



ОГЛЯД ПЛАТФОРМИ .NET

Питання 1.2.

Технології платформи .NET

- Пов'язані платформи для розробки додатків та сервісів, інколи перетинаються



**.NET
Native**

Технология	Возможности	Компилирует в	Хостовая ОС
.NET Framework	Как отработанные, так и современные	Промежуточный язык	Только Windows
Xamarin	Только мобильные	Промежуточный язык	iOS, Android, Windows Mobile
.NET Core	Только современные	Промежуточный язык	Windows, Linux, macOS
.NET Native	Только современные	Машинный код	Windows, Linux, macOS

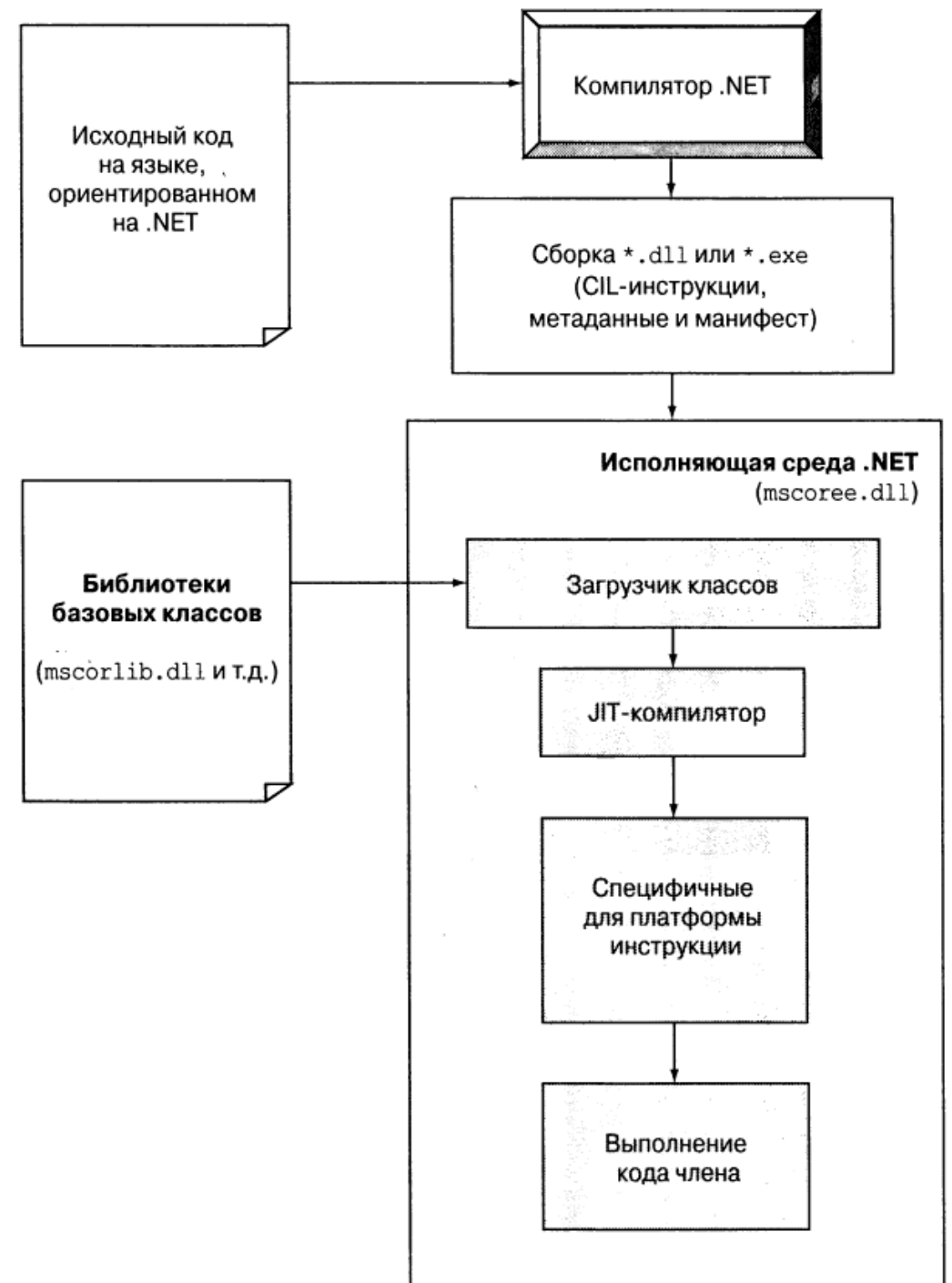
Еволюція стеку технологій Microsoft. .NET Framework



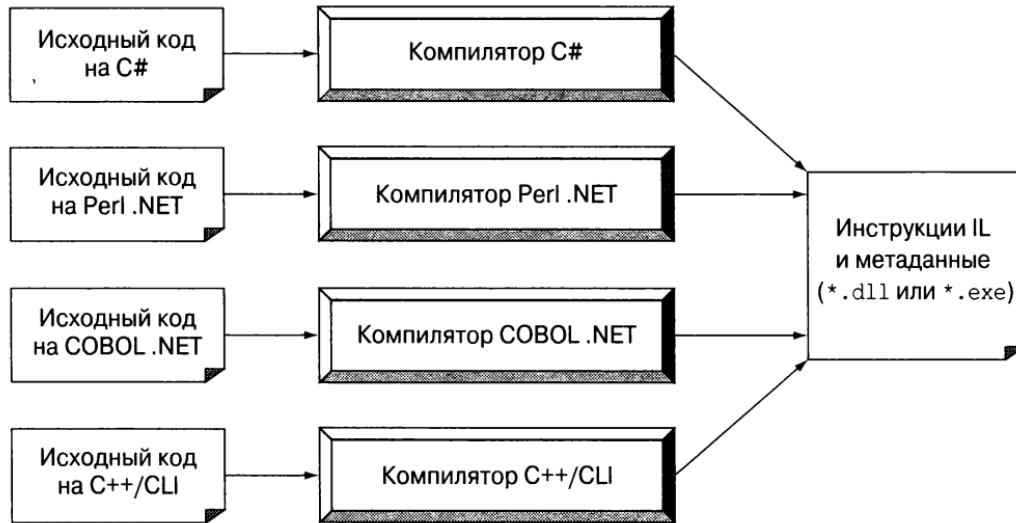
- *Common Language Runtime (CLR)* – спільне (загальномовне) виконавче середовище.
 - Автоматично знаходить, завантажує та керує об'єктами .NET замість програміста
 - Опікується деякими низькорівневими деталями: управління пам'яттю, розміщення додатку, координація потоків та виконання перевірок, пов'язаних з безпекою.
- У програмуванні виконавче середовище – колекція служб, потрібних для виконання скомпільованої одиниці коду.
 - Основний механізм CLR фізично має вигляд бібліотеки mscorlib.dll (спільного механізму виконання виконуючого коду об'єктів (Common Object Runtime Execution Engine))

