C#

GRUPO:

Rafael Igor

Vanderlei Vieira

INTRODUÇÃO

- Paradigmas:
  - Orientado a objeto;
  - Estruturado;
  - Imperativo;
- Surgimento: 2002.
- Criador: Anders Hejlsberg.
- Principais Compiladores: .NET Framework (Windows), Mono (UNIX, MAC, Windows).
- Principais Influências: Java, C++, Modula-3.
- Tipagem: estática e dinâmica, forte, segura e insegura
- Página Oficial: msdn.microsoft.com

HISTÓRIA

- No final da década de 90 a Microsoft tinha diversas linguagens de programação para resolver muitos problemas diferentes.
  - Problema gerado: Migrar para uma nova linguagem se tornou trabalhoso para os programadores.
  - Solução: Microsoft recorreu a linguagem Java, criando sua própria implementação, o J++.

HISTÓRIA

- O J++ só poderia ser executada em Windows. Isso fez com que a Sun Microsystems processasse a Microsoft por questões de violação de licenciamento.
- A Microsoft começou então a trabalhar em uma nova plataforma que seria a base de todas as suas soluções, o .Net. Nela pode-se trabalhar com diversas linguagens de programação, assim diversas linguagens diferentes compartilhariam o mesmo conjunto de bibliotecas.

HISTÓRIA

- Desenvolvimento de um novo projeto de linguagem chamada COOL (C-like Object Oriented Language).
- Design baseado em diversas outras linguagens do mercado como Java, C, C++, Smalltalk, Delphi e VB.
- Em 2002, o projeto COOL foi lançado como linguagem C# 1.0 juntamente com o ambiente .Net 1.0.

O que fazer para começar a aprender C#?

- Microsoft .Net Framework 4.6:
  https://www.microsoft.com/en-us/download/details.aspx?id=44928

- Visual Studio (C# 5.0)
  https://www.visualstudio.com/products/visual-studio-2015-downloads-vs

O que fazer para começar a aprender C#?

Para outras plataformas, como Linux e Mac, temos a plataforma **Mono** e IDE **MonoDevelop.**

Download no Mac:

- mono-project.com/download/#download-mac

Instalação no Ubuntu 14.04:

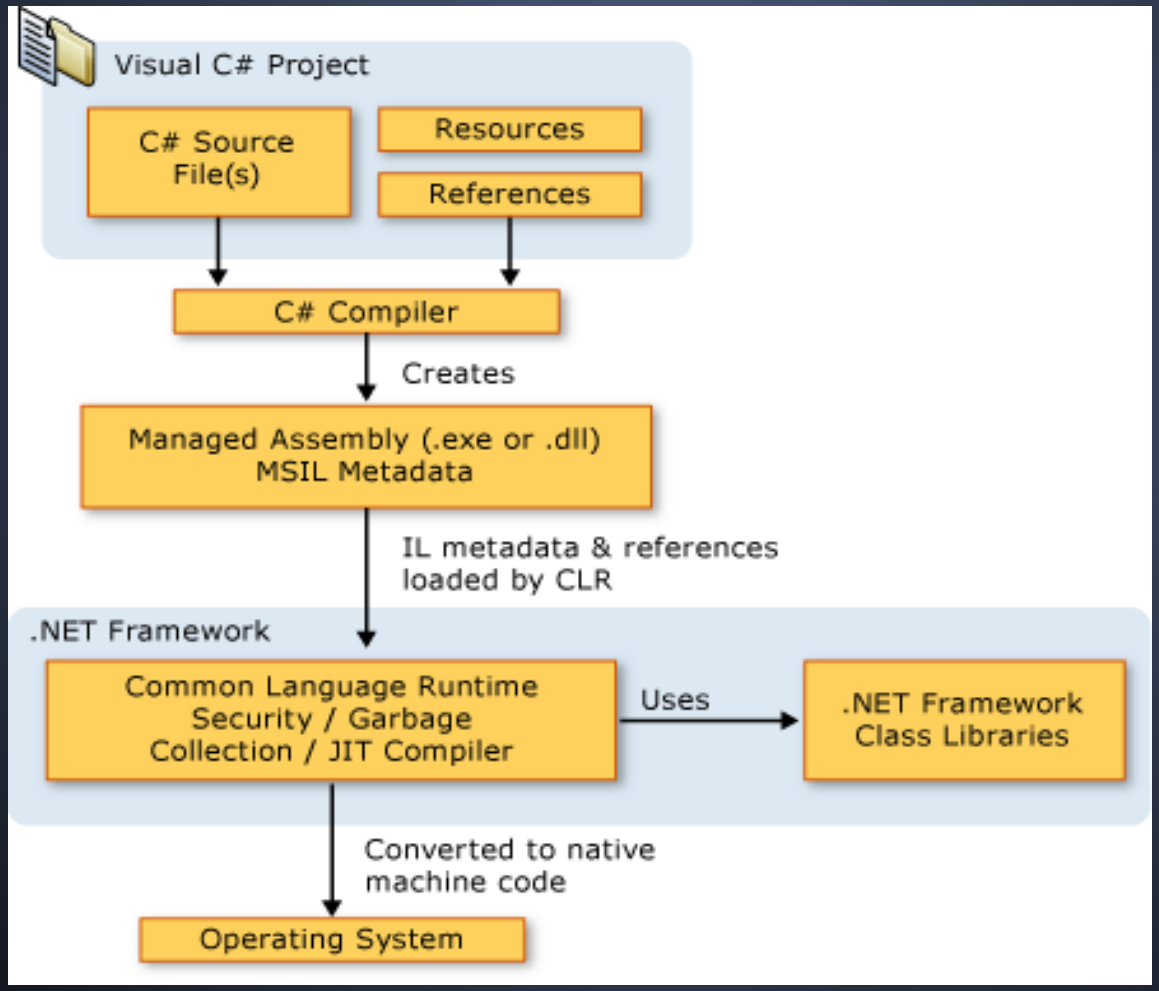
```
$> sudo apt-get install software-properties-common
$> sudo add-apt-repository ppa:inizan-yannick/mono
$> sudo apt-get update
$> sudo apt-get install mono-devel
```

