

FIA/P GRADUAÇÃO

ENTERPRISE APPLICATION DEVELOPMENT

Prof. Me. Thiago T. I. Yamamoto

#02 –C# e Orientação a Objetos



thiagoyama



thiagoyama@gmail.com

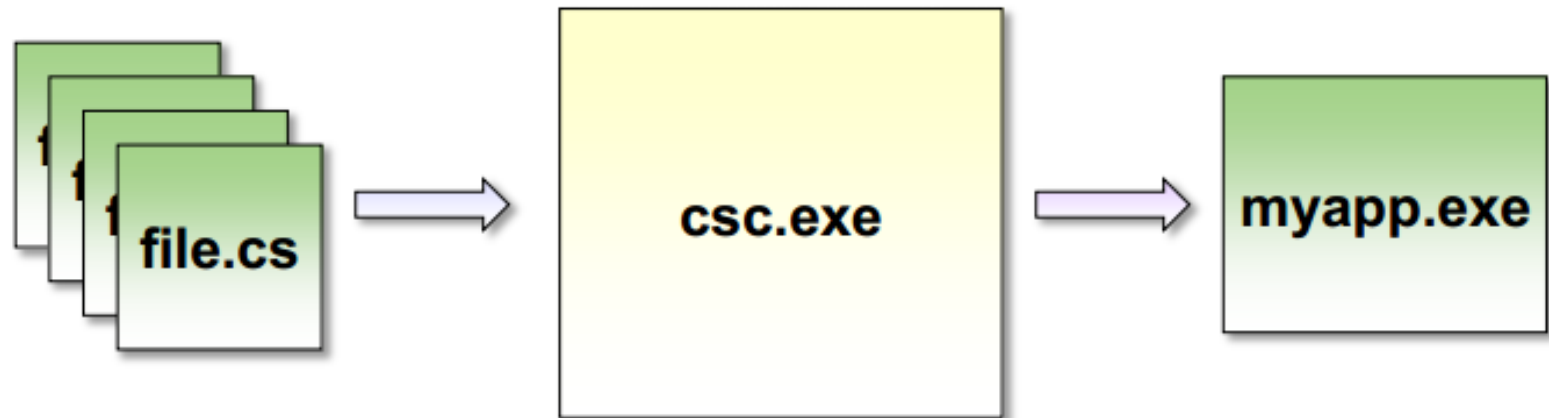
#02 – C# E ORIENTAÇÃO A OBJETOS

- C# Introdução
- C# Instruções Básicas
- C# Orientação a Objetos
- Propriedades
- Métodos
- Modificadores de Acesso
- Palavras chaves
- Properties, Fields e Construtores
- Interfaces e Namespaces
- Collections
- Exceções

- **Linguagem para criação de componentes .NET**
 - Criação de aplicações, serviços e bibliotecas;
 - Orientada a Objetos (POO);
 - Sintaxe similar ao Java e C++

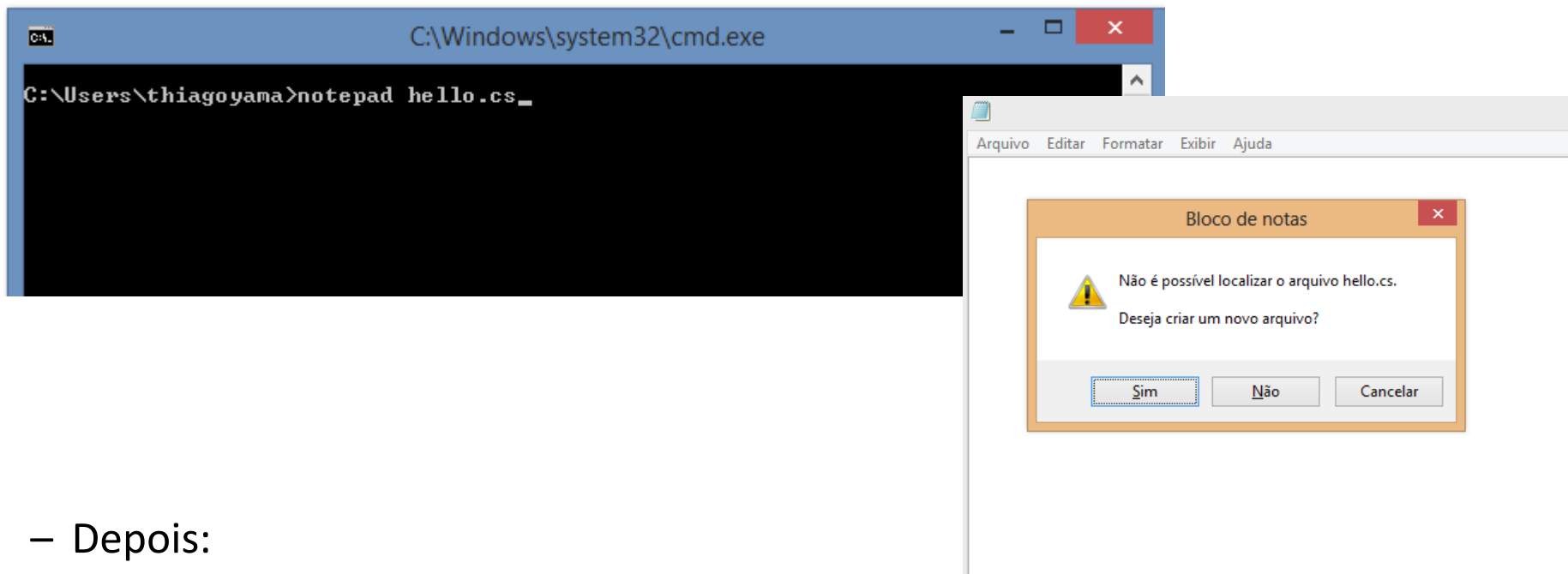
```
public static void Main(string[] args)
{
    if (DateTime.Now.DayOfWeek == DayOfWeek.Friday)
    {
        Console.WriteLine("Opa! Hoje é sexta!");
    }
}
```

- O compilador C#:
 - Transforma o código em C# na Linguagem Intermediária;
 - Produz um assembly (.dll, .exe)



HELLO WORLD!

