عنوان: **اصول و قراردادهای نام‌گذاری در دات‌نت**
نویسنده: یوسف نژاد
تاریخ: ۱۳۹۱/۰۵/۱۷ ۱۱:۵۶
آدرس: www.dotnettips.info
برچسب‌ها: .NET, Naming

نامگذاری (**Naming**) اشیا یک برنامه شاید در نگاه اول دارای اهمیت بالایی نباشه، اما تجربه نشون داده که در پروژه‌های بزرگ که با کمک چندین مجموعه به انجام میرسه نامگذاری صحیح و اصولی که از یکسری قواعد کلی و مناسب پیروی میکنه میتونه به پیشبرد اهداف و مدیریت راحتتر برنامه کمک بسیاری بکنه.
بیشتر موارد اشاره شده در این مطلب از کتاب جامع و مفید Framework Design Guidelines اقتباس شده که خوندن این کتاب مفید رو به خوانندگان توصیه میکنم.

برای کمک به نوشتن اصولی و راحتتر سورسهای برنامه‌ها در ویژوال استودیو نرم افزارهای متعددی وجود داره که با توجه به تجربه شخصی خودم نرم افزار Resharper محصول شرکت Jetbrains یکی از بهترین هاست که در مورد خاص مورد بحث در این مطلب نیز بسیار خوب عمل میکنه.
برخی از موارد موجود در مطلب جاری نیز از قراردادهای پیشفرض موجود در نرم افزار Resharper نسخه 6.0 برگرفته شده است و قسمتی نیز از تجربه شخصی خودم و سایر دوستان و همکاران بوده است.

اصل این مطلب حدود یکسال پیش تهیه شده و اگر نقایصی وجود داره لطفا اشاره کنین.

اصول و قراردادهای نام‌گذاری در دات‌نت

انواع نام‌گذاری

نام‌گذاری اشیا در حالت کلی را می‌توان به سه روش زیر انجام داد:
1. **Pascal Casing** : در این روش حرف اول هر کلمه در نام شی به صورت بزرگ نوشته می‌شود.

```
FirstName
```

2. **camel Casing** : حرف اول در اولین کلمه نام هر شی به صورت کوچک و حرف اول بقیه کلمات به صورت بزرگ نوشته می‌شود.

```
firstName
```

3. **Hungarian** : در این روش برای هر نوع شی موجود یک پیشوند درنظر گرفته می‌شود تا از روی نام شی بتوان به نوع آن پی برد. در ادامه و پس از این پیشوندها سایر کلمات بر اساس روش Pascal Casing نوشته می‌شوند.

```
strFirstName
lblFirstName
```

نکته: استفاده از این روش به جز در نام‌گذاری کنترل‌های UI منسوخ شده است.

قراردادهای کلی

1. نباید نام اشیا تنها در بزرگ یا کوچک بودن حروف با هم فرق داشته باشند. به عنوان مثال نباید دو کلاس با نام‌های MyClass و myClass داشته باشیم. هرچند برخی از زبان‌ها case-sensitive هستند اما برخی دیگر نیز چنین قابلیتی ندارند (مثل VB.NET).
بنابراین اگر بخواهیم کلاس‌های تولیدی ما در تمام محیطها و زبان‌های برنامه نویسی قابل اجرا باشند باید از این قرارداد پیروی کنیم.
2. تا آنجا که امکان دارد باید از به‌کار بردن مخفف کلمات در نام‌گذاری اشیا دوری کنیم. مثلا به جای استفاده از GetChr باید از

GetCharacter استفاده کرد.
البته در برخی موارد که مخفف واژه موردنظر کاربرد گسترده ای دارد می‌توان از عبارت مخفف نیز استفاده کرد. مثل UI به جای UserInterface و یا IO به جای InputOutput.

آ. اصول نام‌گذاری فضای نام (namespace)

1. اساس نام‌گذاری فضای نام باید از قاعده زیر پیروی کند:

```
<Company>.<Technology|Produt|Project>[.<Feature>][.<SubNamespace>]
```

(< > : اجباری [ ] : اختیاری)

2. برای نام‌گذاری فضای نام باید از روش Pascal Casing استفاده شود.
3. در هنگام تعریف فضاهای نام به وابستگی آنها توجه داشته باشید. به عنوان مثال اشیای درون یک فضای نام پدر نباید به اشیای درون فضای نام یکی از فرزندانش وابسته باشد. مثلا در فضای نام System نباید اشیایی وجود داشته باشند که به اشیای درون فضای نام System.UI وابسته باشند.
4. سعی کنید از نام‌ها به صورت جمع برای عناوین فضای نام استفاده کنید. مثلا به جای استفاده از Kara.CSS.HQ.Manager.Entity از Kara.CSS.HQ.Manager.Entities استفاده کنید. البته برای مواردی که از عناوین مخفف و یا برندهای خاص استفاده کرده‌اید از این قرارداد پیروی نکنید. مثلا نباید از عنوانی شبیه به Kara.CSS.Manager.IOs استفاده کنید.
5. از عنوانی پایدار و مستقل از نسخه محصول برای بخش دوم عنوان فضای نام استفاده کنید. بدین معنی که این عناوین با گذر زمان و تغییر و تحولات در محتوای محصول و یا تولیدکننده نباید تغییر کنند. مثال:

```
Microsoft.Reporting.WebForms
Kara.Support.Manager.Enums
Kara.CSS.HQ.WebUI.Configuration
```

6. از عناوین یکسان برای فضای نام و اشیای درون آن استفاده نکنید. مثلا نباید کلاسی با عنوان Manager در فضای نام Kara.CSS.Manager وجود داشته باشد.
7. از عناوین یکسان برای اشیای درون فضاهای نام یک برنامه استفاده نکنید. مثلا نباید دو کلاس با نام Package در فضاهای نام Kara.CSS.Manger و Kara.CSS.Manger.Entities داشته باشید.

