CÁC QUY ƯỚC VIẾT MÃ C#

* Chúng tạo ra giao diện nhất quán cho mã để người đọc có thể tập trung vào nội dung chứ không phải bố cục.
* Chúng cho phép người đọc hiểu mã nhanh hơn bằng cách đưa ra các giả định dựa trên kinh nghiệm trước đó.
* Chúng tạo điều kiện cho việc sao chép, thay đổi và duy trì mã.

1. **Quy ước đặt tên:**
   * **Pascal case:**

-Viết hoa tất cả chữ cái đầu

public class DataService

{

}

public record PhysicalAddress(

string Street,

string City,

string StateOrProvince,

string ZipCode);

public struct ValueCoordinate

{

}

-Về Interface thêm chữ I ngoài tiền tố đầu tiên

public interface IWorkerQueue

{

}

-Khi đặt tên cho public các thành viên của các loại như: fields, properties, events, methods, and local functions, đều dùng Pascal case.VD:

public class ExampleEvents

{

// A public field, these should be used sparingly

public bool IsValid;

// An init-only property

public IWorkerQueue WorkerQueue { get; init; }

// An event

public event Action EventProcessing;

// Method

public void StartEventProcessing()

{

// Local function

static int CountQueueItems() => WorkerQueue.Count;

// ...

}

}

* + **camelCase:**

-Giống như cách viết của nó, từ đầu tiên viết thường, các từ tiếp theo viết hoa chữ cái đầu.VD: $thisIsMyVariable.

-Khi đặt tên cho trường private or internal nên đặt tiền tố cho chúng bằng \_ VD:

public class DataService

{

private IWorkerQueue \_workerQueue;

}

-Khi làm việc với trường static, private, internal nên sử dụng s\_ tiền tố và đối vs luồng tĩnh thì sử dụng t\_ VD:

public class DataService

{

private static IWorkerQueue s\_workerQueue;

[ThreadStatic]

private static TimeSpan t\_timeSpan;

}

-Khi viết với các tham số của phương thức

public T SomeMethod<T>(int someNumber, bool isValid)

{

}

1. **Quy ước đặt tên bổ sung:**

- Hãy sử dụng các tiêu chuẩn về không gian tên. Nếu bạn biết rằng một không gian tên được nhập theo mặc định trong một dự án, thì bạn không cần phải xác định đầy đủ các tên từ không gian tên đó. Tên đủ điều kiện có thể bị ngắt sau dấu chấm (.) nếu chúng quá dài cho một dòng VD:

var currentPerformanceCounterCategory = new System.Diagnostics.

PerformanceCounterCategory();

1. **Quy ước về bố cục:**
   * Sử dụng cài đặt Code Editor mặc định (thụt lề thông minh, thụt lề bốn ký tự, tab được lưu dưới dạng khoảng trắng).
   * Chỉ viết một câu trên mỗi dòng.
   * Chỉ viết một khai báo trên mỗi dòng.
   * Nếu các dòng tiếp tục không được thụt lề tự động, hãy thụt lề cho chúng một điểm dừng tab (bốn dấu cách).
   * Thêm ít nhất một dòng trống giữa định nghĩa phương thức và định nghĩa thuộc tính.
   * Sử dụng dấu ngoặc đơn để làm rõ các mệnh đề trong một biểu thức, VD:

if ((val1 > val2) && (val1 > val3))

{

// Take appropriate action.

}

1. Đặt các lệnh sử dụng bên ngoài khai báo không gian tên:

- Khi 1 lệnh using nằm ngoài khai báo không gian tên, thì không gian tên đã nhập đó là tên đủ điều kiện của nó. Điều đó rõ hơn khi sử dụng lệnh using nằm trong ko gian tên, nó có thể liên quan đến không gian tên đó hoặc tên đủ điều kiện VD:

using Azure;

namespace CoolStuff.AwesomeFeature

{

public class Awesome

{

public void Stuff()

{

WaitUntil wait = WaitUntil.Completed;

…

}

}

- Giả sử có một tham chiếu (trực tiếp hoặc gián tiếp) đến lớp WaitUntil .

namespace CoolStuff.AwesomeFeature

{

using Azure;

public class Awesome

{

public void Stuff()

{

WaitUntil wait = WaitUntil.Completed;

}}}

-Biên dịch thành 2 lỗi:

- error CS0246: The type or namespace name 'WaitUntil' could not be found (are you missing a using directive or an assembly reference?)

