

Elys Sanntos

**C# Collections parte 1: Listas, arrays, listas ligadas, dicionários e conjuntos**

SUMÁRIO

[Introdução 3](#_Toc61511219)

[**Declarando e Populando Arrays** 4](#_Toc61511220)

[**xxxxxxxx** 9](#_Toc61511221)

[Principais comandos 9](#_Toc61511222)

[Definir pontos de interrupção no código-fonte 9](#_Toc61511223)

[Conflitos 10](#_Toc61511224)

[Conclusão 10](#_Toc61511225)



# Introdução

#### **C# Collections parte 1: Listas, arrays, listas ligadas, dicionários e conjuntos**

* Aprenda os fundamentos das coleções C# através dos arrays.
* Explore o poder e flexibilidade das listas.
* Aprenda a trabalhar com conjuntos.
* Comece a usar dicionários para ler e gravar valores facilmente a partir de chaves.
* Entenda como funciona uma lista ligada e como construí-la através dos seus nós.
* Aplique em seus projetos coleções especializadas com prioridade de saída, através de pilhas e filas.

#### Começando com Arrays

Faremos uma introdução acerca dos assuntos que serão abordados:

Começando pelo tipo de coleção mais básica e primitiva do .NET Framework, as **matrizes**, ou ***arrays***. Aprenderemos a modificar e acessá-los por meio do índice, e a copiar e cloná-los também.

Em seguida, iremos explorar o poder das **listas**, os **arrays dinâmicos**, com atenção especial por se tratar de coleções poderosas.

Veremos também como **ordenar uma lista tendo por base elementos de uma classe customizada**, preparando esta lista para que os elementos sejam comparáveis com outros elementos.

Criaremos objetos que expõem coleções do tipo "Somente leitura" para impedir que suas informações sejam manipuladas externamente a este objeto.

Aprenderemos a copiar e clonar listas, totalizando os valores contidos nos objetos da coleção. Veremos como trabalhar com conjuntos no .NET Framework (objetos baseados no conceito matemático de conjuntos).

Aprenderemos a lidar com ***hashing***, e como implementar os métodos necessários para a definição deste espalhamento.

Para ler ou gravar valores facilmente a partir de uma chave, aprenderemos sobre a **coleção dicionário**, a **lista ligada** (*linked list*), e como seus nós se conectam durante a criação desta lista.

Também veremos como trabalhar com a coleção do tipo **pilha**.

# **Declarando e Populando Arrays**

Começaremos pelo tipo mais básico de coleção existente no .NET Framework, o **Array**. Supondo que temos 3 aulas a serem colocadas em uma coleção, representadas por 3 strings diferentes, vamos declará-las, armazenando-as em uma coleção (array).

Faremos isto em uma sintaxe especial, colocando inicialmente o tipo do elemento a ser contido no array, que no caso será uma string. Há duas maneiras de se declarar um array, e uma delas é quando já sabemos quais elementos ele irá conter.

Neste caso, sabemos que haverá 3 elementos, as três aulas, então poderemos simplesmente colocar o valor das três variáveis:

namespace CSharpCollections

{

class Program

{

static void Main(string[] args)

{

string aulaIntro = "Introdução às Coleções";

string aulaModelando = "Modelando a Classe Aula";

string aulaSets = "Trabalhando com Conjuntos";

string[] aulas = new string[]

{

aulaIntro,

aulaModelando,

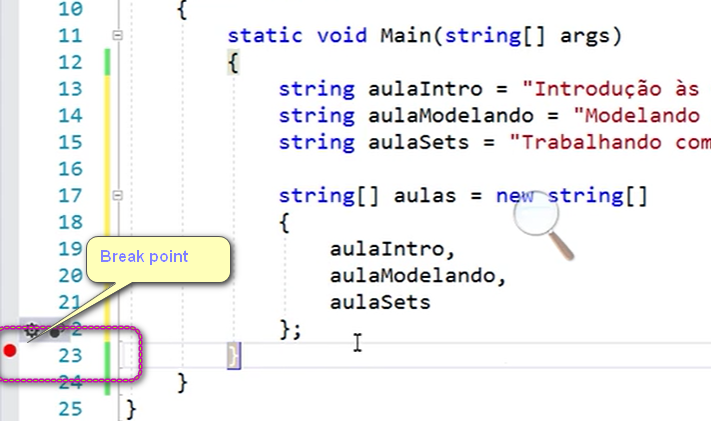
aulaSets

};

