Programação Web 2

Aula 02 - A linguagem C#

Prof. Fulano de Tal
meu@email.dominio.br

Bacharelado em Sistemas de Informação
Instituto Federal de Sergipe
Campus Lagarto

19 de outubro de 2018

Sumário

Conceitos básicas

Introdução ao C#

- Regras Gerais
  - As instruções devem estar dentro de um escopo e sempre ser finalizadas com um ponto e vírgula;
  - C# é *case-sensitive*;
  - C# é uma linguagem fortemente tipada, ou seja, todas as variáveis e objetos devem ter um tipo declarado;
  - Os tipos podem ser divididos em:
    - **Value Types**: possuem o valor da variável.
    - **Reference Types**: possuem uma referência para posição de memória onde o valor está alocado.
  - As variáveis armazenam informações na memória e segue a convenção de nomes da linguagem Java.

Identificadores

- São nomes arbitrários para variáveis, métodos, tipos definidos pelo usuários, etc...
- Devem ser compostos por caracteres Unicode;
- É *case sensitive* e *locale-independente*

```
1 <Tipo> Nome, nome;
2 <Tipo> idade, Idade;
```

Variáveis

- Armazenam informações na memória e segue a convenção de nomes da linguagem Java.

Ex.:

```
3     string nome;
4     int idade = 50;
5
```

- O tipo pode ser definido de forma implícita através do modificador var

Ex.:

```
6     var i = 5;
7     var str = "Olá Mundo";
8
```

Constantes

- Valores imutáveis que não podem ser alterados durante a execução do programa.

Ex.:

```
9  public const int MESES = 12;
10
```

Propriedades

- Uma propriedade é um membro que fornece um mecanismo flexível para ler, escrever ou calcular o valor de um campo privado.
- As propriedades podem ser usadas como se fossem membros de dados públicos, mas eles são realmente métodos especiais chamados acessadores.
- Isso permite que os dados sejam acessados facilmente e ainda ajuda a promover a segurança e a flexibilidade dos métodos.
- As propriedades permitem que uma classe exponha, de maneira pública, a forma de obter e definir valores e ocultar a implementação ou o código de verificação.
- As propriedades podem ser:
  - lidas e gravadas (elas possuem um acessador *get* e outro *set*);
  - somente leitura (elas possuem um acessador de acesso, mas nenhum *set*);
  - somente de gravação (só tem um accessor *set* definido).

Exemplo de propriedade

```
11 using System;
12
13 class TimePeriod
14 {
15     public double Segundos
16     {
17         get ;
18         private set ;
19     }
20 }
```

Exemplo de propriedade

```
21 using System;
22
23 class TimePeriod
24 {
25     private double segundos;
26     public double Segundos
27     {
28         get { return segundos; }
29         protected set { this.segundos = value;}
30     }
31 }
```

Exemplo de propriedade

```
32 using System;
33
34 class TimePeriod
35 {
36     private double seconds;
37     public double Hours
38     {
39         get { return seconds / 3600; }
40         set {
41             if (value < 0 || value > 24)
42             throw new ArgumentOutOfRangeException(
43             $"{nameof(value)} must be between 0 and 24.");
44
45             seconds = value * 3600;
46         }
47     }
48 }
```

Namespaces

- Organizam o código de grandes projetos;
- São delimitados através do uso do operador ponto;
- A diretiva using evita a necessidade de especificar o nome do *namespace* para cada classe;
- O *namespace* global é o *namespace* "raíz": `global::System` sempre irá se referir ao *namespace* System do .NET Framework;
- É uma organização lógica!

Package vs Namespace

No Java o `Package` impõe uma organização física (além da lógica). Já no C# o `namespace` só há organização lógica!

