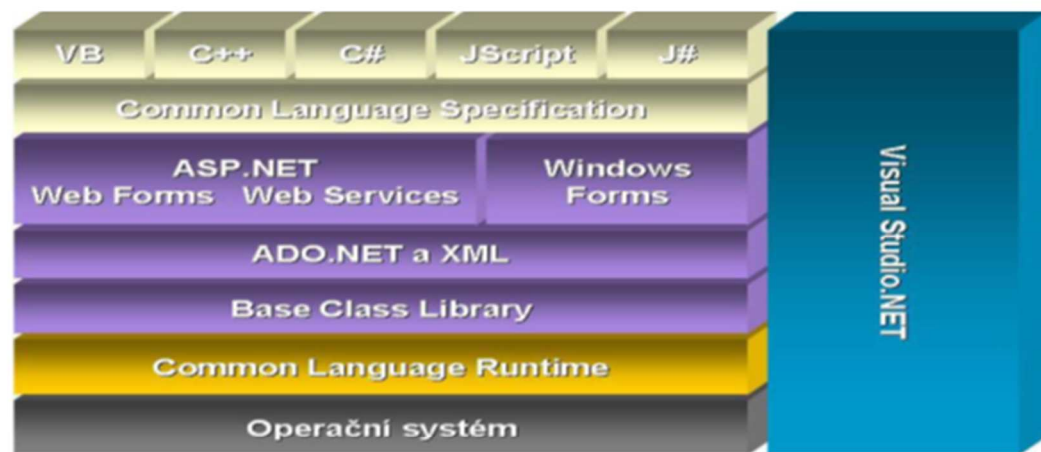


Objektově orientované programování řízené událostmi.

Architektura .NET Framework



Na nejnižší úrovni se nachází **CLR- Common Language Runtime** realizující základní infrastrukturu, nad kterou je framework vybudován.

Nad CLR se nachází několik hierarchicky umístěných knihoven pro klasické i webové aplikace.

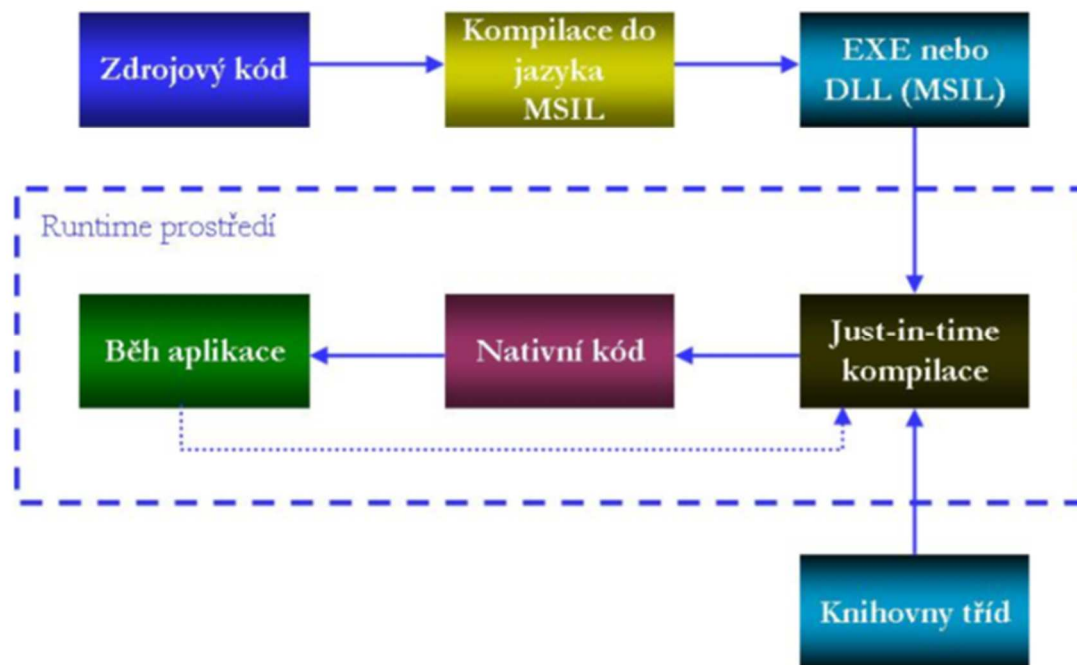
Ty jsou rozděleny do jmenných prostorů.

