.NET

середовище
для розробки і виконання
прикладних програм

.NET : CLR & FCL

- **Common language runtime (CLR)**
  - is the foundation of the .NET Framework
  - providing **core services** such as memory management, thread management, and remoting, while also enforcing strict type safety and other forms of code accuracy that promote security and robustness
- **Framework class library (FCL)**
  - is a comprehensive, object-oriented collection of reusable types
  - for developing apps ranging from traditional command-line or graphical user interface (GUI) apps to apps based on the latest innovations provided by ASP.NET services

What is .NET? Introduction and overview
https://learn.microsoft.com/en-us/dotnet/core/introduction

.NET Standard and implementations

- **.NET Standard**
  - a set of APIs that are implemented by the **Base Class Library** of a .NET implementation
  - APIs make up a **uniform set of contracts** that you compile your code against
  - these contracts are implemented in each .NET implementation
  - enables **portability** across different .NET implementations, effectively allowing code to run everywhere

- **.NET implementations**
  - **.NET Core** -- cross-platform implementation of .NET and designed to handle server and cloud workloads at scale, runs on Windows, macOS and Linux
  - **.NET Framework** -- original .NET implementation that has existed since 2002
  - **Mono** small runtime that powers Xamarin applications on Android, Mac, iOS, tvOS and watchOS
  - **UWP** -- Universal Windows Platform -- used for building modern, touch-enabled Windows applications and software for the Internet of Things (IoT)

.NET Framework and Core History

- **.NET Framework 1.0** (first beta)-- late 2001
  - Microsoft, Hewlett-Packard and Intel standardized Common Language Infrastructure (CLI) and C#

.....

- **.NET Core 1.0** -- June 2016
  - Visual Studio 2015 Update 3
  - **13k** APIs in .NET Standard 1.6
  - high performance and scalability
- **.NET Core 2.0** -- August 2017 (++ **2.1, 2.2** in 2018)
  - Visual Studio 2017 15.3
  - **ASP.NET Core 2.0**
  - **EF Core 2.0**
  - **32k** APIs in .NET Standard 2.0
  - new configuration system and many new other features added to make building web apps easier
  - performance have been more improved
- **.NET Core 3.0**, 3.1 – September, December 2019
  - Visual Studio 2019, C# 8
- **.NET 5.0** – November 2020 -- the next major release of .NET Core following 3.1
  - Visual Studio 2019, **C# 9**, ASP.NET Core 5.0, 6.0
- **.NET 6.0** – November 2021 **-- C# 10**
- **.NET 7.0** – November, 2022 -- Visual Studio 2022**, C# 11**,
  - ASP.NET Core 7.0

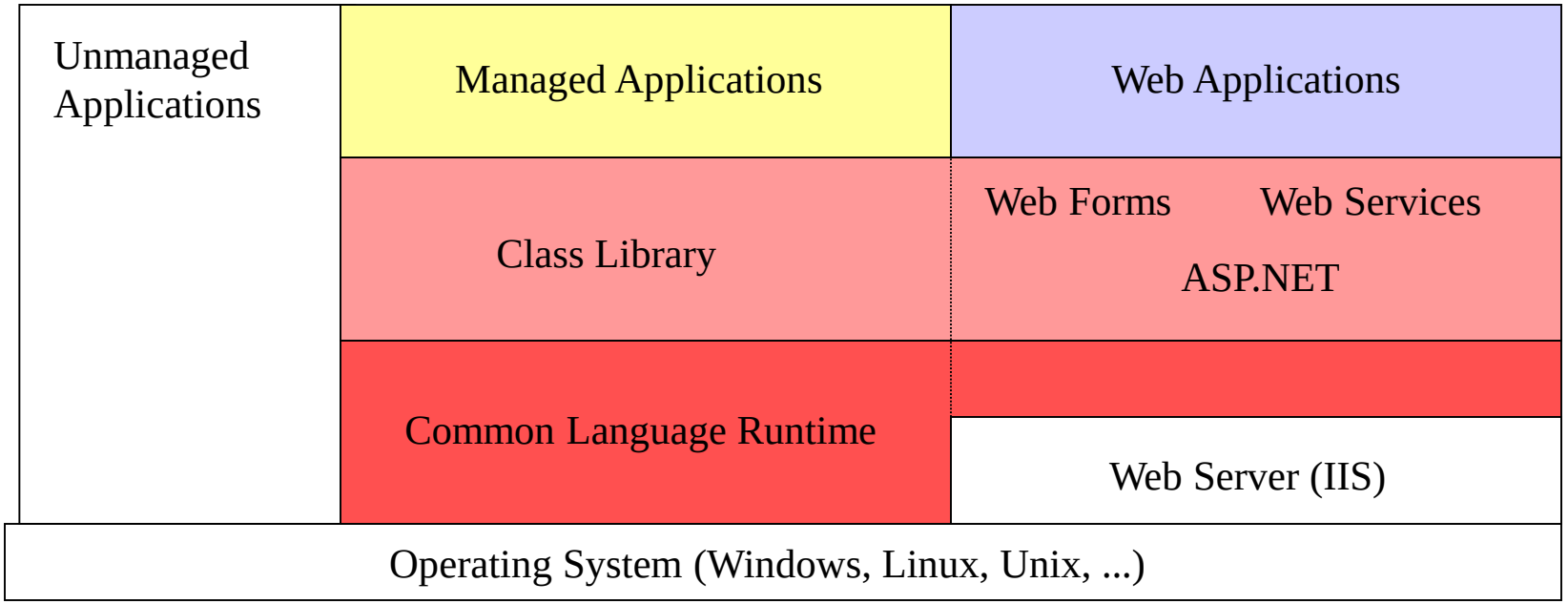
Common Language Runtime (CLR)

