C#

GRUPO:
Igor de Oliveira Nunes
Mateus Tassinari Ferreira
Renan Sarcinelli

PET EngComp

Surgimento do C#

No final de 1990 a Microsoft tinha diversas tecnologias e linguagens de programação. Toda vez que um programador precisava migrar para uma nova linguagem, era necessário aprender tanto a nova linguagem quanto suas bibliotecas e conceitos. Para solucionar esses problemas, a Microsoft recorreu à linguagem Java.

Porém, a linguagem Java possuía um grave problema: ela não se comunicava bem com o as bibliotecas de código nativo (código de máquina) que já existiam. Para resolver isso, a Microsoft decidiu criar a sua própria implementação do Java chamado J++.

PET EngComp

Surgimento do C#

Porém, o J++ era uma versão da linguagem Java que só podia ser executada no ambiente Microsoft, o que violava o licenciamento feito com a Sun e, por isso, a Microsoft foi processada.

Microsoft então cria uma nova plataforma chamada de ***.Net***. Diversas linguagens diferentes compartilhariam o mesmo conjunto de bibliotecas. Um novo projeto de linguagem de programação foi iniciado, o projeto COOL (C-like Object Oriented Language). Em 2002, o projeto COOL foi lançado como linguagem ***C#*** 1.0 junto com o ambiente ***.Net*** 1.0.

PET EngComp

Surgimento do C#

The Transition from C++++ to C#

1\.

C++++

2\.

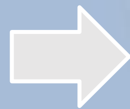
3\.

C#

PET EngComp

Microsoft .NET

Iniciativa da empresa Microsoft, que visa uma plataforma única para desenvolvimento e execução de sistemas e aplicações.

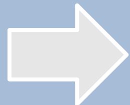


Com idéia semelhante à plataforma Java, o programador deixa de escrever código para um sistema específico, e passa a escrever para a plataforma .NET.

PET EngComp

Microsoft .NET

A plataforma .NET é executada sobre uma - CLR(Ambiente de Execução Independente de Linguagem) interagindo com um Conjunto de Bibliotecas Unificadas (framework).



Capaz de executar mais de 33 diferentes linguagens de programação, interagindo entre si como se fossem uma única linguagem.

Exemplos:
> C#
> C++
> Fortran
> Haskell
> Lua
> Python

Visual Studio

O **Visual Studio** é a plataforma da Microsoft(IDE) destinada a desenvolvedores que trabalham com a linguagem de programação C# e com o framework .NET. Sua principal função é auxiliar programadores na criação de aplicações para o Windows.

PET EngComp

Baixando Visual Studio/Microsoft .NET

Microsoft .NET:

http://msdn.microsoft.com/pt-br/vstudio/aa496123

Visual Studio:

http://msdn.microsoft.com/pt-BR/vstudio

Versões mais atuais:

Microsoft .NET 4.5.2 e linguagem C# 5.0

Tutorias e informações sobre C#:

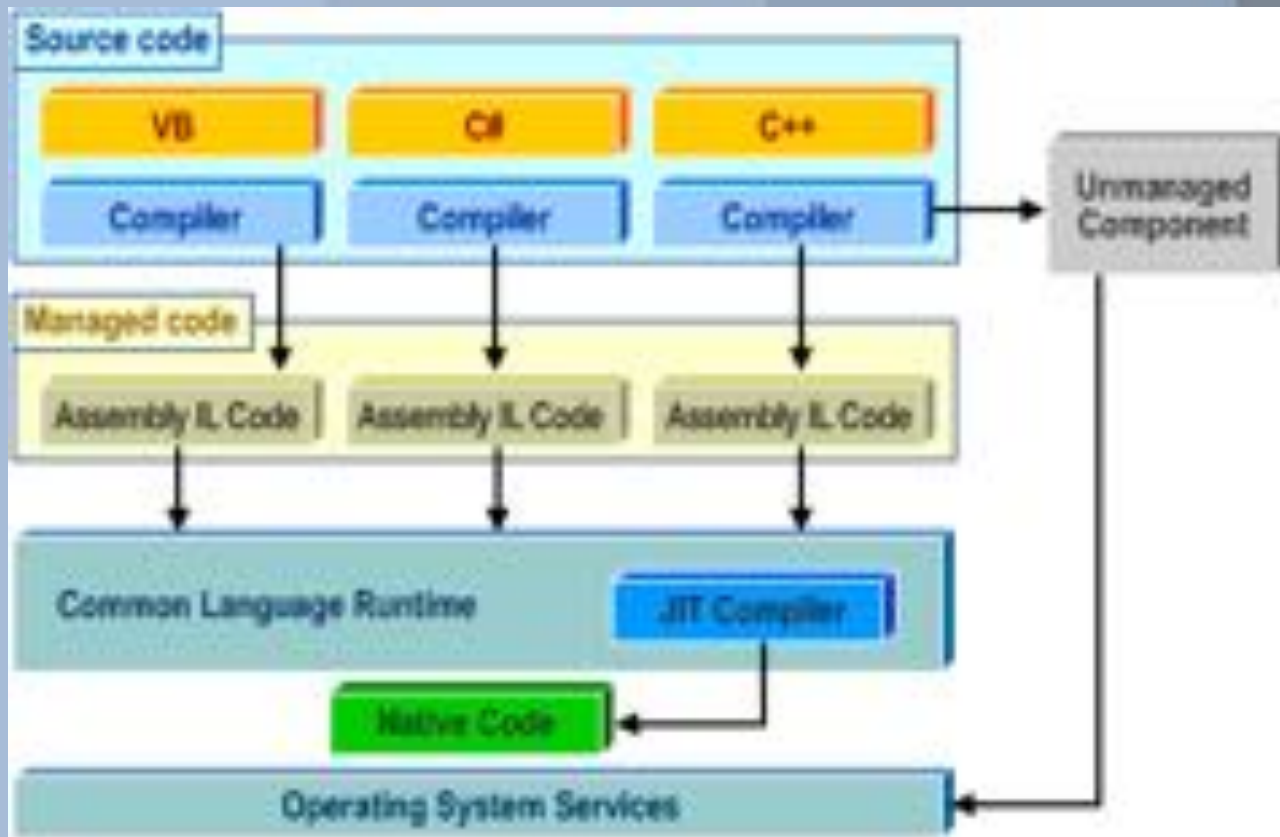
http://msdn.microsoft.com/pt-br/library/kx37x362.aspx (Português)