Atenção

A documentação do .NET Framework recomendam o seguinte critério para nomear os espaços de nomes:

<Empresa>.(<Produto>|<Tecnologia>)
[.<Caracteristica>][.<Subnamespace>]

Tipos

- Descrevem valores e especificam as convenções que todos os valores daquele tipo devem suportar.
- Podem ser:
  - *Value types* (Valores): tipos internos e predefinidos como inteiros e de ponto flutuante;
    - É possível criar um tipo que também aceita o valor nulo (*Nullable Types*)
    - um *nullable type* pode representar o valor correto (dentro da faixa especificada para o tipo) com o valor adicional NULL.
  - *Reference types* (Objetos): tipos que se autodescrevem (objetos), ponteiros e interfaces;
- **Atenção:** duas entidades são do mesmo tipo se possuírem representação e comportamento compatível entre si.

Tipos predefinidos

Tipo [1]	**Alias**[2]	**Descrição**
Boolean	bool	
Char	char	
Object	object	
String	string	Cadeia de caracteres Unicode
Single	float	Número de ponto flutuante IEEE 32-bits
Double	double	Número de ponto flutuante IEEE 64 bits
Int16	short	Inteiro sinalizado de 16-bits
Int32	int	Inteiro sinalizado de 32-bits
Int64	long	Inteiro sinalizado de 64-bits
Byte	byte	Inteiro sem sinal de 8-bits

[1] Usar System.<Tipo>. Ex.: `System.Boolean`

[2] Apelido para o tipo especificado.

Nullable Types

- Os tipos anuláveis representam variáveis de tipo de valor que podem ser atribuídas ao valor de `null`.
  - Não é possível criar um tipo anulável com base em um tipo de referência.
- A sintaxe `T?` É uma abreviatura para `Nullable <T>`, onde T é um tipo de valor.
- Atribua um valor a um tipo anulável, assim como você faria para um tipo de valor comum, por exemplo

```
int?  X = 10; doble?  D = 4.108;
```

- Um tipo anulável também pode ser atribuído o valor nulo: `int?  X = null`.
- Use o método `System.Nullable<T>.GetValueOrDefault` para retornar o valor atribuído ou o valor padrão para o tipo subjacente se o valor for nulo, por exemplo `int j = x.GetValueOrDefault ();`
- Use as propriedades de somente leitura `HasValue` e `Value` para testar nulo e recuperar o valor, como: `if(x.HasValue) j = x.Value;`

Nullable Types

- A propriedade `HasValue` retorna `true` se a variável contiver um valor, ou `false` se for nulo.
  - A propriedade `Value` retorna um valor se foi atribuído um. Caso contrário, um `System.InvalidOperationException` é lançado.
  - O valor padrão para `HasValue` é falso. A propriedade `Value` não tem valor padrão.
  - Você também pode usar os operadores `==` e `! =` com um tipo anulável, como mostrado no exemplo a seguir:
    `if (x! = Null) y = x;` ;
- Use o ?? Operador para atribuir um valor padrão que será aplicado quando um tipo nulo cujo valor atual é nulo é atribuído a um tipo não anulável, por exemplo `int? X = nulo; Int y = x ?? -1;`
- Não são permitidos tipos aninhados aninhados. A seguinte linha não compilará: `Nullable <Nullable<int> > n;`

Vetores

- Tem início na posição zero (0), semelhante ao Java;
- São utilizados os colchetes (**[ ]**) para informar que uma variável é um vetor;
- Devemos informar a quantidade ou iniciar o vetor;

```
49      //Vetor de inteiros com 10 espaços
50      System.Int32[] meuVetor = new System.Int32[10];
51
52      //Vetor de inteiros com 3 espaços, já preenchidos com
        os valores 1, 2 e 4, respectivamente
53      System.Int32[] meuVetor2 = new int[]{1,2,4};
```

Enumeração

- A palavra reservada *enum* é usada para declarar uma enumeração;
- Um *enum* é um tipo distinto que consiste em um conjunto de constantes nomeadas;
- O padrão é que o primeiro enumerador possua o valor 0, e o valor de cada enumerador sucessivamente é aumentado em 1.

Ex

```
54 enum Days {Sat, Sun, Mon, Tue, Wed, Thu, Fri};
55
```

Dica É melhor definir um *enum* diretamente dentro de um *namespace* para que todas as classes no *namespace* pode acessá-lo com a mesma conveniência. No entanto, um *enum* também pode ser aninhada dentro de uma *classe* ou *struct*.