ب. اصول نام‌گذاری کلاس‌ها و Structها

1. عنوان کلاس باید اسم یا موصوف باشد.
2. در نام‌گذاری کلاس‌ها باید از روش Pascal Casing استفاده شود.
3. نباید از عناوین مخففی که رایج نیستند استفاده کرد.
4. از پیشوندهای زائد مثل C یا Cls نباید استفاده شود.
5. نباید از کاراکترهایی به غیر از حروف (و یا در برخی موارد خیلی خاص، شماره نسخه) در نام‌گذاری کلاس‌ها استفاده شود.

مثال:
درست:

```
PackageManager , PacakgeConfigGenerator
Circle , Utility , Package
```

نادرست:

```
CreateConfig , classdata
CManager , ClsPackage , Config_Creator , Config1389
```

6. در نام‌گذاری اشیای Generic از استفاده از عناوینی چون Element, Node, Log و یا Message پرهیز کنید. این کار موجب به‌وجودآمدن کانفلیکت در عناوین اشیا می‌شود. در این موارد بهتر است از عناوینی چون FormElement, XmlNode, EventLog و

SoapMessage استفاده شود. درواقع بهتر است نام اشیای جنریک، برای موارد موردنیاز، کاملا اختصاصی و درعین حال مشخص‌کننده محتوا و کاربرد باشند.

7. از عناوینی مشابه عناوین کتابخانه‌های پایه دات‌نت برای اشیای خود استفاده نکنید. مثلا نباید کلاسی با عنوان Console, Parameter, Action و یا Data را توسعه دهید. هم‌چنین از کابرد عناوینی مشابه اشیای موجود در فضاهای نام غیرپایه دات‌نت و یا غیردات‌نتی اما مورداستفاده در پروژه خود که محتوا و کاربردی مختص همان فضای نام دارند پرهیز کنید. مثلا نباید کلاسی با عنوان Page در صورت استفاده از فضای نام System.Web.UI تولید کنید.

8. سعی کنید در مواردی که مناسب به‌نظر می‌رسد از عنوان کلاس پایه در انتهای نام کلاس‌های مشتق‌شده استفاده کنید. مثل FileStream که از کلاس Stream مشتق شده است. البته این قاعده در تمام موارد نتیجه مطلوب ندارد. مثلا کلاس Button که از کلاس Control مشتق شده است. بنابراین در به‌کاربردن این مورد بهتر است تمام جوانب و قواعد را درنظر بگیرید.

پ. اصول نام‌گذاری مجموعه‌ها (Collections)

یک مجموعه در واقع یک نوع کلاس خاص است که حاوی مجموعه‌ای از داده‌هاست و از همان قوانین کلاس‌ها برای نام‌گذاری آن‌ها استفاده می‌شود.

1. بهتر است در انتهای عنوان مجموعه از کلمه Collection استفاده شود. مثال:

```
CenterCollection , PackageCollection
```

2. البته درصورتی‌که کلاس مورد نظر ما رابط IDictionary را پیاده‌سازی کرده باشد بهتر است از پسوند Dictionary استفاده شود.

ت. اصول نام‌گذاری Delegateها

1. عنوان یک delegate باید اسم یا موصوف باشد.
2. در نام‌گذاری delegateها باید از روش Pascal Casing استفاده شود.
3. نباید از عناوین مخففی که رایج نیستند استفاده کرد.
4. از پیشوندهای زائد مثل D یا del نباید استفاده شود.
5. نباید از کاراکترهایی به غیر از حروف در نام‌گذاری delegateها استفاده شود.
6. **نباید** در انتهای نام یک delegate از عبارت Delegate استفاده شود.
7. بهتر است که درصورت امکان در انتهای نام یک delegate که برای Event Handler استفاده نمی‌شود از عبارت Callback استفاده شود. البته تنها درصورتی‌که معنی و مفهوم مناسب را داشته باشد.

مثال:

نحوه تعریف یک delegate

```
public delegate void Logger (string log);
public delegate void LoggingCallback (object sender, string reason);
```

ث. اصول نام‌گذاری رویدادها (Events)

1. عنوان یک رویداد باید فعل یا مصدر باشد.
2. در نام‌گذاری کلاس‌ها باید از روش Pascal Casing استفاده شود.
3. نباید از عناوین مخففی که رایج نیستند استفاده کرد.
4. از پیشوندهای زائد نباید استفاده شود.
5. نباید از کاراکترهایی به غیر از حروف در نام‌گذاری رویدادها استفاده شود.
6. بهتر است از پسوند EventHandler در عنوان هندلر رویداد استفاده شود.
7. از به کاربردن عباراتی چون AfterXXX و یا BeforeXXX برای نمایش رویدادهای قبل و یا بعد از رخداد خاصی خودداری شود. به جای آن باید از اسم مصدر (شکل ingدار فعل) برای نام‌گذاری رویداد قبل از رخداد و هم‌چنین شکل گذشته فعل برای نام‌گذاری رویداد بعد از رخداد خاص استفاده کرد.