- error CS0103: The name 'WaitUntil' does not exist in the current context

-Một trong những phần phụ thuộc đã giới thiệu lớp này trong một không gian tên rồi kết thúc bằng .Azure:

namespace CoolStuff.Azure

{

public class SecretsManagement

{

public string FetchFromKeyVault(string vaultId, string secretId) { return null; }

}

}

1. **Kiểu dữ liệu chuỗi:**

-Sử dụng string interpolation $ để nối các chuỗi ngắn,VD:

string displayName = $"{nameList[n].LastName}, {nameList[n].FirstName}";

-Để nối các chuỗi trong vòng lặp, đặc biệt khi đang làm việc với số lượng lớn văn bản, hãy sử dụng đối tương StringBuider.

var phrase = "lalalalalalalalalalalalalalalalalalalalalalalalalalalalalala";

var manyPhrases = new StringBuilder();

for (var i = 0; i < 10000; i++)

{

manyPhrases.Append(phrase);

1. **Các biến cục bộ được gõ ngầm**:

-Sử dụng implicit typing cho các biến cục bộ khi loại biến rõ ràng từ phia bên phải của phép gán hoặc khi loại chính xác không quan trọngVD:

var var1 = "This is clearly a string.";

var var2 = 27;

-Không sử dụng var khi loại không rõ ràng. Một loại biến được coi là rõ ràng nếu đó là 1 toán tử new hoặc 1 kiểu rõ ràng:

int var3 = Convert.ToInt32(Console.ReadLine());

int var4 = ExampleClass.ResultSoFar();

-Đừng dựa vào tên biến để chỉ định loại biến .Nó có thể không đúng

var inputInt = Console.ReadLine();

Console.WriteLine(inputInt);

-Tránh sử dụng var thay cho dynamic.

-Sử dụng gõ ẩn để xác định loại biến vòng lặp trong for :

var phrase = "lalalalalalalalalalalalalalalalalalalalalalalalalalalalalala";

var manyPhrases = new StringBuilder();

for (var i = 0; i < 10000; i++)

{

manyPhrases.Append(phrase);

}

//Console.WriteLine("tra" + manyPhrases);

-Không sử dụng cách nhập ẩn để xác định loại biến vòng lặp trong foreach

VD cách gõ rõ ràng trong 1 câu lệnh forech

foreach (char ch in laugh)

{

if (ch == 'h')

Console.Write("H");

else

Console.Write(ch);

}

Console.WriteLine();

-Kiểu dữ liệu 0 dấu: sử dụng int thay vì các loại không dấu. Sử dụng int phổ biến trong toàn bộ C# và việc tương tác vs các thư viện khác sẽ dễ dàng hơn

1. **Array:**

-Sử dụng cú pháp ngắn gọn khi khởi tạo mảng trên dòng khai báoVD:

string[] vowels1 = { "a", "e", "i", "o", "u" };

-Nếu khởi tạo rõ ràng thì sử dụng var:

var vowels2 = new string[] { "a", "e", "i", "o", "u" };

1. **Try-catch and using các câu lệnh trong xử lí ngoại lệ**

- Sử dụng câu lệnh try-catch để xử lý hầu hết các ngoại lệ.

static string GetValueFromArray(string[] array, int index)

{

try

{

return array[index];

}

catch (System.IndexOutOfRangeException ex)

{

Console.WriteLine("Index is out of range: {0}", index);

throw;

}

}

- Nếu bạn có 1 câu lệnh thử cuối cùng trong đó mã duy nhất trong khối finally là lệnh gọi phương thức Dispose, thì hãy sử dụng 1 câu lệnh using để thay thế.

- Trong ví dụ sau, try-finally câu lệnh chỉ gọi Disposetrong finally khối.

