

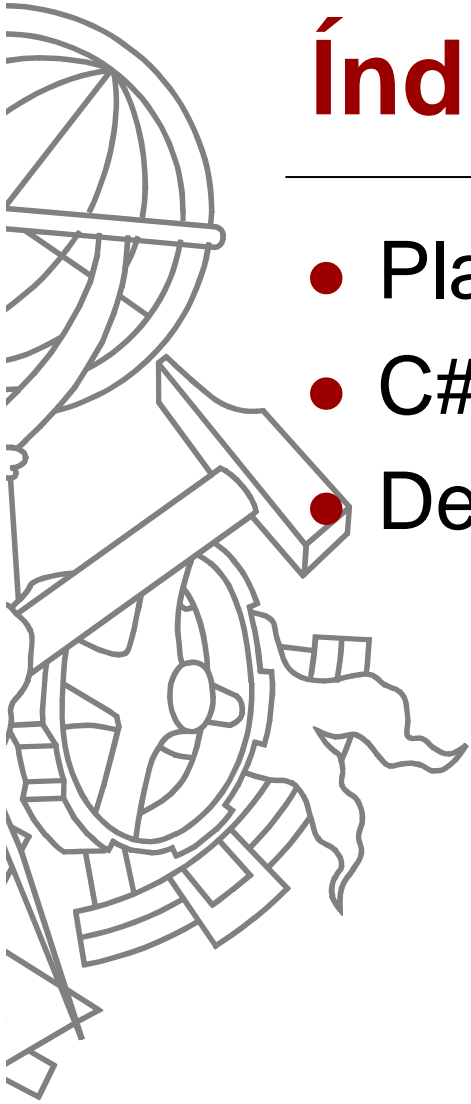


Introdução ao Desenvolvimento .NET

Paulo Sousa

Instituto Superior de Engenharia do Porto

Instituto Politécnico do Porto



Índice

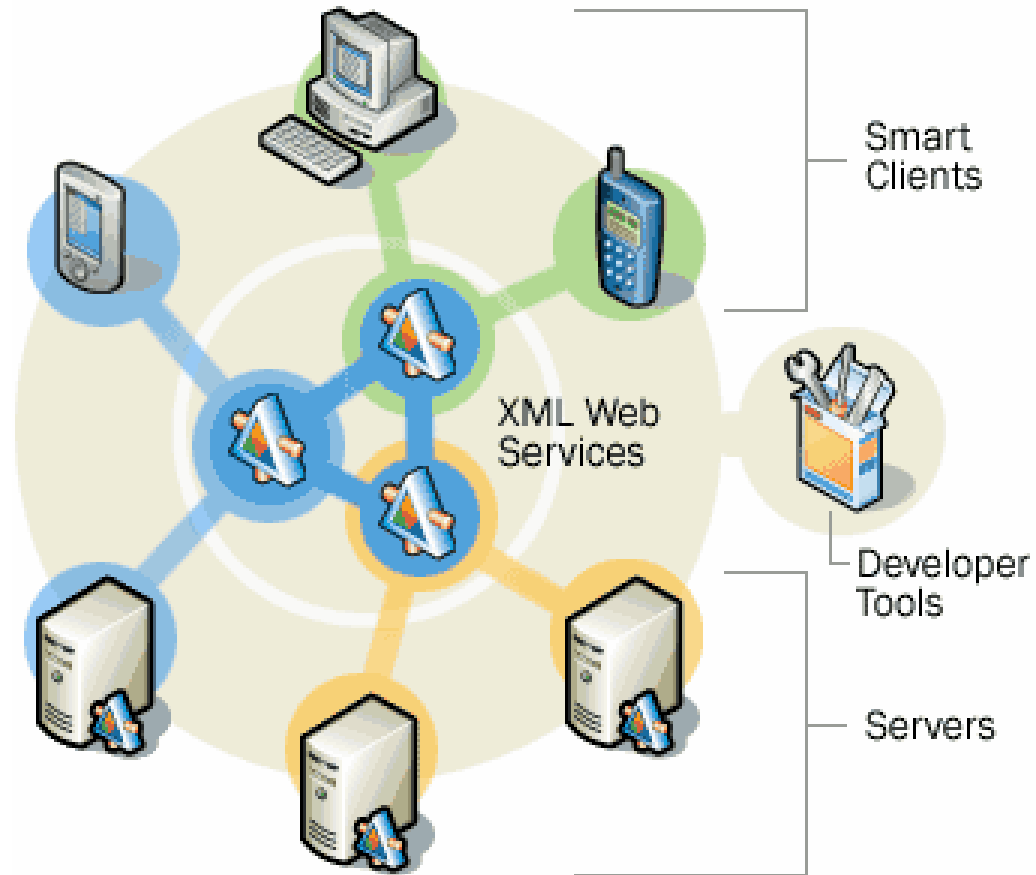
- Plataforma .net
- C#
- Desenvolvimento