Exemplo enum

```
56 [Flags]
57 enum Semana
58 {
59     Domingo = 0x1,
60     Segunda = 0x2,
61     Terca = 0x4,
62     Quarta = 0x8,
63     Quinta = 0x10,
64     Sexta = 0x20,
65     Sabado = 0x40,
66     FinalSemana = Semana.Domingo | Semana.Sabado
67 };
```


Exemplode de Enum

```
68 [Flags]
69 public enum Sexo
70 {
71     Macho = 0x1,
72     Femea = 0x2,
73     NaoInformado = Macho | Femea
74 }
75 ...
76     Sexo sexoDoAnimal = Sexo.Macho;
77     Sexo sexoDoPeixe = Sexo.Femea;
78     Sexo sexoDoDinossauro = Sexo.NaoInformado;
```


Operadores

- Em C# , um **operador** é um elemento de programa que é aplicado a um ou mais operandos em uma expressão ou declaração;
  - **operadores unários**: Os operadores que de um operando, como o operador de incremento (`++`) ou `new` , são chamados de .
  - **operadores binários**: Os operadores que tomam dois operandos, como operadores aritméticos (`+, -, *, /`);
  - Um operador, o operador condicional (? :), leva três operandos e é o único operador ternário em C# .
- A seguinte instrução C# contém um único operador unário e um único operando.

```
79          int x = 0, y = 0;
80          y++; //operador unário
81          y = x + 1; //operador binário;
82
```

Operadores aritméticos

Expressão	Descrição
+	Soma
-	Subtração
%	Resto
*	Multiplicação
/	Divisão

Tabela: Operadores aritméticos

Operadores Relacionais

Expressão	Descrição
x <y	Menor que
x >y	Maior que
x <= y	Menor ou igual a
x >= y	Maior ou igual a
x is T	Retorna verdadeiro se x é do tipo T, caso contrário, falso
x as T	Retorna x convertido para o tipo T, ou null se x não é do tipo T

Tabela: Operadores Relacionais

Operadores Lógicos

Equality Operators	
Expression	Description
x == y	Equal
x != y	Not equal
Logical, Conditional, and Null Operators	
Categoria	Exemplo
Logical AND	x & y
Logical XOR	x ^y
Logical OR	x | y
Conditional AND	x && y
Conditional OR	x || y
Null coalescing	x ?? y
Conditional	x ? y : z

Tabela: Operadores lógicos

Estrutura Geral de um Programa

- Programas em C# podem consistir em um ou mais arquivos.
- Cada arquivo pode conter zero ou mais namespaces.
- Um namespace pode conter tipos como classes, estruturas, interfaces, enumerações e `delegates`, além de outros namespaces.
- O seguinte código é o esqueleto de um programa C# que contém todos esses elementos:

Esqueleto de um programa C#

```
83 // A skeleton of a C# program
84 using System;
85 namespace YourNamespace
86 {
87     class YourClass { }
88
89     struct YourStruct { }
90
91     interface IYourInterface
       { }
92
93     delegate int
       YourDelegate();
94
95     enum YourEnum    { }
96
```

```
97     namespace
       YourNestedNamespace
98     {
99         struct YourStruct{   }
100    }
101
102    class YourMainClass
103    {
104        static void
       Main(string[] args)
105            {
106                //Your program
       starts here...
107            }
108    }
109 }
```

Escrevendo meu primeiro programa

- Um aplicativo console deve possuir um método Main, no qual o controle inicia e termina;
- Possui um método estático e pode retornar um `void` ou um inteiro;
- Adicione o código no projeto e pressione F5

```
110     using System;
111     namespace ConsoleApp1
112     {
113         class Program
114         {
115             static void Main(string[] args)
116             {
117                 Console.WriteLine("Hello World!");
118                 Console.ReadKey();
119             }
120         }
121     }
122
```


