**Code standard**

**0712533 Phạm Xuân Thái**

**0712506 Lê Phước Vinh**

* 1. **Định chuẩn về tài liệu:**

Giống như chuẩn code,tài liệu này yêu cầu các chuẩn để đảm bảo sự rõ ràng khi nêu các quy tắc và hướng dẫn .Một số quy ước được sử dụng trong tài liệu này sẽ được đề cập thêm.

Dưới đây là mốt số quy ước được sử đụng trong tài liệu này:

|  |  |
| --- | --- |
| Màu sắc và sự nhấn mạnh | Miêu tả |
| Blue | Văn bản màu xanh cho biết một từ khóa C# hay kiểu .Net |
| **Bold** | Văn bản nhấn mạnh bổ sung để làm nổi bật nó |

* 1. **Thuật ngữ và định nghĩa:**

**Access Modifier**

-các từ khóa :public ,protected,internal và private khai báo để cho phép truy xuất đến những biến thành viên,interfaces và enum mặc định là public

**Quy ước Camel**

-một từ với kí tự đầu tiên viết thường,kí tự đầu tiên của phần từ tiếp theo viết hoa.  
 -ví dụ:customerName

**Identifier**

-lập trình viên định nghĩa danh tính cho một đối tượng đơn nhất.  
-ví dụ:publicclassMyClassNameIdentifier {....}

**Quy ước Pascal**

-từ với kí tự đầu tiên viết hoa,kí tự tiếp theo của phần từ tiếp theo viết hoa.  
 -ví dụ: CustomerName

* 1. **Flags**

Các Flags sau đây được sử dụng để giúp phân loại hoặc làm rõ một số câu lệnh:  
Một flag để xác định các quy tắc và câu lệnh chỉ áp dụng cho ngôn ngữ C # v2.0 hoặc cao hơn.

**1.4.1. Quy ước đặt tên:**

“c”=camelCase  
“P”=PascalCase  
“\_”=Tiền tố với gạch dưới \_  
 “x”=không áp dụng.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Định danh | public | protected | internal | private | Ghi chú |
| Projecs file | **P** | x | x | x | Phù hợp với hợp ngữ và namespace |
| Source file | **P** | x | x | x | Khớp với lớp chứa |
| Other files | **P** | x | x | x | Áp dụng nơi nào có thể |
| Namespace | **P** | x | x | x |  |
| Class or struct | **P** | **P** | **P** | **P** | Thêm hậu tố hoặc lớp con |
| interface | **P** | **P** | **P** | **P** | Tiền tố I |
| Generic class | **P** | **P** | **P** | **P** | Sử dụng T hay K như kiểu định danh |
| method | **P** | **P** | **P** | **P** | Sử dụng động từ để đặt tên |
| property | **P** | **P** | **P** | **P** | Không dùng tiền tố Get hay Set |
| Field | **P** | **P** | **P** | **\_c** | Chỉ dùng field Private |
| constance | **P** | **P** | **P** | **\_c** |  |
| Static field | **P** | **P** | **P** | **\_c** | Chỉ dùng field Private |
| enum | **P** | **P** | **P** | **P** | Tùy chọn là PascalCase |
| delegate | **P** | **P** | **P** | **P** |  |
| event | **P** | **P** | **P** | **P** |  |
| Inline variable | X | x | x | **c** | Tránh dùng kí tự đơn |
| Parameter | X | x | x | **c** |  |

**1.4.2.Coding Style**

|  |  |
| --- | --- |
| **Code** | **style** |
| Source file | Chỉ một namespace tren một file hoặc môt class trên một file |
| Curly braces | Đặt trên một dòng mới |
| Indention | Sử dụng tab với size 4 |
| Comments | Sử dụng // ,không dùng /\* ….\*/ |
| Variables | Một biến trên một khai báo |

