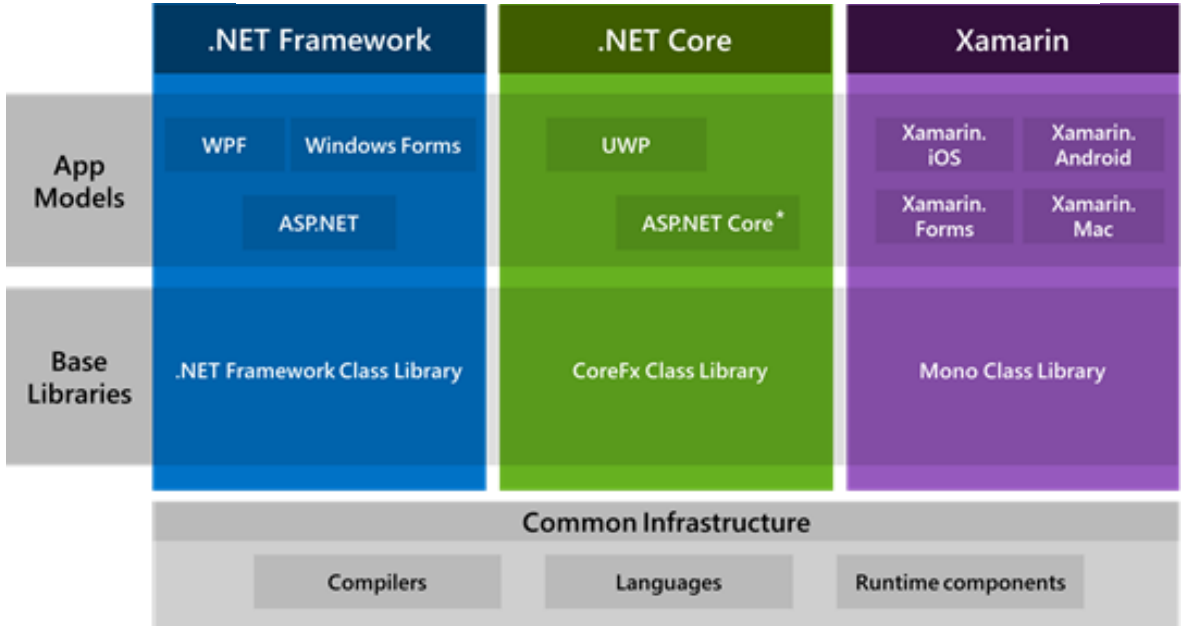
.NET, Assembly a Reflexe

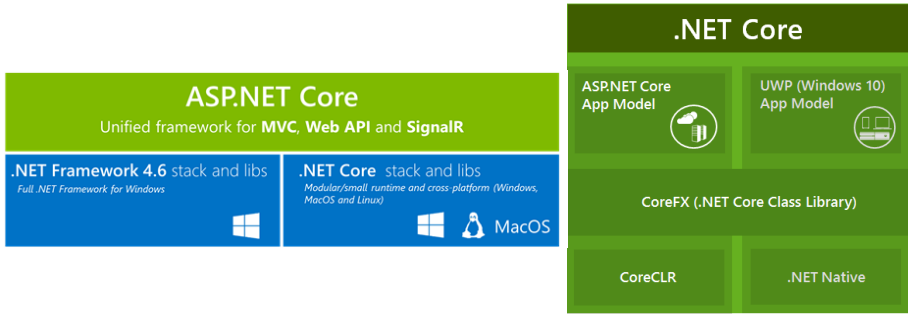
Ing. Michal Radecký, Ph.D.
Ing. Jan Janoušek

