Linguagem C#

Integrantes

Bruno Gama,

Elson Freitas,

Lucas Santana,

Rafael Franco,

Vinicius Risso.

História da Linguagem

- Desenvolvida pela Microsoft
- Arquitetura .NET
- Nome da Linguagem
- Criador da Linguagem

Criador da Linguagem

Anders Hejlsberg

Características da Linguagem

- Multiparadigma
- Alto nível de abstração
- Possui coletor de lixo
- Tipagem dinâmica e estática
- Vinculada ao framework .NET

Objetivos da Linguagem

- Destina-se ser simples e moderna
- Destina-se ser utilizada para desenvolvimento em ambientes distribuídos
- Portabilidade

Hello World

- Windows
  - https://www.microsoft.com/net/learn/dotnet/hello-world-tutorial
- Linux
  - sudo apt-get install monodevelop mono-mcs mono-gmcs

Hello World

```
using System;

public class HelloWorld
{
   static public void Main ()
   {
      Console.WriteLine("Hello World");
   }
}
```

Compilando e executando

- Compilando
  - gmcs teste.cs
- Executando
  - ./teste.exe

Identificadores

Identificadores

- C# é **case sensitive**
- Não é permitido caracteres especiais
- É permitido usar palavras-chave utilizando ‘@’ na frente do identificador
- Não limita número máximo de caracteres para identificadores

Palavras-chave

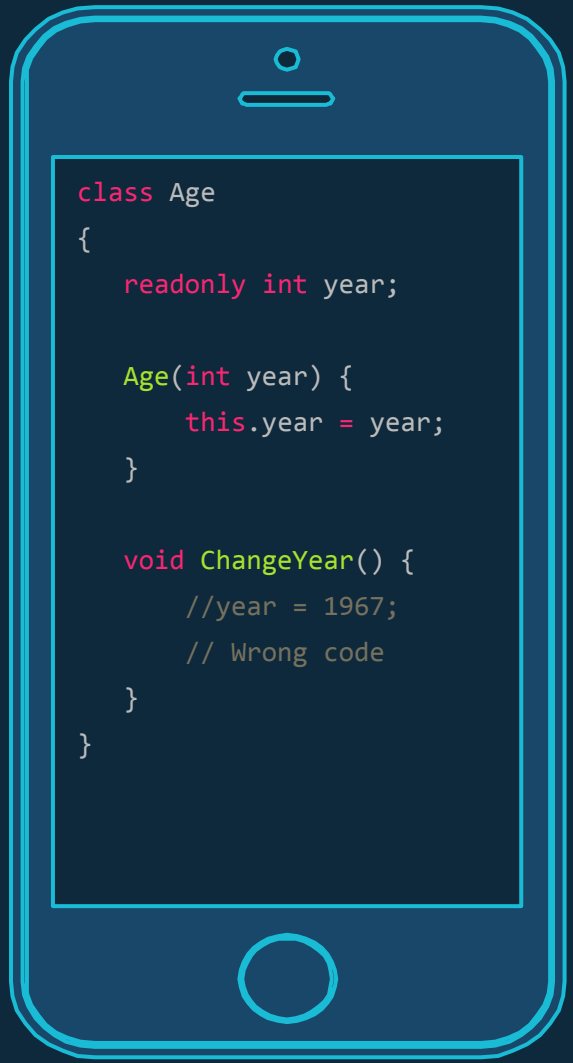
abstract	as	base	bool	break	byte	case	catch
char	checked	class	const	continue	decimal	default	delegate
do	double	else	enum	event	explicit	extern	false
finally	fixed	float	for	foreach	goto	if	implicit
in	int	interface	internal	is	lock	long	namespace
new	null	object	operator	out	override	params	private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

Palavras-chave contextuais

add	alias	ascending	async	await	by
descending	dynamic	equals	from	get	global
group	into	join	let	nameof	on
orderby	Parcial (tipo)	Partial (método)	remove	select	set
value	var	when (condição de filtro)	where (Restrição de tipo genérico)	where (Cláusula de consulta)	yield

Constantes

- Utilizar palavra-chave *const*
  - `const int x = 0;`
  - `public const double pi = 3.1415;`
  - `private const string nome = "Rambo";`
- Também pode-se utilizar *readonly*



```
class Age
{
    readonly int year;

    Age(int year) {
        this.year = year;
    }

    void ChangeYear() {
        //year = 1967;
        // Wrong code
    }
}
```

