Styl Kodowania w C#

1 O Stylu Kodowania C#

Ten dokument można czytać jako przewodnik pisania silnych i niezawodnych programów. Skupia się on na programach napisanych w C #, ale wiele reguł i zasad jest użytecznych nawet jeśli programujesz w innym języku programowania.

2 Organizacja Pliku

2.1 Pliki źródłowe C#

Utrzymuj krótkie klasy/plik, nie przekraczaj 2000 LOC, podziel kod, co rozjaśni strukturę. Wstawiaj każdą klasę w oddzielnym pliku, a plik nazwij tak jak nazywa się klasa (oczywiście z rozszerzeniem . cs). Taka konwencja uczyni wiele rzeczy łatwiejszymi.

2.2 Układ katalogu

Stwórz katalog dla każdej przestrzeni nazw. (Dla MyProject.TestSuite.TestTier użyj MyProject/ TestSuite/TestTier jako ścieżki dostępu, nie używaj nazwy przestrzeni nazw do z kropkami.) To czyni łatwiejszym odwzorowanie przestrzeni nazw do układu katalogu.

3 Wcięcia

3.1 Zawijanie linii

Kiedy wyrażenie nie kończy się w pojedycznej linii, przełam ją zgodnie z poniższymi ogólnymi zasadami:

* Przełam po przecinku.
* Przełam po operatorze.
* Preferuj przełamanie wysokopoziomowe do nisko poziomowego.
* Wyrównuj nowa linię z początkiem wyrażenia przy tym samym poziom w poprzedniej linii

Przykład przełamania:

```
longMethodCall(exprl, expr2,
               expr3, expr4, expr5);
```

Przykłady przełamania wyrażenia arytmetycznego:

PREFEROWANE:

```
var = a * b / ( c - g + f ) +4 * z;
```

UNIKAJ ZŁEGO STYLU:

```
var = a * b / ( c -
      g +f) + 4 *
      z;
```

Pierwszy przykład jest preferowany, ponieważ przełamanie występuje na zewnątrz nawiasów wyrażenia (zasada wysokiego poziomu). Zauważ ,że aby wcinać z tabulatorem do poziomu wcięcia a potem ze spacją do przełamania pozycji w naszym przykładzie wyglądałoby to tak:

```
>   var = a * b / ( c - g + f ) +
>   .............4 * z;
```

Gdzie '>' to znaki tabulatora a '.' są spacjami, (spacje po znaku tabulatora są wcięciami z tabulacją). Dobra praktyką kodowania jest tworzenie widzialnych znaków tabulacji i spacji w edytorze którego używasz.

3.2 Białe spacje

Standard wcięcia używający spacji nigdy nie był osiągnięty. Niektórzy ludzie lubią 2 spacje, niektórzy lubią 4 spacje a inni 8, lub nawet więcej spacji. Lepiej używać tabulatorów. Znaki tabulacji mają pewne zalety:

* Każdy może ustawić swój własny preferowany poziom wcięć
* Jest tylko 1 znak a nie 2, 4, 8 … dlatego też zredukuje się wpisywanie
* Jeśli chcesz zwiększyć wcięcie (lub zmniejszyć) zaznacz jeden blok i zwiększ poziom wcięcia Tab lub z Shift-Tab aby zmniejszyć wcięcie. Jest to prawda prawie w każdym edytorze tekstu.

Tu definiujemy Tab jako standardowy znak wcięcia.

Nie używaj spacji dla wcięć – użyj tabulacji!

4 Komentarze

4.1 Blok Komentarzy

Blok komentarzy powinien być zazwyczaj unikany. Dla opisów używaj komentarzy /// dany w standardowym opisie C #. Kiedy życzysz sobie używać bloku kometarzy powinieneś użyć następujacego stylu :

```
/*  Line 1
 * Line 2
 * Line 3
 */
```

Takie ustawienie bloku uczyni kod łatwiejszy do odczytu (dla człowieka). Alternatywnie możesz użyć starego stylu C dla pojedynczej linii komentarza, chociaż nie jest zalecany. W przypadku użycia tego stylu, linia przerwania powinien nastąpić kometarz, ponieważ trudno jest zobaczyć kod przedstawiony po kometarzu w tej samej linii:

```
/*  blah blah blah  */
```

Blok komentarzy może być użyteczny w rzadkich przypadkach. Generalnie blok komentarza jest użyteczny przy komentowaniu dużych sekcji kodu.

