



**PROVAS DE TI**  
TUDO PARA VOCÊ PASSAR

# C#

Prof. Rodrigo Macedo

# Escopo do Curso

- Conceituação Geral: características, comandos e aplicações.
- Tipos de Dados e Operadores.
- Namespaces, Execução e Interação.
- Operações com Strings.
- Coleções.
- Orientação a Objetos.
- Questões de concursos



# Introdução

- C# foi desenvolvida pela Microsoft.
- Foi apresentada junto a plataforma .NET.
- C# é uma mistura de C++ e Java.
- Criador: Anders Hejlsberg



# Características

- É orientada a objetos.
- Fortemente tipada.
- Suporte a código legado.
- Simplicidade.
- Alto nível de abstração.
- É case-sensitive, ou seja, faz diferenciação entre maiúsculas e minúsculas. Um código escrito “nome” é interpretado de forma diferente pelo compilador comparado a um código escrito “NOME”, por exemplo.



# Características

- C# está bastante vinculada ao framework.NET
- O framework .NET suporta várias linguagens de programação.
- C# usa a biblioteca de classe do framework .NET
- O framework .NET possui mais de 4 mil classes.
- Por meio da tecnologia .NET é possível desenvolver aplicativos para diversas plataformas e dispositivos.



# Comandos C# - Entrada e Saída, Laços

**ESCREVAL**



**Console.WriteLine**

**ESCREVA**



**Console.Write**

**LEIA**



**Console.ReadLine**

**SE**



**if**

**PARA**



**for**

**ENQUANTO**



**while**

**REPITA ATÉ**



**do while**

**ESCOLHA**



**switch**

# Comandos C# - Tipos de Dados

**LOGICO**



**bool**

**INTEIRO**



**int**

**REAL**



**float**

**CARACTERE**



**string**

# Comandos C# - Tipos de Dados

- Além dos tipos de dados básicos, existe uma gama de outros tipos que podem ser utilizadas.
- Existe ainda a possibilidade de criação objetos próprios e tipos personalizados (classes, estruturas, enumeradores, etc).



# Comandos C# - Tipos de Dados

Tipo de dados	Intervalo
byte	0 .. 255
sbyte	-128 ... 127
short	-32.768 .. 32.767
ushort	0 .. 65.535
int	-2.147.483.648 ... 2.147.483.647
uint	0 .. 4.294.967.295
long	-9.223.372.036.854.775.808 ... 9.223.372.036.854.775.807

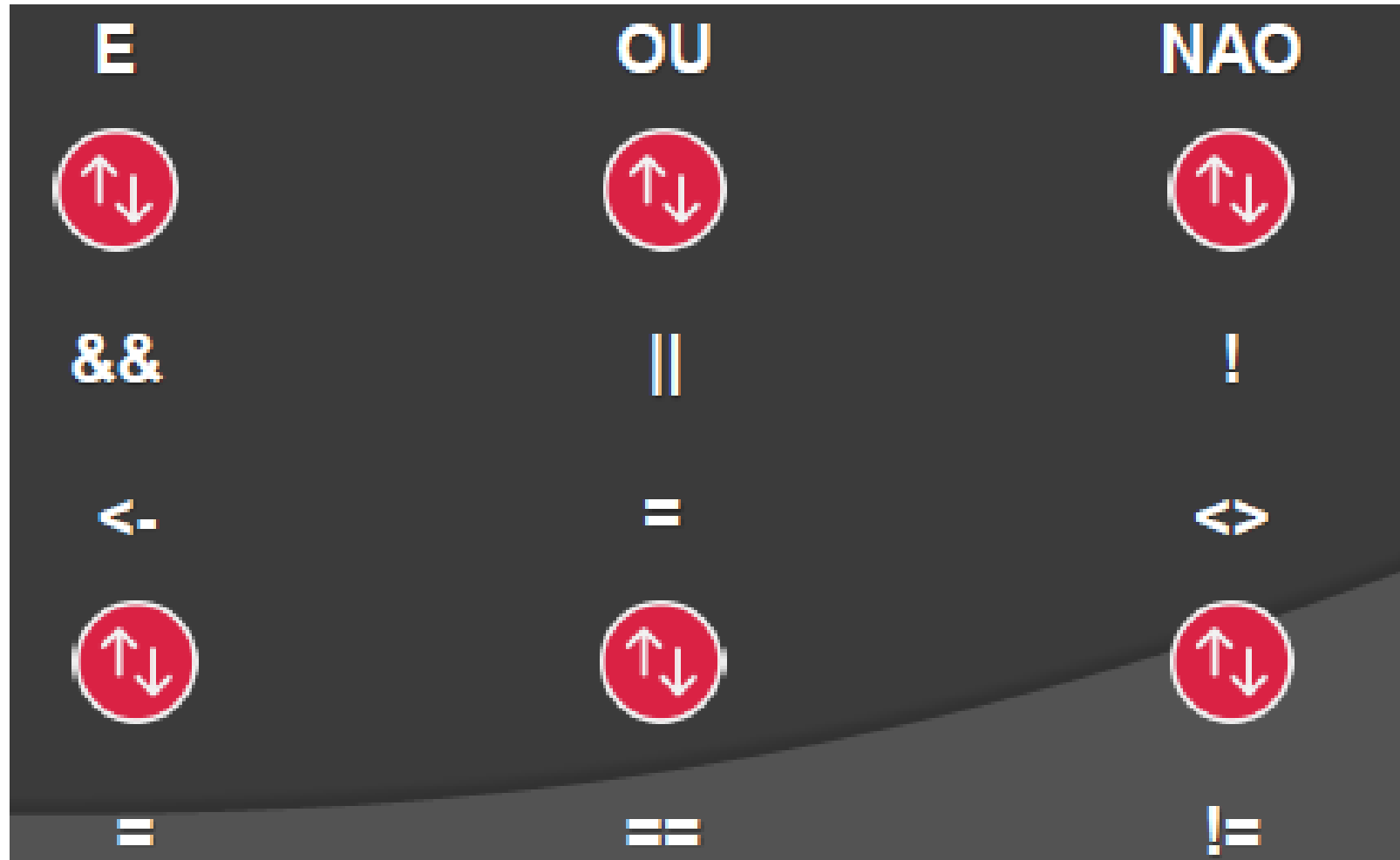
# Comandos C# - Tipos de Dados

Tipo de dados	Intervalo
ulong	0 ... 18.446.744.073.709.551.615
float	-3,402823e38 ... 3,402823e38
double	-1,79769313486232e308 1,79769313486232e308
decimal	-79228162514264337593543950335 ... 79228162514264337593543950335
char	Um caractere Unicode.
string	Uma seqüência de caracteres Unicode.
bool	VERDADEIRO ou FALSO ( <i>true</i> ou <i>false</i> ).

# Comandos C# - Operadores Lógicos

E	OU	NÃO
$V \text{ e } V = V$	$V \text{ ou } V = V$	$\text{Não } V = F$
$V \text{ e } F = F$	$V \text{ ou } F = V$	$\text{Não } F = V$
$F \text{ e } V = F$	$F \text{ ou } V = V$	
$F \text{ e } F = F$	$F \text{ ou } F = F$	

# Comandos C# - Operadores Lógicos



# Exemplo

- Tecle F6 para construir a aplicação e depois tecle a combinação Ctrl + F5 para executar.

```
namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

# Exemplo

- Ctrl + F5 no Visual C# executa sem debugging. Isso força uma pausa no final da execução permitindo que você possa visualizar o resultado da execução.
- Já o atalho F6 é o mesmo que ir no menu Debug e depois clicar em “Build Solution”.

# Retornando o tipo de variável

- Utiliza-se o método GetType() para obter o tipo do dado.

```
using System;

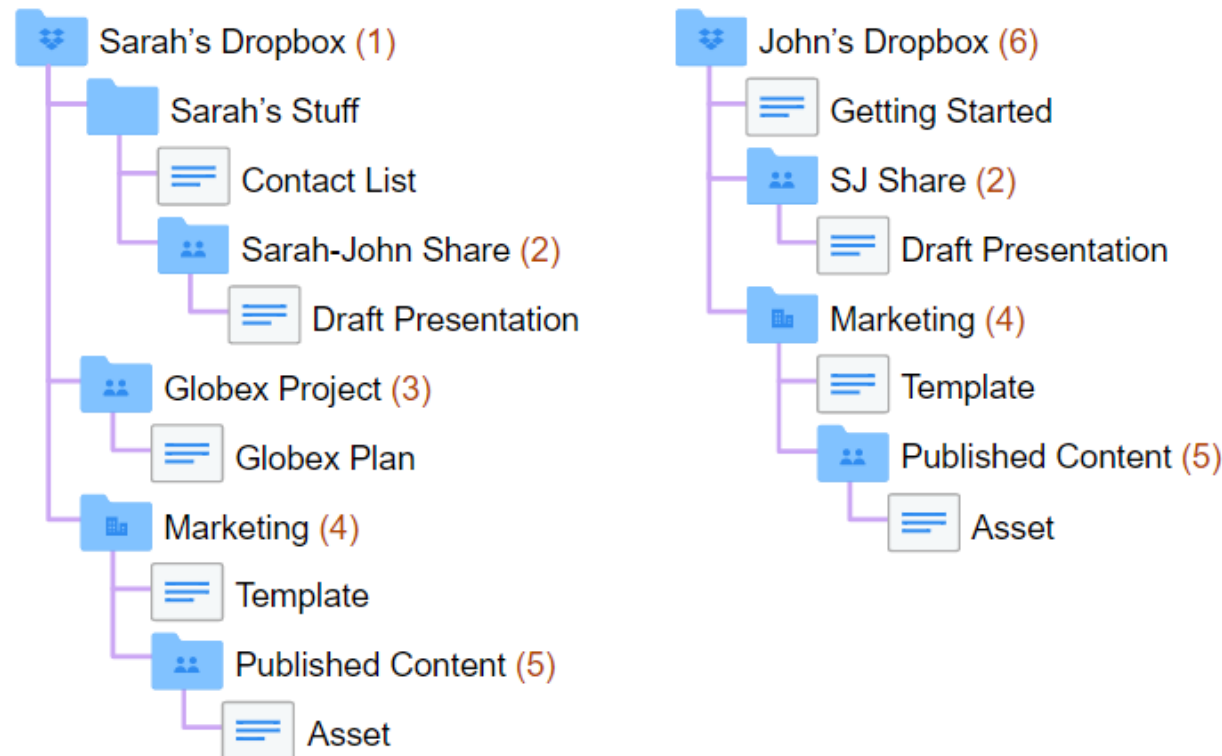
public class Program
{
    public static void Main()
    {
        int idade = 20;
        string nome = "teste";
        char sexo = 'm';
        float pi = 3.14F;
        Console.WriteLine(pi.GetType());
        Console.WriteLine(idade.GetType());
        Console.WriteLine(nome.GetType());
        Console.WriteLine(sexo.GetType());
    }
}
```

---

```
System.Single
System.Int32
System.String
System.Char
```

# Namespaces

- As classes são organizadas em namespaces.
- System é um namespace principal do C#.
- Declarar seus próprios namespaces pode ajudar no controle do escopo da classe e nomes de métodos em grandes projetos.
- Os namespaces ajudam na manutenção de um programa.





# Namespaces

- Quando você criou um projeto em C#, utilizando o Visual C#, ele já adiciona:

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;
```

- Você faz referência a um namespace utilizando a palavra reservada **“using”**.