Compilação e execução por linha de comando:

```
$> mcs exemplo.cs
$> mono exemplo.exe
```

Compilação

- Compilador híbrido



Amarrações

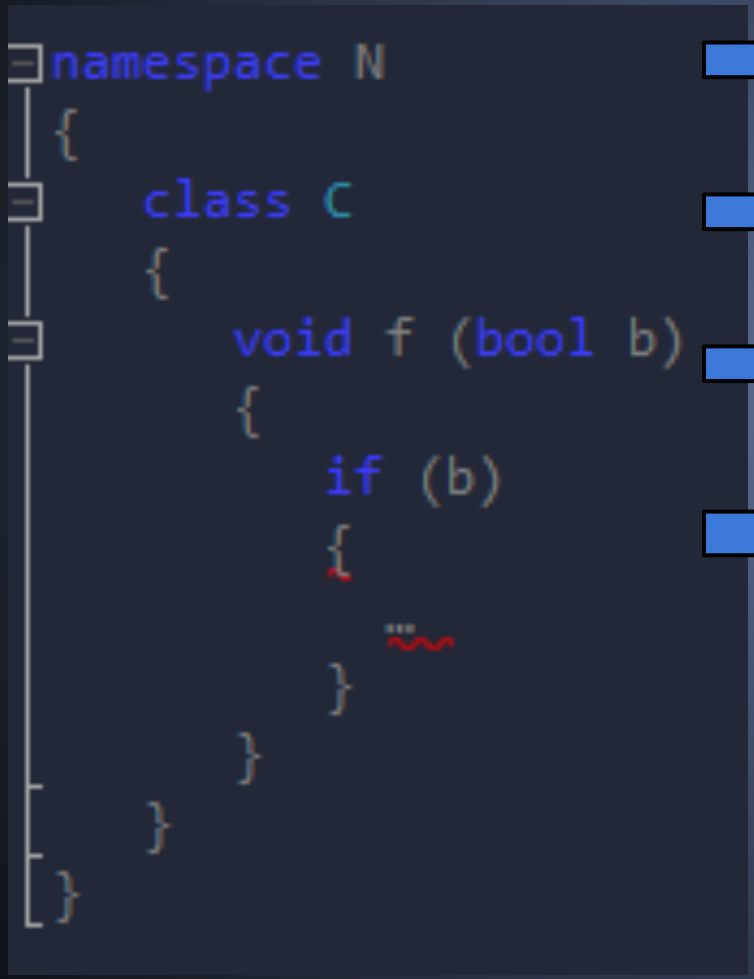
```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6
7  namespace LP
8  {
9      class Client
10     {
11         static void Main(string[] args)
12         {
13             int i = 10;
14             double I = 20.5;
15             string str = "Hello World!";
16             Console.WriteLine(str);
17         }
18     }
19 }
20
```

Amarrações

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6
7  namespace LP
8  {
9      class Client
10     {
11         static void Main(string[] args)
12         {
13             int i = 10;
14             double I = 20.5;
15             string str = "Hello World!";
16             Console.WriteLine(str);
17         }
18     }
19 }
20
```

C# é case sensitive

Escopo



Escopo de Namespace
- Identificadores de grupos

Escopo de Classe
- define/declara váriáveis e funções

Escopo de bloco mais externo (Função)
- inclui instruções executáveis

Escopo de bloco mais interno
- declarações executadas condicionalmente

C# possui escopo estático

Tipos de Dados

- Tipos Valores
  - Primitivos
  - Enumerados
  - Estruturas
- Tipos Referência
  - Arrays
  - Classes
  - Interfaces
  - Delegates

Tipos de Dados e Valores

- Tipos Primitivos

Tipo	Tamanho	Valores Possíveis
bool	1 byte	true e false
byte	1 byte	0 a 255
sbyte	1 byte	-128 a 127
short	2 bytes	-32768 a 32767
ushort	2 bytes	0 a 65535
int	4 bytes	-2147483648 a 2147483647
uint	4 bytes	0 to 4294967295
long	8 bytes	–9223372036854775808L to 9223372036854775807L
ulong	8 bytes	0 a 18446744073709551615
float	4 bytes	Números até 10 elevado a 38. Exemplo: 10.0f, 12.5f
double	8 bytes	Números até 10 elevado a 308. Exemplo: 10.0, 12.33
decimal	16 bytes	números com até 28 casas decimais. Exemplo 10.991m, 33.333m
char	2 bytes	Caracteres delimitados por aspas simples. Exemplo: 'a', 'ç', 'o'

Tipos de Dados e Valores

- Tipos
  - Compostos
    - Strings

```
string str = "Te";
str += "st";
Console.WriteLine(str);
```

Tipos de Dados e Valores

- Tipos
  - Compostos
    - Strings

