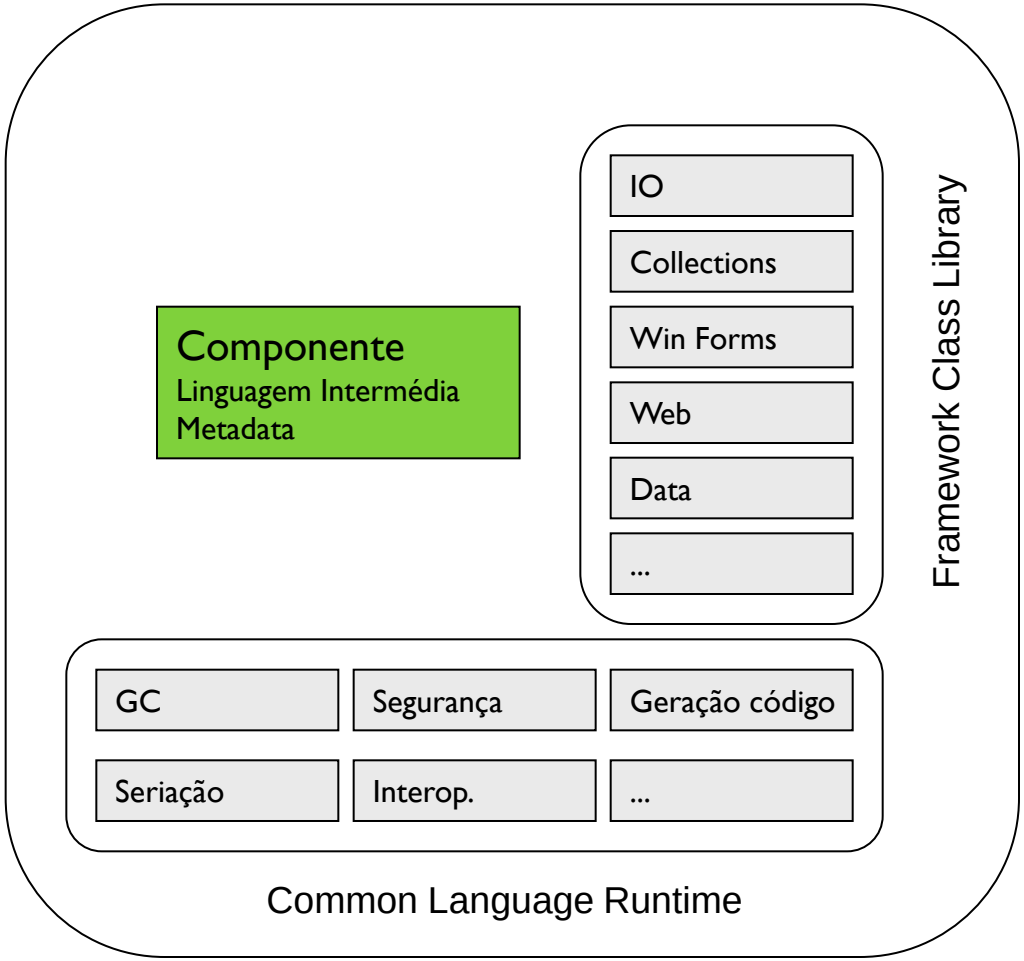
Ambientes Virtuais de Execução

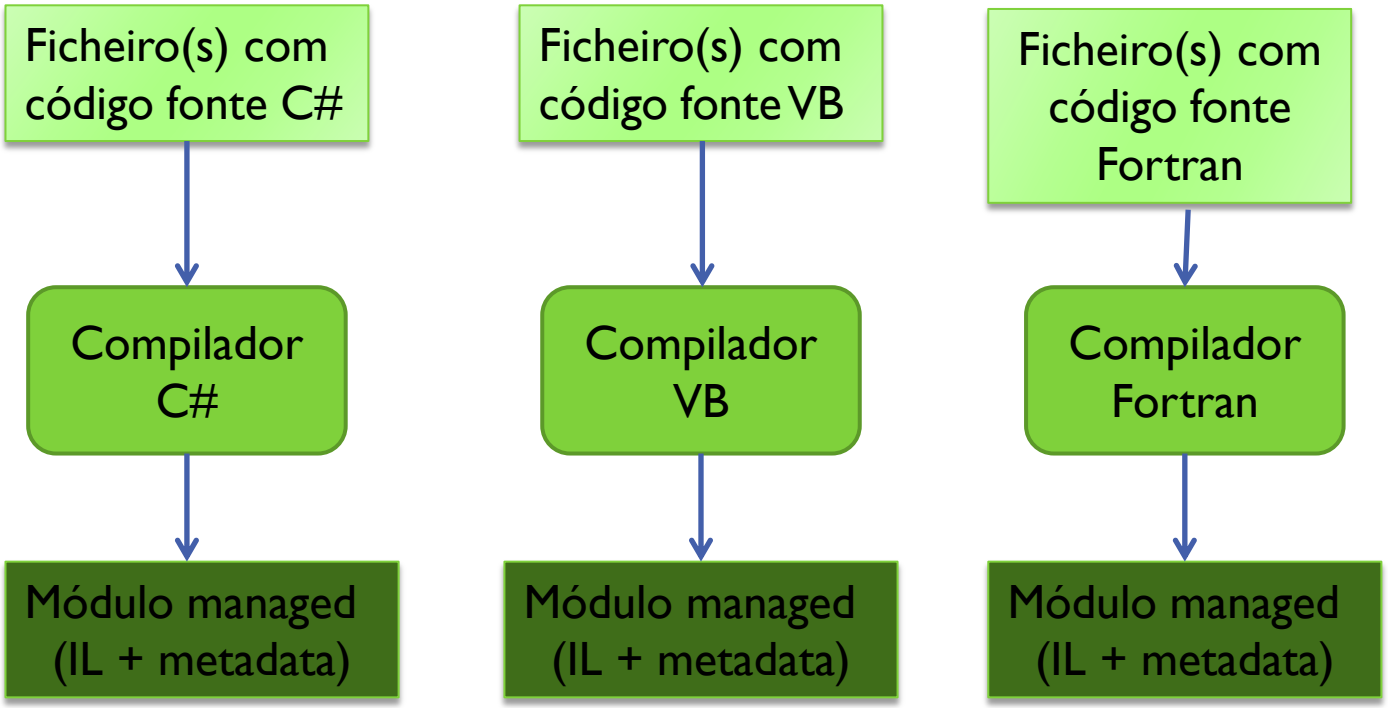
O modelo de execução do CLR

Ambiente de execução



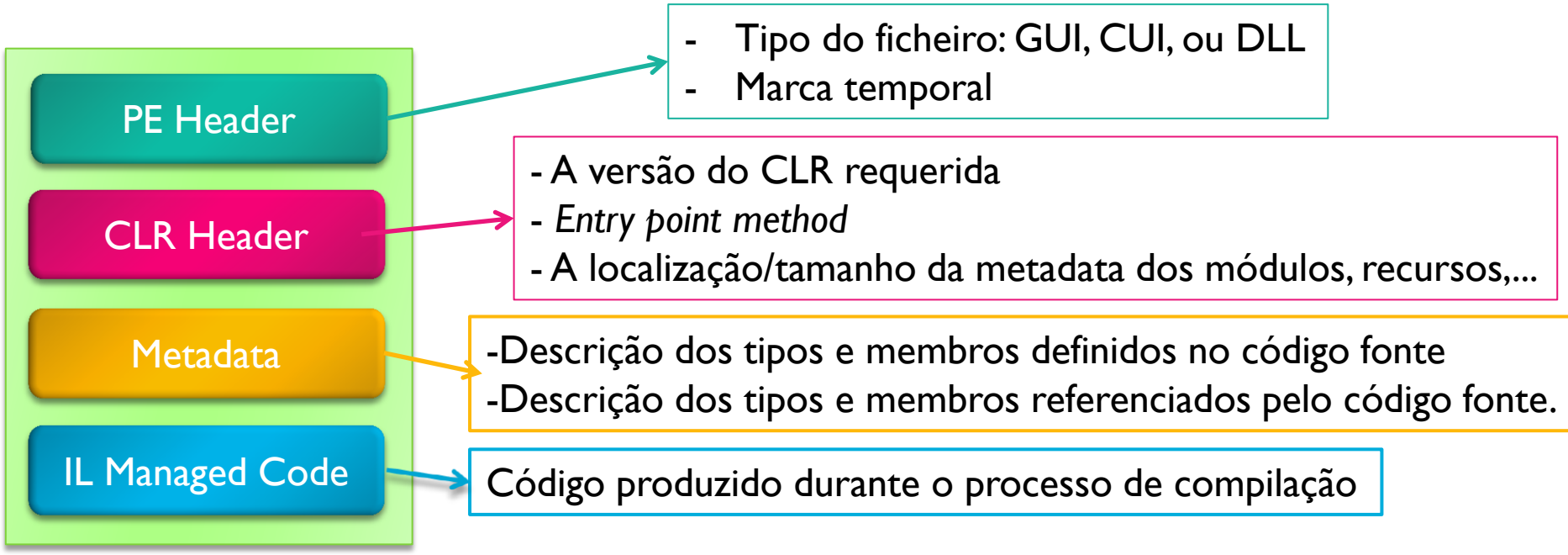
Módulo: Código Intermédio + *Metadata*

- O código fonte é compilado para um módulo *managed*, composto por código intermédio (IL) e metadados



Managed Modules