- **Середовище виконання програм - основа .NET Framework:**
  - Компілювання коду в Intermediate Language (MSIL)
  - just-in-time (JIT) в систему команд процесора
  - ПідтримкКомпілювання а збірок (assemblies → metadata)
  - Керування кодом під час виконання
  - Організація віддаленої взаємодії
  - Керування потоками
  - Строга перевірка типів
  - Мовна інтероперабельність
  - Виділення пам’яті
  - Збір сміття
- **.NET runtimes**
  - CLR for .NET Framework
  - CoreCLR for .NET Core
  - .NET Native for Universal Windows Platform
  - Mono runtime for Xamarin.iOS, Xamarin.Android, Xamarin.Mac, and the Mono desktop framework

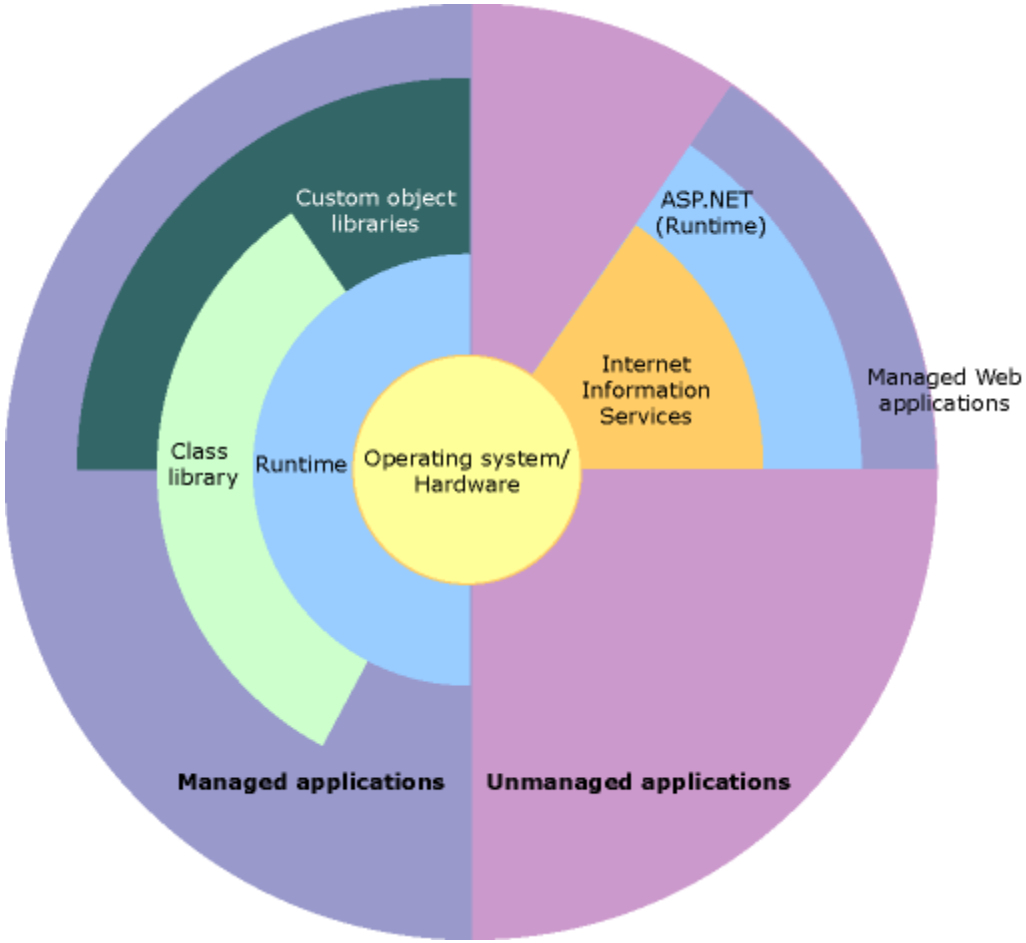
.NET class library

- *System* - низькорівневі типи
- *System.Collections* - контейнери ArrayList, SortedList, Queue, Stack
- *System.ComponentModel* - компоненти і їх контейнери
- *System.Data* - доступ до баз даних
- *System.Drawing* - GDI+
- *System.EnterpriseServices* - середовище для програм рівня підприємства
- *System.IO* - файловий ввід-вивід
- System.Math – математика
- System.Net –протоколи і сервіси мережі
- System.Reflection - RTTI
- System.Security - криптографія, захист
- System.Threading – Багатопотоковість
- System.Web - взаємодія браузер-сервер
- System.Windows.Forms - стандартні програми, форми, контроли
- System.Xml – підтримка XML