Plataforma .net

Introdução ao desenvolvimento .net

Visão .net

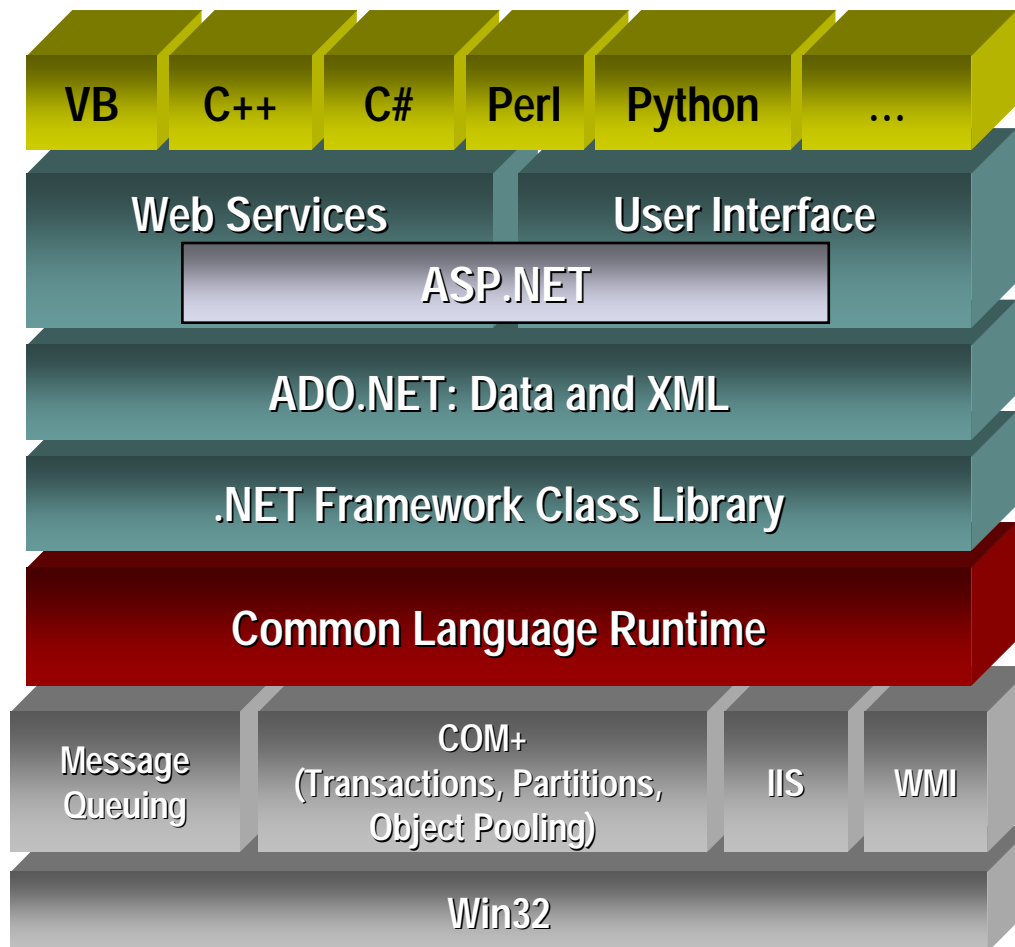




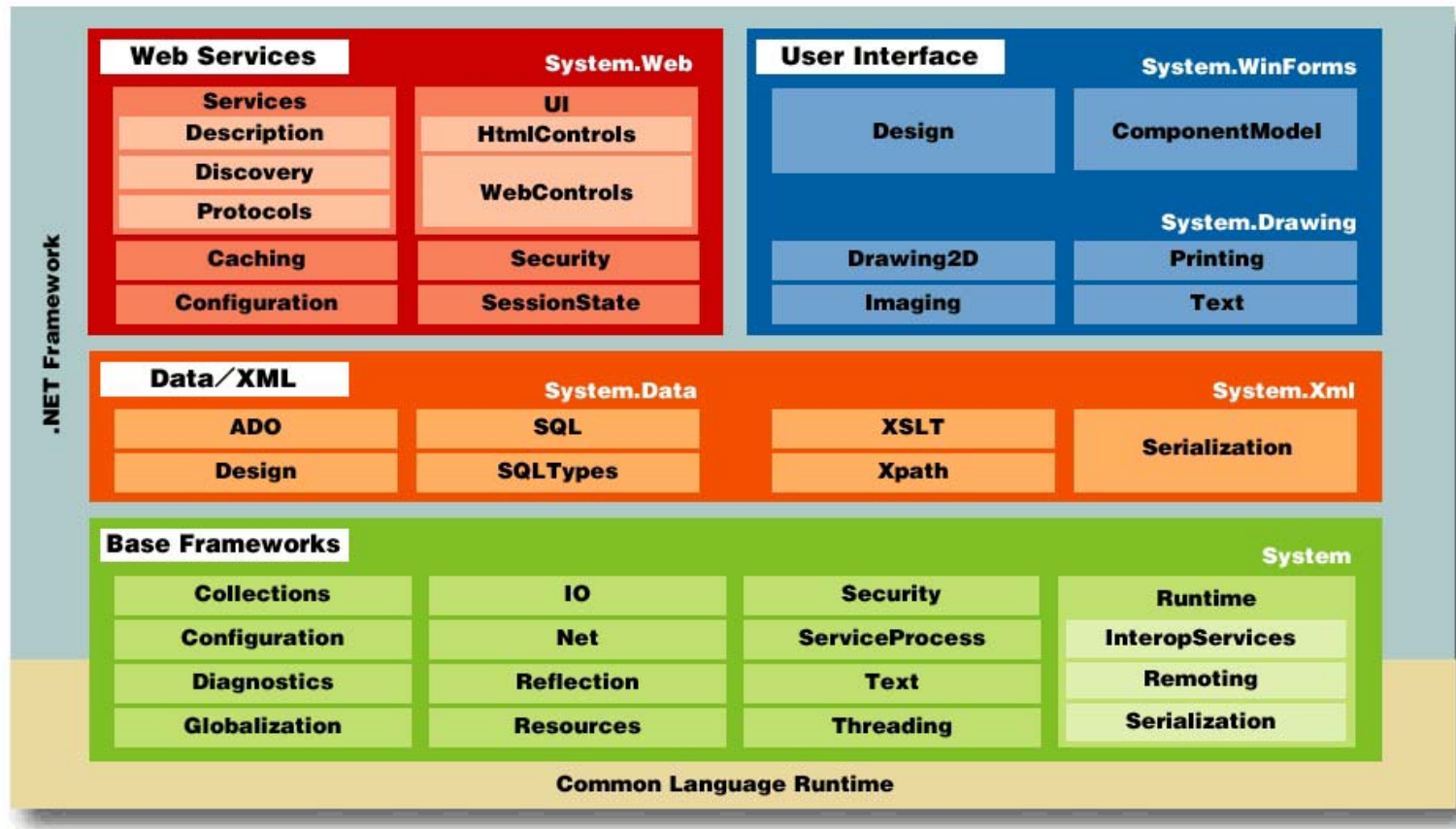
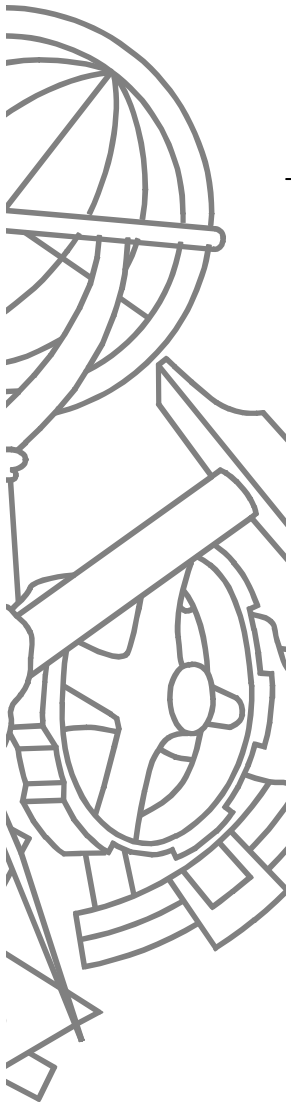
.net framework