- Um *managed module* é um ficheiro PE32 ou PE32+ (32-bit ou 64-bit Windows portable executable)
- Requere a CLR para ser executado.



Metadata de um Managed Module

- Conjunto de tabelas com:
  - os tipos definidos no módulo - DefTables;
  - os tipos referênciados (importados) – RefTables;
- Informação sobre tipos
  - **Sempre** incluída no módulo pelo compilador
    - Inseparável do módulo
    - gravada em formato binário
  - Descreve tudo o que existe no módulo :
    - Tipos, classes, métodos, campos, etc.

Código *managed* versus *unmanaged*

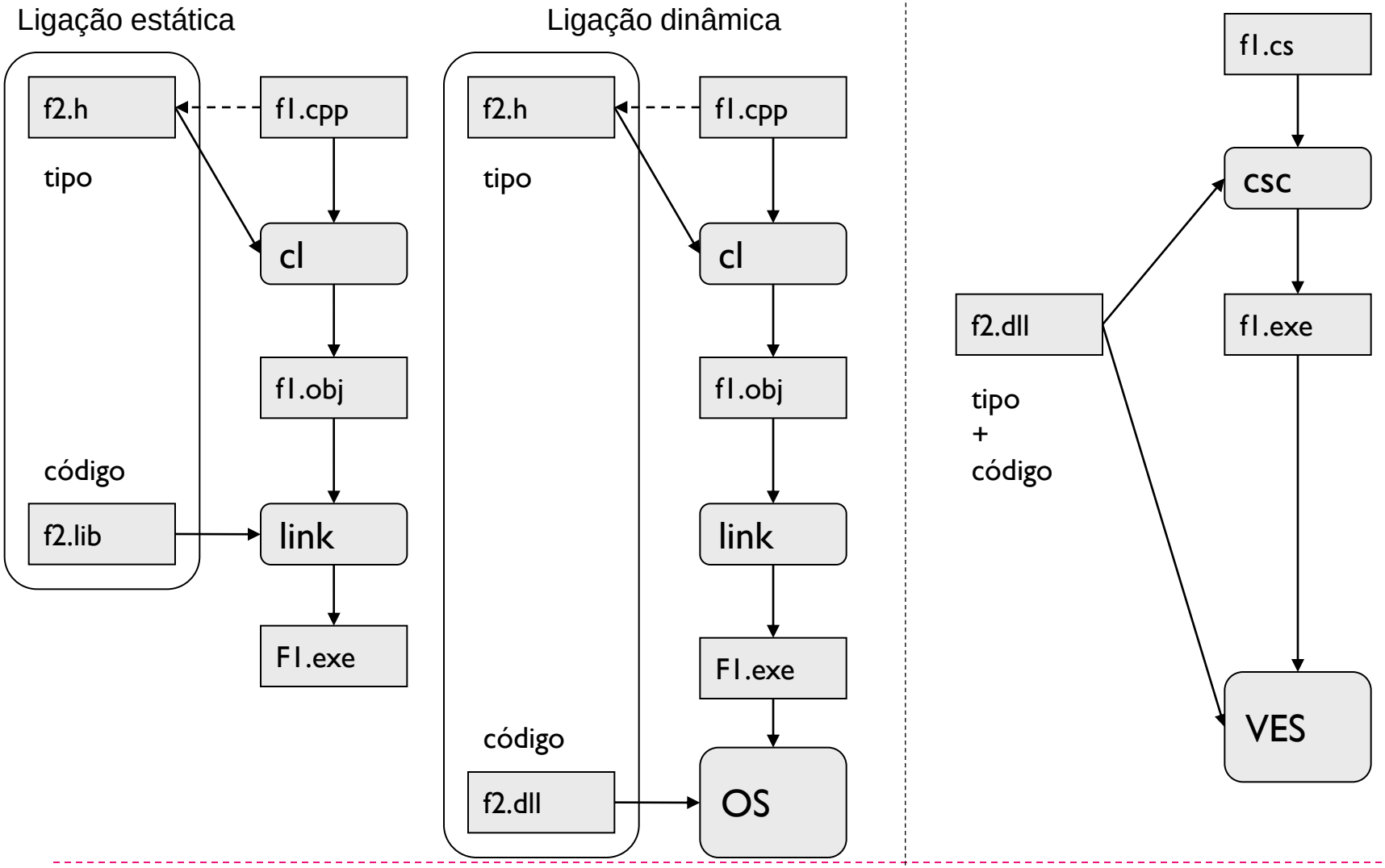
- O código *managed* é executado sob o controlo do CLR.
  - Ex: código escrito em C#, Visual Basic .Net
- O código *unmanaged* é executado independentemente do CLR.
  - Ex: componentes COM, componentes ActiveX, funções da API Win32...

