Instrutor: Alexandre Rech
Empresa: NDDigital Technologies

# Unidade 3

## *3 Um pouco de arrays*

Ao término desse capítulo, você será capaz de:

- Declarar e instanciar arrays;
- Popular e percorrer arrays.

### 3.1 - O problema

Dentro de um bloco, podemos declarar variáveis e usá-las.

```
int idade1;
int idade2;
int idade3;
int idade4;
```

Mas também podemos declarar um **vetor (array)** de inteiros:

```
int[] idades;
```

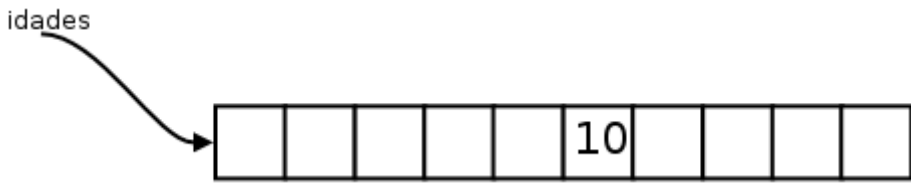
O int[] é um tipo. Uma array é sempre um objeto, portanto, a variável idades é uma referência. Vamos precisar criar um objeto para poder usar a array. Como criamos o objeto-array?

```
idades = new int[10];
```

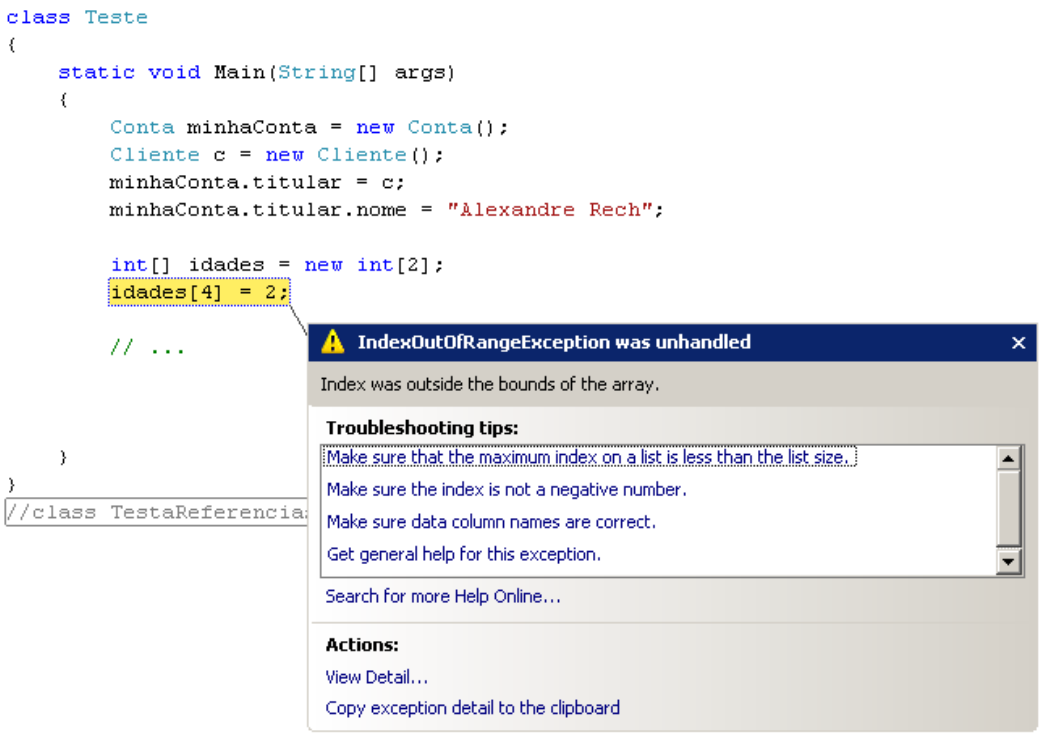
Aqui, o que fizemos foi criar uma array de int de 10 posições, e atribuir o endereço no qual ela foi criada.

Agora, podemos acessar as posições do array.

```
idades[5] = 10;
```




O código acima altera a sexta posição do array. No C#, os índices do array vão de 0 a n-1, onde n é o tamanho dado no momento em que você criou o array. Se você tentar acessar uma posição fora desse alcance, um erro ocorrerá durante a execução.