Poslední vrstvu tvoří programovací jazyky.

Celá tato architektura je podpořena vývojovým nástrojem **Visual Studio .NET**.

CLR – Common Language Runtime

V prostředí .NET jsou zdrojové soubory libovolného programovacího jazyka zkompileovány do intermediárního jazyka (nazvaného *MSIL – Microsoft Intermediate Language*). V případě, že má být taková aplikace spuštěna, systém detekuje, že jde o aplikaci v MSIL a spustí *Just-In-Time kompilátor*. Ten vygeneruje skutečné instrukce cílové platformy.



Assembly

Assembly je logická kolekce jednoho nebo více .exe, .dll nebo .module souboru doplněná o *manifest*.

Ukázka **ASSEMBLY** z naší konzolové aplikace

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

[assembly: AssemblyTitle("LinRov")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("LinRov")]
[assembly: AssemblyCopyright("Copyright © 2025")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

[assembly: ComVisible(false)]

[assembly: Guid("37b70838-39d1-4247-9e77-24279fd3d9ab")]

[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

Ukázka **MANIFESTU** z naší konzolové aplikace

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1"
manifestVersion="1.0">
  <assemblyIdentity version="1.0.0.0" name="MyApplication.app"/>
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
    <security>
      <requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
        <requestedExecutionLevel level="asInvoker" uiAccess="false"/>
      </requestedPrivileges>
    </security>
  </trustInfo>
</assembly>
```

Základy objektově orientovaného programování

CLASS - TŘÍDA

Třídy jsou v podstatě šablony pro vytváření **(OBJEKTŮ)**. Objekt je konkrétní instance třídy.

Třídy mohou také obsahovat funkce **(METODY)**, které jsou dostupné pouze pro objekty daného typu.

Konstruktor: Speciální metoda, která se automaticky zavolá při vytváření nového objektu a slouží k jeho inicializaci.

```
//Deklarace třídy
public class Osoba
{
    public string Jmeno { get; set; }
    public int Vek { get; set; }

    //Konstruktor
    public Osoba(string jmeno, int vek)
    {
        Jmeno = jmeno;
        Vek = vek;
    }
}

//Vytvoření instance třídy (objekt)
Osoba mojeOsoba = new Osoba("Petr", 30);
```

Namespace – jmenný prostor

Obsahuje mnoho základních tříd používaných v jazyce a frameworku .NET

Příklady

```
using System;
```

[Systém Namespace](#)

```
using System.Windows.Forms;
```

[System.Windows.Forms Namespace](#)

Vlastnosti objektů.

Polymorfismus

vlastnost programovacího jazyka, objektově orientovaného programování (OOP), která umožňuje:

- jednomu objektu volat jednu metodu s různými parametry ;
- objektům odvozeným z různých tříd volat tutéž metodu se stejným významem v kontextu jejich třídy, často pomocí rozhraní;
- přetěžování operátorů neboli provedení rozdílné operace v závislosti na typu operandů;

Rozhodnutí o tom, která metoda bude volána, je u polymorfismu prováděno až za běhu programu (tj. dynamicky pomocí virtuálních funkcí). Tím se odlišuje od přetěžování funkcí, kde je rozhodnutí o volání vhodné funkce provedeno již při překladu (tj. staticky).

Encapsulation - zapouzdření

Zapouzdření je způsob, jak omezit přímý přístup k některým komponentám objektu, takže uživatelé nemohou přistupovat k hodnotám stavů všech proměnných daného objektu. Zapouzdření lze použít ke skrytí datových členů i datových funkcí nebo metod spojených s instancí třídy nebo objektu.

Zapouzdření je užitečné při práci se zabezpečenými daty nebo metodami, protože může omezit, které funkce nebo uživatelé mají přístup k určitým informacím.

Dědičnost - inheritance

Dědičnost v programování je koncept objektově orientovaného programování (OOP), který umožňuje nové třídě (potomek) převzít vlastnosti a metody již existující třídy (předek) a zároveň je rozšířit, upravit nebo přepsat.