```
string str = "Te";
str += "st";
Console.WriteLine(str);
```

```
Test
```

Tipos de Dados e Valores

- Tipos
  - Compostos
    - Arrays

```
int[] numbers;
numbers = new int[10];                  //Single-dimensional arrays
string[,] names = new string[5, 4]; //Multidimensional arrays
int[, ,] buttons = new int[4, 5, 3];
string[,] siblings = { { "Mike", "Amy" }, { "Mary", "Albert" } };
Console.WriteLine(siblings[0, 1]);
```

Tipos de Dados e Valores

- Tipos
  - Compostos
    - Arrays

```
int[] numbers;
numbers = new int[10];                  //Single-dimensional arrays
string[,] names = new string[5, 4]; //Multidimensional arrays
int[, ,] buttons = new int[4, 5, 3];
string[,] siblings = { { "Mike", "Amy" }, { "Mary", "Albert" } };
Console.WriteLine(siblings[0, 1]);
```

```
Amy
```

Tipos de Dados e Valores

- Tipos
  - Compostos
    - Struct

```
public struct Ponto {
    public double x, y;

    public Ponto(double p1, double p2) {
      x = p1;
      y = p2;
    }
 }
class Cliente
{
    static void main(string[] args) {
        Ponto p = new Ponto(10.2,59.25);

    }
}
```

Tipos de Dados e Valores

- Tipos
  - Compostos
    - Enum

```
enum diasSemana { Domingo, Segunda, Terça, Quarta, Quinta, Sexta, Sábado };
   /*Por default
    * Domingo = 0
    * Segunda = 1
    * ...
    * Sábado = 6
    */
```

Delegate

```
delegate double MathAction(double num);

static double Double(double input)
{
    return input * 2;
}

static void Main(string[] args)
{
    MathAction metade = delegate(double input)
    {
        return input / 2;
    };
    MathAction areaCubo = s => s * s * s;
    MathAction dobrar = Double;

    double a = 3;
    Console.WriteLine(areaCubo(a));

    double b = 50.80;
    Console.WriteLine( metade(b));

    double c = 15.25;
    Console.WriteLine(dobrar(c));

}
```

Delegate

```
delegate double MathAction(double num);

static double Double(double input)
{
    return input * 2;
}

static void Main(string[] args)
{
    MathAction metade = delegate(double input)
    {
        return input / 2;
    };
    MathAction areaCubo = s => s * s * s;
    MathAction dobrar = Double;

    double a = 3;
    Console.WriteLine(areaCubo(a));

    double b = 50.80;
    Console.WriteLine( metade(b));

    double c = 15.25;
    Console.WriteLine(dobrar(c));

}
```

```
27
25,4
30,5
```

Variáveis e Constantes

```
int i = 10;
string str = "Hello World!";
double doub = 2.658;
decimal dec = 17.31m;

const double pi = 3.14159265359;
const int limiteVelocidade = 90;
```

Variáveis

- Var
  - Permite a tipagem em tempo de compilação dependendo do primeiro valor atribuído a variável.

```
var a = 10;                 // a é implicitamente tipada para int
var b = "Hello World!";     // b é implicitamente tipada para string
var c = 25.5m;              // c é implicitamente tipada para decimal

c = a;      // Ok!
c = b;      // Erro de conversão implicita!
a = c;      // Idem.
```

Variáveis

- Dynamic
  - Permite a tipagem em tempo de execução

```
dynamic a = 10;
dynamic b = "Hello World!";
dynamic c = 25.5m;

c = a;      // c muda de tipo em tempo de execução para int
Console.WriteLine(c);
c = b;      // c muda de tipo em tempo de execução para string
Console.WriteLine(c);
```

Variáveis

- Dynamic
  - Permite a tipagem em tempo de execução

```
dynamic a = 10;
dynamic b = "Hello World!";
dynamic c = 25.5m;

c = a;      // c muda de tipo em tempo de execução para int
Console.WriteLine(c);
c = b;      // c muda de tipo em tempo de execução para string
Console.WriteLine(c);
```

```
10
Hello World!
```

Operadores

Category	Operators
Primary	`x.y  f(x)  a[x]  x++  x--  new typeof checked  unchecked`
Unary	`+  -  !  ~  ++x  --x  (T)x`
Multiplicative	`*  /  %`
Additive	`+  -`
Shift	`<<  >>`
Relational and type testing	`<  >  <=  >=  is  as`
Equality	`==  !=`
Logical	`AND &`
Logical	`XOR ^`
Logical	`OR |`
Conditional	`AND &&`
Conditional	`OR ||`
Conditional	`?:`
Assignment	`=  *=  /=  %=  +=  -=  <<=  >>=  &=  ^=  |=`

Palavras-chave reservadas

abstract	as	base	bool
break	byte	case	catch
char	checked	class	const
continue	decimal	default	delegate
do	double	else	enum
event	explicit	extern	false
finally	fixed	float	for
foreach	goto	if	implicit
in	in (generic modifier)	int	interface
internal	is	lock	long
namespace	new	null	object
operator	out	out (generic modifier)	override
params	private	protected	public
readonly	ref	return	sbyte
sealed	short	sizeof	stackalloc
static	string	struct	switch
this	throw	true	try
typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual
void	volatile	while	

Palavras-chave reservadas

abstract	as	base	bool
break	byte	case	catch
char	checked	class	const
continue	decimal	default	delegate
do	double	else	enum
event	explicit	extern	false
finally	fixed	float	for
foreach	goto	if	implicit
void	volatile	while	

```
bool condition = true;

if (condition)
{
    Console.WriteLine("The variable is set to true.");
}
else
{
    Console.WriteLine("The variable is set to false.");
}
```

Palavras-chave reservadas

abstract	as	base	bool
break	byte	case	catch
char	checked	class	const
continue	decimal	default	delegate
do	double	else	enum
event	explicit	extern	false
finally	fixed	float	for
foreach	goto	if	implicit
in	in (generic modifier)	int	interface
typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual
void	volatile	while	

