Именование в .NET

* Не экономить на понятности и чистоте кода;
* Сопровождать объявление переменной комментариями, зачем она нужна;
* Не использовать малопонятные префиксы и ( или) суффиксы;
* Не используйте подчеркивание для отдельных слов внутри идентификаторов;
* Не размещайте несколько инструкций на одной строке;

Если строка превышает длину 80 символов, то её надо переносить на другую строчку;

* Перенос идёт после запятой;
* Перенос после оператора;
* Перенесённая линия по отношению к верхней должна быть сдвинута вправо как минимум на один стандартный символ табуляции либо на уровне начала переносимого выражения;

Счетчики в циклах объявляются как: **i,j,k,l,m,n**;

Названия для пространств имён пишутся так:

(Название компании или организации) & (название продукта) & (Некое имя значимое).

Множественное число использовать в пространствах имен только в том случае, если оно объединяет некоторое количество разных сущностей. А если пространство имён содержит лишь одну систему – то использовать только единственное число.

* При импортировании **пространств имен** их перечисление идёт в следующей последовательности:
* Пространства имён .NET Framework;
* Пространства имён сторонних производителей;
* Собственные пространства имён из других проектов;
* Пространства имён из текущего проекта.
* Все разделы должны разделяться пустой строкой, имена в разделах отсортированы по алфавиту.

Не используйте аббревиатуры или сокращения как часть имени идентификатора.

Например, предпочтительнее использовать *OnButtonClick*, чем *OnBtnClick*.

**Классы и структуры**

* Для имени класса (структуры) использовать надо существительное (одно или несколько прилагательных и одно существительное);
* Не использовать префикса поясняющего, что это класс или структура, использовать стиль Pascal для регистра букв;
* В подходящих случаях используйте составные слова для производных классов, где вторая часть слова поясняет базовый класс. К примеру ApplicationException. Если производный класс незначительно меняет свойства, поведение или внешний вид базового, используйте новое название отражающее суть класса;
* Используйте составное имя, когда класс принадлежит некоторой специфичной категории, например потоки, коллекции, исключения, ассоциативные контейнеры;
* Для классов предназначенных для наследования использовать суффикс Base. Не стоит забывать про абстрактные классы;

В общем случае в именах типов следует использовать фразы на основе существительных, где существительное является сущностью, представляемой типом. Например, [Button](https://msdn.microsoft.com/ru-ru/library/system.windows.forms.button(v=vs.100).aspx), [Stack](https://msdn.microsoft.com/ru-ru/library/system.collections.stack(v=vs.100).aspx) и [File](https://msdn.microsoft.com/ru-ru/library/system.io.file(v=vs.100).aspx) имеют имена, определяющие сущность, представляемую типом. Выбирайте имена, которые определяют сущность с точки зрения разработчика; имена должны отражать сценарии использования.

При выборе имен типов необходимо следовать приведенным ниже рекомендациям.

**В именах классов, интерфейсов и типов значения используйте существительные, фразы на основе существительных или в некоторых случаях фразы на основе прилагательных. Применяйте правила использования прописных и строчных букв языка Pascal.**

**Не следует использовать в именах префикс (например, букву C).**

Исключением из этого правила являются интерфейсы, имена которых начинаются с буквы I.

**В конце имени производного класса рекомендуется ставить имя базового класса.**

Например, имена типов .NET Framework, наследующих от класса [Stream](https://msdn.microsoft.com/ru-ru/library/system.io.stream(v=vs.100).aspx), оканчиваются словом Stream, а имена типов, наследующих от класса [Exception](https://msdn.microsoft.com/ru-ru/library/system.exception(v=vs.100).aspx), оканчиваются словом Exception.

**Для имени интерфейса следует применять в качестве префикса букву "I". Это указывает, что данный тип является интерфейсом.**

**Необходимо обеспечивать, чтобы при определении пары "класс/интерфейс", где класс является стандартной реализацией интерфейса, имена различались только префиксом "I" в имени интерфейса.**

Например, в библиотеке .NET Framework содержится интерфейс [IAsyncResult](https://msdn.microsoft.com/ru-ru/library/system.iasyncresult(v=vs.100).aspx) и класс [AsyncResult](https://msdn.microsoft.com/ru-ru/library/system.runtime.remoting.messaging.asyncresult(v=vs.100).aspx).

**Имена параметров универсального типа**

**При именовании параметров универсальных типов используйте описательные имена, если только однобуквенное имя не является полностью понятным без пояснений, вследствие чего нет необходимости применять описательное имя.**

В качестве примера имени интерфейса, выбранного в соответствии с данным правилом, можно привести интерфейс [IDictionary<TKey, TValue>](https://msdn.microsoft.com/ru-ru/library/s4ys34ea(v=vs.100).aspx).