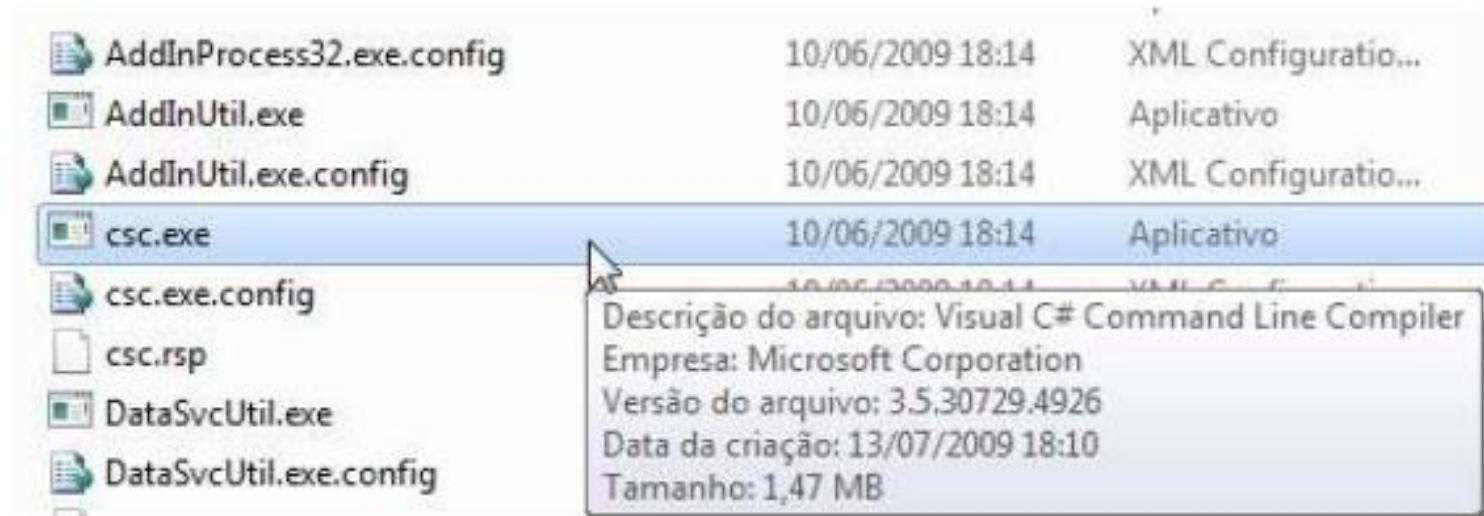
# Observações

- C# é case-sensitive, ou seja, diferencia letras minúsculas de maiúsculas.
- Com o “//” é possível fazer comentários de uma linha.
- Começando com “/\*” e fechando com “\*/” é possível fazer comentários de múltiplas linhas.

```
1 using System;
2
3 public class Program
4 {
5     public static void Main()
6     {
7         // Comentário de uma linha
8         /*
9          Comentário em
10         mais de uma
11         linha
12         */
13         Console.WriteLine("Hello World");
14     }
15 }
```

# Execução sem IDE

- Você pode executar um código sem o Visual C#.
- Para executar, basta ter o framework .NET instalado.
- Procure pelo arquivo “csc.exe”. É esse arquivo que compilará o programa.



# Execução sem IDE

- Considerando que você tenha o arquivo “HelloWorld.cs” na Área de Trabalho.
- Abra o cmd (prompt de comandos).
- Digite os seguintes comandos:
- `cd desktop`
- `C:\Windows\Microsoft.NET\Framework\v3.5\csc.exe HelloWorld.cs`
- `HelloWorld.exe`

Ao executar um script pela IDE, essa complexidade toda é abstraída!!

# Obtendo Dados do Usuário

- Utiliza-se o método `ReadLine()` para obter os dados.
- O “+” é utilizado para concatenar de valores.

---

```
Nome:  
Rodrigo  
Nome digitado: Rodrigo
```

```
1 using System;  
2  
3 public class Program  
4 {  
5     public static void Main()  
6     {  
7         Console.WriteLine("Nome:");  
8         string nome = Console.ReadLine();  
9         Console.WriteLine("Nome digitado: " + nome);  
10    }  
11 }
```

# Declarando constantes

- Basta usar a palavra reservada “const” antes do tipo da variável.
- Na declaração de uma constante, é obrigatório atribuir um valor a ela no momento da declaração.

---

5000

```
1 using System;
2
3 public class Program
4 {
5     public static void Main()
6     {
7         const int n = 5000;
8         Console.WriteLine(n);
9     }
10 }
```

# Conversão de tipos

- Utiliza-se o método Parse().

```
1234  
System.Int32
```

```
1 using System;  
2  
3 public class Program  
4 {  
5     public static void Main()  
6     {  
7         string nome = "1234";  
8         int n = int.Parse(nome);  
9         Console.WriteLine(n);  
10        Console.WriteLine(n.GetType());  
11    }  
12 }
```

# Operações com Strings - Tamanho

- Utiliza-se o método Length.

12

```
1 using System;
2
3 public class Program
4 {
5     public static void Main()
6     {
7         string nome = "Teste fulano";
8         Console.WriteLine(nome.Length);
9     }
10 }
```



# Operações com Strings - Contains

- Verifica se uma string está contida na outra. Retorna True se for verdade e False caso contrário. A saída do programa será True.

True

```
1 using System;
2
3 public class Program
4 {
5     public static void Main()
6     {
7         string nome = "Teste fulano";
8         Console.WriteLine(nome.Contains("fulano"));
9     }
10 }
```

# Operações com Strings - Concatenação

- Basta utilizar o método Concat() da classe String.

---

Teste fulano

```
1 using System;
2
3 public class Program
4 {
5     public static void Main()
6     {
7         string nome = "Teste";
8         Console.WriteLine(String.Concat(nome, " fulano"));
9     }
10 }
```

# Operações com Strings – Retirando caracteres em branco

- Para retirar do final basta utilizar o método TrimEnd() e para retirar do início basta utilizar o método TrimStart().

Teste  
fulano

```
1 using System;
2
3 public class Program
4 {
5     public static void Main()
6     {
7         string nome = "Teste  ";
8         string nome2 = "    fulano";
9         Console.WriteLine(nome.TrimEnd());
10        Console.WriteLine(nome2.TrimStart());
11    }
12 }
```

# Operações com Strings – Dividir String

- Para dividir uma string, pode-se utilizar o método Split() passando o separador que será usado para dividir a string.

C++  
C#  
Python

```
1 using System;
2
3 public class Program
4 {
5     public static void Main()
6     {
7         string nome = "C++ C# Python";
8         string[] vet = nome.Split();
9         for (int i = 0; i < vet.Length; i++){
10             Console.WriteLine(vet[i]);
11         }
12     }
13 }
```

# Operações com Strings – Juntando Strings

- Para juntar, pode-se utilizar o método Join().

---

C++  
C#  
Python  
C++,C#,Python

```
1 using System;
2
3 public class Program
4 {
5     public static void Main()
6     {
7         string nome = "C++ C# Python";
8         string[] vet = nome.Split();
9         for (int i = 0; i < vet.Length; i++){
10             Console.WriteLine(vet[i]);
11         }
12         Console.WriteLine(String.Join(", ", vet));
13     }
14 }
```

# Estruturas de Decisão

- Permite criar fluxos decisórios em programação.

Os salários são iguais

```
1 using System;
2
3 public class Program
4 {
5     public static void Main()
6     {
7         float salario1 = 1000.0f, salario2 = 1000.0f;
8
9         // Estrutura de decisão com 'if'
10        if (salario1 == salario2)
11            Console.WriteLine("Os salários são iguais");
12        else
13            Console.WriteLine("Os salários são diferentes");
14
15    }
16 }
```

# Estruturas de Decisão – Operador Ternário

- Permite criar fluxos decisórios em programação.

---

Estude mais  
Estude mais

```
1 using System;
2
3 public class Program
4 {
5     public static void Main()
6     {
7         int nota = 3;
8         string resultado = "";
9         string resultado1 = "";
10        if (nota >= 7)
11            resultado = "Parabéns";
12        else
13            resultado = "Estude mais";
14        // Agora com o operador condicional (ternário)
15        resultado1 = nota >= 7 ? "Parabéns" : "Estude mais";
16        Console.WriteLine(resultado);
17        Console.WriteLine(resultado1);
18    }
```

# Estruturas de Seleção

- Estrutura que permite seleção única ou múltipla.

Informe o mês: 3  
Mês: Indefinido

```
1 using System;
2
3 public class Program
4 {
5     public static void Main()
6     {
7         Console.Write("Informe o mês: ");
8         var mes = Int16.Parse(Console.ReadLine());
9         string respostaMes = "";
10        switch (mes)
11        {
12            case 1:
13                respostaMes = "Janeiro";
14                break; // Se atente a utilização do 'break', caso contrário o fluxo de execução prossegue para o 'case' posterior
15            case 2:
16                respostaMes = "Fevereiro";
17                break;
18            default:
19                respostaMes = "Indefinido";
20                break;
21        }
22        Console.WriteLine("Mês: " + respostaMes);
23    }
24
25 }
```



# Estruturas de Repetição - While

- Permite iterar valores em um código.

```
1 using System;
2
3 public class Program
4 {
5     public static void Main()
6     {
7         int tabuada = 3;
8         // Como o teste está no início do 'laço' pode ser que as instruções do laço nunca sejam executadas
9         int i = 0;
10        while (i <= 10)
11        {
12            Console.WriteLine(tabuada + " X " + i + " = " + tabuada * i);
13            i++;
14        }
15
16    }
17 }
```

```
3 X 0 = 0
3 X 1 = 3
3 X 2 = 6
3 X 3 = 9
3 X 4 = 12
3 X 5 = 15
3 X 6 = 18
3 X 7 = 21
3 X 8 = 24
3 X 9 = 27
3 X 10 = 30
```

# Estruturas de Repetição – Do While

- Permite iterar valores em um código.

3 X 0 = 0

```
1 using System;
2
3 public class Program
4 {
5     public static void Main()
6     {
7         int tabuada = 3;
8         int i = 0;
9         // Como o teste está no fim do 'laço', as instruções do laço serão executadas pelo menos 1 vez
10        i = 0;
11        do
12        {
13            Console.WriteLine(tabuada + " X " + i + " = " + tabuada * i);
14            i++;
15        } while (i > 3 );
16
17    }
18 }
```

# Estruturas de Repetição – For

`for( ; ; )` – representa um loop infinito, pois não há critério de parada.

- Permite iterar valores em um código.

```
3 X 0 = 0
3 X 2 = 6
3 X 4 = 12
3 X 6 = 18
3 X 8 = 24
3 X 10 = 30
```

```
1 using System;
2
3 public class Program
4 {
5     public static void Main()
6     {
7         int tabuada = 3;
8         for(int j = 0; j <= 10; j += 2)
9             Console.WriteLine(tabuada + " X " + j + " = " + tabuada * j);
10    }
11 }
```

# Estruturas de Repetição – For Break

- Quebra de fluxo com Break.

3 X 0 = 0  
3 X 2 = 6  
3 X 4 = 12

```
1 using System;
2
3 public class Program
4 {
5     public static void Main()
6     {
7         int tabuada = 3;
8         for (int k = 0; k <= 10; k += 2)
9         {
10             if (k == 6)
11                 break;
12             Console.WriteLine(tabuada + " X " + k + " = " + tabuada * k);
13         }
14     }
15
16 }
```

# Estruturas de Repetição – For Continue

- Quebra de fluxo com Continue.