**1.4.3. Ngôn ngữ sử dụng:**

|  |  |
| --- | --- |
| **Code** | **Style** |
| Kiểu dữ liệu tự nhiên | Sử dung int không dung int32 |
| Enums | Không nên thay đổi kiêu mặc định |
| Properties | Không được sử dụng tiền tố Get hay Set |
| Method | Sử dụng tối đa 7 tham số |
| Base and this | Chỉ sử dụng trong khởi tạo |
| Biểu thức điều kiện | Tránh điều kiện phức tạp |
| Câu lệnh foreach | Không định nghĩa items trong câu lệnh foreach |
| Điều kiện | Tránh các giá trị điều kiện Boolean ngược lại true hay false |
| Biệt lệ | Dùng throw ,không dung throw e |
| Events | Luôn kiểm tra null trước khi gọi |
| Depose() va close() | Gọi chúng nếu cần thiết,khai báo nơi cần dùng |

1. **Quy ước đặt tên:**
   1. **Một số quy ước thông thường:**

1. Luôn sử dụng quy ước Camel hoặc Pascal cho cách đặt tên.

2. Tránh viết hoa hết hay viết thường hết cho một tên.

3. Không sử dụng tên bắt đầu bằng số.

4. Không sử dụng số làm hậu tố để phân biệt

5. Luôn chọn tên có ý nghĩa.

6. Biến và phương thức nên mô tả rõ ràng ý nghĩa nào đó.

7. Không sử dụng ký hiệu Hungary.

Vd: strName hayiCount

8. Tránh sử dụng viết tắt trừ khi từ đó quá dài.

9. Tránh sử dụng viết tắt dài hơn 5 ký tự.

10. Bất kỳ từ viết tắt nào đều phải là phổ biến, nhiều người biết.

11. Viết hoa cho từ viết tắt có 2 chữ, dùng quy ước Pascal cho từ viết tắt dài.

12. Không sử dụng những tên dành riêng cho C# để đặt tên.

13. Tránh đặt tên gây xung đột với những namespaces, types của .NET Framework.

14. Tránh sử dụng tiền tố và hậu tố vô nghĩa, thừa thải.

Vd:

// Bad!

public enum ColorsEnum {…}

public class CVehicle {…}

public struct RectangleStruct {…}

15. Tránh thêm tên của lớp cha vào tên property.

vd: Customer.NameKHÔNGCustomer.CustomerName

16. Nên thêm tiền tố “Can”, “Is” hay “Has”cho các biến kiểu Boolean.

17. Nên sử dụng các từ Average, Count, Sum, Min, vàMaxcho các phương thức tính toán.

18. Khi định nghĩa một root namespace thì sử dụng tên Sản phẩm, Công ty, hay tên Developer làm gốc.

Vd:LanceHunt.StringUtilities

* 1. **Cách đặt tên và ký hiệu:**

|  |  |
| --- | --- |
|  | Quy ước đặt tên |
| Project File | Quy ước Pascal  Luôn phải đặt tương xứng giữa Assembly Name & Root Namespace  Vd:  Example:  LanceHunt.Web.csproj -> LanceHunt.Web.dll ->namespace  LanceHunt.Web |
| Source File | Quy ước Pascal  Luôn phải tương xứng giữa tên Class và tên file.  Tránh sử dụng hơn 1 Class, Enum(toàn cục), Delegate(toàn cục) trong 1 file.  Vd:  MyClass.cs =>public class MyClass  {…} |
| Resource hay  Embedded File | Nên dùng quy ước Pascal  Sử dụng tên mô tả được nội dung file |
| Namespace | Quy ước Pascal  Nên đặt tương xứng giữa tên Project/Assembly  Vd:  namespace LanceHunt.Web  {…} |
| Class hay Struct | Quy ước Pascal  Sử dụng danh từ để đặt tên Class  Vd:  private class MyClass  {…}  internal class SpecializedAttribute : Attribute  {…}  public class CustomerCollection : CollectionBase  {…}  public class CustomEventArgs : EventArgs  {…}  private struct ApplicationSettings  {…} |
| Interface | Quy ước Pascal  Luôn sử dụng tiền tố “I”  Vd:  interface ICustomer  {…} |
| Generic Class  &  Generic  Parameter Type  [C#v2+] | Luôn sử dụng một chữ viết hoa, như T hay K  Vd:  public class FifoStack<T>  {  public void Push(<T> obj)  {…}  public<T> Pop()  {…}  } |
| Method | Quy ước Pascal  Nên sử dụng động từ.  Vd:  public void Execute() {…}  private string GetAssemblyVersion(Assembly target) {…} |
| Property | Quy ước Pascal  Tên nên đại diện cho cái nó trả về. Không dùng tiền tố với “Get” hay “Set”  Vd:  public string Name  {  get{…}  set{…}  } |
| Field  (Public, Protected,  hay Internal) | Quy ước Pascal  Vd:  public string Name;  protected IList InnerList; |
| Field (Private) | Quy ước Camel, sử dụng tiền tố “\_”  Vd:  private string \_name; |
| Constant hay  Static Field | Tương tự như Field |
| Enum | Quy ước Pascal  Vd:  public enum CustomerTypes  {  Consumer,  Commercial  } |
| Delegate hay Event | Tương tự như Field  Vd:  public event EventHandler LoadPlugin; |
| Variable (inline) | Quy ước Camel  Tránh sử dụng một ký tự như “x” hay “y” ngoại trừ trong vòng lặp FOR  Tránh sử dụng những tên kiểu đếm như: text1, text2, text3 |
| Parament | Quy ước Camel  Vd:  public void Execute(string commandText, int iterations)  {…} |

