

# Virtual Execution Environments

2021

Week 1

Components and Metadata

# Objectivos?

Respostas de alunos:

- C#

NÃO é o Foco

VB simplicidade  
C++ mais funcionalidade  
C#

- IL (Intermediate Language ↔ bytecodes no Java)

- Ambientes Virtuais

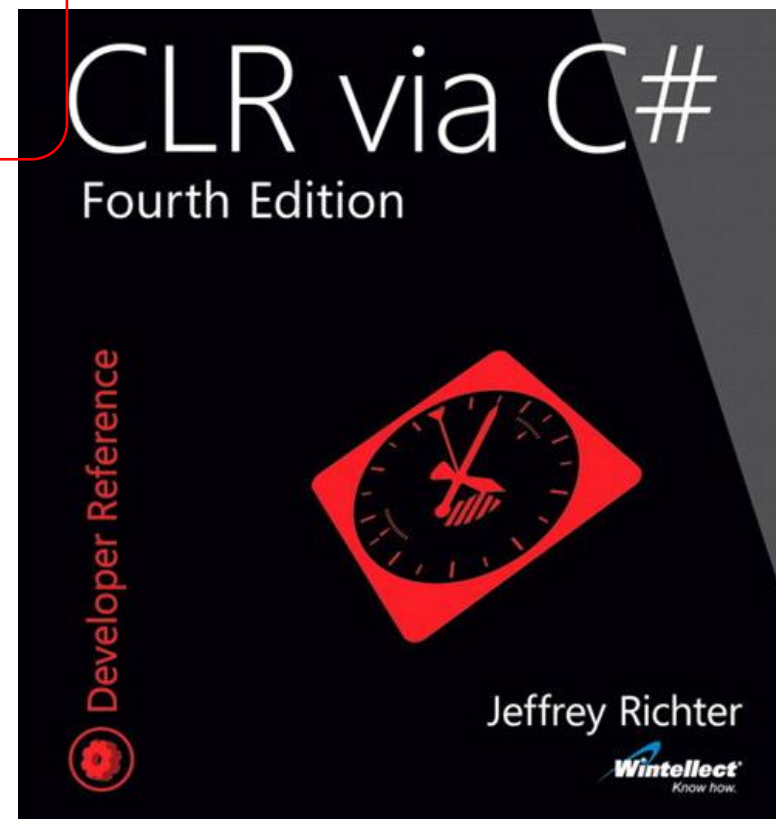
- ~~Async?~~

- Reflexão / Introspeção

- Sw development

- Mem management

- Compilador



# CLR ???

Common Language Runtime ??? => VM no .Net

Virtual Machine / Runtime ↔ JVM (Java Virtual Machine)

## Virtual Execution Environment

### Princípios do .Net <=> JVM:

- Gestão memória (i.e. Garbage Collector)
- Reflection
- Exceptions
- Type Safety
- Dynamic link
- ...

# O que é o Java?

Respostas dos alunos:

- AVE ou **linguagem?**
- **Linguagem** de programação
- *“código é relativamente agnóstico ao sistema operativo”*
- **linguagem** de programação de alto nível
- **linguagem** focada em object-oriented
- *“Java pode correr em todas as plataformas que suportam java sem recompilar”*

Outras linguagens para a JVM: Scala, Clojure, Ruby, Kotlin, etc...

# Porque surge um AVE como o Java? (1993)

## Respostas de alunos

- *“facilitar a compatibilidade de programas para **varias maquinas?**”*
  - O mesmo **componente** (e.g. Point.class) pode ser executado noutra “máquina” (e.g. SO)
- 1 distribuição para várias Arquitecturas.