Principais Estruturas

IF-ELSE

```
123 bool condition = true;
124 if (condition)
125 {
126     if ( 10 > 5 )
127         Console.WriteLine("The variable is set to true.");
128 }
129
130 if (condition)
131 {
132     Console.WriteLine("The variable is set to true.");
133 }
134 else
135 {
136     Console.WriteLine("The variable is set to false.");
137 }
```


Switch

```
138 using System;
139
140 public class Example
141 {
142     public static void Main()
143     {
144         int caseSwitch = 1;
145         switch (caseSwitch)
146         {
147             case 1: Console.WriteLine("Case 1");
148             break;
149             case 2: Console.WriteLine("Case 2");
150             break;
151             default: Console.WriteLine("Default case");
152             break;
153         }
154     }
155 }
```

Loops em C#

- Você pode criar loops em C# utilizando comandos de iteração.
- O loop pode ser encerrado com os seguintes comandos: `break`, `goto`, `return` ou `throw`
- Para passar o controle para a próxima iteração sem sair do loop, use o `continue`
- Comandos de interação:

do Executa uma declaração ou um bloco de instruções repetidamente até que uma expressão especificada seja avaliada como falsa;

while Executa uma declaração ou um bloco de instruções até que a expressão especificada avaliar e obter o valor de `false`

Loops em C#

- Comandos de interação (cont.):

for Você pode executar uma declaração ou um bloco de instruções repetidamente até que uma expressão especificada seja avaliada como falsa;

- Este tipo de loop é útil para iterar sobre arrays e para outras aplicações nas quais você conhece com antecedência quantas vezes você deseja que o loop

foreach Repete um grupo de declarações para cada elemento em uma matriz ou uma coleção de objetos que implementa a interface `System.Collections.IEnumerable` ou `System.Collections.Generic.IEnumerable <T>`;

Comando Do-While

```
156 public class TestDoWhile
157 {
158     public static void Main ()
159     {
160         int x = 0;
161         do
162         {
163             Console.WriteLine(x);
164             x++;
165         } while (x < 5);
166     }
167 }
```


Comando While

```
168 class WhileTest
169 {
170     static void Main()
171     {
172         int n = 1;
173         while (n < 6)
174         {
175             Console.WriteLine("Current value of n is {0}",
    n);
176             n++;
177         }
178     }
179 }
```


Comando for

```
180 class ForLoopTest
181 {
182     static void Main()
183     {
184         for (int i = 1; i <= 5; i++)
185         {
186             Console.WriteLine(i);
187         }
188     }
189 }
```


Comando foreach

```
190 class ForEachTest
191 {
192     static void Main(string[] args)
193     {
194         int[] fibarray = new int[] { 0, 1, 1, 2, 3, 5, 8,
        13 };
195         foreach (int element in fibarray)
196         {
197             System.Console.WriteLine(element);
198         }
199         System.Console.WriteLine();
200     }
201 }
```


Exercícios

- Faça um programa para:
  - Imprimir seu nome e sobrenome;
  - Mostrar se um número é par ou impar;
  - Calcular dos N primeiros números pares;
  - Encontrar quantos números são múltiplos de 3 em um intervalo informado pelo usuário;
  - Calcular o fatorial de um número N;
  - Mostrar a calculadora de um número N. O usuário deverá informar também a operação desejada (+, -, * ou /)
- Faça uma função que recebe 3 parâmetros inteiros e mostra o texto formatado como dd/MM/yyyy
- Elabore um menu que comporte todos os itens de exercício