Comparação entre modelos "managed" e "unmanaged"

- Modelo "unmanaged" – Contrato físico
  - Informação de tipo (declarações) presentes no ficheiro *header*
  - Informação de tipo específica da linguagem
  - Implementação (instruções) presentes no ficheiro biblioteca
  - Problema: sincronização entre *header* e biblioteca
  - Instruções na linguagem nativa
  - Normalmente associado à construção de aplicações "monolíticas" – ligação estática

- Modelo "managed" - Contrato lógico
  - Informação de tipo (metadata) e implementação (linguagem intermédia) presentes no mesmo ficheiro
  - A ligação entre componentes é sempre dinâmica

Modelos *managed* e *unmanaged*

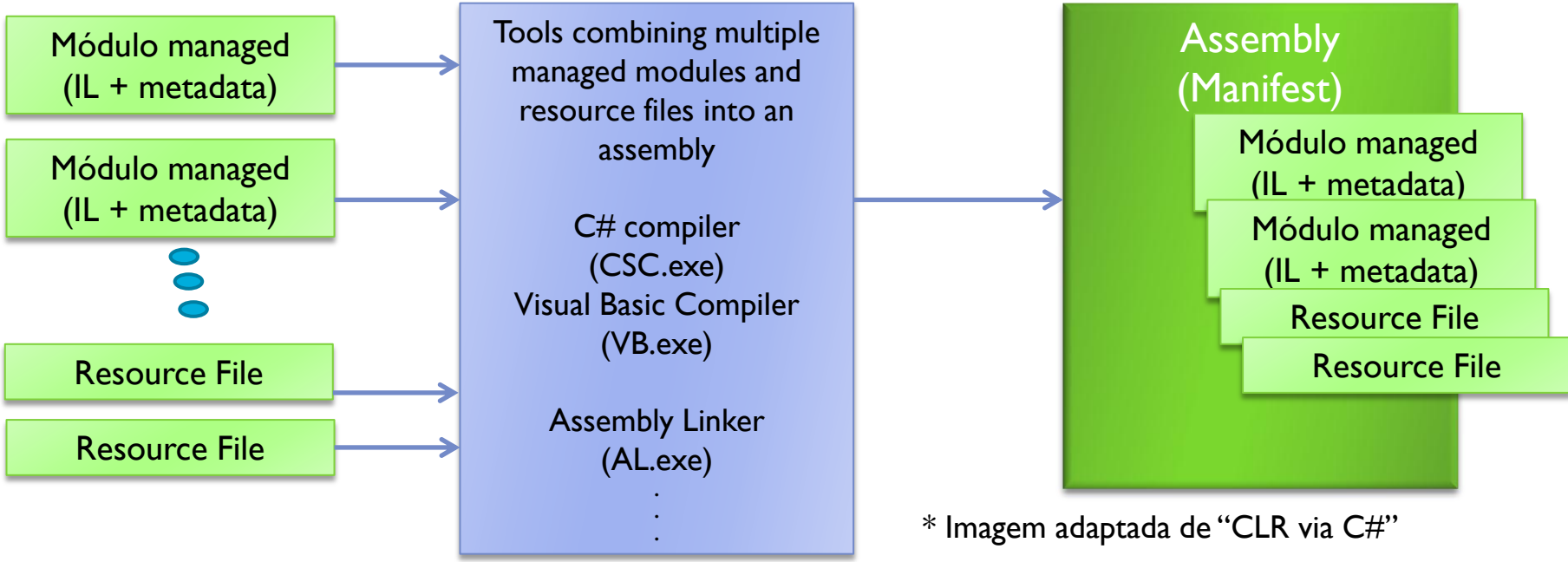


Vantagens da Metadata

- Pelo facto dos compiladores emitirem ao mesmo tempo a *metadata* e o código dos módulos, estas duas entidades nunca podem deixar de estar em sintonia.
  - Dispondo da *metadata* não são necessários os ficheiros *header* e os ficheiros biblioteca para compilar e ligar o código das aplicações. Os compiladores e os *linkers* poderão obter a mesma informação consultando a *metadata* dos *managed modules*.
  - O processo de verificação do código do CLR usa a *metadata* para garantir que o código realiza apenas operações seguras.
  - Usada para suportar os serviços de *runtime*. São exemplos:
    - A *metadata* suporta a serialização/desserialização automática do estado dos objectos
    - Para qualquer objecto, o GC pode determinar o tipo do objecto e, a partir da *metadata*, saber quais os campos desse objecto que são referências para outros objectos.

A unidade de distribuição (componente) em . Net é o Assembly

- Um Assembly é:
  - Um agrupamento lógico de um ou mais *Managed Modules* e *Resource Files*;
  - É a unidade básica de utilização, controle de versões e sujeita a restrições de segurança;
  - Um dos módulos constituintes do Assembly contém um *Manifesto;*



* Imagem adaptada de "CLR via C#"

Assembly

- Por omissão, os compiladores transformam o *managed module* num *assembly*.