- Plataforma de desenvolvimento
- Máquina virtual para execução
 - CLR (Common Language Runtime)
- Biblioteca de classes
 - .net framework Class Library
 - Conjunto de classes base sobre a qual se desenvolve

.net framework (2/2)



.net framework (namespaces)



CLR



.NET Framework Class Library Support

Thread Support

COM Marshaler

Type Checker

Exception Manager

Security Engine

Debugger

**MSIL to Native
Compilers**

**Code
Manager**

**Garbage
Collection**

Class Loader



Características CLR

- Gestão de memória → **Garbage collection**
 - Evita “perdas” de memória
- Excepções
 - Tratamento de erros mais elegante
- Type safety
 - Validações de compile e run time para casts e inicializações
- *Versioning*
- Gestão de processos e *threads*
- Sistema comum de tipos
- Tudo são objectos
- Orientada aos “componentes”



“Máquina virtual”

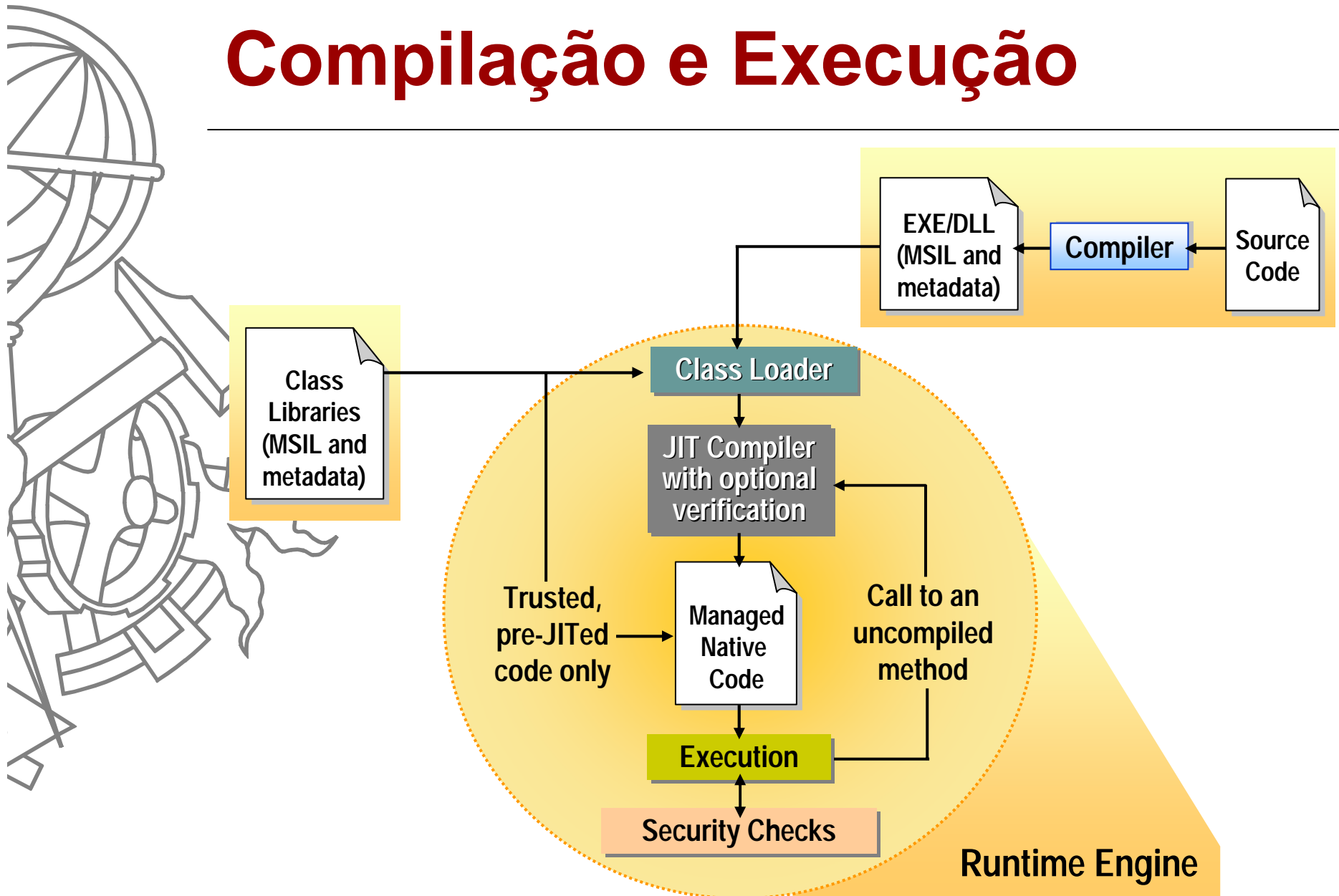
- Instanciação de **Common Language Infrastructure (CLI)**
 - Standard ECMA
- Um mesmo formato de ficheiro binário
- Um sistema de tipos comum
- Meta dados
- Linguagem intermédia (MSIL)
 - Permite várias linguagens de programação
- Conjunto de classes base