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Platforma .NET

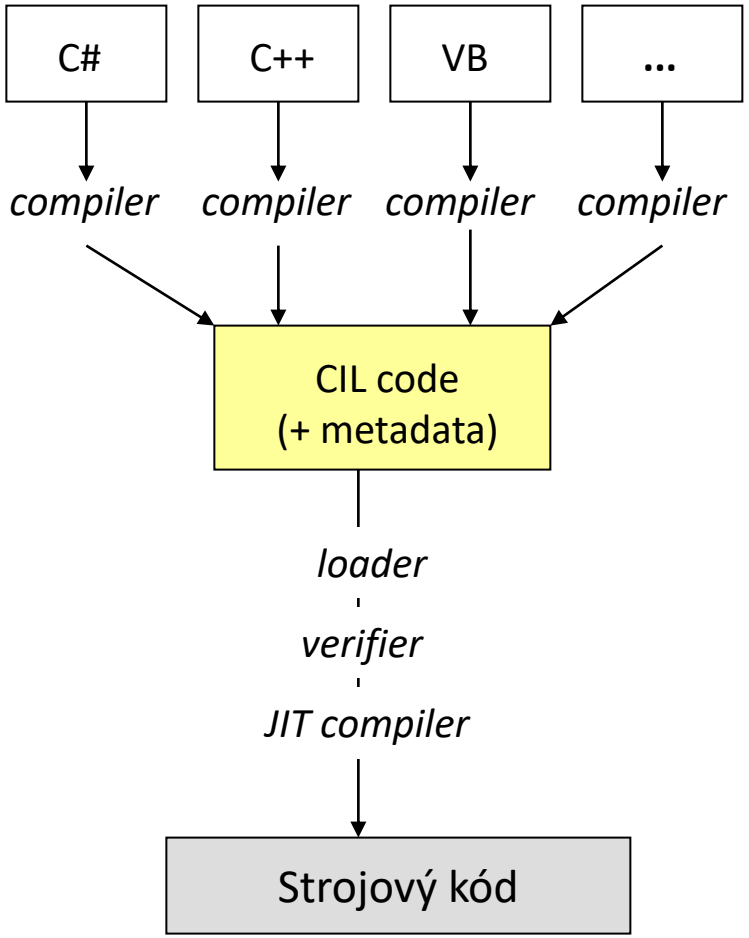




Programovací jazyky: **C#, F#, Visual Basic**

Zdroj: https://devblogs.microsoft.com/cesardelatorre/net-core-1-0-net-framework-xamarin-the-whatand-when-to-use-it/

Interoperabilita díky CIL (Common Intermediate Language)




```
if (a > b) max = a; else max = b;
```

CIL

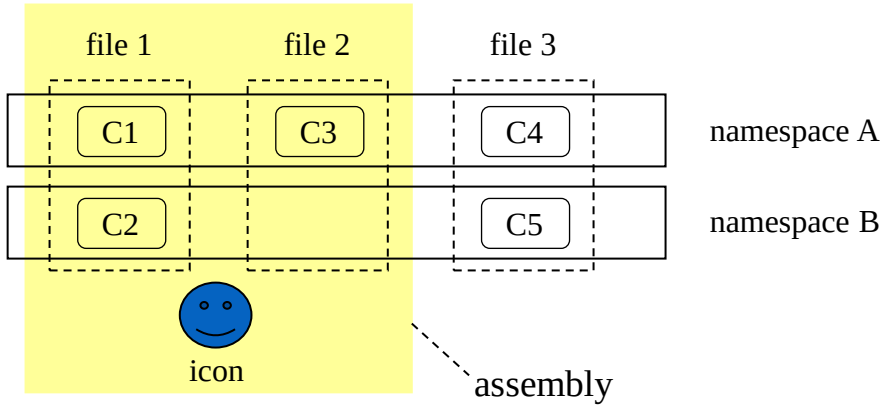
```
IL_0004:  ldloc.0
IL_0005:  ldloc.1
IL_0006:  ble.s     IL_000c
IL_0008:  ldloc.0
IL_0009:  stloc.2
IL_000a:  br.s      IL_000e
IL_000c:  ldloc.1
IL_000d:  stloc.2
```

Intel code

```
mov  ebx,[-4]
mov  edx,[-8]
cmp  ebx,edx
jle  17
mov  ebx,[-4]
mov  [-12],ebx
...
```

Assembly

- Assembly je soubor, který vznikne kompilací .NET aplikace.
- Obsahuje především zkompilovaný kód, ale **může obsahovat i další zdroje** (obrázky, audio, atd…).