  - O *manifest* do *assembly* gerado contém a indicação de que o *assembly* consiste apenas de um ficheiro.
- Para agrupar um conjunto de ficheiros num *assembly* pode ser utilizado o *assembly linker* (AL.exe).
- Cada *assembly* é ou uma aplicação executável (exe) ou uma biblioteca (DLL).
- Um dos módulos constituintes do *assembly* contém obrigatoriamente um manifesto que define os módulos constituintes como um todo inseparável e contém o identificador universal do *assembly*.

Metadata de um Assembly

- Um dos módulos constituintes do Assembly, contém também a *manifest metadata table*.

Informação	Descrição
Nome	String com o nome "amigável" (*friendly name*) do *assembly.* Corresponde ao nome do ficheiro (sem extensão) que inclui o manifesto.
Número versão	*Major, minor, revision* e *build numbers* da versão
Cultura	Localização do *Assembly* (língua, cultura). Usada somente em *assemblies* com resources (*strings*, imagens). Os *assemblies* com a componente *cultura* denominam-se assemblies satélite
Nome criptográfico (*strong* name)	Identifica o fornecedor do componente. É uma chave criptográfica. Garante que não existem 2 assemblies distintos com o mesmo nome e que o *assembly* não foi corrompido
Lista de módulos	Pares (nome, *hash*) de cada módulo pertencente ao assembly
Tipos exportados (Também existe em cada um dos módulos)	Informação usada pelo *runtime* para associar um tipo exportado ao módulo com a sua descrição/implementação
Assemblies referenciados (Também existe em cada um dos módulos)	Lista dos *assemblies* de que o *assembly* depende

Razões para a criação de Assemblies Multi-módulo

- O módulo só é carregado para memória quando é necessário (quando for usado algum tipo exportado no módulo)
- Torna possível a implementação de um *assembly* em mais que uma linguagem
- Separar fisicamente *resources* (imagens, *strings*) do código

HelloWorld in C#

```
using System;
{
    class Program
    {
        static void Main(string[] args)
        {
            string myMessage = "Hello World";
            Console.WriteLine("myMessage");
        }
    }
}
```

