**Array, ArrayList e List(T): O que devemos saber?**

**Por Jacimar Tavares (**[**http://www.linhadecodigo.com.br/artigo/2767/array-arraylist-e-listt-o-que-devemos-saber.aspx**](http://www.linhadecodigo.com.br/artigo/2767/array-arraylist-e-listt-o-que-devemos-saber.aspx)**)**

Muitos desenvolvedores quando começam a migrar para alguma linguagem da plataforma .Net, seja ela C# ou VB, se deparam com algumas dificuldades, quando o assunto é Vetor. Muitos acabam não conseguindo encontrar as respostas que procuram, como por exemplo: Qual a diferença entre Array, ArrayList e List(T) ? Quando eu devo usar um e não o outro?

O objetivo deste artigo é tratar de forma simples, prática e objetiva essas diferenças, apresentando exemplos para melhor entendimento.

**Entendo Arrays**

A Microsoft, através da **FCL** – **Framework Class Library** permite que o desenvolvedor crie Arrays, que são coleções de dados de um mesmo tipo. Uma característica dos Arrays é que eles são estáticos, ou seja, o programador não consegue aumentar a quantidade de posições em tempo de execução. Um Array declarado ocupa posições adjacentes na memória RAM, com o mesmo nome e tipo. Na hierarquia de classes da **FCL**, podemos notar que um array herda da classe **Array**, do **namespace** **System** (**System. Array**), logo para podermos trabalhar com eles, basta incluir a referência para o **namespace** **System** (**using** **System**;), o que por padrão já é feito.

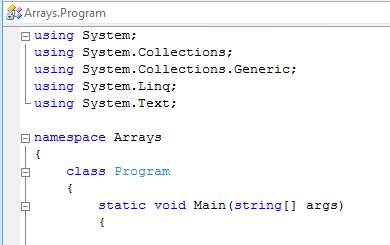


Figura 1 -referência ao namespace System.

**Criando Arrays**

A sintaxe básica de um Array é:

TipoDeDado[] vetorNumeros = new TipoDeDado [TamanhoArray];

Exemplos:

Int32[] vetorNumeros = new Int32 [10]; // Array de Inteiros

String[] vetorNomes = new String [10]; // Array de Strings

Para adicionar valores ao **Array**, devemos escolher a posição através do índice. Veja como fica a atribuição no **Array** de Inteiros.

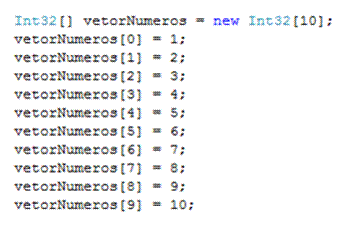


Figura 2 –Array de inteiros.

Para imprimir na tela os valores de cada posição do **Array**, usamos um **for**:

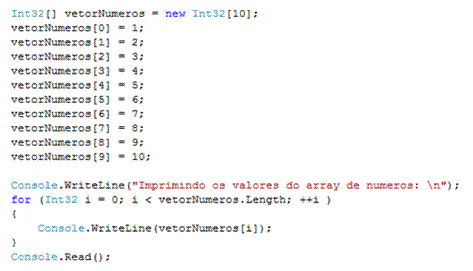


Figura 3 –Varrendo o Array com um for.

Antes de optar por usar um Array, devemos analisar se ele é a melhor opção para resolver o problema que estamos modelando, levando em conta as suas limitações, como tamanho estático.

**Mas preciso de uma Coleção Dinâmica, e agora?**

Dada a limitação dos Arrays, há casos que precisamos usar coleções que permitam que o usuário possa inserir a quantidade de dados que ele desejar, sem que nos preocupemos com o tamanho do **Array**. Os **ArrayLists** são coleções de dados que podem aumentar e diminuir dinamicamente, além de permitir a inserção de qualquer tipo de dado, no mesmo **ArrayList** (inteiros, Strings, Decimais e etc). Um **ArrayList** herda da classe **ArrayList**, do **namespace** **System**.**Collections**; logo para podermos trabalhar com eles, basta incluir a referencia para esse **namespace**.

