Boas Práticas em C#

Nomeando uma variável:

A documentação do Microsoft .Net Framework da as seguintes recomendações para a nomeação das variáveis:

* Evite usar underline;
* Não crie variáveis que apenas se diferenciem apenas pela sua forma. Exemplo: minhaVariavel e outra chamada MinhaVariavel;
* Procure iniciar o nome com uma letra minúscula;
* Evite usar todas as letras maiúsculas;
* Quando o nome tiver mais que uma palavra, a primeira letra de cada palavra após a primeira deve ser maiúscula (conhecido como notação camelCase);

Convenção PascalCasing

Para usar a convenção PascalCasing para nomear suas variáveis, capitalize o primeiro caractere de cada palavra. Exemplo:

```
void InitializeData();
```

A Microsoft® recomenda usar o PascalCasing quando estiver nomeando classes, métodos, propriedades, enumeradores, interfaces, constantes, campos somente leitura e namespaces.

Convenção camelCasing

Para usar esse tipo de convenção, capitalize a primeira letra de cada palavra menos da primeira. Como o exemplo:

```
int loopCountMax;
```

A Microsoft recomenda usar essa convenção na nomeação de variáveis que definem campos e parâmetros.

Para maiores informações sobre convenção de nomes pesquise “Naming Guidelines”, na documentação do Visual Studio.

Palavras reservadas:

A linguagem C# reserva setenta e cinco palavras para seu próprio uso. Estas palavras são chamadas de palavras reservadas e cada uma tem um uso particular.

Palavras reservadas também não são permitidas como nome de variáveis.

Segue uma lista que identifica todas estas palavras:

abstract	as	base	Bool
break	byte	case	Catch
char	checked	class	Const
continue	decimal	default	Delegate
do	double	else	Enum
event	explicit	extern	false
finally	fixed	float	for
foreach	goto	if	implicit
in	int	interface	internal
is	lock	long	namespace
new	null	object	operator
out	override	params	private
protected	public	readonly	ref
return	sbyte	sealed	short
sizeof	stackalloc	static	string
struct	switch	this	throw

No painel de código do Visual Studio .NET as palavras reservadas são identificadas pela cor de letra azul.

Declarando variáveis:

Antes de usar uma variável é necessário declará-la. Neste momento alocamos espaço para esta variável na memória e dizemos que tipo de dado pode ser armazenado nela. O tipo de dado indica qual o tamanho do espaço vai ser reservado para a variável.

O C# pode armazenar diferentes tipos de dados: como inteiros, números de ponto flutuante, textos e caracteres. Assim que declaramos uma variável precisamos identificar que tipo de dado ela armazenará.

Declaramos especificando o tipo de dado seguido do nome da variável como no exemplo:

```
int contador;
```

Esse exemplo declara uma variável chamada contador do tipo integer. Ou seja ela deverá armazenar números inteiros, mais a frente estudaremos melhor o que armazenar em cada tipo de dado.

Podemos também declarar múltiplas variáveis de uma vez, fazemos isso da seguinte maneira:

```
int contador, numeroCarro;
```

Estamos declarando nesse exemplo duas variáveis do tipo integer, uma chamada contador e a outra numeroCarro.

Atribuindo valor a variáveis:

Depois de declarar sua variável você precisa atribuir um valor a ela. No C# você não pode usar uma variável antes de colocar um valor nela, isso gera um erro de compilação.

Exemplo de como atribuir um valor a uma variável:

```
int numeroFuncionario;

numeroFuncionario = 23;
```

Primeiro nos declaramos nossa variável do tipo integer. Depois atribuímos o valor 23 a ela. Entendemos pelo sinal de igual como recebe. Assim numeroFuncionario recebe 23.

Podemos também atribuir um valor a variável quando a declaramos, dessa forma:

```
int numeroFuncionario = 23;
```

Isso faz a mesma coisa que o exemplo anterior, só que tudo em uma linha.

Mais um exemplo:

```
char letraInicial = 'M';
```

Tipos de variáveis:

A seguinte tabela mostra os tipos do C# com sua referencia no Framework.

Os tipos da tabela abaixo são conhecidos como tipos internos ou Built-in.

C# Type	.NET Framework type
bool	System.Boolean
byte	System.Byte
sbyte	System.SByte
char	System.Char
decimal	System.Decimal
double	System.Double
float	System.Single
int	System.Int32
uint	System.UInt32
long	System.Int64
ulong	System.UInt64
object	System.Object
short	System.Int16
ushort	System.UInt16
string	System.String

Cada tipo no C# é um atalho para o tipo do Framework. Isso quer dizer que se declararmos a variável desta forma:

string nome;

ou dessa forma

System.String nome;

teremos o mesmo resultado. O atalho serve apenas para facilitar na hora de desenvolver a aplicação.

A seguinte tabela mostra os tipos de variáveis e os valores possíveis de se armazenar em cada uma delas.