4.2 Pojedyncza linia komentarza

Powinieneś używać stylu komentarza // do "komentowania" kodu (SharpDevelop ma klucz dla niego, Alt+/) . Może być używany do komentowania sekcji kodu. Pojedyncza linia komentarza musi być wcięta do poziomu wcięcia kiedy są używane dla dokumentowania kodu. Zakomentowany kod powinien być zakomentowany w pierwszej linii aby poprawić zakomentowany kod. Zasada kciuka mówi,że długość komentarza nie powinna przekraczać długości objaśnianego kodu zbytnio, ponieważ wystąpić może zbytnia komplikacja w przypadku “odpluskwiania” kodu.

4.3 Komentarze dokumentacji

W .Net Framework, Microsoft wprowadził system generowania dokumentacji oparty o komentarze XML. Te komentarze są formalnie pojedynczymi liniami komentarza C# zawierającymi znaczniki XML. Tak wygląda wzór pojedynczej lini komentarza:

```
      <summary>
/// Ta klasa...
/// </summary>
```

Wieloliniowe komentarze XML zawierają się w takim wzorze:

```
     <exception cref="BogusException">
/// This exception gets thrown as soon as
a /// Bogus flag gets set.
     </exception>
```

Wszystkie linie muszą być poprzedzone potrójnym slashem aby były zaakceptowane jako linie komentarza XML. Znaczniki dzielą się na dwie kategorie categories:

- Elementy dokumentacji
- Formatowanie/Odnośniki

Pierwsza kategoria zawiera znaczniki takie jak <summary>, <param> lub <exception>. Reprezentują one elementy API programu, które muszą być udokumentowane dla tego programu, a które będą użyteczne dla innych programistów. Znaczniki te zazwyczaj mają atrybuty, takie jak name lub cref zademonstrowane w wieloliniowym przykładzie powyżej. Atrybuty te są sprawdzane przez kompilator, więc powiny być poprawne. Kolejna kategoria wpływa na rozkład dokumentacji, używając takich znaczników jak <code>, <list> lub <para>. Dokumentacja może być potem wygenerowana przy użyciu elementu 'documentation' we wbudowanym menu #develop. Dokumentacja wygenerowana jest w formacie HTMLHelp. Po pełne wyjaśnie komentarzy XML zobacz dokumentację Microsoft .Net Framework SDK. Po informację o najlepszych praktykach komentowania zajrzyj do TechNote 'The fine Art of Commenting'.

5 Deklaracje

5.1 Kilka deklaracji na linię

Zalecana jest jedna deklaracja na linię, ponieważ wspiera to komentowanie. Innymi słowy,

```
int level; // poziom wcięcia
size;  // rozmiar tablicy
```

Nie wstawiaj więcej niż jednej zmiennej lub zmiennych różnych typów w tej samej linii kiedy je dekalrujesz.Przykład:

```
int a, b; //Co to jest 'a'? Co symbolizuje 'b'?
```

Powyższy przykład demonstruje również wady nieoczywistych nazw zmiennych. Jasno nazywaj zmienne.

5.2 Inicjalizacja

Spróbuj zainicjalizować lokalne zmienne, jak tylko zostaną zadeklarowane. Na przykład:

```
string  name  = myObject.Name; lub

int     val   = time.Hours;
```

Notka: Jeśli zainicjalizujesz okienko dialogowe, spróbuj użyć instrukcji using:

```
using (OpenFileDialog openFileDialog = new OpenFileDialog()) {

    ....

}
```

5.3 Deklaracje klasy i interfejsu

Kiedy kodujemy klasy i interfejsy C#, powinniśmy zastosować się do poniższych zasad:

- Żadnych spacji między nazwą metody a nawiasami " (" zaczynającymi listę

parametrów.