http://msdn.microsoft.com/en-us/library/67ef8sbd.aspx (Inglês)

PET EngComp

Compilador .NET

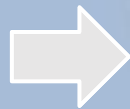
- Compilador Híbrido



PET EngComp

Compilando e Rodando programas.

- Windows
  - Visual Studio
  - csc



- Linux
  - gmcs
  - mono

Palavras chaves

- *Palavras chaves em C# são também 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	int	interface	internal
is	lock	long	namespace	new	null
object	operator	out	override	params	privated
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	

Amarrações

ConsoleApplication1 - Microsoft Visual C# 2010 Express

File Edit View Project Debug Data Tools Window Help

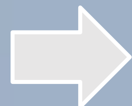
Program.cs*

ConsoleApplication1.Program | main(string[] args)

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


namespace ConsoleApplication1
{
    class Program
    {
        static void main(string[] args)
        {
            int virtual = 0;
            int num = 3;
            Console.WriteLine(Num);

        }
    }
}
```



C# é *case sensitive*

PET EngComp

Escopo

C# possui escopo estático

```
namespace N
{ // Escopo de Namespace
    class C
    { //Escopo de Classe
        void f(bool b)
        {
            //escopo do bloco mais externo
        }
    }
}
```

Tipos de dados

- Em C# existem duas grandes categorias de Tipos, os **tipos valores** e **tipos referência**, porém ambos podem assumir **tipos genéricos** o que leva a necessidade do **tipo parâmetro.**

- Existe ainda um terceiro grupo de tipos os **tipo ponteiros** mas estes só são permitidos em modo inseguro que será detalhado mais a frente.

- A principal diferença entre **tipos valores** e **tipos referência** são o conteúdo de cada um.**Tipos valores** guardam os dados da variável e **tipos referência** as referências.

- O sistema de tipos de C# é unificado, ou seja seja qual for o seu tipo todas as variáveis podem ser tratadas como objetos, pois, todos os tipos derivam direta ou indiretamente do tipo base **objeto**

PET EngComp

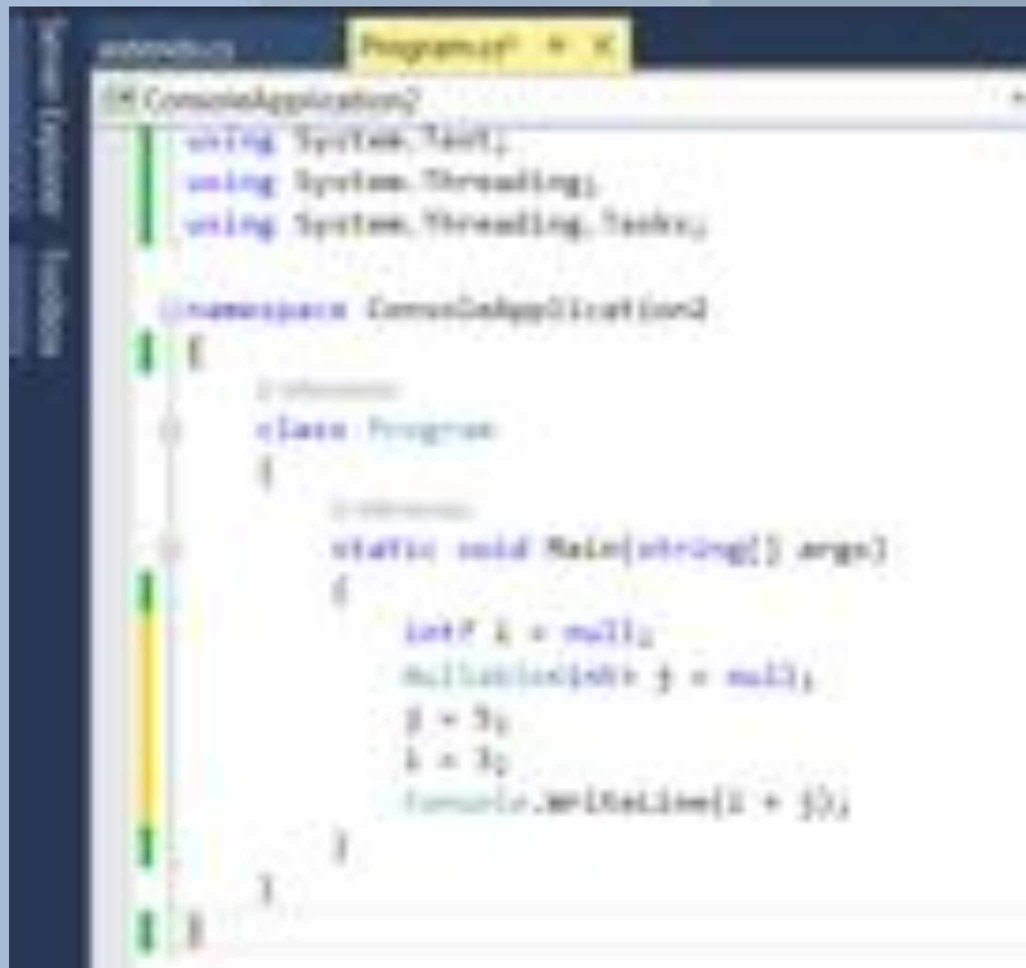
Tipos Valores

- value-type:
    - struct-type:
        - type-name
        - simple-type:
            - numeric-type:
                - integral-type:
                    - **sbyte**
                    - **byte**
                    - **short**
                    - **ushort**
                    - **int**
                    - **uint**
                    - **long**
                    - **ulong**
                    - **char**
                - floating-point-type:
                    - **float**
                    - **double**
                - **decimal**
            - **bool**
        - nullable-type:
            - non-nullable-value-type ?
    - enum-type:
        - type-name

PET EngComp

Nullable type

Permite que qualquer tipo non-nullable (não anulável) assuma o valor nulo (null):



PET EngComp

Struct type