مثلا اگر کلاسی دارای رویداد Open باشد باید از عنوان Openning برای رویداد قبل از Open و از عنوان Opened برای رویداد بعد از

Open استفاده کرد.
8. نباید از پیشوند On برای نام‌گذاری رویداد استفاده شود.
9. همیشه پارامتر e و sender را برای آرگومانهای رویداد پیاده سازی کنید. sender که از نوع object است نمایش دهنده شیی است که رویداد مربوطه را به وجود آورده است و e درواقع آرگومانهای رویداد مربوطه است.
10. عنوان کلاس‌آرگومان‌های رویداد باید دارای پسوند EventArgs باشد.
مثال:
عنوان کلاس‌آرگومان:

```
AddEventArgs , EditEventArgs , DeleteEventArgs
```

عنوان رویداد:

```
Adding , Add , Added
```

تعریف یک EventHandler:

```
public delegate void <EventName>EventHandler
```

```
(object sender, <EventName>EventArgs e);
```

نکته: نیاز به تولید یک EventHandler مختص توسعه یک برنامه به‌ندرت ایجاد می‌شود. در اکثر موارد می‌توان با استفاده از کلاس جنریک EventHandler<TEventArgs> تمام احتیاجات خود را برطرف کرد.
تعریف یک رویداد:

```
public event EventHandler
```

```
<AddEventArgs> Adding;
```

ج. اصول نام‌گذاری Attributeها
1. عنوان یک attribute باید اسم یا موصوف باشد.
2. در نام‌گذاری attributeها باید از روش Pascal Casing استفاده شود.
3. نباید از عناوین مخففی که رایج نیستند استفاده کرد.
4. از پیشوندهای زائد مثل A یا atr نباید استفاده شود.
5. نباید از کاراکترهایی به غیر از حروف در نام‌گذاری attributeها استفاده شود.
6. بهتر است در انتهای نام یک attribute از عبارت Attribute استفاده شود.
مثال:

```
DisplayNameAttribute , MessageTypeAttribute
```

ج. اصول نام‌گذاری Interfaceها
1. در ابتدای عنوان interface باید از حرف I استفاده شود.
2. نام باید اسم، موصوف یا صفتی باشد که interface را توصیف می‌کند.
به عنوان مثال:

```
IComponent  (اسم)
IConnectionProvider (موصوف)
ICloneable (صفت)
```

3. از روش Pascal Casing استفاده شود.
4. خودداری از بکاربردن عبارات مخفف غیررایج.
5. باید تنها از کاراکترهای حرفی در نام interface استفاده شود.

ح. اصول نام‌گذاری Enumerationها

1. استفاده از روش Pascal Casing
2. خودداری از کاربرد عبارات مخفف غیررایج
3. تنها از کاراکترهای حرفی در نام Enumretionها استفاده شود.
4. نباید از پسوند یا پیشوند Enum یا Flag استفاده شود.
5. اعضای یک Enum نیز باید با روش Pascal Casing نام‌گذاری شوند.

مثال:

```
public enum FileMode {
    Append,
    Read, …
}
```

6. درصورتی‌که enum موردنظر از نوع flag نیست باید عنوان آن مفرد باشد.
7. درصورتی‌که enum موردنظر برای کاربرد flag طراحی شده باشد نام آن باید جمع باشد. مثال:

```
[Flag]
public enum KeyModifiers {
    Alt = 1,
    Control = 2,
    Shift = 4
}
```

8. از به‌کاربردن پسوند و یا پیشوندهای اضافه در نام‌گذاری اعضای یک enum نیز پرهیز کنید. مثلا نام‌گذاری زیر نادرست است:

```
public enum OperationState {
    DoneState,
    FaultState,
    RollbackState
}
```

خ. اصول نام‌گذاری متدها

1. نام متد باید فعل یا ترکیبی از فعل و اسم یا موصوف باشد.
2. باید از روش Pascal Casing استفاده شود.
3. خودداری از بکاربردن عبارات مخفف غیررایج و یا استفاده زیاد از اختصار
4. تنها از کاراکترهای حرفی برای نام متد استفاده شود.

مثال:

```
AddDays , Save , DeleteRow , BindData , Close , Open
```

د. اصول نام‌گذاری Propertyها

1. نام باید اسم، صفت یا موصوف باشد.
2. باید از روش Pascal Casing استفاده شود.
3. خودداری از بکاربردن عبارات مخفف غیررایج.
4. تنها از کاراکترهای حرفی برای نام‌گذاری پراپرتی استفاده شود.

مثال:

```
Radius , ReportType , DataSource , Mode , CurrentCenterId
```

5. از عبارت Get در ابتدای هیچ Propertyای استفاده نکنید.
6. نام خاصیت‌هایی که یک مجموعه برمی‌گرداند باید به صورت جمع باشد. عنوان این Propertyها نباید به‌صورت مفرد به همراه پسوند Collection یا List باشد. مثال:

```
public CenterCollection Centers { get; set; }
```

7. خواص Boolean را با عناوینی مثبت پیاده‌سازی کنید. مثلا به‌جای استفاده از CantRead از CanRead استفاده کنید. بهتر است این Propertyها پیشوندهایی چون Is, Can یا Has داشته باشند، البته تنها درصورتی‌که استفاده از چنین پیشوندهایی ارزش افزوده داشته و مفهوم آن را بهتر برساند. مثلا عنوان CanSeek مفهوم روشن‌تری نسبت به Seekable دارد. اما استفاده از Created خیلی بهتر از IsCreated است یا Enabled کاربرد به مراتب راحت‌تری از IsEnabled دارد.
برای تشخیص بهتر این موارد بهتر است از روش ifسنجی استفاده شود. به عنوان مثال

```
if (list.Contains(item))
if (regularExpression.Matches(text))
if (stream.CanSeek)
if (context.Created)
if (form.Enabled)
```

مفهوم درست‌تری نسبت به موارد زیر دارند:

```
if (list.IsContains(item))
if (regularExpression.Match(text))
if (stream.Seekable)
if (context.IsCreated)
if (form.IsEnabled)
```