**Arrays – um problema no aprendizado de muitas linguagens**

Aprender a usar arrays pode ser um problema em qualquer linguagem. Isso porque envolve uma série de conceitos, sintaxe e outros. No C#, muitas vezes utilizamos outros recursos em vez de arrays, em especial os pacotes de coleções do C#, que veremos nos capítulos a frente. Portanto, fique tranqüilo caso não consiga digerir toda sintaxe das arrays num primeiro momento.

## 3.2 - Arrays de referências

É comum ouvirmos “array de objetos”. Porém quando criamos uma array de alguma classe, ela possui referências.

O objeto, como sempre, está na memória principal e, na sua array, só ficam guardadas as **referências** (endereços).

```
Conta[] minhasContas;
minhasContas = new Conta[10];
```

Quantas contas foram criadas aqui? Na verdade, **nenhuma**. Foram criados 10 espaços que você pode utilizar para guardar uma referência a uma Conta. Por enquanto, eles se referenciam para lugar nenhum (null).

Se você tentar:

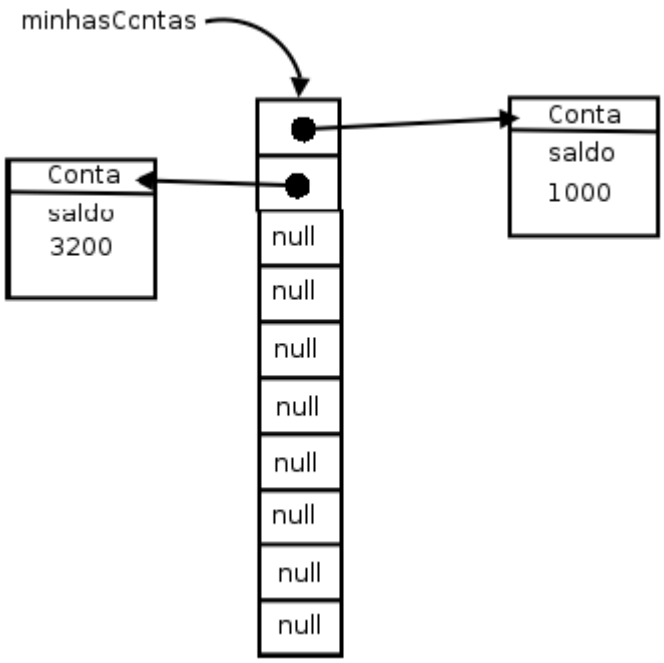
```
Console.WriteLine(minhasContas[0].saldo);
```

Um erro durante a execução ocorrerá! Pois, na primeira posição da array, não há uma referência para uma conta, nem para lugar nenhum. Você deve **popular** sua array antes.

```
Conta contaNova = new Conta();
contaNova.saldo = 1000.0;
minhasContas[0] = contaNova;
```

Ou você pode fazer isso diretamente:

```
minhasContas[1] = new Conta();
minhasContas[1].saldo = 3200.0;
```




Uma array de tipos primitivos guarda valores, uma array de objetos guarda referências.

## 3.3 - Percorrendo uma array

Percorrer uma array é muito simples quando fomos nós que a criamos:

```
static void Main(String[] args)
{
	int[] idades = new int[10];
	for (int i = 0; i < 10; i++)
	{
		idades[i] = i * 10;
	}
	for (int i = 0; i < 10; i++)
	{
		Console.WriteLine(idades[i]);
	}
}
```

Porém, em muitos casos, recebemos uma array como argumento em um método:

```
void imprimeArray(int[] array)
{
	// não compila!!
	for (int i = 0; i < ????; i++)
	{
		Console.WriteLine(array[i]);
	}
}
```

Até onde o for deve ir? Toda array em C# tem um atributo que se chama length, e você pode acessá-lo para saber o tamanho do array ao qual você está se referenciando naquele momento:

```
void imprimeArray(int[] array)
{
	for (int i = 0; i < array.Length; i++)
	{
		Console.WriteLine(array[i]);
	}
}
```

**Arrays não podem mudar de tamanho**

A partir do momento que uma array foi criada, ela **não pode** mudar de tamanho.

Se você precisar de mais espaço, será necessário criar uma nova array e, antes de se referir ela, copie os elementos da array velha.