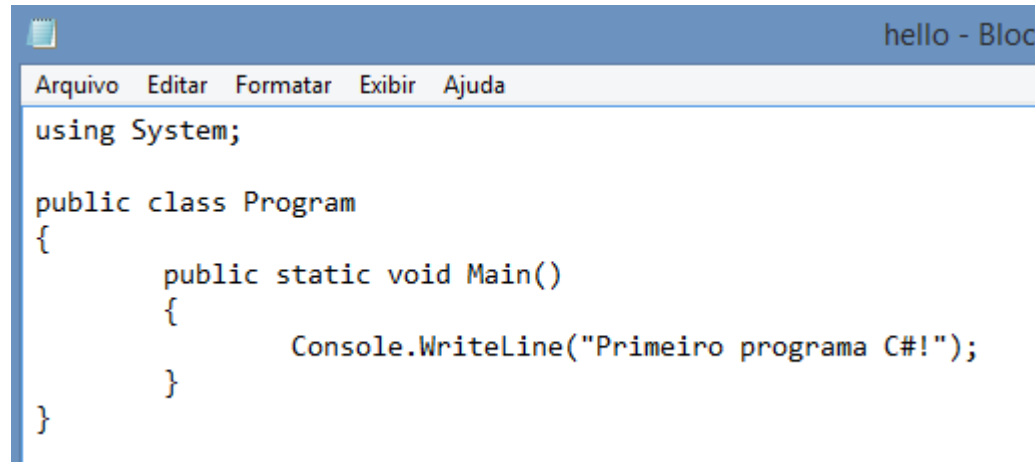
- Vamos criar um hello world sem utilizar IDE (Visual Studio)
- Primeiro vamos criar um arquivo chamado **hello.cs** (.cs é a extensão para códigos escritos em C#)
 - Abra o cmd do windows, navegue até o diretório onde deseja criar o arquivo e digite: `notepad hello.cs`



- Depois:
Sim!, você quer criar um novo arquivo!

HELLO WORLD!

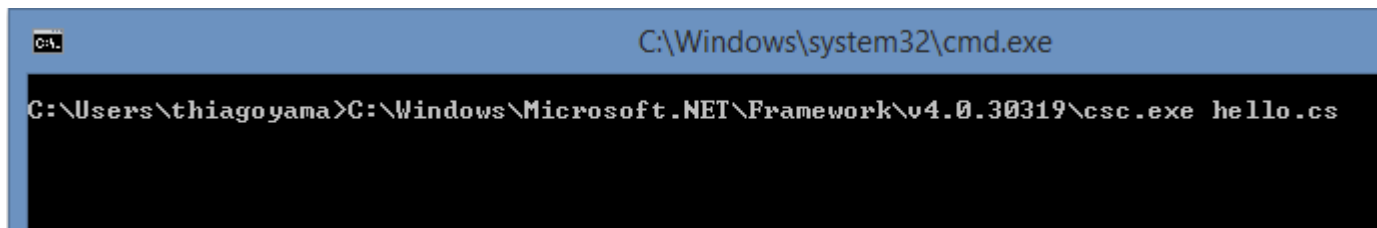
- No arquivo, digite o código abaixo:



```
using System;

public class Program
{
    public static void Main()
    {
        Console.WriteLine("Primeiro programa C#!");
    }
}
```

- Agora é hora de compilar o arquivo! Para isso, navegue até o diretório de instalação do .NET Framework, utilize o `csc.exe` para compilar arquivo passando o nome do arquivo como parâmetro:

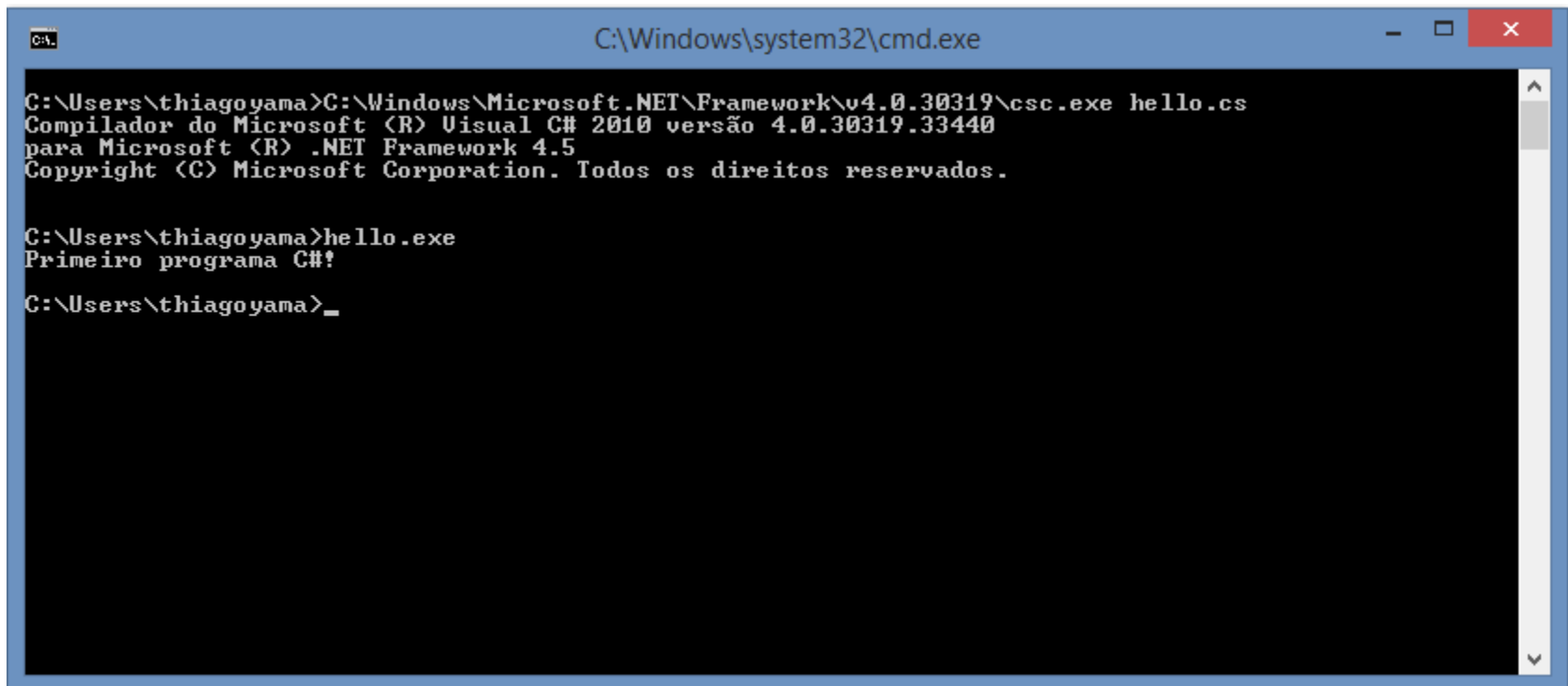


```
C:\Windows\system32\cmd.exe

C:\Users\thiagoyama>C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe hello.cs
```

