**C#**

**Quy Chuẩn đặt tên trong C#**

1. **Vì sao phải tuân theo chuẩn code, mục đích và coding style là gì:**

* Công việc lập trình không thể trở thành hoàn hảo nếu chỉ có duy nhất một cá nhân. Do đó, doanh nghiệp lập trình ngày nay thường làm việc theo từng teamwork (nhóm, đội). Để mỗi member (thành viên) trong một teamwork có thể làm việc đạt hiệu quả cao nhất, chúng ta cần phải đặt ra một số chuẩn làm việc chung. Coding-Style là yêu cầu trước nhất và quan trọng nhất.
* Vậy Coding-Style là gì? Nó là cách coder (lập trình viên) trình bày trên trang code của họ. Để từ đó, họ và những member còn lại trong teamwork có thể hiểu và triển khai tiếp theo. Chúng ta hãy xét một trang code C# sau:

using System;

namespace Foo

{

public class NormalCalculator

{

private int x;

private int y;

public NormalCalculator() : this(0, 0)

{ }

public NormalCalculator(int x, int y)

{

this.x = x;

this.y = y;

}

public int X

{

get { return x; }

set { x = value; }

}

public int Y

{

get { return y; }

set { y = value; }

}

public int Sum()

{

return x + y;

}

public int Sub()

{

return x - y;

}

public static void Main(string[] args)

{

NormalCalculator c1 = new NormalCalculator();

c1.X = 4;

c1.Y = 2;

Console.WriteLine("Sum is: {0}", c1.Sum().ToString());

NormalCalculator c2 = new NormalCalculator(9, 4);

Console.WriteLine("Sub is: {0}", c2.Sub().ToString());

Console.ReadKey(true);

}

}

}

* Đoạn code trên rất đơn giản, bất kỳ C# coder nào cũng có thể đọc, hiểu, compile và run được; nhưng khi bước vào thực hiện một dự án lớn có nhiều người tham gia, thì việc viết code như ví dụ trên là không thể chấp nhận được.
* Giả sử yêu cầu của dự án buộc bạn phải viết những trang code “trời ơi đất hởi” chằng chịt; bạn viết theo kiểu không chú thích, đặt tên tùy ý rồi submit trang code đó lên Server; một lập trình viên khác Update project về, nhìn thấy trang code của bạn; bạn có nghĩ rằng người lập trình viên đó sẽ hiểu những gì bạn viết không? Tôi thì cam đoan rằng: kể cả bạn, nếu muốn edit lại trang code đó, thì bạn cũng sẽ chẳng biết chỗ nào để mà bắt đầu.
* Như vậy, *Coding-Style* *là cách mà một lập trình viên trình bày trên trang code của họ, để chính họ và những thành viên khác trong dự án có thể dễ dàng thao tác trong suốt quá trình làm việc.*

1. **Phạm vi áp dụng:** Sinh viên khoa công nghệ thông tin và các bạn yêu thích lập trình theo nhóm.
2. **Tài liệu liên quan:**

Có đính kèm theo

1. **Các nội dung quy định:**
   1. Các kiểu quy ước viết hoa:  
      Có 3 quy tắc viết hoa:  
        
      ***P****ascal* ***C****ase*  
        
      Chữ cái đầu tiên trong từ định danh và chữ cái đầu tiên của mối từ nối theo sau phải được viết hoa. Sử dụng Pascal case để đặt tên cho một tên có từ 3 ký tự trở lên.  
        
      Ví dụ: **B**ack**C**olor, **N**hap**S**an**P**ham  
       *Camel Case*  
        
      Chữ cái đầu tiên trong từ định danh là chữ thường và chữ cái đầu tiên của mối từ nối theo sau phải được viết hoa.  
        
      Ví dụ: **b**ack**C**olor, **n**hap**S**an**P**ham  
        
      Uppercase  
        
      Tất cả các ký tự trong từ định danh phải được viết hoa. Sử dụng quy tắc này đối với tên định danh có từ 2 ký tự trở xuống.

**Ký pháp Hungarian:** là cách quy định đặt **tên biến** như sau: bắt đầu tên biến thì viết chữ thường và các chữ đầu thể hiện kiểu dữ liệu của biến, và được gọi là các tiền tố

Ví dụ:  
  
System.IO  
System.Web.IO  
  
Bảng tóm tắt:

|  |  |  |
| --- | --- | --- |
| **Kiểu dữ liệu** | **Case** | **Ví Dụ** |
| Variable | Ký pháp Hungarian |  |
| Class | Pascal Case | AppDomain |
| Enum Type | Pascal Case | ErrorLevel |
| Enum Value | Pascal Case | FatalError |
| Exception class | Pascal Case | WebException  Note: Always ends with the suffix Exception |
| Read-only Static field | Pascal Case | RedValue |
| Interface | Pascal Case | IDisposable  Note: Always begins with the prefix I |
| Method | Pascal Case | ToString |
| Namespace | Pascal Case | System.Drawing |
| Parameter | Camel Case | typeName |
| Property | Pascal Case | BlackColor |
| Protected instance file | Camel Case | redValue  Note: Rarely Use. A property is preferable to using a protected instance filed |
| Public instance file | Pascal Case | RedValue  Note: Rarely use. A property is preferable to using a public instance fild |

* 1. Một **số lưu ý trong cách đặt tên cho các thành phần**  
       
     **1** . Không dùng các tên giống nhau(chỉ phân biệt kiểu chữ in hoa hay thường). Ta khó nhận ra các định danh nhất là khi trong cùng ngữ cảnh và chỉ phân biệt các định danh bằng kiểu chữ in hoa/thường.  
       
