1. [Operador ternário](http://www.luiztools.com.br/post/13-dicas-e-truques-da-linguagem-c-sharp/#1)
2. [Operador Null-coalesce](http://www.luiztools.com.br/post/13-dicas-e-truques-da-linguagem-c-sharp/#2)
3. [Inicializadores de objetos](http://www.luiztools.com.br/post/13-dicas-e-truques-da-linguagem-c-sharp/#3)
4. [A diretiva using](http://www.luiztools.com.br/post/13-dicas-e-truques-da-linguagem-c-sharp/#4)
5. [Apelidos para nomes de classe e namespaces muito grandes](http://www.luiztools.com.br/post/13-dicas-e-truques-da-linguagem-c-sharp/#5)
6. [Tipos nullables](http://www.luiztools.com.br/post/13-dicas-e-truques-da-linguagem-c-sharp/#6)
7. [Propriedades automáticas](http://www.luiztools.com.br/post/13-dicas-e-truques-da-linguagem-c-sharp/#7)
8. [Inferência de tipos](http://www.luiztools.com.br/post/13-dicas-e-truques-da-linguagem-c-sharp/#8)
9. [Expressões Lambda](http://www.luiztools.com.br/post/13-dicas-e-truques-da-linguagem-c-sharp/#9)
10. [string.IsNullOrEmpty()](http://www.luiztools.com.br/post/13-dicas-e-truques-da-linguagem-c-sharp/#10)
11. [Use os atalhos do Visual Studio](http://www.luiztools.com.br/post/13-dicas-e-truques-da-linguagem-c-sharp/#11)
12. [Use os snippets de código](http://www.luiztools.com.br/post/13-dicas-e-truques-da-linguagem-c-sharp/#12)
13. [Estude essa lista do StackOverflow](http://www.luiztools.com.br/post/13-dicas-e-truques-da-linguagem-c-sharp/#13)

**1. Operador Ternário ‘?’**

Use-o com moderação ou ele prejudicará a legibilidade de seu código. Basicamente o operador ternário substitui um bloco if tradicional em uma única linha de código. Novidade? NOT! Operadores ternários existem na linguagem C desde a década de 70. Na prática funciona assim, imagine o seguinte bloco de código:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | int x = 10;  int y = 20;  int max;    if(x > y)     max = x;  else     max = y; |

Você pode substituí-lo usando o operador ternário que funciona da seguinte forma (Questão) ? (Ação Positiva) : (Ação Negativa). Vamos ver o exemplo anterior usando operador ternário:



|  |  |
| --- | --- |
| 1  2  3 | int x = 10;  int y = 20;  int max = (x > y) ? x : y; |

**2. Operador Null-coalesce ‘??’**

Quantas vezes você tem de testar se objetos são nulos em seu código? O operador null-coalesce pode vir a ser útil a você então. Considere o seguinte código:



|  |  |
| --- | --- |
| 1  2  3 | object cache = null;  object d = new object();  object e = (c != null) ? c : d; |

Obviamente poderíamos reescrever usando o operador ternário:



|  |  |
| --- | --- |
| 1  2  3 | object cache = null;  object d = new object();  object e = (c != null) ? c : d; |

Porém, usando o operador null-coalesce, conseguimos encurtar ainda mais o código (se o objeto do lado esquerdo é null, então ele retornará o valor do lado direito):



|  |  |
| --- | --- |
| 1  2  3 | object cache = null;  object d = new object();  object e = c ?? d; |

**3. Inicializadores de Objetos**

Cansado de perder várias linhas de código para ficar setando valores de propriedades? Ou então cansado de ficar escrevendo dezenas de sobrecargas de construtores? Os construtores auto-implementados que surgiram na versão 3.0 do C# resolvem seu problema e de quebra ainda melhoram a legibilidade de seu código. Primeiro vamos ver o jeito tradicional de inicializar um objeto:



|  |  |
| --- | --- |
| 1  2  3 | Cliente c = new Cliente();  c.Nome = "João";  c.Endereco = "Rua Lima 204"; |

E agora usando os inicializadores de objeto:



|  |  |
| --- | --- |
| 1 | Cliente c = new Cliente { Nome="João", Endereco = "Rua Lima 204" }; |