```
int n = 1;
while (n < 6)
{
    Console.WriteLine("Current value of n is {0}", n);
    n++;
}
```

Palavras-chave reservadas

abstract	as	base	bool
break	byte	case	catch
char	checked	class	const
continue	decimal	default	delegate
do	double	else	enum
event	explicit	extern	false
finally	fixed	float	for
this	throw	true	try
typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual
void	volatile	while	

```
for (int i = 1; i <= 5; i++)
{
    Console.WriteLine(i);
}
```

Palavras-chave reservadas

abstract	as	base	bool
break	byte	case	catch
char	checked	class	const
continue			delegate
do			enum
event			false
finally			for
foreach			implicit
in			interface
internal			long
namespa			object
operator			override
params			public
readonly			sbyte
sealed			stackalloc
static			switch
this	throw	true	try
typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual
void	volatile	while	

```
int caseSwitch = 1;
switch (caseSwitch)
{
    case 1:
        Console.WriteLine("Case 1");
        break;
    case 2:
        Console.WriteLine("Case 2");
        break;
    default:
        Console.WriteLine("Default case");
        break;
}
```

Palavras-chave reservadas

abstract	as	base	bool
break	byte	case	catch
char	checked	class	const
continue	decimal	default	delegate
do	double	else	enum
event			
finally			
foreach			
in			
internal			
namespace			
operator			
params			
readonly			
sealed			
static	string	struct	switch
this	throw	true	try
typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual
void	volatile	while	

```
SortedDictionary<int, Aluno> alunos = new SortedDictionary<int, Aluno>();
try
{
    alunos.Add(2, new Aluno(326262, "Fulano", 21));
    alunos.Add(10, new Aluno(252145, "Beltrano", 55));
    alunos.Add(1, new Aluno(215248, "Sicrano", 18));
    alunos.Add(5, new Aluno(231525, "Arnold Schwarzenegger", 85));
}
catch (System.ArgumentException e)
{
    Console.WriteLine(e.Message);
}
finally
{
    Console.WriteLine("Sempre executa o Finnaly!");
}
```

Palavras-chave reservadas

abstract	as	base	bool
break	byte	case	catch
char	checked	class	const
continue	decimal	default	delegate
do	double	else	enum
event	explicit	extern	false
finally	fixed	float	for
foreach	goto	if	implicit
operator	out	out (generic modifier)	override
params	private	protected	public
readonly	ref	return	sbyte
sealed	short	sizeof	stackalloc
static	string	struct	switch
this	throw	true	try
typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual
void	volatile	while	

```
foreach (KeyValuePair<int, Aluno> p in alunos)
{
    Console.WriteLine("O Aluno " + p.Value.Nome + " possui a chave " + p.Key);
}
```

Palavras-chave reservadas

abstract	as	base	bool
break	byte	case	catch
char	checked	class	const
continue	decimal	default	delegate
do	double	else	enum
event	explicit	extern	false
finally	fixed	float	for
foreach	goto	if	implicit
operator			
params			
readonly			
sealed	short	sizeof	stackalloc
static	string	struct	switch
this	throw	true	try
typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual
void	volatile	while	

```
foreach (KeyValuePair<int, Aluno> p in alunos)
{
    Console.WriteLine("O Aluno " + p.Value.Nome + " possui a chave " + p.Key);
}
```