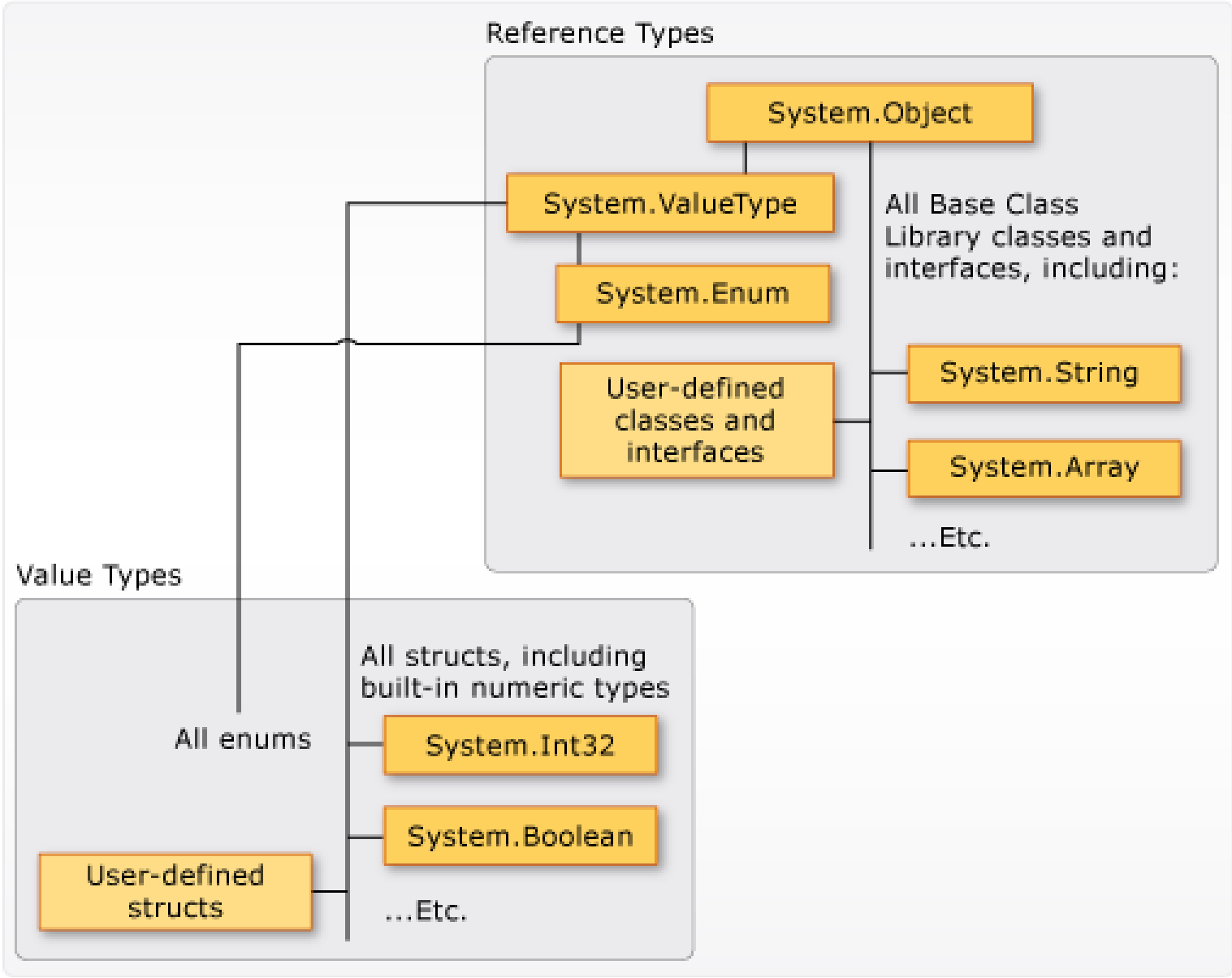
- Estude/Use as APIs
  - EasyConsole
  - Power Console

Classes e Estruturas

Classes e Structs(Estruturas)

- Conjunto de definições que encapsulam o conjunto de dados e ações em uma unidade lógica;
  - Podem possuir métodos, propriedades, variáveis eventos e outras unidades lógicas
  - Uma Class é um **reference type** já um **Struct** é um *value type*
- Em geral, as classes são utilizadas para modelar um comportamento mais complexo, ou de dados que podem ser modificados após a criação do objeto;
- *Structs* são mais adequados para estruturas de dados pequenas que contêm principalmente dados que não são modificados após a sua criação.
- As *Structs* são usadas para encapsular pequenos grupos de variáveis relacionadas, como as coordenadas de um retângulo ou as características de um livro.