Види програм .NET Framework

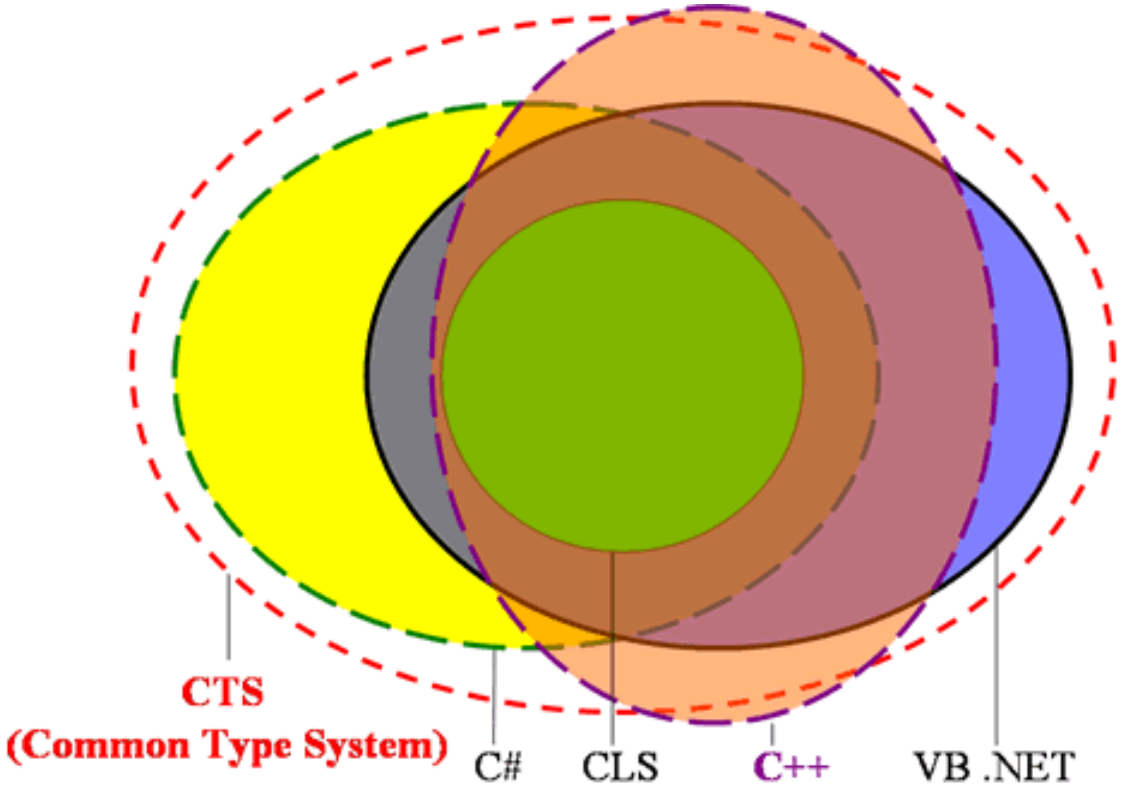


.NET Framework



CTS & CLS

- **Common Type System**, **CTS –** загальна система типів **CLR**
- **Common Language Specification, CLS**
  - загальномовна специфікація мінімального набору властивостей, які підтримують всі мови **CLR ;**
  - основа інтероперабельності



CLI & CLR

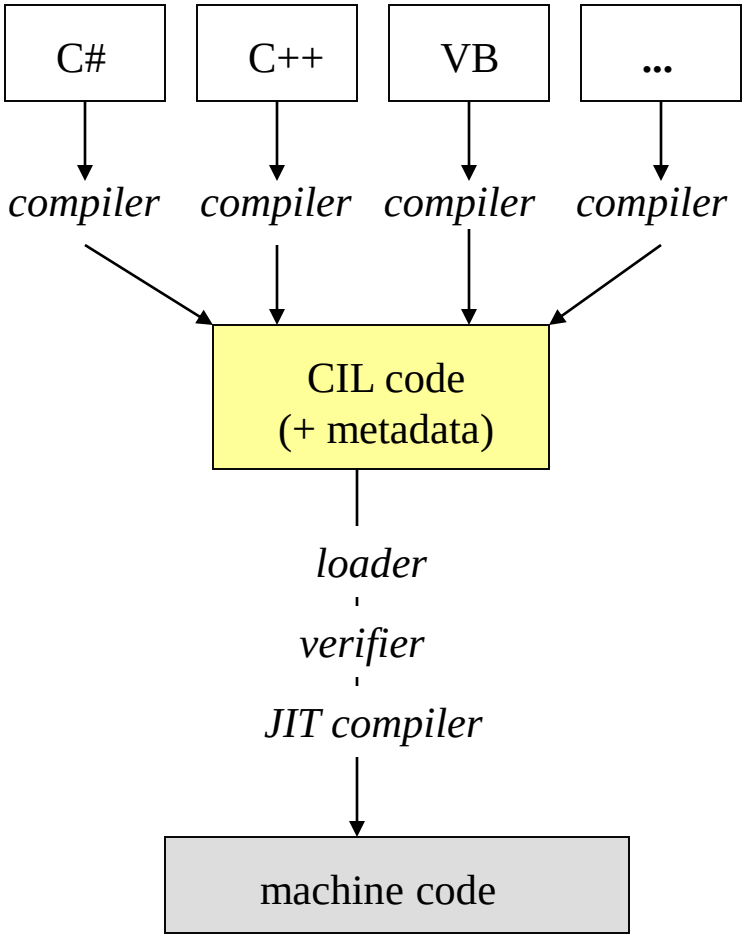
- **Common Language Infrastructure (CLI):**
  - Common Type System (CTS),
  - metadata,
  - Virtual Execution Environment (VES)
  - Intermediate Language (IL)
  - Common Language Specification (CLS)

- CLI is documented through ECMA (European Computer Manufacturer's Association)

- **CLR (Common Language Runtime)** is Microsoft's primary ***implementation*** of the CLI.

- **Managed code** is code executed by a .NET virtual machine in a Microsoft Windows environment. All other code has come to be known as **unmanaged code**.

