C#: ВИМОГИ І РЕКОМЕНДАЦІЇ З НАПИСАННЯ КОДУ

СТИЛЬ

Написати програму – це більше, ніж правильно її оформити та змусити коректно виконуватись. Програми згодом неминуче модифікуються, повторно використовуються для побудови інших програм тощо. Тому велике значення має простота та зрозумілість їхньої внутрішньої структури. Інколи добре написану чужу програму набагато простіше зрозуміти, ніж власну, але написану погано. Культура написання коду сприяє зменшенню помилок у програмах і полегшує їхню модифікацію та повторне використання.

Під *стилем* програмування зазвичай розуміють набір прийомів і методів, що застосовуються з метою одержання правильних і ефективних програм, зручних для сприйняття, повторного застосування й модифікації.

СТИЛЬ

Вибір імені. Імена мають програми, дані, типи, класи, функції, методи. Коли їх у програмі з десяток чи два, то вибір не є дуже принциповим питанням. Однак коли їхня кількість у системі сягає десятків, сотень, а то й тисяч, то, щоб не втратити контроль над системою, при виборі імен необхідно дотримуватись певної строгої дисципліни й порядку.

Доцільно використовувати осмислені імена для глобальних змінних і, за можливості, короткі – для локальних. Глобальні змінні, функції, класи та структури за означенням можуть з'являтись у довільному місці програми, тому найважливішою для них є інформативність, а не довжина. Краще використовувати глобальні імена так, щоб вони явно вказували на смисл понять, що ними подаються, тобто мали відповідну *мнемоніку*. Корисно опис кожної глобальної змінної, функції, типу даних, константи супроводжувати коротким коментарем.

СТИЛЬ

Однак коментарі не мають містити очевидної інформації на зразок того, що оператор **`i++`** збільшує змінну **`i`** тощо.

Для локальних змінних, навпаки, краще підходять короткі імена. Для використання всередині функції цілком підійдуть імена **`i, j, n`** тощо. При цьому для лічильників циклу зазвичай використовуються імена **`i`**, **`j`**, для рядків – **`s, t`**. Якщо це можливо, використовують англомовні назви змінних, типів, класів, функцій, констант, які відповідають їхній ролі в програмі.

Існує багато корпоративних домовленостей і традицій іменування, які фіксуються у спеціальних внутрішніх стандартах виробників. Наприклад, часто для функцій використовуються імена, побудовані з дієслів та іменників

СТИЛЬ

Оператори й вирази. Ці основні конструкції програм слід писати так, щоб їхній зміст був максимально зрозумілим. Найпростіший спосіб досягти цього – форматування коду за допомогою відступів, табуляцій, порожніх рядків. Розглянемо два тексти однієї й тієї самої програми. Очевидно, що другий текст більш читабельний саме завдяки використанню його якісного форматування.

СТИЛЬ

```
public class TestCollections{   public static void TestList()
    {var testList = new List<int>{3, 4, 2, 1};
for(var i = testList.Count -1; i >= 0; i--) {if(testList[i]>2) testList.RemoveAt(i); }
testList.Sort();foreach (int el in testList)  {Console.WriteLine(el); }}}
```

Лістинг 2.

```
public class TestCollections
{
    public static void TestList()
    {
        var testList = new List<int>{3, 4, 2, 1};
        for(var i = testList.Count -1; i >= 0; i--)
        {
         if(testList[i]>2)
              testList.RemoveAt(i); /*видалення за індексом*/
        }
         /*сортування в лексикографічному порядку*/
         testList.Sort();
        foreach(int el in testList)
        {
            Console.WriteLine(el);
        }
    }
}
```

СТИЛЬ

Зрозумілість і стисліть коду – не одне й те саме. Головним критерієм вибору синтаксису для коду має бути простота його сприйняття.

Відступи, табуляції, переходи на новий рядок допомагають краще структурувати код. Чи варто розташовувати відкриваючу фігурну дужку в тому самому рядку, що й `if` або `while`, чи в наступному? Важлива не стільки конкретика вибраного стилю, скільки логічність і послідовність його застосування.