Representação intermédia dos membros da classe HelloWord

```
.class private auto ansi beforefieldinit HelloWord.Program
       extends [mscorlib]System.Object
{
  .method private hidebysig static void  Main(string[] args) cil managed
  {
    .entrypoint
    .maxstack  1
    .locals init ([0] string myMessage)
    IL_0000:  nop
    IL_0001:  ldstr      "Hello"
    IL_0006:  stloc.0
    IL_0007:  ldstr      "myMessage"
    IL_000c:  call       void [mscorlib]System.Console::WriteLine(string)
    IL_0011:  nop
    IL_0012:  ret
  } // end of method Program::Main
.method public hidebysig specialname rtspecialname
          instance void  .ctor() cil managed
  {

    .maxstack  8
    IL_0000:  ldarg.0
    IL_0001:  call       instance void [mscorlib]System.Object::.ctor()
    IL_0006:  ret
  } // end of method Program::.ctor

} // end of class HelloWord.Program
```

Definição da classe

Método main

construtor

Representação intermédia dos membros da classe HelloWorld

```
.method private hidebysig static void  Main(string[] args) cil managed
  {
    .entrypoint
    .maxstack  1
    .locals init ([0] string myMessage)
    IL_0000:  nop
    IL_0001:  ldstr      "Hello World"
    IL_0006:  stloc.0
    IL_0007:  ldloc.0
    IL_000c:  call       void [mscorlib]System.Console::WriteLine(string)
    IL_0011:  nop
    IL_0012:  ret
  } // end of method Program::Main
```

Marca este método como o ponto de entrada do executável

Define a variável local string (no índex 0).

Armazena um objecto string na pilha para o literal "Hello"

Retira um valor da pilha para a variável local no índice 0

Carrega a variável local no índice 0 para a pilha

Invoca o método com o valor actual

return

Representação intermédia dos membros da classe HelloWord

```
.method public hidebysig specialname rtspecialname
          instance void  .ctor() cil managed
  {
    // Code size       7 (0x7)
    .maxstack  8
    IL_0000:  ldarg.0
    IL_0001:  call       instance void [mscorlib]System.Object::.ctor()
    IL_0006:  ret
  } // end of method Program::.ctor

} // end of class HelloWord.Program
```

Assegura que não existem dois métodos numa classe derivada que tenham a mesma assinatura quando interpretada.

Carregar o argumento 0 na pilha.

Invocação do construtor da classe basse, System.Object

Compilar na linha de comandos

- `csc /help` - indica todas as opções
- `csc MyFile.cs` – compila o ficheiro `MyFile`, produzindo o `MyLine.exe`
- `csc /target:library MyFile.cs` – compila o ficheiro `MyFile`, produzindo `MyLine.dll`
- `csc /out:My.exe MyFile.cs` – compila o ficheiro `MyFile`, produzindo o `My.exe`

- Nota: colocar na *path* a directoria onde está o csc.exe
  - Ex: C:\Windows\Microsoft.NET\Framework\v4.0.30319

ILDasm Visualização de IL

- `ildasm MyFile.exe` – para visualizar o código intermédio produzido por csc MyFile.cs
- `ildasm /out=My.il MyFile.exe` – output direccionado para um ficheiro em vez de ser para GUI

- Nota: colocar na *path* a directoria onde está o ildasm.exe
  - Ex: C:\Program Files\Microsoft SDKs\Windows\v7.0A\bin

Demo 1:
ILDasm – Visualização de IL

Exemplos\Aula2\Hello.cs

Demo 2:
ILasm – Assemblador de IL

Exemplos\Aula2\Hello.il

SumToTen.cs

```
using System;
    class SumToTen
    {
        static void Main(string[] args)
        {
           int sum=5;
           for(int i=1;i<=10;i++)
                 sum+=i;

             Console.WriteLine(sum);
        }
}
```

O método Main em IL

```
.method private hidebysig static void  Main(string[] args) cil managed
{
  .entrypoint
  // Code size       35 (0x23)
  .maxstack  2
  .locals init (int32 V_0,  int32 V_1,  bool V_2)
  IL_0000:  nop
  IL_0001:  ldc.i4.5
  IL_0002:  stloc.0
  IL_0003:  ldc.i4.1
  IL_0004:  stloc.1
  IL_0005:  br.s       IL_000f
  IL_0007:  ldloc.0
  IL_0008:  ldloc.1
  IL_0009:  add
  IL_000a:  stloc.0
  IL_000b:  ldloc.1
  IL_000c:  ldc.i4.1
  IL_000d:  add
  IL_000e:  stloc.1
  IL_000f:  ldloc.1
  IL_0010:  ldc.i4.s   10
  IL_0012:  cgt
  IL_0014:  ldc.i4.0
  IL_0015:  ceq
  IL_0017:  stloc.2
  IL_0018:  ldloc.2
  IL_0019:  brtrue.s   IL_0007

  IL_001b:  ldloc.0
  IL_001c:  call   void [mscorlib]System.Console::WriteLine(int32)
  IL_0021:  nop
  IL_0022:  ret
}
```