Збірка mscoree.dll в дії

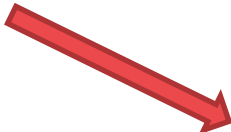
- Головна задача – визначення місця розташування збірки та знаходження запитаного типу в двійковому файлі за рахунок читання його метаданих.
 - Потім CLR розміщує тип у пам'яті, перетворює пов'язаний з ним CIL-код у специфічні для платформи інструкції, виконує всі необхідні безпекові перевірки, а потім виконує потрібний код.
 - Двійкові модулі .NET містять не специфічні, а незалежні від платформи інструкції проміжною мовою (Intermediate Language — IL) та метадані типів.
 - Коли файл *.dll або *.exe було створено за допомогою .NET-компілятора, отриманий великий двійковий об'єкт називається **збіркою**.
- C# не постачається з певною специфічною для мови бібліотекою коду.
 - Використовуються нейтральні до мов програмування бібліотеки .NET.



Кроскомпіляція, збірки та CIL-код



```
using System;
namespace CalculatorExample
{
    // Этот класс содержит точку входа приложения.
    class Program
    {
        static void Main()
        {
            Calc c = new Calc();
            int ans = c.Add(10, 84);
            Console.WriteLine("10 + 84 is {0}.", ans);
            // Ожидать нажатия пользователем клавиши <Enter>
            Console.ReadLine();
        }
    }
    // Калькулятор на C#.
    class Calc
    {
        public int Add(int x, int y)
        { return x + y; }
    }
}
```



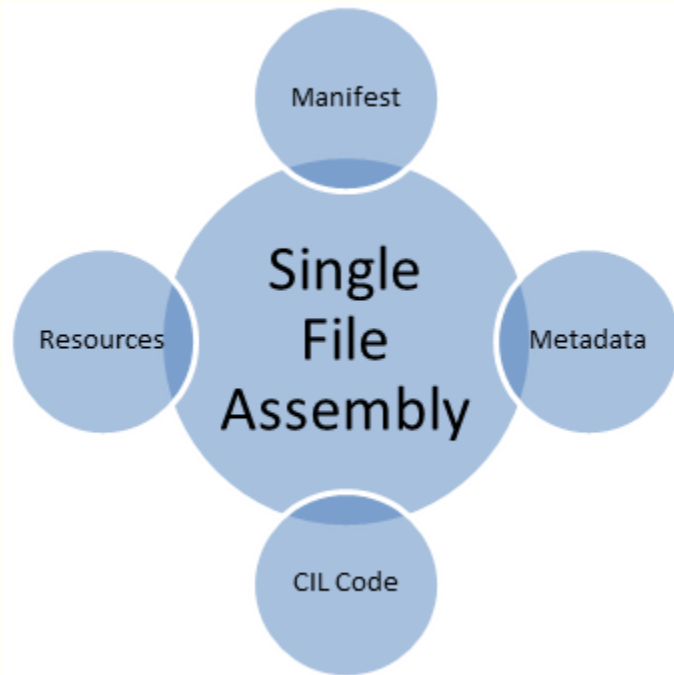
```
.method public hidebysig instance int32 Add(int32 x,
int32 y) cil managed
{
    // Code size 9 (0x9)
    .maxstack 2
    .locals init (int32 V_0)
    IL_0000: nop
    IL_0001: ldarg.1
    IL_0002: ldarg.2
    IL_0003: add
    IL_0004: stloc.0
    IL_0005: br.s IL_0007
    IL_0007: ldloc.0
    IL_0008: ret
} // end of method Calc::Add
```

- Збірка містить код CIL та не компілюється в специфічні для платформи інструкції, поки на блок інструкцій CIL (наприклад, реалізацію методу) не здійснюється посилання для його виконання виконуючим середовищем .NET.

Метадані в збірці

Type Descriptions

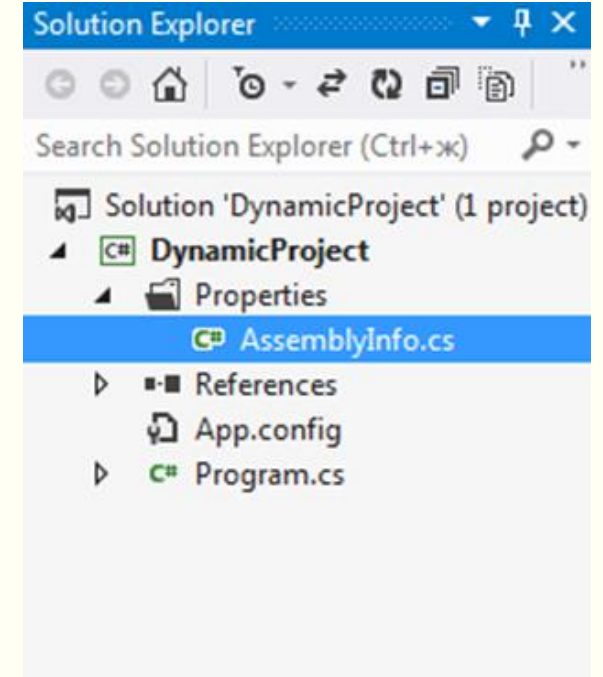
Classes
Base classes
Implemented interfaces
Attributes
Methods



TypeDef #2 (02000003)

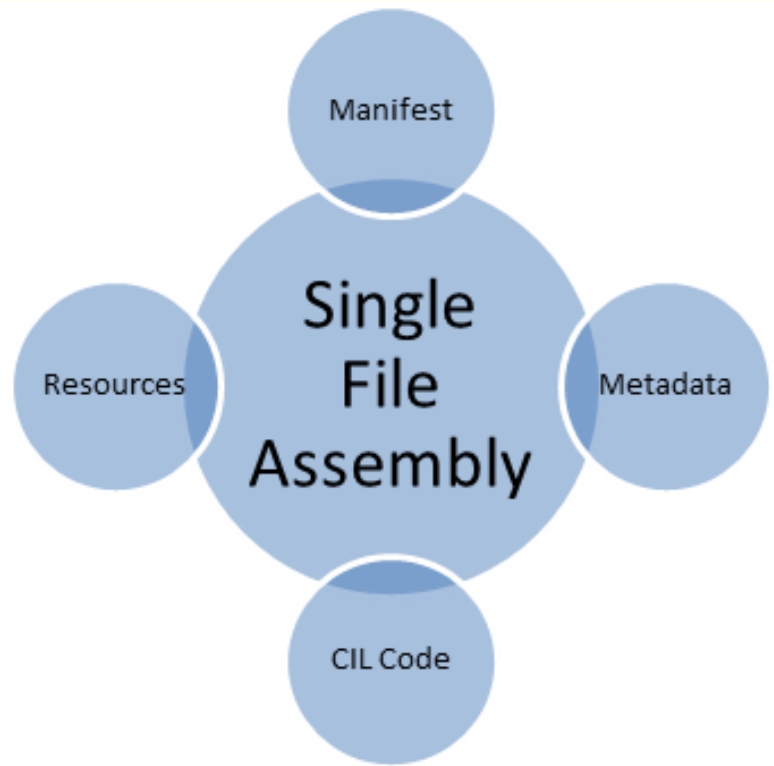
```
TypeDefName: CalculatorExample.Calc (02000003)
Flags       : [NotPublic] [AutoLayout] [Class]
[AnsiClass] [BeforeFieldInit] (00100001)
Extends     : 01000001 [TypeRef] System.Object
Method #1 (06000003)
```

```
MethodName: Add (06000003)
Flags       : [Public] [HideBySig] [ReuseSlot] (00000086)
RVA         : 0x00002090
ImplFlags   : [IL] [Managed] (00000000)
CallConvtn : [DEFAULT]
hasThis
ReturnType: I4
  2 Arguments
  Argument #1: I4
  Argument #2: I4
  2 Parameters
  (1) ParamToken : (08000001) Name : x flags: [none] (00000000)
  (2) ParamToken : (08000002) Name : y flags: [none] (00000000)
```