1. **Phong cách lập trình:**
   1. **Định dạng:**

1. Không bao giờ khai bào hơn 1 không gian tên cho mỗi tập tin.

2. Tránh đặt nhiều lớp vào 1 tập tin đơn lẻ.

3. Luôn đặt dấu ngoặc móc({ và }) trên dòng mới

4. Luôn sử dụng dấu ngoặc móc ({ và }) trong câu lệnh điều kiện.

5. Luôn sử dụng 1 Tab và cỡ thụt vào đầu dòng là 4.

6. Trình bày mỗi biến độc lập-không cùng câu lệnh.

7. Đặt câu lệnh namespace ‘using’ tạo đầu mỗi file.Group .NET namespaces above custom namespaces.

8. Thi hành lớp bên trong nhóm theo kiểu thứ tự sau:

a. Biến thành viên

b. Khởi tạo và hoàn thành

c. Lồng Enums,structs và Classes

d. Thuộc tính

e. Phương thức

9. declarationswithintypegroupsbaseduponaccessmodifierandvisibility:

a.Public

b.Protected

c.Internal

d.Private

10. Thực hiện giao diện độc lập bằng cách sử dụng câu lệnh #region.

11. Nối tên thư mục tới không gian tên cho file nguồn trong thư mục con.

12. Tất cả mã ghi lùi vào trong phần trong dấu ngoặc móc.

13. Sử dụng khoảng trắng (CR/LF,Tabs,v.v.v.v.) tùy nghi để riêng biệt và tổ chức mã.

14.Chỉ khai báo thuộc tính liên quan trên một dòng, nếu không ngăn xếp xếp mỗi thuộc tính khi khai báo riêng biệt.

VD:

//Bad!

[Attrbute1,Attrbute2,Attrbute3]

publicclassMyClass

{…}

//Good!

[Attrbute1,RelatedAttribute2]

[Attrbute3]

[Attrbute4]

publicclassMyClass

{…}

15. Khai báo tập hợp attribute khai báo ở 1 dòng riêng biệt.

16. Khai báo kiểu attribute khai báo ở 1 dòng riêng biệt.

17. Khai báo phương thức attribute khai báo ở 1 dòng riêng biệt.

18. Khai báo thành viên attribute khai báo ở 1 dòng riêng biệt.

19. Khai báo tham số attribute khai báo ở 1 dòng riêng biệt.

20. Nếu còn sự nghi gờ,luôn thiên về tính rõ ràng và nhất quán.

* 1. **Chú thích code:**

21.Tất cả chú thích nên được viết trong cùng ngôn ngữ, đúng văn phạm, và chứa dấu chấm thích hợp.

22.Dùng / / or / / / nhưng không bao giờ dùng / \* \* /.

23.Không " flowerbox " chú thích khối.

VD:

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//Comment Block

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

24. Chú thích trực tiếp trên dòng để giải thích giả định, vấn đề , và thuật toán .

25. Không chú thcíh trực tiếp để giải thcíh các code hiển nhiên.

26. Chỉ dùng chú thích “fix the code” cho đoạn code hỏng-Nếu không xóa hoắc viết lại.

27.Bao gồm chú thích dùng từ khóa Task-List để cho phép chú thích - lọc.