3 X 0 = 0  
3 X 2 = 6  
3 X 4 = 12  
3 X 8 = 24  
3 X 10 = 30

```
1 using System;
2
3 public class Program
4 {
5     public static void Main()
6     {
7         int tabuada = 3;
8         for (int m = 0; m <= 10; m += 2)
9         {
10             if (m == 6)
11                 continue;
12             Console.WriteLine(tabuada + " X " + m + " = " + tabuada * m);
13         }
14     }
15
16 }
```

# For Each

- Serve como alternativa para outros loops para iterar valores.

---

```
FOREACH ITEM: dog  
FOREACH ITEM: cat  
FOREACH ITEM: bird
```

```
1 using System;  
2  
3 public class Program  
4 {  
5     public static void Main()  
6     {  
7         string[] pets = { "dog", "cat", "bird" };  
8  
9         foreach (string value in pets)  
10        {  
11            Console.WriteLine("FOREACH ITEM: " + value);  
12        }  
13  
14    }  
15 }
```

# Enums

- Tipo distinto que consiste em um conjunto de constantes nomeadas denominado lista de enumeradores.

```
Sun = 1  
Fri = 6  
Sun = 1  
Fri = 6
```

Lembrar dos placeholders...

```
1 using System;  
2  
3 public class Program  
4 {  
5     enum Day { Sun=1, Mon, Tue, Wed, Thu, Fri, Sat };  
6  
7     public static void Main()  
8     {  
9         int x = (int)Day.Sun;  
10        int y = (int)Day.Fri;  
11        Console.WriteLine("Sun = "+ x);  
12        Console.WriteLine("Fri = "+ y);  
13        Console.WriteLine("Sun = {0}", x);  
14        Console.WriteLine("Fri = {0}", y);  
15    }  
16  
17 }
```

# Manipulando Datas

- Utiliza-se a classe DateTime.

```
1/22/2019 8:30:15 AM
11/12/2019 12:00:00 AM
11/12/2019 12:02:10 PM
```

```
1 using System;
2
3 public class Program
4 {
5     public static void Main()
6     {
7         // Inicializando uma data em 22/01/2019 às 08:30:15
8         DateTime data1 = new DateTime(2019, 1, 22, 8, 30, 15);
9         Console.WriteLine(data1);
10        // Obtendo a data atual
11        DateTime dataAtual = DateTime.Today;
12        Console.WriteLine(dataAtual);
13
14        // Obtendo a data/hora atual
15        DateTime dataHoraAtual = DateTime.Now;
16        Console.WriteLine(dataHoraAtual);
17    }
18
19 }
```



# Manipulando Datas

- Utiliza-se a classe DateTime.

```
1 using System;
2
3 public class Program
4 {
5     public static void Main()
6     {
7         // Inicializando uma data em 22/01/2019 às 08:30:15
8         DateTime data1 = new DateTime(2019, 1, 22, 8, 30, 15);
9         Console.WriteLine(data1);
10        Console.WriteLine("Somente data: " + data1.Date);
11        Console.WriteLine("Somente hora: " + data1.TimeOfDay);
12        Console.WriteLine("Somente hora: (sem minutos e segundos) " + data1.Hour);
13        Console.WriteLine("Somente mês: " + data1.Month);
14        Console.WriteLine("Dia da semana: " + data1.DayOfWeek);
15        Console.WriteLine("Dia do ano: " + data1.DayOfYear);
16        // É fim de semana?
17        bool fimSemana = data1.DayOfWeek == DayOfWeek.Saturday || data1.DayOfWeek == DayOfWeek.Sunday;
18        Console.WriteLine("É fim de semana? " + fimSemana);
19    }
20
21 }
```

```
1/22/2019 8:30:15 AM
Somente data: 1/22/2019 12:00:00 AM
Somente hora: 08:30:15
Somente hora: (sem minutos e segundos) 8
Somente mês: 1
Dia da semana: Tuesday
Dia do ano: 22
É fim de semana? False
```

# Manipulando Datas

- Utiliza-se a classe DateTime.

```
Data 1: 1/22/2019 8:30:15 AM
Data Atual: 11/12/2019 12:11:18 PM
False
True
False
False
True
```

```
1 using System;
2
3 public class Program
4 {
5     public static void Main()
6     {
7         // Inicializando uma data em 22/01/2019 às 08:30:15
8         DateTime data1 = new DateTime(2019, 1, 22, 8, 30, 15);
9         // Obtendo a data e hora atual
10        DateTime dataAtual = DateTime.Now;
11        Console.WriteLine("Data 1: " + data1);
12        Console.WriteLine("Data Atual: " + dataAtual);
13        Console.WriteLine(DateTime.Now.Equals(data1));
14        DateTime data2 = new DateTime(2019, 1, 22, 8, 30, 15);
15        Console.WriteLine(data1.Equals(data2));
16        Console.WriteLine(!data1.Equals(data2)); // data1 é diferente de data2?
17        Console.WriteLine(data1 > dataAtual);
18        Console.WriteLine(dataAtual > data1);
19
20    }
21 }
22
23 }
```

# Struct

- O tipo struct é um tipo de valor normalmente usado para encapsular pequenos grupos de variáveis relacionadas, tais como coordenadas de um retângulo ou as características de um item em um inventário.



# Struct

---

Criação da struct  
Curso, contendo  
quatro variáveis.

O método ToString(),  
sobrescreve um  
retorno específico  
para a saída da  
struct.

```
1 using System;
2
3 public class Program
4 {
5     public struct Curso
6     {
7         public string nome;
8         public float valor;
9         public string instrutor;
10        public int matriculas;
11
12        public override string ToString()
13        {
14            return nome + " / " + instrutor + " / " + valor;
15        }
16    }
17 }
```

# Struct

```
17     public static void Main()
18     {
19         Curso cSharp;
20         cSharp.nome = "C#";
21         cSharp.instrutor = "Rodrigo Macedo";
22         cSharp.valor = 123.45f;
23         cSharp.matriculas = 15;
24         Console.WriteLine(cSharp);
25         Console.WriteLine("O curso " + cSharp.nome +
26             " possui valor de " + cSharp.valor);
27     }
```

```
C# / Rodrigo Macedo / 123.45
O curso C# possui valor de 123.45
```

# Struct com Inicializador

Criação da struct  
Curso, contendo  
quatro variáveis.

O inicializador deve  
ter o mesmo nome  
da struct.

```
public struct Formacao
{
    public string nome;
    public float valor;
    public string instrutor;
    public int matriculas;

    public Formacao(string pNome, float pValor, string pInstrutor)
    {
        nome = pNome;
        valor = pValor * 1.1f;
        instrutor = pInstrutor;
        matriculas = 0;
    }
}
```

# Struct com Inicializador

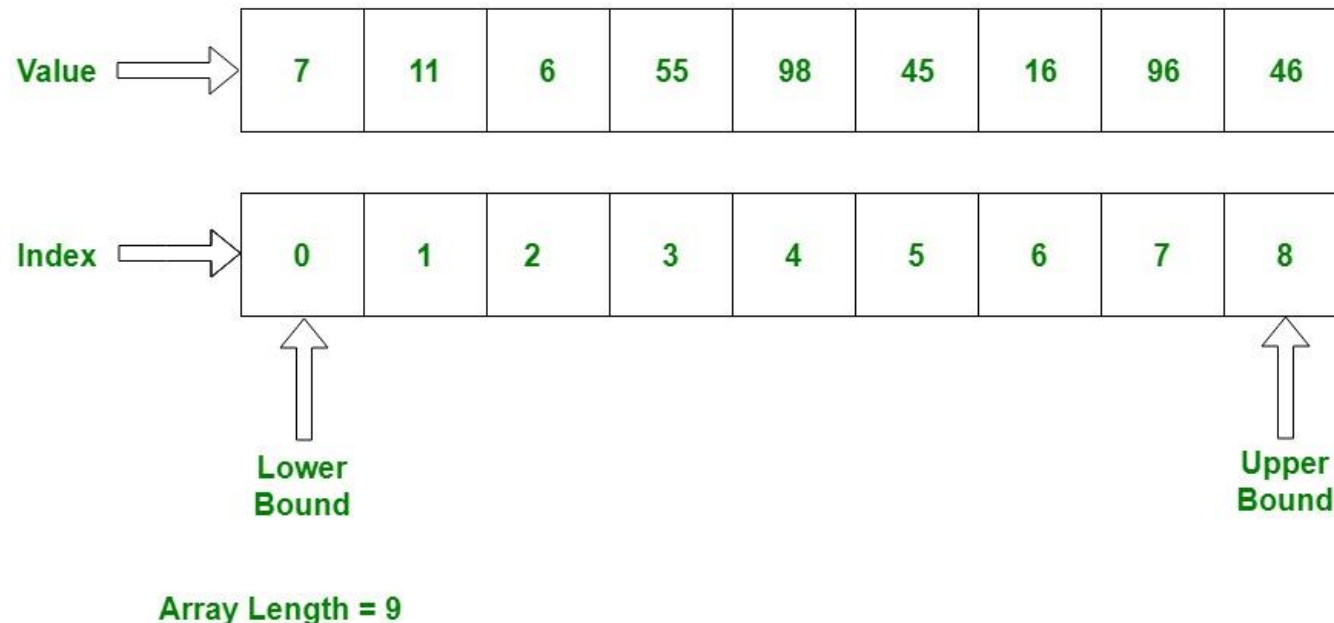
```
public static void Main()
{
    Formacao micro = new Formacao("Formação Microsoft",400,"Rodrigo Macedo");
    Console.WriteLine("A " +micro.nome+ " do instrutor "
+ micro.instrutor + " custa " +micro.valor);
}
```

---

A Formação Microsoft do instrutor Rodrigo Macedo custa 440

# Array

- Um **Array** é um conjunto de elementos de um mesmo tipo de dados onde cada elemento do conjunto é acessado pela posição no array que é dada através de um índice (*uma sequência de números inteiros*). Um array de uma dimensão é também conhecido como vetor, e, um array de mais de uma dimensão é conhecido como uma matriz





# Array

```
1 using System;
2
3 public class Program
4 {
5     public static void Main()
6     {
7         // Também podemos chamar um array de uma matriz unidimensional
8         int[] idades = new int[3]; // teremos 3 posições (0, 1 e 2)
9         idades[0] = 10;
10        idades[1] = 20;
11        idades[2] = 30;
12
13        // Para ler um valor de um array basta acessá-lo pelo índice, assim como na escrita
14        var todasIdades = idades[0] + idades[1] + idades[2];
15        Console.WriteLine("Todas as idades: " + todasIdades);
16    }
17
18 }
```

Todas as idades: 60

# Iterando um Array

Total de elementos no array: 3  
Elemento: Paula  
Elemento: Márcio  
Elemento: Simone  
Elemento: Paula  
Elemento: Márcio  
Elemento: Simone

```
3 public class Program
4 {
5     public static void Main()
6     {
7         // Criando um array e inicializando seus valores
8         string[] nomes = new string[] { "Paula", "Márcio", "Simone" };
9
10        Console.WriteLine("Total de elementos no array: " + nomes.Length);
11
12        for (int i = 0; i < nomes.Length; i++)
13            Console.WriteLine("Elemento: " + nomes[i]);
14
15        foreach (var nome in nomes)
16            Console.WriteLine("Elemento: " + nome);
17
18    }
19
20 }
```