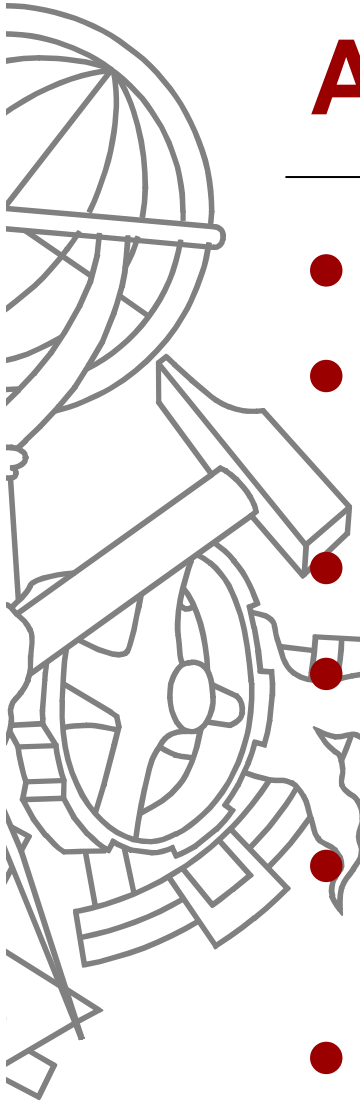


Implementações CLI

- Microsoft → CLR
- Shared Source CLI
 - Mono (Linux)
 - Rotor (FreeBSD)

Compilação e Execução





Assembly

- Blocos de construção de aplicações
- Unidade fundamental de *deployment*, *versioning*, reutilização e segurança
- Colecção de tipos e recursos
- Fornece meta-informação ao CLR para execução
- Todos os tipos existem no contexto de um assembly
- Tipos de assembly: DLL, EXE

Aplicação .net



.NET Applications



The purpose of this animation is to demonstrate how a Microsoft® .NET application is executed and how a .NET application calls a method in another single-file assembly.

(VÍdeo)



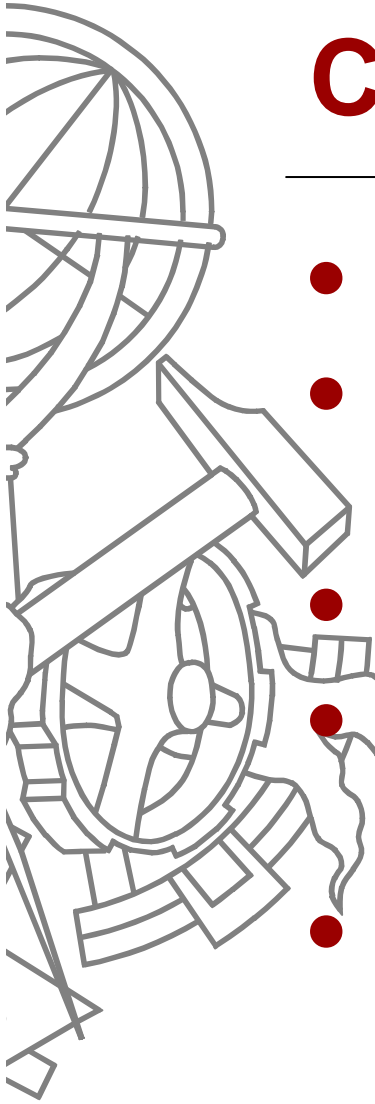
Application Domain

- Fornecem isolamento (execução e segurança) entre aplicações diferentes
- Podem existir diferentes *appdomain* em mais que um processo (ex., IIS)
- Garantia de isolamento e segurança sem overhead de criação de processo
- Permite comunicação entre *appdomain* sem overhead de IPC (mas utiliza a mesma RPC)
- Cada *appdomain* pode ser parado sem parar o processo