Interoperability



C#

```
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
...
```

Приклад C# програми

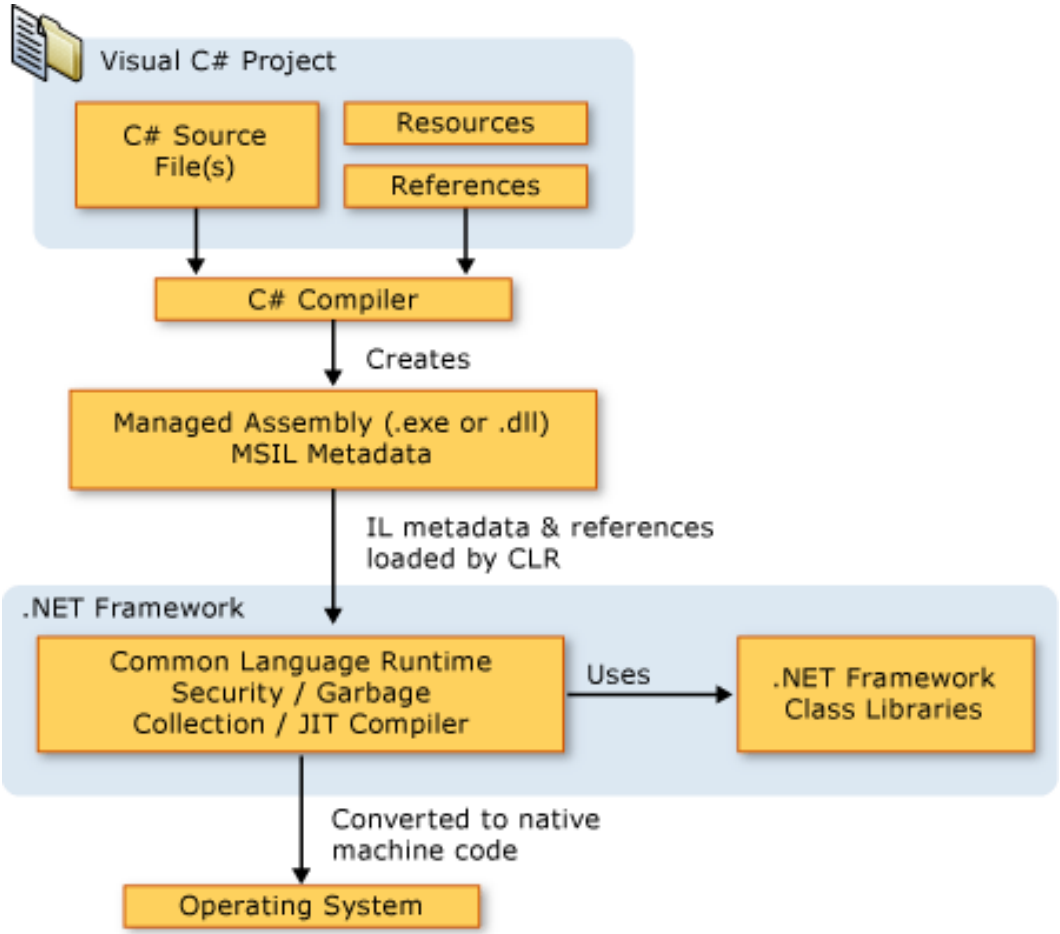
```
namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            System.Console.WriteLine("Hello, World !");
        }
    }
}
```

`Program` - новий тип

`System.Console, System.String` -- типи Microsoft, IL код яких знаходиться в MSCorLib.dll

https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/
https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions