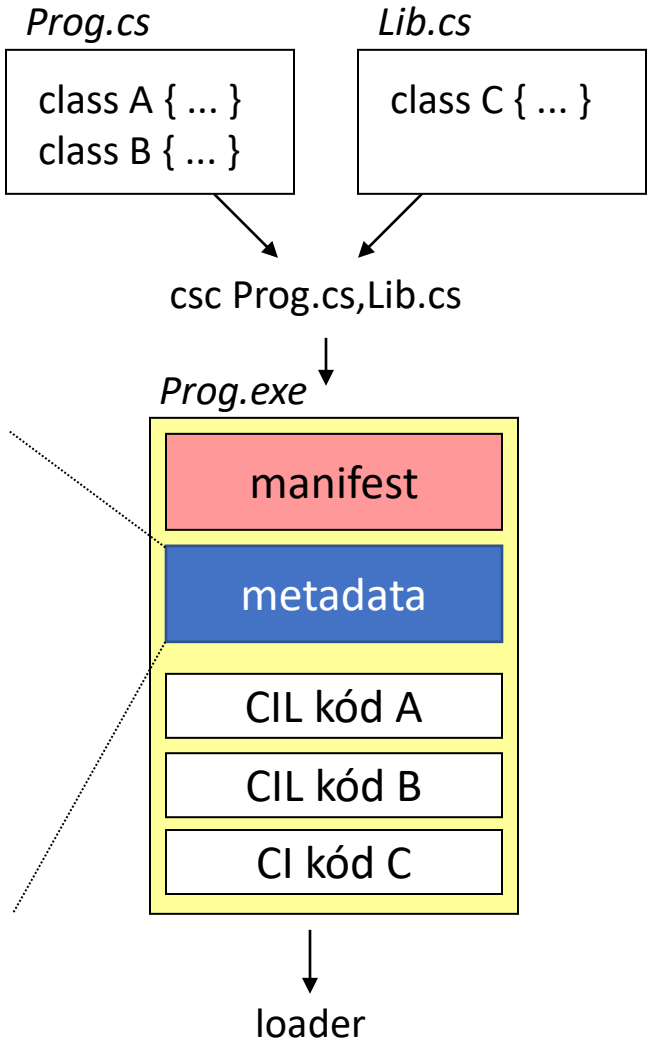


- Nejčastěji platí: 1 assembly = 1 namespace = 1 program.
- Obecně:
  - 1 asembly může obsahovat více namespace.
  - 1 namespace může být součástí více assembly.
  - Assembly může obsahovat více souborů.

Assembly

Assembly jsou nejmenší jednotkou pro

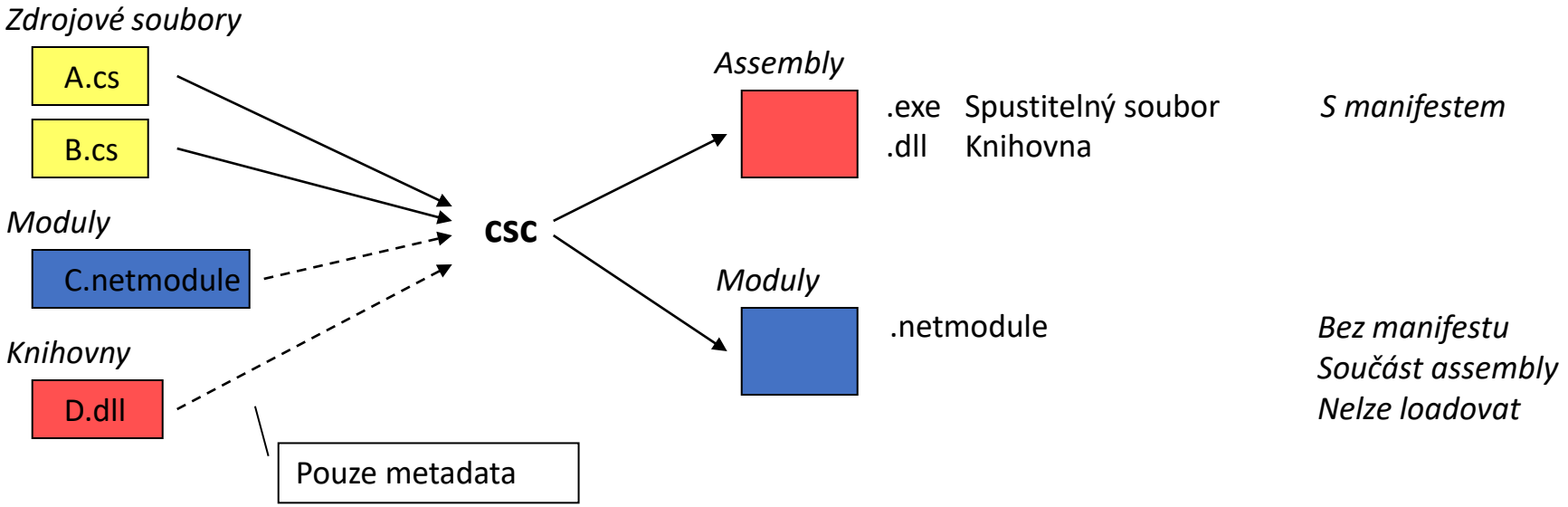
- nasazení
- verzování
- dynamické načítání



Účel metadat:
- Dynamické načítání
- Verzování
- **Reflexe**

Číslo verze
Veřejný klíč
Popis rozhraní:
- třídy
- metody
- vlastnosti
- parametry
- typy
- ...

Jak assembly vznikne?



Načítání knihoven

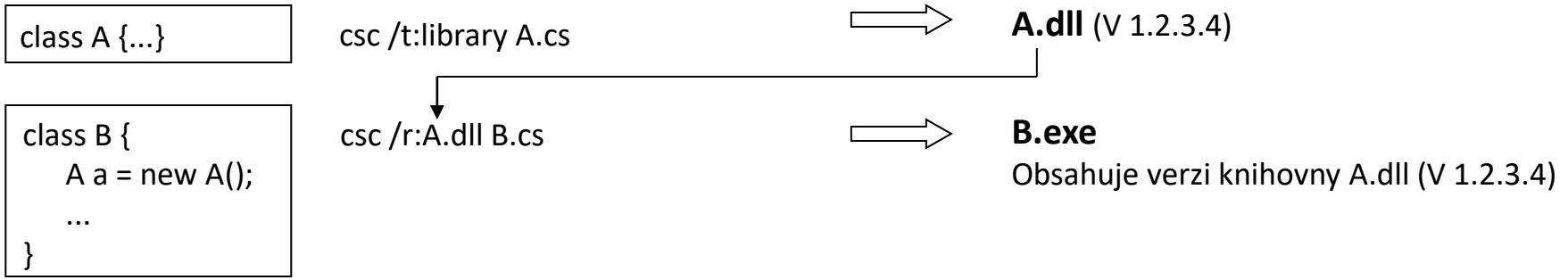
- Knihovny jsou načítány za běhu aplikace.
- K načtení dojde až v okamžiku, kdy je knihovna potřeba.
- Běhové prostředí hledá knihovny:
  - V adresáři s aplikací
  - Ve všech adresářích, které jsou specifikovány v rámci konfiguračního XML souboru (například: *MyApp*.exe.config) v rámci uzlu „probing“:

```
<configuration>
        ...
        <runtime>
            ...
            <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
                <probing privatePath="bin;bin2\subbin;bin3"/>
            </assemblyBinding>
        </runtime>
</configuration>
```

