

Academy Week 1




Antonia Sacchitella

Analyst@icubedsrl

Antonia.sacchitella@icubed.it

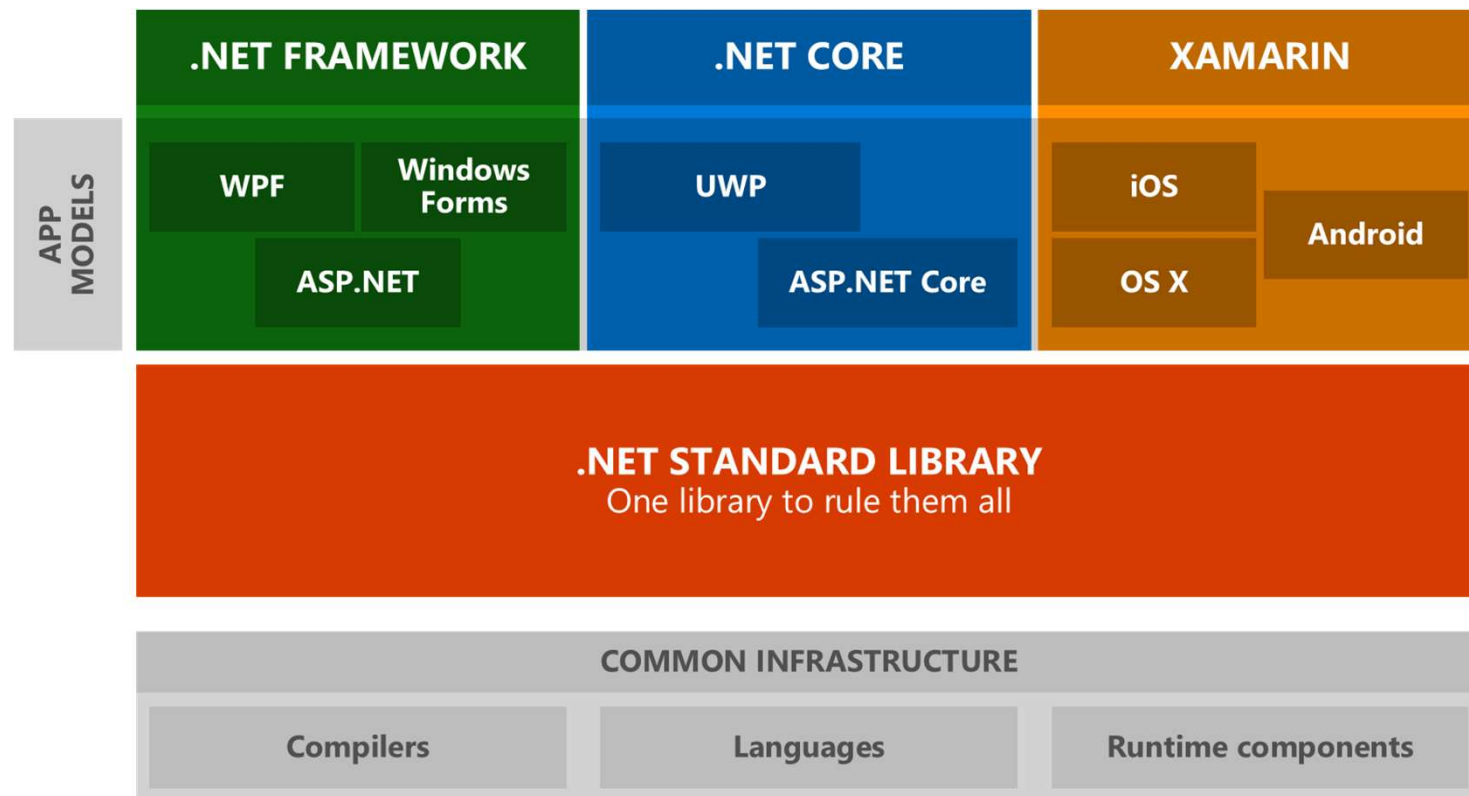


Week 1 - Agenda

- Introduzione a .NET
- Application Lifecycle Management (ALM)
- Linguaggio C#
- Introduzione ai Design Pattern
- Cenni di Test Driven Development (TDD)
-  Esercitazione