Font font1 = new Font("Arial", 10.0f);

try

{

byte charset = font1.GdiCharSet;

}

finally

{

if (font1 != null)

{

((IDisposable)font1).Dispose();}}

-Có thể làm điều tương tự vs using statement.

using (Font font2 = new Font("Arial", 10.0f))

{

byte charset2 = font2.GdiCharSet;

}

-Sử dụng cú pháp using new không yêu cầu dấu ngoặc nhọn

using Font font3 = new Font("Arial", 10.0f);

byte charset3 = font3.GdiCharSet;

1. **&& and || operators**:

-Sử dụng khi thực hiện phép so sánh:

Console.Write("Enter a dividend: ");

int dividend = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter a divisor: ");

int divisor = Convert.ToInt32(Console.ReadLine());

if ((divisor != 0) && (dividend / divisor > 0))

{

Console.WriteLine("Quotient: {0}", dividend / divisor);

}

else

{

Console.WriteLine("Attempted division by 0 ends up here.");}

1. **new operator:**

-Sử dụng 1 trong các hình thức khởi tạo đối tượng ngắn gọn, :

var instance1 = new ExampleClass();

ExampleClass instance2 = new();

ExampleClass instance2 = new ExampleClass();

- Sử dụng bộ khởi tạo đối tượng để đơn giản hóa việc tạo đối tượng, như minh họa trong ví dụ sau.

var instance3 = new ExampleClass { Name = "Desktop", ID = 37414,

Location = "Redmond", Age = 2.3 };

-Trường hợp sau không sử dụng bộ khởi tạo

var instance4 = new ExampleClass();

instance4.Name = "Desktop";

instance4.ID = 37414;

instance4.Location = "Redmond";

instance4.Age = 2.3;

1. **Truy vấn LINQ:**

- Sử dụng tên có ý nghĩa cho các biến truy vấn. Ví dụ sau sử dụng seattleCustomerscho khách hàng ở Seattle.

var seattleCustomers = from customer in customers

where customer.City == "Seattle"

select customer.Name;

- Sử dụng bí danh để đảm bảo rằng tên thuộc tính của các loại ẩn danh được viết hoa chính xác, sử dụng cách viết hoa Pascal.

var localDistributors =

from customer in customers

join distributor in distributors on customer.City equals distributor.City

select new { Customer = customer, Distributor = distributor };

- Đổi tên thuộc tính khi tên thuộc tính trong kết quả không rõ ràng. Ví dụ: nếu truy vấn của bạn trả về tên khách hàng và ID nhà phân phối, thay vì để nguyên như vậy, hãy đổi tên chúng để làm rõ đó Name là tên của khách hàng và là ID của nhà phânphối.IDNameID

var localDistributors2 =

from customer in customers

join distributor in distributors on customer.City equals distributor.City

select new { CustomerName = customer.Name, DistributorID = distributor.ID };

- Sử dụng cách gõ ẩn trong khai báo biến truy vấn và biến phạm vi.

var seattleCustomers = from customer in customers

where customer.City == "Seattle"

select customer.Name;

- Căn chỉnh các mệnh đề truy vấn theo frommệnh đề, như được hiển thị trong các ví dụ trước.

-Sử dụng wherecác mệnh đề trước các mệnh đề truy vấn khác để đảm bảo rằng các mệnh đề truy vấn sau này hoạt động trên tập hợp dữ liệu được lọc, rút gọn.

var seattleCustomers2 = from customer in customers

where customer.City == "Seattle"

orderby customer.Name

select customer;

- Sử dụng nhiều from mệnh đề thay vì một join mệnh đề để truy cập các bộ sưu tập bên trong. Ví dụ: một tập hợp Studentcác đối tượng có thể chứa một tập hợp các điểm kiểm tra. Khi truy vấn sau đây được thực thi, nó sẽ trả về từng điểm trên 90, cùng với họ của học sinh nhận được điểm.

var scoreQuery = from student in students

from score in student.Scores!

where score > 90

select new { Last = student.LastName, score };