Componente = Peça de Software

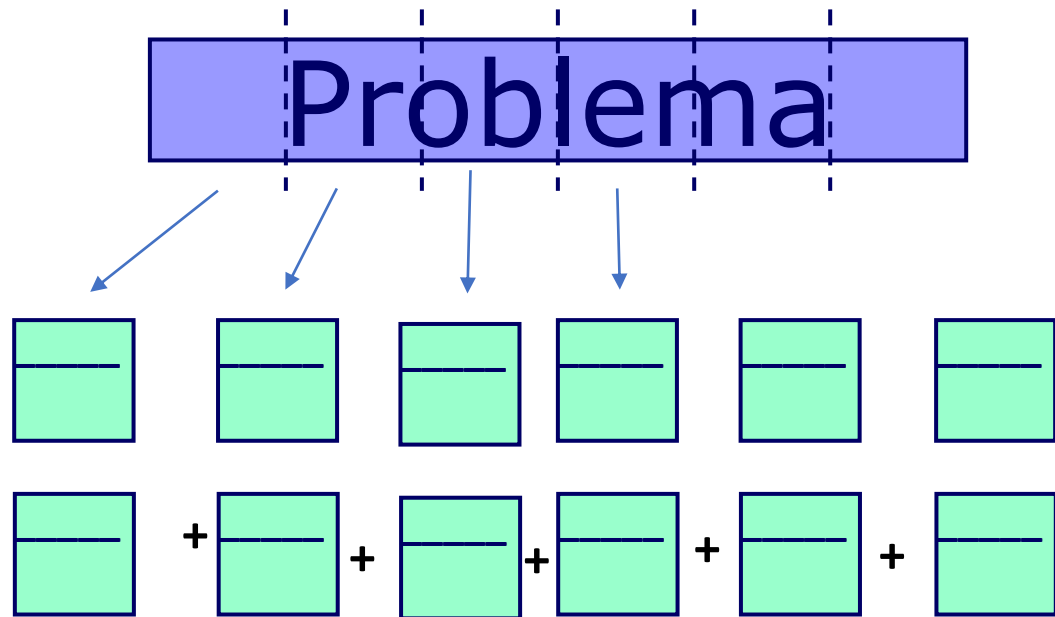
**COMPONENTE = IL + metadata**

=> Promove Reutilização

# Componentes?

1. Reutilização
2. Divisão de trabalho
3. Divisão de Complexidade != oposição monolítica

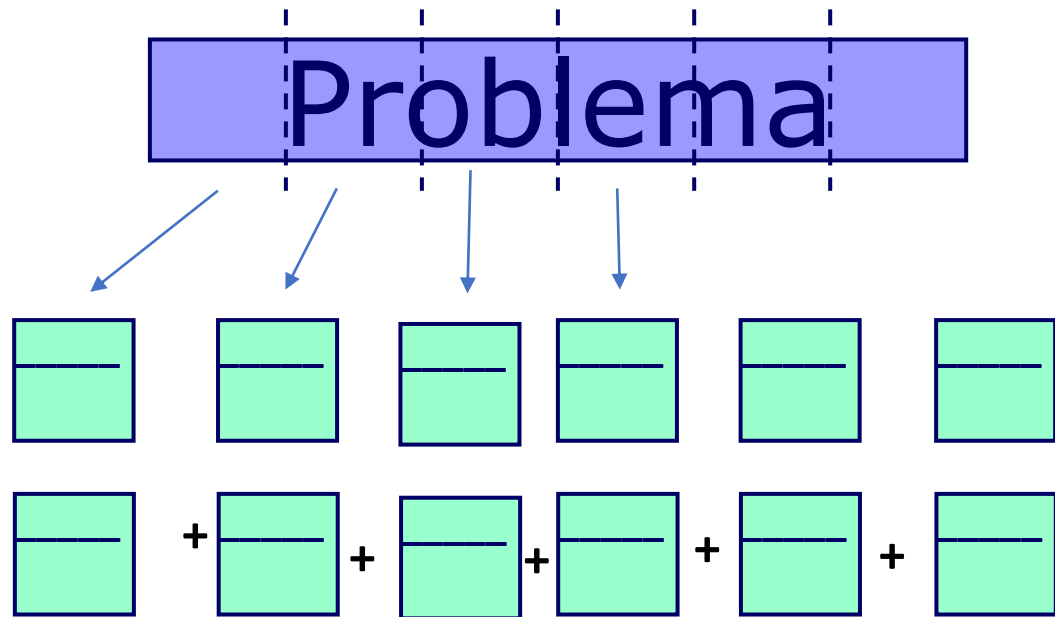
**METADATA**



# Componentes?

1. Reutilização
2. Divisão de trabalho
3. Divisão de Complexidade != oposição monolítica

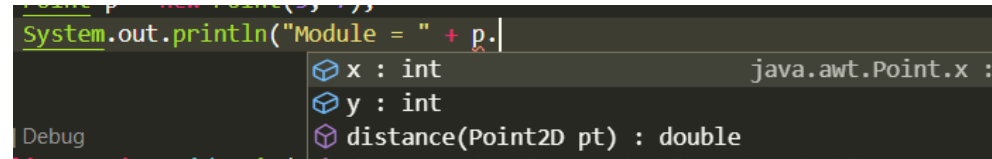
**METADATA**





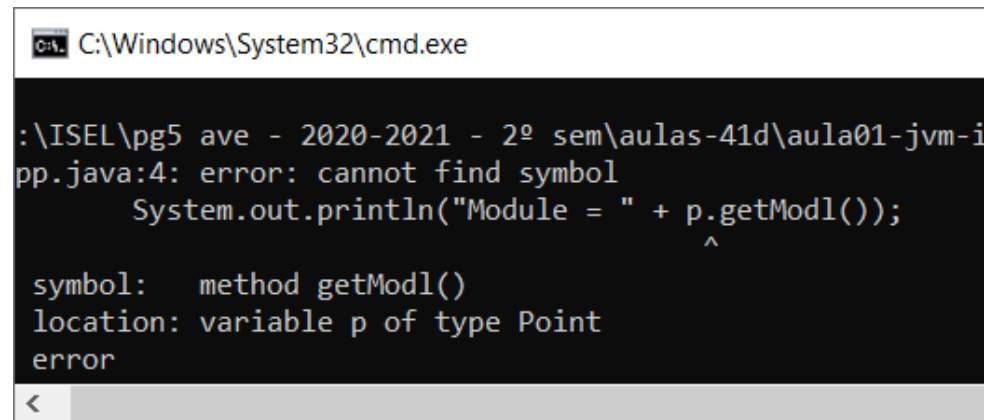
# Exemplos de utilização de metadata

- Intellisense no IDE



A screenshot of an IDE showing a code completion popup for the `Point` class. The popup lists the following members: `x : int` (with a reference to `java.awt.Point.x`), `y : int`, and `distance(Point2D pt) : double`. The background code shows `System.out.println("Module = " + p.` with the cursor at the end of the line.

- Compilador



A screenshot of a Windows command prompt window titled `C:\Windows\System32\cmd.exe`. It shows the output of a Java compilation command. The error message is: `pp.java:4: error: cannot find symbol`, followed by `System.out.println("Module = " + p.getModl());` with a caret under `getModl()`. Below the error, it says: `symbol: method getModl()`, `location: variable p of type Point`, and `error`.

- Leitura em tempo de execução, i.e. Reflexão

# Linguagens

programador

Abstracção

- Linguagens de alto nível: Java, C#
- Representação intermédia (e.g. bytecodes, IL, )
- Nativo , assembly

CPU

# Ligação

## JVM

- App.class
- Point.class

## Alunos:

- “ligação implícita”
- **Ligação dinâmica**, em tempo de execução.