  - V **Global Assembly Cache** (GAC).

Verzování assembly

- U každé assembly lze definovat její verzi.
- V rámci systému může existovat více stejných knihoven v různých verzích.
- Verze používaných knihoven je součástí assembly.



- Verze je kontrolována při načítání assembly.
  - Načte se B.exe
  - Vyhledá se knihovna A.dll.
  - Zkontroluje se verze knihovny. Pokud není nalezena dojde k chybě. V systému může existovat více knihoven A.dll, ale použije se pouze ta s požadovanou verzí! (není potřeba řešit „DLL hell“)
- Standard pro verzování: Major.Minor.Build.Revision (například: 1.5.0.12)

Podepisování assembly

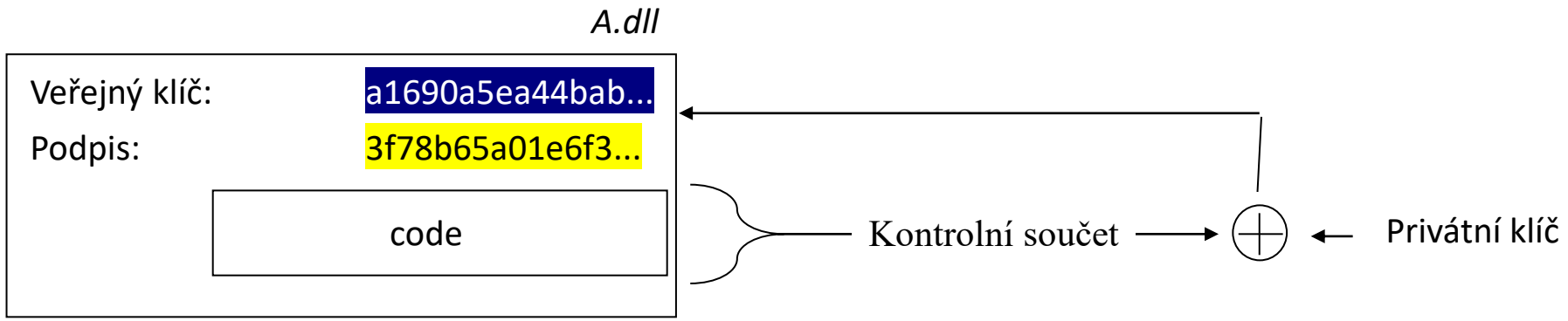
- Využívá se Asymetrická kryptografie (soukromý + veřejný klíč).
- Zajišťuje důvěryhodnost kódu.
- Vygenerovaný kód se podepíše pomocí privátního klíče.
- Podpis a veřejný klíč je součástí assembly.
- Před načtením podepsané knihovny dochází k ověření podpisu.
- Klíč pro podpis se definuje v rámci kódu pomocí tzv. atributů.

```
[assembly:AssemblyKeyFile("myKeyFile.snk")]
```

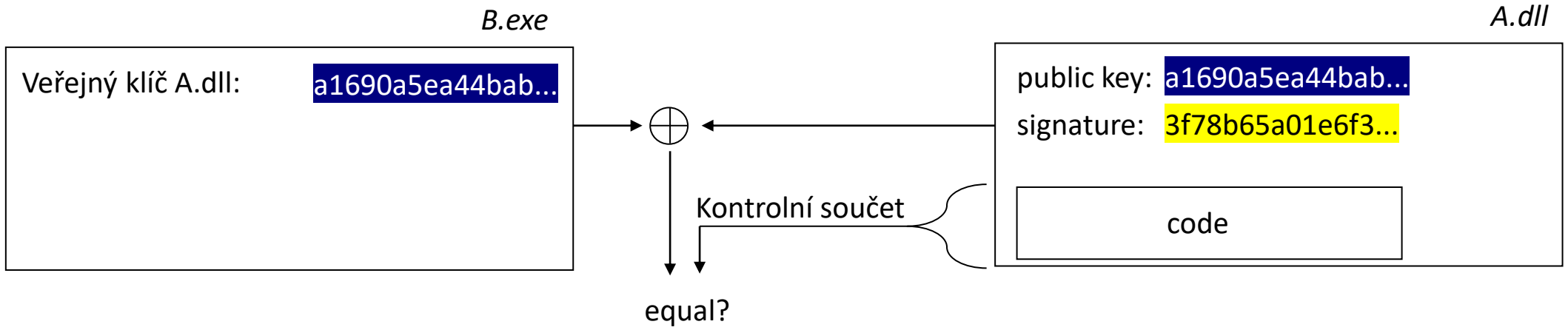
- Podepsaný kód nelze zaměnit za jiný – ochrana proti hackování, atd.