# Operações com Array

---

Paula  
Márcio

```
5      public static void Main()
6      {
7          // Criando um array e inicializando seus valores
8          string[] nomes = new string[] { "Paula", "Márcio", "Simone" };
9
10         // Criando novo array com base no tamanho do array anterior
11         string[] nomes2 = new string[nomes.Length];
12
13         // Copiando elementos de um array para outro
14         Array.Copy(nomes, nomes2, 2); // Copiando apenas 2 elementos
15
16         foreach (var nome in nomes2)
17             Console.WriteLine(nome);
18
19     }
20
21 }
```

# Operações com Array

Método que recebe como parâmetro um array e itera e imprime seus valores.

```
private static void ImprimeArray(string[] array)
{
    Console.WriteLine("-----");
    int i = 0;
    foreach (var str in array)
    {
        Console.WriteLine(string.Format("{0} => ({1})", str, i));
        i++;
    }
    Console.WriteLine("-----");
    Console.WriteLine("");
}
```

# Operações com Array

```
public static void Main()
{
    string[] nomes = { "Paulo", "Roberto", "Maria", "Sílvia", "Antônio", "Roberta" };
    ImprimeArray(nomes);
    Array.Sort(nomes); // Ordena array
    ImprimeArray(nomes);
    Array.Reverse(nomes);
    ImprimeArray(nomes);
}
```

```
-----
Paulo => (0)
Roberto => (1)
Maria => (2)
Sílvia => (3)
Antônio => (4)
Roberta => (5)
-----
```

```
-----
Antônio => (0)
Maria => (1)
Paulo => (2)
Roberta => (3)
Roberto => (4)
Sílvia => (5)
-----
```

```
-----
Sílvia => (0)
Roberto => (1)
Roberta => (2)
Paulo => (3)
Maria => (4)
Antônio => (5)
-----
```

# Exceções

- Os recursos de manipulação de exceção da linguagem C# ajudam você a lidar com quaisquer situações excepcionais ou inesperadas que ocorram quando um programa for executado. A manipulação de exceção usa as palavras-chave try, catch e finally para executar ações que podem não ser bem-sucedidas, lidar com falhas quando decidir se é razoável fazer isso e limpar recursos posteriormente.



# Exceções

Número: 0

```
1 using System;
2
3 public class Program
4 {
5     public static void Main()
6     {
7         int numeroQualquer = 15;
8         try
9         {
10             numeroQualquer = Int16.Parse("ABC");
11         }
12         catch(Exception e)
13         {
14             numeroQualquer = 0;
15         }
16         Console.WriteLine("Número: " + numeroQualquer);
17     }
18 }
```

# Exceções

```
6      public static void Main()
7      {
8          string numStr1 = "15", numStr2 = "0";
9          try
10         {
11             int num1 = Int32.Parse(numStr1);
12             int num2 = Int32.Parse(numStr2);
13             int resultado = num1 / num2;
14             Console.WriteLine(string.Format("Resultado da divisão: {0}", resultado))
15         }
16         catch (FormatException fe)
17         {
18             Console.WriteLine("Verifique se os argumentos estão válidos");
19             Debug.WriteLine("Erro FormatException: " + fe.Message);
20             // Para utilizar o Debug.WriteLine, use a namespace System.Diagnostics;
21         }
22         catch (DivideByZeroException dze)
23         {
24             Console.WriteLine("O divisor não pode ser zero");
25             Debug.WriteLine("Erro DivideByZeroException: " + dze.Message);
26         }
27         finally
28         {
29             Console.WriteLine("Executando bloco finally");
30         }
31     }
32 }
```

O divisor não pode ser zero  
Executando bloco finally



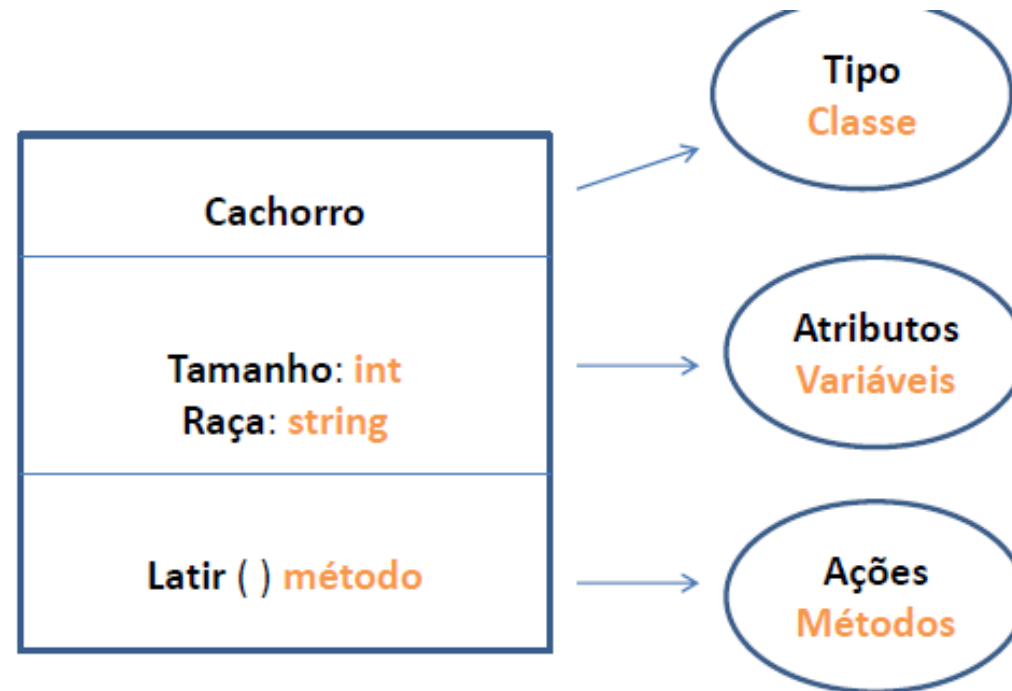
# Orientação a Objetos

- Orientação a objetos é um paradigma de desenvolvimento de software, que tem como objetivo aproximar o mundo real do mundo virtual, a ideia é simular o mundo real dentro do computador, afinal nosso mundo é composto de objetos.



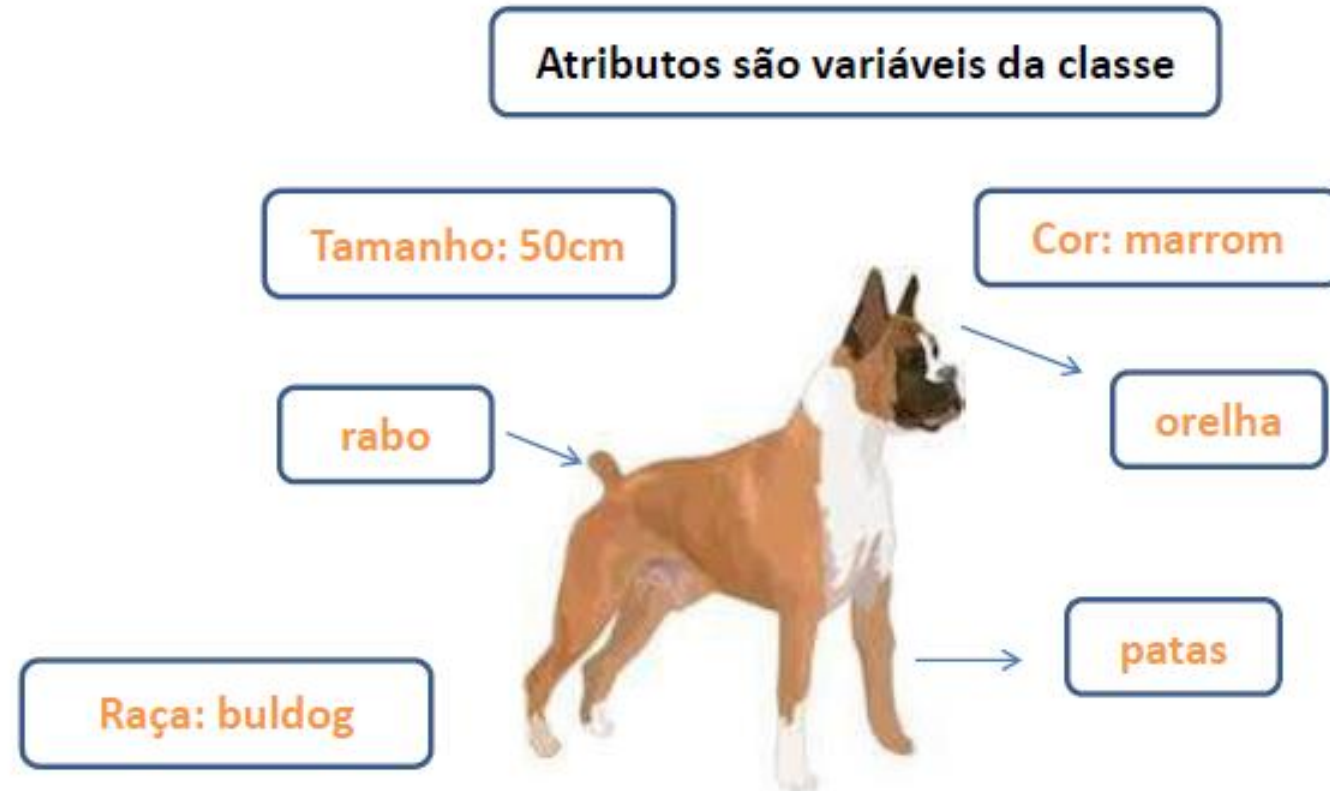
# Classe

- Representa-se a classe como um projeto do objeto, ou seja, objeto é a instancia de uma classe, antes de ser criado um objeto deve-se definir a classe na qual ele pertence. A partir da classe podemos construir objetos na memória do computador que executa a aplicação.



# Objeto

- Objeto é a instancia de uma classe, cria-se um objeto após definir uma classe para o mesmo. Na classe cachorro pode-se ter vários objetos, cada objeto pode possuir um atributo diferente.



# Métodos

- Ações que uma classe possui.