- Крім інструкцій CIL, збірки також містять **метадані**, які детально описують характеристики кожного "типу" всередині двійкового модуля.
 - Метадані .NET автоматично генеруються компілятором відповідної мови .NET.
- Також збірки описуються за допомогою метаданих у **маніфесті**: поточної версії збірки, інформацію про культуру (локалізація ресурсів) та список посилань на всі зовнішні збірки, потрібні для правильного функціонування

Ресурси



- Збірка також може містити ресурси: текст, зображення, XML-файли тощо.
 - Зазвичай у якості ресурсів виступають зображення та дані, що локалізуються.
 - Ресурс у результаті є іменованим байтовим потоком.
 - Збірка зберігає ресурси в словниках з рядковими ключами.

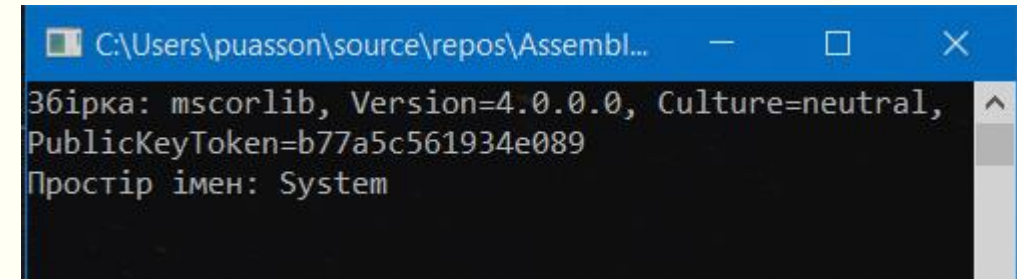
Збірки, NuGet-пакети та платформи

- Бібліотеки попередньо зібраного коду (BCL, CoreFX), складаються зі **збірок** (*assembly*) та **просторів імен** (*namespace*), які спрощують керування десятками тисяч доступних типів.
 - Простір імен надає основу для логічного групування коду, а збірка - фізичного групування.
- **Збірки** використовуються для зберігання типів у файловій системі.
 - По суті, це механізм розгортання коду.
 - Наприклад, збірка System.Data.dll містить типи для керування даними. Для використання цих типів на збірку треба послатись.
 - Збірки часто розповсюджуються в вигляді NuGet-пакетів, які можуть містити кілька збірок та інших ресурсів.
- **Простір імен** – це адреса типу.
 - Механізм унікальної ідентифікації типу через його повну адресу, а не просто коротку назву.
 - Наприклад, в .NET Core інтерфейс IActionFilter з простору імен System.Web.Mvc відрізняється від інтерфейсу IActionFilter з простору імен System.Web.Http.Filters.

Перший додаток (проект AssemblyVsNamespace)

```
using System;           // тип Console для вводу-виводу в консоль
                        // тип Type для отримання даних про тип
using System.Reflection; // рефлексія для отримання збірки
using System.Text;       // для узгодження кодування тексту
                        // в редакторі (UTF-8) та консолі (chcp 1251)

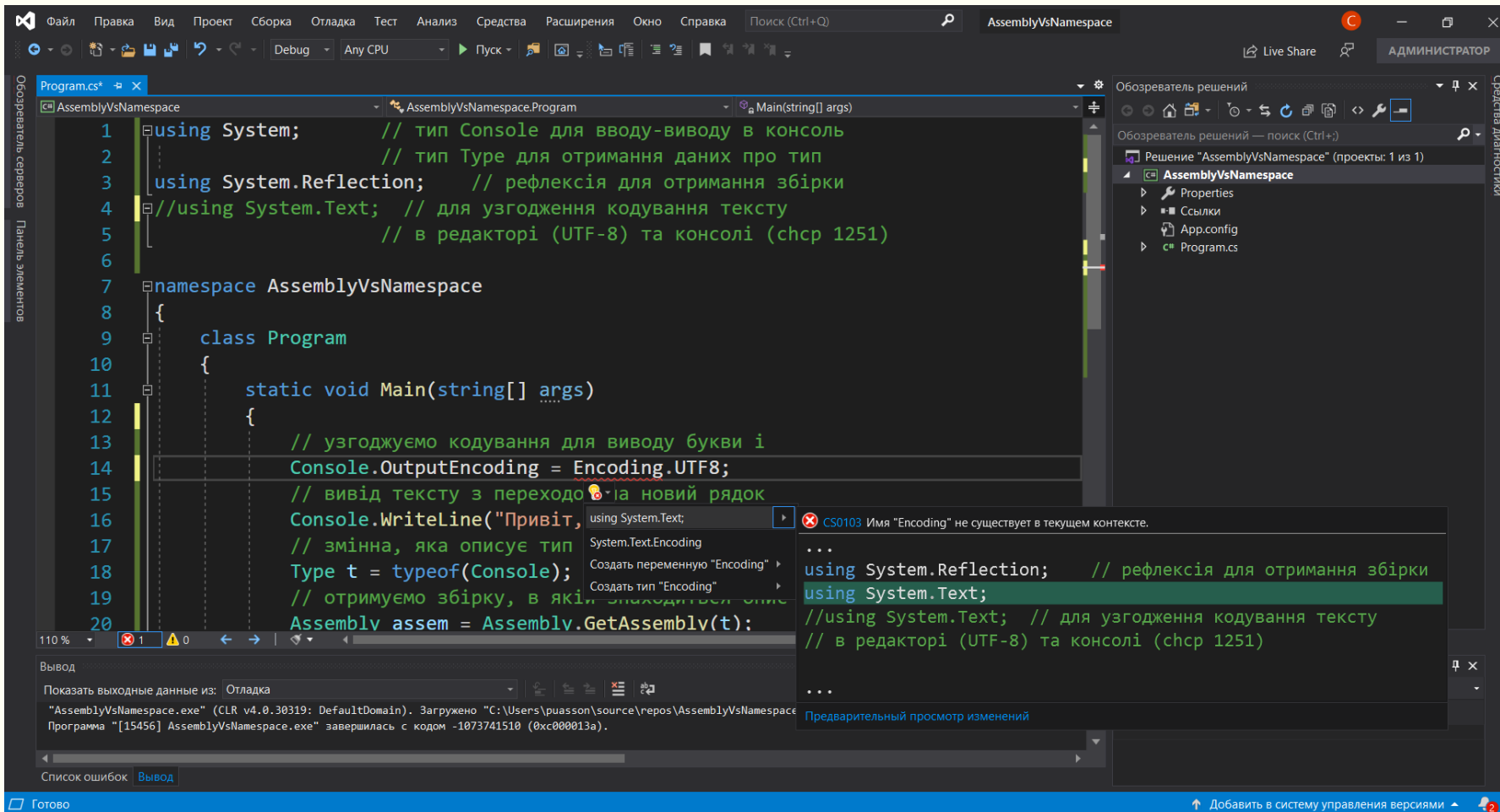
namespace AssemblyVsNamespace {
    class Program {
        static void Main(string[] args)
        {
            // узгоджуємо кодування для виводу букви і
            Console.OutputEncoding = Encoding.UTF8;
            // вивід тексту з переходом на новий рядок
            Console.WriteLine("Привіт, світ!");
            // змінна, яка описує тип Console
            Type t = typeof(Console);
            // отримуємо збірку, в якій знаходиться опис типу
            Assembly assem = Assembly.GetAssembly(t);
            // виводимо дані про збірку та простір імен
            Console.WriteLine("Збірка: {0}", assem.FullName);
            Console.WriteLine("Простір імен: {0}", t.Namespace);
            Console.ReadKey(); // щоб вікно консолі не закрилось
        }
    }
}
```