```
using System.Threading;
using System.Threading.Tasks;

namespace ConsoleApplication2
{
    public struct Book
    {
        public decimal price;
        public string title;
        public string author;
    }
    class Program
    {
        static void Main(string[] args)
        {

            Book lCompletoTotal;
            lCompletoTotal.price = 100.50m;
            Console.WriteLine(lCompletoTotal.price);
            Console.ReadKey();
        }
```

Enum type

```
namespace ConsoleApplication2
{
    class Program
    {
        enum Days { Sunday, Monday, TuesDay, WedNesday, Thursday, Friday, Saturday };
        enum Months : byte { Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec };
        static void Main(string[] args)
        {
            Days Hoje = Days.Sunday;
            Console.WriteLine(Hoje);
            //Output: Sunday
            Console.WriteLine(Months.Jan);
            //Output: Jan
            Console.ReadKey();
        }
    }
}
```

PET EngComp

Tipos referência

- reference-type:
  - class-type:
    - type-name
    - **object**
    - **dynamic**
    - **string**
  - interface-type:
    - type-name
  - array-type:
    - non-array-type   rank-specifiers
      - rank-specifier
        - rank-specifiers   rank-specifier
          - [dim-separators$_{opt}$]
            - ,
            - dim-separators   ,
  - delegate-type:
    - type-name

PET EngComp

Class type

```
public class Pessoa
{
    public int idade { get; private set; }
    string nome;

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

    public bool IsIdoso()
    {
        return (idade >= 70);
    }

    public override string ToString()
    {
        return this.nome;
    }
}
```

```
class Program
{

    static void Main(string[] args)
    {
        Pessoa joao = new Pessoa(20, "Joao");
        Pessoa pedro = new Pessoa(71, "Pedro");

        Console.WriteLine(joao+ " é idoso? " + joao.IsIdoso() );
        //output: Joao é idoso? false
        Console.WriteLine(pedro +" é idoso? " + pedro.IsIdoso());
        //output: Pedro é idoso? true
    }
}
```

Class List<>

```
namespace A
{ // Escopo de Namespace
    class Y
    { //Escopo de Classe
        static void Main(string[] args)
        {
            List<int> l= new List<int>();
            l.Add(3);
            l.Add(5);
            l.Add(2);
            l.Sort();
            Console.WriteLine("Lista1:");
            foreach(int j in l)
                Console.WriteLine(j);
            l.RemoveAt(1);
            Console.WriteLine("Lista2:");
            foreach (int j in l)
                Console.WriteLine(j);
            Console.ReadKey();
        }
    }
}
```

String