Definição de Tipos

- Struct, Interface e Enum

```
public struct CoOrds
{
   public int x, y;

   public CoOrds(int p1, int p2)
   {
       this.x = p1;
       this.y = p2;
   }

}
```

```
interface ISample
{
   void SampleMethod();
}

class Implementation : ISample
{
   void ISample.SampleMethod(){
       // Method implementation
   }
}
```

```
enum Day {
   Sat, // 0
   Sun, // 1
   Mon, // 2
   Tue, // 3
   Wed, // 4
   Thu, // 5
   Fri  // 6
};
```

Declarações de Variáveis

- `string nome;`
- `int inteiro = 1;`
- `float x;`
- `decimal numero;`
- `double valor;`
- `char letra = 'C';`
- `int[] vet = { 0, 1, 2, 3, 4, 5 };`
- `Int[,] matriz = {{1,2,3},{4,5,6}};`
- `bool condicao = true;`

Valores e Tipos de Dados

Tipos primitivos

Tipo	Tamanho	Valores Possíveis	Tipo	Tamanho	Valores Possíveis
bool	1 byte	true e false	long	8 bytes	-9223372036854775808 a 9223372036854775807
byte	1 byte	0 a 255	ulong	8 bytes	0 a 18446744073709551615
sbyte	1 byte	-128 a 127	float	4 bytes	Números até 10 elevado a 38
short	2 bytes	-32768 a 32767	double	8 bytes	Números até 10 elevado a 308
ushort	2 bytes	0 a 65535	decimal	16 bytes	Números com até 28 casas decimais
int	4 bytes	-2147483648 a 2147483647	char	2 bytes	Caracteres delimitados por aspas simples
uint	4 bytes	0 a 4294967295			

Tipos primitivos

- Inteiros

```
int tamanho = 30;
uint numero = 300;
long valor = 0x100000000; //  4294967296
ulong valor = 123456789;
byte num = 2;
sbyte i = -2;
short j = 20;
ushort j = 0xFE0A; //  65034
```

Tipos primitivos

- Reais

```
double largura = 25.5;
float altura = 1.70;
decimal passagem = 3.40m;
```

Tipos primitivos

- Boleanos

```
bool verdadeiro = true;
bool @false = false;
```

- Caractere

```
char letra = ‘a’;
```

Tipo enumerado

- enum é usada para declarar uma enumeração
- Por padrão, o primeiro enum tem o valor 0

```
using System;
public class Program
{
   enum WeekDays
{
  Monday = 10, Tuesday, Wednesday, Thursday, Friday,
  Saturday, Sunday
}
   public static void Main()
   {
      Console.WriteLine((int)WeekDays.Monday); //10
      Console.WriteLine(WeekDays.Monday); //Monday
      Console.WriteLine((int)WeekDays.Friday); //14
      Console.WriteLine(WeekDays.Friday); //Friday
   }
}
```

Tipo anônimo

```
using System;

public class Program{
  public static void Main()
  {
      var employee = new { Nome = "Bruno", Last = "Gama" };
      Console.WriteLine(employee.Nome); // Bruno
      Console.WriteLine(employee.Last); // Gama
  }
}
```

- Encapsular um conjunto de propriedades somente leitura em um único objeto sem precisar primeiro definir explicitamente um tipo

Tipo struct

- Tipo adequado para representar objetos leves como Ponto, Coordenada, etc...

```
using System;

public partial class Program{

    public struct Coordenada
    {
        public int x, y;

        public Coordenada(int p1, int p2)
        {
            x = p1;
            y = p2;
        }
    }
}
```

```
public partial class Program
{
    public static void Main()
    {
        Coordenada coord = new
        Coordenada(10,20);
        Console.WriteLine(coord.x); // 10
        Console.WriteLine(coord.y); // 20
    }
}
```

Tipo interface

```
interface ISample
{
   void SampleMethod();
}

class Implementation : ISample
{
   void ISample.SampleMethod(){
      // Method implementation
   }
}
```

- Uma interface contém apenas as assinaturas de métodos, propriedades, eventos
- Um classe ou struct que implementa a interface deve implementar os membros da interface

Tipo delegate

```
using System;
  public delegate void SimplesDelegate();   // Declaração

  public class ExemploDeDelegate
  {
     public static void minhaFuncao()
     {
        Console.WriteLine("Eu fui chamada por um delegate ...");
     }
     public static void Main()
     {
        SimplesDelegate simplesDelegate = new SimplesDelegate(minhaFuncao);
        // Instanciação
        simplesDelegate();// Invocação
     }
  }
```

- O Delegate é um tipo de variável que guarda o endereço de um método
- A principal utilidade dele é centrada no uso dos eventos

Método

```
static int AreaQuadrado(int i)
{
    int lado = i;
    return lado * lado;
}
```

- Um método é um bloco de código que contém uma série de instruções
- Um programa faz com que as instruções sejam executadas chamando o método e especificando os argumento de método necessários.

