



Ambientes de Desenvolvimento Avançados

<http://www.dei.isep.ipp.pt/~jtavares/ADAV>

Aula 2

Engenharia Informática

2005/2006

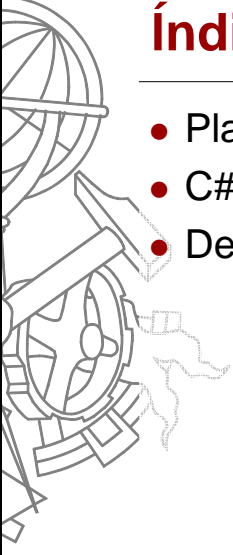
José António Tavares
jrt@isep.ipp.pt



BASEADO na “Introdução ao Desenvolvimento .NET”

de Paulo Sousa

Instituto Superior de Engenharia do Porto
Instituto Politécnico do Porto



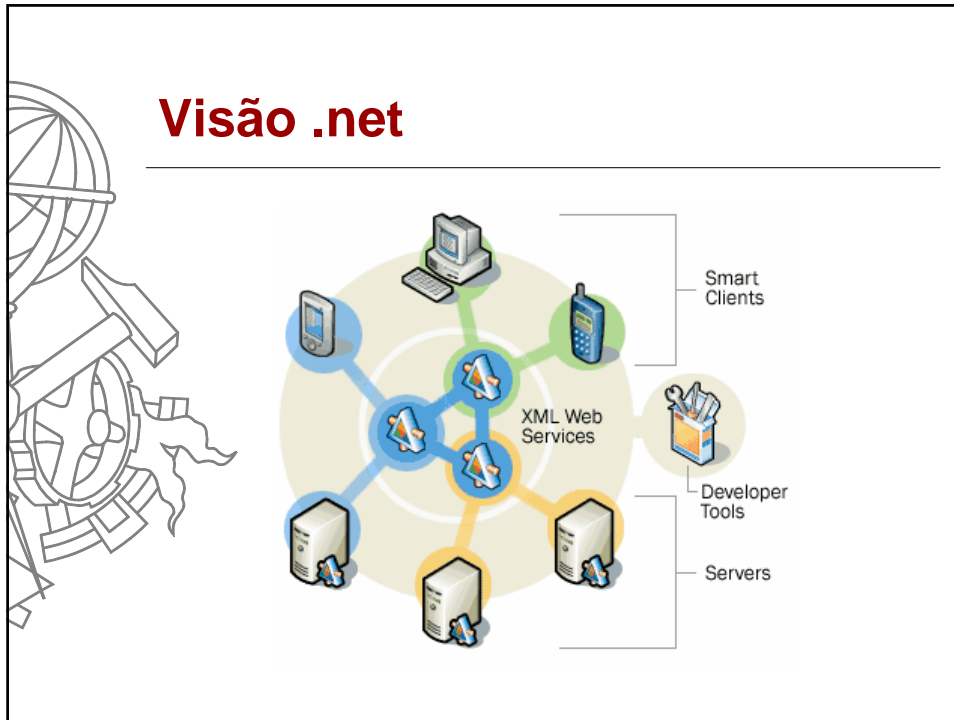
Índice

- Plataforma .net
- C#
- Desenvolvimento

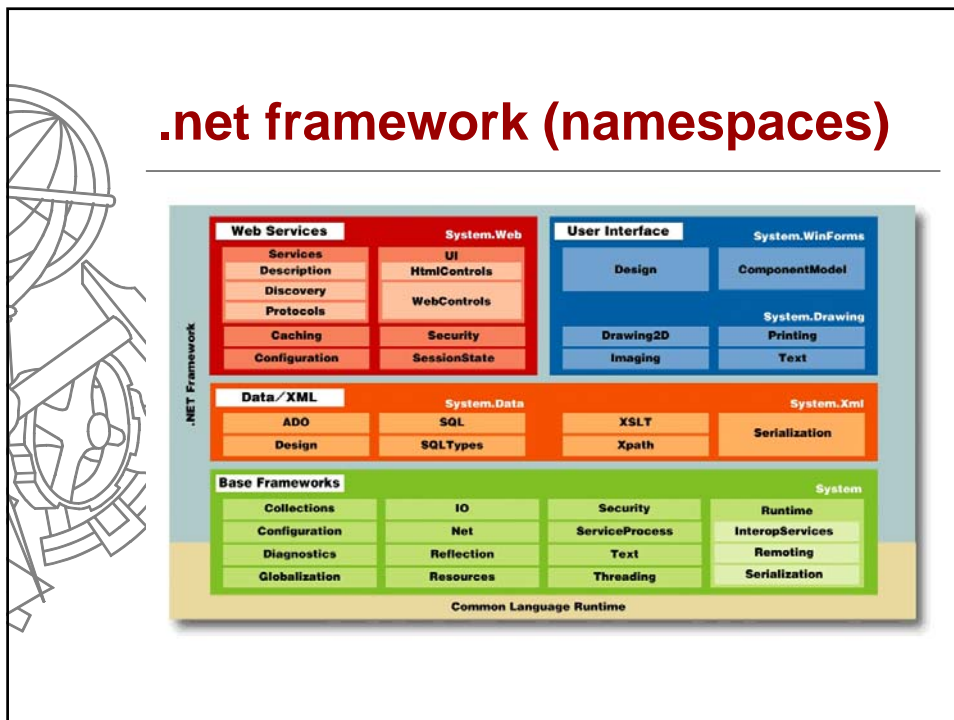
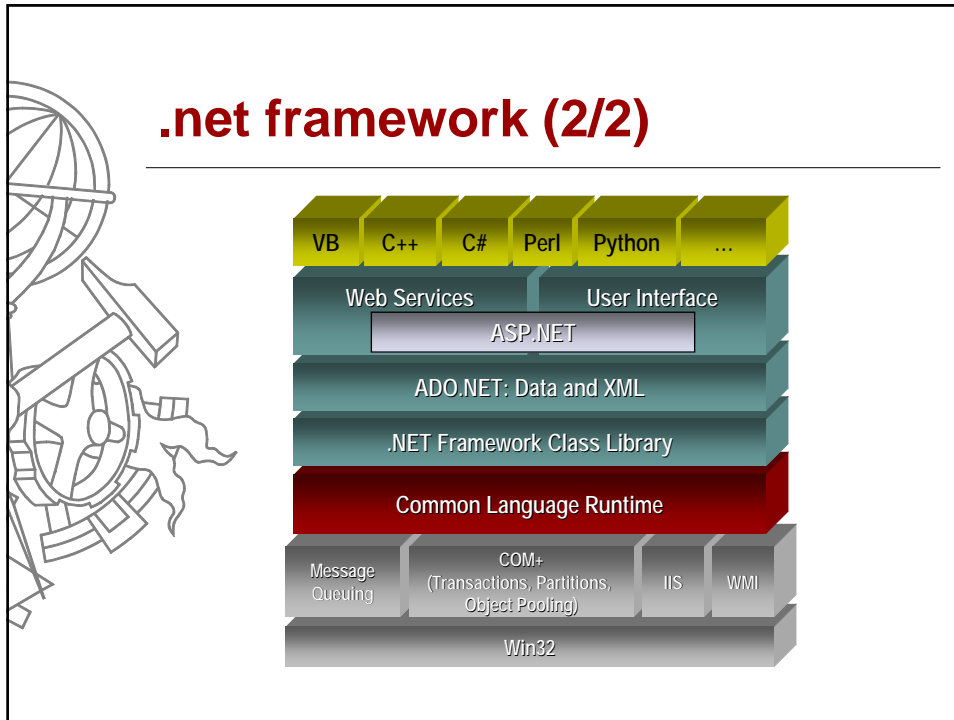