     **2 .**Không tạo 2 namespace cùng tên và chỉ khác nhau ở kiểu chữ viết(chữ hoa/Chữ thường), ví dụ:  
       
     Namespace IridiumSoftware  
       
     Namespace iridiumsoftware  
       
     Không nên xây dựng 1 method với các tham số có cùng tên và chỉ khác nhau kiểu chữ, ví dụ:  
     void MyFunction(string a, string A)  
       
     Không xây dựng 1 kiểu với các tên property giống nhau và chỉ phân biệt ở kiểu chữ, ví dụ:  
     int Color {get, set}  
       
     int COLOR {get, set}  
       
     Không đặt tên các phương thức có cùng tên và chỉ khác nhau ở kiểu chữ, ví dụ:  
     void calculate()  
       
     void Calculate()  
       
     Không dùng từ viết tắt hoặc 1 phần của các tên định danh như: GetWindow viết thành GetWin là ko hợp quy ước.  
     Ta có thể sử dụng kí tự đầu tiên của các tên có một nhóm từ để đặt tên cho nó.  
     Ví dụ: UI thay cho User Interface.  
       
     OLAP thay cho On-Line Analytical Processing  
       
     Không sử dụng các từ viết tắt đã được thừa nhận từ các lĩnh vực tin học khác. (Ví dụ: XML, TTL, DNS, UI, IP và IO …)  
     Khi đặt tên viết tắt, bạn có thể sử dụng cách viết Pascal case hay Camel case để đặt tên, như trường hợp này bạn có thể dùng tên nào cũng được **HtmlButton** hoặc **HTMLButton**. Tuy nhiên với trường hợp tên có 2 ký tự hoặc ít hơn thì bạn nên viết như **System.IO** thay cho **System.Io**.  
     Không dùng từ viết tắt trong tên định danh hoặc tên tham sô. Nếu bạn cần thiết phải dùng đến từ viết tắt, dùng cách viết Camel Case cho từ viết tắt có từ 2 kí tự trở lên thậm chí là nó phủ định lại chuẩn viết tắt của 1 từ.  
     Tránh sử dùng lại các từ khóa hoặc tên các class chuẩn được dùng trong .NET Framework namespaces.
  2. **Cách tổ chức file trong 1 project:**
     1. **File sourcecode C#**  
        Giữ file nguồn/ các class không quá dài(không được vượt quá 2000 LOC (Lines of Code), phân chia code thành những đơn vị code nhỏ tạo nên các cấu trúc code sáng sủa hơn. Đặt mỗi class vào mỗi file source riêng biệt và tên file cũng là tên class(với phần đuôi mở rộng .cs)  
          
        Thứ tự trong file source C#:  
          
        • các lệnh using  
        • lệnh namespace  
        • Các khai báo Class and interface  
          
        Các lệnh Namespace và using:  
          
        Các dòng không phải là dòng comment ở hầu hết các file source của C# là các lệnh using, kế đến là lệnh namespace như ví dụ sau:  
          
        using System.Data;  
        namespace Business.Framework;  
          
        Cả 2 lệnh này đề được canh sát lề trái.  
          
        Các khai báo class và interface:  
          
        Thứ tự  
        Các phần trong khai báo class và interface  
          
        Ghi chú  
        1  
          
        Class/interface documentation /// <summary>  
        /// The Person class provides …  
          
        /// </summary>  
          
        public class Person  
          
        2  
          
        class or interface   
        3  
          
        Fields Trình tự khai báo các fields theo mức độ cho phép truy cập:  
        Private  
        Protected  
        Internal  
        Public  
          
        4  
          
        Properties Trình tự khai báo các Properties theo mức độ cho phép truy cập:  
        Private  
        Protected  
        Internal  
        public  
          
        5  
          
        Constructors Trình tự khai báo các Constructors theo mức độ cho phép truy cập:  
        Private  
        Protected  
        Internal  
        Public  
          
        Trước tiên là constructor mật định, sau đó là các constructor theo thứ tự độ phức tạp tăng dần.  
          
        6  
          
        Methods Các Method nên được nhóm theo chức năng đúng hơn là nhóm theo mức độ truy cập.
     2. **Bố trí thư mục sourcecode C#**  
        Tạo 1 thư mục cho mỗi namespace trong project.(Với namespace MyProject.TestSuite.TestTier ta sẽ có 1 đường dẫn thư mục tương ứng MyProject/  
          
        TestSuite/TestTier). Và ta dễ dàng ánh xạ các namespace sang cấu trúc thư mục trong Project Explorer.
  3. **Quy định về đặt tên biến**
     1. **Hướng dẫn đặt tên Namespace**Quy tắc chung cho việc đặt tên Namespace là dùng tên của công ty theo sau là tên công nghệ và kế đến là tùy chọn tên đặc trưng hay thiết kế:  
          
        CompanyName.TechnologyName[.Feature][.Design]  
          
        Ví dụ:  
          
        IridiumSoftware.IridiumX  
          
        IridiumSoftware.IridiumX.Design  
          
        Microsoft.Office
     2. **Hướng dẫn đặt tên class**  
        Thường dùng danh từ hoặc một cụm danh từ để đặt tên cho 1 class.  
        Kí tự đầu tiên của mỗi từ là chữ in hoa.(Dùng quy tắc **Pascal Case**).  
        Không sử dụng dạng tiền tố:  
        Ví dụ: **FileStream** là Tên chuẩn.  
          