Ações (**Métodos**)

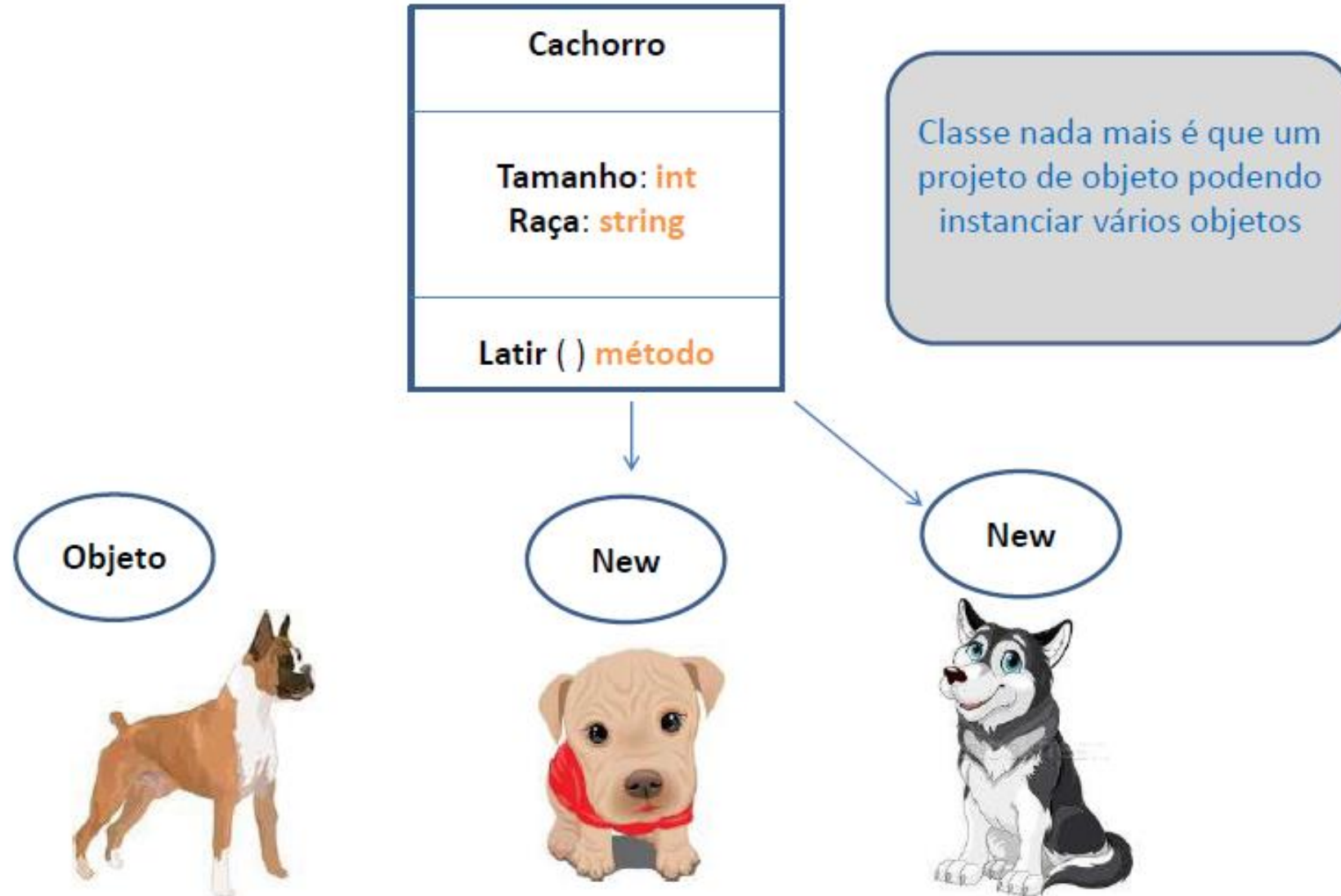


correr

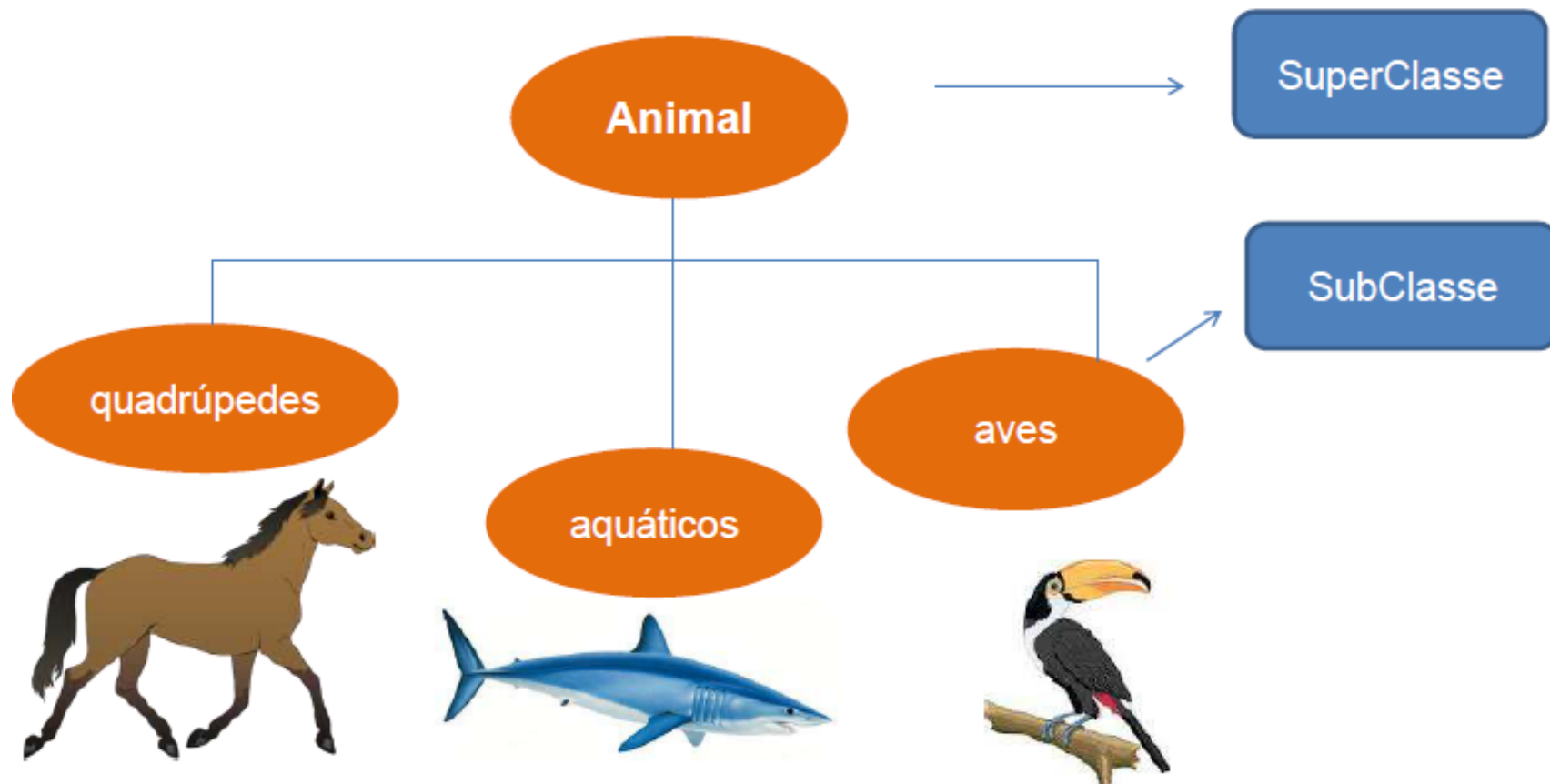
brincar

comer

# Instância de objetos



# Herança



Nota: Uma superclasse pode ser herdada por diversas subclasses. Nesta imagem a classe Animal é herdada pelas classes: quadrúpede, aquáticos e aves, sendo assim, a superclasse Animal possui as mesmas características que as outras subclasses.

# Modificadores de Visibilidade

private

→ O membro não pode ser acessado fora da classe, nem mesmo por outras classes derivadas da classe base.

protected

→ O membro não pode ser acessado fora da classe, porém o membro está disponível para outras classes derivadas da classe base.

internal

→ O membro só é visível na unidade de código onde o mesmo está definido. É um meio termo entre public e protected, uma vez que o membro pode ser acessado por todas as classes definidas na mesma unidade.

Protected-internal

→ Quer tipo de código a partir de derivados ou de código da mesma assembleia. Combinação de protegidos ou internos.

public

→ Torna o membro acessível de fora da definição da classe.

# Métodos

- Determinam os comportamentos dos objetos de uma classe.

Tipos:

❑ Sem parâmetro:

➤ Métodos que não sofre alteração externa (parâmetro).

```
public double soma() {  
    return a + b;  
}
```



## ❑ Métodos com Parâmetros:

- São métodos que sofrem influência externa, pela presença de algum parâmetro.

```
public double somaParametro(double a, double b) {  
    return a + b;  
}
```

Fonte: O autor.

## ❑ Método Principal:

- É o método que executa instruções numa linguagem.

```
public static void main(String[] args) {  
    Calculo c1 = new Calculo();  
    System.out.println(c1.soma());  
    System.out.println(c1.somaParametro(1,2));  
}
```

Fonte: O autor.

## ❑ Método Construtor:

- Tem por objetivo definir uma configuração inicial para uma classe.



Fonte: Disponível em: <https://pixabay.com/pt/artes-configura%C3%A7%C3%A3o-%C3%ADcone-s%C3%ADmbolo-47203/>

```
public Calculo(double a, double b) {  
    this.a = a;  
    this.b = b;  
}
```

Fonte: O autor.

# Sobrecarga de Métodos



Fonte: Disponível em:

<http://www.taringa.net/posts/imagenes/14435293/Sobrecarga.html>

**É possível a sobrecarga de métodos  
desde que haja diferentes assinaturas  
nos métodos comuns**

# Polimorfismo

- Permite a adaptação de métodos escritos na superclasse.
- Técnica que permite que uma implementação de determinado método possa ser de diferentes formas.
- A implementação vai variar de acordo com as especificações do objeto.

# Exemplo: Método mover.



Fonte: Disponível em:  
<https://socialspirit.com.br/fanfics/historia/fanfiction-historias-originais-uma-fada-em-minha-vida-1615087/capitulo4>

Podem até herdar da mesma classe Animal, porém implementam o método mover de formas diferentes.



Fonte: Disponível em:  
<http://www.ninha.bio.br/biologia/palhaco.html>

# Interfaces

- Atua como um contrato em relação ao desenvolvimento de um software.
- Contém apenas declarações de métodos numa interface.
- E devem ser implementadas por uma classe.



# Exemplo – Classe e Objetos

40

```
1 using System;
2
3 public class Point
4 {
5     public int x, y;
6
7
8     public static void Main()
9     {
10         Point ponto = new Point();
11         ponto.x = 50;
12         ponto.y = -10;
13         Console.WriteLine(ponto.x + ponto.y);
14     }
15
16 }
```

# Exemplo – Classe e Objetos com Inicializador

---

30

```
1 using System;
2
3 public class Point
4 {
5     public int x, y;
6     public Point(int x, int y)
7     {
8         this.x = x;
9         this.y = y;
10    }
11
12    public static void Main()
13    {
14        Point ponto = new Point(10,20);
15        Console.WriteLine(ponto.x + ponto.y);
16    }
17
18 }
```



# Exemplo – Sobrecarga de métodos

---

10  
130

```
1 using System;
2
3 public class Point
4 {
5     public int x, y;
6
7     public static void imprimeResultado(){
8         Console.WriteLine("10");
9     }
10
11     public static void imprimeResultado(int valor){
12         Console.WriteLine(valor);
13     }
14
15     public static void Main()
16     {
17         imprimeResultado();
18         imprimeResultado(130);
19     }
20
```

# Exemplo – Herança

```
1 using System;
2
3 public class Shape
4 {
5     public int X { get; private set; }
6     public int Y { get; private set; }
7
8     public virtual void Draw()
9     {
10         Console.WriteLine("Teste com desenho");
11     }
12
13 }
14 class Rectangle : Shape
15 {
16     public override void Draw()
17     {
18         // Code to draw a rectangle...
19         Console.WriteLine("Desenhando retângulo");
20         base.Draw();
21     }
22 }
```

# Exemplo – Polimorfismo

Desenhando retângulo  
Teste com desenho

```
1 using System;
2
3 public class Shape
4 {
5     public virtual void Draw()
6     {
7         Console.WriteLine("Teste com desenho");
8     }
9
10    public static void Main()
11    {
12        Rectangle ret = new Rectangle();
13        ret.Draw();
14    }
15 }
16 class Rectangle : Shape
17 {
18     public override void Draw()
19     {
20         // Code to draw a rectangle...
21         Console.WriteLine("Desenhando retângulo");
22         base.Draw();
23     }
24 }
```



**Q1) [FCC TRF4 2019 Adap]** As linguagens de programação como Java e C# têm seus códigos fontes transformados em uma linguagem intermediária (específica de cada linguagem), que será interpretada pela máquina virtual da linguagem quando o programa for executado.