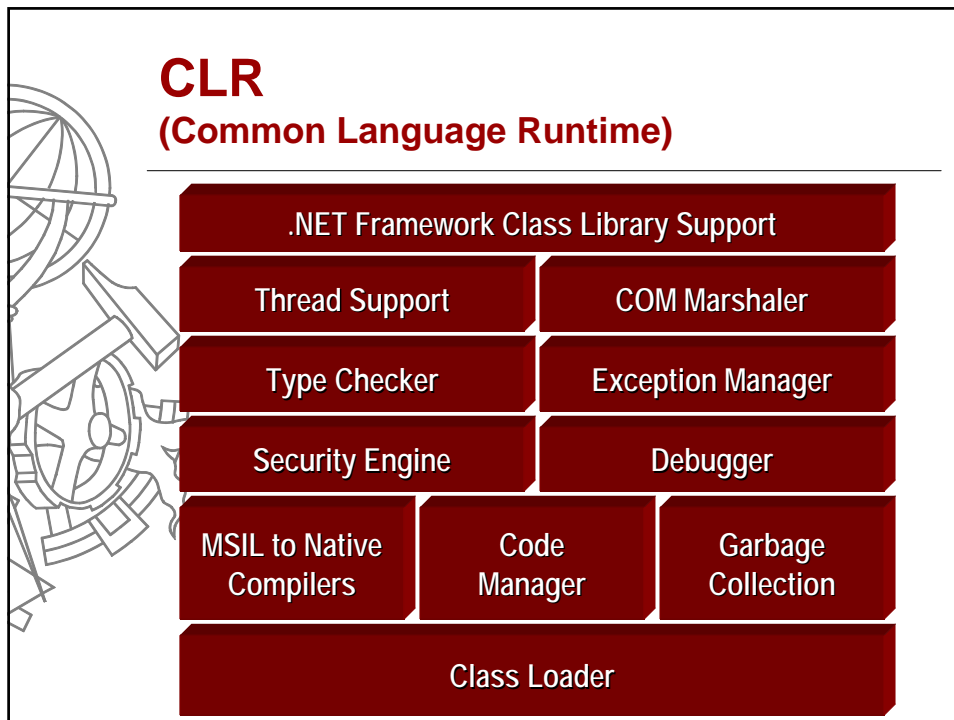
Plataforma .net

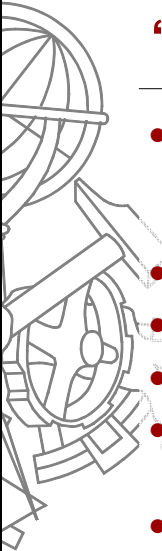
Introdução ao desenvolvimento .net



- ## .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

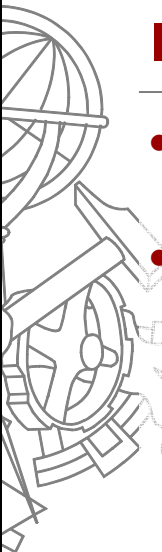






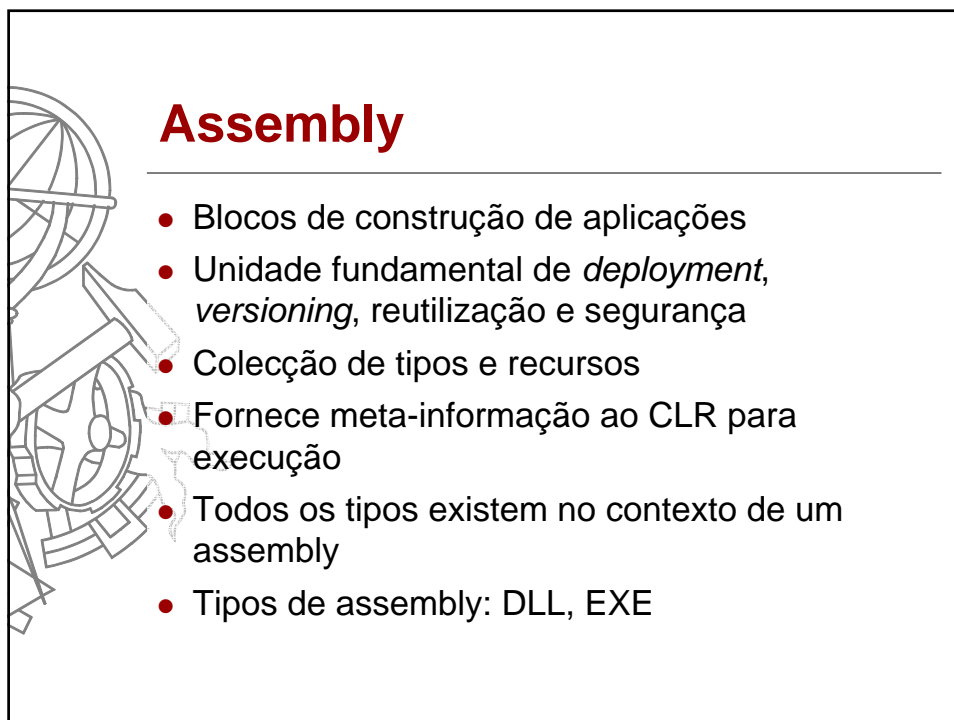
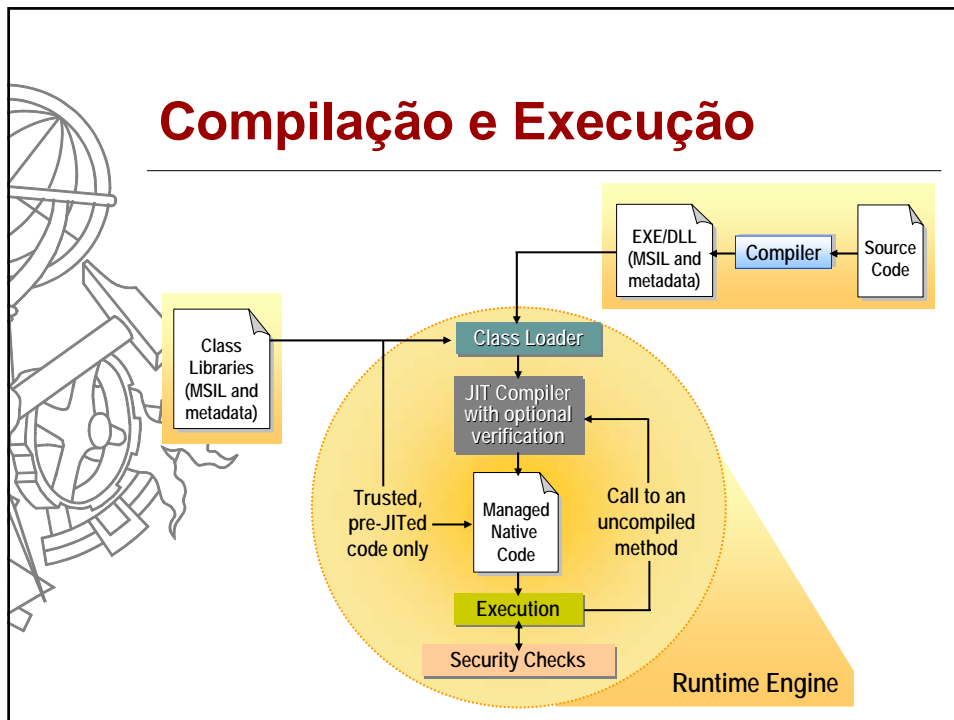
“Máquina virtual”