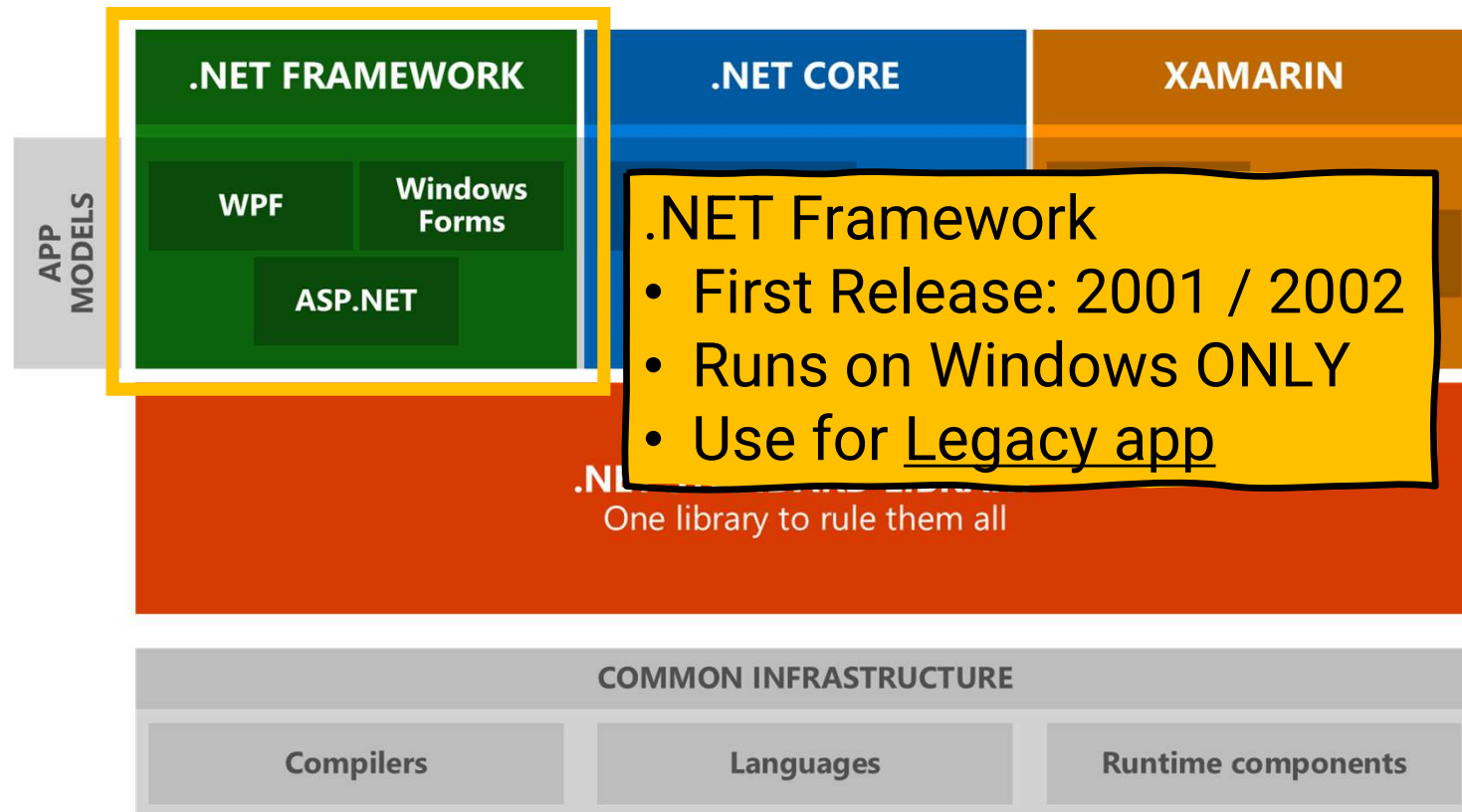


.NET

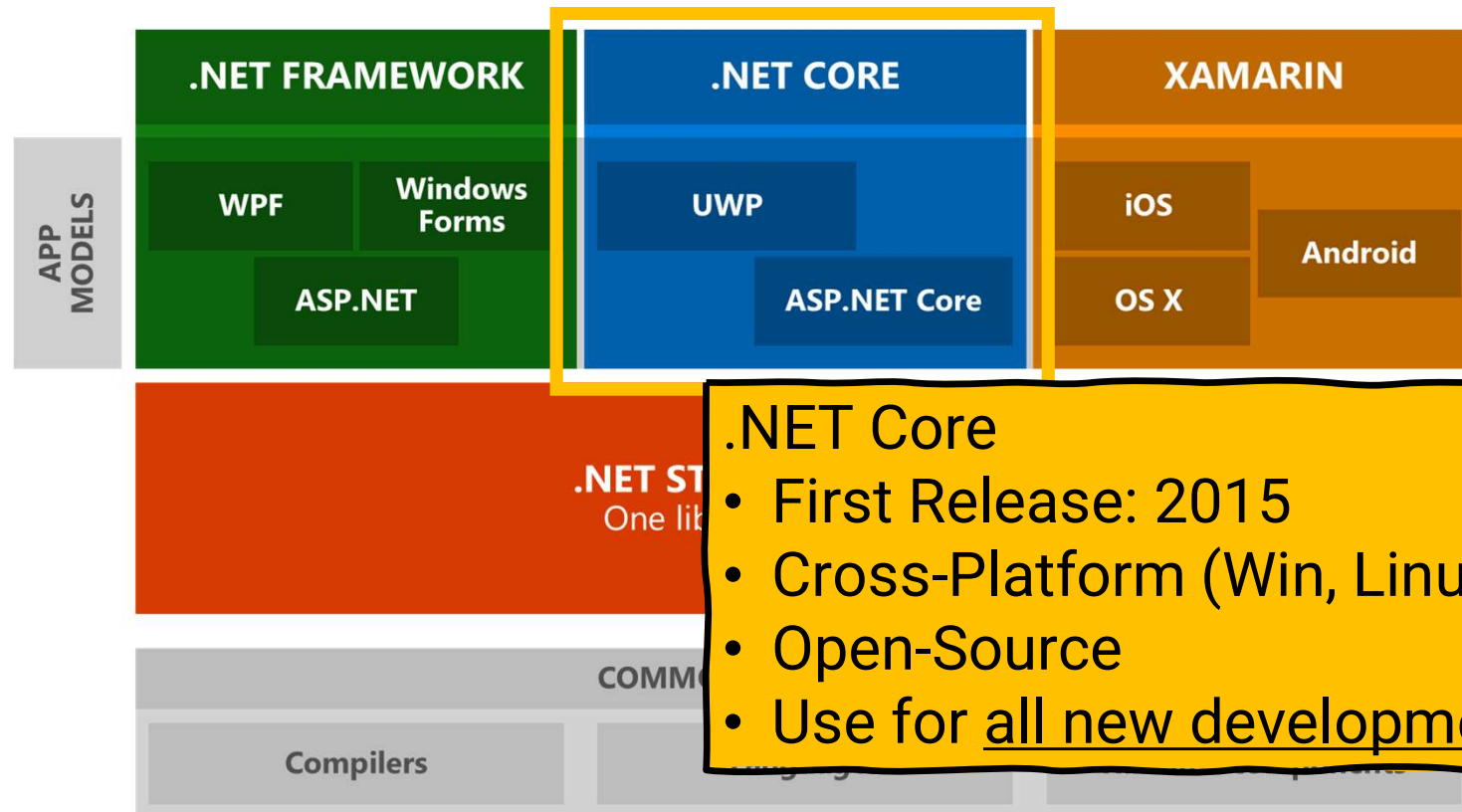