HELLO WORLD!

- Foi gerado um arquivo hello.exe! (Executável)
- Agora é só executar e ver o resultado:



```
C:\Windows\system32\cmd.exe

C:\Users\thiagoyama>C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe hello.cs
Compilador do Microsoft (R) Visual C# 2010 versão 4.0.30319.33440
para Microsoft (R) .NET Framework 4.5
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

C:\Users\thiagoyama>hello.exe
Primeiro programa C#!

C:\Users\thiagoyama>_
```


COMANDOS BÁSICOS

- Operadores
 - Matemáticos: +, -, *, /
 - Relacionais : <, >, <=, >=, ==, !=
 - Lógicos: &&, ||, !
 - Atribuição: =, +=, -=, *=, /=

```
int x = 5;
int y = 10;
if (x != y)
{
    x++;
}
else
{
    y += x;
}
```

```
if (idade <= 18)
{
    //Código...
}
else
{
    //Código
}
```

C# - SWITCH

- Usado com integers, characters, strings e enums;
- Default é opcional;

```
switch (nome)
{
    case "Thiago":
        //Código..
        break;
    case "Leandro":
        //Código
        break;
    default:
        //Código
        break;
}
```

```
for (int i = 0; i < 10; i++)  
{  
    //Código  
}
```

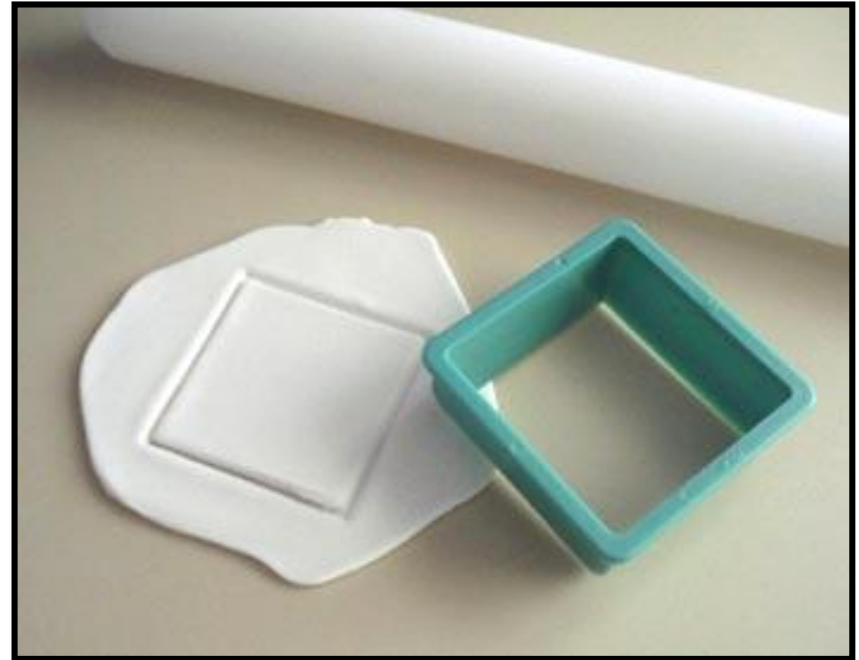
```
while (j < 10)  
{  
    j++;  
    //Código  
}
```

```
do  
{  
    j++;  
    //Código  
} while (j < 10);
```

ORIENTAÇÃO A OBJETOS

- **Classes define:**

- Estado;
- Comportamento;



- **Objeto é uma instancia de uma classe:**

- É possível criar várias instancias;
- Cada instância possui diferentes estados;

■ Classe Professor:



Nome: Roberto
Matrícula: 567138
Admissão: 25/06/2002
Cargo: Docente
Disciplina: Química
Carga horária: 16 horas



Professor Roberto

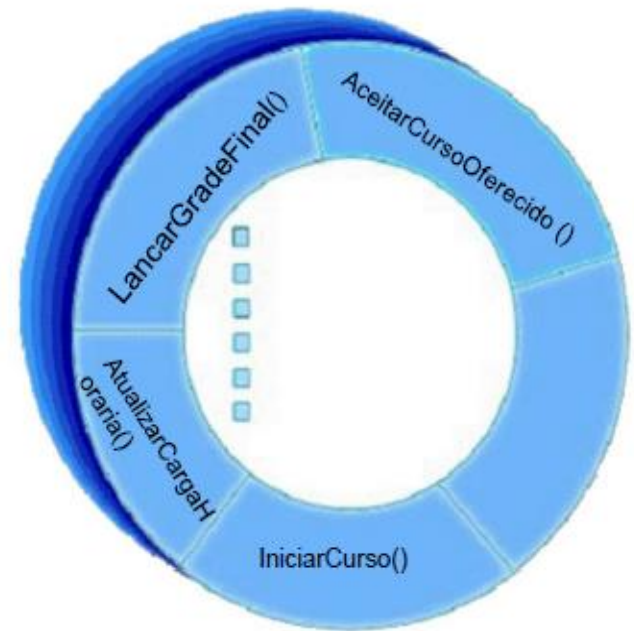
Estados ou propriedades

■ Classe Professor:



Comportamento do Professor Roberto

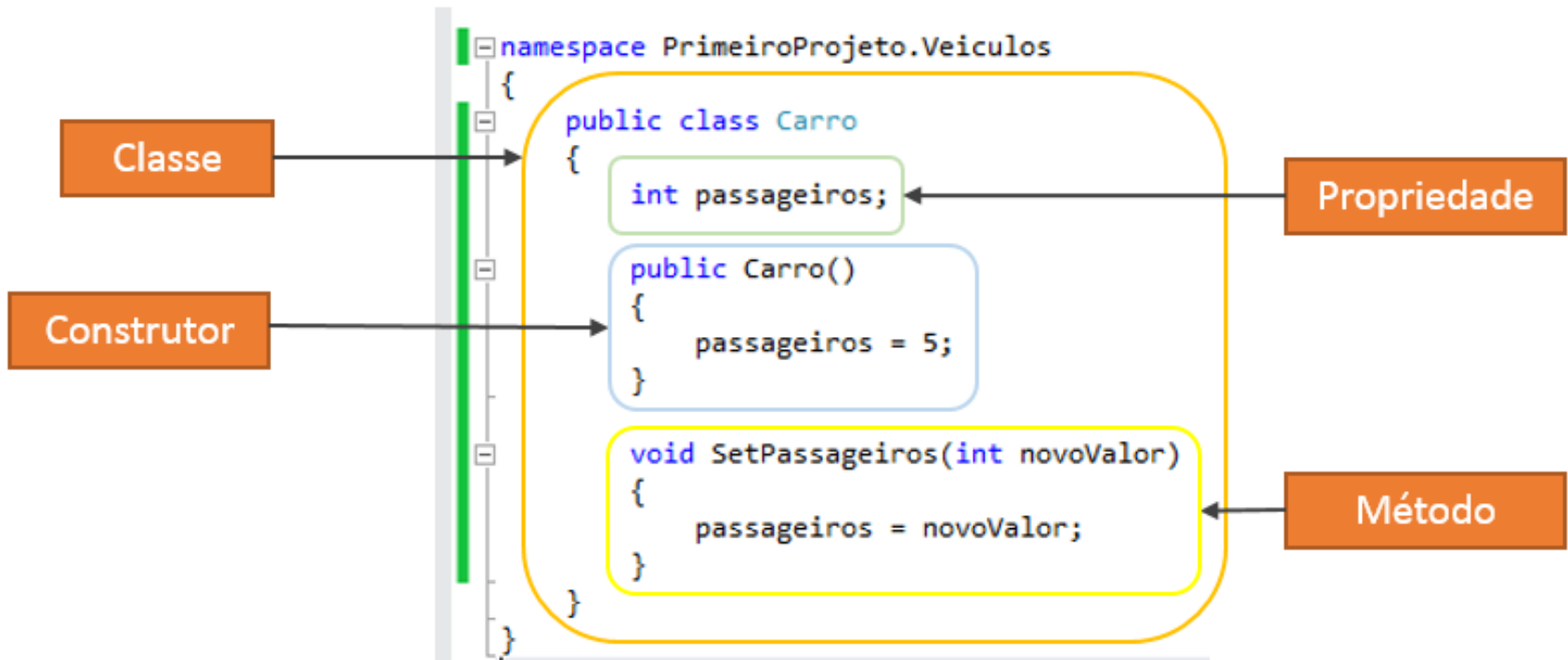
- Aceitar o Curso Oferecido
- Atualizar Carga Horária
- Lançar Grade Final
- Iniciar Curso