```
O Aluno Sicrano possui a chave 1
O Aluno Fulano possui a chave 2
O Aluno Arnold Schwarzenegger possui a chave 5
O Aluno Beltrano possui a chave 10
```

Palavras-chave não reservadas

add	alias	ascending
async	await	descending
dynamic	from	get
global	group	into
join	let	orderby
partial (type)	partial (method)	remove
select	set	value
var	where (generic type constraint)	where (query clause)
yield		

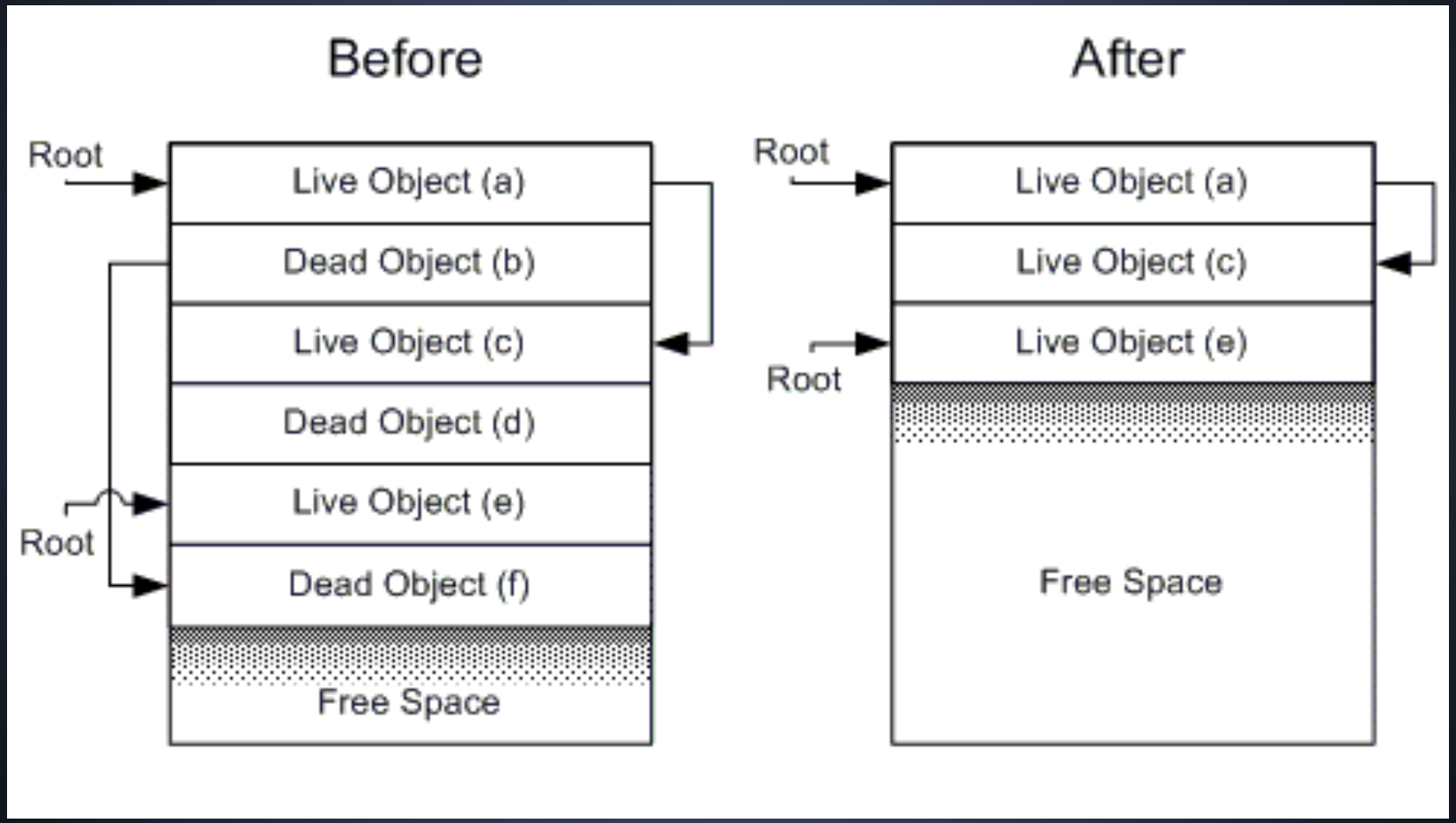
Palavras-chave não reservadas

```
private int idade;
public int Idade
{
    get
    {
        return idade;
    }
    set
    {
        idade = value;
    }

}
```

```
Aluno a1 = new Aluno(231525, "Arnold Schwarzenegger", 85);
a1.Idade = 86;
Console.WriteLine(a1.Idade);
```

		get
		into
		orderby
	nethod)	remove
select	set	value
var	where (generic type constraint)	where (query clause)
yield		

Garbage Collector (Coletor de lixo)



Passagem de Parâmetros

- Existem 4 formas diferentes de passar parâmetros em C#
  - Por Referencia:
    - Out
    - Ref
  - Por Valor:
    - Params
    - Value

Passagem de Parâmetros

- Out

```
static void traduzir(out string param)
{
    param = "Olá!";
}

static void Main(string[] args)
{

    string msg = "Hello!";
    traduzir(out msg);
    Console.WriteLine(msg);
}
```

Passagem de Parâmetros

- Out

```
static void traduzir(out string param)
{
    param = "Olá!";
}

static void Main(string[] args)
{

    string msg = "Hello!";
    traduzir(out msg);
    Console.WriteLine(msg);
}
```

```
Olá!
```

Passagem de Parâmetros

- Out

```
static void traduzir(out string param)
{
    string s = param; // Erro, param só poderá ser usado pra saída
    param = "Olá!";
}
```

Passagem de Parâmetros

- Ref

```
static void flamengoFaz(ref string s)
{
    s += " Vice";
}
static void Main(string[] args)
{
    string str = "Vasco";
    flamengoFaz(ref str);

    Console.WriteLine(str);
}
```

Passagem de Parâmetros

- Ref

```
static void flamengoFaz(ref string s)
{
    s += " Vice";
}
static void Main(string[] args)
{
    string str = "Vasco";
    flamengoFaz(ref str);

    Console.WriteLine(str);
}
```

```
Vasco Vice
```

Passagem de Parâmetros

- Value

```
static void imprimir(string param)
{
    Console.WriteLine(param);
}

static void Main(string[] args)
{

    imprimir("Hello!");
}
```

Passagem de Parâmetros

- Params

```
static int soma(params int[] Param1)
{
    int val = 0;
    foreach(int P in Param1)
    {
        val = val + P;
    }
    return val;
}
static void Main(string[] args)
{
     Console.WriteLine(soma(10,20,30));
}
```

Passagem de Parâmetros

- Params

```
static int soma(params int[] Param1)
{
    int val = 0;
    foreach(int P in Param1)
    {
        val = val + P;
    }
    return val;
}
static void Main(string[] args)
{
     Console.WriteLine(soma(10,20,30));
}
```

```
60
```

Classe

```
class Pessoa
{
    internal string nome;
    private int idade;
    public int Idade
    {
        get
        {
            return idade;
        }
        set
        {
            idade = value;
        }
    }
    public Pessoa() { }

    public Pessoa(string nome, int idade)
    {
        this.nome = nome;
        Idade = idade;
    }
}
```

Classe Abstrata

```
namespace LP
{
    abstract class Forma
    {
        abstract public double area();
    }

    class Quadrado : Forma
    {
        internal double lado;

        public Quadrado(double lado)
        {
            this.lado = lado;
        }

        public override double area()
        {
            return lado * lado;
        }
    }