8. بهتر است در موارد مناسب عنوان Property با نام نوعش برابر باشد. مثلا

```
public Color Color { get; set; }
```

ذ. اصول نام‌گذاری پارامترها

پارامتر درحالت کلی به آرگومان وروی تعریف شده برای یک متد گفته می‌شود.
1. حتما از یک نام توصیفی استفاده شود. از نام‌گذاری پارامترها براساس نوعشان به‌شدت پرهیز کنید.
2. از روش camel Casing استفاده شود.
3. تنها از کاراکترهای حرفی برای نام‌گذاری پارامترها استفاده شود.
مثال:

```
firstName , e , id , packageId , centerName , name
```

4. نکاتی برای نام‌گذاری پارامترهای **Operator Oveloading** :
- برای operatorهای دو پارامتری (binary operators) از عناوین left و right برای پارامترهای آن استفاده کنید:

```
public static MyType operator +(MyType left, MyType right)
public static bool operator ==(MyType left, MyType right)
```

- برای operatorهای تک‌پارامتری (unary operators) اگر برای پارامتر مورد استفاده هیچ عنوان توصیفی مناسبی پیدا نکردید حتما از عبارت value استفاده کنید:

```
public static MyType operator ++(MyType value)
```

- درصورتی‌که استفاده از عناوین توصیفی دارای ارزش افزوده بوده و خوانایی کد را بهتر می‌کند حتما از این نوع عناوین استفاده کنید:

```
public static MyType operator /(MyType dividend, MyType divisor)
```

- نباید از عبارات مخفف یا عناوینی با اندیس‌های عددی استفاده کنید:

```
public static MyType operator -(MyType d1, MyType d2) // incorrect!
```

ر. اصول نام‌گذاری متغیر (Variable)ها

- نام متغیر باید اسم، صفت یا موصوف باشد.

نام‌گذاری متغیرها باید با توجه به نوع آن انجام شود.

1. متغیرهای عمومی (public) و protected و Constantها

- استفاده از روش Pascal Casing

- تنها از کاراکترهای حرفی برای نام متغیر عمومی استفاده شود.

مثال:

```
Area , DataBinder , PublicCacheName
```

2. متغیرهای private (در سطح کلاس یا همان field)

- نام این نوع متغیر باید با یک "_" شروع شود.

- از روش camel Casing استفاده شود.

مثال:

```
_centersList
_firstName
_currentCenter
```

3. متغیرهای محلی در سطح متد

- باید از روش camel Casing استفاده شود.

- تنها از کاراکترهای حرفی استفاده شود.

مثال:

```
parameterType , packageOperationTypeId
```

ز. اصول نام‌گذاری کنترل‌های UI

1. نام باید اسم یا موصوف باشد.

2. استفاده از روش Hungarian !

3. عبارت مخفف معرفی‌کننده کنترل، باید به اندازه کافی برای تشخیص نوع آن مناسب باشد.

مثال:

```
lblName (Label)
txtHeader (TextBox)
btnSave (Button)
```

ژ. اصول نام‌گذاری Exceptionها

تمام موارد مربوط به نام‌گذاری کلاس‌ها باید در این مورد رعایت شود.

1. باید از پسوند Exception در انتهای عنوان استفاده شود.

مثال:

```
ArgumentNullException , InvalidOperaionException
```

س. نام‌گذاری اسمبلی‌ها و DLLها

1. عناوینی که برای نام‌گذاری اسمبلی‌ها استفاده می‌شوند، باید نمایش‌دهنده محتوای کلی آن باشند. مثل:

```
System.Data
```

2. از روش زیر برای نام‌گذاری اسمبلی‌ها استفاده شود:

```
<Company>.<Component>.dll
<Company>.<Project|Product|Technology>.<Component>.dll
```

مثل:

```
Microsoft.CSharp.dll , Kara.CSS.Manager.dll
```

ش. نام‌گذاری پارامترهای نوع Generic (type parameter)

1. از حرف T برای پارامترهای تک‌حرفی استفاده کنید. مثل:

```
public int IComparer<T> {…}
public delegate bool Predicate<T> (T item)
```

2. تمامی پارامترهای جنریک را با عناوینی توصیفی و مناسب که مفهوم و کاربرد آنرا برساند نام‌گذاری کنید، مگر آنکه یافتن چنین عباراتی ارزش افزوده‌ای در روشن‌تر کردن کد نداشته باشد. مثال:

```
public int ISessionChannel<TSession> {…}
public delegate TOutput Converter<TInput, TOutput> (TInput from)
public class Nullable<T> {…}
public class List<T> {…}
```

3. در ابتدای عناوین توصیفی حتما از حرف T استفاده کنید.

4. بهتر است تا به‌صورتی روشن نوع قید قرار داده شده بر روی پارامتری خاص را در نام آن پارامتر نمایش دهید. مثلا اگر قید ISession را برای پارامتری قرار دادید بهتر است نام آن پارامتر را TSession درنظر بگیرید.

ص. نام‌گذاری کلید Resourceها

به دلیل شباهت ساختاری که میان کلیدهای resource و propertyها وجود دارد قواعد نام‌گذاری propertyها در اینجا نیز معتبر هستند.

1. از روش نام‌گذاری Pascal Casing برای کلیدهای resource استفاده کنید.

2. از عناوین توصیفی برای این کلیدها استفاده کنید. سعی کنید تا حد امکان به هیچ وجه از عناوین کوتاه و یا مخففی که مفهوم را به‌صورت ناکامل می‌رساند استفاده نکنید. درواقع سعی کنید که خوانایی بیشتر کد را فدای فضای بیشتر نکنید.