- 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

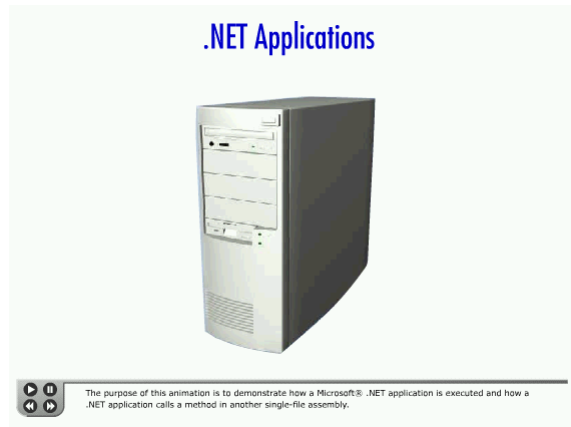


Implementações CLI

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



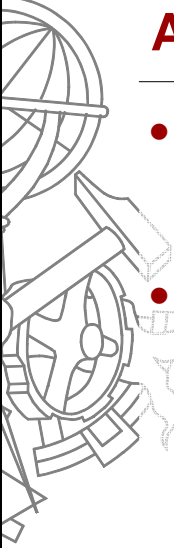
Aplicação .net



(VÍdeo)

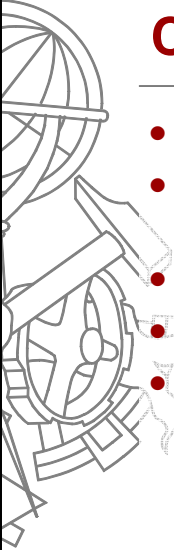
Application Domain

- 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

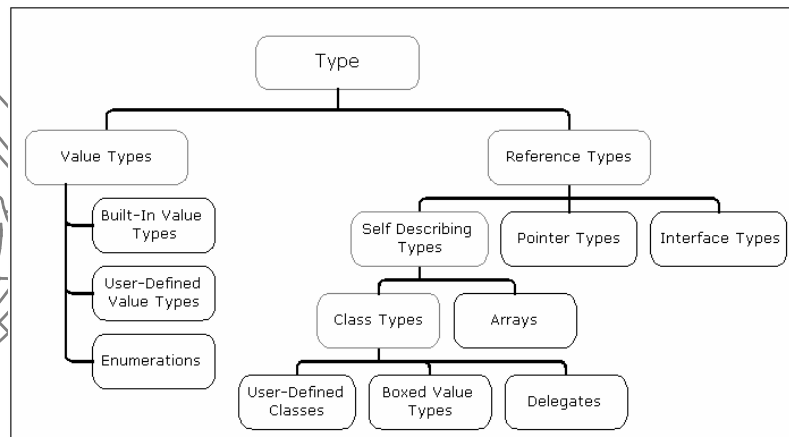
- Vários assemblies são tipicamente carregados para um appdomain
- É possível partilhar código de um assembly utilizado em vários appdomain mas não os dados



CTS

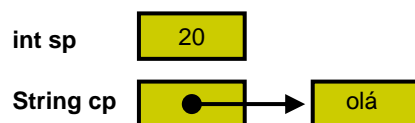
- **Common Type System**
- Infra-estrutura para inter-operabilidade entre linguagens de programação
- Orientado a objectos
- Suporta tipos de referência e tipos de valor
- Compatível com linguagens procedimentais

CTS (2/2)

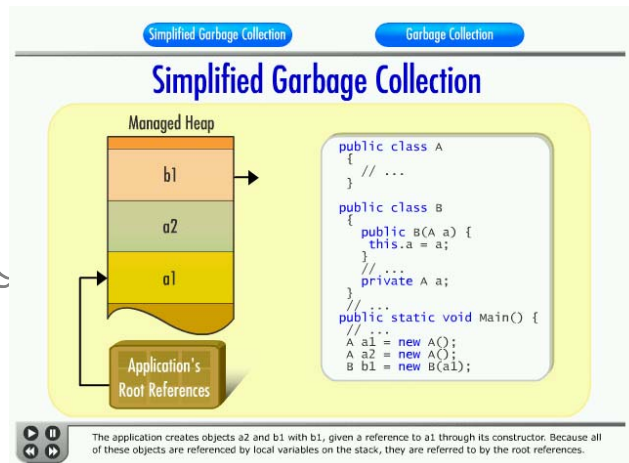


Value-types & Reference-types

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



Garbage Collection



(VÍdeo)