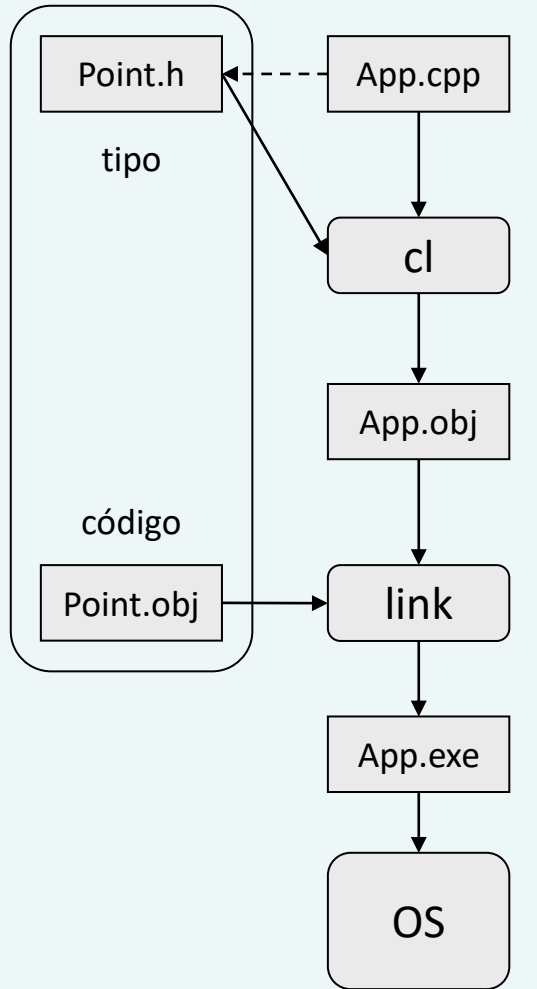
## C++ unmanaged:

- App.obj
- Point.obj

## Alunos:

- “*une o conteúdo dos obj*”
- ... mais as libs
- Junta num executável
- **Ligação estática**, em tempo de compilação.

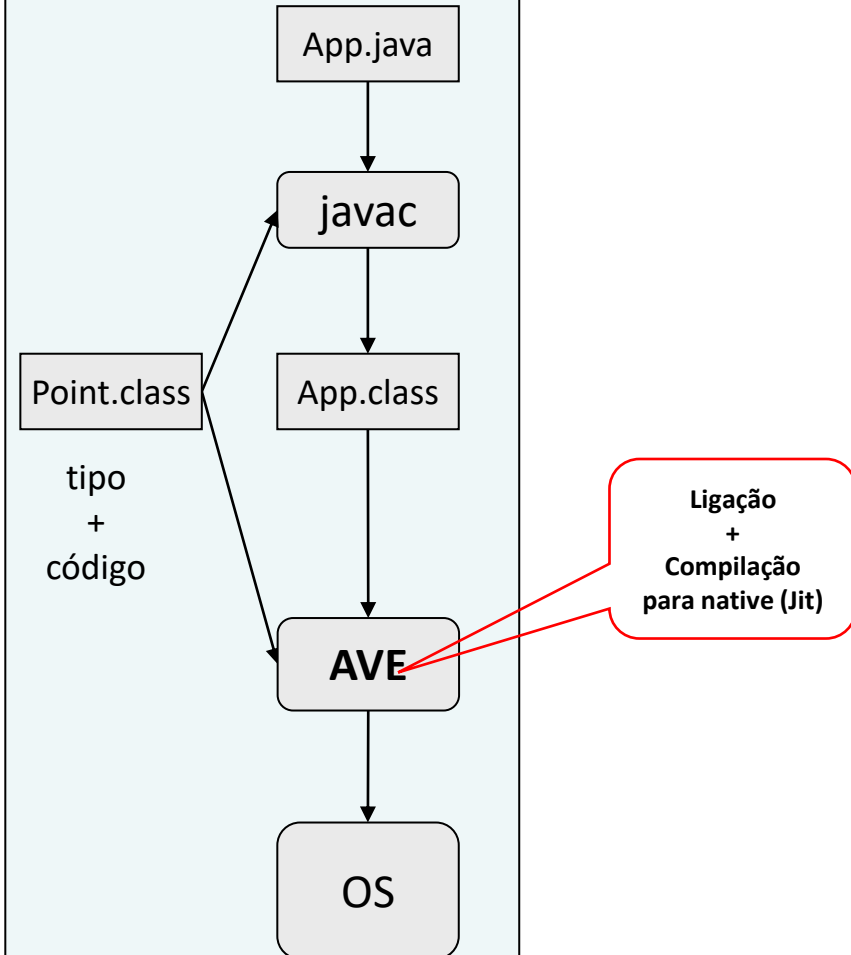
### Ligação estática: (em unmanaged)