Professor Roberto

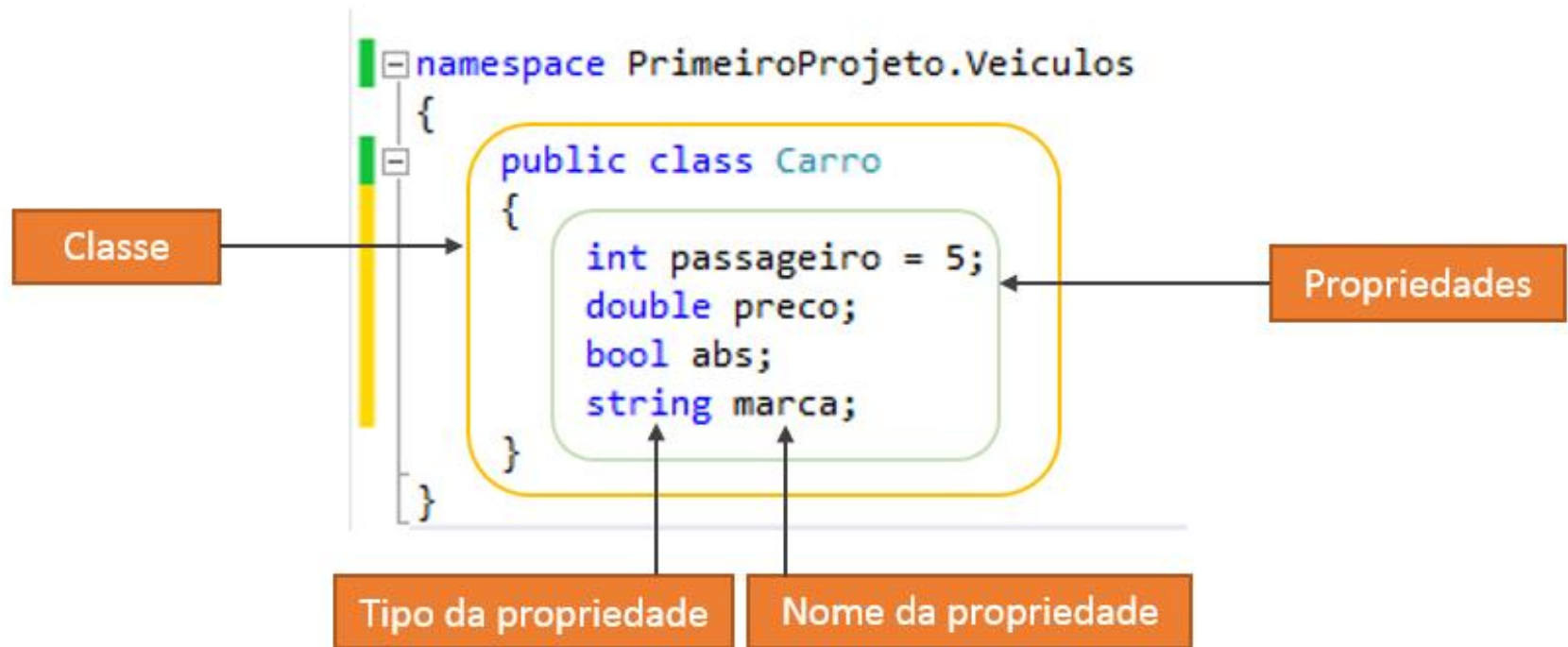
Comportamento ou métodos

Visão geral de uma classe em C#:



Sintaxe:

<tipo> <nome da variável>



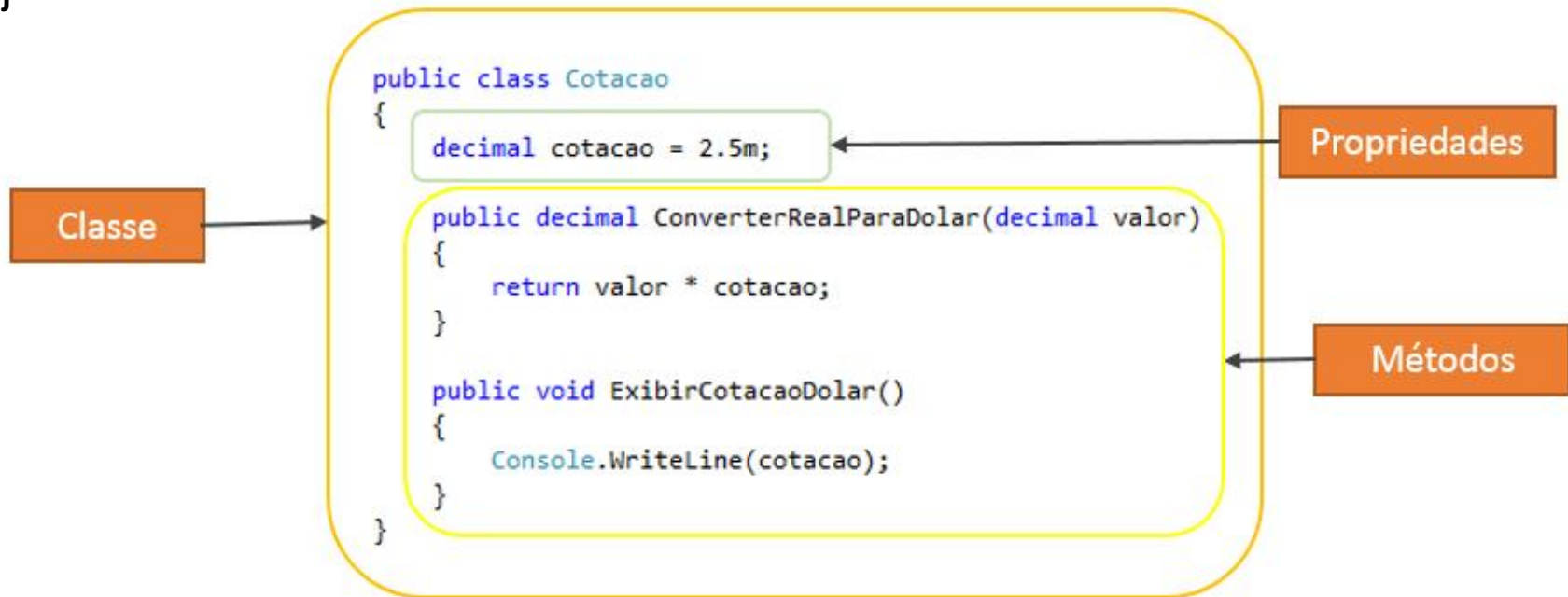
- Responsável por armazenar informações da classe;
- Sempre começa com uma letra em minúscula;

Nome abreviado	Classe do .NET	Type (Tipo)	Width	Intervalo (bits)
byte	Byte	Inteiro sem sinal	8	0 a 255
sbyte	SByte	inteiro com sinal com sinal	8	-128 a 127
int	Int32	inteiro com sinal com sinal	32	-2,147,483,648 to 2,147,483,647
uint	UInt32	Inteiro sem sinal	32	0 a 4294967295
short	Int16	inteiro com sinal com sinal	16	-32.768 a 32.767
ushort	UInt16	Inteiro sem sinal	16	0 a 65535
long	Int64	inteiro com sinal com sinal	64	-922337203685477508 to 922337203685477507
ulong	UInt64	Inteiro sem sinal	64	0 a 18446744073709551615
float	Single	Tipo de ponto flutuante de precisão simples	32	-3.402823e38 para 3.402823e38
double	Double	Tipo de ponto flutuante de precisão dupla	64	-1.79769313486232e308 para 1.79769313486232e308
char	Char	Um único caractere Unicode	16	Unicode símbolos usados no texto
bool	Boolean	Tipo booleano lógico	8	True ou false
object	Object	tipo de base de todos os outros tipos		
string	String	Uma sequência de caracteres		
decimal	Decimal	Preciso tipo fracionário ou integral que pode representar números Decimal com 29 dígitos significativos	128	$\pm 1.0 \times 10e-28$ para $\pm 7.9 \times 10e28$