        Không đặt tên theo dạng CfileStream hay ClassFileStream  
          
        Trong tên class không sử dụng dấu gạch chân (\_).  
        Trong trường hợp cần thiết, nếu kí tự đâu tiên của tên 1 class bắt đầu bằng kí tự ‘I’ và kí tự này là kí tự bắt đầu của một từ và từ đó là 1 phần của tên class thì tên đó vẫn được chấp nhận.  
         - Ví dụ: IdentityStore  
          
        Sử dụng một từ ghép để đặt tên cho 1 class dẫn xuất, từ thứ 2 trong tên của class dẫn xuất nên lấy tên của class cơ sở.  
        Ví dụ:  
          
        - ApplicationException là tên class dẫn xuất từ class cơ sở Exception, và với cách đặt tên này ta có thể hiểu rằng ApplicationException là một trong các loại Exception.  
          
        Với quy tắc này thì tùy trường hợp mà chúng ta có thể áp dụng để tránh đặt tên class dài lê thê với những từ không cần thiết.  
          
        Ví dụ như trường hợp sau:  
          
        Class Button được dẫn xuất từ class Control nhưng nếu ta đặt tên theo quy tắc trên thì tên class này là ButtonControl. Tên class này trở nên dài ra và không cần thiết vì mật định thì button được xem như là một control rồi.  
          
        Dưới đây là cách đặt tên class chuẩn:  
          
        public class FileStream  
          
        public class Button  
          
        public class String
     3. **Hướng dẫn đặt tên Interface**  
          
        - Thường dùng danh từ hoặc một cụm danh từ, hay một tính từ mà nó mô tả hành vi để đặt tên cho 1 interface.  
          
        Ví dụ:  
          
        Tên interface Icomponent sử dụng 1 danh từ.  
          
        ICustomAttributeProvider sử dụng 1 cụm danh từ.  
          
        IPersistable sử dụng một tính từ.  
          
        - Sử dụng quy tắc PacalCase.  
          
        - Tên đặt cho interface phải ngắn gọn.  
          
        - Khi định nghĩa 1 cặp class/interface mà class này là 1 thực thi chuẩn của interface thì tên giữa class và interface nên đặt tương tự nhau. Và điểm khác nhau là tên interface có kí tự tiền tố “I”.  
          
        - Không sử dụng kí tự gạch chân “\_” trong tên interface.  
          
        Dưới đây là các ví dụ đặt tên interface chuẩn:  
          
        public interface IServiceProvider  
          
        public interface IFormatable  
          
        Dưới đây là đoạn code minh họa cách định nghĩa interface, một class thực thi chuẩn của interface:  
          
        public interface IComponent  
          
        {  
          
         // Implementation goes here.  
          
        }  
          
        public class Component : IComponent  
          
        {  
          
         // Implementation goes here.  
          
        }
     4. **Hướng dẫn đặt tên Attribute**  
        Bạn nên thêm vào hậu tố Attribute đối với các lớp attribute.  
          
        public class ObsoleteAttribute{}
     5. **Hướng dẫn đặt tên Enumeration**  
        Kiểu giá trị Enum thừa kế từ Enum class. Sau đây là các quy tắc hướng dẫn đặt tên cho enumerations:  
          
        1 . Sử dụng Pascal Case cho các tên kiểu và giá trị enum.  
          
        2 . Sử dụng tên ngắn gọn và tường minh.  
          
        3 . Không sử dụng hậu tố Enum trong tên kiểu Enum.
     6. **Hướng dẫn đặt tên Static Field**  
          
        Quy tắc đặt tên cho các static field:  
          
        1 . Dùng các danh từ, cụm danh từ hay các danh từ viết tắt để đặt tên cho static fields.  
          
        2 . Dùng Pascal Case.  
          
        3 . Sử dụng tiền tố ký pháp Hungarian trong tên của static field.  
          
        4 . Người ta khuyên dùng thuộc tính static để thay thế cho các static fields có cấp độ chia sẻ dạng public.
     7. **Hướng dẫn đặt tên Parameter**  
          
        1 . Tên Parameter phải được mô tả một cách đầy đủ và nó bao hàm cả tên của tham số và kiểu dữ liệu của nó  
          
        2 . Dùng Camel Case.  
          
        3 . Tên tham số nên đặt theo hướng mô tả ý nghĩa của tham số tốt hơn là đặt theo kiểu dữ liệu của tham sô. Vì môi trường lập trình đã cung cấp các thông tin về kiểu dữ liệu tham số.  
          
        4 . Không nên dùng các tham số để dành , mà khi cần thiết ta có thể thêm vào trong các version sau này của các thư viện class.  
          
        5 . Không sử dụng các tên tham số tiền tố với kí pháp Hungarian.  
          
        Các tên tham số chuẩn:  
          
        Type GetType(string typeName)  
          
        string Format(string format, object[] args)
     8. **Hướng dẫn đặt tên Method**  
         1 . Dùng các động từ hay các cụm động từ để đặt tên cho methods.  
          
         2 . Dùng Pascal Case.  
          
         Ví dụ tên các methods chuẩn:  
         RemoveAll()  
         GetCharArray()  
         Invoke()
     9. **Hướng dẫn đặt tên Property**1 . Dùng 1 danh từ hay 1 cụm danh từ để đặt tên cho 1 property.  
          
        2 . Dùng Pascal Case.  
          
        3 . Không dùng ký pháp Hungarian.  
          
        4 . Tạo 1 property và đặt tên tương tự như kiểu dữ liệu của nó.  
          
        Ví dụ minh họa cho tên một property:  
          
        public class SampleClass  
        {  
          
         public Color BackColor  
         {  
         // Code for Get and Set accessors goes here.  
         }  
        }
     10. **Hướng dẫn đặt tên Event**1 . Dùng 1 hậu tố EventHandler trong các tên của event handler.  
           