■ Завдання:

- Показати вивід текст у консоль
- Порівняти дані про збірку та про простір імен для одного типу

Імпорт простору імен



- Закоментуємо простір імен `System.Text` для подальшого його імпорту
 - Можна використати команду **Ctrl + «.»**

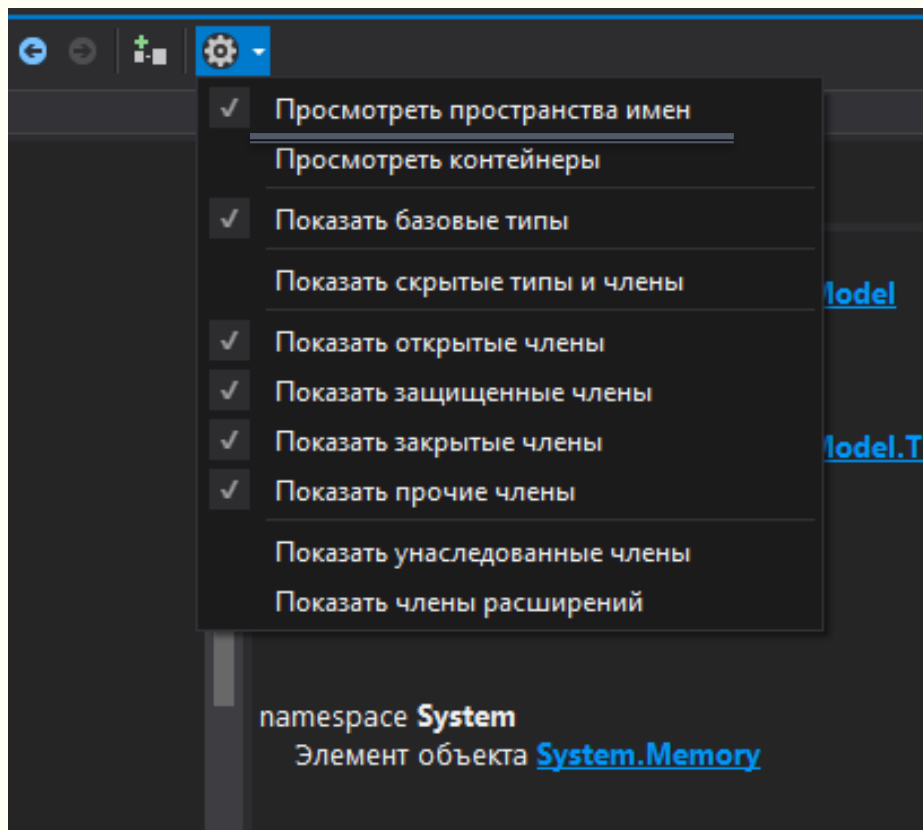
Зв'язані збірки та простори імен (Вид → Обозреватель объектов)

The screenshot displays the Visual Studio IDE with two main windows open:

- Обозреватель объектов (Object Explorer):** Shows the hierarchy of the `Program.cs` project. The `System` namespace is expanded, showing various classes and methods. The `Console` class is highlighted, showing its methods: `Beep()`, `Beep(int, int)`, `Clear()`, `MoveBufferArea(int, int, int, int, int, int)`, `MoveBufferArea(int, int, int, int, int, int, char, System.ConsoleColor, System.ConsoleColor)`, `OpenStandardError()`, `OpenStandardError(int)`, `OpenStandardInput()`, `OpenStandardInput(int)`, `OpenStandardOutput()`, `OpenStandardOutput(int)`, `Read()`, `ReadKey()`, `ReadKey(bool)`, `ReadLine()`, `ResetColor()`, `SetBufferSize(int, int)`, `SetCursorPosition(int, int)`, `SetError(System.IO.TextWriter)`, `SetIn(System.IO.TextReader)`, `SetOut(System.IO.TextWriter)`, `SetWindowPosition(int, int)`, `SetWindowSize(int, int)`, `Write(bool)`, `Write(char)`, `Write(char[])`, `Write(char[], int, int)`, and `Write(decimal)`. Below the list, the `public static class Console` is shown, with a note: "Элемент объекта System".
- Обозреватель решений (Solution Explorer):** Shows the project structure. The `ConsoleApp` project is selected, showing its dependencies: `Microsoft.NETCore.App (2.2.0)`, `Microsoft.NETCore.DotNetHostPolicy (2.2.0)`, `Microsoft.NETCore.Platforms (2.2.0)`, `Microsoft.NETCore.Targets (2.0.0)`, `Microsoft.CSharp.dll`, `Microsoft.VisualBasic.dll`, `Microsoft.Win32.Primitives.dll`, `mscorlib.dll`, `netstandard.dll`, `System.AppContext.dll`, `System.Buffers.dll`, `System.Collections.Concurrent.dll`, `System.Collections.dll`, `System.Collections.Immutable.dll`, `System.Collections.NonGeneric.dll`, `System.Collections.Specialized.dll`, `System.ComponentModel.Annotations.dll`, `System.ComponentModel.DataAnnotations.dll`, `System.ComponentModel.dll`, `System.ComponentModel.EventBasedAsync.dll`, `System.ComponentModel.Primitives.dll`, `System.ComponentModel.TypeConverter.dll`, `System.Configuration.dll`, `System.Console.dll`, `System.Core.dll`, `System.Data.Common.dll`, `System.Data.dll`, `System.Diagnostics.Contracts.dll`, `System.Diagnostics.Debug.dll`, `System.Diagnostics.DiagnosticSource.dll`, `System.Diagnostics.FileVersionInfo.dll`, `System.Diagnostics.Process.dll`, and `System.Diagnostics.StackTrace.dll`.