```
namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            string lp = "lp";
            Console.WriteLine(lp.Length);
            //output: 2
            String LP = (String)lp;
            Console.WriteLine(LP);
            //output: lp
            LP = "Linguagem de Prog";
            lp = LP;
            string[] lps = lp.Split();
            Console.WriteLine(lps[0]+"/"+lps[2]);
            //output: Linguagem/Prog
        }
    }
}
```

PET EngComp

Object and Dynamic

- O tipo objeto é a base para todos os outros tipos. Como dito anteriormente todos os outros tipos herdam do tipo objeto

- Assim como o tipo objeto o tipo dynamic também referencia qualquer outro tipo. A diferença do tipo dynamic para o object é que suas amarrações são feitas em tempo de execução permitindo tipagem dinâmica entre outros artifícios.

```
static void Main(string[] args)
{
    dynamic d = 1;
    var testSum = d + 3;
    System.Console.WriteLine(testSum);
    //Output: 4
    d = true;
    System.Console.WriteLine(d);
    //Output: True
    Console.ReadKey();
}
```

Array type

```
namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            int[] objInt = new int[5];
            int[] inteiro = {2,4};
            int[,] inteiros = { { 1, 2 }, { 3, 4 } };
            int[,][] inteiroInteiros = { { inteiro }, { inteiro } };
            int[][,] inteirosInteiro = { inteiros, inteiros };

            Console.WriteLine(inteiro[1]);
            //Output: 4
            Console.WriteLine(inteiros[1,0]);
            //Output: 3
            Console.WriteLine(inteiroInteiros[0,0][0]);
            //Output: 2
            Console.WriteLine(inteirosInteiro[1][0,0]);
            //Output: 1
            Console.ReadKey();
        }
    }
```

PET EngComp

Delegate types

```
public delegate int comparador (int x,int y);
class Program
{
    static public void comparar(comparador comparador,int x,int y){
        Console.WriteLine(comparador(x,y));
    }

    static public int maior(int x, int y)
    {
        if (x > y)
            return x;
        else
            return y;
    }
```

```
    static public int menor(int x, int y)
    {
        if (x < y)
            return x;
        else
            return y;
    }

    static void Main(string[] args)
    {
        comparar(new comparador(maior), 5, 7);
        //output: 7
        comparar(new comparador(menor), 5, 7);
        //output: 5
        Console.ReadKey();
    }
}
```

Tipo ponteiros

int* p1, p2, p3;   // Ok em C#

int *p1, *p2, *p3;   // Não funciona em C#