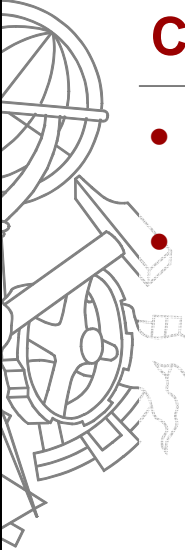
Eventos

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



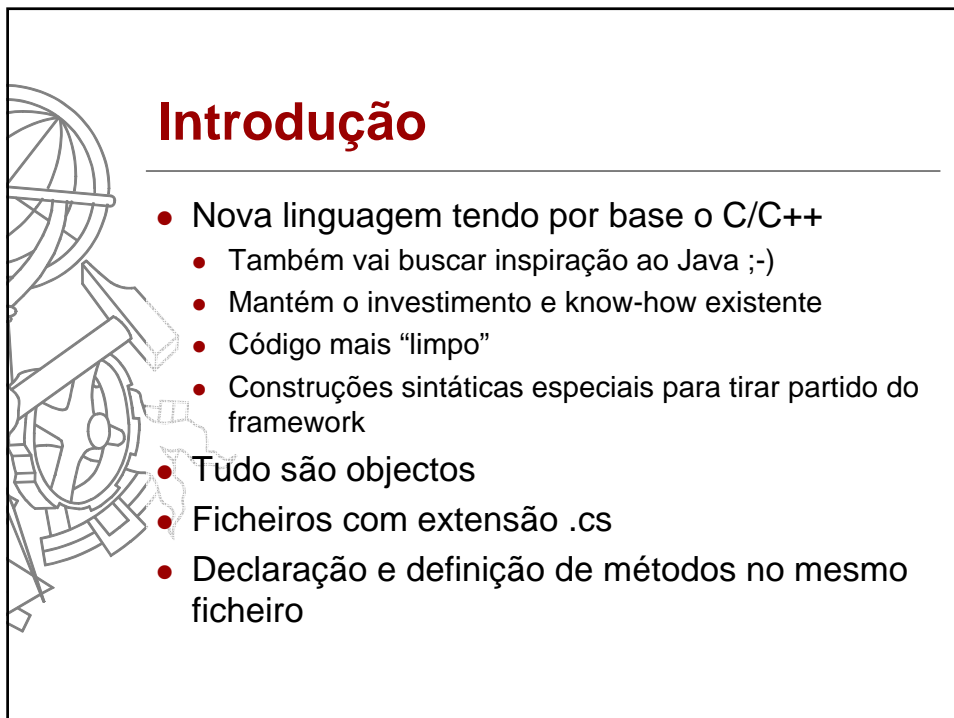
Componentes .net

- 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**





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

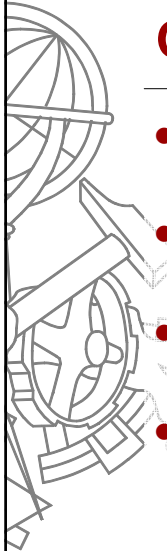
- Sintaxe alternativa para acesso a membros de dados da classe mas com as vantagens dos métodos

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

Propriedades (example)


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

    public override void Paint(Graphics g, Rectangle r)
    {
        // Painting code goes here
    }
}
```



Operadores

- Atribuição
 - =
- Relacionais
 - < <= > >= == !=
- Lógicos
 - && || !
- Aritméticos
 - + - * / %
 - += -= *= /= ++ --



Constantes

- Pré-definidas
 - nul l
 - 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; x < vec.Length; x++)
    Console.WriteLine(vec[x]);

// 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

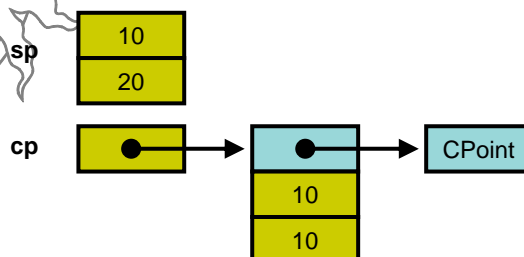
```
enum Color : byte {
    Red = 1,
    Green = 2,
    Blue = 4,
    Black = 0,
    White = Red | Green | Blue
}
```

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);
```