         2 . Chỉ định rõ 2 tham số có tên là sender và e. Tham sô sender mô tả object gọi event. Tham sô sender luôn luôn có kiểu dữ liệu là object. Trạng thái kết hợp với sự kiện được đóng gói trong một thể hiện của event class có tên là e. Kiểu của tham số e phải là một lớp sự kiện rõ ràng và thích hợp.  
           
         3 . Đặt tên 1 lớp đối số sự kiện với hậu tố EventArgs.  
           
         4 . Nên đặt tên 1 sự kiện là 1 động từ.  
           
         5 . Dùng danh động từ(dạng thêm “ing” của động từ) để đặt tên cho một sự cho một sự kiện mà nó thể hiện khái niệm pre\_event và quá khứ của động từ để thể hiện sự kiện post\_event.  
           
         Ví dụ, 1 sự kiện ‘Close’ có thể được hủy bỏ vì thể nên có thêm sự kiện ‘Closing’ và ‘Closed’. Không dùng mẫu đặt tên sự kiện dạng BeforeXX/AfterXXX.  
           
         6 . Không dùng tiền tố và hậu tố trong khai báo event có dạng “OnXXX”. Ví dụ: dùng Close thay vì dùng OnClose.  
           
         7 . Tổng quát hơn, bạn nên cung cấp 1 method có mức truy cập protected gọi sự kiện có dạng “OnXXX” mà sự kiện này có thể bị ghi đè trong 1 class dẫn xuất. Method này chỉ nên có tham số sự kiện e.  
           
         Ví dụ minh họa 1 event handler với 1 tên chuẩn và các tham sô:  
           
         public delegate void MouseEventHandler(object sender, MouseEventArgs e);
     11. **Hướng dẫn đặt tên Control**  
         Bảng danh sách các tiền tố của các kiểu controls thông dụng:

|  |  |
| --- | --- |
| Control | Prefix |
| Label | lbl |
| LinkLabel | llbl |
| Button | btn |
| TextBox | Txt |
| MainMenu | Mmu |
| RadioButton | rdo |
| GroupBox | grp |
| PictureBox | pic |
| Grid | grd |
| ListBox | lst |
| ComboBox | cbo |
| ListView | lstv |
| TreeView | tre |
| TabControl | tab |
| DateTimePicker | dtm |
| MonthCalendar | mon |
| ScrollBar | scr |
| Timer | tmr |
| Splitter | spl |
| DomainUpDown | dud |
| NumericUpDown | nud |
| TrackBar | trk |
| ProgressBar | pro |
| RichTextBox | rtb |
| ImageList | img |
| HelpProvider | hlp |
| ToolTip | tip |
| ContextMenu | cmnu |
| ToolBar | tbr |
| Form | frm |
| StatusBar | stb |
| NotifyIcon | nico |
| OpenFileDialog | ofd |
| SaveFileDialog | sfd |
| FontDialog | fod |
| ColorDialog | cod |
| PrintDialog | prd |
| PrintPreviewDialog | ppd |
| PrintPreviewControl | ppc |
| ErrorProvider | err |
| PrintDocument | pdoc |
| PageSetupDialog | psd |
| CrystalReportViewer | Crv |
| PrintDialog | prd |
| FileSystemWatcher | Fsw |
| EventLog | Evl |
| DrectoryEntry | dire |
| DirectorySearcher | dirs |
| MessageQueue | msq |
| PerformanceCounter | pco |
| Process | pro |
| ServiceController | ser |
| ReportDocument | rpt |
| DataSet | ds |
| OleDbDataAdapter | olea |
| OleDbConnection | olec |
| OleDbCommand | oled |
| SqlDbDataAdapter | sqla |
| SqlDbConnection | sqlc |
| SqlDbCommand | sqld |
| DataView | dvw |
|  |  |

**Menu Controls**  
Menu controls nên đặt tên bằng cách dùng tag “mnu” theo sau là đường dẫn xuống đầy đủ theo cây menu. Ví dụ về tên 1 biến control menu:  
  
mnuFile  
mnuFileNew  
mnuEdit  
mnuEditCopy  
mnuInsertIndexAndTables  
mnuTableCellHeightAndWidth