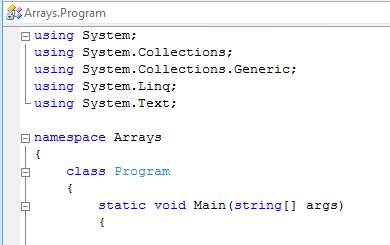


Figura 4 -referência ao namespace System.Collections

**Criando ArrayList**

A sintaxe básica de um **ArrayList** é:

ArrayList nomedoArrayList = new ArrayList();

Para adicionar dados ao **ArrayList** criado, devemos fazer como segue:

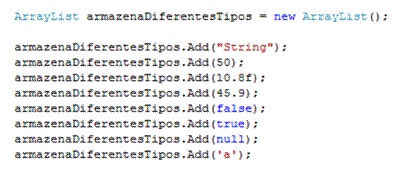


Figura 5 –Inserindo dados no ArrayList recém-declarado

Como dissemos, um **ArrayList** declarado herda da classe **ArrayList**, a qual possui vários métodos. Um deles é o método **Add**, que permite inserir dados no **ArrayList**. Note que no **ArrayList** da figura 5 estamos inserindo dados de vários tipos.

**Imprimindo os valores desse ArrayList:**

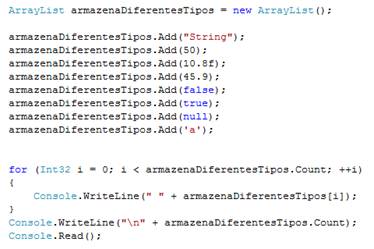


Figura 6 –Varrendo o ArrayList. Imprime o tamanho no final.

A diferença básica do **Array** para o **ArrayList** é quanto a capacidade de armazenar dados. Um não permite aumento em Runtime, já o outro sim. Neste caso, a escolha gira em torno dessa característica.

**Nota**: existem várias outras diferenças, mas o objetivo aqui é apresentar situações de uso para o desenvolvedor, deixando informações técnicas em segundo plano.

**Conhecendo o List(T)**

**Pergunta natural**: O que o **List(T)** pode me oferecer já que agora conheço coleções estáticas e dinâmicas?

O **List(T)** é uma coleção que possui a mesma característica dos **ArrayLists**, que é a de ser dinâmica, e a característica dos **Arrays** que é de permitir um único tipo de dado.

**Vantagem sobre os Arrays:**

* Pode ser aumentado dinamicamente.

**Vantagem sobre o ArrayList.**

* Permitir apenas um tipo de dado.

Esta vantagem merece detalhes: Um **ArrayList** por permitir qualquer tipo de dado, sofre do problema de Boxing e Unboxing. Quando um item é inserido no **ArrayList**, (boxing) ele é armazenado como tipo **object** automaticamente. Quando o usuário retira do **ArrayList** o dado (unboxing) ele é novamente convertido para o tipo que era antes de ir para o **ArrayList**. Essa conversões são internas ao **ArrayList**, o que diminui o seu desempenho em relação aos **List(T)**, que não sofrem desse problema, já que em suas posições estão dados de um único tipo, dispensando essa análise.

O **List(T)** então se torna uma ótima opção, quando precisamos de coleções dinâmicas, e eficientes.

Os **List(T)** são coleções que herdam da classe **List(T)**, do **namespace**: **System.Collections.Generics** , portanto para trabalhar com **List(T)**, devemos incluir a referência ao **namespace** genérico, como já mostramos nas figuras 1 e 4.

**Criando List(T)**

A sintaxe básica de um List(T) é:

List<tipodedado> nomedoList = new List<tipodedado>();

Para adicionar dados ao **List(T)** criado, devemos fazer como segue:

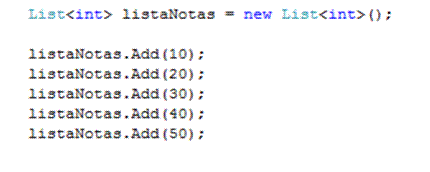


Figura 7 –Inserindo dados no List(T) recém-criado

**Imprimindo os dados do List(T):**

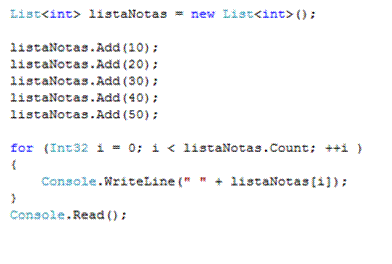


Figura 8 –Varrendo o List(T) com um for.

Cientes de todas essas particularidades, agora você desenvolvedor saberá optar por uma coleção ou outra, dado uma determinada situação. É importante ressaltar que a melhor coleção não é a mais fácil de escrever, e sim a que se encaixa melhor no contexto do nosso problema.