VD :

//TODO:PlaceDatabaseCodeHere

//UNDONE:RemovedP\InvokeCalldueto errors

//HACK:Temporaryfixuntilableto refactor

28.Luôn áp dụng chú thích khối C# để khai báo public,protected và internal

29.Chỉ dùng C# chú thích - khối cho tài liệu API.

30.luôn chú thích include<summary>.Include<param>,<return>,and chú thích <exception>phần nơi thích hợp.

31. Include<seecref=””/>and<seeAlsocref=””/>ở nơi thích hợp

32.Luôn thêm thẻ CDATA tới chú thích chứa mã và đánh dấu nhúng khác để tránh mã hóa vấn đề.

VD:

///<example>

///Addthefollowingkeytothe“appSettings”sectionofyourconfig:

///<code><![CDATA[

/// <configuration>

/// <appSettings>

/// <addkey=”mySetting”value=”myValue”/>

/// </appSettings>

/// </configuration>

///]]></code>

///</example>

1. **Ngôn ngữ sử dụng:**
   1. **Tổng quát:**

1. Không quên truy cập từ bổ nghĩa. Trình bày rõ ràng tất cả các mã nhận biết từ bổ nghĩa thích hợp thay vì để mặc định.

VD:

// Bad!

Void WriteEvent(string message)

{…}

// Good!

private Void WriteEvent(string message)

{…}

2. Không sử dụng phiên bản (‘1.0.\*’) mặc định. Tăng giá trị AssemblyVersionAttribute bằng tay.

3. Thiết lập ComVisibleAttribute để false với mọi cụm.

4. Chỉ chọn lọc cho phép ComVisibleAttribute cho các lớp cá nhân khi cần.

VD:

[assembly: ComVisible(false)]

[ComVisible(true)]

public MyClass

{…}

5. Phân tích các lớp chứa khối mã không an toàn thành các thành phần riêng biệt.

6. Tránh tham chiếu hỗ tương giữa các cụm.

* 1. **Giá trị và kiểu:**

7. Cố gắng khởi tạo biến tại nơi khai báo chúng.

8. Luôn chọn kiểu dữ liệu, danh sách, hoặc đối tượng bắt buộc một cách đơn giản nhất.

9. Luôn sử dụng biệt danh kiểu dữ liệu có sẵn trong C#, không phải hệ thống .NET dùng chung (CTS).

VD:

shortNOT System.int16

intNOT System.int32

longNOT System.int64

stringNOT System.string

10.Chỉ khai báo biến thành viên là private,dùng thuộc tính để cung cấp sự truy cập vào chúng với public.protected,internal.

11. Hãy cố gắng dùng int làm bất cứ giá trị số nào phần sẽ đặt vừa kiểu dữ liệu int - thậm chí cho số không âm.

12. Chỉ sử dụng khai bào biến kiểu long cho giá trị quá lớn với kiểu int.

13. Hãy cố gắng sử dụng kiểu double cho các phân số để đảm bảo độ chính xác số thập phân trong tính toán.

14.Chỉ dùng kiểu float cho các phân số không đặt vừa kiểu double hay decimal.

15. Tránh sử dụng kiểu float trừ phi bạn hiểu đầy đủ ngụ ý trên mọi tính toán.

16. Hãy cố gắng sử dụng kiểu decimal khi các phân số phải làm tròn đến độ chính xác cố định cho tính toán.

17. Tránh sử dụng sbyte,short, uint, và ulong trừ phi nó dùng để interop ( P / gọi ) với thư viện có sẵn.

18. Tránh xác định kiểu cho 1 enum - sử dụng mặc định là int trừ phi bạn có cần có 1 sự rõ ràng thì dùng long ( rất hiếm ).

19.Tránh sử dụng chữ số trực tiếp .Thay vào đó, sử dụng constant(hằng số) hoặc enum.

20.Tránh khai báo trực tiếp 1 chuỗi.thay vì vậy dùng Resources, Constants, Configuration Files, Registry hay các nguồn dữ liệu khác.

21. Khai báo biến readonly hay static readonly thay vì hằng số cho các kiểu phức tạp.

22.Chí khai báo constant(hằng) cho những kiểu đơn giản.