3. از کلیدواژه‌های CLR و یا زبان مورداستفاده برای برنامه‌نویسی در نام‌گذاری این کلیدها استفاده نکنید.

4. تنها از حروف و اعداد و _ در نام‌گذاری این کلیدها استفاده کنید.
5. سعی کنید از عناوین توصیفی همانند زیر برای پیام‌های مناسب خطاها جهت نمایش به کاربر برای کلیدهای مربوطه استفاده کنید. درواقع نام کلید باید ترکیبی از نام نوع خطا و یک آی‌دی مشخص‌کننده پیغام مربوطه باشد:

```
ArgumentExceptionIllegalCharacters
ArgumentExceptionInvalidName
ArgumentExceptionFileNotFound
```

نکاتی درمورد کلمات مرکب

کلمات مرکب به کلماتی گفته می‌شود که در آن از بیش از یک کلمه با مفهوم مستقل استفاده شده باشد. مثل Callback یا FileName.

باید توجه داشت که با تمام کلمات موجود در یک کلمه مرکب نباید همانند یک کلمه مستقل رفتار کرد و حرف اول آن را در روش‌های نام‌گذاری موجود به‌صورت بزرگ نوشت. کلمات مرکبی وجود دارند که به آن‌ها closed-form گفته می‌شود. این کلمات مرکب با اینکه از 2 یا چند کلمه دارای مفهوم مستقل تشکیل شده‌اند اما به‌خودی‌خود دارای مفهوم جداگانه و مستقلی هستند. برای تشخیص این کلمات می‌توان به فرهنگ لغت مراجعه کرد (و یا به‌سادگی از نرم‌افزار Microsoft Word استفاده کرد) و دریافت که آیا کلمه مرکب موردنظر آیا مفهوم مستقلی برای خود دارد، یعنی درواقع آیا عبارتی closed-form است. با کلمات مرکب از نوع closed-form همانند یک کلمه ساده برخورد می‌شود!

در جدول زیر مثال‌هایی از عبارات رایج و نحوه درست و نادرست استفاده از هریک نشان داده شده است.

Wrong	camel Casing	Pascal Casing
CallBack	callback	Callback
Bitflag / bitflag	bitFlag	BitFlag
Cancelled	canceled	Canceled
Donot / Don’t	doNot	DoNot
EMail	email	Email
EndPoint / endPoint	endpoint	Endpoint
Filename / filename	fileName	FileName
GridLine / gridLine	gridline	Gridline
HashTable / hashTable	hashtable	Hashtable
ID	id	Id
Indices	indexes	Indexes

Wrong	camel Casing	Pascal Casing
Logoff / LogOut !	logOff	LogOff
Logon / LogIn !	logOn	LogOn
Signout / SignOff	signOut	SignOut
Signin / SignOn	signIn	SignIn
MetaData / metaData	metadata	Metadata
MultiPanel / multiPanel	multipanel	Multipanel
MultiView / multiView	multiview	Multiview
NameSpace / nameSpace	namespace	Namespace
OK	ok	Ok
PI	pi	Pi
PlaceHolder / placeHolder	placeholder	Placeholder
Username / username	username	UserName
Whitespace / whitespace	whiteSpace	WhiteSpace
Writeable / writeable	writable	Writable

همان‌طور که در جدول بالا مشاهد می‌شود در استفاده از قوانین عبارات مخفف دو مورد استثنا وجود دارد که عبارتند از Id و Ok. این کلمات باید همان‌طور که در اینجا نشان داده شده‌اند استفاده شوند.

نکاتی درباره عبارات مخفف

1. عبارات مخفف (Acronym) با خلاصه‌سازی کلمات (Abbreviation) فرق دارند. یک عبارت مخفف شامل حروف اول یک عبارت طولانی یا معروف است، درصورتی‌که خلاصه‌سازی یک عبارت یا کلمه از حذف بخشی از آن به‌دست می‌آید. تا آنجاکه امکان دارد از خلاصه‌سازی عبارات نباید استفاده شود. هم‌چنین استفاده از عبارات مخفف غیررایج توصیه نمی‌شود.

مثال: IO و UI

2. براساس تعریف، یک مخفف حداقل باید 2 حرف داشته باشد. نحوه برخورد با مخفف‌های دارای بیشتر از 2 حرف با مخفف‌های دارای 2 حرف باهم متفاوت است. با مخفف‌های دارای 2 حرف همانند کلمه‌ای یک حرفی! برخورد می‌شود، درصورتی‌که با سایر مخفف‌ها همانند یک کلمه کامل چندحرفی برخورد می‌شود. به‌عنوان مثال IOStream برای روش PascalCasing و ioStream برای

روش camelCasing استفاده می‌شود. هم‌چنین از HtmlBody و htmlBody به‌ترتیب برای روش‌های Pascal و camel استفاده می‌شود.

3. هیچ‌کدام از حروف یک عبارت مخفف در ابتدای یک واژه نام‌گذاری‌شده به روش camelCasing به‌صورت بزرگ نوشته نمی‌شود!

نکته : موارد زیادی را می‌توان یافت که در ابتدا به‌نظر می‌رسد برای پیاده‌سازی آنها باید قوانین فوق را نقض کرد. این موارد شامل استفاده از کتابخانه‌های سایر پلتفرم‌ها (مثل MFC, HTML و غیره)، جلوگیری از مشکلات جغرافیایی! (مثلا در مورد نام کشورها یا سایر موقعیت‌های جغرافیایی)، احترام به نام افراد درگذشته، و مواردی از این دست می‌شود. اما در بیشتر قریب به اتفاق این موارد هم می‌توان بدون نقض قوانین فوق اقدام به نام‌گذاری اشیا کرد، بدون اینکه با تغییر عبارت اصلی (که موجب تطابق با این قوانین می‌شود) لطمه‌ای به مفهوم آن بزند. البته تنها موردی که به‌نظر می‌رسد می‌تواند قوانین فوق را نقض کند نام‌های تجاری هستند. البته استفاده از نام‌های تجاری توصیه نمی‌شود، چون این نام‌ها سریع‌تر از محتوای کتابخانه‌های برنامه‌نویسان تغییر می‌کنند!