Appdomain & assembly

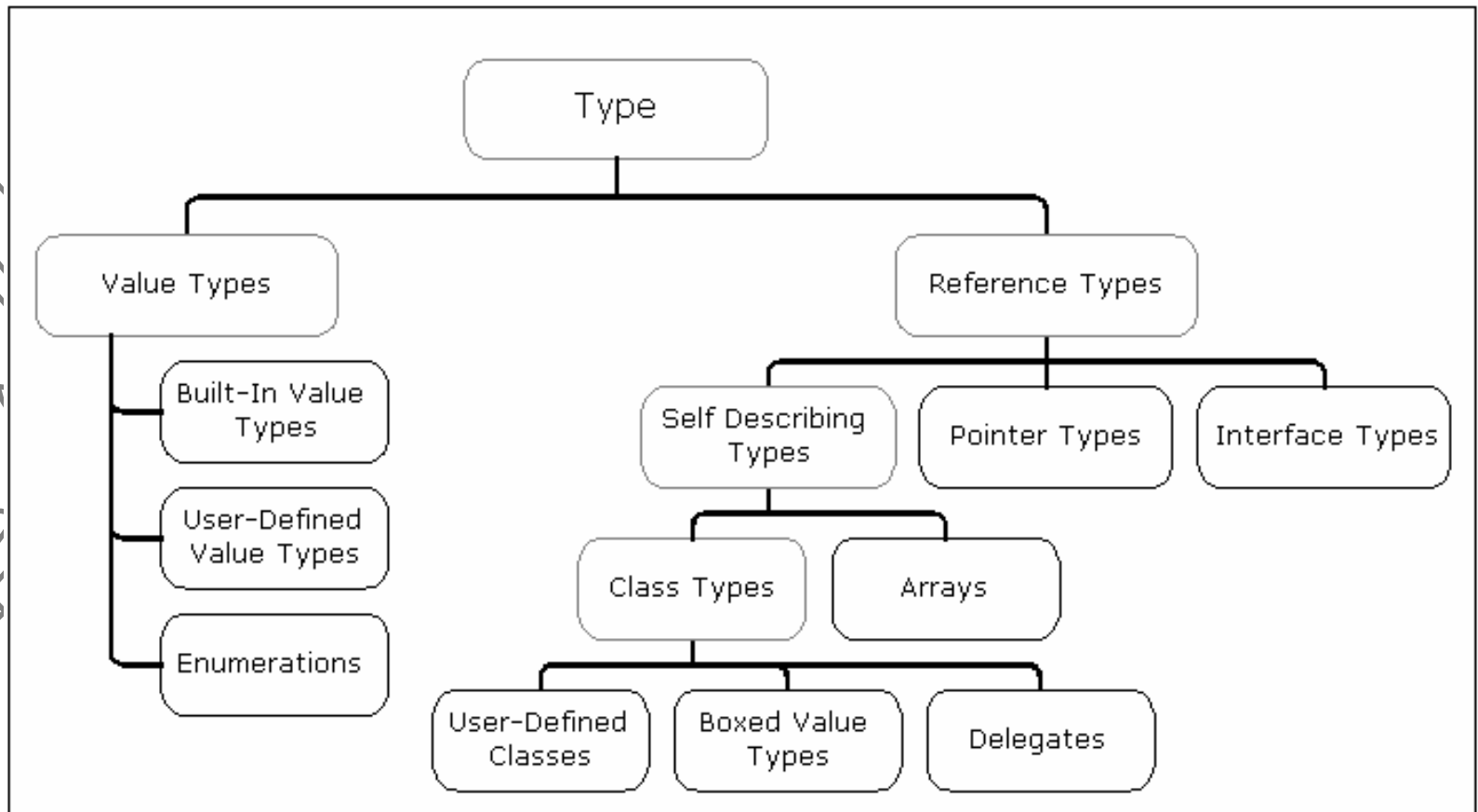
- Vários assemblies são tipicamente carregados para um appdomain
- É possível partilhar código de um assembly utilizado em vários appdomain mas não os dados



CTS

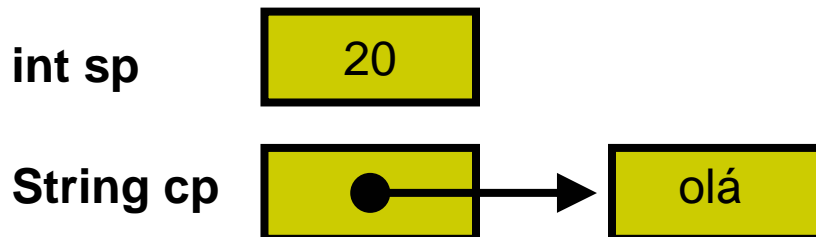
- Common Type System
- Infra-estrutura para inter-operabilidade entre linguagens de programação
- Orientado a objectos
- Suporta tipos de referência e tipos de valor
- Compatível com linguagens procedimentais

CTS (2/2)



Value-types & Reference-types

- Value types
 - Contém directamente os dados
 - Não pode ser null
- Reference types
 - Contém referência para objecto
 - Pode ser null



Garbage Collection



Simplified Garbage Collection Garbage Collection

Simplified Garbage Collection

```
public class A
{
    // ...
}

public class B
{
    public B(A a) {
        this.a = a;
    }
    // ...
    private A a;
}

// ...
public static void Main() {
    // ...
    A a1 = new A();
    A a2 = new A();
    B b1 = new B(a1);
}
```

The application creates objects a2 and b1 with b1, given a reference to a1 through its constructor. Because all of these objects are referenced by local variables on the stack, they are referred to by the root references.