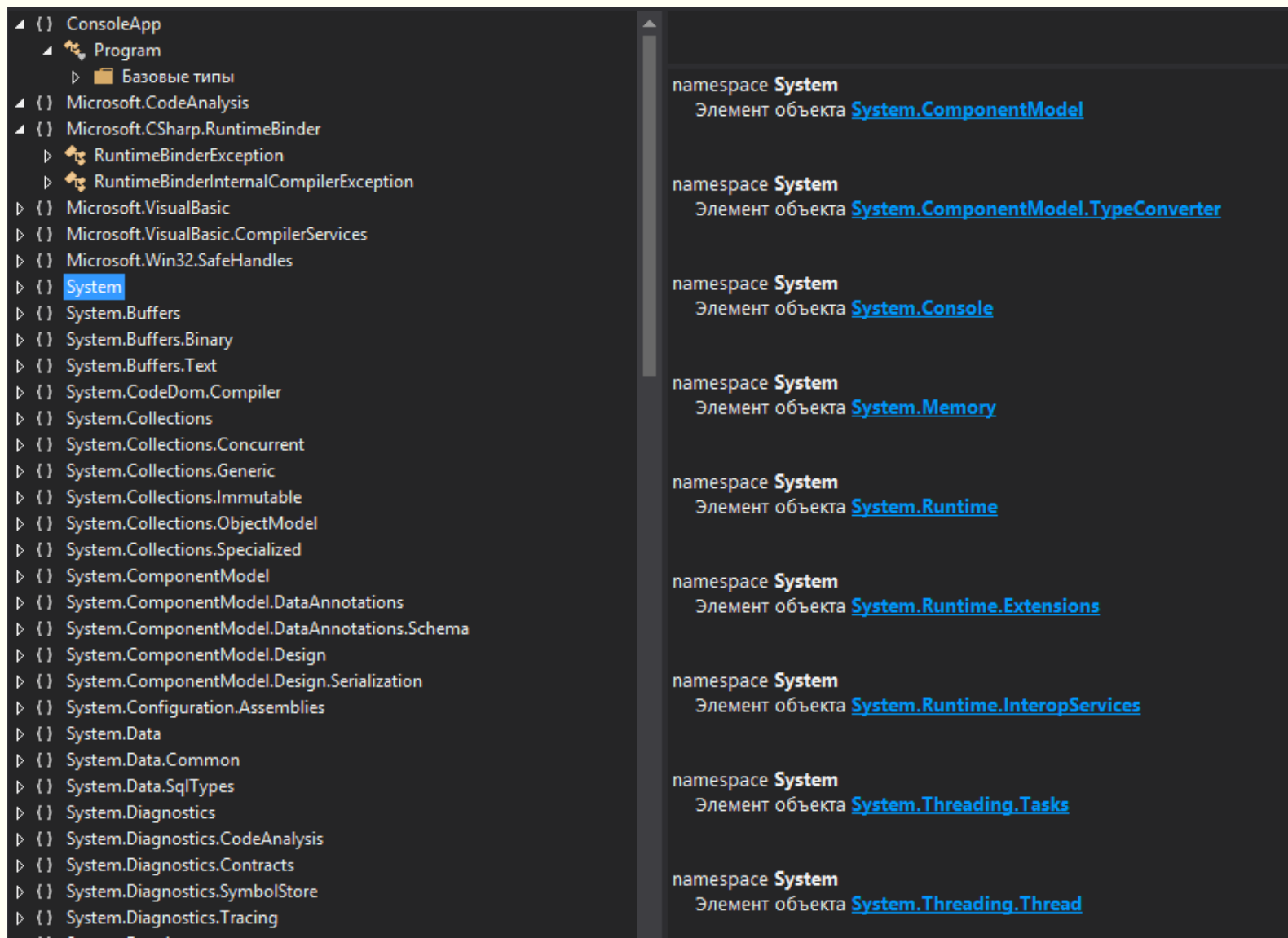
At the bottom of the screen, the text "@Марченко С.В., ЧДБК, 2020" is visible.

Зв'язані збірки та простори імен. Перегляд просторів імен



- Тепер панель Object Browser (Обозреватель объектов) відображає типи, згруповані за збірками.

28.08.2020



Головні компоненти .NET Framework



- *Common Type System (CTS)* – специфікація, яка описує всі можливі типи даних та програмні конструкції, що підтримуються середовищем.
 - У ній показано, як сутності можуть взаємодіяти та як вони представлені у вигляді метаданих .NET.
- *Common Language Specification (CLS)* – описує підмножину спільних типів та програмних конструкцій, які повинні підтримувати всі мови програмування
 - Якщо створювані .NET-типи відповідають CLS, ними можуть користуватись усі .NET-мови
 - Інакше гарантувати можливість взаємодії з такою бібліотекою коду неможливо
 - Можна вказати компілятору, щоб перевінив код на відповідність CLS

Спільна система типів. Вбудовані типи даних CTS

Тип данных CTS	Ключевое слово VB	Ключевое слово C#	Ключевое слово C++/CLI
System.Byte	Byte	byte	unsigned char
System.SByte	SByte	sbyte	signed char
System.Int16	Short	short	short
System.Int32	Integer	int	int или long
System.Int64	Long	long	__int64
System.UInt16	UShort	ushort	unsigned short
System.UInt32	UInteger	uint	unsigned int или unsigned long
System.UInt64	ULong	ulong	unsigned __int64
System.Single	Single	float	float
System.Double	Double	double	double
System.Object	Object	object	object^
System.Char	Char	char	wchar_t
System.String	String	string	String^
System.Decimal	Decimal	decimal	Decimal
System.Boolean	Boolean	bool	bool

Типи .NET та ключові слова C#

```
5 class Program
6 {
7     static void Main(string[] args)
8     {
9         Console.WriteLine("Hello World!");
10
11         int age = 30;
12     }
13 }
14
15
```

struct System.Int32
Represents a 32-bit signed integer.