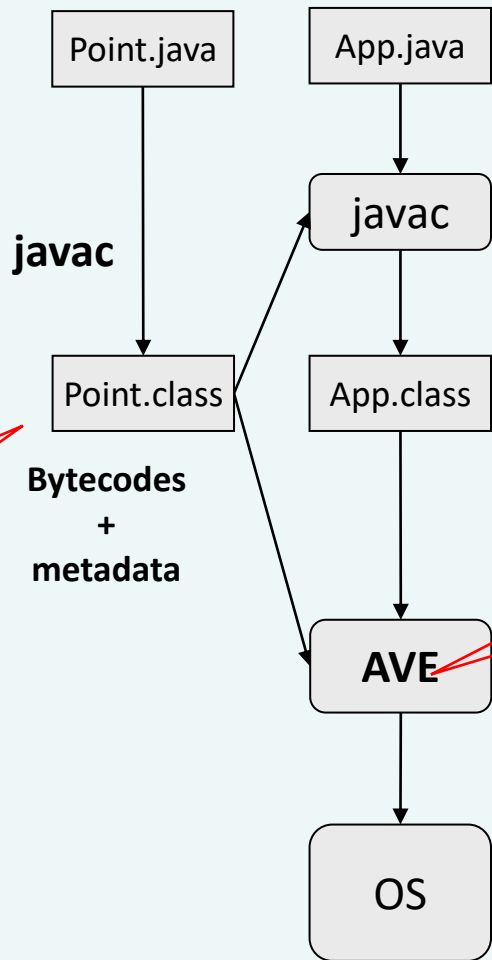
### Ligação dinâmica: (unmanaged)

...

### Ligação dinâmica: (managed)

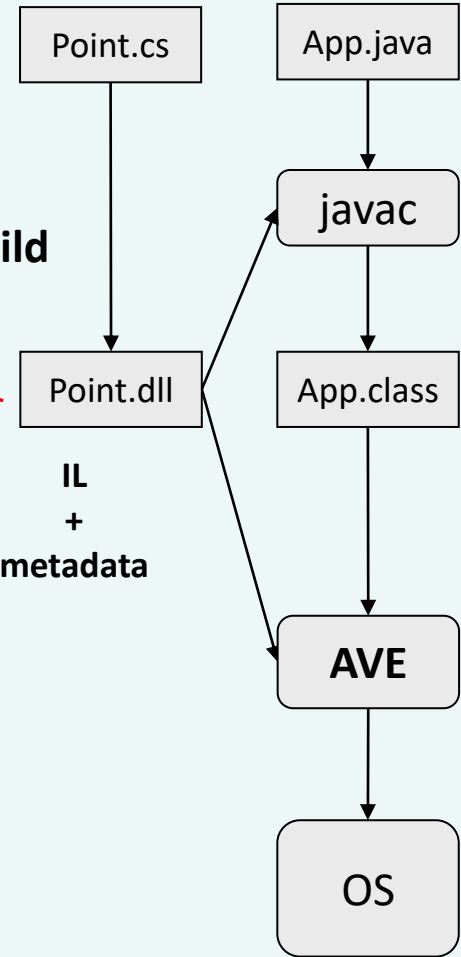


### Java:



1 tipo

### dotnet:



\* tipos

dotnet build

Ligação + Compilação para native (Jit)