(Video)



Eventos

- Mecanismo de “sinalização”
- Intrínseco ao framework
- extensivamente utilizado internamente
- Permite programação assíncrona
- Publish/subscribe



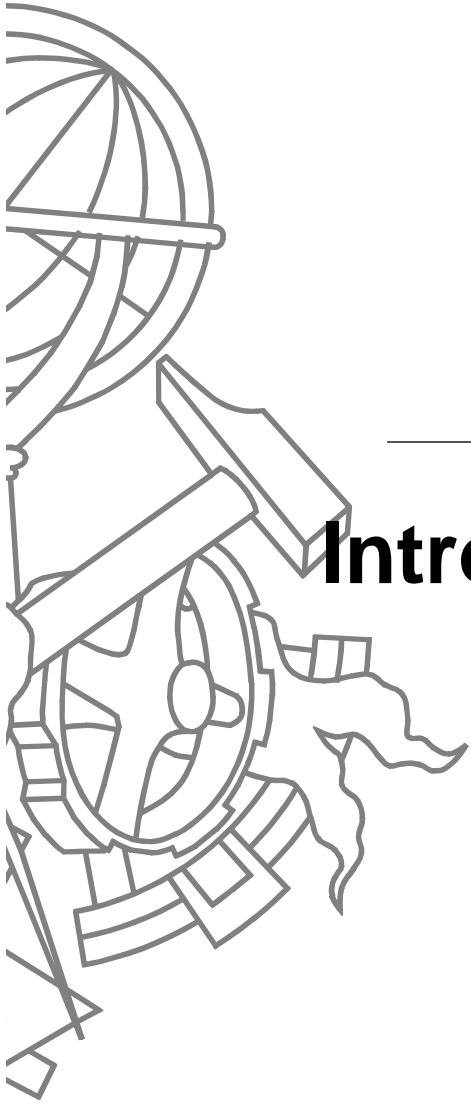
Componentes .net

- Orientada aos “componentes”
 - Propriedades, métodos e eventos
 - Design e run –time
 - Especialmente vocacionados para utilização com “design surfaces” (ex., Visual Studio)
- O termo componente em .net corresponde a uma classe que implementa a interface **IComponent** ou deriva directa ou indirectamente de **System.ComponentModel.Component**



Componentes .net (2/2)

- Componentes com interface gráfica são chamados *Control*.
- Devem derivar directa ou indirectamente de `System.Windows.Forms.Control` ou `System.Web.UI.Control`



C#

Introdução ao desenvolvimento .net



Introdução

- Nova linguagem tendo por base o C/C++
 - Também vai buscar inspiração ao Java ;-)
 - Mantém o investimento e know-how existente
 - Código mais “limpo”
 - Construções sintáticas especiais para tirar partido do framework
- Tudo são objectos
- Ficheiros com extensão .cs
- Declaração e definição de métodos no mesmo ficheiro



Tipos de dados

- object
- string
- sbyte, short, int, long
- byte, ushort, uint, ulong
- char
- float, double, decimal
- bool
- Estes tipos são *alias* para os tipos definidos na framework
 - Ex., `int == System.Int32`



Classes e namespaces

- Organização do código dentro de classes
- Classes organizadas dentro de namespaces

```
namespace Demo {  
    public class MyClass {  
        ...  
    }  
}
```



Métodos

- Sintaxe semelhante ao C/C++
- Podem ser públicos ou privados
- Suporta overloading

```
public class MyHelloWorld {  
    ...  
    public void SayHello()  
    { ... }  
  
    private void SetTitle(String Title)  
    { ... }  
}
```



Passagem de parâmetros

- Por valor
- Por referência
 - out – parâmetro de saída
 - ref – parâmetro de entrada e saída

```
public void func1(int x)
{ ... }
```

```
public void func2(out int x)
{ ... }
```

```
public void func2(ref int x)
{ ... }
```



Herança

- Apenas existe herança simples

```
public class MyClassBase {  
    ...  
    public void Func() { ... }  
}
```

```
public class MyClassDeriv : MyClassBase {  
    ...  
    public new void Func() { base.Func(); ... }  
}
```



Herança (2/2)

- Métodos **não** são virtuais por defeito

```
public class MyClassBase {  
    ...  
    public virtual void Func() { ... }  
}  
  
public class MyClassDeriv : MyClassBase {  
    ...  
    public override void Func() { base.Func(); ... }  
}
```



Propriedades

- Sintaxe alternativa para acesso a membros de dados da classe mas com as vantagens dos métodos

```
public class Button : Control
{
    private string caption;
    public string Caption {
        get { return caption; }
        set { caption = value; Repaint(); }
    }
    ...
}
```




Operadores

- Atribuição

- =

- Relacionais

- < <= > >= == !=

- Lógicos

- && || !