**4. A diretiva ‘using’**

Uma vez que você precisa alocar um objeto em memória um pouco mais pesado que o normal, como streams e arquivos de mídia, é importante que seja feito um controle mais rigoroso na hora de liberar a memória novamente, evitando problemas futuros. Ou seja, sempre temos de fazer três etapas: criamos o objeto, usamos ele e o descartamos. Isso pode ser ilustrado com o trecho de código abaixo:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | // 1. Alocando o objeto  Font font1 = new Font("Arial", 10.0f);  try  {  // 2. Usando o recurso  }  finally  {  // 3.Descartando o objeto  if (font1 != null)  ((IDisposable)font1).Dispose();  } |

A diretiva using permite comprimir o código anterior em:



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | // alocando o recurso  using (Font font1 = new Font("Arial", 10.0f))  {      // Usando o recurso  }  // Descarte automático |

A diretiva using somente pode ser utilizada em objetos que implementem a interface IDisposable.

**5. Apelidos para nomes de classes e namespaces muito grandes**

Nomes em C# podem se tornar muito longos. Se você está desenvolvendo algo para Microsoft Office, pode estar querendo abrir um processo do Word para criar um documento, o que cria objetos com namespaces enormes. O código abaixo, usando a diretiva using para criar um apelido de namespace ilustra outro uso deste canivete suíço da programação C#:



|  |  |
| --- | --- |
| 1  2  3 | using Word = Microsoft.Office.Interop.Word;  ...  Word.Application = new Word.Application() { Visible = True; } |

**6. Tipos Nullables**

Algumas vezes é necessário que suas variáveis possam conter valores nulos, mesmo quando é um simples inteiro. Isto ocorre com mais frequência quanto trabalhos com mapeamento objeto-relacional, pois na maioria dos bancos de dados sempre teremos tabelas com colunas que podem conter valores nulos. Para tornar um variável comum em uma variável que aceita null, basta adicionar um ‘?’ ao final do tipo e pronto!



|  |  |
| --- | --- |
| 1  2 | Nullable x = null;  int? x = null; |

Os nullable-types surgiram na versão 2.0 do framework e podem ser invocados das duas formas exibidas anteriormente.

**7. Propriedades Automáticas**

No C# 3.0 ganhamos as propriedades automáticas. Geralmente a maioria das propriedades das classes são apenas exposições de atributos privados ao restante do sistema. Dificilmente elas possuem alguma lógica ou verificação que realmente exija a criação de um atributo privado e de uma propriedade pública. Desta forma, para que ficar repetindo gets e sets sempre iguais, retornando o valor de um atributo privado?

Sendo assim, as propriedades automáticas vieram para agilizar a tarefa de criação de propriedades que nada mais fazem do que retornar ou receber valores. Vale salientar que em tempo de compilação vão ser gerados atributos privados para onde os valores dessas propriedades ficarão armazenados. Ou seja, não muda nada para o sistema, apenas para o programador.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public class Pessoa{  private string \_primeiroNome;  public string PrimeiroNome  {      get { return \_primeiroNome; }      set { \_primeiroNome= value; }  }  } |

O bloco de código anterior mostra o jeito antigo de fazer as coisas. O bloco seguinte mostra as propriedades automáticas:



|  |  |
| --- | --- |
| 1  2  3  4 | public class Pessoa  {  public string PrimeiroNome { get; set; }  } |

Para quem acha que criar propriedades automáticas públicas é a mesma coisa que criar atributos públicos, está redondamente enganado. O método DataBind presente na maioria dos controles de exibição de dados somente efetua o Bind sobre propriedades públicas e nunca sobre atributos de classe. Desta forma, se pretende que sua classe possa ser exibida em controles deste tipo, é bom usar as propriedades corretamente.

**8. Inferência de Tipos**

Esta é outra dica que deve ser usada com cuidado. Eu particularmente uso apenas quando estou trabalhando com namespaces desconhecidos ou com LINQ e Lambda Expressions. Sendo o C# uma linguagem fortemente tipada, é natural que aja toda uma preocupação com escolher o melhor tipo para as variáveis e objetos que serão utilizadas. Da mesma forma, é importante conhecer os diversos tipos existentes quando se está manipulando métodos de terceiros. A inferência de tipos permite que você deixe de escrever o tradicional código:



|  |  |
| --- | --- |
| 1 | string myString = “Hello World”; |

Desta forma:



|  |  |
| --- | --- |
| 1 | var myString = “Hello World”; |

Peraí, igual no Javascript? NOT!