```
class Program{
    static void Main(){
        unsafe{
            int i = 10;
            int* ponteiro;
            ponteiro = &i;
            Console.WriteLine(*ponteiro);
            Console.Read();
        }
    }
```

PET EngComp

Stack e Heap

O gerenciamento da memória é feito em duas áreas de memória, o Stack e o Heap.

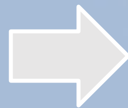
O Stack é uma área bem pequena de memória (alguns KB) e funciona no formato de pilha.

Os dados armazenados na Stack são os “Value Types” (Tipos valores).

PET EngComp

Stack e Heap

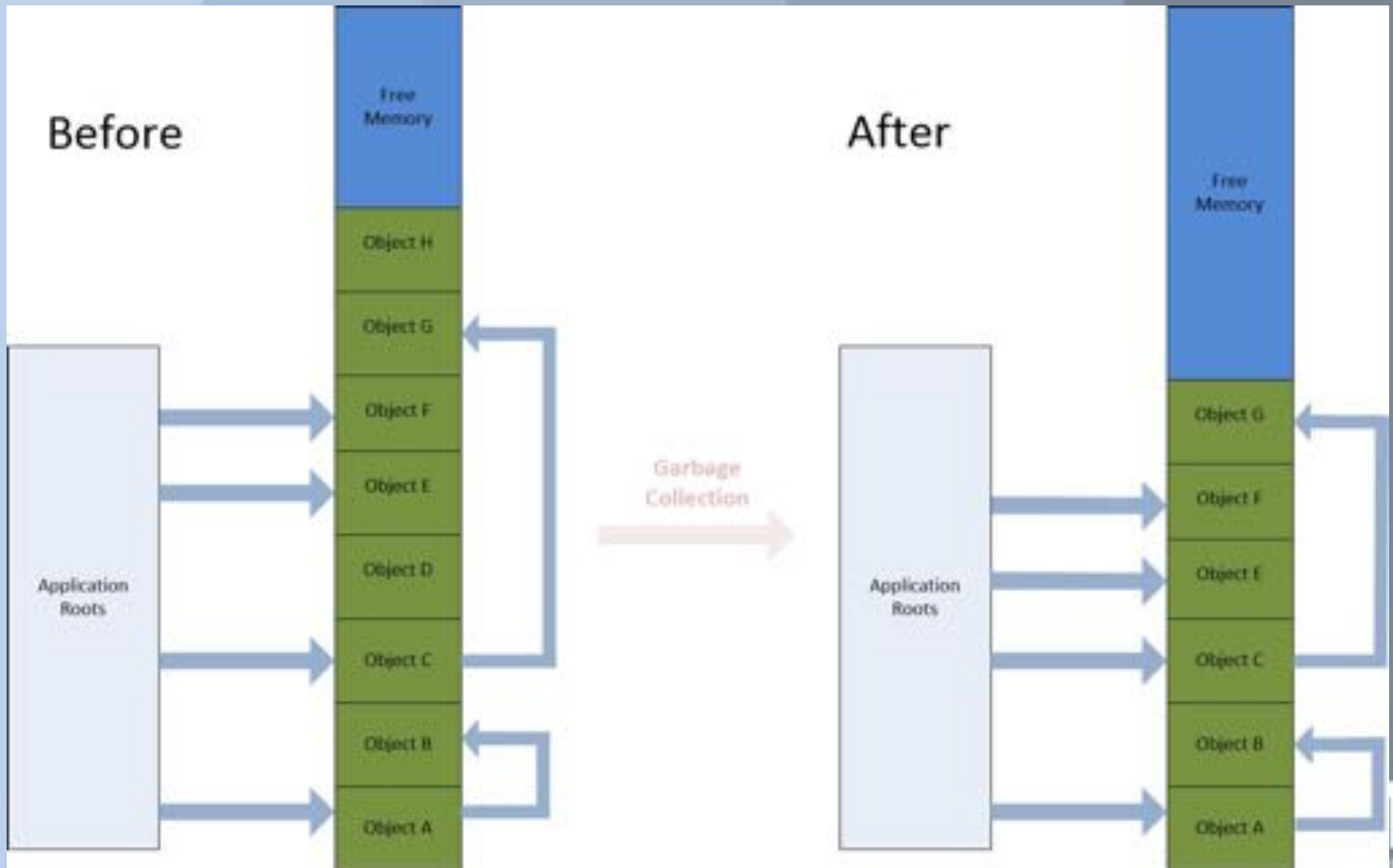
Os dados armazenados na Heap são os “Reference Types” (Tipos referência).



Garbage Collector é responsável pela limpeza desta área de memória.

PET EngComp

Garbage Collector



Leitura e escrita em arquivos

O C# utiliza o Stream para ler e escrever em arquivos. Sempre que se pretender ler ou escrever dados (bytes) num arquivo, deve ser utilizado ou criado um objeto Stream.

A classe Stream está disponível na biblioteca IO, por isso devemos inclui-la no arquivo da classe.

Using System.IO;

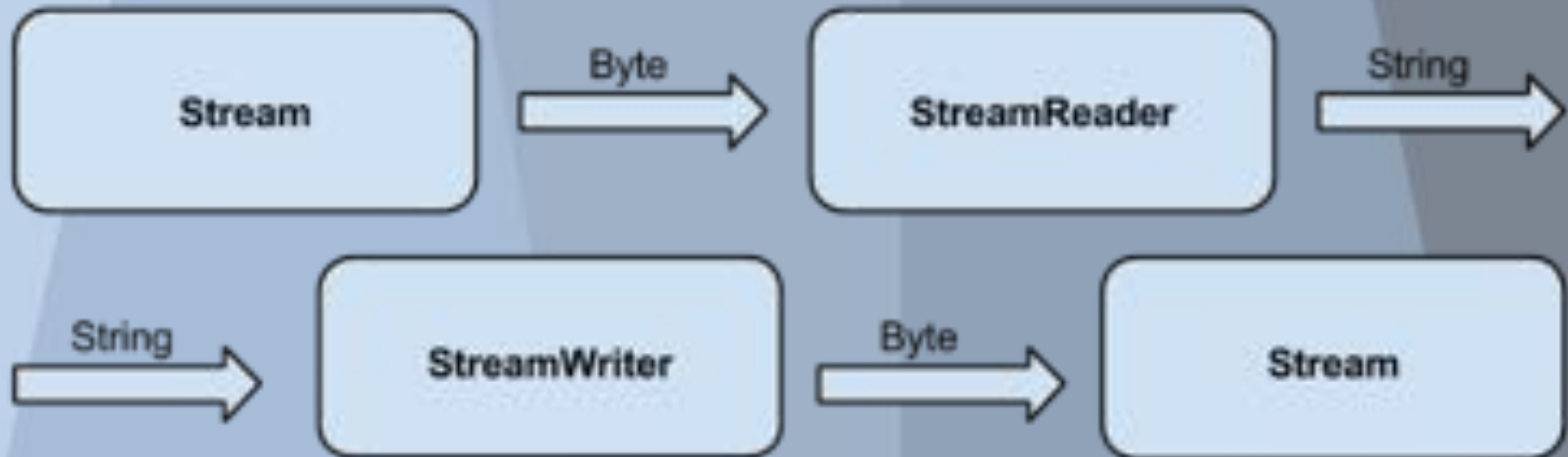
PET EngComp

Leitura e escrita em arquivos