- Всі ключові слова типів у C# є псевдонімами для типів .NET у збірці бібліотеки класів.

Keyword	.NET type
string	System.String
sbyte	System.SByte
short	System.Int16
int	System.Int32
long	System.Int64
float	System.Single
decimal	System.Decimal
object	System.Object

Keyword	.NET type
char	System.Char
byte	System.Byte
ushort	System.UInt16
uint	System.UInt32
ulong	System.UInt64
double	System.Double
bool	System.Boolean
dynamic	System.Dynamic.DynamicObject

Спільна система типів. Класи та інтерфейси

- **Класи та їх характеристики** (запечатаність, реалізація інтерфейсів, абстрактність, видимість)

```
// Тип класу C# з одним методом.  
class Calc  
{  
    public int Add(int x, int y)  
    {  
        return x + y;  
    }  
}
```

- **Інтерфейси** — це іменовані колекції визначень абстрактних членів, які можуть підтримуватись (бути реалізованими) в заданому класі або структурі.

```
// Тип інтерфейсу в C# обычно объявляется  
// открытым, чтобы позволить типам в других  
// сборках реализовать его поведение.  
public interface IDraw  
{  
    void Draw();  
}
```

Спільна система типів. Структури та перелічення

- структуру можна вважати полегшеним типом класу, який має семантику на основі значень.
- Перелічення — зручна програмна конструкція, яка дозволяє групувати пари «назва-значення».

- За умовчанням для зберігання кожного елементу виділяється блок пам'яті, відповідний 32-бітному цілому, проте за потреби це можна змінити.

```
// Тип структури в C#.
struct Point
{
    // Структури можуть contain поля.
    public int xPos, yPos;

    // Структури можуть contain параметризовані конструктори.
    public Point(int x, int y)
    { xPos = x; yPos = y; }

    // Структури можуть определять методи.
    public void PrintPosition()
    {
        Console.WriteLine("{0}, {1}", xPos, yPos);
    }
}
```

```
enum CharacterType
{
    Wizard = 100,
    Fighter = 200,
    Thief = 300
}
```

Спільна система типів. Делегати

- Делегати є .NET-еквівалентом безпечних до типів вказівників на функції в стилі C.
 - Основна відмінність: делегат .NET – це клас, породжений від System.MulticastDelegate, а не просто вказівник на низькорівневу адресу в пам'яті.
 - У C# делегати оголошуються за допомогою ключового слова delegate.
 - Вони критично важливі, коли потрібно забезпечити об'єкт можливістю перенаправлення виклику іншому об'єкту та формують основу архітектури подій .NET.

```
// Этот тип делегата в C# может "указывать" на любой метод,  
// возвращающий значение int и принимающий два значения int.  
delegate int BinaryOp(int x, int y);
```

Спільна система типів. Члени типів CTS

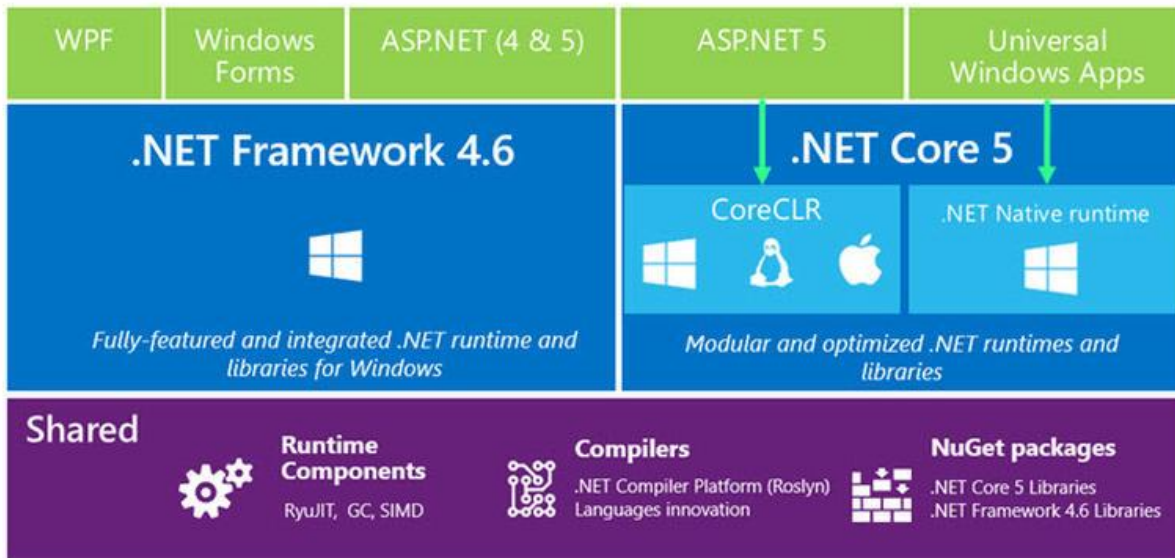
- Формально член типу обмежений набором:
 - {конструктор, фіналізатор, статичний конструктор, вкладений тип, операція, метод, властивість, індексатор, поле, поле тільки для зчитування, константа, подія}.
- У специфікації CTS описуються різні *характеристики* (adornments), які можуть з ним асоціюватись:
 - Наприклад, характеристика *доступності* (відкритий, закритий або захищений).
 - Деякі члени можуть оголошуватись як *абстрактні* або як *віртуальні*.
- Більшість членів можуть конфігуруватись як *статичні* або *члени екземпляру*.

Спільна специфікація мов (CLS) та забезпечення сумісності

- Набір правил, які детально описують мінімальну та повну множину характеристик, які повинен підтримувати окремий компілятор .NET, щоб генерувати програмний код, який обслуговуватиметься CLR і в той же час буде уніфіковано доступним іншим .NET-мовам.
 - Часто CLS можна розглядати як підмножину повної функціональності, визначеної в CTS.
- Компілятор C# можна змусити виконувати перевірку коду на сумісність з CLS, використовуючи .NET-атрибут [CLSCompliant]:

```
using System.Text; // для узгодження кодування тексту
                  // в редакторі (UTF-8) та консолі (chcp 1251)
[assembly: CLSCompliant(true)]
namespace AssemblyVsNamespace
{
    class Program
    {
        static void Main(string[] args)
        {
```

Тренд на кросплатформність



Сторонні розробники створили кросплатформну реалізацію .NET під назвою Mono.

- Вона значно відставала від .NET Framework.
- Пізніше Mono ліг в основу платформи створення мобільних додатків Xamarin одноіменної компанії.
- У 2016р. Microsoft придбала компанію Xamarin і тепер безкоштовно надає програмні рішення Xamarin як розширення для середовища розробки Visual Studio.