**Q2) [VUNESP Câmara SP 2019]** No C#, a coleção HashSet:

- a) mantém os dados ordenados.
- b) somente permite a remoção do último elemento adicionado à coleção.
- c) somente permite a remoção do primeiro elemento da coleção.
- d) não pode ser acessada usando LINQ.
- e) não permite a inclusão de elementos duplicados.

**Q1) [FCC TRF4 2019 Adap]** As linguagens de programação como Java e C# têm seus códigos fontes transformados em uma linguagem intermediária (específica de cada linguagem), que será interpretada pela máquina virtual da linguagem quando o programa for executado. CERTO.

**Q2) [VUNESP Câmara SP 2019]** No C#, a coleção HashSet:

- a) mantém os dados ordenados.
- b) somente permite a remoção do último elemento adicionado à coleção.
- c) somente permite a remoção do primeiro elemento da coleção.
- d) não pode ser acessada usando LINQ.
- e) não permite a inclusão de elementos duplicados.

**Q3) [FGV Prefeitura Niterói 2018]** Sabendo-se que a variável `path` contém o endereço completo de um arquivo texto, e que a variável `X` foi declarada como `string`, analise o código C# a seguir.

```
X = File.ReadAllText(path)
```

Assinale a opção que apresenta o efeito desse trecho quando executado.

- a) A leitura completa do referido arquivo para a variável `X`.
- b) A leitura da primeira linha do referido arquivo para a variável `X`.
- c) A leitura de cada uma das linhas do referido arquivo para a variável `X`, que é convertida para *string array*.
- d) A comparação do conteúdo da `string X` com o conteúdo completo do referido arquivo.
- e) A cópia completa do referido arquivo para um outro, nomeado de acordo com o conteúdo da variável `X`.

**Q3) [FGV Prefeitura Niterói 2018]** Sabendo-se que a variável `path` contém o endereço completo de um arquivo texto, e que a variável `X` foi declarada como `string`, analise o código C# a seguir.

```
X = File.ReadAllText(path)
```

Assinale a opção que apresenta o efeito desse trecho quando executado.

- a) A leitura completa do referido arquivo para a variável `X`.
- b) A leitura da primeira linha do referido arquivo para a variável `X`.
- c) A leitura de cada uma das linhas do referido arquivo para a variável `X`, que é convertida para *string array*.
- d) A comparação do conteúdo da `string X` com o conteúdo completo do referido arquivo.
- e) A cópia completa do referido arquivo para um outro, nomeado de acordo com o conteúdo da variável `X`.



**Q4) [FCC Sefaz-SC 2018]** Herança e interfaces são conceitos da orientação a objetos que permitem, respectivamente, a reutilização de código e o estabelecimento de contratos de obrigatoriedade na implementação de certas funcionalidades. Em C#,

- a) métodos declarados em uma interface só podem ser públicos ou protegidos e não podem possuir implementação.
- b) para declarar que uma classe chamada PessoaJuridica implementa uma interface chamada ITributavel utiliza-se o comando `public class PessoaJuridica: implements ITributavel { }`.
- c) interfaces são mais complexas do que classes já que necessitam de atributos e métodos com implementação e sintaxe diferenciada.
- d) herança múltipla (quando uma subclasse é filha de mais de uma superclasse) não é suportada, porém, cada classe pode implementar diversas interfaces diferentes.
- e) para declarar que a classe chamada Funcionario herda a classe Pessoa e implementa a interface ITributavel utiliza-se o comando `public class Funcionario extends Pessoa: implements ITributavel { }`.

**Q4) [FCC Sefaz-SC 2018]** Herança e interfaces são conceitos da orientação a objetos que permitem, respectivamente, a reutilização de código e o estabelecimento de contratos de obrigatoriedade na implementação de certas funcionalidades. Em C#,

- a) métodos declarados em uma interface só podem ser públicos ou protegidos e não podem possuir implementação.
- b) para declarar que uma classe chamada PessoaJuridica implementa uma interface chamada ITributavel utiliza-se o comando `public class PessoaJuridica: implements ITributavel { }`.
- c) interfaces são mais complexas do que classes já que necessitam de atributos e métodos com implementação e sintaxe diferenciada.
- d) herança múltipla (quando uma subclasse é filha de mais de uma superclasse) não é suportada, porém, cada classe pode implementar diversas interfaces diferentes.
- e) para declarar que a classe chamada Funcionario herda a classe Pessoa e implementa a interface ITributavel utiliza-se o comando `public class Funcionario extends Pessoa: implements ITributavel { }`.

**Q5) [FCC Sefaz-SC 2018]** Um Auditor está trabalhando junto com uma equipe de desenvolvimento de uma aplicação em C#, que necessita de uma estrutura capaz de armazenar diversos objetos de um tipo de classe específico, de tal forma que seja facilitada a adição, consulta e remoção de elementos. Para tal, a equipe optou pelo uso da lista abaixo.

```
List<Pessoa> lista = new List<Pessoa>();  
Pessoa p1 = new PessoaJuridica();  
Pessoa p2 = new PessoaFisica();  
Pessoa p3 = new PessoaJuridica();  
lista.Add(p1);  
lista.Add(p2);  
lista.Add(p3);
```

Para percorrer os elementos desta lista obtendo cada objeto do tipo Pessoa, utiliza-se o comando

- a) `foreach (Pessoa p in lista) { }`
- b) `for(int indice=0; indice<=lista.Length(); indice++){lista.Get(Pessoa p);}`
- c) `while(Pessoa p from lista) { }`
- d) `for(int i=1; i<=lista.Length(); i++) {List.GetAt(Pessoa p);}`
- e) `foreach(Get Pessoa in lista) { }`

**Q5) [FCC Sefaz-SC 2018]** Um Auditor está trabalhando junto com uma equipe de desenvolvimento de uma aplicação em C#, que necessita de uma estrutura capaz de armazenar diversos objetos de um tipo de classe específico, de tal forma que seja facilitada a adição, consulta e remoção de elementos. Para tal, a equipe optou pelo uso da lista abaixo.

```
List<Pessoa> lista = new List<Pessoa>();  
Pessoa p1 = new PessoaJuridica();  
Pessoa p2 = new PessoaFisica();  
Pessoa p3 = new PessoaJuridica();  
lista.Add(p1);  
lista.Add(p2);  
lista.Add(p3);
```

Para percorrer os elementos desta lista obtendo cada objeto do tipo Pessoa, utiliza-se o comando

- a) `foreach (Pessoa p in lista) { }`
- b) `for(int indice=0; indice<=lista.Length(); indice++){lista.Get(Pessoa p);}`
- c) `while(Pessoa p from lista) { }`
- d) `for(int i=1; i<=lista.Length(); i++) {List.GetAt(Pessoa p);}`
- e) `foreach(Get Pessoa in lista) { }`

**Q6) [UFPR COREN-PR 2018]** Em relação à sobrecarga de operadores em C#, é INCORRETO afirmar:

- a) Os operadores são declarados públicos.
- b) Os operadores são declarados estáticos.
- c) Os operadores não são polimórficos e não podem utilizar os modificadores virtual, abstract, override ou sealed.
- d) Os operadores apresentam ausência do parâmetro this oculto.
- e) A precedência e a associatividade dos operadores sobrecarregados são redefinidas pelo programador.

**Q6) [UFPR COREN-PR 2018]** Em relação à sobrecarga de operadores em C#, é INCORRETO afirmar:

- a) Os operadores são declarados públicos.
- b) Os operadores são declarados estáticos.
- c) Os operadores não são polimórficos e não podem utilizar os modificadores virtual, abstract, override ou sealed.
- d) Os operadores apresentam ausência do parâmetro this oculto.
- e) A precedência e a associatividade dos operadores sobrecarregados são redefinidas pelo programador.

**Q7) [UFPR COREN-PR 2018]** Em relação ao mecanismo de passagem de parâmetros em C#, é correto afirmar:

- a) Quando você passa um argumento para um método, o parâmetro correspondente é inicializado com uma cópia do argumento, inclusive para um tipo-referência.
- b) As palavras-chaves ref e out indicam, respectivamente, passagem de parâmetro por referência e por valor.
- c) Parâmetros declarados com o modificador virtual indicam que as superclasses precisam definir seu tipo.
- d) O modificador override atribuído a um argumento de método, implica o ocultamento do tipo definido na classe-pai.
- e) Argumentos do tipo `int` que recebam parâmetros do tipo `WrappedInt` sofrem a operação de boxing antes da cópia dos valores.

**Q7) [UFPR COREN-PR 2018]** Em relação ao mecanismo de passagem de parâmetros em C#, é correto afirmar:

- a) Quando você passa um argumento para um método, o parâmetro correspondente é inicializado com uma cópia do argumento, inclusive para um tipo-referência.
- b) As palavras-chaves ref e out indicam, respectivamente, passagem de parâmetro por referência e por valor.
- c) Parâmetros declarados com o modificador virtual indicam que as superclasses precisam definir seu tipo.
- d) O modificador override atribuído a um argumento de método, implica o ocultamento do tipo definido na classe-pai.
- e) Argumentos do tipo `int` que recebam parâmetros do tipo `WrappedInt` sofrem a operação de boxing antes da cópia dos valores.



**Q8) [FGV AL-RO 2018]** No contexto do C#, analise o comando a seguir.

```
for( ; ; )  
{  
    // ...  
}
```

O efeito disso é

- a) um erro de sintaxe, porque nenhuma das expressões que definem o comando *for* é opcional.
- b) um erro de sintaxe, porque é obrigatória pelo menos a especificação de uma expressão de inicialização.
- c) um *loop* infinito, porque na sua ausência a expressão condicional é considerada verdadeira.
- d) um *loop* com zero iterações, porque na sua ausência a expressão condicional é considerada falsa.
- e) um erro de sintaxe, porque é obrigatória pelo menos a especificação de uma expressão de iteração.

**Q8) [FGV AL-RO 2018]** No contexto do C#, analise o comando a seguir.

```
for( ; ; )  
{  
    // ...  
}
```

O efeito disso é

- a) um erro de sintaxe, porque nenhuma das expressões que definem o comando *for* é opcional.
- b) um erro de sintaxe, porque é obrigatória pelo menos a especificação de uma expressão de inicialização.
- c) um *loop* infinito, porque na sua ausência a expressão condicional é considerada verdadeira.
- d) um *loop* com zero iterações, porque na sua ausência a expressão condicional é considerada falsa.
- e) um erro de sintaxe, porque é obrigatória pelo menos a especificação de uma expressão de iteração.

**Q9) [FGV AL-RO 2018]** Considere o seguinte trecho de código C#.

```
int? num1 = null;  
int num2 = 45;  
int? num3 = null;  
Console.WriteLine("{0}", num1 > num2);  
Console.WriteLine("{0}", num1 < num2);  
Console.WriteLine("{0}", num2 != num1);  
Console.WriteLine("{0}", num2 == num1);  
Console.WriteLine("{0}", num3 == num1);
```

Assinale a opção que contém os valores exibidos, na ordem correta.

- a) False, False, False, False, True.
- b) False, False, True, False, False.
- c) False, True, True, False, True.
- d) False, False, True, False, True.
- e) True, False, True, False, True.