}
```

Namespace

- Ajuda na organização das classes.
- Não estão ligados a pastas (diferente dos packages em Java).
  - pastas com classes de namespace diferentes do nome da pasta.
  - pastas com classas de namespaces diferentes.

Namespace

```
namespace LP
{
    class Aluno : Pessoa, IAluno
    {
        private int matricula;
        public int Matricula
        {
            get
            {
                return matricula;
            }
         }

        internal List<double> notas;

        public Aluno(int matricula,string nome, int idade)
        {
            this.nome = nome;
            Idade = idade;
            this.matricula = matricula;
            notas = new List<double>();
        }

        public void addNota(double nota)
        {
            this.notas.Add(nota);
        }

        public override string ToString()
        {
            return "O Aluno " + this.nome + " possui a matricula " + this.Matricula;
        }
    }
}
```

Namespace

```
class Client
{
    static void Main(string[] args)
    {
        LP.Aluno a1 = new LP.Aluno(19746958, "Arnold Schwarzenegger", 85);

    }
}
```

Namespaces são identificados pelo uso do operador ‘.’
(classe Aluno pertence ao namespace “LP”)

Namespace

```
namespace LP
{
    class Client
    {
        static void Main(string[] args)
        {
            Aluno a1 = new Aluno(19746958, "Arnold Schwarzenegger", 85);

        }
    }
}
```

Namespace

- C# possui a palavra reservada using para importar classes e evitar repetições
- exemplo:

```
System.Console.WriteLine("Hello!");
```

```
using System;
Console.WriteLine("Hello!");
```

Criando Objetos

```
Aluno a1 = new Aluno(19746958, "Arnold Schwarzenegger", 85);
a1.addNota(5.5);

Console.WriteLine(a1);
```

Criando Objetos

```
Aluno a1 = new Aluno(19746958, "Arnold Schwarzenegger", 85);
a1.addNota(5.5);

Console.WriteLine(a1);
```

O método toString será chamado!

Criando Objetos

```
Aluno a1 = new Aluno(19746958, "Arnold Schwarzenegger", 85);
a1.addNota(5.5);

Console.WriteLine(a1);
```

```
O Aluno Arnold Schwarzenegger possui a matricula 19746958
```

Leitura e Escrita em Arquivos

- Leitura:

```
try{
    string[] linhas = System.IO.File.ReadAllLines(@"D:\Documents\text.txt");

    foreach (string lin in linhas)
        Console.WriteLine(lin);

    using (StreamReader sr = new StreamReader(@"D:\Documents\text.txt") )
    {
        string line = sr.ReadToEnd();
        Console.WriteLine(line);
    }
}
catch (System.IO.IOException e){
    Console.WriteLine(e.Message);
}
```

Ambos os procedimentos imprimem na tela o conteúdo do arquivo text.txt.

Leitura e Escrita em Arquivos

- Leitura:
  - Se o arquivo não for encontrado, uma exceção será lançada e a seguinte mensagem irá aparecer

```
Não foi possível localizar o arquivo 'D:\Documents\text.txt'.
```

Leitura e Escrita em Arquivos

- Escrita:

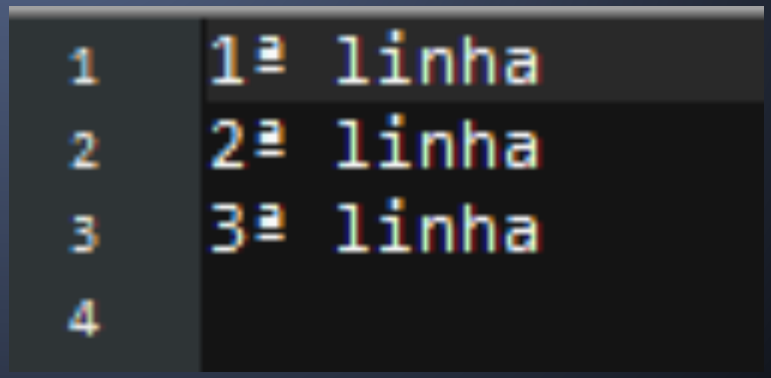
```
string[] lines = { "1ª linha", "2ª linha", "3ª linha" };
File.WriteAllLines(@"D:\Documents\Rafael\C++\text.txt", lines);

string text = "1ª linha\n" + "2ª linha\n" + "3ª linha\n";
File.WriteAllText(@"D:\Documents\Rafael\C++\text.txt", text);
```

Ambos os procedimentos escrevem no arquivo text.txt o mesmo texto.

Leitura e Escrita em Arquivos

- Escrita:

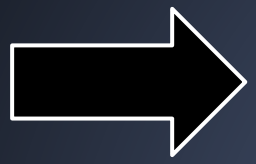
```
string[] lines = { "1ª linha", "2ª linha", "3ª linha" };
File.WriteAllLines(@"D:\Documents\Rafael\C++\text.txt", lines);

string text = "1ª linha\n" + "2ª linha\n" + "3ª linha\n";
File.WriteAllText(@"D:\Documents\Rafael\C++\text.txt", text);
```



Leitura e Escrita em Arquivos

- Escrita:
  - Se o arquivo que será escrito não existir ele será criado.
  - Se o arquivo já existir ele será sob escrito.

Leitura e Escrita em Arquivos



Todas as funções mostradas anteriormente abrem e fecham os arquivos automaticamente.

Polimorfismo

- Tipos de polimorfismo
  - Ad-hoc
    - Sobrecarga
    - Coerção
  - Universal
    - Paramétrico
    - Inclusão

Polimorfismo

- Sobrecarga