- Aritméticos

- + - * / %


- += -= *= /= ++ --



Constantes

- Pré-definidas
 - null
 - true false
- De utilizador
 - `const string Ver = "1.0b";`

Criação de objectos



```
// definição da classe
public class MyClass { ... }

// definição da variável
MyClass obj;

// criação do objecto
obj = new MyClass();
```



Construtores

- Seguem as regras do C/C++
 - Mesmo nome da classe
 - Sem tipo de retorno
 - Podem ter ou não argumentos

```
public class MyClass {  
    ...  
    public MyClass() { ... }  
    public MyClass(String Title) { ... }  
}
```



Arrays

- Suportados ao nível da biblioteca base de classes em `System.Array`

```
// declaração do vector  
String[] vec;
```

```
// criação do vector  
vec = new String[10];
```

```
// número de elementos pode ser dinâmico  
vec = new String[n];
```

Ciclos

```
// repetição n vezes  
for (int x = 0; i < vec.Length; i++)  
    Console.WriteLine(vec[i]);
```

```
// repetição condicional  
int i = 0;  
while (i < vec.Length)  
{  
    Console.WriteLine(vec[i]);  
    i++;  
}
```

```
// enumeração  
foreach (String x in vec)  
    Console.WriteLine(x);
```





Condicionais

```
// teste de decisão
if (i < vec.Length)
    Console.WriteLine(vec[i]);
else
    Console.WriteLine("Erro!!!");

// teste múltiplo
switch (x)
{
    case 1: ...; break;
    case 2: ...; goto case 3;
    case 3: ...; break;
    default: ...; break;
}
```



Interfaces

- Semelhantes a classes mas não têm implementação dos métodos
- Apenas definem as assinaturas
- Todos os métodos são públicos

```
public interface IMovimentavel
{
    void MoverEsquerda();
    void MoverDireita();
    ...
}
```




Implementação de Interfaces

- Qualquer classe pode implementar uma ou mais interfaces

```
public class Pessoa : IMovimentavel
{
    void MoverEsquerda() { ... }
    void MoverDireita() { ... }
    ...
}
```



Enumerados

- Fortemente “tipados”
 - Sem conversão automática para int
 - Suportam operadores +, -, ++, --, &, |, ^, ~
- Pode-se definir tipo de dados base
 - Byte, short, int, long

```
enum Color : byte {  
    Red = 1,  
    Green = 2,  
    Blue = 4,  
    Black = 0,  
    White = Red | Green | Blue  
}
```

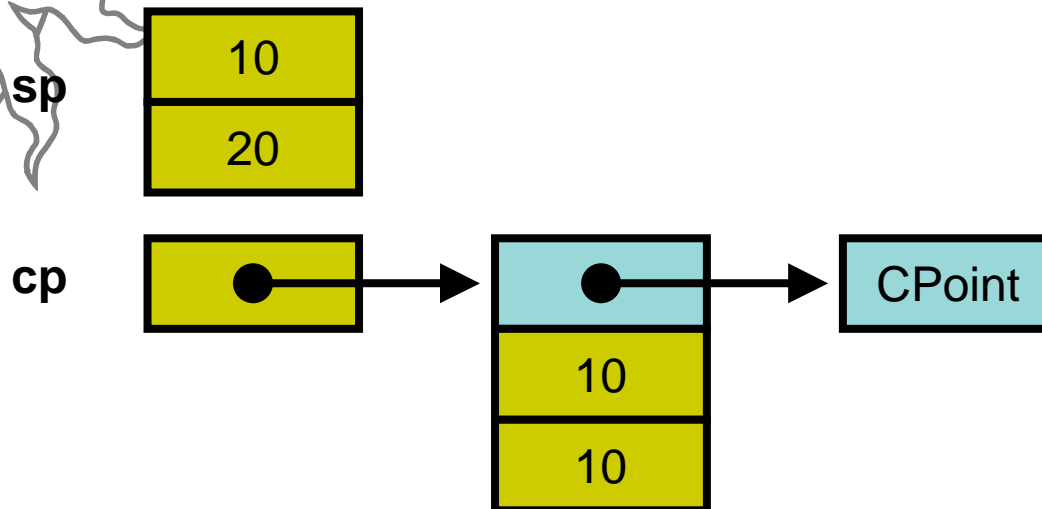


structs

- Semelhantes a classes, excepto
 - Alocação na stack e não no heap
 - Não suporta herança
 - Cópia (atribuição) de conteúdo e não de referência
 - Ideal para conceitos pequenos (ex., Complex)
 - Utilizada nos tipos primitivos da framework (ex. int)
- **Benefícios**
 - Como não são alocadas no heap não colocam carga sobre o mecanismo de garbage collection

Classes e estruturas

```
class CPoint { int x, y; ... }  
struct SPoint { int x, y; ... }  
SPoint sp = new SPoint(10, 20);  
CPoint cp = new CPoint(10, 20);
```





delegates

- Ponteiros (orientados a objectos) para métodos
- Permite múltiplos receptores
 - Cada delegate tem uma lista de invocação
 - Publish/subscribe
- Base para o mecanismo de eventos


```
delegate void MouseEvent(int x, int y);
```

```
delegate double Func(double x);
```

```
Func fn = new Func(Math.Sin);
```

```
double x = fn(1.0);
```