**Q9) [FGV AL-RO 2018]** Considere o seguinte trecho de código C#.

```
int? num1 = null;  
int num2 = 45;  
int? num3 = null;  
Console.WriteLine("{0}", num1 > num2);  
Console.WriteLine("{0}", num1 < num2);  
Console.WriteLine("{0}", num2 != num1);  
Console.WriteLine("{0}", num2 == num1);  
Console.WriteLine("{0}", num3 == num1);
```

Assinale a opção que contém os valores exibidos, na ordem correta.

- a) False, False, False, False, True.
- b) False, False, True, False, False.
- c) False, True, True, False, True.
- d) False, False, True, False, True.
- e) True, False, True, False, True.

**Q10) [FCC Prefeitura São Luís 2018]** Em um programa construído na linguagem C# da plataforma Microsoft .NET, um Auditor se deparou com uma condição `if(x && y)`, em que `x` e `y` são valores booleanos (`bool`). Na instrução `if`,

- a) caso `x` seja *false*, `y` não será avaliado.
- b) tanto `x` quanto `y` serão sempre avaliados.
- c) caso `y` seja *true*, `x` não será avaliado.
- d) somente `x` será avaliado.
- e) somente `y` será avaliado.

**Q10) [FCC Prefeitura São Luís 2018]** Em um programa construído na linguagem C# da plataforma Microsoft .NET, um Auditor se deparou com uma condição `if(x && y)`, em que `x` e `y` são valores booleanos (`bool`). Na instrução `if`,

- a) caso `x` seja *false*, `y` não será avaliado.
- b) tanto `x` quanto `y` serão sempre avaliados.
- c) caso `y` seja *true*, `x` não será avaliado.
- d) somente `x` será avaliado.
- e) somente `y` será avaliado.

**Q11) [FGV MPE AL 2018]** No C#, a classe `FileStream` permite operações sobre arquivos, tais como leitura e gravação, dentre outras. Na criação de um objeto dessa classe, é preciso fornecer um valor para o parâmetro `FileMode`, que define como o arquivo é aberto. Assinale a opção que não é uma escolha válida para esse parâmetro.

- a) *Append.*
- b) *CreateNew.*
- c) *OpenOrCreate.*
- d) *Rewrite.*
- e) *Truncate.*

**Q11) [FGV MPE AL 2018]** No C#, a classe FileStream permite operações sobre arquivos, tais como leitura e gravação, dentre outras. Na criação de um objeto dessa classe, é preciso fornecer um valor para o parâmetro FileMode, que define como o arquivo é aberto. Assinale a opção que não é uma escolha válida para esse parâmetro.

- a) *Append.*
- b) *CreateNew.*
- c) *OpenOrCreate.*
- d) *Rewrite.*
- e) *Truncate.*



**Q12) [FGV MPE AL 2018]** Considere os modificadores a seguir.

I. *Internal*. II. *Private*. III. *Protected*. IV. *External*.

Assinale a opção que indica os modificadores da lista acima que, além do modificador *public*, são válidos no C#, quando da especificação da acessibilidade de um membro ou tipo.

- a) I, II e III, apenas.
- b) I, II e IV, apenas.
- c) I, III e IV, apenas.
- d) II, III e IV, apenas.
- e) II e III, apenas.

**Q12) [FGV MPE AL 2018]** Considere os modificadores a seguir.

I. *Internal*. II. *Private*. III. *Protected*. IV. *External*.

Assinale a opção que indica os modificadores da lista acima que, além do modificador *public*, são válidos no C#, quando da especificação da acessibilidade de um membro ou tipo.

- a) I, II e III, apenas.
- b) I, II e IV, apenas.
- c) I, III e IV, apenas.
- d) II, III e IV, apenas.
- e) II e III, apenas.

**Q13) [FGV MPE AL 2018]** Considere os seguintes operadores: *Igual a* *Negação lógica* *Módulo (resto da divisão)* *Ou lógico* *And lógico* Assinale a lista dos símbolos que, respectivamente, representam esses operadores no C#.

a) == ! % || &&

b) == ~ %% || &&

c) = ! & or and

d) == ~ mod || &&

e) = != & or and

**Q13) [FGV MPE AL 2018]** Considere os seguintes operadores: *Igual a* *Negação lógica* *Módulo (resto da divisão)* *Ou lógico* *And lógico* Assinale a lista dos símbolos que, respectivamente, representam esses operadores no C#.

a) == ! % || &&

b) == ~ %% || &&

c) = ! & or and

d) == ~ mod || &&

e) = != & or and

**Q14) [FGV BANESTES 2018]** Sobre a classe *Hashtable* disponível em C#, analise as afirmativas a seguir.

I. Não são permitidas chaves duplicadas.

II. Há uma propriedade *Keys* para recuperar todas as chaves da tabela.

III. Há uma propriedade *Values* para recuperar todos os valores da tabela.

IV. O método *Remove* requer como argumentos a chave e o valor do item a ser removido.

Está correto somente o que se afirma em:

a) I e II;

b) I, II e III;

c) I, II e IV

d) II, III e IV;

e) III e IV.

**Q14) [FGV BANESTES 2018]** Sobre a classe *Hashtable* disponível em C#, analise as afirmativas a seguir.

I. Não são permitidas chaves duplicadas.

II. Há uma propriedade *Keys* para recuperar todas as chaves da tabela.

III. Há uma propriedade *Values* para recuperar todos os valores da tabela.

IV. O método *Remove* requer como argumentos a chave e o valor do item a ser removido.

Está correto somente o que se afirma em:

a) I e II;

b) I, II e III;

c) I, II e IV

d) II, III e IV;

e) III e IV.

**Q15) [CETAP MPC-PA 2015]** A herança entre classes em C#, define-se utilizando o caractere

- a) #
- b) :
- c) /
- d) \$
- e) @

**Q15) [CETAP MPC-PA 2015]** A herança entre classes em C#, define-se utilizando o caractere

a) #

b) :

c) /

d) \$

e) @



**Q16) [FCC TRF4 2019 Adap]** As linguagens de programação:

como Java e C# têm seus códigos fontes transformados em uma linguagem intermediária (específica de cada linguagem), que será interpretada pela máquina virtual da linguagem quando o programa for executado.

**Q17) [FGV CM Caruaru 2015]** Analise o código C# a seguir.

```
for (int i = -5; i <= 7; i += 3)
{
    Console.WriteLine(i);
}
```

Assinale a opção que apresenta corretamente o resultado produzido pela execução do trecho acima.

- a) -2, 1, 4, 7, 10
- b) -5, -2, 1, 4, 10
- c) -2, 1, 4, 7
- d) -5, -2, 1, 4
- e) -5, -2, 1, 4, 7

**Q16) [FCC TRF4 2019 Adap]** As linguagens de programação:

como Java e C# têm seus códigos fontes transformados em uma linguagem intermediária (específica de cada linguagem), que será interpretada pela máquina virtual da linguagem quando o programa for executado. CERTO.

**Q17) [FGV CM Caruaru 2015]** Analise o código C# a seguir.

```
for (int i = -5; i <= 7; i += 3)
{
    Console.WriteLine(i);
}
```

Assinale a opção que apresenta corretamente o resultado produzido pela execução do trecho acima.

a) -2, 1, 4, 7, 10

b) -5, -2, 1, 4, 10

c) -2, 1, 4, 7

d) -5, -2, 1, 4

e) -5, -2, 1, 4, 7

i1: i->-5 res->-2

i2: i->-2 res->1

i3: i->1 res->4

i4: i->4 res->7

i5: i->7 res->10

**Q18) [VUNESP TCE SP 2014]** Observe a declaração de um vetor em C#:

```
int[] vetor = new int[3] { 1, 2, 3 };
```

Sem alterar o resultado, essa mesma declaração poderia ser escrita como:

a) `int[] vetor = { 1, 2, 3 };`

b) `int[] vetor = int { 1, 2, 3 };`

c) `int[] vetor = new { 1, 2, 3 };`

d) `int[] vetor = new int[];`

e) `int[] vetor = new int[] = { 1, 2, 3 };`

**Q18) [VUNESP TCE SP 2014]** Observe a declaração de um vetor em C#:

```
int[] vetor = new int[3] { 1, 2, 3 };
```

Sem alterar o resultado, essa mesma declaração poderia ser escrita como:

a) `int[] vetor = { 1, 2, 3 };`

b) `int[] vetor = int { 1, 2, 3 };`

c) `int[] vetor = new { 1, 2, 3 };`

d) `int[] vetor = new int[];`

e) `int[] vetor = new int[] = { 1, 2, 3 };`

**Q19) [FGV TJ BA 2015]** Observe o trecho inicial, criado no Visual Studio 2010 Ultimate, para uma aplicação de console escrita em C#.

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;
```

A diretiva `using.System.Linq` refere-se a um conjunto de padrões e artefatos destinados à:

- a) criação e utilização de relatórios;
- b) realização de consultas e atualizações de dados;
- c) criação e utilização de expressões regulares;
- d) comunicação remota direta com outras aplicações C#;
- e) identificação e utilização de Web Services.

**Q19) [FGV TJ BA 2015]** Observe o trecho inicial, criado no Visual Studio 2010 Ultimate, para uma aplicação de console escrita em C#.

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;
```

A diretiva `using.System.Linq` refere-se a um conjunto de padrões e artefatos destinados à:

- a) criação e utilização de relatórios;
- b) realização de consultas e atualizações de dados;**
- c) criação e utilização de expressões regulares;
- d) comunicação remota direta com outras aplicações C#;
- e) identificação e utilização de Web Services.

**Q20) [VUNESP IMESC 2013]** Na linguagem C#, para inserir um elemento no final de um ArrayList, deve ser utilizado o método

- a) Add.
- b) AddToEnd.
- c) Append.
- d) Insert.
- e) Put.

**Q21) [VUNESP PRODEST-ES 2014]** Na linguagem de programação C#, a sintaxe correta para declarar um objeto do tipo Carro e produzir uma nova instância desse objeto é:

- a) Carro obj = new Carro();
- b) Carro obj = Carro.new;
- c) Carro obj = Carro();
- d) Carro = new Carro();
- e) obj = Carro();

**Q20) [VUNESP IMESC 2013]** Na linguagem C#, para inserir um elemento no final de um ArrayList, deve ser utilizado o método

a) Add.

b) AddToEnd.

c) Append.

d) Insert.

e) Put.

**Q21) [VUNESP PRODEST-ES 2014]** Na linguagem de programação C#, a sintaxe correta para declarar um objeto do tipo Carro e produzir uma nova instância desse objeto é:

a) Carro obj = new Carro();

b) Carro obj = Carro.new;

c) Carro obj = Carro();

d) Carro = new Carro();

e) obj = Carro();



**Q22) [VUNESP IMESC 2013]** Analise o seguinte trecho de código em linguagem C#:

```
int x = 10;  
int y = 20;  
x += x == 20 ? x / y : y / x;  
y -= y == 10 ? y / x : x / y;
```

Após a execução desse trecho de código, o valor das variáveis “x” e “y” serão, respectivamente,

- a) 2 e 0.
- b) 10 e 18.
- c) 10 e 20.
- d) 12 e 19.
- e) 12 e 20.