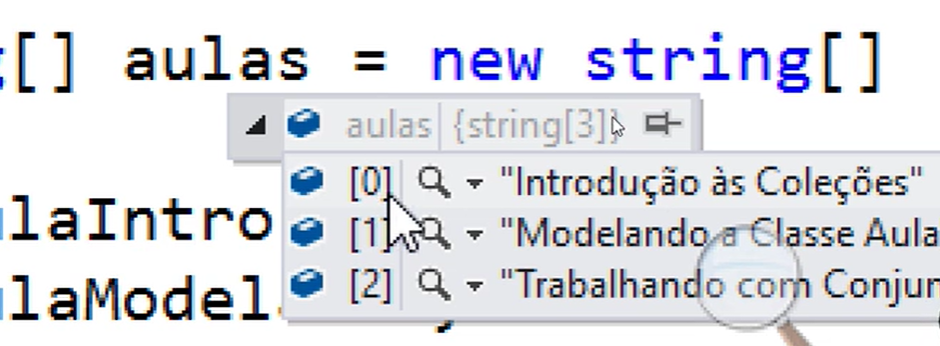
}

}

}

Vamos rodar a aplicação colocando um *breakpoint (*[*Veja aqui a documentação sobre BreakPoint*](https://docs.microsoft.com/pt-br/visualstudio/debugger/using-breakpoints?view=vs-2019%23:~:text=Voc%C3%AA%20tamb%C3%A9m%20pode%20selecionar%20a,interrup%C3%A7%C3%A3o%20%3E%20Inserir%20ponto%20de%20interrup%C3%A7%C3%A3o)*.)* na linha que fecha as strings. Veremos que em aulas há três elementos ("Introdução às Coleções", "Modelando a Classe Aula" e "Trabalhando com Conjuntos"), como gostaríamos.

O primeiro deles começa no índice 0, o segundo no 1 e o último no 2.

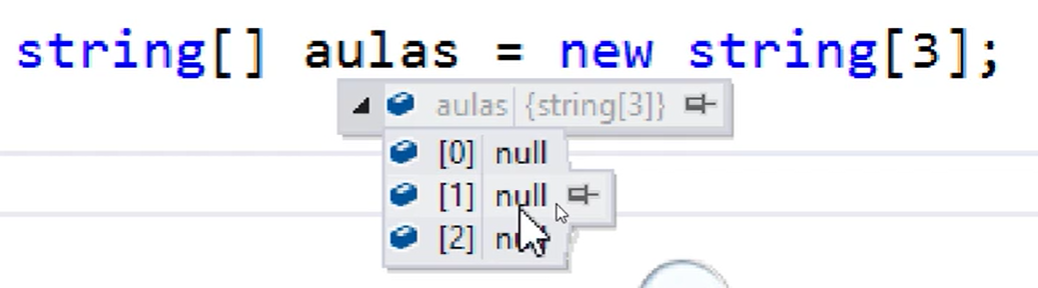
Ou seja, **o índice de um elemento será igual à sua posição menos um**. Esta é a primeira forma de declararmos um array no caso de já sabermos de antemão quais elementos ele irá conter.

A outra forma de declararmos um array consiste em incluirmos o tamanho que ele irá ocupar, algo que foi declarado implicitamente na primeira forma. Desta vez o faremos explicitamente.

É importante lembrarmos que um array é uma coleção de tamanho fixo!

string[] aulas = new string[3];

Rodando a app desta forma, veremos que em aulas estão armazenadas três posições, cada uma com valor null (nulo), e estes elementos ainda são desconhecidos, porém já temos o array declarado.

Completaremos o código:

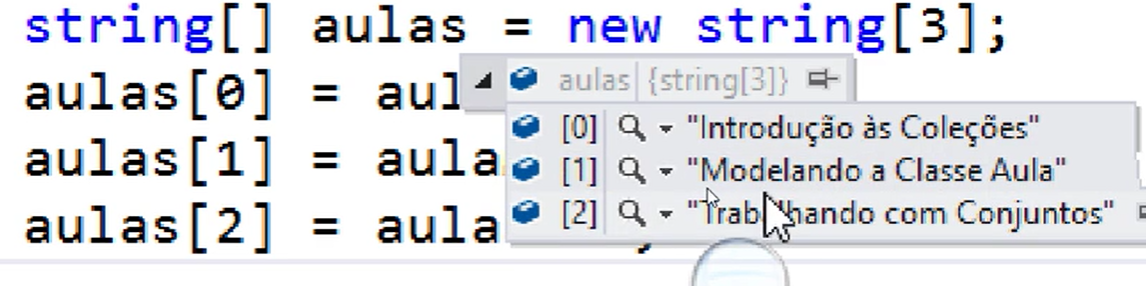
string[] aulas = new string[3];

aulas[0] = aulaIntro;

aulas[1] = aulaModelando;

aulas[2] = aulaSets;