* + 1. **Hướng dẫn đặt tên Data**  
       Nếu cần dùng tên CustomerCode và hiển thị lên 1 text box, thì text box phải đặt tên là txtCustomerCode.  
       Nếu dùng combobox để hiển thị các mã số khách hàng, thì control đó đặt tên là cboCustomerCode.  
       Nếu muốn lưu mã khách hàng vào biến của 1 module thì nó được đặt tên là mCustomerCode.  
       Nếu muốn chuyển sang chuỗi thì dùng câu lệnh sau:  
       String mCustomerCode = CustomerCode.ToString()  
       Fields in Databases:  
       Với các quy tắc tổng quát, chúng ta có thể dùng Property Naming Guidelines khi đặt tên cho các field trong database. Nhưng quy tắc này không phù hợp lắm với những database đã có sẵn.
  1. **Quy định về comment:**  
     Các chương trình C# có thể có 2 loại comment là implementation comments và documentation comments.  
     + Implementation comments là những comment trong C++, mà được định nghĩa bới các dấu /\*…\*/, và //.  
     + Documentation comments thì chỉ có trong C#, và được định nghĩa bởi các tag XML. Implementation comments dùng để giải thích cho 1 đoạn code hay để chú thích cho 1 implemention một cách cụ thể hơn.  
     Chú ý: Tần số suất hiện của các comment nhiều đôi khi mang lại sự nghèo túng cho code. Vì thế khi bạn cảm thấy cần phải thêm comment, thì nên xem lại có thể viết code lại cho nó tường minh đựơc không.  
     Dưới đây là một số kỹ thuật comment:  
     • Khi sửa code nên nhớ comment ngày sửa code.  
     • Comments nên chứa các câu giải thích hoàn.  
     • Tránh sử dụng comment cuối dòng cho các đoạn code (trừ các khai báo biến).  
     • Tránh dùng các dòng comment chứa các đầy dâu hoa thị mà nên dùng các khoảng trắng thay vào đó.  
     • Tránh dùng các khối comment có khung viền nghệ thuật. Mặc dù nhì thì đẹp nhưng rất khó khăn khi bảo trì.  
     • Trước khi chuẩn bị triển khai nên xóa các comment tạm thời hay các comment không liên quan để tránh gặp rối trong quá trình bảo trì sản phẩm sau này.  
     • Dùng các câu comment hoàn thiện. Comment nên sáng sủa không nên thêm vào các câu nhập nhằng.  
     • Để tránh comment lặp lại nhiều lần thì ta nên thực hiện động tác comment vào lúc fix bug cho code, đặc biệt là khi làm việc trong môi trường nhóm.  
     • Nên dùng comments cho code chứa các vòng lặp và điều kiện rẽ nhánh. Nó sẽ hỗ trợ người đọc khi đọc source code.
     1. **Định dạng Implementation Comments**  
        **a) Block comments**  
        Với C# thì comment (///) thường được sử dụng(do VS tự generate). Khi dùng comments cho khối chú thích thì nên dùng theo style sau:  
        /\* Line 1  
        \* Line 2  
        \* Line 3  
        \*/  
        Comment như bên dưới là rất hiếm:  
        /\* blah blah blah \*/  
        Block comments được dùng để mô tả cho các file, method, cấu trúc dữ liệu và các thuật toán.  
        Mỗi file code nên có khối comment header, khối header nên trình bày như bên dưới:  
        #region Copyright Softech@ 2007  
        //  
        // All rights are reserved. Reproduction or transmission in whole or // in part, in any form or by any means, electronic, mechanical or  
        // otherwise, is prohibited without the prior written consent of the // copyright owner.  
        //  
        // Filename: Convention.cs  
        //  
        #endregion **b) Single Line Comments**  
        Nên dùng dấu comment // để comment đối với 1 dòng code và có thể dùng để comment cho 1 đoạn code cũng được.  
        Single line comments phải thuộc vào so với mức canh lề của code khi dùng cho tài liệu code. Comment nên được đặt ở dòng đầu tiên của đoạn code muốn comment để làm tăng tính trong sáng và tường minh của đoạn code.  
        Thông thường thì độ dài của comment không được vượt quá chiều dài của đoạn code muốn được comment, điều này sẽ làm cho code trở nên rối rắm thêm.  
        if (condition)  
        {  
         // Handle the condition.  
         …  
        }  
        **c) Trailing Comments**  
        Dùng với việc mô tả các khai báo biến hay là các mô tả ngắn cho các câu lệnh:  
        if (a == 2)  
        {  
         return true; // Special case  
        }  
        else  
        {  
         return isPrime(a); // Works only for odd a  
        }  
        **d) Code-Disabling Comments**  
        Dấu comment // dùng để che đoạn code được chọn. Để dùng comment này ta dùng các tổ hợp phím CTRL+K, CTRL+C. Để hủy bỏ ấn CTRL+K, CTRL+U.  
        if (foo > 1)  
        {  
         // Do a double-flip.  
         …  
        }  
        else  
        {  
         return false; // Explain why here.  
        }  
          
        // if (bar > 1)  
        // {  
        //  
         // // Do a triple-flip.  
         // …  
        // }  
        // else  
        // {  
         // return false;  
        // }
     2. **Documentation Comments**  
        Trong .net framework, Microsoft đưa ra hệ thống phát sinh tài tài liệu (documentation generation system) dựa vào các comment dạng thẻ XML. Những comments sử dụng dấu comment /// cùng với các thẻ XML. Ví dụ lời comment trên 1 dòng:  
          
        /// <summary>  
        /// This class…  
        /// </summary>  
        Ví dụ về lời comment trên nhiều dòng:  
        /// <exception cref=”BogusException”>  
        /// This exception gets thrown as soon as a  
        /// Bogus flag gets set.  
        /// </exception>  
        Trước tất cả các dòng đều có 3 dấu / đi trước nó. Các Tag chia thành 2 loại:  
        • Documentation items  
        • Formatting/Referencing  
        Loại đầu tiên bao gồm các tag như <summary>, <param> hay <exception>. Những tag này sẽ mô tả các thành phần API của chương trình và nó được tài liệu hóa để cho các lập trình viên khác có thể dùng lại nó. Những tag này thường có các thuộc tính như name.  
        Loại thứ 2 liên quan đến việc bố trí của tài liệu, sử dụng các tag như as <code>, <list> hoặc <para>.
  2. **Quy định về định dạng code**
     1. **Quy định xuống dòng**Khi có 1 biểu thức không còn chứa được trên 1 dòng thì nên xuống dòng tùy vào các nguyên tắc sau:  
          
        • Xuống dòng sau 1 dấu phẩy (,).  
        • Xuống dòng sau 1 toán tử.  
        • Canh lề dòng mới xuống bằng với lề bắt đầu của biểu thức ở dòng trước ở cùng mức như trong ví dụ:  
          
        longMethodCall(expr1, expr2,  
        expr3, expr4, expr4);  
          