**Q22) [VUNESP IMESC 2013]** Analise o seguinte trecho de código em linguagem C#:

```
int x = 10;  
int y = 20;  
x += x == 20 ? x / y : y / x;  
y -= y == 10 ? y / x : x / y;
```

Após a execução desse trecho de código, o valor das variáveis “x” e “y” serão, respectivamente,

- a) 2 e 0.
- b) 10 e 18.
- c) 10 e 20.
- d) 12 e 19.
- e) 12 e 20.**

```
if (x == 20){  
    x / y;  
} else {  
    y / x;  
}  
if (y == 10){  
    y / x;  
} else {  
    x / y;  
}
```

**Q23) [VUNESP PRODEST-ES 2014]** Na linguagem de programação C#, a declaração dos tipos e de seus membros permite que seja determinada a sua visibilidade por meio de modificadores de acesso. Os modificadores disponíveis para esse fim são:

- a) default, open, closed, partial e full.
- b) full-access, write, write-only, read e read-only.
- c) global, local, nested e virtual.
- d) public, private, published e protected.
- e) public, private, protected, internal e protected internal.

**Q23) [VUNESP PRODEST-ES 2014]** Na linguagem de programação C#, a declaração dos tipos e de seus membros permite que seja determinada a sua visibilidade por meio de modificadores de acesso. Os modificadores disponíveis para esse fim são:

- a) default, open, closed, partial e full.
- b) full-access, write, write-only, read e read-only.
- c) global, local, nested e virtual.
- d) public, private, published e protected.
- e) public, private, protected, internal e protected internal.

**Q24) [VUNESP MPE-ES 2013]** Na linguagem C#, é possível dividir a definição de uma classe em diversos arquivos. Para tanto, é necessário que a declaração da classe contenha a palavra chave:

- a) split.
- b) partial.
- c) external.
- d) continue.
- e) abstract.

**Q24) [VUNESP MPE-ES 2013]** Na linguagem C#, é possível dividir a definição de uma classe em diversos arquivos. Para tanto, é necessário que a declaração da classe contenha a palavra chave:

a) split.

b) partial.

c) external.

d) continue.

e) abstract.

**Q25) [VUNESP MPE-ES 2013]** Na linguagem C#, a forma correta de declarar a classe B, derivada da classe A, é:

- a) `public class B inherits A { }`
- b) `public class B => A { }`
- c) `public class A extends B { }`
- d) `public class B : A { }`
- e) `public class B implements A { }`

**Q25) [VUNESP MPE-ES 2013]** Na linguagem C#, a forma correta de declarar a classe B, derivada da classe A, é:

- a) `public class B inherits A { }`
- b) `public class B => A { }`
- c) `public class A extends B { }`
- d) `public class B : A { }`
- e) `public class B implements A { }`



**Q26) [VUNESP MPE-ES 2013]** Na linguagem C#, a palavra reservada “*sealed*” pode ser utilizada na declaração de classes. Ela tem a função de

- a) indicar que a classe possui métodos que precisam ser sobrescritos.
- b) impedir que a classe seja instanciada mais de uma vez.
- c) impedir que a classe seja derivada por outras classes.
- d) garantir que a classe não seja instanciada por classes que não estejam no mesmo namespace.
- e) indicar que o conteúdo da classe é imutável, isto é, uma vez instanciada, seu conteúdo não é mais alterado.

**Q26) [VUNESP MPE-ES 2013]** Na linguagem C#, a palavra reservada “*sealed*” pode ser utilizada na declaração de classes. Ela tem a função de

- a) indicar que a classe possui métodos que precisam ser sobrescritos.
- b) impedir que a classe seja instanciada mais de uma vez.
- c) impedir que a classe seja derivada por outras classes.
- d) garantir que a classe não seja instanciada por classes que não estejam no mesmo namespace.
- e) indicar que o conteúdo da classe é imutável, isto é, uma vez instanciada, seu conteúdo não é mais alterado.

**Q27) [AOCP TCE PA 2012]** Em C#, os métodos chamados pelo mecanismo de execução do programa quando o objeto está prestes a ser removido da memória são denominados de

- a) Garbage Collection.
- b) Destruidores.
- c) Propriedades.
- d) Indexadores.
- e) Eventos.

**Q27) [AOCP TCE PA 2012]** Em C#, os métodos chamados pelo mecanismo de execução do programa quando o objeto está prestes a ser removido da memória são denominados de

a) Garbage Collection.

b) Destruidores.

c) Propriedades.

d) Indexadores.

e) Eventos.

**Q28) [AOCP TCE PA 2012]** Segundo a Microsoft, um conjunto de recursos introduzidos no Visual Studio 2010 que estende as capacidades de consultas à sintaxe da linguagem de C# e Visual Basic é conhecido como

- a) LINQ
- b) T-SQL.
- c) OQL.
- d) ADO.
- e) ASP.

**Q29) [AOCP TCE PA 2012 ADAP]** É a única linguagem de programação que suporta herança múltipla pura, ou seja, cada classe pode herdar características de uma ou mais classes.

**Q30) [AOCP TCE PA 2012 ADAP]** É uma linguagem de programação orientada a objetos, desenvolvida pela Microsoft como parte da plataforma .NET caracterizada por ser fracamente tipada.

**Q28) [AOCP TCE PA 2012]** Segundo a Microsoft, um conjunto de recursos introduzidos no Visual Studio 2010 que estende as capacidades de consultas à sintaxe da linguagem de C# e Visual Basic é conhecido como

- a) LINQ
- b) T-SQL.
- c) OQL.
- d) ADO.
- e) ASP.

**Q29) [AOCP TCE PA 2012 ADAP]** É a única linguagem de programação que suporta herança múltipla pura, ou seja, cada classe pode herdar características de uma ou mais classes. **ERRADO.**

**Q30) [AOCP TCE PA 2012 ADAP]** É uma linguagem de programação orientada a objetos, desenvolvida pela Microsoft como parte da plataforma .NET caracterizada por ser fracamente tipada. **ERRADO.**

**Q31) [FGV DPE-RJ 2014]** Considere o código escrito na linguagem C# mostrado a seguir.

```
using System.IO;
using System;

public class Veiculo
{ public virtual void mover()
  { Console.Write("Movendo");
  }
}

public class Automovel:Veiculo
{ public override void mover()
  { Console.Write("Acelerando");
  }
}

public class Fusca:Automovel
{ public override void mover()
  { Console.Write ("Passeando");
  }
}

class Program
{ static void Main()
  { Veiculo veiculo = new Fusca();
    veiculo.mover();
  }
}
```

O resultado produzido pela execução desse código é

- a) Acelerando.
- b) Passeando.
- c) Movendo.
- d) MovendoAcelerandoPasseando.
- e) AcelerandoPasseando.

**Q31) [FGV DPE-RJ 2014]** Considere o código escrito na linguagem C# mostrado a seguir.

```
using System.IO;
using System;

public class Veiculo
{ public virtual void mover()
  { Console.Write("Movendo");
  }
}

public class Automovel:Veiculo
{ public override void mover()
  { Console.Write("Acelerando");
  }
}

public class Fusca:Automovel
{ public override void mover()
  { Console.Write ("Passeando");
  }
}

class Program
{ static void Main()
  { Veiculo veiculo = new Fusca();
    veiculo.mover();
  }
}
```

O resultado produzido pela execução desse código é

- a) Acelerando.
- b) Passeando.**
- c) Movendo.
- d) MovendoAcelerandoPasseando.
- e) AcelerandoPasseando.



**Q32) [CESGRANRIO EPE 2010]** Determinado órgão público federal deseja implantar um sistema de consulta na Internet. A plataforma utilizada será ASP.NET e a linguagem de programação, C#. Na modelagem orientada a objetos desse sistema, é importante considerar que a linguagem adotada

- a) impede, no contexto de instanciação de objetos, o uso de classes abstratas.
- b) impede somente o uso de polimorfismo a fim de assegurar a legibilidade do código.
- c) apresenta o conceito de namespaces para implementar associações entre classes.
- d) proíbe o uso de interfaces para garantir a coesão e a modularidade do código.
- e) implementa, no âmbito da generalização, somente herança simples

**Q32) [CESGRANRIO EPE 2010]** Determinado órgão público federal deseja implantar um sistema de consulta na Internet. A plataforma utilizada será ASP.NET e a linguagem de programação, C#. Na modelagem orientada a objetos desse sistema, é importante considerar que a linguagem adotada

- a) impede, no contexto de instanciação de objetos, o uso de classes abstratas.
- b) impede somente o uso de polimorfismo a fim de assegurar a legibilidade do código.
- c) apresenta o conceito de namespaces para implementar associações entre classes.
- d) proíbe o uso de interfaces para garantir a coesão e a modularidade do código.
- e) implementa, no âmbito da generalização, somente herança simples

**Q33) [CESGRANRIO EPE 2012]** Na programação orientada a objeto, na linguagem C# em particular, a capacidade de construir vários métodos com um mesmo nome, porém com parâmetros diferentes na mesma classe, é chamada de

- a) Polimorfismo universal
- b) Polimorfismo paramétrico
- c) Polimorfismo de subtipo
- d) Sobrecarga de operadores
- e) Sobrecarga de métodos

**Q33) [CESGRANRIO EPE 2012]** Na programação orientada a objeto, na linguagem C# em particular, a capacidade de construir vários métodos com um mesmo nome, porém com parâmetros diferentes na mesma classe, é chamada de

- a) Polimorfismo universal
- b) Polimorfismo paramétrico
- c) Polimorfismo de subtipo
- d) Sobrecarga de operadores
- e) Sobrecarga de métodos

**Q34) [CESGRANRIO EPE 2014]** Observe o código C# abaixo, encontrado em uma implementação de um sistema em ASP.Net.

```
namespace Loja {  
    public class Product {  
        public int IdProduto { get; set; }  
        public string Nome { get; set; }  
        public string Descricao { get; set; }  
        public decimal Preco { get; set; }  
        public string Categoria { set; get; }  
    }  
}
```

As cinco declarações presentes nesse código são exemplos de

- a) variáveis dinâmicas
- b) propriedades automáticas
- c) mensagens abstratas
- d) métodos abstratos
- e) inicializadores automáticos

**Q34) [CESGRANRIO EPE 2014]** Observe o código C# abaixo, encontrado em uma implementação de um sistema em ASP.Net.

```
namespace Loja {  
    public class Product {  
        public int IdProduto { get; set; }  
        public string Nome { get; set; }  
        public string Descricao { get; set; }  
        public decimal Preco { get; set; }  
        public string Categoria { set; get; }  
    }  
}
```

As cinco declarações presentes nesse código são exemplos de

a) variáveis dinâmicas

b) propriedades automáticas

c) mensagens abstratas

d) métodos abstratos

e) inicializadores automáticos

# GABARITO

Q1 – ERRADO. Q14 – LETRA B.  
Q2 - LETRA E. Q15 – LETRA B.  
Q3 – LETRA A. Q16 – CERTO.  
Q4 - LETRA D. Q17 – LETRA E.  
Q5 - LETRA A. Q18 – LETRA A.  
Q6 - LETRA E. Q19 - LETRA B.  
Q7 – LETRA A. Q20 - LETRA A.  
Q8 – LETRA C. Q21 - LETRA A.  
Q9 - LETRA D. Q22 - LETRA E.  
Q10 – LETRA A. Q23 - LETRA E.  
Q11 – LETRA D. Q24 - LETRA B.  
Q12 - LETRA A. Q25 – LETRA D.  
Q13 - LETRA A. Q26 - LETRA C.  
Q27 – LETRA B. Q28 – LETRA A.  
Q29 - ERRADO Q30 – ERRADO.  
Q31 - LETRA B. Q32 - LETRA E.  
Q33 – LETRA E. Q34 – LETRA B.