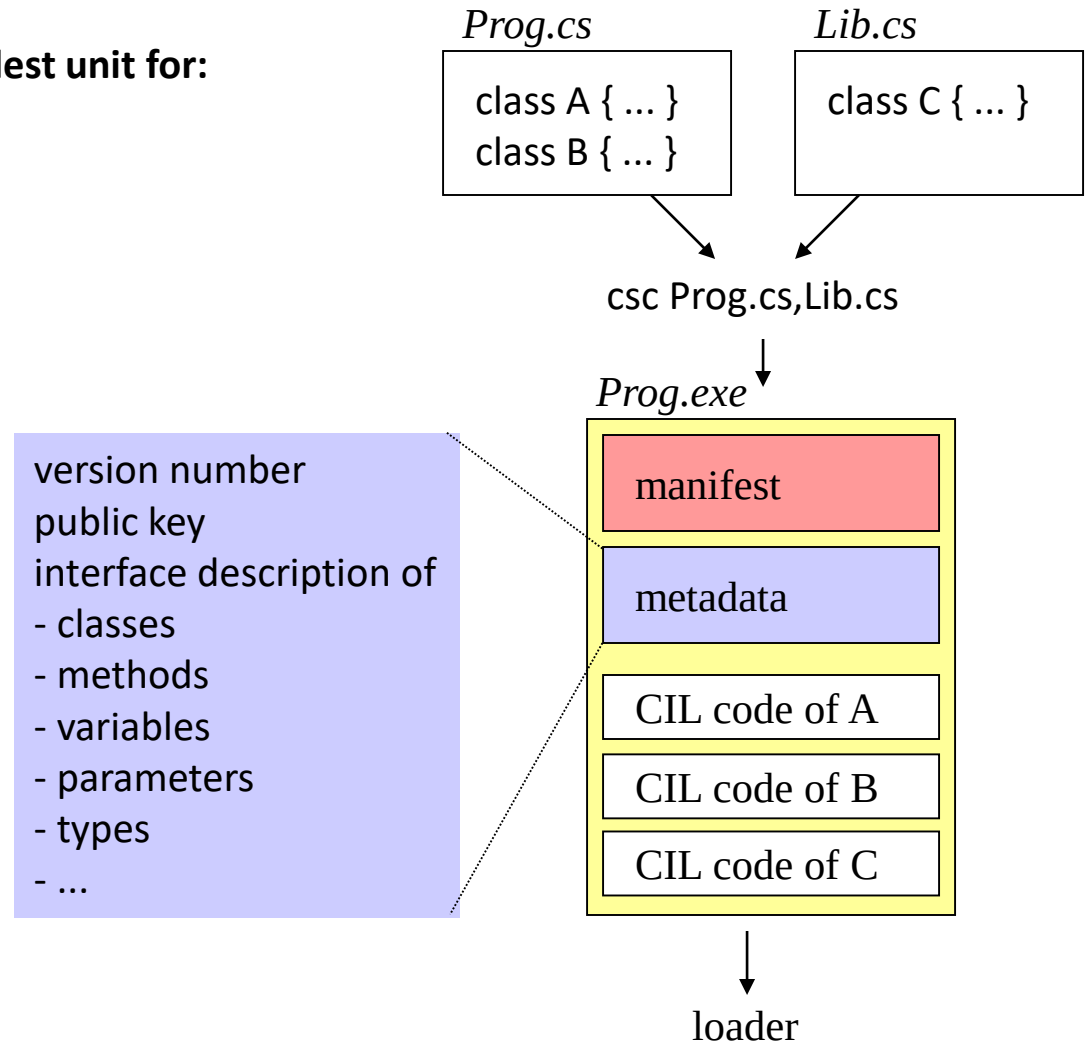
Execution of C# application



Assemblies

- **Assemblies are the smallest unit for:**
  - deployment
  - versioning
  - dynamic loading

- **Metadata is used for:**
  - dynamic loading
  - versioning
  - reflection



Структура керованого модуля

Part	Description
PE header	The standard Windows PE file header. This header indicates the type of file: GUI, CUI, or DLL, and it also has a timestamp indicating when the file was built. For modules that contain only IL code, the bulk of the information in the PE header is ignored. For modules that contain native CPU code, this header contains information about the native CPU code.
CLR header	Contains the information (interpreted by the CLR and utilities) that makes this a managed module. The header includes the version of the CLR required, some flags, the MethodDef metadata token of the managed module's entry point method (**Main** method), and the location/size of the module's metadata, resources, strong name, some flags, and other less interesting stuff.
Metadata	Every managed module contains **metadata tables**. There are two main types of tables: tables that describe the types and members ***defined*** in your source code and tables that describe the types and members ***referenced*** by your source code.
Intermediate language (IL) code	Code that the compiler produced as it compiled the source code. The CLR later compiles the IL into native CPU instructions.

Common Definition Metadata Tables

Metadata Definition Table Name	Description
ModuleDef	Module version ID (in the form of a GUID created by the compiler).
TypeDef	Includes the type's name, base type, flags (i.e., public, private, etc.) and points to the methods it owns in the MethodDef table, the fields it owns in the FieldDef table, the properties it owns in the PropertyDef table, and the event it owns in the EventDef table
MethodDef	Contains one entry for each method defined in the module. Each entry includes the method's name, flags (private, public, virtual, abstract, static, final, etc), signature, and offset within the module where IL code can be found. Each entry can also refer to a ParamDef table entry where more information about the method's parameters can be found.
FieldDef	Contains one entry for every field defined in the module. Each entry includes a name, flags (i.e., private, public, etc.), and type.
ParamDef	Contains one entry for each parameter defined in the module. Each entry includes a name and flags (in, out, retval, etc).
PropertyDef	Contains one entry for each property defined in the module. Each entry includes a name, flags, type, and backing field (which can be null).
EventDef	Contains one entry for each event defined in the module. Each entry includes a name and flags.

Common Reference Metadata Tables

Metadata Reference Table Name	Description
AssemblyRef	Contains one entry for each assembly referenced by the module. Each entry includes the information necessary to bind to the assembly: the assembly's name (without path and extension), version number, culture, and public key token (normally a small hash value, generated from the publisher's public key, identifying the referenced assembly's publisher).
ModuleRef	Contains one entry for each PE module that implements types referenced by this module. Each entry includes the module's filename and extension (without path). This table is used to bind to types that are implemented in different modules of the calling assembly's module.
TypeRef	Contains one entry for each type referenced by the module. Each entry includes the type's name and a reference to where the type can be found. If the type is implemented within another type, then the reference indicates a TypeRef entry. If the type is implemented in the same module, then the reference indicates a ModuleDef entry. If the type is implemented in another module within the calling assembly, then the reference indicates a ModuleRef entry. If the type is implemented in a different assembly, then the reference indicates an AssemblyRef entry.
MemberRef	Contains one entry for each member (fields and methods, as well as property and event methods) referenced by the module. Each entry includes the member's name and signature, and points to the TypeRef entry for the type that defines the member.

Manifest Metadata Tables

Manifest Metadata Table Name	Description
AssemblyDef	Contains a single entry if this module identifies an assembly. The **entry** includes the assembly's name (without path and extension), version (major, minor, build, and revision), culture, flags, hash algorithm, and the publisher's public key(which can be **null**).
FileDef	Contains one entry for each PE and resource file that is part of the assembly. The entry includes the file's name and extension (without path), hash value, and flags. If this assembly consists only of its own file, the FileDef table has no entries.
ManifestResourceDef	Contains one entry for each resource that is part of the assembly. The entry includes the resource's name, flags (public, private), and an index into the FileDef table indicating the file that contains the resource file or stream. If the resource isn't a stand-alone file (such as .jpeg or a .gif), the resource is a stream contained within a PE file. For an embedded resource, the entry also includes an offset indicating the start of the resource stream within the PE file.
ExportedTypesDef	Contains one entry for each public type exported from all the assembly's PE modules. The entry includes the type's name, an index into the FileDef table(indicating which of this assembly's files implements the type), and an index into the TypeDef table. *Note*: To save file space, types exported from the file

