Fundamentos de C#

Strings e Enumerations

Softblue
cursos online

Tópicos Abordados

- Strings
  - Caracteres de escape
  - Verbatim strings
  - Métodos e properties
  - Strings na memória
  - Strings nulas e vazias
  - A classe *StringBuilder*
  - Conversão de dados
- Enumerations
  - Mapeamentos em enums
  - A classe *System.Enum*

Strings

- Strings são cadeias de caracteres
- Representadas pela classe **System.String** ou pela palavra-chave **string**
- O texto associado a uma string é definido dentro de aspas duplas

```
string text = "abcdef";
```

Criação de uma *string*

Caracteres de Escape

- Alguns caracteres têm um significado especial quando fazem parte de uma string

Caractere	Significado
\n	Nova linha
\t	Tabulação
\"	Aspas
\a	Aviso sonoro
\\	Barra invertida

Caracteres de Escape

- Exemplos

```
string s = "Texto em\nduas linhas";
```
Texto em
duas linhas

```
string s = "Texto entre \"aspas\"";
```
Texto entre “aspas”

```
string s = "Nome:\tMaria";
```
Nome: Maria

```
string s = "C:\\Projetos\\CSharp";
```
C:\Projetos\CSharp

Verbatim Strings

- Strings prefixadas com @
- Desabilita o processamento dos caracteres de escape

```
string s = @"Texto dividido
em três
linhas";
```
Texto dividido
em três
linhas

```
string s = @"Texto entre ""aspas""";
```
Texto entre “aspas”

```
string s = @"C:\Projetos\CSharp";
```
C:\Projetos\CSharp

Métodos e Properties de Strings

- *String* é uma classe
  - Define seus próprios métodos e properties
- *String* herda de *object*
  - Tem acesso aos métodos da sua superclasse

Elemento	Significado
Length	Retorna o tamanho da string
Compare()	Compara duas strings alfabeticamente
Contains()	Verifica se uma string está contida em outra
PadLeft()	Preenche a string com caracteres desejados (à esqueda)
PadRight()	Preenche a string com caracteres desejados (à direita)
Replace()	Substitui caracteres em uma string
Trim()	Remove os espaços em branco iniciais e finais da string
ToUpper()	Converte todos os caracteres da string para maiúsculo
ToLower()	Converte todos os caracteres da string para minúsculo

Métodos e Properties de Strings

- Exemplos

```
string s = "abc";
int length = s.Length;
```
length = 3

```
string s = "123";
s = s.PadLeft(6, '0');
```
s = “00123”

```
string s = "0ba2ba4";
s = s.Replace("ba", "_");
```
s = “0_2_4”

```
string s = "softblue";
s = s.ToUpper();
```
s = “SOFTBLUE”

```
string s1 = "Garbage";
string s2 = "Collector";
int c = s1.CompareTo(s2);
```
c = 1

Concatenação de Strings

- O operador “+” é redefinido na string e permite a concatenação

```
string s1 = "Curso";
string s2 = " de ";
string s3 = "C#";
string concat = s1 + s2 + s3;
```

“Curso de C#”

Internamente, o método *Concat()* é chamado

Igualdade de Strings

- Strings são *reference types*
  - Armazenadas no managed heap
- Os operadores '==' e '!=' podem ser usados para fazer a comparação do valor, e não da referência

```
string s1 = "Texto 1";
string s2 = "Texto 2";
```

```
s1 == s2   False
s1 != s2   True
```

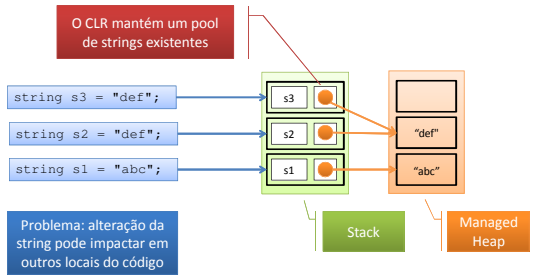
```
string s1 = "Texto";
string s2 = "Texto";
```

```
s1 == s2   True
s1 != s2   False
```

A comparação é *case sensitive*

Strings na Memória

- Strings são *reference types*



Problema: alteração da string pode impactar em outros locais do código

Strings são imutáveis

Strings são Imutáveis

- Quando uma string é criada na memória com um determinado valor, seu valor não pode mais ser alterado
  - Esta abordagem permite uma série de otimizações favorece a segurança
  - Não é possível criar uma subclasse de *string*
    - A classe é definida como *sealed*

