov substitution principle
```

```
I -> Interface Segregation principle
```

```
D-> Dependency Inversion principle
```

اصل 1) S - SRP - Single responsibility Principle

به کد زیر توجه کنید :

```
public class Employee
{
    public string EmployeeName { get; set; }
    public int EmployeeNo { get; set; }

    public void Insert(Employee e)
    {
        //Database Logic written here
    }
    public void GenerateReport(Employee e)
    {
        //Set report formatting
    }
}
```

در کد بالا هر زمان تغییری در یک قسمت از کد ایجاد شود این احتمال وجود دارد که قسمت دیگری از آن مورد تاثیر این تغییر قرار بگیرد و به مشکل برخورد کنید. دلیل نیز مشخص است : هر دو در یک خانه‌ی مشابه و دارای یک والد یکسان هستند.

برای مثال با تغییر یک پراپرتی ممکن است متدهای هم خانه که از آن استفاده میکنند با مشکل مواجه شوند و باید این تغییرات را نیز در آنها انجام داد. در هر صورت خیلی مشکل است که همه چیز را کنترل کنیم. بنابراین تنها تغییر موجب دوبرابر شدن عملیات تست میشود و شاید بیشتر.

اصل **SRP** برای رفع این مشکل میگوید **"هر ماژول نرم افزاری میبایست تنها یک دلیل برای تغییر داشته باشد"** .

(منظور از ماژول نرم افزاری همان کلاس‌ها ، توابع و ... است و عبارت "دلیل برای تغییر" همان مسئولیت است.) به عبارتی هر شی باید یک مسئولیت بیشتر بر عهده نداشته باشد. هدف این قانون جدا سازی مسئولیت-های چسبیده به هم است. به عنوان مثال کلاسی که هم مسئول ذخیره سازی و هم مسئول ارتباط با واسط کاربر است، این اصل را نقض می-کند و باید به دو کلاس مجزا تقسیم شود.

برای رسیدن به این منظور میتوانیم مثال بالا را به صورت 3 کلاس مختلف ایجاد کنیم :

1- Employee : که حاوی خاصیت‌ها است.

2- EmployeeDB : عملیات دیتابیسی نظیر درج رکورد و واکشی رکوردها از دیتابیس را انجام میدهد.

3- EmployeeReport : وظایف مربوط به ایجاد گزارش‌ها را انجام میدهد.

کد حاصل :

```
public class Employee
{
    public string EmployeeName { get; set; }
    public int EmployeeNo { get; set; }
}

public class EmployeeDB
{
    public void Insert(Employee e)
    {
        //Database Logic written here
    }
    public Employee Select()
    {
        //Database Logic written here
    }
}

public class EmployeeReport
{
    public void GenerateReport(Employee e)
```

```
    {
        //Set report formatting
    }
}
```

این روش برای متدها نیز صدق میکند به طوری که هر متد باید مسئولیت واحدی داشته باشد.
برای مثال قطعه کد زیر اصل SRP را نقض میکند :

```
//Method with multiple responsibilities – violating SRP
public void Insert(Employee e)
{
    string StrConnectionString = "";
    SqlConnection objCon = new SqlConnection(StrConnectionString);
    SqlParameter[] SomeParameters=null;//Create Parameter array from values
    SqlCommand objCommand = new SqlCommand("InertQuery", objCon);
    objCommand.Parameters.AddRange(SomeParameters);
    ObjCommand.ExecuteNonQuery();
}
```

این متد وظایف مختلفی را انجام میدهد مانند اتصال به دیتابیس ، ایجاد پارامترها برای مقادیر، ایجاد کوئری و در نهایت اجرای آن بر روی دیتابیس.
اما با توجه به اصل SRP میتوان آن را به صورت زیر بازنویسی کرد :

```
//Method with single responsibility – follow SRP
public void Insert(Employee e)
{
    SqlConnection objCon = GetConnection();
    SqlParameter[] SomeParameters=GetParameters();
    SqlCommand ObjCommand = GetCommand(objCon,"InertQuery",SomeParameters);
    ObjCommand.ExecuteNonQuery();
}

private SqlCommand GetCommand(SqlConnection objCon, string InsertQuery, SqlParameter[] SomeParameters)
{
    SqlCommand objCommand = new SqlCommand(InsertQuery, objCon);
    objCommand.Parameters.AddRange(SomeParameters);
    return objCommand;
}

private SqlParameter[] GetParaeters()
{
    //Create Paramter array from values
}

private SqlConnection GetConnection()
{
    string StrConnectionString = "";
    return new SqlConnection(StrConnectionString);
}
```

نظرات خوانندگان

نویسنده: حسن دهیاری
تاریخ: ۱۳۹۲/۰۷/۲۳ ۲۰:۲۷

با سلام.ضمن تشکر.یک سوال داشتم.آیا با توجه به اصل SRP نباید کلاس EmployeeDB تفکیک شود.چون که دو کار را انجام میدهد.ممنون میشم توضیح دهید.

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۷/۲۳ ۲۳:۲۶

تنها دلیل تغییر کلی این کلاس در آینده، تغییر خاصیت‌های شیء کارمند است. بنابراین اصل تک مسئولیتی را نقض نمی‌کند. اگر این کلاس برای مثال دو Select داشت که یکی لیست کارمندان و دیگری لیست نقش‌های سیستم را بازگشت می‌داد، در این حالت تک مسئولیتی نقض می‌شد. ضمنا این نوع طراحی تحت عنوان الگوی مخزن یا لایه سرویس و امثال آن، یک طراحی پذیرفته شده و عمومی است. اگر قصد دارید که کوئری‌های خاص آن‌را طبقه بندی کنید می‌شود مثلا از Specification pattern استفاده کرد.

نویسنده: فراز
تاریخ: ۱۳۹۳/۰۵/۱۹ ۰:۴۹

با سلام ممنون از مقاله ای که گذاشتین من 3 سال هست تقریبا به صورت دست پا شکسته دنبال OOAD می‌گردم که در عمل توضیح داده باشد که شما این کار رو انجام دادین با سپاس فراوان