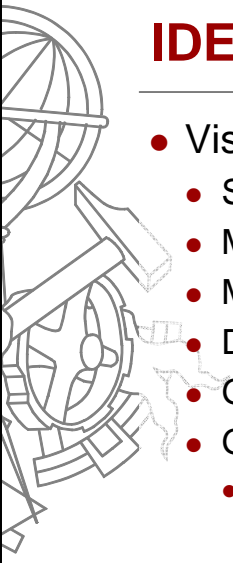
Comentários XML

```
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

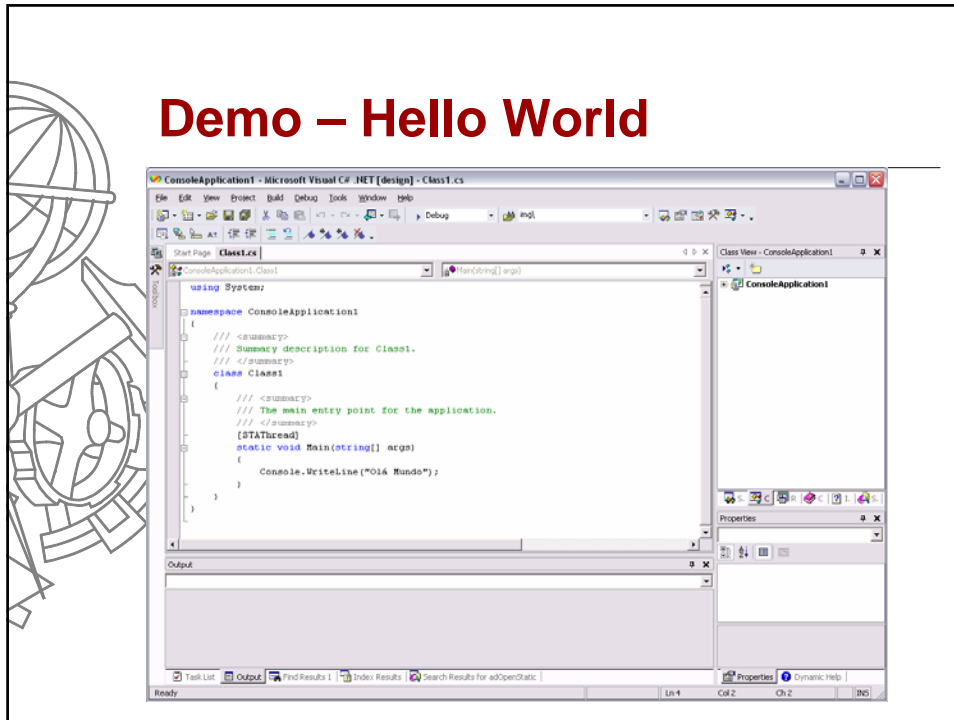
Introdução ao desenvolvimento .net



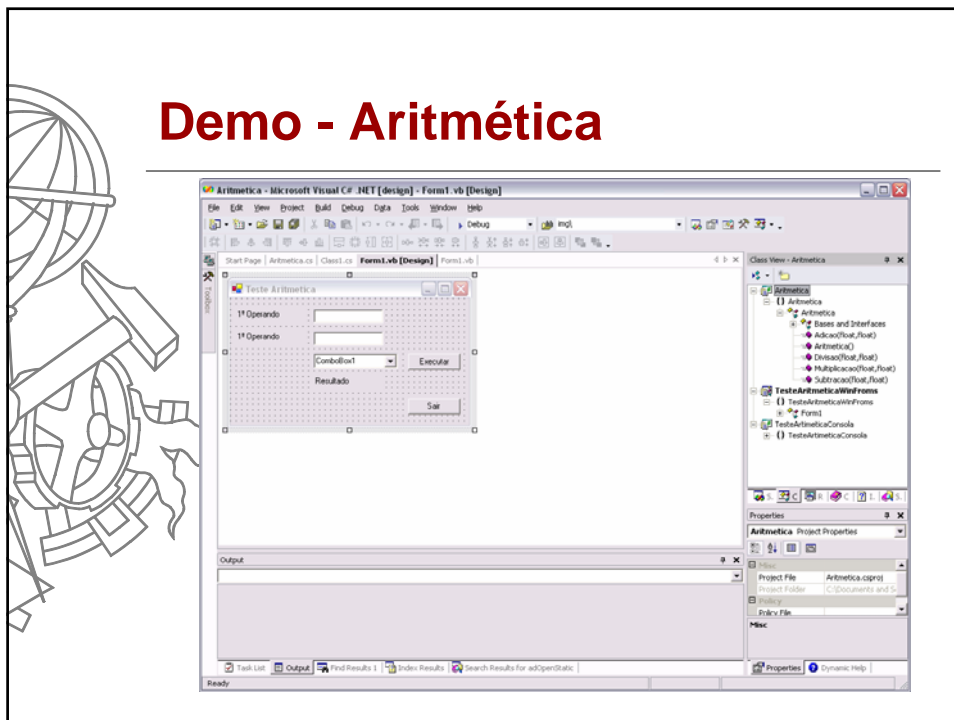
IDE


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

Demo – Hello World




Demo - Aritmética