        Ví dụ xuống dòng với 1 biểu thức toán:  
          
        Nên dùng:  
        var = a \* b / (c - g + f) +  
        4 \* z;  
          
        Nên tránh(rất khó nhìn):  
        var = a \* b / (c - g +  
        f) + 4 \* z;  
          
        Chú ý, nên thụt đầu dòng với các kí tự tab đối với mức thụt đầu dòng và dùng các ký tự space đối với việc canh lề của các vị trí xuống dòng:  
          
        > var = a \* b / (c - g + f) +  
        > ......4 \* z;  
        Ở ví dụ trên thì ‘>’ là các ký tự tab và ‘.’ là các khỏang trắng. Một thói quen coding tốt là nên cho hiện rõ các ký tự tab và space trong trình soạn thảo mà bạn đang sử dụng.
     2. **White Spaces**Nếu dùng các khoảng trẳng để cho các chuẩn thụt đầu dòng thì sẽ không có sự thống nhất. Có người thích dùng 2,4,8 hay nhiều kí tự trắng hơn, vì vậy tốt nhất là dùng các ký tự **tab** để thụt đầu dòng. Dùng kí tự tab để thụt dòng thì ta có những tiện lợi sau:  
          
        • Mọi người có thể thiết lập mức thụt lề tùy vào sở thích của họ.  
        • Chỉ có 1 lần gõ phím chứ không phải gõ nhiều lần nếu ta dùng đến kí tự space.  
        • Nếu muốn tăng mức thụt lề hay giảm thì chỉ cần bôi đen 1 đoạn code và dùng phím Tab để tăng mức thụt lề và dùng phím Shift-Tab để giảm mức thụt lề. Hầu hết các trình soạn thảo code đều có chức năng này.  
          
        Chúng ta có thể xem **Tab** là kí tự thụt lề chuẩn.  
          
        Các dòng trống(Blank Lines)  
        Các dòng trống giúp người coding dễ đọc. Các dòng trống thường được dùng để phân cách giữa các đoạn code với nhau. Hai dòng trống thường dùng chèn giữa:  
          
        • Các đoạn code của source file  
        • Các định nghĩa Class và interface  
        Một dòng trống được dùng để chèn vào giữa:  
          
        • Methods  
        • Properties  
        • Biến cục bộ trong method với dòng lệnh đầu tiên  
        • Giữa các đoạn code trong 1 method, giúp dễ đọc hơn  
          
        Cách khoảng:  
        **Nên cách khoảng sau 1 dấu phẩy “,” hay dấu chấm phẩy “;” ví dụ:**  
          
        Use: TestMethod(a, b, c);   
        don't use : TestMethod(a,b,c);   
        hay TestMethod( a, b, c );  
        Dùng 1 dấu cách trước và sau 1 toán tử, ví dụ:  
        a = b; // don't use a=b;  
        for (int i = 0; i < 10; ++i)   
        // don't use for (int i=0; i<10; ++i)  
        // or  
        // for(int i=0;i<10;++i)
  3. **Các khai báo**
     1. **Số lượng các khai báo trên mỗi dòng**Một khai báo trên mối dòng nên viết khi chú sau mỗi khai báo:  
          
        int level; // indentation level  
        int size; // size of table  
        Không được đặt nhiều hơn 1 biến hay các biến có kiểu dữ liệu khác nhau trên cùng 1 dòng khi khai báo biến:  
          
        int a, b; //What is 'a'? What does 'b' stand for?  
        Ví dụ trên cũng minh họa cho mặt hạn chế của việc đặt tên biến không rõ ràng.
     2. **Khởi tạo giá trị**Cố gắng khởi tạo giá trị cho các biến ngay sau khi khai báo:  
          
        string name = myObject.Name;  
        hay:  
        int val = time.Hours;  
        Chú ý: Nếu bạn khởi tạo cho 1 dialog thì nên dùng lệnh sau:  
          
        using (OpenFileDialog openFileDialog = new OpenFileDialog()) {  
        ...  
        }
     3. **Khai báo class và interface**  
        Khi coding cho các class hay interface trong C# thì nên kèm theo các quy tắc định dạng:  
          
        • Không dùng khoảng trắng giữa tên method và dấu ngoặc mở ” (” bắt đầu các tham số của nó.  
        • Dấu ngoặc móc mở “{” nên đặt trên dòng kế tiếp của câu lênh khai báo.  
        • Dấu ngoặc nhọn đóng “}” cũng nên đặt trên 1 dòng vào bằng với mức thụt lề của dấu ngoặc mở.  
          
        Ví dụ:  
          
        class MySample : MyClass, IMyInterface  
        {  
         int myInt;  
         public MySample(int myInt)  
         {  
         this.myInt = myInt ;  
         }  
         void Inc()  
         {  
         ++myInt;  
         }  
         void EmptyMethod()  
         {  
         }  
        }
  4. **Statements**
     1. **Các lệnh đơn**Mỗi dòng nên chứa 1 câu lệnh.
     2. **Câu lệnh Return**Mỗi lệnh return không nên dùng cặp ngoặc đơn bao phía ngoài.  
          