# Componentes e.g. .class, .dll

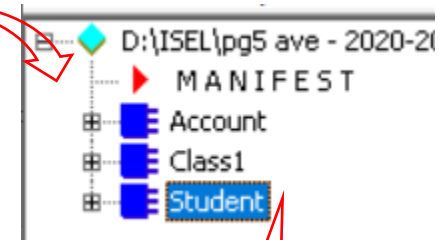
```
App.java x
: > ISEL > pg5 ave - 2020-2021 - 2º sem > aula
1 > public class App { ...
11 |
12 | class Student {}
13 | class Account {}
```

```
Class1.cs x
: > pg5 ave - 2020-2021 - 2º sem
public class Class1 {}
class Student {}
class Account {}
```

```
184 Account.class
1 009 App.class
294 App.java
META-INF
891 Point.class
91 Point.kt
250 README.md
184 Student.class
```

```
416 PointLib.deps.json
4 096 PointLib.dll
9 348 PointLib.pdb
```

ildasm



```
IL_0000: ldarg.0
IL_0001: call instance void [System.Runtime]System.Object::.ctor()
IL_0006: nop
IL_0007: ret
```

Javap -c

metadata

bytecodes

metadata

IL

```
Compiled from "App.java"
class Student {
  public Student();
  Code:
    0: aload_0
    1: invokespecial #1 // Method java/lang/Object."<init>":()V
    4: return
}
```

# Dependências

```
javap -cp . App.Java
```

**Classpath – caminho para outros componentes**

```
EL > pg5 ave - 2020-2021 - 2º sem > aulas-41d > aula03-dotnet-dy
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net5.0</TargetFramework>
  </PropertyGroup>
  <ItemGroup>
    <Reference Include="point">
      <HintPath>PointLib.dll</HintPath>
    </Reference>
  </ItemGroup>
</Project>
```

**Reference to PointLib.dll**

# Unmanaged demo

- `cl /c /Zi`
- `link /DEBUG`
- `devenv App.exe`



# History

Sun Java (1995) -> Oracle -> HotSpot -> (Proprietary Oracle)  
-> Open JDK (open source)  
(2007)

Visual J++ (1996) -> Ms .Net Framework (2001) -----> .Net  
-> Dotnet Core (open source) /  
(2016)  
-> Mono .Net (Linux)  
(2004)

# Tecnologias baseadas em componentes

- 1986: Objective-C by Brad Cox
- 1990: System Object Model (SOM) da IBM
- 1990: Object Linking and Embedding (OLE) da Microsoft
- 1993: Component Object Model (COM) da Microsoft
  
- 1996: Java Development Kit (JDK) 1.0.

Introduz componentes **auto descritos**.

O programador é obrigado a descrever o componente em IDL, ou numa TLB, ou .h.

**Dificuldades:**

- **Não existe um vínculo** entre o **componente** e o seu contracto (IDL ou TLB).
- **Componentes em código nativo** compatíveis com uma arquitectura.
- Diferentes compiladores podem usar diferentes convenções de chamada por omissão.

# AVE

- Portabilidade;
- Interoperabilidade:
  - entre linguagens;
  - entre módulos de *software* (serviços e aplicações).
- Serviços:
  - Gestão automática de memória (***garbage collection***);
  - Segurança – ***type safety***, controle de acessos e isolamento;
  - Ligação dinâmica;
  - Exceções;
  - AppDomains.
  - ...
- Funcionalidades:
  - IO, contentores, *networking*, entre outras

# Componente

- **auto descritos – metadata**

- Independente da linguagem de programação usada no desenvolvimento do componente (C#, C++, etc).
- Substitui os ficheiros *header* – .h – do modelo *unmanaged* ou IDLs e TLB do COM e DCOM.

- **instruções em linguagem intermédia**

- **CIL** – *common intermediate language*;
- Independente da arquitectura do processador;
- Traduzida em tempo de execução.