Classes e Objetos

```
public class Pessoa
{
   public string Nome { get; set; }
   public int Idade { get; set; }

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

Pessoa pessoa1 = new Pessoa("Leonardo", 6);
```

- Uma classe é uma estrutura de dados que combina ações e estado em uma única unidade
- Uma classe fornece uma definição para instâncias da classe criadas dinamicamente chamadas de objetos.

Namespace

```
namespace MyNamespace
{
    class Classe
    {
        public void Metodo()
        {
            System.Console.WriteLine("Exemplo");
        }
    }
}
```

- A palavra-chave namespace é usada para declarar um escopo que contém um conjunto de objetos relacionados.
- Utilizado para organizar classes.

Valores e Tipos de Dados

- Um **valor** é uma entidade que existe durante uma computação
- Um tipo de **dado** é um conjunto cujos valores exibem comportamento uniforme nas operações associadas

Tipagem estática e dinâmica

- Estática:
  - C# usa tipagem estática
  - O tipo de todas as variáveis é fixado em suas definições (em tempo de compilação)
- Dinâmica:
  - O tipo de uma variável pode mudar em tempo de execução de acordo com o valor atribuído a ela.
  - O C# 4 apresenta um novo tipo, dynamic. Sendo possível usar tipagem dinâmica.

Propriedades de Variáveis

Endereço

- Por padrão a linguagem o endereço dela não é acessível.
- Unsafe
  - Modificador que permite criar um contexto onde é possível fazer aritmética de ponteiros.

Tempo de vida

- Por padrão todas a variáveis e parâmetros de função tem tempo de vida local.
- Para definir um tempo de vida global usa-se a declaração static.
  - Todas as funções têm tempo de vida estático.

Escopo de visibilidade

- Escopo de arquivo
- Escopo de função
- Escopo de bloco
- Escopo de protótipo de função

CLR(Common Language Runtime)

- O coletor de lixo que aloca e libera memória para o usuário no heap gerenciado.
- Método GC.Collect pode ser chamar o coletor de lixo.

Memória secundária

- **Northwnd**
  - **LINQ to SQL**
- **StreamReader**
  - **StreamReader sr = new StreamReader("Texto.txt");**

Serialização

Serialização

- Serialização é o processo de conversão de um objeto em um fluxo de bytes para armazenar o objeto ou fluxo na memória, em um banco de dados, ou em um arquivo.
- Sua finalidade principal é salvar o estado de um objeto para ser capaz de recriá-lo quando necessário.
- O processo inverso é chamado desserialização.

Json(Serialização)

```
using System.Text;
using System.IO;
using System.Runtime.Serialization.Json;

namespace Converte_Object_Json
{
    public class JsonConversao
    {
        public string ConverteObjectParaJSon<T>(T obj)
        {
            DataContractJsonSerializer ser = new DataContractJsonSerializer(typeof(T));
            MemoryStream ms = new MemoryStream();
            ser.WriteObject(ms, obj);
            string jsonString = Encoding.UTF8.GetString(ms.ToArray());
            ms.Close();
            return jsonString;
        }
    }
}
```

Json(Desserialização)

```
using System.Text;
using System.IO;
using System.Runtime.Serialization.Json;

namespace Converte_Object_Json
{
    public class JsonConversao
    {
        public T ConverteJSonParaObject<T>(string jsonString)
        {
            DataContractJsonSerializer serializer = new DataContractJsonSerializer(typeof(T));
            MemoryStream ms = new MemoryStream(Encoding.UTF8.GetBytes(jsonString));
            T obj = (T)serializer.ReadObject(ms);
            return obj;
        }
    }
}
```

Expressões e comandos

Operadores

Primários

x.y	Acesso ao membro
x?.y	Acesso de membro condicional nulo
x?[y]	Acesso de índice condicional nulo
f(x)	Invocação de função
a[x]	Indexação de objeto de agregação
x++	Incremento de sufixo
x--	Decremento de sufixo
new	Instanciação de tipo