.NET

.NET

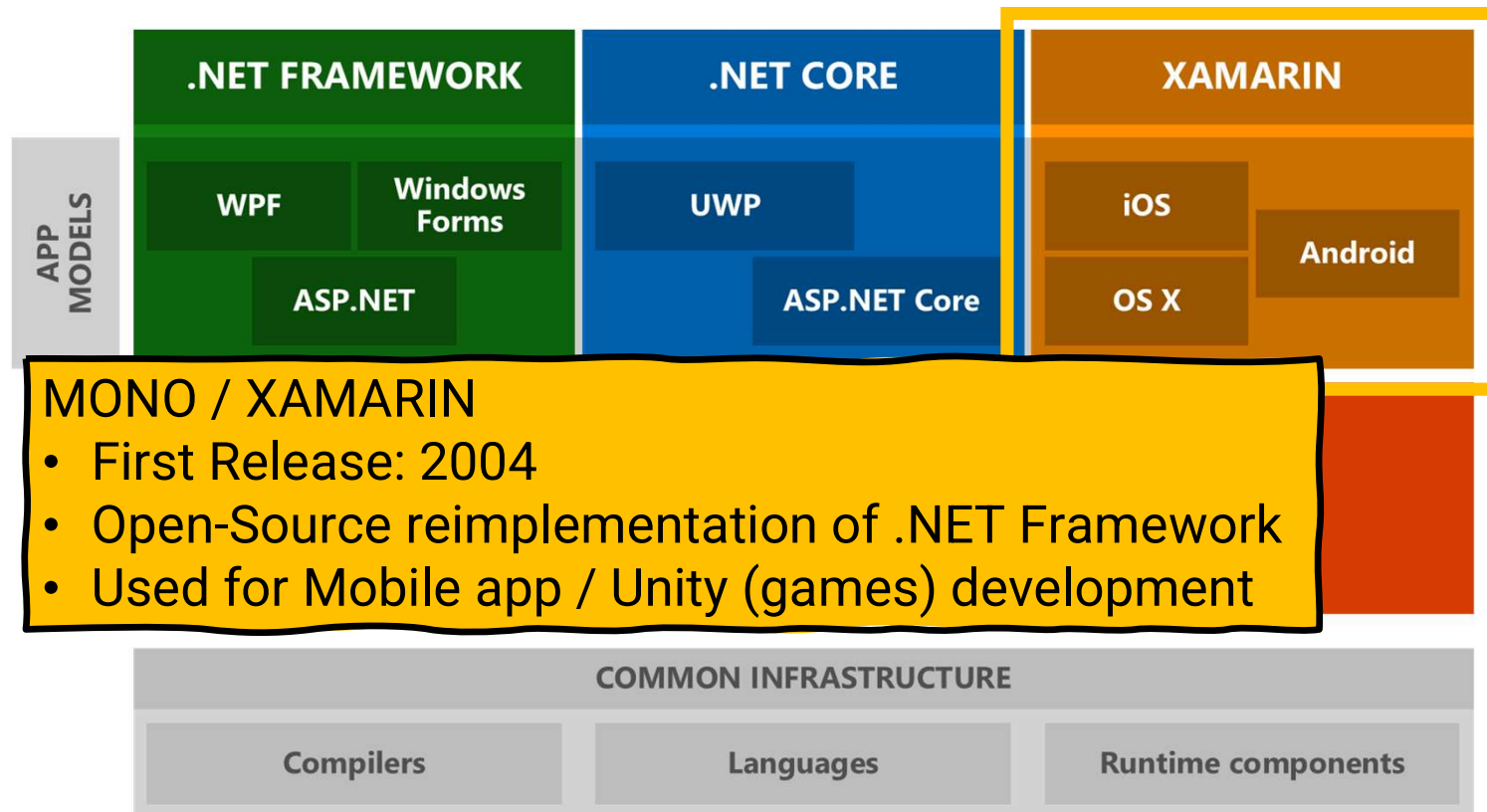


.NET



.NET

.NET