Podepisování assembly

Podepisování



Ověřování



Strong Name

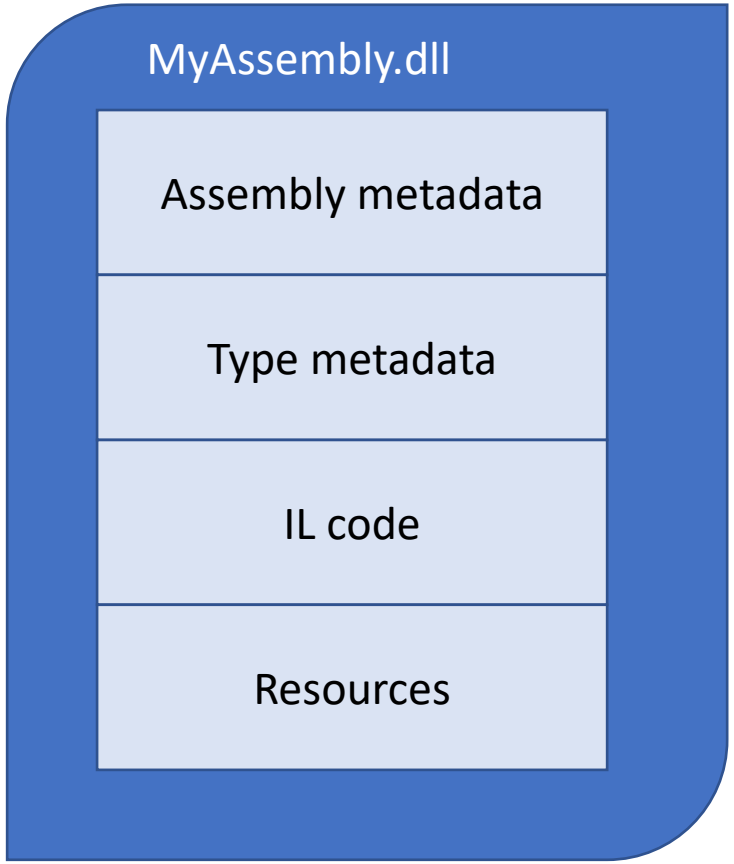
- Skládá se ze 4 částí:
  - Název assemly (například A.dll)
  - Verze assembly (například 1.0.1033.17)
  - Kultura assembly (Systém.Globalization.CultureInfo) – kultura určuje národní a jazykové zvyklosti (formát data, čísel, atd..)
  - Veřejný klíč.
- Konfiguruje se v kódu pomocí atributů.

```
using System.Reflection;
[assembly:AssemblyVersion("1.0.1033.17")]
[assembly:AssemblyCulture("en-US")]
[assembly:AssemblyKeyFile("myKeyFile.snk")]
class A {
    ...
}
```

- Assembly které mají Strong Name lze umístit do Global Assembly Cache. Jedná se o tzv. „veřejné/sdílené assembly“. Mohou je využívat všechny aplikace v systému.

Metadata

- Metadata assembly (manifest)
  - Popisují assembly
    - Název
    - Verze
    - Klíč,
    - Kultura
- Metadata typů
  - Popisují datové typy
    - Namespace
    - Názvy typů (třídy, interface, delegáti, atd..)
    - Metody, vlastnosti, konstruktory, atd..
    - …



Reflexe

- Umožňuje analýzu, konstrukci a spouštění kódu za běhu aplikace.
- Pro analýzu kódu využívá metadat.
- Umožňuje spouštět kód aplikace bez jeho předešlé znalosti.
- Teoreticky umožňuje interpretaci kódů – spouštění kódu dynamicky.
- Příklady využití: objektově relační mapování, serializace/deserializace objektů, systém rozšíření (pluginů), atd...
- Vstupním bodem je namespace „System.Reflection.Assembly“.

Třída Assembly

- Je „vstupní třídou“ pro reflexi.
- Reprezentuje jednu assembly a nabízí metody pro práci s ní.
- Důležité statické metody:
  - GetAssembly – získání assembly na základě předaného typu.
  - Load – načtení assembly na základě jejího názvu.
  - LoadFile – načtení assembly na základě cesty k souboru.
  - GetExecutingAssembly – vrátí assembly ve které je umístěn kód, který tuto metodu volá.
- Důležité vlastnosti:
  - EntryPoint – vstupní bod – většinou metoda „Main“ ve třídě „Program“.
  - FullName – název assembly (Strong name).
  - GlobalAssemblyCache – je assembly v GAC?
  - Location – cesta k assembly.
  - ReflectionOnly – je assembly načtena jen pro reflexi? (kód nelze spouštět).