Vamos rodar o código mais uma vez. Veremos que o array aulas possui os três elementos preenchidos. Imprimiremos isto na tela



acrescentando Console.WriteLine(); - que também pode ser feito com o atalho "cw + Tab + Tab":

string[] aulas = new string[3];

aulas[0] = aulaIntro;

aulas[1] = aulaModelando;

aulas[2] = aulaSets;

Console.WriteLine(aulas);

Ao rodarmos a aplicação para verificar o resultado, temos impresso na tela System.String[], o que não é exatamente o que queríamos. Queremos imprimir os elementos do array de aulas e, para isto, varreremos esta coleção (aulas) e imprimiremos cada um dos itens.

Uma das maneiras de se fazer isso é utilizar a instrução foreach para informar que, para cada aula em aulas, se faça algo (que no caso é a impressão).

string[] aulas = new string[3];

aulas[0] = aulaIntro;

aulas[1] = aulaModelando;

aulas[2] = aulaSets;

Console.WriteLine(aulas);

foreach (var aula in aulas)

{

Console.WriteLine(aula);

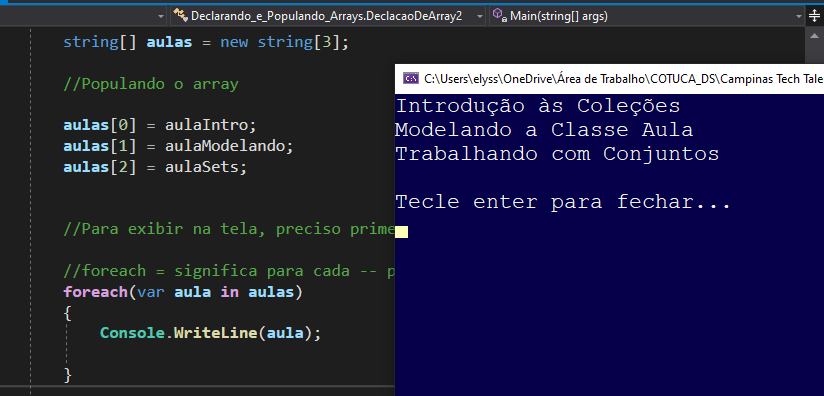
}

Rodando novamente o código, obteremos como *Output* o seguinte:

Introdução às Coleções

Modelando a Classe Aula

Trabalhando com Conjuntos



Pegaremos esta instrução e a jogaremos a outro método para a reutilização conforme nossa necessidade. Utilizando "Ctrl + . (ponto)" extrairemos o método denominado Imprimir(). Nele, teremos nosso laço foreach.

Como segunda forma de imprimirmos estes elementos, utilizaremos outro laço o for que varrerá o comprimento do nosso array de um índice até a posição final. Comentaremos o código do laço foreach

{

// strings omitidos

Console.WriteLine(aulas);

Imprimir(aulas);

}

private static void Imprimir(string[] aulas)

{

//foreach (var aula in aulas)

//{

// Console.WriteLine(aula);

//}

for (int i = 0; i < aulas.Length; i++)

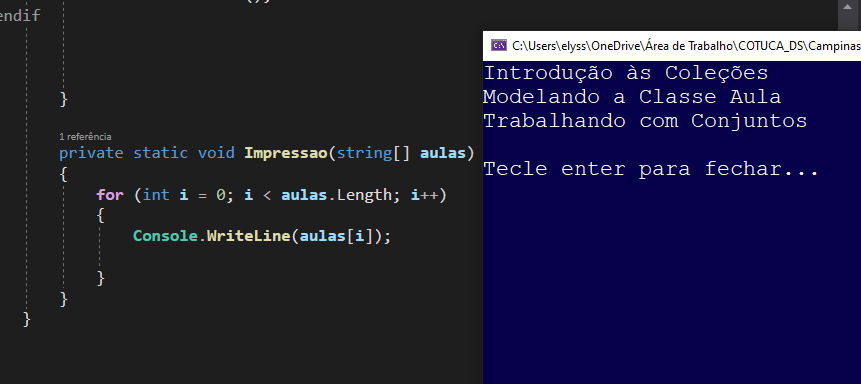
{

Console.WriteLine(aulas[i]);

}

}

Feito isto, podemos rodar a aplicação mais uma vez. O resultado é exatamente o mesmo que o anterior, embora a forma de chegarmos a ele seja outra.