```
class Complex
{
    protected double real;
    protected double imaginario;

    public Complex(double real, double imaginario)
    {
        this.real = real;
        this.imaginario = imaginario;
    }
    public static Complex operator +(Complex c1, Complex c2)
    {
        return new Complex(c1.real + c2.real, c1.imaginario + c2.imaginario);
    }
    public override string ToString()
    {
        return real + " + " + imaginario + "i";
    }
}
```

Polimorfismo

- Sobrecarga

```
Complex x = new Complex(25.5, 3);
Complex y = new Complex(14.5, 7);

Complex z = x + y;

Console.WriteLine(z);
```

Polimorfismo

- Sobrecarga

```
Complex x = new Complex(25.5, 3);
Complex y = new Complex(14.5, 7);

Complex z = x + y;

Console.WriteLine(z);
```

```
40 + 10i
```

Polimorfismo

- Coerção
  - Conversão Implícita

```
double x = 25.9, y = 23.78;
int z = x + y;
```

Polimorfismo

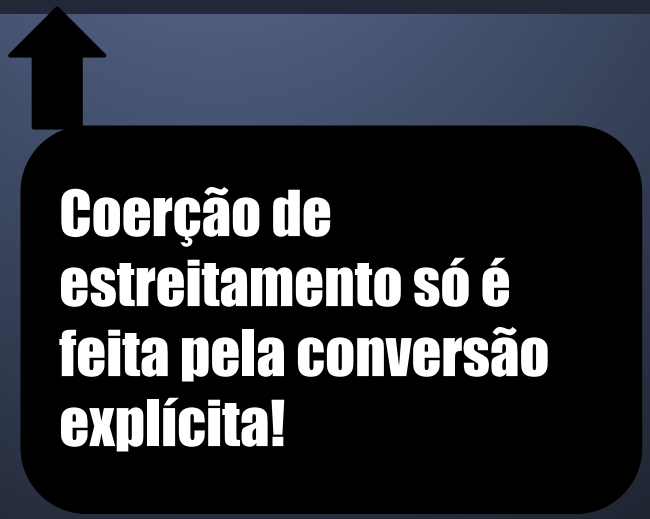
- Coerção
  - Conversão Implícita

```
double x = 25.9, y = 23.78;
int z = x + y;
```

Polimorfismo

- Coerção
  - Conversão Implícita

```
double x = 25.9, y = 23.78;
int z = (int) (x + y);
```



Polimorfismo

- Coerção
  - Conversão Implícita

```
int x = 25, y = 23;
double z = x + y;
```

Coerção de ampliação é aceita em C#!!

Polimorfismo

- Paramétrica

```
static void swap<T>(ref T x, ref T y)
{
    T aux;
    aux = x;
    x = y;
    y = aux ;
}
static void Main(string[] args)
{

    int x = 2, y = 99;
    swap<int>(ref x, ref y);

    Console.WriteLine(" x = {0}  e y = {1}", x, y);

}
```

Polimorfismo

- Inclusão
  - Herança
    - C# não suporta herança múltipla.

Herança

```
class Aluno : Pessoa, IAluno
{
    private int matricula;
    public int Matricula
    {
        get
        {
            return matricula;
        }
     }

    internal List<double> notas;

    public Aluno(int matricula,string nome, int idade)
    {
        this.nome = nome;
        Idade = idade;
        this.matricula = matricula;
        notas = new List<double>();
    }

    public void addNota(double nota)
    {
        this.notas.Add(nota);
    }

    public override string ToString()
    {
        return "O Aluno " + this.nome + " possui a matricula " + this.Matricula;
    }
}
```

Interface

```
class Aluno : Pessoa, IAluno
{
    private int matricula;
    public int Matricula
    {
        get
        {
            return matricula;
        }
     }

    internal List<double> notas;

    public Aluno(int matricula,string nome, int idade)
    {
        this.nome = nome;
        Idade = idade;
        this.matricula = matricula;
        notas = new List<double>();
    }

    public void addNota(double nota)
    {
        this.notas.Add(nota);
    }

    public override string ToString()
    {
        return "O Aluno " + this.nome + " possui a matricula " + this.Matricula;
    }
}
```

```
interface IAluno
{
    string ToString();
    void addNota(double nota);
}
```

Exceções

- O tratamento de exceções em C# é semelhante ao de C++.

```
static double div(double x, double y)
{
    if(y == 0)
        throw new System.DivideByZeroException();

    return x / y;
}
static void Main(string[] args)
{

    try
    {
        double q = div(5, 0);
    }
    catch (System.DivideByZeroException e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Exceções

- O tratamento de exceções em C# é semelhante ao de C++.

```
static double div(double x, double y)
{
    if(y == 0)
        throw new System.DivideByZeroException();

    return x / y;
}
static void Main(string[] args)
{

    try
    {
        double q = div(5, 0);
    }
    catch (System.DivideByZeroException e)
    {
        Console.WriteLine(e.Message);
    }
}
```