Strings são Imutáveis

- Exemplo com *ToUpper()*

```
string s = "abc";
s.ToUpper();
Console.WriteLine(s);
```

abc

A string não é modificada

```
string s1 = "abc";
string s2 = s1.ToUpper();
Console.WriteLine(s2);
```

ABC

Uma nova string é criada

- Exemplo de concatenação

```
string s1 = "ab";
string s2 = "cd";
s1 = s1 + s2;
Console.WriteLine(s1);
```

abcd

Uma terceira string é criada para guardar o resultado da concatenação

Strings Nulas e Vazias

- Uma string é nula quando a variável não referencia nenhum objeto *string* no managed heap

```
string s = null;
```

- Uma string é vazia quando não tem caracteres

```
string s = "";
```

```
string s = string.Empty;
```

- O comportamento é diferente

```
string s = null;
int i = s.Length;
```

NullReferenceException

```
string s = "";
int i = s.Length;
```

i = 0

Strings Nulas e Vazias

- É possível testar se uma string é nula ou vazia

```
string.IsNullOrEmpty(s);
string.IsNullOrWhiteSpace(s);
s != null
s == null
```

Retorna um **bool**

Acessando Caracteres

- O C# permite que os caracteres que formam uma *string* sejam acessados de forma individual
  - Como se fosse um array de *char*

```
string s = "Texto";
char c1 = s[0];
char c2 = s[4];
```

c1 = 'T'
c2 = 'o'

Conversão de Dados

- A conversão de um dado numérico para *string* e vice-versa é muito utilizada na programação
- Os tipos de dados possuem métodos para fazer esta conversão
  - *ToString()*
  - *Parse()*

```
int x = 250;
string s = x.ToString();
```

s = "250"

```
string s = "250";
int x = int.Parse(s);
```

x = 250

- A classe **System.Convert** possui uma série de métodos estáticos que também auxiliam na conversão

A Classe *StringBuilder*

- Realizar muitas modificações em strings pode trazer problemas de performance para a aplicação
  - Strings são imutáveis, portanto cada "mudança" acaba gerando um novo objeto na memória

```
string s = "";
for (int i = 0; i < 100000; i++)
{
    s += i;
}
```

Uma nova string é criada para cada iteração

A Classe *StringBuilder*

- Em situações como esta, utilizar um objeto **StringBuilder** é recomendado
  - Permite manipular strings sem criar novos objetos na memória
  - A classe pertence ao namespace *System.Text*

```
StringBuilder sb = new StringBuilder();
for (int i = 0; i < 100000; i++)
{
    sb.Append(i);
}

string s = sb.ToString();
```

Cria um *StringBuilder*

Concatena sem criar novos objetos

Cria um objeto *string* só no final

Enumerations

- Enumerations (ou *enums*) são úteis para mapear símbolos em números
  - A palavra-chave **enum** é utilizada
  - Um *enum* é um tipo de dado

```
enum Turno
{
    Manha,
    Tarde,
    Noite
}
```

```
Turno t = Turno.Tarde;

if (t == Turno.Tarde)
{
    //...
}
```

Referenciando um *enum*

Elementos que pertencem ao *enum*

Mapeamento em Enums

- O elemento de um enum é sempre mapeado para algum valor numérico
  - Por padrão, os valores são do tipo **int** e iniciam em **0**

Tipo de dado do mapeamento (*byte*, *int*, *short* ou *long*)

```
enum Turno
{
    Manha,
    Tarde,
    Noite
}
```

=

```
enum Turno : int
{
    Manha = 0,
    Tarde = 1,
    Noite = 2
}
```

Valor associado ao elemento

Mapeamento em Enums

- O valor associado pode ser especificado de forma explícita

```
enum Turno
{
    Manha = 10,
    Tarde = 15,
    Noite = 20
}
```

- É possível especificar apenas o primeiro valor

```
enum Turno
{
    Manha = 100,
    Tarde,
    Noite
}
```

Tarde = 101
Noite = 102

A Classe *System.Enum*

- Enums são do tipo **System.Enum**
  - Têm acesso aos elementos desta classe
  - Enums são value types
    - Criados na stack, e não no managed heap

```
string s = Turno.Noite.ToString();
```
s = "Noite"

```
int i = (int) Turno.Tarde;
```
i = 1

```
Turno t = (Turno) Enum.Parse(typeof(Turno), "Noite");
```

```
Array a = Enum.GetValues(typeof(Turno));
```

Nested Enums

- Enums podem ser definidos dentro de classes ou estruturas

```
class Pessoa
{
    public enum Tipo
    {
        Fisica,
        Juridica
    }
}
```

Nested enum

```
Pessoa.Tipo t = Pessoa.Tipo.Fisica;
```

O tipo externo faz parte do nome do enum


Softblue