Com o laço foreach não precisávamos de muitas informações, bastando passar o nome do array. Com o for, conseguimos controlar o avanço do índice. Neste caso, é necessário sabermos seu comprimento e o índice que está sendo impresso.

Para imprimirmos o primeiro elemento do nosso array, usaremos Console.WriteLine(); novamente, acessando este elemento, cujo índice é 0.

{

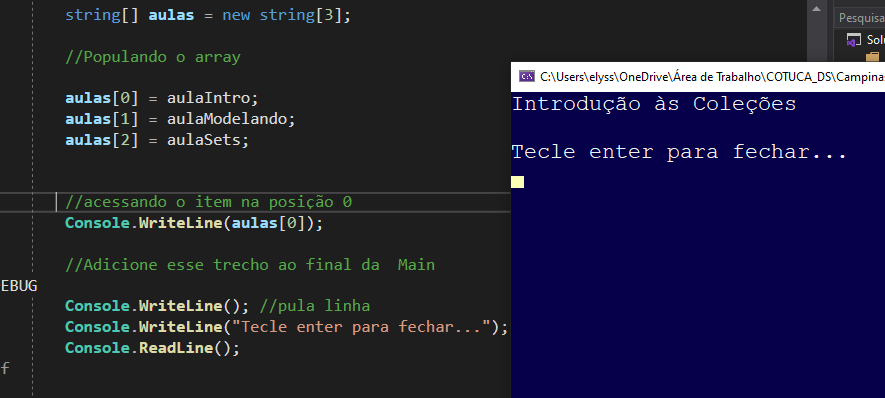
// strings omitidos

Console.WriteLine(aulas);

Imprimir(aulas);

Console.WriteLine(aulas[0]);

}



Rodando a app, veremos a impressão de "Introdução às Coleções" na tela. Para acessarmos o último elemento, precisaremos realizar uma conta:

{

// strings omitidos

Console.WriteLine(aulas);

Imprimir(aulas);

Console.WriteLine(aulas[0]);

Console.WriteLine(aulas[aulas.Length - 1]);

}

Verificaremos que tudo funciona conforme esperado, quando rodamos a app vemos "Trabalhando com Conjuntos" no Output. Se quisermos trocar um elemento deste array, como faremos?

{

// strings omitidos

Console.WriteLine(aulas);

Imprimir(aulas);

Console.WriteLine(aulas[0]);

Console.WriteLine(aulas[aulas.Length - 1]);

aulaIntro = "Trabalhando com Arrays";

Imprimir(aulas);

}

Rodaremos a aplicação e não teremos nenhuma mudança! Isso ocorre pois aulaIntro não está sendo armazenado no nosso array de aulas. O que armazenamos ali foi o **valor copiado de** aulaIntro, portanto não adianta mudarmos seu valor.

Precisaremos acessar diretamente a primeira posição do nosso array de aulas. Para atribuirmos um valor a esta posição, de índice 0, deixaremos o código assim:

{

// strings omitidos

Console.WriteLine(aulas);

Imprimir(aulas);

Console.WriteLine(aulas[0]);

Console.WriteLine(aulas[aulas.Length - 1]);

aulaIntro = "Trabalhando com Arrays";

Imprimir(aulas);

aulas[0] = "Trabalhando com Arrays";

Imprimir(aulas);

}COPIAR CÓDIGO

Rodaremos a aplicação e veremos que conseguimos trocar com sucesso o primeiro elemento do nosso array.

# **xxxxxxxx**

#### Lista, a Coleção Flexível

#### O Poder dos Sets

#### Dicionários, Trabalhando com Chave e Valor

#### Lista Ligada, Pilha e Fila

#### Arrays e Tipos genéricos

#### Collections

#### Qual Coleção Usar

# Principais comandos

|  |  |
| --- | --- |
| Definir pontos de interrupção no código-fonte | clique na margem da extrema esquerda ao lado de uma linha de código. Você também pode selecionar a linha e pressionar **F9**, selecionar > **ponto de interrupção de alternância** de depuração, ou clicar com o botão direito do mouse e selecionar **ponto de** interrupção > **Inserir ponto** de interrupção. O ponto de interrupção aparece como vermelho na margem esquerda. |
| Cw + tab + tab | Atalho para digitar Console.WriteLine() |
| Ctrl . | \*\*Selecione o trecho de código que se tornará um metodo\*\* Para extrair o metodo |
| WriteLine(); -- ,\n | Para pular linha |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Conflitos

# Conclusão