- Nawiasy otwierające " {" pojawiają się w kolejnej linii po instrukcji deklaracji.
- Nawias zamykający " }" zaczyna się w linii wcięcia pasującej do odpowiedniego nawiasu otwierającego.

Na przykład :

```
  class MySample : MyClass, IMylnterface

{

      int myInt;
      public MySample(int mylnt)
      {
          this.mylnt = mylnt ;
      }

      void Inc() {
          ++mylnt; }

  void EmptyMethod() {

   }

}
```

6 Instrukcje

6.1 Proste instrukcje

Każda linia powinna zawierać tylko jedną instrukcję.

6.2 Instrukcja return

Instrukcja return nie powinna używać nawiasów zewnętrznych.
Nie używaj : return (n * (n + 1) / 2) ;
użyj : return n * (n + 1) / 2 ;

6.3 Instrukcje if, if-else, if else-if else

Instrukcje if, if-else i if else-if else powinny wyglądać tak jak poniżej:

```
  if (warunek) {
      RóbCoś();



  if (warunek) {
      RóbCoś ();

  } else {
      RóbCośInnego () ;



  if (warunek) {
      RóbCoś ();

  } else if (warunek) { RóbCośInnego () ;

  } else {
      RóbCośJeszczeInnego ();
```

6.4 Instrukcje For/ Foreach

Instrukcja for powinna mieć następującą postać :

```
for (int i = 0; i < 5; ++i) {
.....
}
```

lub pojedynczą linię :

```
for (inicjalizacja; warunek; aktualizacja) ;
```

foreach powinna wyglądać tak :

```
foreach (int i in IntList) {
.....
}
```

Notka: Generalnie używajmy nawiasów nawet jeśli jest tylko jedna instrukcja w pętli.

6.5 Instrukcja While/do-while

Instrukcja while powinna być zapisana jak następuje:

```
while (condition) {
......
}
```

Pusta pętla while powinna mieć następującą formę:

```
while (warunek) ;
```

Instrukcja do-while powinna mieć formę:

```
do {
.....
} while (warunek);
```

6.6 Instrukcja Switch

Instrukcja switch powinna mieć następującą formę:

```
switch (condition) {
        case A:
        ........
        break;
        case B:
        ........
```

```
            break;

            default:

            ........

            break;

}
```

6.7 Instrukcja Try-catch

Instrukcja try-catch powinna mieć następującą formę**:**

```
    try {
    ..........

    } catch (Wyjątek) { }
```

lub

```
    try {

    ..........

    } catch (Wyjątek e) {

    .........

    }
```

lub

```
   try {

   } catch (Wyjątek e) {

   .......

   }  finally  {

   ........

   }
```

7 Spacje

7.1 Puste linie

Puste linie poprawiają czytelność. Rozpoczynają blok kodu, który jest logicznie powiązany. Dwie puste linie powinny zawsze być używane pomiędzy:

* Logicznymi sekcjami pliku źródłowego

* Definicjami klasy i interfejsu (spróbuj jednej klasy/interfejsu na plik aby zabezpieczyć się przed tym przypadkiem)

Jedna pusta linia powinna być używana pomiędzy:

* Metodami
* Właściwościami
* Lokalnymi zmiennymi w metodzie i jej pierwszą instrukcją
* Logicznymi sekcjami wewnątrz metody do poprawy czytelności

Zauważ ,że puste linie muszą być wcięte ponieważ zawierają instrukcje, które tworzą wstawianie tych linii dużo łatwiejszym.