Třída Assembly

- Důležité instanční metody:
  - CreateInstance – umožňuje vytvářen instance předaného typu (nejčastěji třídy).
  - GetCustomAttributes – vrátí pole atributů.
  - GetModules – vrátí pole modulů ze kterých se assembly skládá.
  - GetTypes – vrátí pole všech typů definovaných v dané assembly.

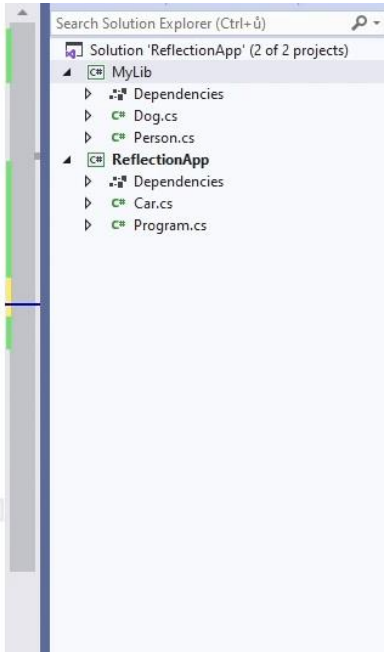
```
using System;
using System.IO;
using System.Reflection;

namespace ReflectionApp
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Assembly assembly = Assembly.LoadFile(Path.GetFullPath(path: "../../../../MyLib/bin/Debug/netcoreapp3.1/MyLib.dll"));


            object obj = assembly.CreateInstance(typeName: "MyLib.Person");

            Console.WriteLine(obj);

        }
    }
}
```