23. Tránh chuyển đổi kiểu dữ liệu trực tiếp.thay vì vậy dùng toán tử as và ktra cho null.

VD:

object dataObject = LoadData();

DataSet ds = dataObject as DataSet;

if(ds != null)

{…}

24.Luôn luôn ưa thích các kiểu C# chung hơn là chuẩn hoặc kiểu…..(C#v2+)

25.Luôn khởi tạo kiểu mảng tham chiếu rõ ràng cho vòng lặp.

26. Tránh boxingandunboxingvaluetypes.

VD:

int count = 1;

object refCount = count; // Implicitly boxed.

int newCount = (int)refCount; // Explicitly unboxed.

27. Giá trịdấu chấm động nên bao gồm ít nhất một chữ số trước chữ số thập phân và một đằng sau.

Vd:totalPercent=0.05

28.Hãy cố gắng dùng tiền tố " @ "cho chuỗi chữ sai thay vì thoát chuỗi.

29. Nên dùng String.Format() hayStringBuilderhơn là chuỗi ghép nối.

30. Không bao giờ ghép nối chuỗi bên trong vòng lặp.

31. Không so sánh chuỗi với String.Empty hay “”để kiểm tra chuỗi rỗng. Thay vào đó ta sẽ dùng String.Length == 0.

32. Sử dụng String.Compare()

VD:

// Bad!

int id = -1;

string name = “lance hunt”;

for(int i=0; i < customerList.Count; i++)

{

if(customerList[i].Name.ToLower()== name)

{

id = customerList[i].ID;

}

}

// Good!

int id = -1;

string name = “lance hunt”;

for(int i=0; i < customerList.Count; i++)

{

// The “ignoreCase = true” argument performs a

// case-insensitive compare without new allocation.

if(**String.Compare**(customerList[i].Name, name, true)== 0)

{

id = customerList[i].ID;

}

}

* 1. **Kiểm soát luồng**

33. Tránh cách gọi phương thức trong một biểu thức điều kiện.

34. Tránh tạo các phương thức đệ quy. Thay vào đó sử dụng vòng lặp hay vòng lặp lồng nhau.

35. Tránh sử dụng foreach để lặp trên các bộ loại dữ liệu bất biến. **Ví dụ**: mảng chuỗi.

36. Không chỉnh sửa các mục được liệt kê trong câu lệnh (statement) foreach.

37. Chỉ sử dụng các toán tử có điều kiện bậc ba trong các điều kiện bình thường. Tránh các toán tử bậc ba phức tạp và đa hợp.

VD: int result = isValid ? 9 : 4;

38. Tránh đánh giá các điều kiện Boolean là true hay flase.

VD:

//Bad!

if (isValid == true)

{…}

//Good!

if (isValid)

{…}

39. Tránh phép gán bên trong một biểu thức điều kiện.

VD:if ((**i=2**) == 2) {…}

40. Tránh các biểu thức đa điều kiện – sử dụng các biến Boolean để chia thành nhiều biểu thức quản lý.

VD:

// Bad!

if (((value > \_highScore) && (value != \_highScore)) && (value < \_maxScore))

{…}

// Good!

isHighScore = (value >= \_highScore);

isTiedHigh = (value == \_highScore);

isValid = (value < \_maxValue);

if ((isHighScore && ! isTiedHigh) && isValid)

{…}

41. Tránh kiểm tra các biến Boolean rõ ràng trong các biểu thức điều kiện.

VD:

//Bad!

if (isValid == true)

(…)

//Good!

if (isValid)

(…)

42. Chỉ sử dụng switch/case cho các phép tính đơn giản với các điều kiện logic song song.

43. Ưu tiên sử dụng lồng if/else hơn switch/case cho các chuỗi điều kiện ngắn và các điều kiện phức tạp.

44. Ưu tiên đa hình hơn switch/case để đóng gói và đại diện cho các phép toán phức tạp.