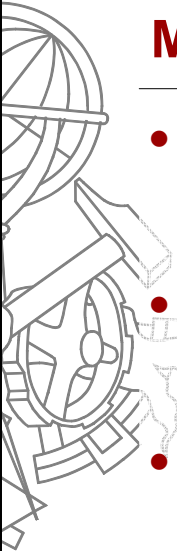
Perguntas & Respostas

Introdução ao desenvolvimento .net



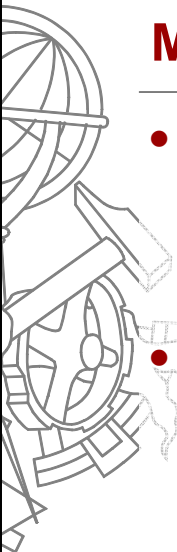
Mais Informação...

- 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/>



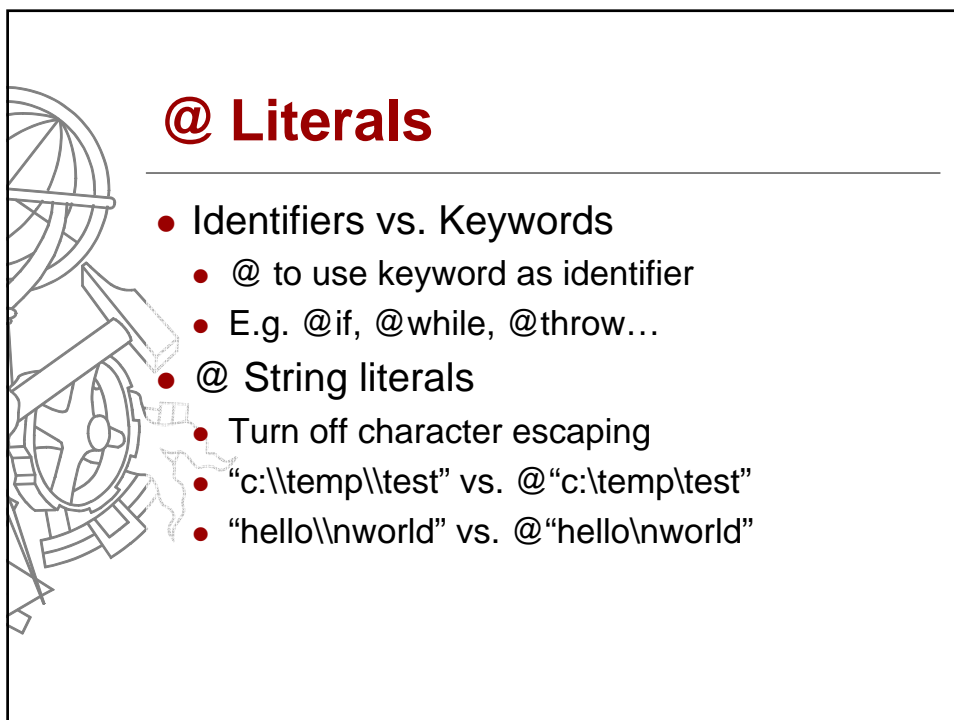
Mais Informação...

- 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>



@ Literals (examples)