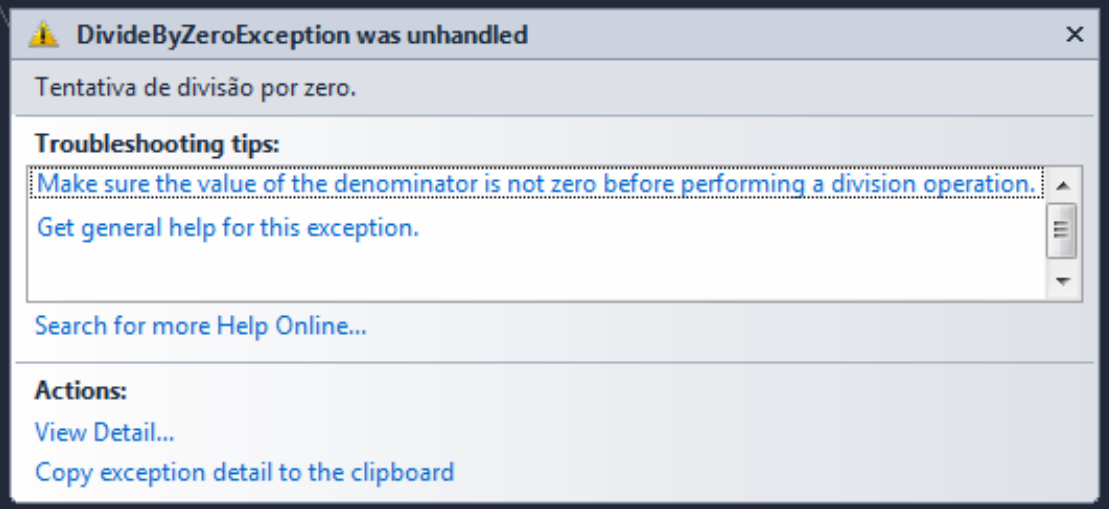
```
Tentativa de divisão por zero.
```

Exceções

- Se uma exceção for lançada e não for tratada o programa é interrompido na execução. Isto quer dizer que, o compilador não faz o trabalho de checar se todas as possíveis exceções lançadas serão tratadas ou não.

Exceções

```
static double div(double x, double y)
{
    if(y == 0)
        throw new System.DivideByZeroException();

    return x / y;
}
static void Main(string[] args)
{
    double q = div(5, 0);
}
```



Concorrência

- Threads:
  - Para manipulações de Threads, C# possui o namespace System.Threading.
  - Nela há uma série de métodos, como: Sleep, Suspend, Name, etc.
  - Semáforos podem ser usados pelo namespace System.Threading.Semaphore.

Concorrência

```
private static Semaphore _recurso;
public static void Main()
{
    // semaforo para 3 requisições concorrentes
    _recurso = new Semaphore(0, 3);

    // Cria 5 threads numeradas
    for(int i = 1; i <= 5; i++)
    {
        Thread t = new Thread(new ParameterizedThreadStart(Worker));
        t.Start(i);
    }
    Thread.Sleep(500);
    Console.WriteLine("Main thread calls Release(3).");
    _recurso.Release(3);

    Console.WriteLine("Main thread exits.");
}
private static void Worker(object num)
{
    // cada trabalho começa requisitando o semáforo
    Console.WriteLine("Thread {0} begins " +
        "and waits for the semaphore.", num);
    _recurso.WaitOne();

    Console.WriteLine("Thread {0} enters the semaphore.", num);

    Thread.Sleep(1000);

    Console.WriteLine("Thread {0} releases the semaphore.", num);
    Console.WriteLine("Thread {0} previous semaphore count: {1}",
        num, _recurso.Release());
}
```

Concorrência

- Saida...

```
Thread 1 begins and waits for the semaphore.
Thread 2 begins and waits for the semaphore.
Thread 3 begins and waits for the semaphore.
Thread 4 begins and waits for the semaphore.
Thread 5 begins and waits for the semaphore.
Main thread calls Release(3).
Thread 5 enters the semaphore.
Main thread exits.
Thread 2 enters the semaphore.
Thread 4 enters the semaphore.
Thread 4 releases the semaphore.
Thread 4 previous semaphore count: 0
Thread 3 enters the semaphore.
Thread 5 releases the semaphore.
Thread 2 releases the semaphore.
Thread 2 previous semaphore count: 0
Thread 1 enters the semaphore.
Thread 5 previous semaphore count: 0
Thread 3 releases the semaphore.
Thread 3 previous semaphore count: 1
Thread 1 releases the semaphore.
Thread 1 previous semaphore count: 2
```

Avaliação da Linguagem

Critérios Específicos	C#	C++	Java
Aplicabilidade	Sim	Sim	Parcial
Confiabilidade	Parcial	Não	Sim
Aprendizado	Não	Não	Não
Eficiência	Parcial	Sim	Parcial
Partabilidade	Sim	Não	Sim
Método de projeto	OO	Estruturado e OO	OO
Evolutibilidade	Sim	Parcial	Sim
Reusabilidade	Sim	Sim	Sim
Integração	Sim	Sim	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Avaliação da Linguagem

Critérios Específicos	C#	C++	Java
Escopo	Sim	Sim	Sim
Expressões e comandos	Sim	Sim	Sim
Tipos primitivos e compostos	Sim	Sim	Sim
Gerenciamento de memória	Sistema	Programador	Sistema
Persistência de dados	Biblioteca de classes e funções	Biblioteca de classes e funções	JDBC, biblioteca de classes, serilização
Passagem de parâmetros	Por valor, por referência, lista variável e default	Por valor, por referência, lista variável e default	Lista variável, por valor e por cópia de referência
Encapsulamento	Sim	Sim	Sim
Sistema de tipos	Parcial	Parcial	Sim
Verificação de tipos	Estática/Dinâmica	Estática/Dinâmica	Estática/Dinâmica
Polimorfismo	Todos	Todos	Todos

Avaliação da Linguagem

Critérios Específicos	C#	C++	Java
Exceções	Parcial	Parcial	Sim
Concorrência	Sim	Não (biblioteca de funções)	Sim

Referências

- https://msdn.microsoft.com/
- https://linhadecodigo.com.br/
- https://www.caelum.com.br