Operadores

Primários

typeof	Retorna o objeto Type que representa o operando
checked	Habilita a verificação de estouro para operações de inteiros
unchecked	Desabilita a verificação de estouro para operações de inteiros
default(T)	Produz o valor padrão do tipo (T)
delegate	Declara e retorna uma instância delegada
sizeof	Retorna o tamanho em bytes do operando do tipo
->	Desreferência de ponteiro combinada com acesso de membro

Operadores

Unários

+x	Retorna o valor de x
-x	Negação numérica
!x	Negação lógica
~x	Complemento bit a bit.
++x	Incremento de prefixo
--x	Decremento de prefixo
(T)x	Conversão de tipo
await	Aguarda um Task
&x	Endereço

Operadores

Aritméticos

x * y	Multiplicação
x / y	Divisão
x % y	Restante
x + y	Adição
x – y	Subtração

Shift

x << y	Bits de deslocamento para a esquerda e preenche com zero à direita
x >> y	Bits de deslocamento para a direita. Se o operando esquerdo for int ou long, então, os bits à esquerda serão preenchidos com o bit de sinal. Se o operando esquerdo for uint ou ulong, então, os bits à esquerda serão preenchidos com zero

Operadores

Teste de tipo e relacional

x < y	Menor que (verdadeiro se x for menor que y)
x > y	Maior que (verdadeiro se x for maior que y)
x <= y	Menor ou igual a
x > = y	Maior que ou igual a
is	Compatibilidade de tipo. Retornará true se o operando esquerdo avaliado puder ser convertido no tipo especificado no operando à direita (um tipo estático)
as	Conversão de tipo.Retorna o operando esquerdo convertido para o tipo especificado pelo operando à direita (um tipo estático), mas as retorna null em que (T)x lançaria uma exceção

Operadores

Lógicos

x == y	Igualdade
x != y	Não igual
x & y	AND lógico ou bit a bit (geralmente utilizado com tipos inteiros e tipos enum)
x ^ y	XOR lógico ou bit a bit
x | y	OR lógico ou bit a bit.
x && y	AND lógico
x || y	OR lógico

Operadores

Atribuição e operadores Lambda

x = y	Atribuição
x += y	Incremento
x =- y	Diminuir
x *= y	Atribuição de multiplicação
x /= y	Atribuição de divisão
x %= y	Atribuição restante
x &= y	Atribuição AND. AND o valor de y com o valor de x, armazene o resultado em x e retorne o novo valor

Operadores

Atribuição e operadores Lambda

x |= y	Atribuição OR
x ^= y	Atribuição XOR
x <<= y	Atribuição de deslocamento para a esquerda. Desloque o valor de x para a esquerda em y casas decimais, armazene o resultado em x e retorne o novo valor
x >>= y	Atribuição de deslocamento para a direita. Desloque o valor de x para a direita em y casas decimais, armazene o resultado em x e retorne o novo valor
=>	Declaração de lambda

Operadores

Outros

x ?? y	Retornará x se for não null; caso contrário, retornará y
t ? x : y	Se o teste t for avaliado como verdadeiro, então ele avaliará e retornará x; caso contrário, avaliará e retornará y

Estruturas de Controle

If

A partir de uma condição, executa um bloco de código.

```
if(Condição)
{
   /* bloco de código */
}
```

Switch

Dado uma variável, verifica com condições, e executa blocos de código.

```
switch (Variável)
{
   case Argumento 1:
       /*bloco de código */
       break;
   case Argumento 2:
       /*bloco de código */
       break;
}
```

Estruturas de Controle

While

Enquanto uma condição for verdadeira, executa o mesmo bloco de código.

```
while(Condição)
{
    /* bloco de código */
}
```

Do While

Similar ao while, mas garante que o bloco de código dentro do loop seja executado pelo menos uma vez. Verificação, diferentemente do While, é feita no final do bloco.

```
do{
   /* bloco de código */
}while(Condição);
```

Estruturas de Controle

For

Executa um bloco de código enquanto uma expressão booliana especificada é avaliada como true.

```
for (int i = 0; i < 5; i++)
{
  Console.WriteLine(i);
}
```

For Each

Dado uma array, executa um bloco de código para cada item da array.

```
foreach (int element in Array)
{
  /*bloco de código*/
}
```

Estruturas de Controle

Break

Termina o loop delimitador mais próximo ou a instrução switch na qual está presente.

```
for (int i = 0; i < 5; i++)
{
 if(Condição)
   break;
}
```