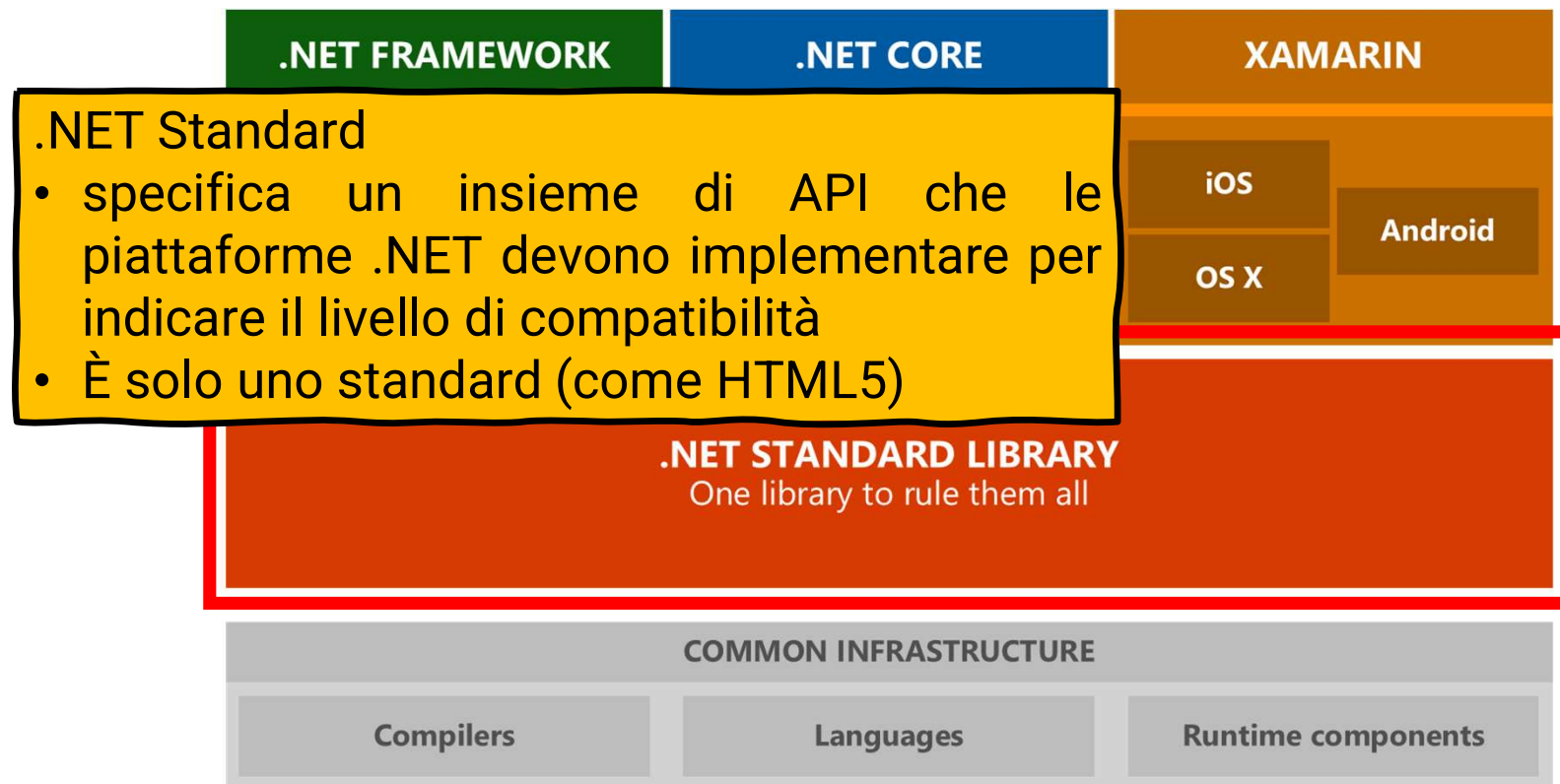


MONO / XAMARIN

- First Release: 2004
- Open-Source reimplementation of .NET Framework
- Used for Mobile app / Unity (games) development