        Don’t use : return (n \* (n + 1) / 2);  
          
        use : return n \* (n + 1) / 2;
     3. **Các câu lệnh If, if-else, if else-if else**Các lệnh if, if-else and if else-if else nên bố trí như sau:  
          
        if (condition)   
        {  
         DoSomething();  
         ...  
        }  
        hoặc  
        if (condition)   
        {  
         DoSomething();  
         . ..  
        }   
        else   
        {  
         DoSomethingOther();  
         ...  
        }  
        hoặc  
        if (condition)   
        {  
         DoSomething();  
         ...  
        }   
        else if (condition)   
        {  
         DoSomethingOther();  
         ...  
        }   
        else   
        {  
         DoSomethingOtherAgain();  
         ...  
        }
     4. **Các lệnh For / Foreach**Một câu lệnh for nên có dạng sau:  
          
        for (int i = 0; i < 4; ++i)   
        {  
         ...  
        }  
        Lệnh foreach nên có dạng:  
        foreach (int i in IntList)   
        {  
        ...  
        }  
        Chú ý: Thông thường thì người ta hay dùng cặp ngoặc móc “{” và “}” ngay cả khi trong thân vòng lặp chỉ có 1 câu lệnh.
     5. **Các câu lệnh While/do-while**Câu lệnh while nên viết như sau:

while (condition)   
{  
...  
}  
Một lệnh while rỗng nên trình bày dạng:  
  
while (condition) ;  
Một lện do-while nên có dạng:  
  
do   
{  
 ...  
} while (condition);

* + 1. **Câu lệnh Switch**Câu lệnh switch nên viết dạng:  
         
       switch (condition)   
       {  
        case A:  
        {  
        ...  
        break;  
        }  
        case B:  
        {  
        ...  
        break;  
        }  
        default:  
        {  
        ...  
        break;  
        }  
       }

**Câu lệnh Try-catch**Câu lệnh try-catch nên code dạng sau:  
  
try   
{  
 ...  
}   
catch (Exception)   
{  
}  
hoặc  
try   
{  
 ...  
}   
catch (Exception e)   
{  
 ...  
}  
hoặc  
try   
{  
 ...  
} catch (Exception e)   
{  
 ...  
}   
finally   
{  
 ...  
}

**Ví dụ và phân tích cụ thể:**

**Ví dụ:**

#region - file information(s) -

/\*

\* Name : Cat.cs

\* Fullpath : ...\SharpCodeHunter.Zoo\Cat.cs

\* Programmer : O'Wicked Fox

\* Datetime : 4/4/09 - 19:17

\* Reversion : 1

\*/

#endregion

#region - using directive(s) -

#region - dotNET 2.0 -

using System;

#endregion

#endregion

namespace SharpCodeHunter.Zoo

{

/// <summary>

/// A lovely pet in house.

/// </summary>

public class Cat : ITerrestrialAnimal

{

#region - field(s) -

/// <summary>

/// the ID of a cat;

/// </summary>

private int mCatID;

/// <summary>

/// the name of a cat.

/// </summary>

private string mCatName;

/// <summary>

/// the age of a cat.

/// </summary>

private int mCatAge;

#endregion

#region - constructor(s) -

/// <summary>

/// Constructs a default cat.

/// </summary>

public Cat()

{

mCatID = -1;

mCatName = "";

mCatAge = -1;

}

#endregion

#region - property(ies) -

#region - implements IAnimal member(s) -

/// <summary>

/// Gets, sets the cat's ID.

/// </summary>

public int ID

{

get { return mCatID; }

set { mCatID = value; }

}

/// <summary>

/// Gets, sets the cat's name.

/// </summary>

public string Name

{

get { return mCatName; }

set { mCatName = value; }

}

/// <summary>

/// Gets, sets the cat's age.

/// </summary>

public int Age

{

get { return mCatAge; }

set { mCatAge = value; }

}

#endregion

#endregion

#region - method(s) -

#region - implements IAnimal member(s) -

/// <summary>

/// The way a cat moves.

/// </summary>

public void Move()

{

Console.WriteLine("Run");

}

#endregion

#region - overriddens Object member(s) -

public override int GetHashCode()

{

int hashCode = 0;

unchecked

{

hashCode += 1000000007 \* mCatID.GetHashCode();

}

return hashCode;

}

public override bool Equals(object other)

{

Cat otherOne = other as Cat;

if (otherOne == null) return false;

return this.mCatID == otherOne.mCatID;

}

public override string ToString()

{

return string.Format("[Cat No.{0}; Name:{1}; Age:{2}]",

this.mCatID,

this.mCatName,

this.mCatAge);