Continue

Passa o controle para a próxima iteração do laço.

```
for (int i = 0; i < 5; i++)
{
 if(Condição)
   continue;
}
```

Estruturas de Controle

Goto

Transfere o controle do programa diretamente para uma instrução rotulada.

```
switch (n)
{
  case 1:
      cost += 25;
      break;
  case 2:
      cost += 25;
      goto case 1;
}
```

Return

Finaliza o controle do método em que aparece e retorna o controle para o método de chamada.

```
for (int i = 0; i < 5; i++)
{
 if(Condição)
   Return i;
}
```

Exceções

Exceções

- Um erro em tempo de execução em um programa, que viola uma condição que não foi especificada para acontecer durante a operação normal.
- Um exemplo é quando um programa tenta fazer a divisão por 0
- Quando esse erro acontece o sistema pega o erro e lança uma exceção
- Se o programa não provê código para tratar uma exceção, o sistema irá parar o programa

Exceções

- Um jeito de verificar um possível erro é com if
- Mas, qual é o problema de se verificar possíveis erros com if?
- Exceções deixam seu programa mais tolerantes a falhas
- Também deixam seu código mais robusto e claro

```
public class ContaPoupanca : Conta
{
  public override bool Saca(double valor)
  {
     if (valor + 0.10 <= this.Saldo)
     {
        this.Saldo -= valor + 0.10;
        return true;
     }
     else
     {
        return false;
     }
  }
  // Resto do código da classe
}
```

Lançando uma Exceção

- Você pode lançar uma exceção usando a declaração throw
- Throw é usada para sinalizar a ocorrência de situação anormal
- A exceção gerada é um objeto cuja classe é derivada de System.Exception

```
private void Teste(string valor)
{
  if (string.IsNullOrEmpty(value))
  {
  throw new ArgumentNullException("valor", "parâmetro não
pode ser null");
  }
// ...
}
```

Capturando uma Exceção

- Geralmente a instrução throw é usada com um bloco try/catch/finally
- Após o bloco try há o bloco catch que captura a FileNotFoundException

```
try
{
  FazAlgumaCoisa(null);
}
catch (ArgumentNullException ex)
{
  Console.WriteLine("Exception: " + ex.Message);
}
```

Capturando mais de uma Exceção

- É possível usar mais de uma cláusula catch específica na mesma instrução try-catch
- Nesse caso, a ordem das cláusulas catch é importante

```
try
{
  throw new ArgumentNullException();
}
catch (ArgumentNullException ex)
{
  //será alcançada aqui
}
catch (ArgumentException ex)
{
  //captura qualquer outra
ArgumentException ou filha
}
```

Relançando uma Exceção

- Se você precisar lidar com uma exceção em um nível e em seguida, repassar a exceção para um nível superior de código para ser tratada, você deve relançar a exceção

```
StreamWriter stream = null;
try
{
  stream = File.CreateText("temp.txt");
  stream.Write(null, -1, 1);
}
catch (ArgumentNullException ex)
{
  Console.WriteLine("No catch: ");
  Console.WriteLine(ex.Message);
}
finally
{
  Console.WriteLine("No finally: Fechando o
arquivo");
  if (stream != null)
  {
    stream.Close();
  }
}
```

Finally

- O bloco finally garante que o código definido no bloco sempre será executado mesmo ocorrendo uma exceção
- Finally é útil para limpar todos recursos que foram alocados em um bloco try

```
try
{
  DoSomething();
}
catch (ArgumentNullException ex)
{
  LogException(ex);
  throw;
}
```

Concorrência

Processos

- Programas de computador são uma coleção passiva de instruções.
- Enquanto que um processo é a execução real dessas instruções.
- Vários processos podem estar associados ao mesmo programa.
- C# possui suporte para Programação Assíncrona, Programação Paralela e Threading.

Processamento paralelo

- Programação Assíncrona

Existem mecanismos para programação assíncrona fornecida pelo .NET.

- Programação paralela

Em um modelo de programação baseado em tarefa que simplifica o desenvolvimento paralelo, permitindo escrever código paralelo refinado, eficiente e escalável em uma linguagem natural sem a necessidade de trabalhar diretamente com threads ou o pool de threads.

- Threading

Mecanismos básicos de sincronização e simultaneidade fornecidos pelo .NET.

Threads

- O namespace System.Threading, contém classes para uso e sincronização de threads.
- Para criar uma thread é utilizada a classe Thread.
- Para terminar a execução de um thread, use o método Thread.Abort.