```
namespace B
{ // Escopo do Namespace
    class [illegible]
    { //Escopo da Classe
        static void Main(string[] args)
        {
            Stream saida = File.Open("arquivosaida", FileMode.Create);
            StreamWriter escritor = new StreamWriter(saida);
            escritor.WriteLine("minha mensagem");
            escritor.Close();
            saida.Close();
            Stream entrada = File.Open("arquivosaida", FileMode.Open);
            StreamReader leitor = new StreamReader(entrada);
            while (!leitor.EndOfStream)
            {
                string linha = leitor.ReadLine();
                Console.WriteLine(linha);
            }
            leitor.Close();
            entrada.Close();
        }
    }
}
```

PET EngComp

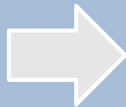
Leitura e escrita em arquivos



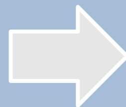
PET EngComp

Persistência de Dados

- *Serialização:*



Muitas aplicações necessitam de armazenar ou transferir objetos. O .NET inclui muitas técnicas de serialização.



As principais são converter objetos em binário ou documentos XML.

PET EngComp

Persistência de Dados

- *Serialização binária:*

```
namespace N
{ // Escopo do Namespace
    class C
    { //Escopo da Classe
        static void Main(string[] args)
        {
            string s = "teste";
            FileStream saida = new FileStream ("arquivoserializado", FileMode.Create);
            BinaryFormatter b = new BinaryFormatter();
            b.Serialize(saida,s);
            saida.Close();
            FileStream entrada = new FileStream ("arquivoDeserializado", FileMode.Open);
            BinaryFormatter b1 = new BinaryFormatter();
            string s1 = " ";
            s1 = (string)b1.Deserialize(entrada);
            entrada.Close();
            Console.WriteLine(s);

        }
    }
}
```

Persistência de Dados

- *Serialização XML:*

```
public class Cliente
{
    public string Nome { get; set; }
    public string Email { get; set; }
    public void Exportar(string caminho)
    {
        FileStream stream = new FileStream(caminho, FileMode.Create);
        XmlSerializer serializador = new XmlSerializer(typeof(Cliente));
        serializador.Serialize(stream, this);
    }
    public static Cliente Importar(string caminho)
    {
        FileStream stream = new FileStream(caminho, FileMode.Open);
        XmlSerializer desserializador = new XmlSerializer(typeof(Cliente));
        Cliente cliente = (Cliente)desserializador.Deserialize(stream);
        return cliente;
    }
```

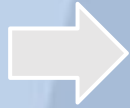
Valores Default

```
namespace N
{ // Escopo do Namespace
    class C
    { //Escopo da Classe
        static int soma5(int num1 , int num2, int num3, int num4=0, int num5=0)
        {
            return num1 + num2 + num3 + num4 + num5;
        }
        static void Main(string[] args)
        {
            Console.WriteLine(soma5(1, 2, 3));
            //output: 6
            Console.WriteLine(soma5(1, 2, 3, 10));
            //output: 16
            Console.WriteLine(soma5(1, 2, 3, 20 , 50));
            //output: 76
            Console.ReadKey();
        }
    }
}
```

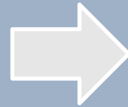
Variable Arguments

```
namespace N
{ // Escopo do Namespace
    class C
    { //Escopo da Classe
        static int somaN(params int[] varargs)
        {
            int soma = 0;
            foreach (dynamic i in varargs)
                soma += i;
            return soma;
        }
        static void Main(string[] args)
        {
            Console.WriteLine(somaN(1, 2 , 3));
            //output: 6
            Console.WriteLine(somaN(1, 2, 3, 10));
            //output: 16
            Console.WriteLine(somaN(1, 2, 3, 20 , 50));
            //output: 76
            Console.ReadKey();
        }
    }
}
```

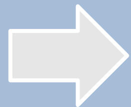
Passagem de Parâmetros



A passagem de parâmetros de C# é posicional.



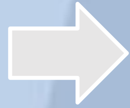
Pode ser por cópia ou referência (através da palavra *ref*).



O momento de passagem é normal.

PET EngComp

Namespace

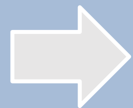


São delimitados através do uso do operador . (ponto).

using evita a necessidade de especificar o nome do *namespace* para cada classe.
Exemplo: System.Console.WriteLine("Hello World!");

ou

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

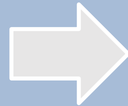


O namespace global é o namespace "raíz": global::System sempre irá se referir ao namespace System do .NET Framework.

PET EngComp

Namespace

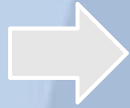
- *Comparação com Package(Java):*



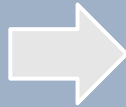
- Um pacote representa uma pasta.
- Namespace não está relacionado a pasta.
- Bibliotecas são importadas com import, e situa-se depois da declaração do pacote.
- As classes são importadas através do using, e se encontram antes da declaração do namespace, pois em C# a classe é delimitada pelo namespace a que pertence.

PET EngComp

Verificação de Tipos



C# faz verificação de tipos mista.



Maior parte das verificações de tipos em tempo de compilação.

Outras em tempo de execução.

PET EngComp

Verificação de tipos

C# possui tipagem estática e dinâmica (através do tipo dynamic).

C# é quase fortemente tipada

Ex.:
```
double soma(double a, int b) {
        return a + b;
    }
```

Os tipos de dados que função suporta estão bem definidos(double e int) e o tipo de dado que a função devolve também(double).

PET EngComp

Polimorfismo

Conversões implícitas:

- Identity conversions
- Implicit numeric conversions
- Implicit enumeration conversions.
- Implicit nullable conversions
- Null literal conversions
- Implicit reference conversions
- Boxing conversions
- Implicit dynamic conversions
- Implicit constant expression conversions
- User-defined implicit conversions
- Anonymous function conversions
- Method group conversions

PET EngComp

Polimorfismo

Conversões explícitas:

- All implicit conversions.
- Explicit numeric conversions.
- Explicit enumeration conversions.
- Explicit nullable conversions.
- Explicit reference conversions.
- Explicit interface conversions.
- Unboxing conversions.
- Explicit dynamic conversions
- User-defined explicit conversions.

PET EngComp

Polimorfismo