Coloca o 5 na stack

Coloca 1 na pilha se valor1>valor2, caso contrário coloca 0

Coloca 1 na pilha se valor1 = valor2, caso contrário coloca 0

Vai para IL_0007 se o valor é diferente de 0 (true)

Storage.cs

```
using System;

    class Storage
    {
        static void Main()
        {
                        int[] array=new int[10];
                        for(int i=0;i<10;i++)
                                array[i]=i;
        }

}
```

O método Main em IL

```
.method private hidebysig static void  Main() cil managed
{
  .entrypoint
  // Code size       31 (0x1f)
  .maxstack  3
  .locals init (int32[] V_0,
           int32 V_1,
           bool V_2)
  IL_0000:  nop
  IL_0001:  ldc.i4.s   10
  IL_0003:  newarr     [mscorlib]System.Int32
  IL_0008:  stloc.0
  IL_0009:  ldc.i4.0
  IL_000a:  stloc.1
  IL_000b:  br.s       IL_0015

  IL_000d:  ldloc.0
  IL_000e:  ldloc.1
  IL_000f:  ldloc.1
  IL_0010:  stelem.i4
  IL_0011:  ldloc.1
  IL_0012:  ldc.i4.1
  IL_0013:  add
  IL_0014:  stloc.1
  IL_0015:  ldloc.1
  IL_0016:  ldc.i4.s   10
  IL_0018:  clt
  IL_001a:  stloc.2
  IL_001b:  ldloc.2
  IL_001c:  brtrue.s   IL_000d
  IL_001e:  ret
} // end of method Storage::Main
```

Substitui o elemento do array no índice actual com o valor da pilha

UmaClasse.cs

- Considere-se a seguinte classe em C#

```
class UmaClasse {
  int umCampo;
  int UmMetodo(int arg1) {
   return arg1 + umCampo;
  }
}
```

Representação intermédia dos membros da classe UmaClasse

```
.class private auto ansi beforefieldinit UmaClasse
      extends [mscorlib]System.Object  {
  .field private int32 umCampo
  .method private hidebysig
        instance int32  UmMetodo(int32 arg1) cil managed  {
    .maxstack  2
    .locals init (int32 V_0)
    IL_0000: ldarg.1
    IL_0001: ldarg.0
    IL_0002: ldfld      int32 UmaClasse::umCampo
    IL_0007: add
    IL_0008: stloc.0
    IL_0009: br.s       IL_000b

    IL_000b: ldloc.0
    IL_000c: ret
  } // end of method UmaClasse::UmMetodo
```

UmaClasse é uma classe completamente caracterizada (visibilidade, herança, ...)

umCampo é um campo completamente caracterizado (visibilidade, tipo, nome)

UmMetodo é um metodo completamente caracterizado (visibilidade, tipos, ...)

Carregar primeiro argumento

Carregar **this**

Carregar **this.umCampo**

Representação intermédia: propriedades

- Definição completa da classe **UmaClasse**
  - ".**class** private auto ansi beforefieldinit **UmaClasse** extends [mscorlib]System.Object"
  - Visibilidade, nome, herança, ...
- Definição completa do campo **umCampo**
  - ".field private int32 umCampo"
  - Visibilidade, tipo, nome
- Definição completa do método **UmMetodo**
  - **".method** private hidebysig instance int32 **UmMetodo**(int32 arg1) cil managed"
  - Visibilidade, tipo de retorno, nome, argumentos, ...
- O código utiliza nomes e não localizações em memória
  - "ldarg.1" – Carregar o *primeiro* operando (e não [ebp+4])
  - "ldfld int32 UmaClasse::umCampo" – Carregar o campo umCampo (e não [this+0])