Fonte: [http://msdn.microsoft.com/pt-br/library/ms228360\(v=vs.90\).aspx](http://msdn.microsoft.com/pt-br/library/ms228360(v=vs.90).aspx)

Sintaxe:

```
<modificador> <tipo de retorno> <NomeDoMetodo>(<[lista de argumentos]>){  
    [instruções];  
}
```



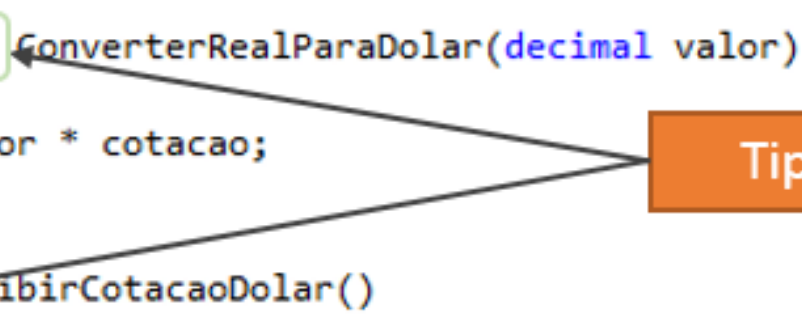
- Responsável por executar alguma lógica;
- Sempre começa com uma letra em maiúscula;

▪ Tipo de Retorno:

```
public class Cotacao
{
    decimal cotacao = 2.5m;

    public decimal ConverterRealParaDolar(decimal valor)
    {
        return valor * cotacao;
    }

    public void ExibirCotacaoDolar()
    {
        Console.WriteLine(cotacao);
    }
}
```



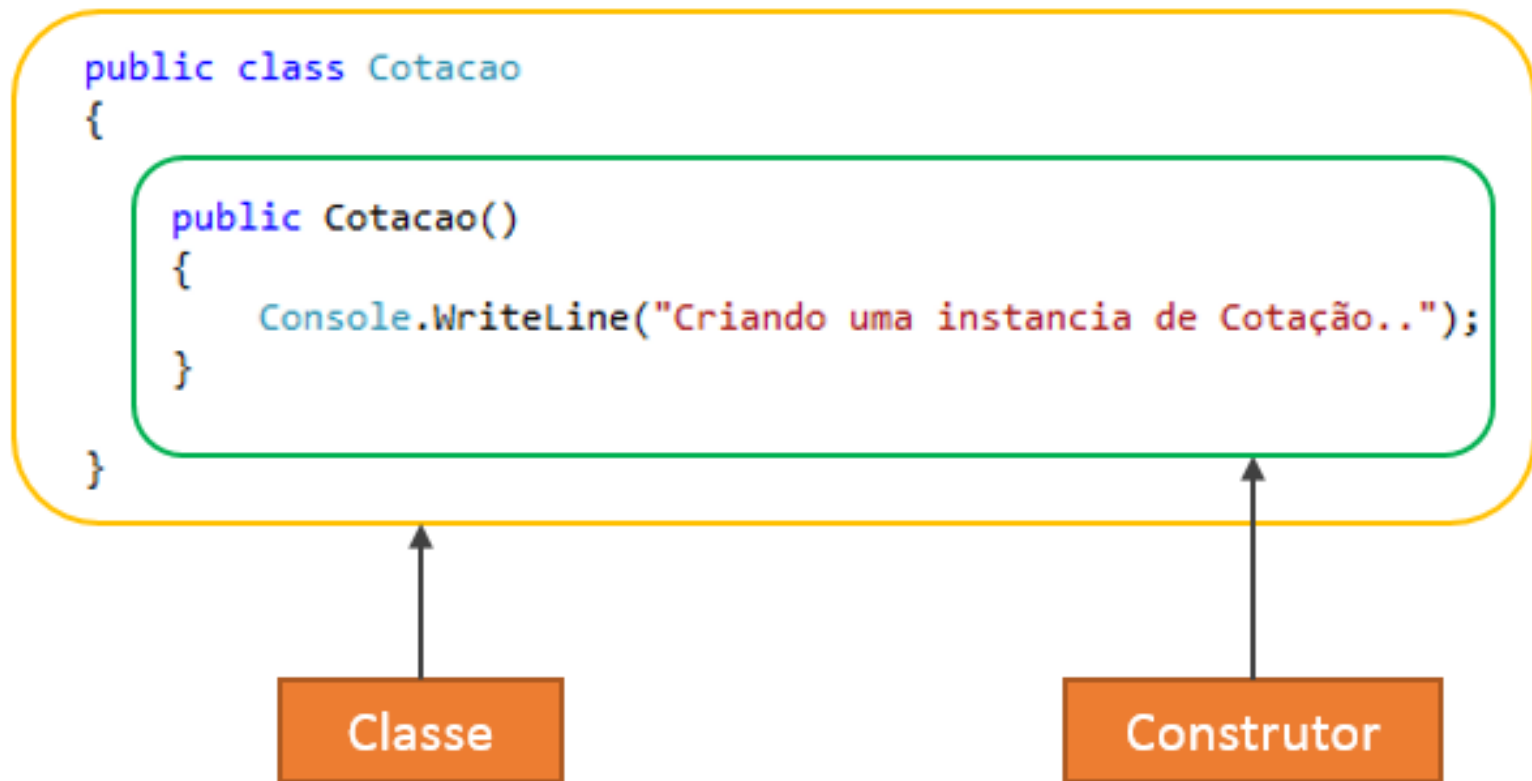
- A palavra **return** é utilizada para retornar um valor;