Поняття про типи даних

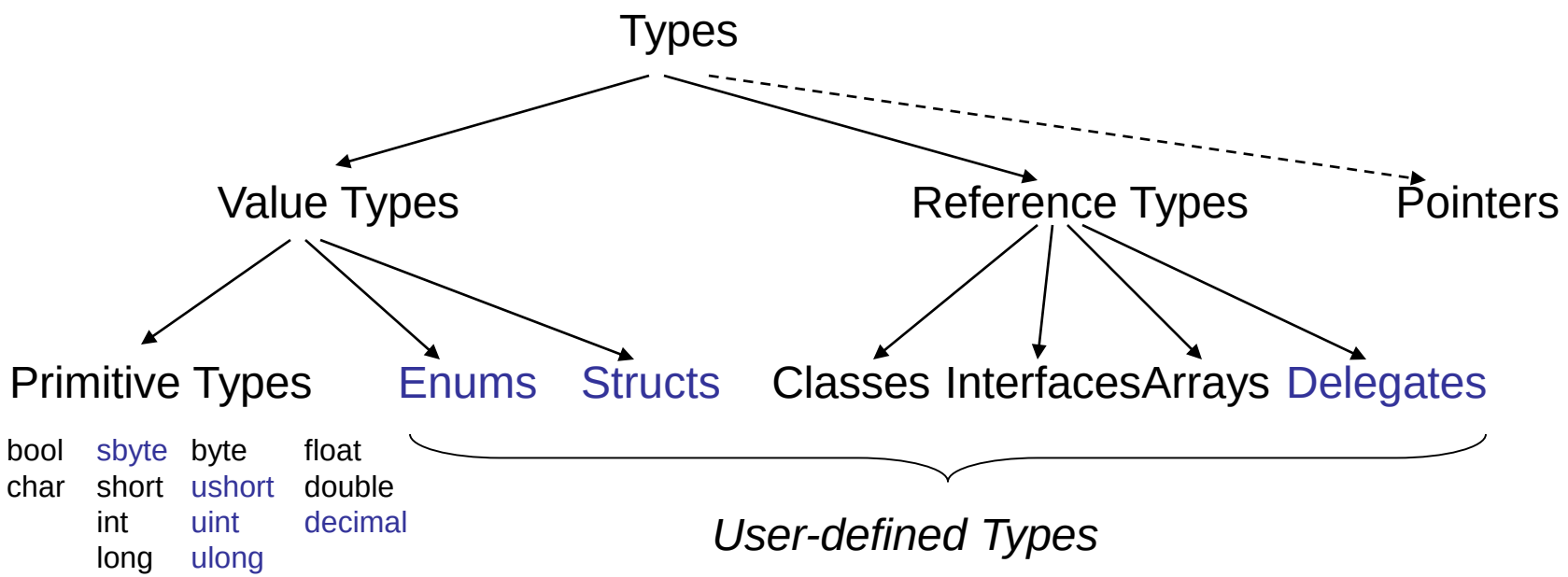
- Тип – *фундаментальне* поняття у програмуванні
  C# - сильно типізована мова.
- Типи у програмі *ідентифікуються* назвою
  - одне з ключових слів C#(*вбудовані* типи): ***bool, int, char***...
  - ідентифікатор (*бібліотечні* або типи *користувача,* визначені засобами class або struct): ***string,*** ....
- ***Тип*** – сутність, якій притаманні такі атрибути:
  1. конкретна структура даних у оперативній пам'яті:
     - обсяг пам'яті, яку займають об'єкти
     - спосіб представлення
  2. множина операцій, яка задає функціональність, застосовну до об'єктів
  3. інтерпретація операцій, як саме вони відбуваються