```

string a = "hello, world";           // hello, world
string b = @"hello, world";         // hello, world
string c = "hello \t world";        // hello      world
string d = @"hello \t world";       // hello \t world
string e = "Joe said \"Hello\" to me"; // Joe said "Hello" to me
string f = @"Joe said ""Hello"" to me"; // Joe said "Hello" to me
string g = "\\server\share\file.txt"; // \server\share\file.txt
string h = @"\\server\share\file.txt"; // \server\share\file.txt
string i = "one\two\nthree";
string j = @"one
two
three";

```

Enumerations

- Have underlying type
- First class value (not int as in C/C++)
- Defines conversions
- Defines parsing
- Runtime information

Reference and Output parameters

- ref keyword
 - Does not accept null
 - Needs to be initialized
- out keyword
 - Cannot be read inside the function
 - Need not be initialized outside
- Example:
 - Declaration: `void f (int x, ref int y, out int z);`
 - Call: `f(5, ref a, out b);`

Operator Overloading

- Overloadable
 - Unary: `+` `-` `!` `~` `++` `--` `true` `false`
 - Binary: `+` `-` `*` `/` `%` `&` `|` `^` `<<` `>>` `==` `!=` `<` `>` `<=` `>=`
- Non-overloadable
 - `.` `->` `=` `&&` `||` `?` `:` `new` `typeof` `sizeof` `is` `as`
- Implicit assignment overloading
 - E.g. `*=` when `*` is overloaded
- Cast overloading
 - Conversions
- `[]` Overloading
 - Indexer properties
- At least one parameter is the declaring class



Operator Overloading (cont.)

- Advanced cases:
 - Operators “true” and “false”
 - Used in Boolean conditions and `&&` `||`
 - `&&` is `T.false(x) ? x : T.&(x, y)`
 - `||` is `T.true(x) ? x : T.|(x, y)`



Variable argument lists

- Conventional way
 - `void f (int x, int y, params float[] f);`
 - Call: `f(4, 5, 4.5, 2.3, ...);`
- Undocumented way
 - More general
 - Slower
 - `void f(__arglist);`
 - If you are interested see <http://www.iunknown.com/Weblog/fog0000000098.html>




checked and unchecked

- Checked and unchecked contexts
 - [un]checked(...)
 - [un]checked ...;
- Affected:
 - ++ -- + - * /
 - Explicit conversions



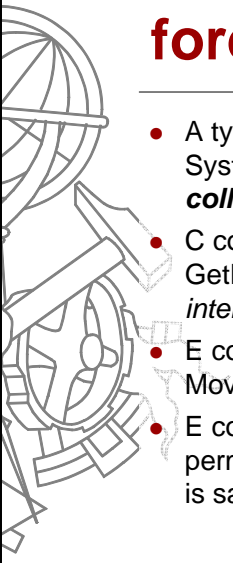
“is” and “as”

- Dynamic type checking
 - O is T
 - O as T
- Is can be very powerful
 - Knows about boxing and stuff...
- As in not like a cast
 - Returns null on error (not exception)




foreach

- Use of enumerators in the language
 - IEnumerable
 - Enumeration patter...
- `foreach (type identifier in expression) embedded-statement`



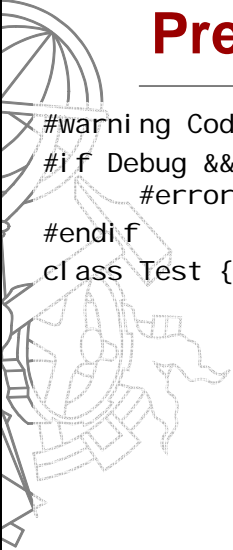
foreach (formal)

- A type C is said to be a **collection type** if it implements the System.IEnumerable interface or implements the **collection pattern** by meeting all of the following criteria:
 - C contains a public instance method with the signature GetEnumerator() that returns a *struct-type*, *class-type*, or *interface-type*, which is called E in the following text.
 - E contains a public instance method with the signature MoveNext() and the return type bool.
 - E contains a public instance property named Current that permits reading the current value. The type of this property is said to be the **element type** of the collection type.




Preprocessor

- #define
- #undef
- #if
- #else
- #endif
- #region
- #endregion
- #error
- #warning
- #line




Preprocessor (examples)