**Для типов с однобуквенными параметрами рекомендуется использовать букву "T" в качестве имени параметра типа.**

**К описательным именам параметров типа добавляйте префикс в виде буквы "T".**

**В имени параметра рекомендуется указывать ограничения, накладываемые на параметр типа. Например, параметр, предназначенный только для интерфейса "Isession", может называться "TSession".**

**Имена общих типов**

К классам пользовательских атрибутов добавляйте суффикс "Attribute".

В качестве примеров имен типов, удовлетворяющих этой рекомендации, можно назвать атрибуты [ObsoleteAttribute](https://msdn.microsoft.com/ru-ru/library/system.obsoleteattribute(v=vs.100).aspx) и [AttributeUsageAttribute](https://msdn.microsoft.com/ru-ru/library/system.attributeusageattribute(v=vs.100).aspx).

Добавляйте суффикс "EventHandler" к именам типов, используемых в событиях (таким как возвращаемые типы события C#).

[AssemblyLoadEventHandler](https://msdn.microsoft.com/ru-ru/library/system.assemblyloadeventhandler(v=vs.100).aspx) является именем делегата, удовлетворяющим данной рекомендации.

Добавляйте суффикс "Callback" к имени делегата, не являющегося обработчиком событий.

Не добавляйте к имени делегата суффикс "Delegate".

Добавляйте суффикс "EventArgs" к именам классов, расширяющих класс System.EventArgs.

Не создавайте производных классов от класса "System.Enum"; используйте вместо этого ключевое слово, поддерживаемое используемым языком программирования. Например, в C# используйте ключевое слово "enum".

Добавляйте суффикс "Exception" к типам, наследующим от класса "System.Exception".

Добавляйте суффикс "Dictionary" к типам, реализующим интерфейсы "System.Collections.IDictionary" или "System.Collections.Generic.IDictionary<TKey, TValue>". Обратите внимание, что интерфейс "System.Collections.IDictionary" является особым типом коллекции, но эта рекомендация превалирует над приведенной ниже более общей рекомендацией относительно коллекций.

Добавляйте суффикс "Collection" к типам, реализующим интерфейсы "System.Collections.IEnumerable", "System.Collections.ICollection", "System.Collections.IList", "System.Collections.Generic.IEnumerable<T>", "System.Collections.Generic.ICollection<T>" или "System.Collections.Generic.IList<T>".

Добавляйте суффикс "Stream" к типам, производным от класса "System.IO.Stream".

Добавляйте суффикс "Permission" к типам, производным от класса "System.Security.CodeAccessPermission" или реализующим интерфейс "System.Security.IPermission".

**Имена свойств**

**В именах свойств используйте существительные, фразы на основе существительных или прилагательные.**

Фразы на основе существительных и прилагательные предпочтительны для свойств, так как свойства содержат данные.

**Не используйте имена свойств, совпадающие с именами методов Get.**

Например, не называйте одновременно свойство EmployeeRecord, а метод — GetEmployeeRecord. Разработчики не смогут понять, какой член использовать для выполнения задачи программирования.

**В именах логических свойств используйте утвердительную фразу (CanSeek вместо CantSeek). К именам логических свойств можно дополнительно прибавлять префикс Is, Can или Has, но только тогда, когда он действительно необходим.**

**Рекомендуется присваивать свойству то же имя, что и у его базового типа.**

При наличии свойства со строго определенным типом перечисления имя свойства может совпадать с именем перечисления. Например, если перечисление имеет имя CacheLevel, свойство, возвращающее одно из его значений, также может быть названо CacheLevel.

**Имена полей**

**Применяйте в именах полей правила использования прописных и строчных букв языка Pascal.**

**Используйте в именах полей существительные или фразы на основе существительных.**

**Не используйте в именах полей префиксы. Например, не используйте "g\_" или "s\_" для разделения статических и нестатических полей.**

**Атрибуты**

Класс являющийся атрибутом должен иметь суффикс Attribute. Если семантика класса требует в названии слова что-то вроде Attribute – то заменять это что-то синонимом: Description, Sign, Qualifier, Specifier, Declarator;

**Поля**

* Непубличные поля именуются в стиле Camel и начинаются с префикса «\_»;
* Публичные поля именуются в соответствии с правилами именования свойств (см. далее);
* Одна декларация должна содержать в себе не более одного поля и должна располагаться в одной строке. Тут следуют применить комментарии до конца строки для непубличных полей или XML комментарии для публичных;
* Избегайте передачи полей по ссылке;

**Общие правила разработки классов**

1. Допускается определять в классе публичные (public и internal) поля, но они должны однозначно интерпретироваться как lightweight-поля. Оформляйте их стилем Паскаль и указывайте в документации, что не следует передавать такие поля по ссылке. Старайтесь (если, конечно, на то нет особых причин) избегать смешивания в одном классе/структуре публичных полей и свойств.