https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/index

Categories of types in CTS

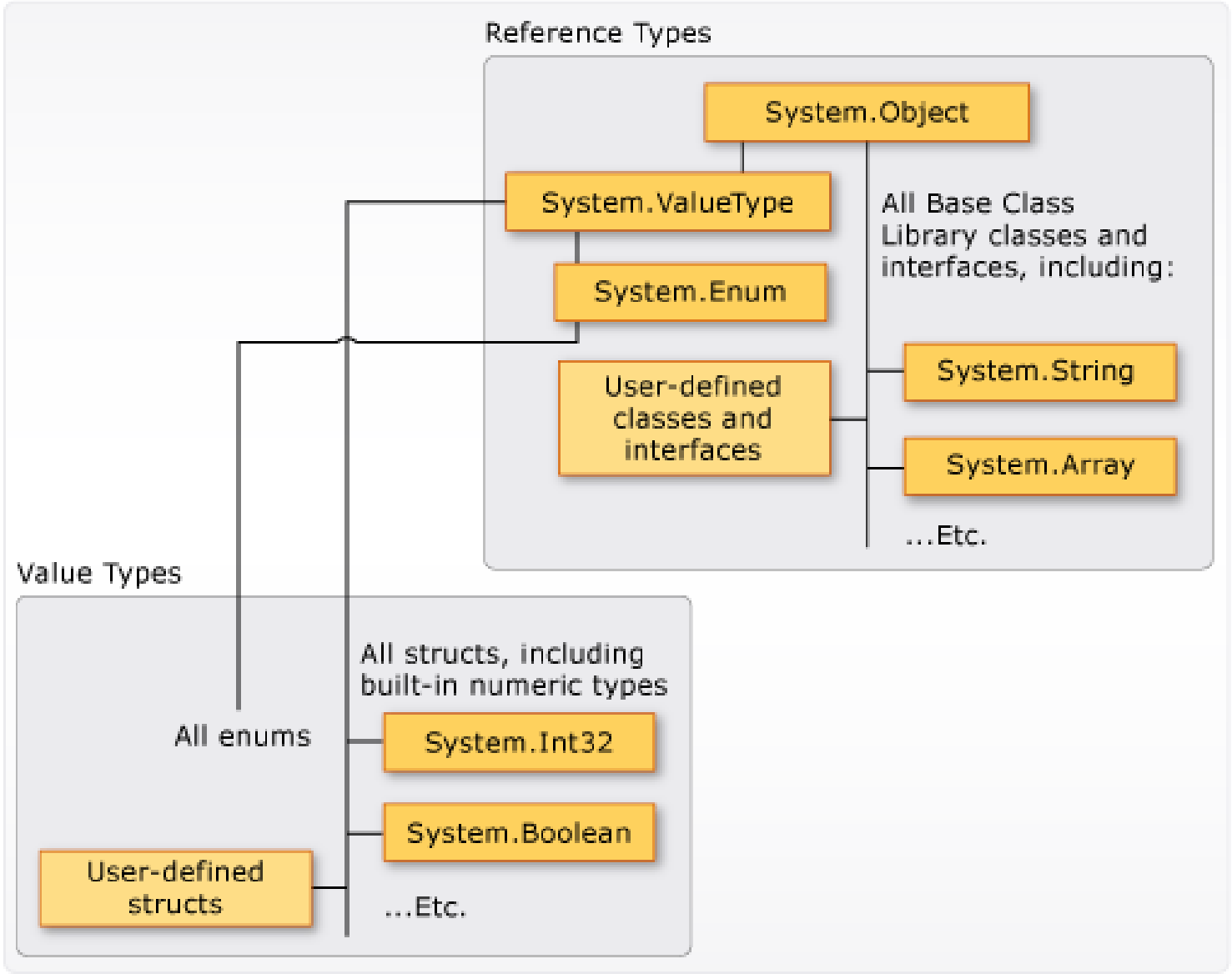
- Class -- reference type:
  - can be derived directly from another class or implicitly from System.Object
  - can implement any number of interfaces, but it can inherit from only one base class
  - defines operations that can perform (methods, events, or properties) and the data that object contains (fields)
  - generally includes both definition and implementation but it can have one or more operations that have no implementation (abstract class)
- Structure -- value type:
  - derived implicitly from System.ValueType (is derived from System.Object)
  - useful for representing values whose memory requirements are small
  - all primitive data types (Boolean, Byte, Char, DateTime, Decimal, Double, Int16, Int32, Int64, SByte, Single, UInt16, UInt32, and UInt64)
- Enumeration (enum) -- value type:
  - inherits directly from System.Enum
  - supplies alternate names for the values of an underlying primitive type
- Interface -- specifies a "can do" relationship or a "has a" relationship such as comparing and sorting (IComparable and IComparable<T>), testing for equality (the IEquatable<T>) or enumerating items in a collection (IEnumerable and IEnumerable<T>)
- Delegate -- serve a purpose similar to that of function pointers in C++
- Records -- a reference type that provides synthesized methods to provide value semantics for equality. Records are immutable by default.

Uniform Type System



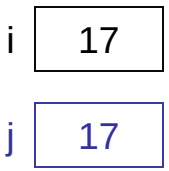
- All types are compatible with *object*
  - can be assigned to variables of type *object*
  - all operations of type *object* are applicable to them
- Implicit types
- Anonymous types
- Nullable value types (int? is an int type that can also have the value null)

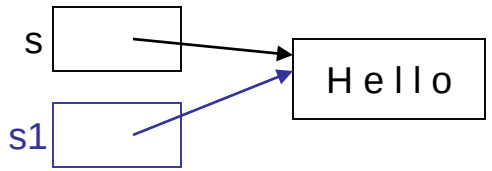
Relationship between value types and reference types in the CTS



Value Types and Reference Types

	Value Types	Reference Types
variable contains	value	reference
stored on	stack (or in an object)	heap
initialization	0, false, '\0'	null
assignment	copies the value	copies the reference
example	`int i = 17;` `int j = i;`	`string s = "Hello";` `string s1 = s;`






```
int[ ] a1 = new int[10];
int[,] a2 = new int[10, 5];
int[,,] a3 = new int[10, 5, 2];
```

System.Object

- **Public Methods**

`bool Equals()`	Посилання - чи вказують точно на один і той же об'єкт Значення – чи типи об'єктів ідентичні і значення рівні
`int GetHashCode()`	Повертає хеш-код об'єкта для розміщення в хеш-таблиці з метою підвищення продуктивності
`Type GetType()`	Для отримання інформації про тип
`string ToString()`	За замовчуванням повертає назву об'єкта, в похідних класах може перевизначатися для текстового представлення об'єкта

- **Protected Methods**

`void Finalize()`	Може викликатися при звільненні пам'яті, яку займав об'єкт
`Object MemberwiseClone()`	Копіювання посилання (а *shallow copy)* без утворення дійсної копії *(* а *deep copy)* об'єкта в купі

CTS Types

CTS Type Name	C# Alias	Description	Range
`System.Object`	`object`	Base class for all CTS	
`System.SByte`	`sbyte`	Signed 8-bit byte	[-128, 127]
`System.Byte`	`byte`	Unsigned 8-bit byte	[0, 255]
`System.Int16`	`short`	Signed 16-bit value	[-32768, 32767]
`System.UInt16`	`ushort`	Unsigned 16-bit value	[0, 65535]
`System.Int32`	`int`	Signed 32-bit value	[-2^{31}, $2^{31}-1$]
`System.UInt32`	`uint`	Unsigned 32-bit value	[0, 4 294 967 295]
`System.Int64`	`long`	Signed 64-bit value	[-2^{63}, $2^{63}-1$]
`System.UInt64`	`ulong`	Unsigned 64-bit value	[0, 18 446 744 073 709 551 615]
`System.Single`	`float`	IEEE 32-bit float	[1.4×10^{-45} , 3.4×10^{38}] 6-7
`System.Double`	`double`	IEEE 64-bit float	[5.0×10^{-324} , 1.7×10^{308}] 15-16
`System.Decimal`	`decimal`	128-bit data	[1.0×10^{-28} , 7.9×10^{28}] 28-29
`System.Boolean`	`bool`	Boolean value	*true* / *false*
`System.Char`	`char`	16-bit Unicode char	[u+0000, u+ffff] '\u0058'
`System.String`	`string`	String	