Relação entre *Value Types* e *Reference Types*



Exemplo de *Struct* em C#

```
202 public struct Livro
203 {
204     public string autor;
205     public string titulo;
206     public int anoPublicacao;
207 }
```

Definição da Struct Livro em C#

Exemplo de classe em C#

```
209 using System;
210
211 namespace Slides.Slide1
212 {
213     class Pessoa
214     {
215         private int idade;
216         public int Idade
217         {
218             get { return idade; }
219             set { idade = value; }
220         }
221         public Sexo Sexo { get; set; }
222     }
223 }
```

Definição da classe Pessoa em C#

Construtores

- Sempre que uma classe ou estrutura é instanciada, o seu construtor é chamado;
- Uma classe ou estrutura pode ter vários construtores que tomam diferentes argumentos;
- Os Construtores permitem que o programador defina valores padrão, limitar a instanciação e escrever códigos flexíveis ao contexto e fáceis de ler.
- Quando não informado, é criado um construtor padrão sem parâmetros
- Um construtor é um método cujo nome é o mesmo que o nome do seu tipo;
- A assinatura do método inclui apenas o nome do método e sua lista de parâmetros;
  - Não inclui um tipo de retorno.
- Os construtores de instâncias são usados para criar e inicializar variáveis de membros da instância quando você usa a expressão `new` para criar um objeto de uma classe.

Exemplo construtor

- Construtor para uma classe chamada Pessoa.

Ex.:

```
225 public class Pessoa
226 {
227     private string ultimoNome;
228     private string primeiroNome;
229
230     public Pessoa(string primeiroNome, string
        ultimoNome)
231     {
232         this.primeiroNome = primeiroNome;
233         this.ultimoNome = ultimoNome;
234     }
235 }
```

- Construtor implementado com um simples comando

Ex.:

```
236     public Pessoa(string name) => primeiroNome =
        name;
```

Modificadores de Acesso

- Os modificadores de acesso são palavras-chave usadas para especificar a acessibilidade declarada de um membro ou de um tipo.
  - **public**: o acesso não está restrito.
  - **protected**: o acesso é limitado à classe ou tipos de conteúdo derivados da classe que contém.
  - **internal**: o acesso é limitado ao assembly corrente.
  - **protected internal**: o acesso é limitado ao assembly corrente ou tipos derivados da classe que contém.
  - **private**: o acesso é limitado ao tipo de conteúdo.

Interface

- A definição de uma interface é dada pela seguinte estrutura:

```
237 interface <<NomeDaInterface>>
238 {
239     << Comportamentos >>
240 }
```

- Use o caractere `:` (dois pontos) para indicar que uma classe/interface implementará os comportamento. Este mesmo caractere é utilizado para a herança de classes.

Herança/Polimorfismo

- O membro derivado deve usar o modificador `override` explicitamente;
- Uma classe derivada somente pode substituir um membro da classe se ele foi declarado como `abstract` ou `virtual`.
- A palavra reservada `base` é equivalente ao `super` do Java;

```
241 class SerVivo
242 {
243     // ...
244     // Campos
245     // ...
246     public SerVivo(int idade)
247     {
248         ....
249     }
250 }
```

```
251 class Pessoa : SerVivo
252 {
253     // ...
254     // Campos
255     // ...
256     public Pessoa(): base(0){
257     }
258 }
```


Atenção

- Tanto uma classe quanto um método pode ser abstrato.
- Para isso utilize a palavra reservada `abstract`;
- Métodos virtuais (`virtual`) podem ser redefinidos através da palavra `override`;
- Quando um método/classe é `selead` não é permitida a sobrescrita dele(a).
- Para criar constantes use a palavra reservada `const`
  - Quando criadas, seus valores são computados em tempo de compilação;

```
259 internal const int IDADE_ESPERADA_NO_INICIO = 0;
260
```

- Também é possível fazer com que um campo seja `readonly`

```
261 public static readonly MinhasCores Vermelho = new
        MinhasCores(255, 0, 0);
262
```