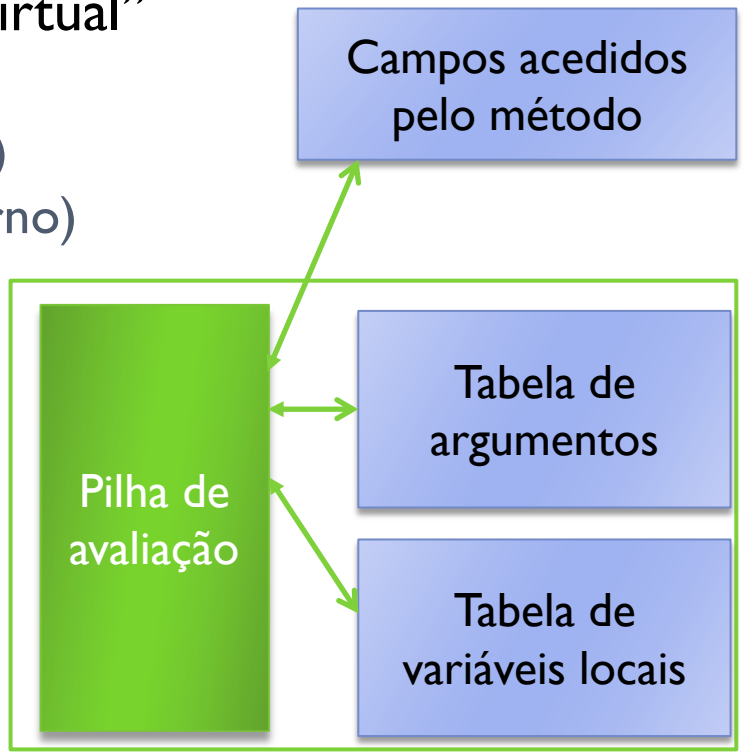
A linguagem intermédia (IL)

- A linguagem IL é *stack based* (execução de uma máquina de *stack*)
  - Todas as instruções empilham os operandos (push) no *stack* e obtêm os resultados do topo do *stack* (pop)
- Não existem instruções para manipulação de registos
  - O que implica uma diminuição significativa no número de instruções necessárias
- Todas as instruções são polimórficas (*dependendo do tipo dos operandos podem ter sucesso ou não, gerando, em caso de insucesso, uma excepção ou falhando a fase de verificação, se existir*).
  - Ex: add - adiciona os dois operandos presentes no topo do *stack* e retorna o resultado no topo do *stack*

IL: Modelo de execução

- Conjunto de instruções duma "máquina virtual"
- Quatro espaços de endereçamento
  - Tabela de argumentos do método (interno)
  - Tabela de variáveis locais do método (interno)
  - Campos acedidos pelo método (externo)
  - *Stack* de avaliação
- Formas de endereçamento
  - Tabelas de argumentos e variáveis – índice
    - **ldarg.1**
    - **stloc.0**
  - Campos – object + id. do campo
    - **ldfld int32 UmaClasse::umCampo**

      (o objecto referido é o do topo do *stack* de avaliação)
- *Stack* – relativo ao topo
  - **add**



Exemplo: Point0.cs

```
public class Point0 {
      private int x, y;
      public Point0(int x, int y) {
            this.x = x;
            this.y = y;
      }
}
```

Exemplo: Point0.cs

```
.class public auto ansi beforefieldinit Point0
       extends [mscorlib]System.Object
{
  .field private int32 x
  .field private int32 y
  .method public hidebysig specialname rtspecialname
          instance void  .ctor(int32 x,
                               int32 y) cil managed
  {
  .maxstack  8
    IL_0000:  ldarg.0
    IL_0001:  call       instance void [mscorlib]System.Object::.ctor()
    IL_0006:  nop
    IL_0007:  nop
    IL_0008:  ldarg.0
    IL_0009:  ldarg.1
    IL_000a:  stfld      int32 Point0::x
    IL_000f:  ldarg.0
    IL_0010:  ldarg.2
    IL_0011:  stfld      int32 Point0::y
    IL_0016:  nop
    IL_0017:  ret
  } // end of method Point0::.ctor
} // end of class Point0
```

IL: Suporte para POO

- Inclusão de instruções para o suporte ao paradigma da
- "orientação por objectos"
  - Noção de campo de objecto
    - **ldfld** e **stfld**
  - Chamada a métodos
    - **callvirt, call**
  - Criação de instâncias
    - **newobj**, **initobj**
  - *Casting*
    - **castclass**, **isinst**
  - Excepções
    - **throw**, **rethrow**