Threads e exceções

- As exceções sem tratamento nos threads, até threads em segundo plano, geralmente encerram o processo. Há três exceções a essa regra:
- Uma ThreadAbortException é gerada em um thread porque Abort foi chamado.
- Um AppDomainUnloadedException é gerado em um thread, pois o domínio do aplicativo no qual o thread está em execução está sendo descarregado.
- O Common Language Runtime(CLR) ou um processo de host encerra o thread lançando uma exceção interna.

Monitor Class

- Fornece um mecanismo que sincroniza o acesso a objetos.

using System.Threading;

Adquire um bloqueio exclusivo em um objeto especificado.

Enter(Object)

Libera um bloqueio exclusivo no objeto especificado

Exit(Object)

Semaphore Class

- Limita o número de threads que podem acessar um recurso ou um pool de recursos simultaneamente.

using System.Threading;

Inicializa uma nova instância da classe Semaphore, especificando o número inicial de entradas e o número máximo de entradas simultâneas.

Semaphore(Int32, Int32)

Bloqueia o thread atual até que o WaitHandle atual receba um sinal.

WaitOne()

Sai do sinal e retorna à contagem anterior.

Release()

Threads

```
using System;
using System.Threading;

namespace Program
{
  public partial class Program
  {
    static void Main(string[] args)
    {
      // Criando uma thread e a fazendo rodar FazerAlgo()
      Thread t = new Thread(new ThreadStart(FazerAlgo));
      t.Start(); // Executando a thread

      Thread.Sleep(2000);
      Console.WriteLine("
        Thread Main dormiu por 2 segundos");
      Console.ReadKey();
    }
  }
}
```

```
using System;
using System.Threading;

namespace Program
{
  public partial class Program
  {
    public static void FazerAlgo()
    {
      Thread.Sleep(5000);
      Console.WriteLine("Tarefa Terminada.");
    }
  }
}
```

Comparaçõs entre linguagens

Comparações

CRITÉRIOS GERAIS	C	C++	C#
Aplicabilidade	Sim	Sim	Sim
Confiabilidade	Não	Não	Parcial
Aprendizado	Não	Não	Não
Eficiência	Sim	Sim	Sim
Portabilidade	Não	Não	Sim
Método do Projeto	Estruturado	Estruturado e OO	Multiparadigma
Evolutibilidade	Não	Parcial	Sim
Reusabilidade	Parcial	Sim	Sim
Integração	Sim	Sim	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Comparações

CRITÉRIOS GERAIS	C	C++	C#
Aplicabilidade	Sim	Sim	Sim
Confiabilidade	Não	Não	Parcial
Aprendizado	Não	Não	Não
Eficiência	Sim	Sim	Sim
Portabilidade	Não	Não	Sim
Método do Projeto	Estruturado	Estruturado e OO	Multiparadigma
Evolutibilidade	Não	Parcial	Sim
Reusabilidade	Parcial	Sim	Sim
Integração	Sim	Sim	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Linguagem de propositos diversos

Comparações

CRITÉRIOS GERAIS	C	C++	C#
Aplicabilidade	Sim	Sim	Sim
Confiabilidade	Não	Não	Parcial
Aprendizado	Não	Não	Não
Eficiência	Sim	Sim	Sim
Portabilidade	Não	Não	Sim
Método do Projeto	Estruturado	Estruturado e OO	Multiparadigma
Evolutibilidade	Não	Parcial	Sim
Reusabilidade	Parcial	Sim	Sim
Integração	Sim	Sim	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Pois permite que o programador gerencie a memória utilizada

Comparações

CRITÉRIOS GERAIS	C	C++	C#
Aplicabilidade	Sim	Sim	Sim
Confiabilidade	Não	Não	Parcial
Aprendizado	Não	Não	Não
Eficiência	Sim	Sim	Sim
Portabilidade	Não	Não	Sim
Método do Projeto	Estruturado	Estruturado e OO	Multiparadigma
Evolutibilidade	Não	Parcial	Sim
Reusabilidade	Parcial	Sim	Sim
Integração	Sim	Sim	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Por ser multiparadigma

Comparações

CRITÉRIOS GERAIS	C	C++	C#
Aplicabilidade	Sim	Sim	Sim
Confiabilidade	Não	Não	Parcial
Aprendizado	Não	Não	Não
Eficiência	Sim	Sim	Sim
Portabilidade	Não	Não	Sim
Método do Projeto	Estruturado	Estruturado e OO	Multiparadigma
Evolutibilidade	Não	Parcial	Sim
Reusabilidade	Parcial	Sim	Sim
Integração	Sim	Sim	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Palavra-chave dynamic tem potencial de deixar o programa eficiente