## 3.4 - Percorrendo uma array no C#

O C# traz uma sintaxe para percorremos arrays (e coleções, que veremos mais a frente).

No caso de você não ter necessidade de manter uma variável com o índice que indica a posição do elemento no vetor, podemos usar `foreach`.

```
static void Main(String[] args)
{
	int[] idades = new int[10];

	foreach (int idade in idades)
	{
		Console.WriteLine(idade);
	}
}
```

## 3.5 - Exercícios: Arrays

1) Volte ao nosso sistema de Funcionario e crie uma classe Empresa. A Empresa tem um nome, cnpj e uma array de Funcionario, além de outros atributos que você julgar necessário.

```
class Empresa
{
	// outros atributos
	Funcionario[] funcionarios;
	String cnpj;
}
```

2) A empresa deve ter um método adiciona, que recebe uma referência a Funcionario como argumento, e guarda esse funcionário. Algo como:

```
void adiciona(Funcionario f)
{
	// algo tipo: this.funcionarios[ ??? ] = f
	// mas que posição colocar?
}
```

Você deve inserir o Funcionario em uma posição da array que esteja livre. Existem várias maneiras para você fazer isso: guardar um contador para indicar qual a próxima posição vazia ou procurar por uma posição vazia toda vez. O que seria mais interessante?

É importante reparar que o método adiciona não recebe nome, rg, salário, etc. Essa seria uma maneira nem um pouco estruturada, muito menos orientada a objetos de

se trabalhar. Você antes cria um Funcionario e já passa a referência dele, que dentro do objeto possui rg, salário, etc.

3) Crie uma outra classe que possuirá o seu método Main. Dentro dele crie algumas instâncias de Funcionário e passe para a empresa pelo método adiciona. Repare que antes você vai precisar criar a array, pois inicialmente o atributo funcionarios da classe Empresa não referencia lugar nenhum (null):

```
Empresa empresa = new Empresa();
empresa.funcionarios = new Funcionario[10];
```

Ou você pode construir a array dentro da própria declaração da classe Empresa.

Crie alguns funcionários e passe como argumento para o adiciona da empresa:

```
Funcionario f1 = new Funcionario();
f1.salario = 1000;
empresa.Adiciona(f1);
```

Você pode criar esses funcionários dentro de um loop, se preferir.

Opcional: o método adiciona pode gerar uma mensagem de erro indicando que o array está cheio.

4) Percorra o atributo funcionarios da sua instância da Empresa e imprima o salários de todos seus funcionários. Ou você pode chamar o método Mostra() de cada Funcionario da sua array. Use também o foreach do C#.

Cuidado: alguns índices do seu array podem não conter referência para um Funcionario construído, isto é, ainda se referirem para null.

5) (Opcional) Crie um método para verificar se um determinado Funcionario se encontra ou não como funcionário desta empresa:

```
bool Contem(Funcionario f)
{
	// ...
}
```

Você vai precisar fazer um for na sua array e verificar se a referência passada como argumento se encontra dentro da array. Evite ao máximo usar números hard-coded, isto é, use o .length.

6) (Opcional) Caso a array já esteja cheia no momento de adicionar um outro funcionário, criar uma nova maior e copiar os valores. Isto é, fazer a realocação já que C# não tem isso: uma array nasce e morre com o mesmo length.

## 3.7 - Desafios

1) No capítulo anterior, você deve ter reparado que a versão recursiva para o problema de Fibonacci é lenta porque toda hora estamos recalculando valores. Faça com que a versão recursiva seja tão boa quanto a versão iterativa. (Dica: use arrays para isso)

## 3.8 - Testando o conhecimento

1) O objetivo dos exercícios a seguir é fixar os conceitos vistos até agora. Se você está com dificuldade em alguma parte desse capítulo, aproveite e treine tudo o que vimos até agora no pequeno programa abaixo:

Programa:

Classe: Casa - Atributos: cor, totalDePortas, portas[] - Métodos: void Pinta(String s), int QuantasPortasEstaoAbertas(), void AdicionaPorta(Porta p), int TotalDePortas(). Crie uma casa, pinte-a. Crie três portas e coloque-as na casa através do método AdicionaPorta, abra e feche-as como desejar. Utilize o método QuantasPortasEstaoAbertas para imprimir o número de portas abertas e o método TotalDePortas para imprimir o total de portas em sua casa.