نکته: برای درک بهتر قانون "عدم استفاده از عبارات مخففی که رایج نیستند" مثالی از زبان توسعه دهندگان دات‌نت‌فریمورک ذکر می‌شود. در کلاس Color متد زیر با Overloadهای مختلف در دسترس است:

```
public class Color {
    …
    public static Color FromArgb(…)
    { … }
}
```

همان‌طور که مشاهده می‌شود برای رعایت قوانین فوق به‌جای استفاده از ARGB از عبارت Argb استفاده شده است. اما این نحوه استفاده موجب شده تا این سوال به‌ظاهر خنده‌دار اما درست پیش‌آید:

"چطور می‌شود در این کلاس رنگی را از ARGB تبدل کرد؟ هرچه که من میبینم فقط تبدیل از طریق (Argb) آرگومان b است!"

حال در این نقطه به‌نظر می‌رسد که در اینجا باید قوانین فوق را نقض کرد و از عنوان FromARGB که مفهوم درست را می‌رساند استفاده کرد. اما با کمی دقت متوجه می‌شویم که این قوانین در ابتدا نیز با پیاده‌سازی نشان داده شده در قطعه کد بالا نقض شده‌اند! همه می‌دانیم که عبارت RGB مخفف معروفی برای عبارت Red Green Blue است. اما استفاده از ARGB برای افزودن کلمه Alpha به ابتدای عبارت مذکور چندان رایج نیست. پس استفاده از مخفف Argb از همان ابتدا اشتباه به‌نظر می‌رسد. بنابراین راه‌حل بهتر می‌تواند استفاده از عنوان FromAlphaRgb باشد که هم قوانین فوق را نقض نکرده و هم مفهوم را بهتر می‌رساند.

دیگر نکات

1. قراردادهای اشاره شده در این سند حاصل کار شبانه روزی تعداد بسیاری از برنامه نویسان در سرتاسر جهان در پروژه‌های بزرگ بوده است. این اصول کلی تنها برای توسعه آسان‌تر و سریع‌تر پروژه‌های بزرگ تعیین شده‌اند و همان‌طور که روشن است تنها ازطریق تجربه دست‌یافتنی هستند. بنابراین چه بهتر است که در این راه از تجارب بزرگان این عرصه بیشترین بهره برده شود.

2. هم‌چنین توجه داشته باشید که بیشتر قراردادهای اشاره شده در این سند از راهنمای نام‌گذاری تیم توسعه BCL دات‌نت‌فریمورک گرفته شده است. این تیم طبق اعتراف خودشان زمان بسیار زیادی را برای نام‌گذاری اشیا صرف کرده‌اند و توصیه کرده‌اند که دیگران نیز برای نام‌گذاری، زمان مناسب و کافی را در توسعه پروژه‌ها درنظر بگیرند.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۵/۱۸ ۱۱:۳۶

ضمن تشکر از مطلب مفید شما، علاوه بر ReSharper که می‌تونه در دراز مدت اثر ذهنی قابل ملاحظه‌ای در تطابق با اصول نامگذاری داشته باشه، نرم افزار FxCop هم یک سری از مواردی را که ReSharper تشخیص نمی‌ده می‌تونه به خوبی گزارش بده:

```
CA1717:OnlyFlagsEnumsShouldHavePluralNames
CA1704:IdentifiersShouldBeSpelledCorrectly
CA1709:IdentifiersShouldBeCasedCorrectly
CA1702:CompoundWordsShouldBeCasedCorrectly
...
```

نویسنده: حسین مرادی نیا
تاریخ: ۱۳۹۱/۰۵/۱۹ ۹:۴۷

سلام
مرسی بابت مطلب خوبتون
اما یه سوال
Resharper معمولا توصیه میکنه که برای متغییرهای محلی از this استفاده نشه.حالا میخواستم نظرتون رو در این زمینه بدونم.
استفاده از this خوبه یا بد؟

نویسنده: یوسف نژاد
تاریخ: ۱۳۹۱/۰۵/۱۹ ۲۱:۲۷

به نظر من استفاده بی مورد و اضافی از this اشتباهه. فقط در موارد لازم برای از بین بردن کانفلیکت (مثلا بین نامهای فیلدها و پارامترها) باید استفاده بشه. هرچند اگه اصول و قراردادها رعایت بشه معمولا این تضادها و کانفلیکتها پیش نمیاد.

نویسنده: حسین
تاریخ: ۱۳۹۱/۰۷/۲۴ ۱۲:۱

خیلی ممنون بابت مطلب مفیدتون. خواهشاً یک مطلب هم درباره نام گذاری *پروژه ها* قرار بدید (در مواردی که یک پروژه به چند Class Library و ... تقسیم میشود).
ممنون.