Вважається, що один метод не має займати більш ніж один екран. Якщо це не так, то їх краще розбити на дрібніші. **Якщо певний фрагмент коду вимагає коментування, то його теж краще виділити в окремий метод.**

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

Pascal casing

Описуються імена:

- всіх визначень типів, у тому числі призначених для користувача класів, перерахувань, подій, делегатів і структур;
- значення перерахувань;
- **`readonly`** полів і констант;
- інтерфейсів;
- методів;
- просторів імен (**`namespace`**);
- властивостей;
- публічних полів;

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

```
namespace SampleNamespace
{
  enum SampleEnum
  {
    FirstValue,
    SecondValue
  }

  struct SampleStruct
  {
    public int FirstField;
    public int SecondField;
  }

  interface ISampleInterface
  {
    void SampleMethod();
  }
```

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

```
public class SampleClass: SampleInterface
  {
    const int SampleConstValue = 0xffffff;

    readonly int SampleReadonlyField;

    public int SampleProperty
    {
      get;
      set;
    }
    public int SamplePublicField;

    SampleClass()
    {
      SampleReadonlyField = 1;
    }
    delegate void SampleDelegate();
    event SampleDelegate SampleEvent;

    public void SampleMethod()
    {
    }
  }
```

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

Camel casing

Описуються імена:

- локальних змінних;
- аргументів методів;
- захищених (**protected**) полів.

```
protected int sampleProtectedField;

int SampleMethod(int sampleArgument)
{
  int sampleLocalVariable;
}
```

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

Upper case

При цьому угоді в іменуванні використовуються тільки великі літери. Цей стиль використовується тільки при іменуванні «коротких» констант, наприклад **`PI`** або **`E`**. В інших випадках бажано використовувати Pascal Casing.

наприклад:

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

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

Зведена таблиця використання іменувань

Type	Case	Notes
Class / Struct	Pascal Casing	
Interface	Pascal Casing	Starts with I
Enum values	Pascal Casing	
Enum type	Pascal Casing	
Events	Pascal Casing	
Exception class	Pascal Casing	End with Exception
public Fields	Pascal Casing	
Methods	Pascal Casing	

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

Суфікси і префікси

Застосовуються такі суфікси і префікси:

- імена користувальницьких класів винятків завжди закінчуються суфіксом "**`Exception`**";
- імена інтерфейсів завжди починаються із префікса «**`I`**»;
- імена користувальницьких атрибутів завжди закінчуються суфіксом «**`Attribute`**»;
- імена делегатів обробників подій завжди закінчуються суфіксом **`EventHandler`**, імена класів-спадкоємців від **`EventArgs`** завжди закінчуються суфіксом **`EventArgs`**.

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

```
public class SampleException: System.Exception
{
  public SampleException()
  {  }
}
interface ISample
{
  void SampleMethod();
}

[System.AttributeUsage(System.AttributeTargets.All,  Inherited  =  false,
AllowMultiple = true)]
sealed class SampleAttribute: System.Attribute
{
  public SampleAttribute()
  {  }
}     public  delegate  void  AnswerCreatedEventHandler(object  sender,
AnswerCreatedEventArgs e);
  public class AnswerCreatedEventArgs: EventArgs
  {
    public int CreatedId;
    public int ParentId;
    public string CreatorName;
  }
```

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

Абревіатури

При використанні абревіатур в іменах, **капіталізації підлягають абревіатури з двома символами**, в інших абревіатурах необхідно приводити до верхнього регістру тільки перший символ.

```
namespace Sample.IO
{
}

class HttpUtil
{
}
```

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

Іменування методів

Використовуйте конструкцію **дієслово-об'єкт** для іменування методів

```
ShowUserInfo ()
```

В окремому випадку, для методів, які повертають значення, використовуйте в парі дієслово-об'єкт для дієслова «**Get**», а для об'єкта - опис значення, що повертається.

```
GetUserId ()
```

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ

Змінні, поля і властивості C#

- При іменуванні змінних уникайте використання скорочених варіантів таких як **i** і **t**, використовуйте **index** і **temp**. Не скорочуйте слова, використовуйте **number**, а не **num**.
- Рекомендується для імен елементів управління вказувати префікси, що описують тип елемента. Наприклад: **txtSample, lblSample, cmdSample** або **btnSample**. Ця ж рекомендація поширюється на локальні змінні складних типів: **ThisIsLongTypeName tltnSample =**

```
new ThisIsLongTypeName ();
```

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ

Змінні, поля і властивості C#

- не використовуйте публічних або захищених полів, замість цього використовуйте властивості;
- використовуйте автоматичні властивості;
- завжди вказуйте модифікатор доступу **private**, навіть якщо дозволено його опускати;
- завжди ініціалізуйте змінні, навіть коли існує автоматична ініціалізація.

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

Додаткові рекомендації

- використовуйте порожній рядок між логічними секціями у вихідному файлі, класі, методі;
- використовуйте проміжну змінну для передачі bool-значення результату функції в умовний вираз **if**;

```
bool boolVariable = GetBoolValue();
if (boolVariable)
{
}
```

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

Об'єм коду

- уникайте файлів з більш ніж 500 рядками коду;
- уникайте методів з більш ніж 20 рядками коду;
- уникайте методів з більш ніж 5 аргументами, використовуйте структури для передачі великого числа параметрів;
- один рядок коду не повинен мати довжину більше 120 символів.