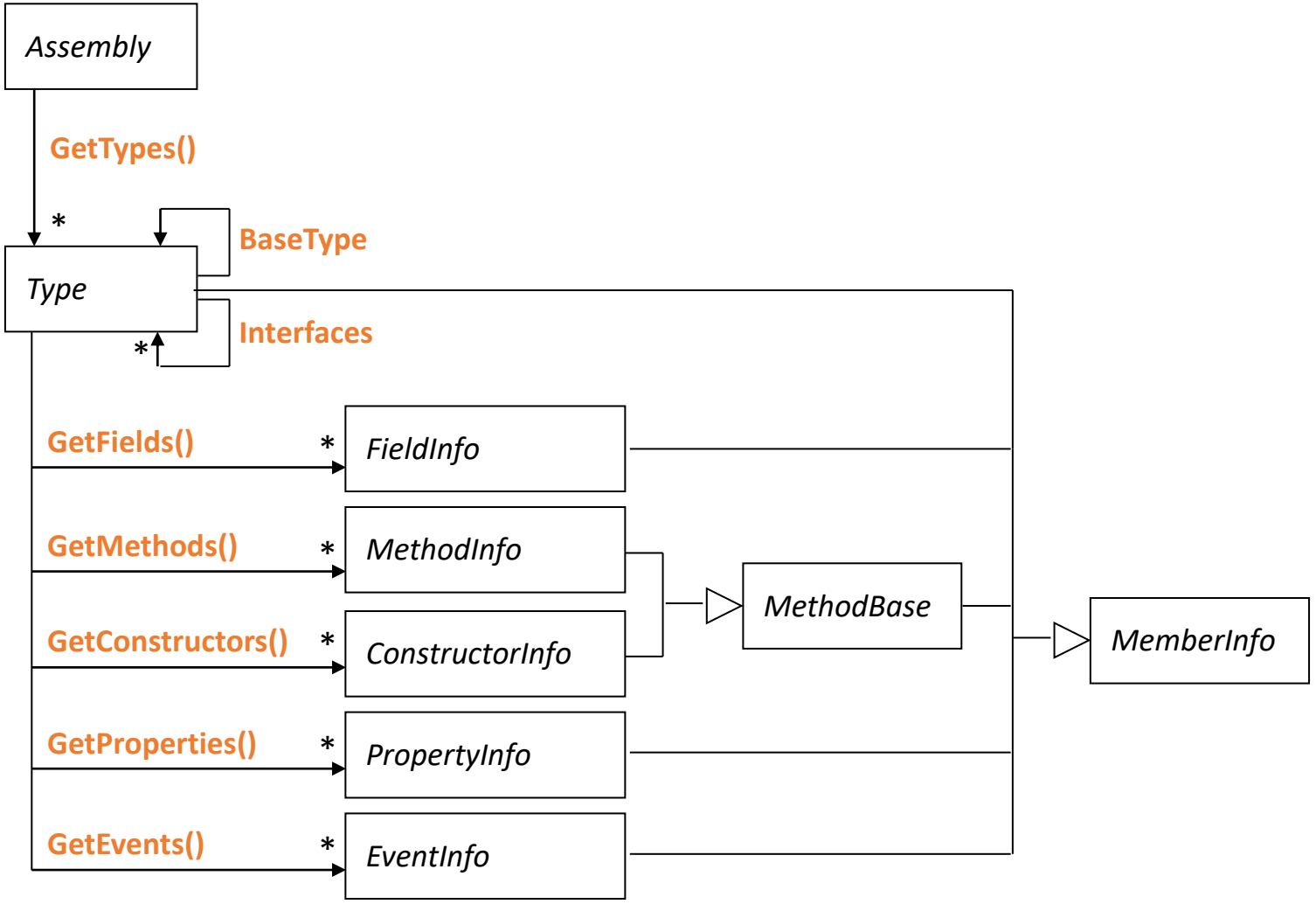
Typ – třída Type

- Třída Type **popisuje datový typ – metadata objektu/proměnné**
- Instanci třídy Type lze získat:
  1. Z assembly pomocí metody GetTypes – musíme znát název typu a assembly, ve které je umístěn.
  2. Z modulu pomocí metody GetTypes - musíme znát název typu a modul, ve které je umístěn.
  3. Z instance třídy pomocí metody GetType – stačí mít k dispozici instanci.
  4. Z třídy pomocí klíčového slova „typeof“.
- Instance třídy Type obsahuje informace o vlastnostech daného typu a nabízí metody pro přístup k metodám, proměnným, vlastnostem, konstruktorům, atd.

Důležité instanční vlastnosti a metody třídy Type

- Vlastnosti
  - FullName
  - BaseType
  - IsAbstract
  - IsClass
  - IsPublic
- Metody
  - GetInterface(s)
  - GetConstructor(s)
  - GetEvent(s)
  - GetField(s)
  - GetMethod(s)
  - GetPropertie(s)
  - GetMember(s)

Základní hiearchie reflexe



Binding Flags

- Modifikátory ovlivňující návratovou hodnotu metod jako je GetMembers, GetMethods, GetEvents, atd...
- Fungují v podstatě jako filtry. Výchozí filtrem je: *BindingFlags.Instance | BindingFlags.Static | BindingFlags.Public*
- Důležité hodnoty:
  - DeclaredOnly
  - Default
  - FlattenHierarchy
  - IgnoreCase
  - Instance
  - NonPublic
  - Public
  - Static
- Příklad přístupu k privátním instančním metodám:

```
Type myType = typeof(MyClassName);
MethodInfo[] methods = myType.GetMethods(BindingFlags.NonPublic | BindingFlags.Instance);
```

Vytvoření instance třídy

- Metoda CreateInstance na Assembly.
  - Vytvoří instanci na základě **názvu typu**.
  - Má velké množství parametrů.
- Activator.CreateInstance
  - Statická metoda na třídě Activator.
  - Vytvoří instanci na základě **instance třídy Type**.
  - Interně volá metodu CreateInstance na Assembly.
  - Slouží k zjednodušení vytváření instancí objektů.

```
//object person = assembly.CreateInstance("MyLib.Person");

Type t = assembly.GetType( name: "MyLib.Person");
object person = Activator.CreateInstance(t, new object[] { "Adam" });
```

Volání metod

- Pomocí metody „Invoke“ na MethodInfo.
- MethodInfo je pouze popis metody. Nejedná se o metodu!
- Prvním parametrem musí být objekt, na kterém se má daná metoda zavolat.
- Druhý parametr je pole parametrů pro volání metody. Typy musí korespondovat s definicí volané metody!
- Návratový hodnota je vždy typu object (ale obsahuje reálný návratový datový typ).

```
Type t = assembly.GetType(name: "MyLib.Person");
object person = Activator.CreateInstance(t, new object[] { "Adam" });

MethodInfo mi = t.GetMethod(name: "GetFullName");

string name = person.GetFullName();
Console.WriteLine(name);
```

```
MyClassName instance = new MyClassName();
Type myType = instance.GetType();
MethodInfo myMethod = myType.GetMethod("TestMethod");
object reurnedValue = myMethod.Invoke(instance, new object[] { 5, "Ahoj" });
```

Nastavení a čtení vlastností

- Pomocí metody „SetValue“ a „GetValue“ na PropertyInfo - u proměnných (Fields) je použití identické.
- PropertyInfo je pouze popis vlastnosti. Nejedná se o vlastnost!
- Návratový hodnota GetValue je vždy typu object (ale obsahuje reálný návratový datový typ).
- Při nastavení hodnoty pomocí SetValue musí mít hodnota správný datový typ!

```
MyClassName instance = new MyClassName();
Type myType = instance.GetType();
PropertyInfo myProperty = myType.GetProperty("TestProperty");
object originalValue = myProperty.GetValue(instance);
myProperty.SetValue(instance, 8);
```