نویسنده: محمد علی
تاریخ: ۱۳۹۱/۱۱/۲۸ ۱۱:۳۳

عالی بود.ممنون

نویسنده: M.Q
تاریخ: ۱۳۹۲/۰۲/۰۵ ۲۰:۴۷

با سلام و تشکر

اگه یک متد داشته باشیم که این متد پارامتری با نام مثلا (id) داشته باشد و ما بخواهیم به هر دلیلی متغیری محلی برای نگهداری "کد" در بدنه متد نیز داشته باشیم، نام گذاری متغیر محلی چگونه باید باشد؟

```
public void Save(int id, string name)
        {
            int id_ = id; // ?
```

```
        /*
        ادامه دستورات
        */
    }
```

با تشکر

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۰۵ ۲۱:۳۹

روش خاصی نداره. فقط همان اصول کلی نامگذاری متغیرها در اینجا نیز باید رعایت شود. مثلا `localId` خوبه.

عنوان: **روش نامگذاری Smurf ایی!**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۱ ۱۱:۱
آدرس: www.dotnettips.info
برچسب‌ها: C#, Design patterns, Naming

اگر به یک سری از کتابخانه‌ها دقت کنید، تمام کلاس‌های آن‌ها دارای یک پیشوند تکراری هستند؛ مثلا **Smurf** XMLDataRow، **Smurf** XMLElement و الی آخر در مورد تمام کلاس‌های موجود در پروژه. به این رویه « Smurf Naming Convention » گفته می‌شود! در این نوع کتابخانه‌ها زمانیکه کاربری بر روی دکمه‌ای کلیک می‌کند، **Smurf** AccountView اطلاعات **Smurf** AccountDTO را به **Smurf** AccountController منتقل می‌کند. در ادامه از خاصیت **Smurf** ID دریافتی، مقدار **Smurf** OrderHistory دریافت شده و به **Smurf** HistoryReportingView جهت نمایش ارسال خواهد شد. اگر استثنای **Smurf** ErrorEvent رخ دهد، توسط **Smurf** ErrorLogger در فایلی به نام **smurf** / **smurf** log.log /log ثبت خواهد شد.

کلمه Smurf هم از شخصیتی کارتونی به همین نام اخذ شده است که در زبان مخصوص آن‌ها اکثر افعال و نام‌ها از کلمه Smurf مشتق می‌شود! برای مثال در مورد ماهیگیری کردن در یک رودخانه عنوان می‌کنند «We're going **smurfing** on the River **Smurf**today».

خوب، چکار باید کرد؟ روش صحیح معرفی نام یک شرکت در حین طراحی و نامگذاری کلاس‌های یک کتابخانه چیست؟
در مطلب بسیار جامع و عالی « اصول و قراردادهای نام‌گذاری در دات‌نت » عنوان شده است که اساس نام‌گذاری فضاهای نام باید از قاعده زیر پیروی کند:

```
<Company>.<Technology|Produt|Project>[.<Feature>][.<SubNamespace>]
```

مثلا مایکروسافت یکبار فضای نام Microsoft.Reporting.WebForms را تعریف کرده است و ... همین! دیگر به ابتدای هر کلاسی در این کتابخانه، پیشوند Microsoft یا MS و امثال آن اضافه نشده است تا بر روی اعصاب و روان استفاده کننده تاثیر منفی داشته باشد.

نظرات خوانندگان

نویسنده: رحمت اله رضایی
تاریخ: ۱۳۹۱/۰۶/۱۱ ۱۸:۳۸

مشکل اینجاست که در پروژه ای، کلاسهایی هم اسم کلاسهای دات نت داشته باشیم. همیشه باید فضای نام را در ابتدای کلاسها نوشت.
مثلا فرض کنید کنترلهای TextBox و Button و ... را در یک پروژه وب فرم یا ویندوز فرم سفارشی کرده باشیم. در این حالت چکار باید بکنیم؟

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۶/۱۱ ۱۹:۲۰

- برنامه FxCop می‌تونه اسمبلی‌های شما رو آنالیز کنه و دقیقا گزارش بده که چه مواردی هم نام کلاس‌های پایه دات نت هستند و بهتر است تغییر نام پیدا کنند. بنابراین به این صورت می‌تونید خیلی سریع حجم بالایی از کدها رو بررسی و رفع اشکال کنید.
- به علاوه زمانیکه طراح شما هستید، محدودیتی در نامگذاری نهایی وجود ندارد. مثلا نام کلاس مشتق شده را NumericTextBox قرار دهید و مواردی مانند این که بیانگر عملکرد سفارشی و ویژه کلاس مشتق شده جدید هستند:

```
public class RequiredTextBox : TextBox
```

عنوان: Best Practice هایی برای طراحی RESTful API - قسمت دوم
نویسنده: محسن درپرستی
تاریخ: ۱۳۹۲/۱۲/۰۳ ۱۴:۰
آدرس: www.dotnettips.info
گروه‌ها: ASP.NET Web API, Naming, Web API, RESTful API, Best Practice

طراحی Url در Restful API

Url بخش اصلی و راه ارتباطی API شما با توسعه دهنده است .بنابراین طراحی یک ساختار مناسب و یکپارچه برای Url ها دارای اهمیت زیادی است .

Url پایه API خود را ساده و خوانا ، حفظ کنید . داشتن یک Url پایه ساده استفاده از API را آسان کرده و خوانایی آن را بالا میبرد و باعث می‌شود که توسعه دهنده برای استفاده از آن نیاز کمتری به مراجعه به مستندات داشته باشد. پیشنهاد می‌شود که برای هر منبع تنها دو Url پایه وجود داشته باشد . یکی برای مجموعه ای از منبع موردنظر و دیگری برای یک واحد مشخص از آن منبع .
برای مثال اگر منبع موردنظر ما کتاب باشد ، خواهیم داشت :

```
.../books
```

برای مجموعه‌ی کتابها و

```
.../books/1001
```

برای کتابی با شناسه 1001
استفاده از این روش یک مزیت دیگر هم به همراه دارد و آن دور کردن افعال از Url ها است.