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

Закриті поля

Імена закритих полів завжди починаються із префікса «_» інша частина імені описується за допомогою Camel Casing.

```
private int _samplePrivateField;
```

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

Блоковий коментар

Як правило, використовується блоковий коментар для опису, а так само стандартний «///» коментар для самодокументування.

Для стандартних блокових коментарів використовуються наступні стилі:

```
/ *
* Line 1
* Line 2
* Line 3
* /
або такий стиль:
/ * коментар * /
```

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

Коментар «до кінця рядка»

Для коментарів у один рядок використовується «C++» подібний стиль: «**//**» Цей стиль найбільш зручний при документуванні параметрів. Переважно використовувати даний стиль замість **/ * коментар * /** там, де це можливо.

```
int i; // змінна для циклу
```

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

Кількість оголошень на одній лінії

Загальноприйнятим стандартом в оголошеннях є один рядок на екземпляр. Завдяки цьому з'являється зручність читання і написання коментаря:

```
int level; // indentation level
int size; // size of table
```

Бажано не ставити оголошення в один рядок.

```
int a, b; // Неправильно!
```

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

Ініціалізація

За можливості ініціалізуйте локальні змінні в тому ж місці де й оголошуєте.

Наприклад:

```
string name = myObject.Name;
```

або

```
int val = time.Hours;
```

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

Оголошення класів та інтерфейсів

У процесі розробки класів та інтерфейсів, дотримуйтесь наступних правил:

- Між назвою методу і відкриваючою дужкою з перерахуванням параметрів не повинно бути пропусків.
- Відкриваюча фігурна дужка «{« повинна починатися на наступному рядку відразу під назви класу або його методу.
- Код, який слідує під відкриває фігурною дужкою «{«, повинен знаходиться на зміщенні одного Tab символу вправо.
- Закриваюча фігурна дужка «}» повинна знаходиться на рівні з відкриваючою і охоплювати весь блок коду.

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

```
class MySample : MyClass, IMyInterface
   {
      int myint;

      public MySample(int myint)
      {
         this.myint = myint;
      }

      void EmptyMethod()
      {
      }
   }
```

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

Порожні рядки

Порожні рядки допомагаю розбивати код програми на логічні сегменти.

Декількома рядками можуть відділятися:

- секції у вихідному файлі;
- класи та інтерфейси;

Одним порожнім рядком відокремлюються один від одного:

- методи;
- локальні змінні від перших операторів;
- логічні секції всередині методу для більш зручного читання

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

Прогалики

Один прогалик використовується в оголошенні методів після коми, але не перед дужками: **`TestMethod (a, b, c);`**

Приклад неправильного використання:

`TestMethod (a,b ,c);` або **`TestMethod (a, b, c );`**

Так само одиночний прогалик може бути використаний для виділення операторів: **`a = b;`**

неправильне використання: **`a=b;`**

Також прогалики використовуються і при форматуванні циклів:

```
for (int i = 0; i < 10; ++i);
```

так не треба використовувати:

```
for (int i=0; i<10; ++i)
```

або

```
for (int i = 0;i <10;+ + i)
```

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

«Табличне» форматування

При оголошенні і ініціалізації змінних бажано використовувати «табличне» форматування:

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

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

Додаткові вимоги

- завжди розташовуйте відкриваючі та закриваючі фігурні дужки на новому рядку;
- завжди використовуйте фігурні дужки для виразів if, навіть коли у вираз входить лише один рядок коду;
- не використовуйте пробіл ЗАМІСТЬ символ табуляції.

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

Додаткові вимоги

- завжди розташовуйте відкриваючі та закриваючі фігурні дужки на новому рядку;
- завжди використовуйте фігурні дужки для виразів if, навіть коли у вираз входить лише один рядок коду;

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

«Видимість»

Намагайтеся застосовувати максимум інкапсуляції для примірників об'єктів які ви створюєте. Чи не ставте public там де це не потрібно. Використовуйте максимальний рівень захисту. Тобто намагайтеся відкривати доступ від мінімального (**`private`**) до максимального (**`public`**).

Використовуйте властивості замість прямого відкриття **`public`**, для змінних класу.

ДОМОВЛЕНОСТІ ПО НАЙМЕНУВАННЯМ C#

«Магічні числа»

Не використовуйте так званих «магічних чисел». Замість цього оголошуйте константи і статичні змінні:

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