Microsoft активно працює над відокремленням платформи .NET від Windows.

- У 2015р. з'явилась перша кросплатформна версія .NET – .NET Core
- .NET Framework 4.8 – остання версія .NET Framework
- .NET Native – технологія попередньої компіляції, яка використовується при створенні універсальних Windows-додатків.

Технология	Возможности	Компилирует в	Хостовая ОС
.NET Framework	Как отработанные, так и современные	Промежуточный язык	Только Windows
Xamarin	Только мобильные	Промежуточный язык	iOS, Android, Windows Mobile
.NET Core	Только современные	Промежуточный язык	Windows, Linux, macOS
.NET Native	Только современные	Машинный код	Windows, Linux, macOS

Відмінності між .NET Framework та .NET Core

.NET Core	.NET Framework
Розповсюджується як NuGet-пакети, тому кожний додаток може розгортатись із власною, локальною копією .NET Core	Розповсюджується як системний набір збірок (фактично в Global Assembly Cache (GAC))
Розбивається на малі, шаруваті компоненти, тому може виконуватись мінімальний процес розгортання	Єдине, монолітне розгортання
Не включає старіші технології, на зразок ASP.NET Web Forms, та некросплатформні технології (AppDomains, .NET Remoting, бінарна серіалізація).	Зберігає деякі старі технології, на зразок ASP.NET Web Forms

- Пакети .NET Core визначають API, а платформи – групують пакети.
 - Платформа без пакетів не зможе визначити жоден API.
 - Кожний з пакетів .NET Core підтримує визначений набір платформ. Наприклад, пакет System.IO.FileSystem підтримує платформи NET Standard версії 1.3; .NET Framework версії 4,6; шість платформ Xamarin (зокрема, Xamarin.iOS 10)

Переваги використання пакетів

- Виробник може розповсюджувати їх за власним розкладом;
- Кожний пакет можна тестувати незалежно від інших пакетів;
- Пакети можуть підтримувати різні ОС та процесорні архітектури;
- Пакети можуть мати залежності, характерні лише для певної бібліотеки;
- Додатки мають менший розмір, оскільки невикористані пакети виключаються з дистрибутива

Пакет	Важные типы
System.Runtime	Object, String, Array
System.Collections	List<T>, Dictionary<TKey, TValue>
System.Net.Http	HttpClient, HttpResponseMessage
System.IO.FileSystem	File, Directory
System.Reflection	Assembly, TypeInfo, MethodInfo

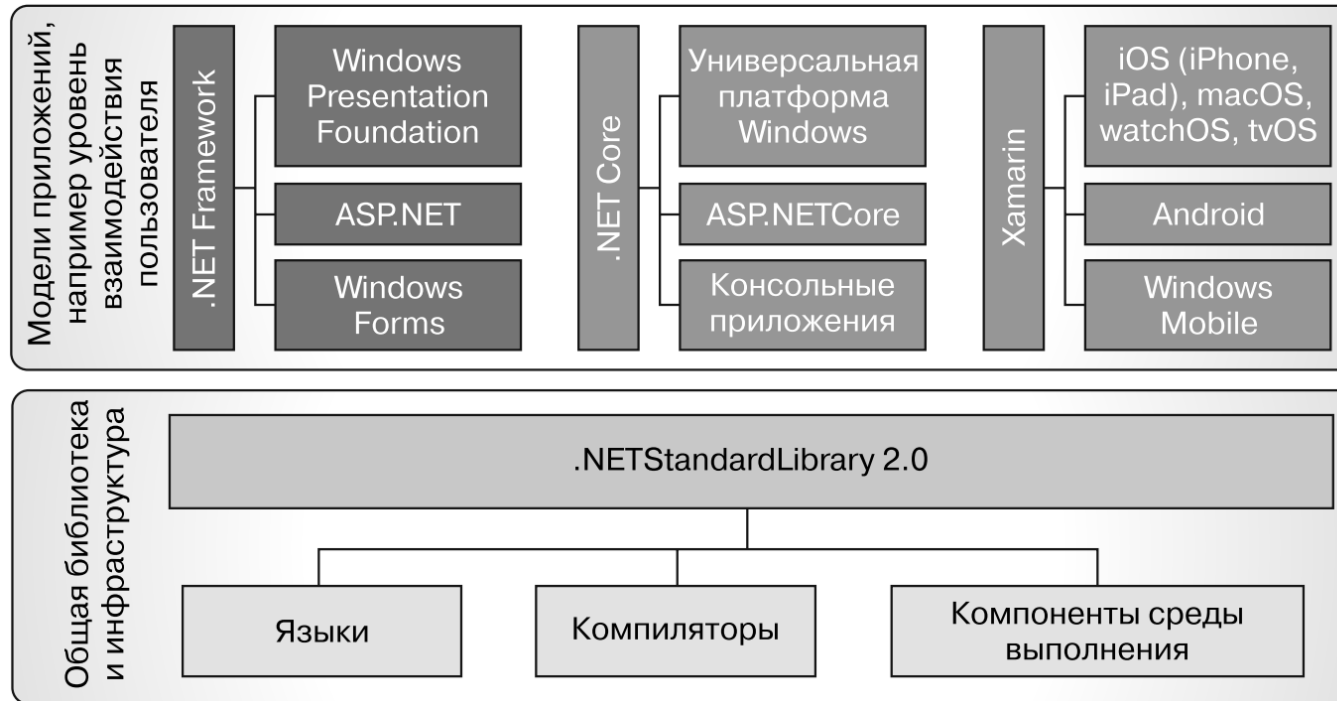
Еволюція платформи .NET Core

- .NET Core vs .NET Framework
 - .NET Core 1.0 (2016): набагато менший API в порівнянні з .NET Framework 4.6.1.
 - .NET Core 2.0: досяг паритету сучасних API з .NET Framework 4.7.1, оскільки обидві технології реалізують .NET Standard 2.0.
 - .NET Core 3.0: крупніший сучасний API в порівнянні з .NET Framework, оскільки .NET Framework 4.8 не реалізує .NET Standard 2.1.
- .NET Core 1.0 випущений в червні 2016р. та концентрується на реалізації API для побудови сучасних кросплатформних додатків, включаючи web- та хмарні додатки та служби для Linux, застосовуючи ASP.NET Core.
 - .NET Core 1.1 представлено в листопаді 2016р. Націлений на виправлення помилок, підвищення кількості підтримуваних дистрибутивів Linux, підтримку .NET Standard 1.6 та покращення продуктивності, особливо стосовно ASP.NET Core.