* 1. **Exceptions**

45. Không nên sử dụng các blocks try/catch để kiểm soát các luồng.

46. Chỉ catch các ngoại lệ bạn có thể xử lý.

47. Không bao giờ khai báo 1 catch block rỗng.

48. Tránh lồng 1 try/catch bên trong 1 catch block.

49. Luôn luôn catch các ngoại lệ có nguồn gốc từ bộ lọc ngoại lệ.

50. Sắp xếp bộ lọc ngoại lệ từ nguồn gốc loại ngoại lệ.

51. Tránh trả lại (re-throwing) một ngoại lệ. Thay vào đó dùng phương pháp nổi bọt.

52. Nếu phải trả lại (re-throwing) một ngoại lệ, bảo toàn các thủ tục gọi ban đầu của stack bỏ qua đối số ngoại lệ từ throw statement.

VD:

//Bad!

catch(Exception ex)

{

Log(ex);

throw ex;

}

//Good!

catch(Exception)

{

Log(ex);

throw;

}

53. Chỉ sử dụng block cuối cùng để giải phóng tài nguyên từ một lệnh try.

54. Luôn sử dụng sự xác nhận để tránh các ngoại lệ.

VD:

//Bad!

try

{

Conn.Close();

}

catch(Exception ex)

{

//handle exception if already closed!

}

//Good!

if(conn.State != ConnectionState.Closed)

{

conn.Close();

}

55. Luôn thiết lập thuộc tính innerException vào thrown ngoại lệ, như vậy các thủ tục chain & call stack luôn được duy trì.

56. Tránh xác định các ngoại lệ tùy chỉnh. Thay vào đó sử dụng các ngoại lệ có sẵn.

57. Khi buộc phải sử dụng ngoại lệ tùy chỉnh:

a. Luôn luôn xuất phát từ ngoại lệ (Exception) không phải từ ApplicationException.

b. Luôn luôn thêm hậu tố “exception” vào tên các lớp ngoại lệ

c. Luôn thêm SerializationAttribute vào các lớp ngoại lệ.

d. Luôn thực hiện các tiêu chuẩn “Exception Constructor Pattern”:

public MyCustomException ();

public MyCustomException (string message);

public MyCustomException (string message, Exception innerException);

e. Luôn thực hiệnconstructor deserialization:

protected MyCustomException(SerializationInfo info, StreamingContext contxt);

58. Luôn đặt giá trị Hresult thích hợp cho các ngoại lệ tùy chỉnh:

(Lưu **ý:**ApplicationExceptionHResult = -2146232832)

59. Khi định nghĩa các lớp ngoại lệ tùy chỉnh mà nó chứa các thuộc tính bổ sung:

a. Luôn ghi đè lên các thuộc tính Messenger, phương thức ToString(), và implicit operator string bằng các giá trị tùy chỉnh.

b. Luôn thay đổi deserialization constructor để lấy lại các giá trị của thuộc tính tùy chỉnh.

c. Luôn ghi đè lên phương thức GetObjectData(…) để thêm các thuộc tính tùy chỉnh vào serialization collection.

VD:

public override void GetObjectData(SerializationInfo info, StreamingContext context)

{

base.GetObjectData (info, context);

info.AddValue("MyValue", \_myValue);

}

* 1. **Events, Delegates, & Threading**

60. Luôn kiểm tra giá trị null của Event/delegate trước khi gọi.

61. Dùng mặc định EventHandler và EventArgs cho hầu hết các event đơn giản.

62. Luôn lấy 1 lớp EventArgs tùy chỉnh để cung cấp dữ liệu bổ sung.

63. Sử dụng lớp CancelEventArgs để cho phép người đăng ký event kiểm soát các event.

64. Luôn sử dụng từ khóa “lock” thay cho các loại Monitor.

65. Chỉ khóa trên 1 đối tượng private hoặc private static

**Ví dụ:**lock (myVariable);

66. Tránh khóa một kiểu dữ liệu.

VD: lock(typeof(MyClass));

67. Tránh khóa đối tượng vừa mới sử dụng.

VD: lock(this);

* 1. **Sự sắp xếp đối tượng**

68. Luôn thể hiện các kiểu dữ liệu một cách rõ ràng trong một namespace. Không sử dụng namespace mặc định “{global}”.

69. Tránh lạm dụng thay đổi mô hình public. Điển hình là ít hơn 10% của kiểu dữ liệu và thành phần của bạn là phần trong API đại trà, trừ khi bạn đang viết một thư viện lớp.