بسیاری در زمان طراحی Url ها و در نامگذاری از فعل‌ها استفاده می‌کنند. برای هر منبعی که مدلسازی می‌کنید هیچ وقت نمی‌توانید آن را به تنهایی و جداافتاده در نظر بگیرید. بلکه همیشه منابع مرتبطی وجود دارند که باید در نظر گرفته شوند. در مثال کتاب می‌توان منابعی مثل نویسنده ، ناشر ، موضوع و ... را بیان کرد. حالا سعی کنید به تمام Url هایی که برای پوشش دادن تمام درخواست‌های مربوط به منبع کتاب نیاز داریم فکر کنید . احتمالا به چیزی شبیه این می‌رسیم :

```
.../getAllBooks
.../getBook
.../newBook
.../getNewBooksSince
.../getComputerBooks
.../BooksNotPublished
.../UpdateBookPriceTo
.../bookForPublisher
.../GetLastBooks
.../DeleteBook
….
```

خیلی زود یک لیست طولانی از Url ها خواهید داشت که به علت نداشتن یک الگوی ثابت و مشخص استفاده از API شما را واقعا سخت می‌کند.

پس حالا این درخواست‌های متنوع را چطور با دو Url اصلی انجام دهیم ؟
-1 از افعال Http برای کار کردن بر روی منابع استفاده کنید . با استفاده از افعال Http شامل PUT ، GET ، POST و DELETE و دو Url اصلی ، یک مجموعه‌ی مناسب از عملیات‌ها در دسترس توسعه دهنده خواهد بود . به جدول زیر نگاه کنید .

منبع	POST Create	GET Read	PUT Update	DELETE Delete
/books	ثبت کتاب جدید	لیست کتابها	بروزرسانی کلی کتابها	حذف تمام کتابها
/books/1001	خطا	نمایش کتاب ۱۰۰۱	اگر وجود داشته باشد بروزرسانی وگرنه خطا	حذف کتاب ۱۰۰۱

توسعه دهندگان احتمالا نیازی به این جدول برای درک اینکه API چطور کار می‌کند نخواهند داشت.

-2 با استفاده از نکته قبلی بخشی از Url های بالا حذف خواهند شد. اما هنوز با روابط بین منابع چکار کنیم؟ منابع تقریبا همیشه دارای روابطی با دیگر منابع هستند . یک روش ساده برای بیان این روابط در API چیست ؟ به مثال کتاب برمیگردیم. کتاب‌ها دارای نویسنده هستند. اگر بخواهیم کتاب‌های یک نویسنده را برگردانیم چه باید بکنیم؟ با استفاده از Url های پایه و افعال Http می‌توان اینکار را انجام داد. یکی از ساختارهای ممکن این است :

```
GET .../authors/1001/books
```

اگر بخواهیم یک کتاب جدید به کتابهای این نویسنده اضافه کنیم :

```
POST .../authors/1001/books
```

و حدس زدن اینکه برای حذف کتابهای این نویسنده چه باید کرد ، سخت نیست .

-3 بیشتر API ها دارای پیچیدگی‌های بیشتری نسبت به Url اصلی یک منبع هستند . هر منبع مشخصات و روابط متنوعی دارد که قابل جستجو کردن، مرتب سازی، بروزرسانی و تغییر هستند. Url اصلی را ساده نگه دارید و این پیچیدگی‌ها را به کوئری استرینگ منتقل کنید.

برای برگرداندن تمام کتابهای با قیمت پنج هزار تومان با قطع جیبی که دارای امتیاز 8 به بالا هستند از کوئری زیر می‌شود استفاده کرد :

```
GET .../books?price=5000&size=pocket&score=8
```

و البته فراموش نکنید که لیستی از فیلدهای مجاز را در مستندات خود ارائه کنید.

4 - گفتیم که بهتر است افعال را از Url ها خارج کنیم . ولی در مواردی که درخواست ارسال شده در مورد یک منبع نیست چطور؟ مواردی مثل محاسبه مالیات پرداختی یا هزینه بیمه ، جستجو در کل منابع ، ترجمه یک عبارت یا تبدیل واحدها . هیچکدام از اینها ارتباطی با یک منبع خاص ندارند. در این موارد بهتر است از افعال استفاده شود. و حتما در مستندات خود ذکر کنید که در این موارد از افعال استفاده می‌شود.

```
.../convert?value=25&from=px&to=em
.../translate?term=web&from=en&to=fa
```

5 - استفاده از اسامی جمع یا مفرد

با توجه به ساختاری که تا اینجا طراحی کرده ایم بکاربردن اسامی جمع بامعناتر و خواناتر است. اما مهمتر از روشی که بکار می‌برید ، اجتناب از بکاربردن هر دو روش با هم است ، اینکه در مورد یک منبع از اسم منفرد و در مورد دیگری از اسم جمع استفاده کنید . یکدستی API را حفظ کنید و به توسعه دهنده کمک کنید راحت‌تر API شما را یاد بگیرد.

-6 استفاده از نام‌های عینی به جای نام‌های کلی و انتزاعی

API ی را در نظر بگیرید که محتواهایی را در فرمت‌های مختلف ارائه می‌دهد. بلاگ ، ویدئو ، اخبار و .... حالا فرض کنیداین API منابع را در بالاتری سطح مدسازی کرده باشد مثل /items یا /assets . درک کردن محتوای این API و کاری که می‌توان با این API انجام داد برای توسعه دهنده سخت است . خیلی راحت‌تر و مفیدتر است که منابع را در قالب بلاگ ، اخبار ، ویدئو مدلسازی کنیم .