Еволюція платформи .NET Core

- .NET Core 2.0 концентрується на впровадженні .NET Standard 2.0, можливості додавання посилань на бібліотеки .NET Framework та підвищення продуктивності.
 - .NET Core для UWP-додатків випущено з Windows 10 Fall Creators Update у жовтні 2017р.
- .NET Core 2.1 представлено в травні 2018р. Націлений на реалізацію extendable tooling system, додавання нових типів, як Span<T>, нових API для криптографії та стиснення даних.
 - Windows Compatibility Pack з додатковими 20 000 API допомагає портувати старі Windows-додатки
 - Додано перетворення значень (value conversions) в Entity Framework Core, перетворення в LINQ GroupBy, заповнення даних (data seeding), query types.
- .NET Core 2.2 випущено в грудні 2018р. Фокусується на покращенні діагностичних інструментів для runtime, optional tiered compilation та додаванні нових можливостей в ASP.NET Core та Entity Framework Core: підтримка spatial data за допомогою типів з бібліотеки NetTopologySuite (NTS), теги запитів, collections of owned entities.

Стан платформи .NET до появи .NET Core 3.0



Технологія **.NET Standard** - специфікація спільного для **всіх** .NET-платформ набору API.

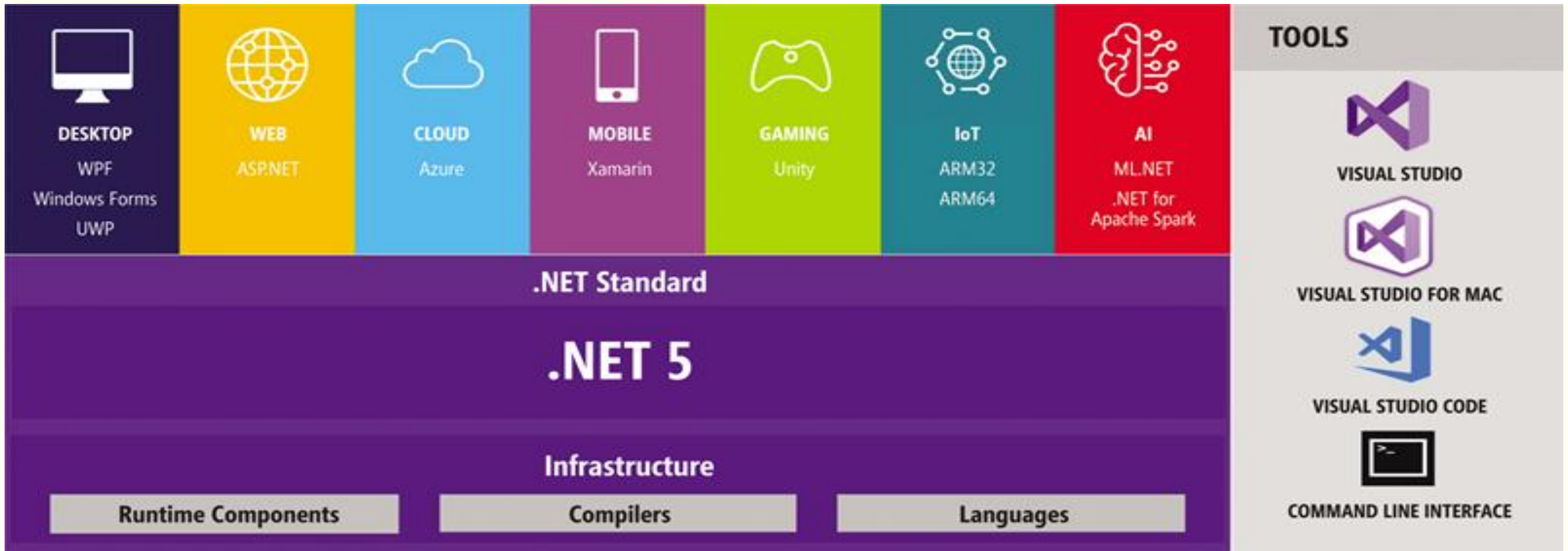
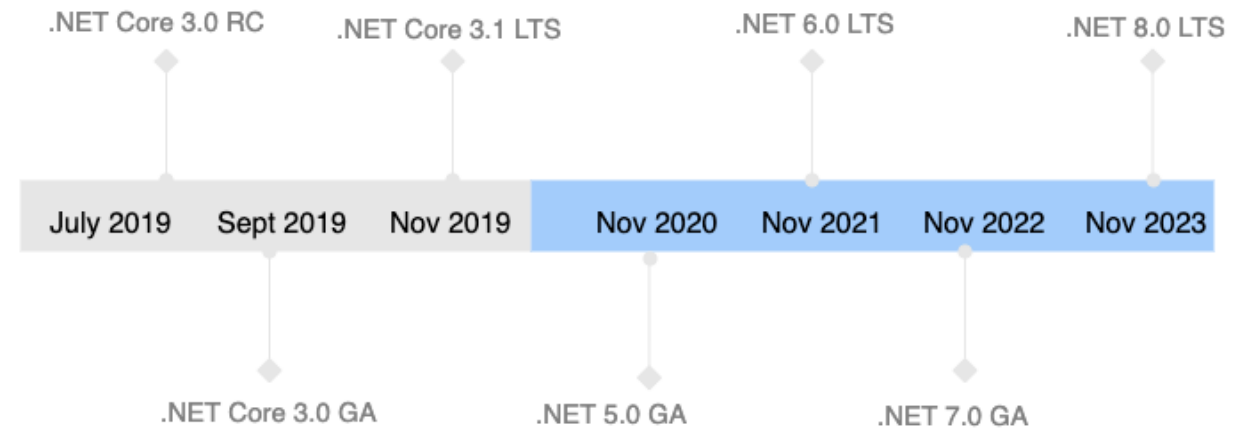
- До появи .NET Standard використовувались портативні бібліотеки класів (Portable Class Libraries, PCL).
- Існувала можливість створити бібліотеку коду та явно вказати підтримувані платформи: Xamarin, Silverlight, Windows 8 і т. д. Ваша бібліотека могла використовувати перетин можливостей API, підтримуваних вказаними платформами.

.NET Core 3.0 представлено у вересні 2019р.

- Націлений на додавання підтримки Windows Forms (2001), Windows Presentation Foundation (WPF; 2006), Entity Framework 6.3.
- Доповнює фреймворк side-by-side and app-local deployments, швидким читачем JSON reader, serial port access and other PIN access for Internet of Things (IoT) solutions та за умовчанням ввімкненою багаторівневою (tiered) компіляцією.

- **.NET Core 3.1** – грудень 2019р.

Майбутні платформи .NET



Джерела для самоопрацювання та розвитку

- Глосарій по .NET
- Структура и модель выполнения .NET Core приложений
- Эволюция .NET-стека: что изменилось за последние несколько лет



ДЯКУЮ ЗА УВАГУ!

Наступне питання: Структура програми мовою C#.