■ Parâmetros:

```
public void CadastrarFuncionario(string nome, int codigo, double salario)  
{  
}  
}
```



- Auxilia na construção de objetos;
- Possui o mesmo nome da classe e não possui tipo de retorno;



- Um classe pode possuir vários construtores:

```
public class Cotacao
{
    decimal cotacaoDolar;
    decimal taxaJuros;

    public Cotacao()
    {

    }

    public Cotacao(decimal cotacaoAtual)
    {
        cotacaoDolar = cotacaoAtual;
    }
    public Cotacao(decimal cotacaoAtual, decimal taxaAtual)
    {
        cotacaoDolar = cotacaoAtual;
        taxaJuros = taxaAtual;
    }
}
```

- Instanciando uma classe:
 - Utiliza a palavra **new**

```
Cotacao cotacao = new Cotacao();  
  
Cotacao cotacao2 = new Cotacao(10, 10);
```

- Palavras chaves que declara o nível de acesso:

Palavra Chave	Aplicável para	Descrição
public	Class, Membros	Sem restrição
protected	Membros	Acesso para classe e seus filhos
internal	Class, Membros	Acesso para todos do mesmo Assembly
protected internal	Membros	Acesso para todos do mesmo Assembly e Classes filhas
private	Membros	Acesso somente para a classe

Membros: Métodos, Propriedades

C# - NAMESPACES

- Organização dos tipos (Classe, interface..);
- Não é necessário possuir o mesmo nome da pasta que o arquivo está;

```
HelloWorld.Program
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace HelloWorld
8  {
9      class Program
10     {
11         public static void Main(string[] args)
12         {
13             if (DateTime.Now.DayOfWeek == DayOfWeek.Friday)
14             {
15                 Console.WriteLine("Opa! Hoje é sexta!");
16             }
17         }
18     }
19 }
20
```

PRINCÍPIOS DA ORIENTAÇÃO A OBJETOS

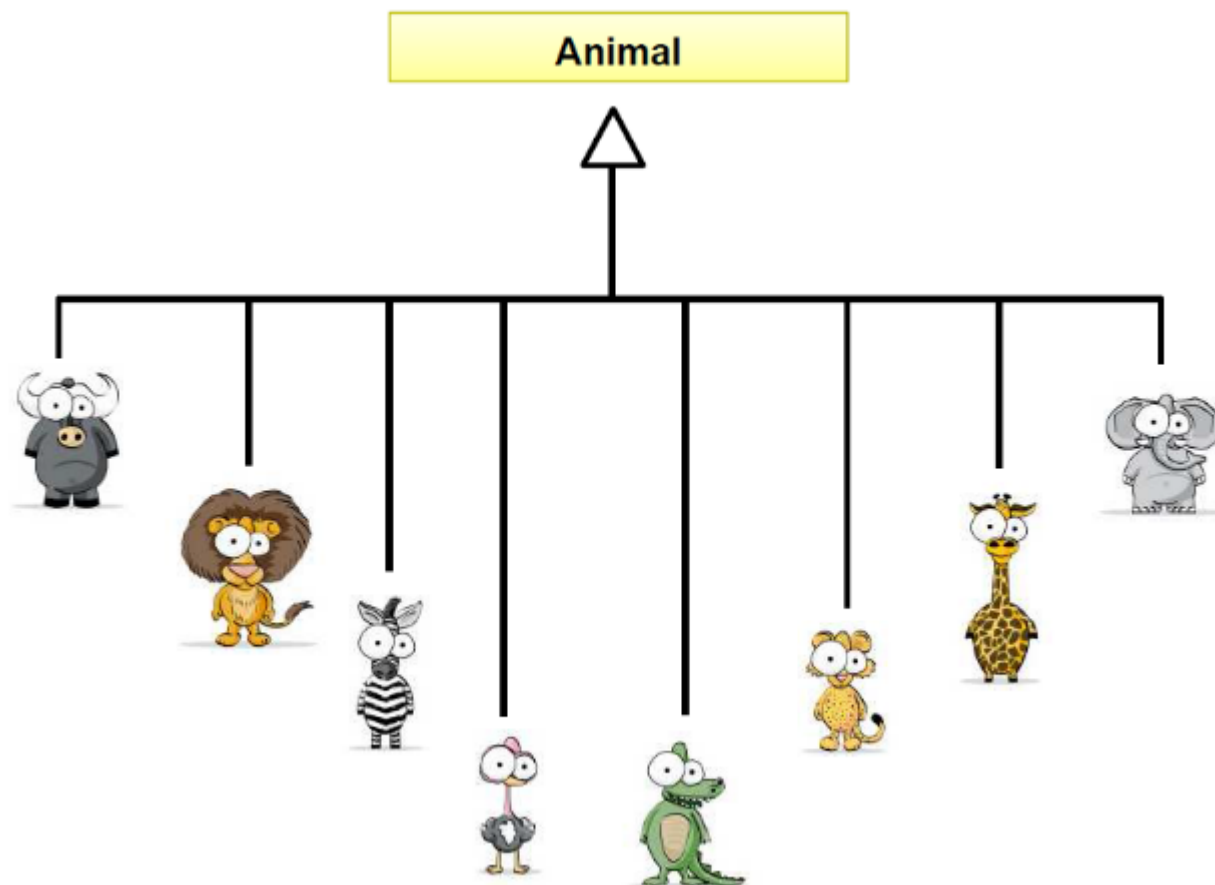
- Princípios da orientação a objetos:

Herança

Encapsulamento

Polimorfismo

- A Herança é um dos mecanismos fundamentais para as linguagens que suportam o paradigma OO (Orientação a Objetos). É um mecanismo que possibilita a criação de novas classes a partir de uma já existente. É utilizada como uma forma de reutilizar os atributos e métodos de classes já definidas.