Exercício

- Implemente um programa que é responsável por gerenciar figuras geométricas como Quadrado, Retângulo e Circulo;
  - O programa deverá conseguir gerenciar uma lista de até 20 figuras;
  - Deverá apresentar um menu para:
    1. Manter (CRUD) uma figura;
    2. Listar a área de figura com base no ID;
    3. Apresentar um relatório contendo o ID|Nome|Data Criação|área ordenado pelo Nome;
    4. Listar a figura com maior lado.
  - O `quadrado` é um subtipo de `Retângulo`
- Toda figura atende a *interface* `IFigura` conforme especificado abaixo:

```
263 public interface IFigura
264 {
265     long ID {get;}
266     string Nome { get; }
267     double Area { get; }
268     DateTime DataCriacao { get; }
269 }
```

Exceções

Exceções

- O bloco `try...catch.. finally` captura e executa ações adicionais para o tratamento da exceção;
- A herança da classe `Exception` permite personalizar a exceção;
- Assim como no Java, o `throw` lança um erro no programa

```
270 try      //bloco obrigatório
271 {
272     //código para ser executado normalmente
273 }
274 catch ( Exception erro) //Bloco opcional
275 {
276     //interceptação do erro
277     Console.WriteLine(erro.Message);
278 }
279 finally      //Bloco opcional
280 {
281     //executa indepente da situação (erro ou sucesso)
282 }
```

Generics e Partial Classes and Methods

Generics

- Generics apresenta o conceito de parâmetros tipados;
- Tornam possível a concepção classes e métodos que adiam a especificação de um ou mais tipos até que a classe ou método seja declarado e instanciado pelo código do cliente do .NET Framework;
- Seu uso maximizar reutilização de código, segurança de tipo e desempenho;
- A palavra `default(T)` inicializa a variável com o valor padrão para o tipo da classe;

Exemplo Generics

- Definição Genérica

```
283 public class GenericList<T>
284 {
285     void Add(T input) { }
286     public int Count()
287     {
288         return 0;
289     }
290     public bool FazNada<U>(T parametro1, U parametro2)
291     {
292         return parametro1.Equals(parametro2);
293     }
294 }
```

- Uso:

```
295 GenericList<int> list1 = new GenericList<int>();
296 list1.FazNada<bool>(0,false);
```

Exercício

- Implemente uma pilha usando `Generics`. As operações permitidas são:

```
297 interface IPilha<T>
298 {
299     bool IsEmpty();
300     bool IsFull();
301     T Pop(); //Exceção se estiver vazia
302     void Push(T value);
303 }
304
```


Teste para a implementação

```
305 [Test]
306 public void TestPilha()
307 {
308     int[] valores = new int[] { 1, 2, 3, 4 };
309     Pilha<int> pilha = new Pilha<int>();
310     pilha.IsEmpty().Should().BeTrue();
311     foreach (int i in valores)
312     {
313         pilha.Push(i);
314     }
315     pilha.IsEmpty().Should().BeFalse();
316     int total = 0;
317     for (int i = 3; i>=0; i--)
318     {
319         var r = pilha.Pop();
320         total += r;
321         r.Should().Be(valores[i]);
322     }
323     total.Should().Be(10);
324     pilha.IsEmpty().Should().BeTrue();
```

Partial Classes and Methods

- Com o .NET é possível quebrar a definição de um método, `class`, `struct` ou `interface` em um ou mais arquivos;
- Todas as partes são combinadas no momento da instanciação;
- Fazer uso da palavra reservada `partial`;

Leitura complementar

Leitura complementar

- *Tipos de dados do C#*
  - Types (C# Programming Guide)
  - Built-In Types Table (C# Reference)
- Propriedades
  - Trabalhando com propriedades
  - Comparison Between Properties and Indexers (C# Programming Guide)
- Construtores e Instâncias
  - Using Constructors
  - Instance Constructors
- Operadores
  - Operators
- C# Coding Conventions
- Modificadores de acesso
- Case/Switch