

Fundamentos de C#

Arrays



Tópicos Abordados



- Arrays
- Arrays na memória
- Acessando elementos
- Inicializando arrays
- Arrays multidimensionais
- Jagged arrays
- A classe *Array*
- Percorrendo elementos
- Ordenando elementos em um array
- Parâmetros via linha de comando

Arrays



- Arrays são estruturas de dados que agrupam dados do mesmo tipo

```
int[] distancias;  
distancias = new int[8];
```

Array de int com 8 posições

```
double[] notas = new double[5];
```

Array de double com 5 posições

Arrays na Memória

• Arrays são reference types
– Alocados no managed heap

int[] a = new int[3];

Stack

Managed Heap

Os elementos são armazenados numa área contínua de memória

Arrays na Memória

• Arrays também podem conter *reference types*

string[] a = new string[3];

Stack

Managed Heap

Cada elemento referencia algum objeto do managed heap

Acessando Elementos de um Array

• Cada elemento do array tem um índice

double[] notas = new double[8];

notas[3] = 7.5;

notas[5] = 4.0;

double x = notas[6];

x = 0;

O array é inicializado

O array é indexado de 0 ao seu tamanho - 1

Considerações Sobre Arrays



- Os índices do array vão de 0 a $n-1$ (onde n é o tamanho do array)
 - Acessos fora deste intervalo resultam na exceção *IndexOutOfRangeException*
- Não é possível declarar arrays com tamanho negativo

```
int[] array = new int[-5];
```

- Arrays podem ter tamanho 0

```
int[] array = new int[0];
```

Inicializando Arrays



- A inicialização de arrays pode ser feita de várias formas

```
int[] array = new int[5];
```

```
int[] array = new int[2]{ 1, 2 };
```

```
int[] array = new int[] { 1, 2 };
```

```
int[] array = { 1, 2 };
```

- Posições não inicializadas recebem um valor padrão
 - 0 para *value types*
 - null* para *reference types*

Arrays Multidimensionais



- Arrays podem ter uma ou mais dimensões

```
int[,] array = new int[3, 4];
```

2 dimensões, 3x4

	0	1	2	3
0	2	5	3	6
1	4	9	0	9
2	1	7	8	9

```
array[1, 3] = 9;
```

```
int[,] array = {  
    { 2, 5, 3, 6 },  
    { 4, 9, 0, 2 },  
    { 1, 7, 8, 9 }  
};
```

Arrays Multidimensionais

- Mais dimensões também são suportadas

```
int[, ,] array = new int[3, 4, 2];
```

```
array[1, 2, 0] = 9;
```

```
int[, ,] array = {
    {
        { 2, 5, 3, 6 },
        { 4, 9, 0, 2 },
        { 1, 7, 8, 9 }
    },
    {
        { 0, 4, 2, 1 },
        { 6, 9, 8, 0 },
        { 2, 6, 8, 7 }
    }
};
```

3 dimensões, 3x4x2

Acesso ao elemento

Inicialização

Arrays com mais de 3 dimensões são menos utilizados

Jagged Arrays

- São arrays de mais de uma dimensão onde o número de colunas varia para cada linha

```
int[][] array = new int[3][];
```

```
array[0] = new int[2];
```

```
array[1] = new int[4];
```

```
array[2] = new int[1];
```

Inicialização

A Classe *Array*

- Todos os arrays, quando instanciados, herdam de **System.Array**
 - Herdam properties e métodos desta classe

Property	Descrição
<i>Length</i>	Retorna o tamanho da dimensão do array
<i>Rank</i>	Retorna o número de dimensões do array

Método	Descrição
<i>CopyTo()</i>	Copia os dados para outro array
<i>Clear()</i>	Limpa os elementos do array
<i>Reverse()</i>	Inverte os elementos do array
<i>Sort()</i>	Ordena os elementos do array

static

Arrays com a Classe *Array*

- É possível utilizar a classe *Array* diretamente para manipular arrays

```
Array array = Array.CreateInstance(typeof(int), 5);
```