7.2 Wstawianie spacji między elementami

Powinna być pojedyncza spacja o przecinku lub średniku, na przykład:

```
TestMethod(a, b, c);      nie używaj : TestMethod(a,b,c)
                                       lub
                                       TestMethod( a, b, c
                                    );
```

Pojedyncze spacje otaczają operatory (z wyjątkiem operatorów jednoargumentowych, jak zwiększanie lib logiczne nie), przykład:

```
a = b;                          // nie używaj a=b;
for (int i = 0; i < 10; ++i) // nie używaj (int i=0; i<10; ++i)
                                //            lub
                                //            for(int i=0;i<10;++i)
```

7.3 Tablica jako formatowanie

Logiczny blok linii powinien być sformatowany jako tablica:

```
string name    = "Mr. Ed";
int    myValue = 5 ;
Test   aTest   = Test.TestYou;
```

Użyj spacji dla tablicy jako formatowanie a nie tabulacji ponieważ formatowanie tablicy może wyglądać dziwnie.

8 Konwencje nazewnicze

8.1 Styl pisania wielkimi literami

8.1.1 Styl Pascala

Ta konwencja zaczyna każde słowo z dużej litery (**T**est**C**ounter).

8.1.2 Styl Wielbłąda

Ta konwencja zaczyna pierwszy znak każdego słowa dużą literą z wyjątkeim pierwszego słowa.Np. testCounter .

8.1.3 Duże litery

Używaj wszystkich dużych liter dla identyfikatorów, które są skrótami skałdającymi się z jednej lub dwóch liter, identyfikatory z większą ilością znaków powinny używać stylu Pascala. Na przykład:

```
public class Math
{
        public const PI = ... public const E = . . . public const
        feigenBaumNumber = ...
```

8.2. Wskazówki co do nazewnictwa

Generalnie używamy znaku podkreślenia wewnątrz nazw i nazewnictwa zgodnie ze wskazówkami dotyczącymi notacji węgierskiej, co jest złą praktyka. Notacja węgierska definiuje zbiór przed i przyrostków, które są stosowane z nazwami odzwierciedlając typ zmiennej. Ten styl nazewnictwa był szeroko używany we wcześniejszym programowaniu Windows ale teraz jest przestarzały. Pamiętaj: dobra nazwa zmiennej opisuje semantykę a nie typ. Wyjątkiem od tej zasady jest kod GUI. Wszystkie nazwy pól i zmiennych zawierających elementy GUI takie jak przycisk powinny zawierać przedrostek z nazwą typu bez skrótów. Na przykład:

```
System.Windows.Forms.Button  cancelButton;
System.Windows.Forms.TextBox nameTextBox;
```

8.2.1 Wskazówki nazewnictwa klas

* Nazwa klasy musi być rzeczownikiem lub grupą rzeczownikową.
* Używaj stylu Pascala
* Nie używaj żanego prefiksu klasy

8.2.2 Wskazówki nazewnictwa interfejsu

* Nazwa interfejsu rzeczownikiem lub grupą rzeczownikowa lub przymiotnikiem opisuje zachowanie. (Przykład Icomponent lub IEnumberable)
* Użyj stylu Pascala
* Użyj prefiksu dla nazw, po którym następuje duża litera (pierwszy znak nazwy interfejsu)

8.2.3 Wskazówki nazewnictwa numeracji

* Użyj stylu Pascala dla nazw wartości numeracji i nazw typów numeracji
* Nie wstawiaj przedrostków (lub przyrostków) typom numeracyjnym lub wartościom numeracyjnym
* Użyj liczby pojedynczej dla nazw numeracji
* Użyj liczby mnogiej dla nazw pól bitów.