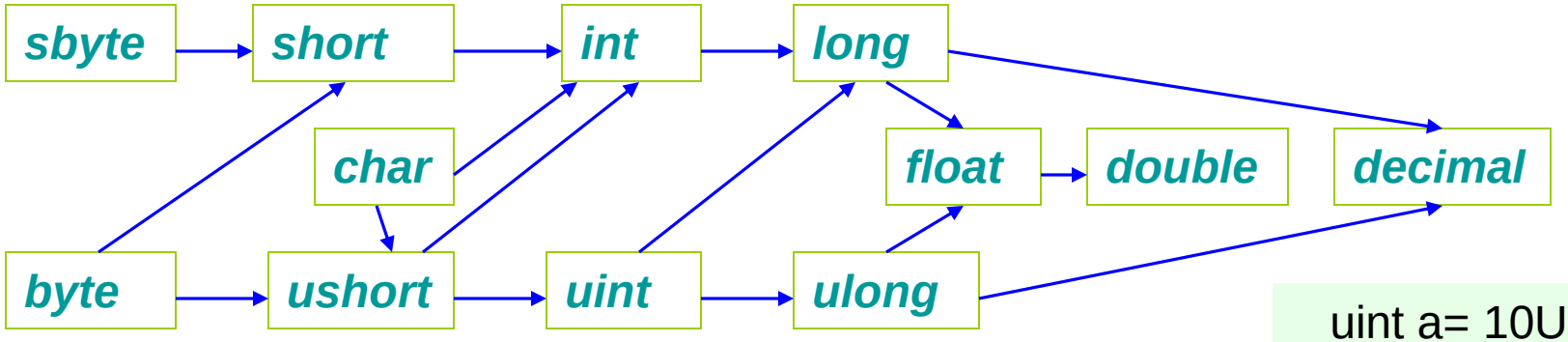
struct Char

```
[SerializableAttribute]
[ComVisibleAttribute(true)]
public struct Char : IComparable, IConvertible,
                     IComparable<char>, IEquatable<char>
```

Неявні перетворення типів

Неявні перетворення (допустиме використання замість очікуваного типу):

- просування вбудованих типів




```
uint a= 10U;
ulong b = a;

// void f(ulong x){…}
f(a);
```

- пакування (boxing) вбудованих типів

```
int foo = 42;          // Value type.
object bar = foo;      // foo is boxed to bar.
```

- підстановка замість об’єкта базового класу (upcasting)

```
//  class B{…}      class D: B{…}      void f(B x){…}
    B x = new B();     f( x );
    D y = new D();     f( y );
```

Явні перетворення типів

Явні перетворення (контроль наслідків некоректних перетворень!!!) :

- приведення(casting) вбудованих типів

```
long a= 10L;
int b = (int) a;
```

- пакування (boxing) вбудованих типів

```
int foo = 42;              //  Value type
object bar = foo;          //  foo is boxed to bar
int foo2 = (int)bar;       //  Unboxed back to int
```

- приведення(downcasting) базового класу до похідного

```
//  class B{…}      class D: B{…}
  B x = new B();      D y = (D)x; // System.InvalidCastException при недоступності
  D z = x as D;  if (z == null) {...} // System.NullReferenceException при z.method()
```

- перетворення (parsing) рядка до числового типу

```
string s=”123456”;
long a = long.Parse(s);
```

Простори назв

- Глобальний неіменований простір назв
- Визначення простору назв
  - послідовно в одному модулі
  - вкладені з необхідною глибиною

namespace *ідентифікатор* {
визначення просторів назв
визначення типів
}

- Використання простору назв
  - **using** перед просторами назв стосується всіх класів у модулі компіляції
  - **using** всередині простору назв стосується класів лише цього простору

using *специфікований ід простору назв* ;

- Використання аліясів (псевдонімів)

using *аліяс* = *специфікований ід простору назв* ;
using *аліяс* = *специфікований ід класу* ;

```
using  System;

namespace A
{
    namespace B
    {
        class X{ public int mx; …}
     }
     class Y{ public int my; …}
}
namespace C
{
     class Z{  public int mz; …}
}
```

```
using  A;   using  A.B;
using  D=A.B.X;
…{  … X.mx   … Y.my
       A.B.X.mx… D.mx
   } …
```

Загальна схема визначення типів

атрибути $_{opt}$ *модифікатори*$_{opt}$
class *ідентифікатор* : *ідентифікатор базового класу*$_{opt}$, *список інтерфейсів* $_{opt}$
{
визначення членів класу
}

атрибути $_{opt}$ *модифікатори*$_{opt}$
struct *ідентифікатор*: *список інтерфейсів* $_{opt}$
{
визначення членів класу
}

атрибути $_{opt}$ *модифікатори*$_{opt}$
interface *ідентифікатор*: *список інтерфейсів* $_{opt}$
{
оголошення членів інтерфейсу
}

Модифікатор:

- **new, abstract, sealed**
- *модифікатори доступу:* **public, protected, internal, private**