```
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            int i = 2;
            bool b = true;
            i + b;
        }
    }
}
```

```
using System.Text;
using System.Threading;
using System.Threading.Tasks;

namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            int i = 2;
            char b = 'b';
            i = i + b;
            Console.WriteLine(i);
            Console.ReadKey();
        }
```

Polimorfismo

```
namespace N
{ // Escopo de Namespace
    class C
    { //Escopo de Classe
        enum dia { Seg, Ter, Qua };
        static void Main(string[] args){
            dia hoje = (dia)2;
            Console.WriteLine(hoje);
            //Output:
            Console.ReadKey();
        }
    }
}
```

```
namespace N
{ // Escopo de Namespace
    class C
    { //Escopo de Classe
        enum dia { Seg, Ter, Qua };
        static void Main(string[] args){
            int i = 5;
            float? j = 0;
            j = i;
            Console.WriteLine(j);
            //Output: 5
        }
    }
}
```

PET EngComp

Polimorfismo

```
class Palavra
{
    string palavra;
    public Palavra(string s){
        this.palavra = s;
    }
    static public implicit operator string(Palavra p){
        return p.palavra;
    }
    static public implicit operator Palavra(string s){
        return new Palavra(s);
    }
}
class [illegible]
{ //Escopo de Classe
    static void Main(string[] args){
        Palavra p = new Palavra("iP");
        string s = p;
        Console.WriteLine(s);
        //Output: iP
        Console.ReadKey();
    }
}
```

PET EngComp

Polimorfismo de Sobrecarga

```
class Cafeteira
{
    public int xicaras{get;private set;}
    string qualidade;
    public Cafeteira(string q)
    {
        this.xicaras = 0;
        this.qualidade = q;
    }
    static public int operator +(Cafeteira c, int n){
        c.xicaras += n;
        return c.xicaras;
    }
    static public bool operator ==(Cafeteira c1, Cafeteira c2)
    {
        return (c1.qualidade == c2.qualidade);
    }
```

PET EngComp

Polimorfismo de Sobrecarga

```
    static public bool operator ==(Cafeteira c1, Cafeteira c2)
    {
        return (c1.qualidade == c2.qualidade);
    }
    static public bool operator !=(Cafeteira c1, Cafeteira c2)
    {
        return (c1.qualidade != c2.qualidade);
    }
}
class C
{ //Escopo de Classe
    static void Main(string[] args){
        Cafeteira c1 = new Cafeteira("bom");
        Cafeteira c2 = new Cafeteira("ruim");
        int i = c1 + 4;
        Console.WriteLine(c1+3);
        Console.WriteLine(c1 ==c2);
        Console.ReadKey();
    }
}
```

Polimorfismo Paramétrico

```
namespace N
{ // Escopo de Namespace
    public class [illegible]<T>
    {
        private T elemento;
        public void colocar(T elem)
        {
            elemento = elem;
        }
        public T retirar()
        {
            return elemento;
        }
    }
    class [illegible]
    { //Escopo de Classe
        static void Main(string[] args){
            [illegible]<int> inteiro = new [illegible]<int>();
            inteiro.colocar(5);
        }
    }
}
```

Polimorfismo de Inclusão

C# Não Suporta Herança Múltipla

Sobrescrita de Métodos

- Na superclasse devemos ter o modificador de acesso virtual.
- Virtual indica que o método pode ser sobrescrito na classe derivada.
- O método da subclasse que irá sobrescrever, deverá ter o modificador de acesso override.

Classe Abstrata

```
namespace N
{ // Escopo de Namespace
    public abstract class C1
    {
        public abstract void imprime();
    }
    public class C2 : C1
    {
        public override void imprime()
        {
            Console.WriteLine("Classe2");
        }
    }
```