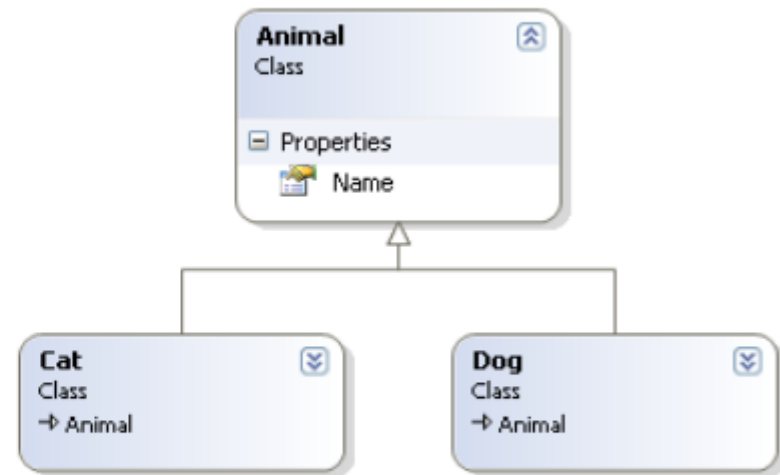


- Classes que estendem de outras classes:
 - Todas as classes descendem de System.Object;
 - Ganham todos os estados e comportamentos da classe base.

```
class Animal
{
    public string Name { get; set; }
}

class Dog : Animal
{
}

class Cat : Animal
{
}
```



C# - ENCAPSULAMENTO: CAMPOS

- Fields são variáveis da classe:
 - Estáticos ou de instância;
- Fields readonly:
 - É possível atribuir valores somente na declaração ou construtor;

```
class Animal
{
    private readonly string _nome;

    private int _idade;

    public Animal(string nome)
    {
        _nome = nome;
    }
}
```

C# - ENCAPSULAMENTO: PROPRIEDADES

- Cada propriedade define get e/ou set;
- Usado para expor e controlar fields;
- Nível de acesso para get e set são independentes;

```
private string _nome;

public string Nome
{
    get { return _nome; }
    set
    {
        if (!string.IsNullOrEmpty(value))
        {
            _nome = value;
        }
    }
}
```

- Implementação automática de propriedades usando campo oculto;
 - Nível de acesso de get e set são independentes

```
public string Nome { get; set; }
```

Dica: Digite **prop** e depois clique duas vezes tab, uma propriedade é criada automaticamente

C# - CONSTRUTORES

- Inicialização de propriedades:
 - Fácil inicialização de valores às propriedades;

```
public class Cachorro
{
    public string Nome { get; set; }
    public string Raca { get; set; }
}
```

```
Cachorro dog = new Cachorro()
{
    Nome = "Duke",
    Raca = "Pug"
};
```

Dica: Digite **ctor** e depois clique duas vezes tab, uma propriedade é criada automaticamente

- Base: utilizado para chamar construtores ou métodos da classe pai;

```
public class Animal
{
    public string Nome { get; set; }

    public Animal(string nome)
    {
        this.Nome = nome;
    }
}

public class Cachorro : Animal
{
    public string Raca { get; set; }

    public Cachorro(string nome, string raca)
        : base(nome)
    {
        this.Raca = raca;
    }
}
```

C# - CLASSE ABSTRATA

- Palavras chave **abstract**:
 - Aplicado a classe e membros;
- Classe Abstrata não pode ser instanciada:
 - Define uma classe base para outras classes;
 - Pode implementar métodos abstratos para que as classes filhas implemente-as.

```
public abstract class Animal
{
    public abstract void Eat();
}
```

```
public class Dog : Animal
{
    public override void Eat()
    {
        //Código
    }
}
```

- Palavras chave **virtual**:
 - Permite **sobrescrever** o método da classe;

```
public class Animal
{
    public virtual void Eat()
    {
        //Código
    }
}
```

```
public class Dog : Animal
{
    public override void Eat()
    {
        //Código
        base.Eat();
    }
}
```

- Conjunto de constantes;
- Por padrão, o valor armazenado para cada constante é int;

```
public enum Sexo  
{  
    Masculino,  
    Feminino,  
    Outros  
}
```


C# - INTERFACES

- Define um grupo com mesmos métodos, propriedades e eventos;
 - Todos os membros são públicos;
 - Classes e structs podem herdar várias interfaces;
 - Por padrão, a primeira letra do nome da interface deve ser I

```
interface IMessage
{
    void SendMessage(string msg);
}
```

```
class EmailMessage : IMessage
{
    public void SendMessage(string msg)
    {
        //Send email message code
    }
}
```

- Estrutura para trabalhar com coleções de variáveis:
 - Armazena o mesmo tipo de variável;
 - Começa com index 0;

```
ICollection<String> lista = new List<String>();  
lista.Add("Thiago");  
lista.Add("Yamamoto");  
  
Console.WriteLine(lista[0]);
```

- Iterar uma coleção de itens:
 - Não podemos modificar a coleção durante a iteração;

```
foreach (var item in lista)
{
    Console.WriteLine(item);
}
```

- Derivam de System.Exception
- Podemos lançar e tratar exceções

```
if (age < 18)
{
    throw new ArgumentException("Menor de idade");
}
```

Algumas Exceções:

- ✓ System.DivideByZeroException
- ✓ System.IndexOutOfRangeException
- ✓ System.InvalidCastException
- ✓ System.NullReferenceException

C# - TRATANDO EXCEÇÕES

- Try, Catch e Finally;
- Finally: executado sempre;

```
try
{
    CheckAge(17);
}
catch (ArgumentException)
{
    //Código
}
finally
{
    //Código
}
```

```
try
{
    CheckAge(17);
}
catch (ArgumentException)
{
    //Código
}
catch (Exception)
{
    //Código
}
```

- Desenvolva o exercício de acordo com o enunciado fornecido pelo professor!



Copyright © 2013 - 2017 - Prof. Me. Thiago T. I. Yamamoto

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).

*“Aprender é a única coisa que a mente nunca se cansa,
nunca tem medo e nunca se arrepende”*