C# Type	Valores possíveis de se armazenar
bool	Verdadeiro ou Falso (Valores booleandos)
byte	0 a 255 (8 bits)
sbyte	-128 a 127 (8 bits)
char	Um caractere (16 bits)
decimal	±1.0 × 10−28 a ±7.9 × 1028 (128 bits)
double	±5.0 × 10−324 a ±1.7 × 10308 (64 bits)
float	±1.5 × 10−45 a ±3.4 × 1038 (32 bits)
int	-2,147,483,648 a 2,147,483,647 (32 bits)
uint	0 a 4,294,967,295 (32 bits)
long	–9,223,372,036,854,775,808 a 9,223,372,036,854,775,807 (64 bits)
ulong	0 a 18,446,744,073,709,551,615 (64 bits)
object	Qualquer tipo.
short	-32,768 a 32,767 (16 bits)
ushort	0 a 65,535 (16 bits)
string	Seqüência de caracteres (16 bits por caractere)

Todos os tipos na tabela com exceção dos tipos object e string são conhecidos como tipos simples.

Para retornar o tipo de qualquer variável do C# você pode usar o método GetType(); Como no exemplo:

```
Console.WriteLine(minhaVariavel.GetType());
```

Isso retornaria o tipo da variável minhaVariavel.

Para maiores informações sobre tipos de variáveis consulte a documentação do Visual Studio por “data types”.

Adicionando valor a uma variável :

É muito comum precisarmos adicionar ou subtrair valores de uma variável usando no calculo o valor que já esta armazenado na mesma.

O código seguinte declara uma variável do tipo integer chamada contador e armazena o valor 2 nesta variável, depois incrementa o valor 40:

```
int contador;

contador = 2;

contador = contador + 40;
```

No final do código acima a variável contador tem qual valor?

A resposta é 42, claro, criamos a variável, adicionamos o valor 2 nela e após, pegamos o valor dela (que era 2) e adicionamos 40, e armazenamos o valor na mesma.

Preste atenção na seguinte linha de código:

```
contador = contador + 40;
```

Perceba que para somar o valor a variável precisamos repetir o nome da variável.

Podemos fazer da seguinte forma também em C#:

```
contador += 40;
```

Isso teria o mesmo resultado e é uma maneira mais elegante.

Você pode subtrair também valores, como o exemplo:

```
contador -= 23;
```

Isso subtrairia 23 do valor da variável.

Na verdade você pode fazer isso com todos os operadores aritméticos, como multiplicação e divisão também.

Comentários

Use 3 barras “///”para transformar comentários em documentação de código.

Comentários com mais de uma linha poderiam ser assim:

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

ou assim:

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

e para comentários simples de uma linha use:

// seu comentário

Operadores

Podem ser definidos na linguagem C# da seguinte forma:
- Aritméticos
- Unários
- Lógicos
- Condicionais
- Relacionais
- Igualdade

Operadores Aritméticos	
Operador	**Uso**
+	Soma tipos numéricos. Também é usado para concatenar strings
-	Efetua a diferença de tipos numéricos
/	Efetua a divisão de tipos numéricos
*	Efetua o produto de tipos numéricos
%	Retorna o resíduo da divisão

Operadores Unários	
Operador	**Uso**
+	Especifica números positivos
-	Especifica números negativos
!	Negação booleana
~	Complemento bit a bit
++ (pré)	Incremento pré-fixado. Primeiro é efetuado o incremento e depois a variável é avaliada. Exemplo: ++x
++ (pós)	Incremento pós-fixado. Primeiro a variável é avaliada e depois é efetuado o incremento. Exemplo: x++
– (pré)	Decremento pré-fixado. Primeiro é efetuado o incremento e depois a variável é avaliada. Exemplo: --x
– (pós)	Decremento pós-fixado. Primeiro a variável é avaliada e depois é efetuado o decremento. Exemplo: x--
(type)E	Operador de mascaramento de tipos. Exemplo: (int) var.

Operadores Lógicos	
Operador	**Uso**
&	Operador lógico AND. Efetua a operação AND bit a bit em tipos inteiros
|	Operador lógico OR. Efetua a operação OR bit a bit em tipos inteiros
^	Efetua a operação XOR em tipos inteiros
>>	Efetua o deslocamento de um bit à direita de tipos numéricos
<<	Efetua o deslocamento de um bit à esquerda de tipos numéricos

Operadores Condicionais	
Operador	**Uso**
&&	Operador lógico AND usado para comparar expressões booleanas
||	Operador lógico OR usado para comparar expressões booleanas

?:	Operador ternário usado da seguinte forma:
	expr. a: expr. b? expr. c.
	Isso equivale a dizer:
	`if (expr. a)`
	`expr. b;`
	`else`
	`expr. c;`

Operadores Relacionais	
Operador	**Uso**
<	Condição "menor que" para tipos numéricos
>	Condição "maior que" para tipos numéricos
>=	Condição "maior ou igual que" para tipos numéricos
<=	Condição "menor ou igual que" para tipos numéricos
is	Compara em tempo de execução se um objeto é compatível como um tipo qualquer. Esse assunto será abordado mais adiante
as	Efetua mascaramento de tipos e caso este falhe, o resultado da operação será null. Esse assunto será abordado mais adiante

Operadores de Igualdade	
Operador	**Uso**
==	Avalia a igualdade de dois tipos
!=	Avalia a desigualdade de dois tipos

Biblioteca MSDN (português) - http://msdn.microsoft.com/pt-br/library/ms123401