8.2.4 Nazwy stałych pól i tylko do odczytu

* Nazwij pola statyczne rzeczownikami, grupą rzeczowników lub skrótami rzeczowników
* Użyj stylu Pascala

8.2.5 Nazwy parametru / nie stałych pól

* Używaj nazw opisowych, które powinny być wystarczające dla określenia znaczenia zmiennej i jej typu. Ale lepsze są nazwy oparte o znaczenie parametru.
* Użyj stylu Wielbłąda

8.2.6 Nazwy zmiennych

* Zmienne zliczające najlepiej nazywać i, j, k, 1, m, n kiedy używane są w 'trywialnych' pętlach zliczających.
* Użyj stylu Wielbłąda

8.2.7 Nazwy metod

* Nazwy metod są czasownikami lub grupą czasownikową.
* Użyj stylu Pascala

8.2.8 Nazwy właściwości

* Nazwij właściwości używając rzeczowników lub grupy rzeczownikowej
* Użyj stylu Pascala
* Rozważ nazywanie właściwości tą samą nazwą jak jej typ

8.2.9 Nazwy zdarzeń

* Nazwij obsługę zdarzeń z przyrostkiem EventHandler.
* Użyj dwóch parametrów nazwanych sender i e
* Użyj stylu Pascala
* Nazwij argument zdarzenia klas z przyrostkiem EventArgs.
* Nazwij nazwę zdarzenia, które ma jest przed lub po, używając czasu teraźniejszego lub przeszłego.
* Rozważ nazywanie zdarzeń czasownikami.

8.2.10 Podsumowanie kapitalizacji

Typ	*Styl*	*Notka*
Klasa / Struktura	Pascal	
Interfejs	Pascal	Zacznij z I

Wartości numeracyjne	Pascal	
Typ enumeracyjny	Pascal	
Zdarzenia	Pascal	
Klasa Exception	Pascal	Zakończ z Exception
Pola public	Pascal	
Metody	Pascal	
Przestrzeń nazw	Pascal	
Właściwości	Pascal	
Pola Protected/private	Wielbłąd	
Parametry	Wielbłąd	

9 Praktyka Programistyczna

9.1 Widzialność

Nie czyń żadnej instancji lub zmiennej klasy publiczną, uczyń je prywatnymi. Dla prywatnych elementów lepiej nie używać "private" jako modyfikatora, lepiej nie pisać nic. Private jest domyślne i każdy programista C # powinien na to zważać. Zamiast tego użyj właściwości. Możesz użyć publicznych pól statycznych (lub stałych) jako wyjątku od tej zasady, ale to nie powinna być zasada.

9.2 Żadnych liczb 'magicznych'

Nie używaj żadnych magicznych liczb, tj. umieszczanie stałych wartości numerycznych bezpośrednio w kodzie źródłowym. :

```
public class MyMath
{
    public const double PI = 3.14159...
```

10 Przykłady kodu

10.1 Przykład umieszczenia nawiasów klamrowych

```
namespace ShowMeTheBracket

    public enum Test {
        TestMe, TestYou


    public class TestMeClass

        Test test;

        public Test Test { get
            {
                return test;

            set {
                test  = value;



        void DoSomething()

            if (test == Test.TestMe) {
                //...stuff gets done
            } else {
                //...other stuff gets done
```

```
                }
            }
        }
}
```

Nawiasy klamrowe powinny być w nowej linii tylko po:

- Deklaracjach przestrzeni nazw
- Deklaracjach Klasy/Interfejsu/Struktury
- Deklaracjach metod

10.2 Przykład nazywania zmiennych

instead of:

```
for (int i = 1; i < num; ++i) {
    meetsCriteria[i] = true; }
for (int i = 2; i < num / 2; ++i) {
    int j = i + i; while (j <= num)
    {
        meetsCriteria[j] = false; j
        += i;


for (int i = 0; i < num; ++i) {
    if (meetsCriteria[i]) {
        Console.WriteLine(i + " meets criteria");
```

try intelligent naming :

```
for (int primeCandidate = 1; primeCandidate < num; ++primeCandidate)
{
    isPrime[primeCandidate] = true;
}

for (int factor =2; factor < num / 2; ++factor) {
    int factorableNumber = factor + factor; while
    (factorableNumber <= num) {
isPrime[factorableNumber] = false;
factorableNumber += factor; } }

for (int primeCandidate = 0; primeCandidate < num;
++primeCandidate) {
    if (isPrime[primeCandidate] ) {
        Console.WriteLine(primeCandidate + " is prime.");
        }
   }
```