Comentários XML

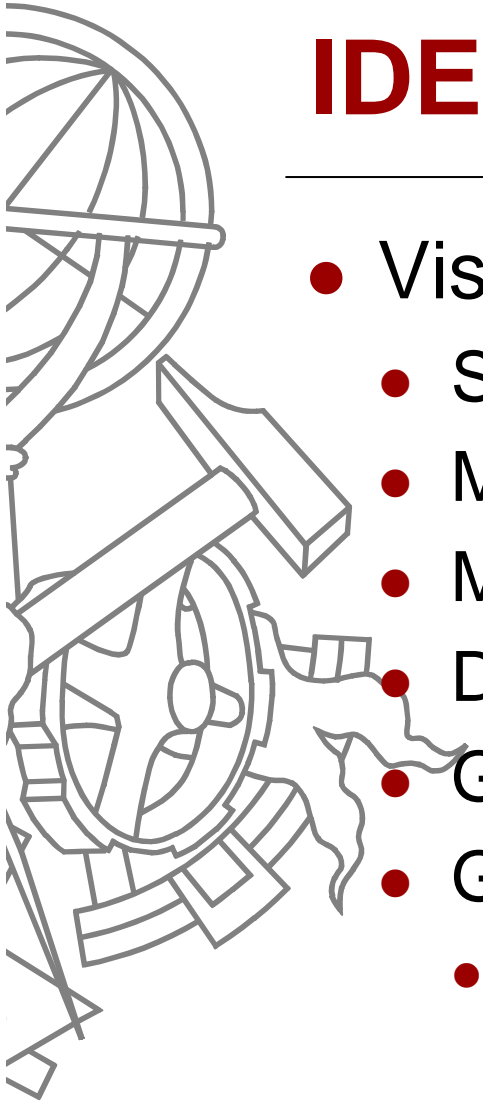


```
class XmlElement
{
    /// <summary>
    /// Returns the attribute with the given name and
    /// namespace</summary>
    /// <param name="name">
    /// The name of the attribute </param>
    /// <param name="ns">
    /// The namespace of the attribute, or null if
    /// the attribute has no namespace</param>
    /// <return>
    /// The attribute value, or null if the attribute
    /// does not exist</return>
    /// <seealso cref="GetAttr(string)"/>
    public string GetAttr(string name, string ns) {
        ... ..
    }
}
```



Desenvolvimento .Net

Introdução ao desenvolvimento .net



IDE

- Visual Studio .net 2003
 - Solução multi-projecto
 - Multi-linguagem
 - Múltiplos tipos de projecto
 - Debugger
 - Geração de código .net
 - Geração de código nativo
 - Evolução do VC++ 6.0

Demo – Hello World



```
using System;

namespace ConsoleApplication1
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
    class Class1
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            Console.WriteLine("Olá Mundo");
        }
    }
}
```

The screenshot shows the Microsoft Visual C# .NET [design] environment. The main window displays the source code for Class1.cs. The code defines a namespace ConsoleApplication1 and a class Class1. The class contains a static void Main method that writes "Olá Mundo" to the console. The interface includes a menu bar (File, Edit, View, Project, Build, Debug, Tools, Window, Help), a toolbar, a Solution Explorer on the left showing the project structure, a Class View on the right showing the class hierarchy, and a Properties window at the bottom right. The status bar at the bottom indicates the current line (Ln 4), column (Col 2), and character (Ch 2).

Demo - Aritmética



The screenshot displays the Microsoft Visual Studio .NET IDE in design mode for a project named 'Aritmetica'. The main window shows the design view of 'Form1.vb', which contains a form titled 'Teste Aritmetica'. The form has two text boxes for '1º Operando', a 'ComboBox1', and buttons for 'Executar' and 'Sair'. The 'Resultado' label is positioned below the text boxes. The IDE interface includes a menu bar (File, Edit, View, Project, Build, Debug, Data, Tools, Window, Help), a toolbar, a Class View on the right showing the project structure, and a Properties window at the bottom right. The status bar at the bottom indicates 'Ready'.

Class View - Aritmetica

- Aritmetica
 - Aritmetica
 - Bases and Interfaces
 - Adicao(float, float)
 - Aritmetica()
 - Divisao(float, float)
 - Multiplicacao(float, float)
 - Subtracao(float, float)
 - TesteAritmeticaWinForms
 - TesteAritmeticaWinForms
 - Form1
 - TesteAritmeticaConsole
 - TesteAritmeticaConsole

Properties

Aritmetica Project Properties

Misc	
Project File	Aritmetica.csproj
Project Folder	C:\Documents and Si
Policy	
Policy File	

Misc

Task List Output Find Results 1 Index Results Search Results for adOpenStatic

Ready



Perguntas & Respostas

Introdução ao desenvolvimento .net



Mais Informação...

- MSDN Library
 - <http://msdn.microsoft.com/library>
- .net framework center
 - <http://msdn.microsoft.com/netframework/>
- C#
 - <http://msdn.microsoft.com/vcsharp/>
- ASP.net
 - <http://www.asp.net>
- Laboratório .net do ISEP/IPP
 - <http://www.dei.isep.ipp.pt/labdotnet/>



Mais Informação...

- Open CLI
 - <http://sourceforge.net/projects/ocl>
- Mono (.net @ Unix)
 - <http://www.go-mono.com/>
- ECMA
 - <http://www.ecma-international.org/>



Mais Informação...

- Introduction to C# @ ECMA
 - <http://www.ecma-international.org/activities/Languages/Introduction%20to%20Csharp.pdf>
- Common Language Infrastructure @ ECMA
 - <http://www.ecma-international.org/activities/Languages/ECMA%20CLI%20Presentation.pdf>