Comparações

CRITÉRIOS GERAIS	C	C++	C#
Aplicabilidade	Sim	Sim	Sim
Confiabilidade	Não	Não	Parcial
Aprendizado	Não	Não	Não
Eficiência	Sim	Sim	Sim
Portabilidade	Não	Não	Sim
Método do Projeto	Estruturado	Estruturado e OO	Multiparadigma
Evolutibilidade	Não	Parcial	Sim
Reusabilidade	Parcial	Sim	Sim
Integração	Sim	Sim	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Código fonte é compilado para CIL que é interpretado pela VM CLR

Comparações

CRITÉRIOS GERAIS	C	C++	C#
Aplicabilidade	Sim	Sim	Sim
Confiabilidade	Não	Não	Parcial
Aprendizado	Não	Não	Não
Eficiência	Sim	Sim	Sim
Portabilidade	Não	Não	Sim
Método do Projeto	Estruturado	Estruturado e OO	Multiparadigma
Evolutibilidade	Não	Parcial	Sim
Reusabilidade	Parcial	Sim	Sim
Integração	Sim	Sim	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Estruturada, Funcional, Imperativa e OO

Comparações

CRITÉRIOS GERAIS	C	C++	C#
Aplicabilidade	Sim	Sim	Sim
Confiabilidade	Não	Não	Parcial
Aprendizado	Não	Não	Não
Eficiência	Sim	Sim	Sim
Portabilidade	Não	Não	Sim
Método do Projeto	Estruturado	Estruturado e OO	Multiparadigma
Evolutibilidade	Não	Parcial	Sim
Reusabilidade	Parcial	Sim	Sim
Integração	Sim	Sim	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Pode ser utilizada como OO

Comparações

CRITÉRIOS GERAIS	C	C++	C#
Aplicabilidade	Sim	Sim	Sim
Confiabilidade	Não	Não	Parcial
Aprendizado	Não	Não	Não
Eficiência	Sim	Sim	Sim
Portabilidade	Não	Não	Sim
Método do Projeto	Estruturado	Estruturado e OO	Multiparadigma
Evolutibilidade	Não	Parcial	Sim
Reusabilidade	Parcial	Sim	Sim
Integração	Sim	Sim	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Possui Classes e Polimorfismo

Comparações

CRITÉRIOS GERAIS	C	C++	C#
Aplicabilidade	Sim	Sim	Sim
Confiabilidade	Não	Não	Parcial
Aprendizado	Não	Não	Não
Eficiência	Sim	Sim	Sim
Portabilidade	Não	Não	Sim
Método do Projeto	Estruturado	Estruturado e OO	Multiparadigma
Evolutibilidade	Não	Parcial	Sim
Reusabilidade	Parcial	Sim	Sim
Integração	Sim	Sim	Parcial
Custo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Similar a Java

Comparações

CRITÉRIOS GERAIS	C	C++	C#
Escopo	Sim	Sim	Sim
Expressões e comandos	Sim	Sim	Sim
Tipos primitivos e compostos	Sim	Sim	Sim
Gerenciamento de memória	Programador	Programador	Programador/Sistema
Persistência dos dados	Biblioteca de Funções	Biblioteca de classes e funções	Biblioteca de classes, Serialização, Entity Framework, Nhinbernate
Passagem de parâmetros	Lista variável e por valor	Lista variável, default, por valor e referência	Lista variável, valor ou referência

Requer definição explícita de entidades, associando-as a um escopo de visibilidade

Comparações

CRITÉRIOS GERAIS	C	C++	C#
Escopo	Sim	Sim	Sim
Expressões e comandos	Sim	Sim	Sim
Tipos primitivos e compostos	Sim	Sim	Sim
Gerenciamento de memória	Programador	Programador	Programador/Sistema
Persistência dos dados	Biblioteca de Funções	Biblioteca de classes e funções	Biblioteca de classes, Serialização, Entity Framework, Nhinbernate
Passagem de parâmetros	Lista variável e por valor	Lista variável, default, por valor e referência	Lista variável, valor ou referência