Array de *int* com 5 posições

```
array.SetValue(10, 3);
```

Atribui o valor 10 no índice 3

```
int i = (int)array.GetValue(3);
```

Lê o valor do índice 3

```
int[] array2 = (int[])array;
```

Converte para a notação de arrays

Arrays e a Herança/Implementação

- Um array de um determinado tipo pode conter objetos de subclasses ou implementações deste tipo

```
object[] array = new object[3];
array[0] = 5;
array[1] = "Texto";
array[2] = new StringBuilder();
```

Pode ser qualquer elemento que herde de *object*

Percorrendo Elementos com *for*

- Usando a estrutura *for*

```
int[] array = new int[10];
for (int i = 0; i < array.Length; i++)
{
    int e = array[i];
    //...
}
```

Retorna o tamanho do array

```
int[,] array = new int[10, 5];
for (int i = 0; i < array.GetLength(0); i++)
{
    for (int j = 0; j < array.GetLength(1); j++)
    {
        int e = array[i, j];
        //...
    }
}
```

Retorna o tamanho da dimensão do array

Percorrendo Elementos com *foreach*



- Usando a estrutura *foreach*

```
int[] array = new int[10];  
  
foreach (int e in array)  
{  
    // 'e' é o elemento  
}
```

Itera sobre todos os elementos
de todas as dimensões do array

Ordenando Elementos



- A classe *Array* possui o método **Sort()**
 - Permite ordenar os elementos de um array

```
int[] array = { 5, 2, 4, 6, 1 };  
  
Array.Sort(array);  
  
foreach (int e in array)  
{  
    Console.Write(e + " ");  
}
```

Ordena os elementos




1 2 4 5 6

A Interface *IComparable<T>*



- A ordenação só pode ser feita se o critério de ordenação for especificado
 - Implementação da interface **IComparable<T>**
- Os tipos de dados numéricos e strings implementam esta interface
 - **Números**: ordem crescente
 - **Strings**: ordem alfabética

A Interface *IComparable<T>*



- Para ordenar objetos de classes criadas por você, a interface *IComparable<T>* deve ser implementada


CompareTo() retorna um int

```

class Prova : IComparable<Prova>
{
    public double Nota { get; set; }

    public int CompareTo(Prova other)
    {
        if (Nota == other.Nota)
        {
            return 0;
        }
        else if (Nota > other.Nota)
        {
            return 1;
        }
        else
        {
            return -1;
        }
    }
}
    
```

A Interface *IComparer<T>*




- A interface *IComparer<T>* é uma alternativa à interface *IComparable<T>*

```

class ProvaComparer : IComparer<Prova>
{
    public int Compare(Prova p1, Prova p2)
    {
        if (p1.Nota == p2.Nota)
        {
            return 0;
        }
        else if (p1.Nota > p2.Nota)
        {
            return 1;
        }
        else
        {
            return -1;
        }
    }
}

class Prova
{
    public double Nota { get; set; }
}
    
```

A Interface *IComparer<T>*




- O objeto *IComparer<T>* deve ser fornecido ao método *Sort()* como parâmetro

```

ProvaComparer comparer = new ProvaComparer();
Array.Sort(provas, comparer);
    
```

- O uso de *IComparer<T>* permite utilizar vários critérios de comparação para a mesma classe
 - Com *IComparable<T>* apenas um critério pode ser especificado

Parâmetros Via Linha de Comando



- Um exemplo de uso de arrays é no processamento de parâmetros passados via linha de comando


```
static void Main() { }  
static void Main(string[] args) { }
```

Array com os parâmetros de linha de comando

```
static void Main(string[] args)  
{  
    string param1 = args[0];  
    string param2 = args[1];  
    //...  
}
```

Cada posição do array representa um parâmetro

Parâmetros Via Linha de Comando




- Existe uma forma alternativa de ler os parâmetros passados via linha de comando

```
static void Main()  
{  
    string[] args = Environment.GetCommandLineArgs();  
    string param1 = args[0];  
    string param2 = args[1];  
}
```

O método *Main()* não precisa ser declarado como *Main(string[])*

Desvantagens no Uso de Arrays



- Depois de criado, não é possível modificar o tamanho de um array
- Dificuldade em encontrar elementos dentro do array quando o índice não é conhecido
- Ao remover elementos, sobram “buracos” no array

O uso de coleções de dados pode resolver estes problemas