70. Cân nhắc việc sử dụng mô hình internal hay private cho các kiểu và thành phần trừ khi bạn có ý định hỗ trợ chúng như là môt phần của API chung.

71. Không bao giờ sử dụng thay đổi mô hình của protected trong những lớp sealed trừ khi bỏ qua một thành phần của protected trong một kiểu dữ liệu thừa kế.

72. Tránh những cách thể hiện với nhiều hơn 5 tham số. Xem xét việc cải tiến mã nguồn này.

73. Cố gắng thay những cụm tham số lớn ( >5 ) bằng một hoặc nhiều hơn class hay struct tham số đặc biệt khi nó được sử dụng trong những dấu hiệu phong phú.

74. Không sử dụng từ khóa “new” với một phương thức và những thể hiện đặc trưng để giấu những thành phần của một kiểu dữ liệu được định nghĩa.

75. Chỉ sử dụng từ khóa “base” khi gọi xây dựng một lớp nền hoặc bổ sung cơ bản với một sự lạm dụng.

76. Cân nhắc việc sử dụng phương thức quá tải thay cho những thuộc tính params (nhưng hãy cẩn thận đừng bẻ gãy CLS Compliance của API của bạn).

77. Luôn luôn hiệu lực hóa bảng liệt kê giá trị biến hoặc tham số trước khi sử dụng nó. Cúng có thể chứa bất kì giá trị nào mà bảng kiểu dữ liệu nền (mặc định int) hỗ trợ.

VD:

public void Test(BookCategory cat)

{

if (Enum.IsDefined(typeof(BookCategory), cat))

{…}

}

78. Cân nhắc việc bỏ qua Equals()trên một struct.

79. Luôn bỏ qua Equality Operator (==)khi đã bỏ qua phương thức Equals().

80. Luông bỏ qua String Implicit Operator khi bỏ qua phương thức ToString().

81. Luôn gọi Close() hoặc Dispose() trên những lớp có đưa ra nó.

82. Bao phủ sự cụ thể hóa của những đối tượng IDisposablevới một   
”using” trình bày nhằm bảo đãm rằng Dispose() sẽ không được gọi một cách tự động.

VD:

using(SqlConnection cn = new SqlConnection(\_connectionString))

{…}

83. Luôn thực thi giao diện và mô hình IDisposable trên những lớp tham chiếu đến những tài nguyên bên ngoài.

VD:

public void Dispose()

{

Dispose(true);

GC.SuppressFinalize(this);

}

protected virtual void Dispose(bool disposing)

{

if (disposing)

{

// Free other state (managed objects).

}

// Free your own state (unmanaged objects).

// Set large fields to null.

}

// C# finalizer. (optional)

~Base()

{

// Simply call Dispose(false).

Dispose (false);

}

84. Tránh thực thi một sự kết thúc.

Không bao giờ định nghĩa phương thức Finalize() như là một kết thúc. Thay thế bằng những cú pháp định nghĩa sẵn bởi C#.

VD:

// Good

~MyClass {…}

// Bad

void Finalize(){…}

1. **Mô hình đối tượng và thiết kế API**

1.Luôn ưu tiên kết khối hơn là thừa kế.

2. Tránh “Tổng quát hóa sớm”. Chỉ tạo ra những sự trừu tượng hóa khi ý định đã được thông suốt.

3. Làm những thứ đơn giản nhưng có thể hoạt động, sau đó cải tiến mã nguồn khi cần thiết.

4. Luôn tạo trạng thái trong suốt đối với API của đối tượng.

5. Tránh những tác động bên ngoài không mong đợi khi thực thi đặc tính, phương thức và cấu trúc.

6. Luôn tách lớp trình diễn ra khỏi logic nghiệp vụ.

7. Luôn ưu tiên những giao diện hơn là lớp trừu tượng.

8. Cố gắng đưa vào tên của những phần thiết kế như “Bridge”, “Adapter”, hay “Factory” như là hậu tố của tên lớp nơi nào thích hợp.

9. Chỉ tạo những thành phần ảo nếu chúng được tạo ra và kiểm tra nếu có thể mở rộng.

10. Thường xuyên cải tiến mã nguồn!