```
#warning Code review needed before check-in
#if Debug && Retail
    #error A build can't be both debug and retail
#endif
class Test {...}
```



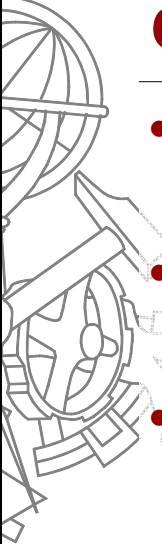
Others

- readonly keyword
- Static constructors
- “switch” works on strings
- “using” keyword
- “using” directives
 - nesting
 - aliases
- “lock” keyword
- “sealed” and “new” keywords



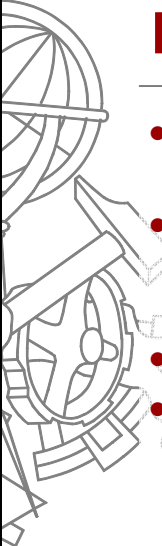
Classes

- Special Classes
 - Array, Delegate, Enum, ValueType
- “const” vs. “readonly” fields
 - “static readonly” fields
- “abstract”, “virtual”, “override” and “sealed” methods
- “external” methods
- Strange constructor behavior
 - Code for initialized fields executed before the constructor in object creation!



Classes (cont.)

- Constructors
 - Instance
 - Class (static)
- Overloading
 - this(...)
 - base(...)
- Default constructors
 - base()



Properties

- Look like fields
 - Access pattern
- Behave like functions
 - Can be virtual, ...
 - Have “get” and “set” methods
- Can be static
- Treated as a single member of the class
 - “new” hiding example



Properties (example)

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

    public override void Paint(Graphics g, Rectangle r)
    {
        // Painting code goes here
    }
}
```



Indexers

- Essentially overriding the [] operator
- Behaves as a property
 - With parameters – the index
 - “get”, “set” again
- Can be overloaded

Indexers (example)

```

using System;
class BitArray
{
    int[] bits;
    int length;
    public BitArray(int length) {
        if (length < 0) throw new ArgumentException();
        bits = new int[((length - 1) >> 5) + 1];
        this.length = length;
    }
    public int Length {
        get { return length; }
    }
    public bool this[int index] {
        get {
            if (index < 0 || index >= length) {
                throw new IndexOutOfRangeException();
            }
            return (bits[index >> 5] & 1 << index) != 0;
        }
        set {
            if (index < 0 || index >= length) {
                throw new IndexOutOfRangeException();
            }
            if (value) {
                bits[index >> 5] |= 1 << index;
            }
            else {
                bits[index >> 5] &= ~(1 << index);
            }
        }
    }
}

```

Topic 5: C# Syntax Features

Arrays

- Inherit from System.Array
- Initialization
- “foreach” works
- Jagged arrays
- Multi-dimensional arrays
- Array covariance
 - Example to follow...
 - Runtime check in assignment