2. Старайтесь реализовать в виде свойств только то, что отражает состояние класса или объекта. Например, если вы делаете свою коллекцию, то количество элементов (Count) должно быть свойством, а операцию преобразования ее в массив (GetArray) лучше сделать методом. Причем вместо Get в данном случае лучше использовать другое слово, например, ToXxxx. То есть метод будет называться ToArray().

3. Используйте методы, если выполняемая операция является преобразованием, имеет побочный эффект или долго выполняется.

4. Используйте метод, если важен порядок выполнения операций.

5. Свойство не должно менять своего значения от вызова к вызову, если состояние объекта не изменяется. Если результат при новом вызове может быть другим при том же состоянии объекта, используйте метод.

6. Не используйте свойства «только для записи». Потребность в таком свойстве может быть признаком плохого проектирования.

7. Запечатанные (sealed) классы не должны иметь ни защищенных, ни виртуальных методов. Такие методы используются в производных классах, а sealed-класс не может быть базовым.

8. Классы, реализующие интерфейс IDisposable, ни в коем случае не должны передаваться по значению.

9. Классы, определяющие только статические методы и свойства, не должны иметь открытых или защищенных конструкторов, поскольку никогда не нужно создавать их экземпляры.

10. Не используйте литеральные константы (магические числа, зашитые в код размеры буферов, времена ожидания и тому подобное). Лучше определите константу (если вы никогда не будете ее менять) или переменную только для чтения (если она может измениться в будущих версиях вашего класса).

11. Старайтесь обрабатывать только известные вам исключения. Если вы все же обрабатываете все исключения, то или, проведя необходимую обработку, генерируйте исключение повторно, чтобы его могли обработать последующие фильтры, или выводите пользователю максимально полную информацию об ошибке. Если целью перехвата исключений является очистка ресурсов после сбоя, лучше воспользоваться секцией finally.

ОФОРМЛЕНИЯ ИНСТРУКЦИЙ

If, if-else, if else-if else операторы

Код с использованием операторов if, if-else и if else-if else должен выглядеть следующим образом:

В простом случае:

if (condition) { DoSomething(); ... }

В более усложненном:

if (condition) { DoSomething(); ... } else { DoSomethingOther(); ... }

И в самом сложном:

if (condition) { DoSomething(); ... } else if (condition) { DoSomethingOther(); ... } else { DoSomethingOtherAgain(); ... }

For / Foreach операторы

Стиль оформления кода для оператора For должен иметь следующий вид:

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

Также допускается формат в одну строку:

for(initialization; condition; update) ;

А оператор foreach должен выглядеть следующим образом:

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

While/do-while операторы

Оператор While подлежит следующему форматированию:

while(condition) { ... }

Пустой оператор While может быть иметь вид:

while(condition);

Для Do-While применим следующий формат:

do { ... } while (condition);

Switch операторы

Оператор Switch должен отвечать следующему формату:

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

Try-catch операторы

Формат написания операторов try – catch должен иметь следующий вид:

try { ... } catch (Exception) {}

либо

try { ... } catch (Exception e) { ... }

либо

try { ... } catch (Exception e) { ... } finally { ... }

Using операторы

Одиночный using:

using (initialization)

{

...

}

Вложенные using:

using (initialization1)

using (initialization2)

{

...

}

|  |  |  |
| --- | --- | --- |
| **Идентификатор** | **Регистр** | **Пример** |
| Класс | Pascal | User |
| Локальная переменная | Camel | user |
| Интерфейс | Pascal | IDisposable |
| Generic | Pascal | T, TKey, TValue |
| Public функция | Pascal | Authenticate |
| Private функция | Pascal | Authenticate |
| Параметр функции | Camel | userID |
| Public свойство | Pascal | FirstName |
| Private свойство | Pascal | FirstName |
| Public поле | Pascal | FirstName |
| Private поле | Camel | firstName |
| Enum | Pascal | UserStatus |
| Значение enum’а | Pascal | Active |
| Exception | Pascal | UserAuthenticationException |
| Event | Pascal | StatusChanged |
| Namespace | Pascal | UserManager |

http://habrahabr.ru/post/26068/