Mesmo com a inferência de tipos C# ainda é uma linguagem fortemente tipada e uma vez que a inferência esteja concluída (i.e. a variável recebeu um valor) o tipo de seu conteúdo jamais poderá ser alterado. Novamente, isto é apenas a nível de programação. Quando o compilador entra em ação, ele atribui os tipos corretos conforme a necessidade, nos poupando do trabalho de decorar os mais variados tipos e retornos de chamadas de métodos. O objeto do tipo inferido pode perfeitamente ser utilizado como um objeto normal, como o código a seguir nos mostra:



|  |  |
| --- | --- |
| 1  2  3 | var elencoSenior = Empregados.Where(s => s.Idade > 35);  foreach(var membro in elencoSenior)             Console.WriteLine("Nome: {0} Idade: {1}",membro.Nome, membro.Idade); |

Mas então, qual o tipo de elencoSenior?



|  |  |
| --- | --- |
| 1 | ((System.Linq.Enumerable.Iterator<<>f\_\_AnonymousType0>)(SeniorStaff)) |

Bizarro não? Já imaginou ter de trabalhar com este tipo maluco? Muito mais fácil com inferência de tipos.

**9. Expressões Lambda**

Expressões Lambda são expressões de consulta feitas sobre coleções de dados de forma extremamente otimizada e simples, criando um poderosa ferramenta de produtividade. Por exemplo, se tivéssemos uma lista de 30 funcionários e quiséssemos saber todos os funcionários que possuem mais de 35 anos?

Tradicionalmente teríamos de fazer um laço para percorrer todos os elementos da lista comparando a idade de cada elemento e adicionando-os em uma outra fila à medida em que forem encontrados os funcionários mais velhos. Com Lambda, tal tarefa se restringe somente a:



|  |  |
| --- | --- |
| 1 | var elencoSenior = Empregados.Where(s => s.Idade > 35); |

**10. string.IsNullOrEmpty()**

Não necessariamente uma feature da linguagem, mas um método estático muito útil da classe string. Me impressiono como existem pessoas que não conhecem esta poderosa ferramenta para testar se strings estão nulas (null) ou vazias (“”). Segue exemplo de código;



|  |  |
| --- | --- |
| 1  2  3  4 | if (String.IsNullOrEmpty(s) == true)      return "é nula ou vazia";  else       return String.Format("(\"{0}\") é nula ou vazia.", s); |

**11. Use os atalhos do Visual Studio**

Sim, nada é mais produtivo do que saber usar sua IDE ao máximo e atalhos são uma boa parte disso. Seguem os que mais utilizo:

* **Ctrl + K + D:** identa todo o seu código automaticamente;
* **Ctrl + K + C:** comenta um bloco de linhas;
* **Ctrl + K + U:** descomenta um bloco de linhas;
* **Ctrl + Shift + Arrow Up:** durante a programação, volta um método que você estava navegando;
* **Ctrl + Shift + Arrow Down:** durante a programação, avança um método que você estava navegando;
* **Ctrl + .:** implementa o método/classe inexistente que você acabou de escrever;
* **Ctrl + M + O:** esconde todas as regions da classe atual;
* **Ctrl + Shift + B:** compila todos os projetos;
* **F5:** manda depurar um projeto;
* **Ctrl + F5:** manda executar um projeto;
* **F10:** durante a depuração, avança uma linha;
* **F11:** durante a depuração, avança uma linha entrando nos detalhes da mesma;

**12. Use os snippets de código**

Snippets são palavras encurtadas que quando usamos no Visual Studio (e outras IDEs), seguidas de um TAB, elas se transformam em estruturas de código completas. Experimente digitar os seguintes snippets seguido de TAB:

* for: monta um laço for padrão;
* if: monta uma estrutura if padrão;
* ctor: monta um construtor vazio para a classe atual;
* cw: chama um Console.WriteLine();
* do: cria um bloco do/while;
* equals: sobrescreve o método equals de Object;
* foreach: monta um laço foreach padrão;
* try: cria um bloco try/catch padrão;
* while: cria um bloco while padrão;

**13. Estude essa lista do StackOverflow**

Sinceramente, uma hora que eu estiver com mais paciência eu dou um resumida nos principais pontos que tem lá no StackOverflow para aumentar sua produtividade com C#. Por enquanto dá uma olhada nesta lista completíssima que o pessoal reuniu por lá, tem muita coisa boa: <http://stackoverflow.com/questions/9033/hidden-features-of-c>