}s

#endregion

#endregion

}

}

Phân tích

|  |  |  |  |
| --- | --- | --- | --- |
| **T.Hợp** | **Đối Tượng** | **Kiểu** | **Cách đặt tên** |
| **1** | Object | Class | * Độ dài kí tự phải <= 40. * Đặt tên theo chuẩn Pascal-case, và mang ý nghĩa một danh từ.   *Ví dụ*:  ...  public class Cat  {  ...  }   * Một đối tượng kế thừa từ Form, sẽ thêm hậu tố   ‘-Screen’.  *Ví dụ*:  ...  public partial class AccountScreen : Form  {  ...  }   * Một đối tượng chứa phương thức Main(), sẽ có tên là ‘Program’.   *Ví dụ*:  ...  internal sealed class Program  {  ...  private static void Main(string[] args)  {  ...  }  } |
| **2** | Abstract Class | * Đặt tên giống như Class, nhưng tùy trường hợp, mà thêm tiền tố ‘Abstract-’ hoặc hậu tố ‘-Base’.   *Ví dụ*:  ...  public abstract class CollectionBase  {  ...  }  ...  public abstract class AbstractDatabase  {  ...  } |
| **3** | Interface | * Đặt tên giống Class, bắt buộc thêm tiền tố ‘I-’.   *Ví dụ*:  ...  public interface IAnimal  {  ...  } |
| **4** | Variable (biến) | Tất cả | * Độ dài kí tự phải <= 40. * Đặt tên biến phải bao quát ý nghĩa khởi tạo của biến; * nếu tên biến dài quá, có thể viết tắt. * Đối với những biến có kiểu boolean, chúng ta nên tiền tố ‘is-’, ‘has-’, ‘can-’.   *Ví dụ*:  ...  private bool mHasNextElement = false;  ...  bool isExist = false;  ... |
| **4** | Global variable | * Thêm tiền tố ‘m-’   *Ví dụ*:  ...  private int mCrocodileID;  ...  protected int mLeg;  ... |
| **6** | Global static variable | * Thêm tiền tố ‘ms-’   *Ví dụ*:  ...  private static Stack<object> msUndoStack =  new Stack<object>();  ... |
| **7** | Global readonly variable | * Thêm tiền tố ‘mr-’   *Ví dụ*:  ...  private readonly string mrName;  ... |
| **8** | Global static readonly variable | * Thêm tiền tố ‘msr-’   *Ví dụ*:  ...  private static readonly string msrVersion = "Alpha";  ... |
| **9** | Global constant variable | * Upper-case và mỗi từ cách nhau bởi ‘\_’   *Ví dụ*:  ...  private const int DEFAULT\_CAPACITY = 10;  ... |
| **10** | Public global variable | * Lower-case và mỗi từ cách nhau bởi ‘\_’   *Ví dụ*:  ...  public int tail\_index;  ... |
| **11** | Các trường hợp còn lại | * Đặt tên theo chuẩn Camel-case. |
| **12** | Method (phương thức) | Tất cả | * Độ dài kí tự phải <= 100. * Đặt tên theo chuẩn Pascal-case, và từ đầu tiên phải là một động từ.   *Ví dụ*:  ...  public void Move()  {  Console.WriteLine("Swim");  }  ... |
| **13** | Property (thuộc tính) | Tất cả | * Độ dài kí tự phải <= 40. * Không có bất kỳ một tiền tố nào, đặt tên theo chuẩn Pascal-case theo tên đối tượng sẽ làm việc.   *Ví dụ*:  ...  private int mCrocodileAge;  ...  public int Age  {  get { return mCrocodileAge; }  set { mCrocodileAge = value; }  }  ... |

Muc luc :

[1. **Vì sao phải tuân theo chuẩn code, mục đích và coding style là gì:** 1](#_Toc259304005)

[2. **Phạm vi áp dụng:**. 4](#_Toc259304006)

[3. **Tài liệu liên quan:** 4](#_Toc259304007)

[4. **Các nội dung quy định:** 4](#_Toc259304008)

[4.1. Các kiểu quy ước viết hoa:. 4](#_Toc259304009)

[4.2. Một **số lưu ý trong cách đặt tên cho các thành phần** . 5](#_Toc259304010)

[4.3. **Cách tổ chức file trong 1 project:** 7](#_Toc259304011)

[4.3.1. **File sourcecode C#**  . 7](#_Toc259304012)

[4.4. **Quy định về đặt tên biến** 9](#_Toc259304014)

[4.4.1. **Hướng dẫn đặt tên Namespace**  9](#_Toc259304015)

[4.4.2. **Hướng dẫn đặt tên class**  9](#_Toc259304016)

[4.4.4. **Hướng dẫn đặt tên Attribute**  12](#_Toc259304018)

[4.4.5. **Hướng dẫn đặt tên Enumeration**  12](#_Toc259304019)

[4.4.6. **Hướng dẫn đặt tên Static Field** . 12](#_Toc259304020)

[4.4.7. **Hướng dẫn đặt tên Parameter** 12](#_Toc259304021)

[4.4.8. **Hướng dẫn đặt tên Method**  13](#_Toc259304022)

[4.4.9. **Hướng dẫn đặt tên Property**  13](#_Toc259304023)

[4.4.10. **Hướng dẫn đặt tên Event**  14](#_Toc259304024)

[4.4.11. **Hướng dẫn đặt tên Control**  15](#_Toc259304025)

[4.4.12. **Hướng dẫn đặt tên Data**  17](#_Toc259304027)

[4.5. **Quy định về comment:** 17](#_Toc259304028)

[4.5.1. **Định dạng Implementation Comments** **a) Block comments** 18](#_Toc259304029)

[4.5.2. **Documentation Comments**. 20](#_Toc259304030)

[4.6. **Quy định về định dạng code** 20](#_Toc259304031)

[4.6.1. **Quy định xuống dòng**. 21](#_Toc259304032)

[4.6.2. **White Spaces**) 21](#_Toc259304033)

[4.7. **Các khai báo** 22](#_Toc259304034)

[4.7.1. **Số lượng các khai báo trên mỗi dòng** 23](#_Toc259304035)

[4.7.2. **Khởi tạo giá trị** 23](#_Toc259304036)

[4.7.3. **Khai báo class và interface** 23](#_Toc259304037)

[4.8. **Statements** 24](#_Toc259304038)

[4.8.1. **Các lệnh đơn** Mỗi dòng nên chứa 1 câu lệnh. 24](#_Toc259304039)

[4.8.2. **Câu lệnh Return** 24](#_Toc259304040)

[4.8.3. **Các câu lệnh If, if-else, if else-if else** 24](#_Toc259304041)

[4.8.4. **Các lệnh For / Foreach**. 25](#_Toc259304042)

[4.8.5. **Các câu lệnh While/do-while** 25](#_Toc259304043)

[4.8.6. **Câu lệnh Switch** 26](#_Toc259304044)