Ampla variedade de expressões e comandos

Comparações

CRITÉRIOS GERAIS	C	C++	C#
Escopo	Sim	Sim	Sim
Expressões e comandos	Sim	Sim	Sim
Tipos primitivos e compostos	Sim	Sim	Sim
Gerenciamento de memória	Programador	Programador	Programador/Sistema
Persistência dos dados	Biblioteca de Funções	Biblioteca de classes e funções	Biblioteca de classes, Serialização, Entity Framework, Nhinbernate
Passagem de parâmetros	Lista variável e por valor	Lista variável, default, por valor e referência	Lista variável, valor ou referência

Possui grande variedade de tipos compostos

Comparações

CRITÉRIOS GERAIS	C	C++	C#
Escopo	Sim	Sim	Sim
Expressões e comandos	Sim	Sim	Sim
Tipos primitivos e compostos	Sim	Sim	Sim
Gerenciamento de memória	Programador	Programador	Programador/Sistema
Persistência dos dados	Biblioteca de Funções	Biblioteca de classes e funções	Biblioteca de classes, Serialização, Entity Framework, Nhinbernate
Passagem de parâmetros	Lista variável e por valor	Lista variável, default, por valor e referência	Lista variável, valor ou referência

O programador escolhe se vai usar ou não o coletor de lixo.

Comparações

CRITÉRIOS GERAIS	C	C++	C#
Encapsulamento e proteção	Parcial	Sim	Sim
Sistema de tipos	Não	Parcial	Sim
Verificação de tipos	Estática	Estática/Dinâmica	Estática/Dinâmica
Polimorfismo	Coerção e sobrecarga	Todos	Todos
Exceções	Não	Parcial	Sim
Concorrência	Não(Biblioteca de funções)	Não (Biblioteca de funções)	Sim

Mecanismos de visibilidade e classes.

Comparações

CRITÉRIOS GERAIS	C	C++	C#
Encapsulamento e proteção	Parcial	Sim	Sim
Sistema de tipos	Não	Parcial	Sim
Verificação de tipos	Estática	Estática/Dinâmica	Estática/Dinâmica
Polimorfismo	Coerção e sobrecarga	Todos	Todos
Exceções	Não	Parcial	Sim
Concorrência	Não(Biblioteca de funções)	Não (Biblioteca de funções)	Sim

Cada variável é de um tipo diferente e este tipo não pode mudar durante a execução do programa.

Comparações

CRITÉRIOS GERAIS	C	C++	C#
Encapsulamento e proteção	Parcial	Sim	Sim
Sistema de tipos	Não	Parcial	Sim
Verificação de tipos	Estática	Estática/Dinâmica	Estática/Dinâmica
Polimorfismo	Coerção e sobrecarga	Todos	Todos
Exceções	Não	Parcial	Sim
Concorrência	Não(Biblioteca de funções)	Não (Biblioteca de funções)	Sim

Na nova versão foi introduzido a **DLR - Dynamic Language Runtime** que é um ambiente de *runtime*.

Comparações

CRITÉRIOS GERAIS	C	C++	C#
Encapsulamento e proteção	Parcial	Sim	Sim
Sistema de tipos	Não	Parcial	Sim
Verificação de tipos	Estática	Estática/Dinâmica	Estática/Dinâmica
Polimorfismo	Coerção e sobrecarga	Todos	Todos
Exceções	Não	Parcial	Sim
Concorrência	Não(Biblioteca de funções)	Não (Biblioteca de funções)	Sim

Possui todos, mas em critérios de coerção não aceita estreitamento.

Comparações

CRITÉRIOS GERAIS	C	C++	C#
Encapsulamento e proteção	Parcial	Sim	Sim
Sistema de tipos	Não	Parcial	Sim
Verificação de tipos	Estática	Estática/Dinâmica	Estática/Dinâmica
Polimorfismo	Coerção e sobrecarga	Todos	Todos
Exceções	Não	Parcial	Sim
Concorrência	Não(Biblioteca de funções)	Não (Biblioteca de funções)	Sim

Possui um sistema de tratamento de exceções.

Comparações

CRITÉRIOS GERAIS	C	C++	C#
Encapsulamento e proteção	Parcial	Sim	Sim
Sistema de tipos	Não	Parcial	Sim
Verificação de tipos	Estática	Estática/Dinâmica	Estática/Dinâmica
Polimorfismo	Coerção e sobrecarga	Todos	Todos
Exceções	Não	Parcial	Sim
Concorrência	Não(Biblioteca de funções)	Não (Biblioteca de funções)	Sim

.Net Fornece uma série de métodos para tratamento de concorrência e multiprocessamento