Atributy

- Umožňuji přidání dodatečných informací k třídám, metodám, vlastnostem, atd.
- Slouží pouze pro účely reflexe.
- Musí dědit z abstraktní třídy „Attribute“.
- Standardně mají sufix „Attribute“ (například: MyExtraAttribute).
- Přistupuje se k ním pomocí metody „GetCustomAttributes“ na typu, metodě, vlastnosti, atd.
- Aplikuje se pomocí hranatých závorek. Suffix „Attribute“ se již uvádět nemusí.

```
class AuthAttribute : Attribute
{
     public string Role { get; set; }
     public AuthAttribute(string role)
     {
          this.Role = role;
     }
}


[Auth("Admin")]
public class UserStore {

}
```

Dynamické spouštění kódu

- Tento kód lze napsat čistě pomocí reflexe.

```
Hashtable table = new Hashtable();
table.Add("Ahoj", "Hello");
Console.WriteLine("Records: {0}", table.Count);
```

Dynamické spouštění kódu

```
// Hashtable table = new Hashtable();
Assembly assembly = Assembly.Load("mscorlib.dll");
Type tableType = assembly.GetType("System.Collections.Hashtable");
ConstructorInfo ctor = tableType.GetConstructor(Type.EmptyTypes);
object table = ctor.Invoke(new object[] { });


// table.Add("Ahoj", "Hello");
MethodInfo method = tableType.GetMethod("Add");
method.Invoke(table, new object[] { "Ahoj", "Hello" });


// Console.WriteLine("Records: {0}", table.Count);
PropertyInfo property = tableType.GetProperty("Count");
int count = (int)property.GetValue(table);
Type console = typeof(Console);
MethodInfo writeLine = console.GetMethod("WriteLine", new Type[] { typeof(string), typeof(string) });
writeLine.Invoke(null, new object[] {"Records: {0}", count.ToString()});
```

Dynamické generování assembly

- Assembly lze vytvářet i za běhu programu.
- Namespace: System.Reflection.Emit
- Důležité třídy:
  - AssemblyBuilder
  - ConstructorBuilder
  - EnumBuilder
  - EventBuilder
  - MethodBuilder
  - PropertyBuilder
  - TypeBuilder
  - …
- CIL instrukce: https://en.wikipedia.org/wiki/List_of_CIL_instructions

Příklad generování assembly

```
// Vytvoříme assembly "Test"
AssemblyName name = new AssemblyName("Test");
AssemblyBuilder assemblyBuilder = AssemblyBuilder
          .DefineDynamicAssembly(name, AssemblyBuilderAccess.RunAndCollect);
ModuleBuilder moduleBuilder = assemblyBuilder.DefineDynamicModule("TestModule");


// Vytvoříme datový typ "MyType"
TypeBuilder typeBuilder = moduleBuilder.DefineType("MyType", TypeAttributes.Public | TypeAttributes.Class);



// Vytvoříme metodu "SayHello" v rámci typu
Type[] prms = new Type[] { typeof(string) };
Type ret = typeof(string);
MethodBuilder method = typeBuilder.DefineMethod("SayHello", MethodAttributes.Public, ret, prms);
```

Příklad generování assembly

```
// Vygenerujeme CIL kód metody
ILGenerator ilGen = method.GetILGenerator();
ilGen.Emit(OpCodes.Ldstr, "Hello ");
ilGen.Emit(OpCodes.Ldarg_1);
Type t = Type.GetType("System.String");
MethodInfo mi = t.GetMethod("Concat", new Type[] { typeof(string), typeof(string) });
ilGen.Emit(OpCodes.Call, mi);
ilGen.Emit(OpCodes.Ret);

// sestavíme typ
typeBuilder.CreateType();


// zavoláme metodu
MethodInfo testMethod = typeBuilder.GetMethod("SayHello", new Type[] { typeof(string) });
object obj = Activator.CreateInstance(typeBuilder);
object result = testMethod.Invoke(obj, new string[] { "Word!" });
Console.WriteLine(result);
```

Zdroje

- https://docs.microsoft.com/cs-cz/dotnet/
- https://devblogs.microsoft.com/
- https://www.codeguru.com/csharp/

- YAMIKANI FUKIZI, Kenneth, Jason DE OLIVEIRA a Michel BRUCHET. *Learn ASP.NET Core 3: Develop modern web applications*. Second edition. Packt Publishing, 2019. ISBN 978-1789610130.
- ALBAHARI, Joseph. *C# 10.0 in a Nutshell: The Definitive Reference*. O'Reilly Media; 1st edition, 2022. ISBN 978-1098121952.
- ALBAHARI, Joseph. *C# 10 and .NET 6 – Modern Cross-Platform Development: Build apps, websites, and services with ASP.NET Core 6, Blazor, and EF Core 6 using Visual Studio 2022 and Visual Studio Code*. 6th edition. Packt Publishing, 2021. ISBN 978-1801077361.