```
    class P
    { //Escopo de Classe
        static void Main(string[] args){

            C2 c2 = new C2();
            c2.imprime();
            //Output: Classe2
            Console.ReadKey();
        }
    }
}
```

PET EngComp

Exceções

- C# possui mecanismos de tratamento de excessões.
- Funcionam, basicamente, como os de C++.
- A classe *System.Exception* é o tipo base de todas as exceções e contém duas propriedades (especificadas na chamada do contrutor) que todas as exceções compartilham:
  - *Message*: contém a descrição do motivo da exceção;
  - *InnerException*: refere-se à exceção que gerou a exceção atual (o valor é *null* se não for causada por outra exceção).

PET EngComp

Exceções

- As exceções são lançadas de duas formas diferentes:

  - pela instrução *throw* (imediatamente e incondicionalmente).
  - por meio de condições excepcionais que ocorrem durante o processo de instruções ou expressões (e. g. *System.DivideByZeroException*).

PET EngComp

Exceções

- Bloco *try* inclui comandos e métodos que podem gerar uma situação de exceção.
- Quando a exceção ocorre, o sistema busca o *catch* mais próximo e prepara-se para executar as instruções deste bloco.
- Porém, antes de iniciar a execução do bloco *catch*, é executado qualquer bloco *finally* associado à chamadas *try* mais aninhadas do que aquela que causou a exceção.

PET EngComp

Exceções

```
try
{
    doSomething();
}
catch
{
    catchSomething();
}
finally
{
    alwaysDoThis();
}
```

PET EngComp

Exceções

System.ArithmeticException	A base class for exceptions that occur during arithmetic operations, such as System.DivideByZeroException and System.OverflowException.
System.ArrayTypeMismatchException	Thrown when a store into an array fails because the actual type of the stored element is incompatible with the actual type of the array.
System.DivideByZeroException	Thrown when an attempt to divide an integral value by zero occurs.
System.IndexOutOfRangeException	Thrown when an attempt to index an array via an index that is less than zero or outside the bounds of the array.
System.InvalidCastException	Thrown when an explicit conversion from a base type or interface to a derived type fails at run time.
System.NullReferenceException	Thrown when a null reference is used in a way that causes the referenced object to be required.
System.OutOfMemoryException	Thrown when an attempt to allocate memory (via new) fails.
System.OverflowException	Thrown when an arithmetic operation in a checked context overflows.
System.StackOverflowException	Thrown when the execution stack is exhausted by having too many pending method calls; typically indicative of very deep or unbounded recursion.
System.TypeInitializationException	Thrown when a static constructor throws an exception, and no catch clauses exists to catch it.

PET EngComp

Concorrência

- No *.NET* o *namespace* ***System.Threading*** fornece as classes e interfaces que permitem a programação concorrente (Semaphore, Mutex, Monitor, etc.).

- Para criar uma thread é utilizada a classe *Thread*.

PET EngComp

Concorrência

```
using System.Threading;

Thread thread = new Thread(new ThreadStart(WorkThreadFunction));
thread.Start();
```

```
public void WorkThreadFunction()
{
  try
  {
    // do any background work
  }
  catch (Exception ex)
  {
    // log errors
  }
}
```

PET EngComp

Avaliação da Linguagem

Critérios	C#	C++	Java
Aplicabilidade	Parcial	Sim	Parcial
Confiabilidade	Parcial	Não	Sim
Aprendizado	Não	Não	Não
Eficiência	Parcial	Sim	Parcial
Portabilidade	Sim	Não	Sim
Método de projeto	OO	Estruturado e OO	OO
Evolutibilidade	Parcial	Parcial	Sim
Reusabilidade	Sim	Sim	Sim
Integração	Sim	Sim	Parcial

PET EngComp

Avaliação da Linguagem

Critérios	C#	C++	Java
Encapsulamento e Proteção	Sim	Sim	Sim
Sistemas de Tipos	Parcial	Parcial	Sim
Verificação de Tipos	Estática / Dinâmica	Estática / Dinâmica	Estática / Dinâmica
Polimorfismo	Todos	Todos	Todos
Excessão	Parcial	Parcial	Sim
Concorrência	Sim	Não (biblioteca de funções)	Sim

PET EngComp

Avaliação da Linguagem

Critérios	C#	C++	Java
Escopo	Sim	Sim	Sim
Expressões e Comandos	Sim	Sim	Sim
Gerênciamento de Memória	Programador / Sistema	Programador	Sistema
Passagem de Parâmetros	Lista variável, default, por valor e por referência	Lista variável, default, por valor e por referência	Lista variável, por valor e por cópia de referência

PET EngComp

Referências:

- http://msdn.microsoft.com/pt-br/library/kx37x362.aspx (Português)

- http://msdn.microsoft.com/en-us/library/67ef8sbd.aspx (Inglês)

- ***C# Language Specification 5.0***, disponível em http://www.microsoft.com/en-us/download/details.aspx?id=7029

PET UFES
EngComp