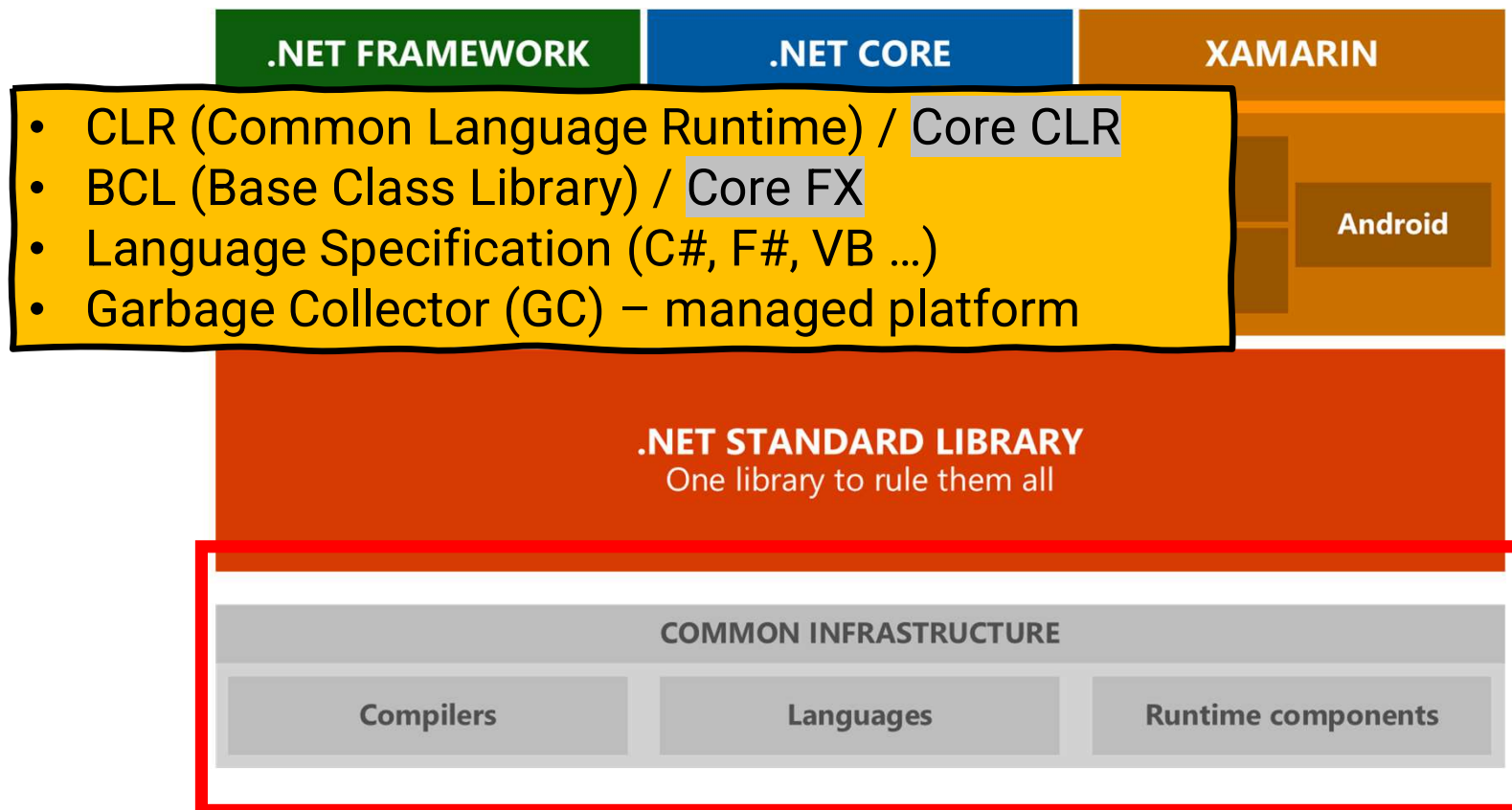
.NET

.NET



.NET

.NET



.NET



.NET

.NET FRAMEWORK

.NET CORE

XAMARIN

- Il codice sorgente viene convertito in **Intermediate Language (IL)** e memorizzato in un assembly (un file DLL o EXE)
- In fase di esecuzione, il CLR carica l'IL ed un compilatore just-in-time (JIT) lo compila in istruzioni CPU native, che vengono eseguite dalla CPU sulla macchina

COMMON INFRASTRUCTURE

Compilers

Languages

Runtime components

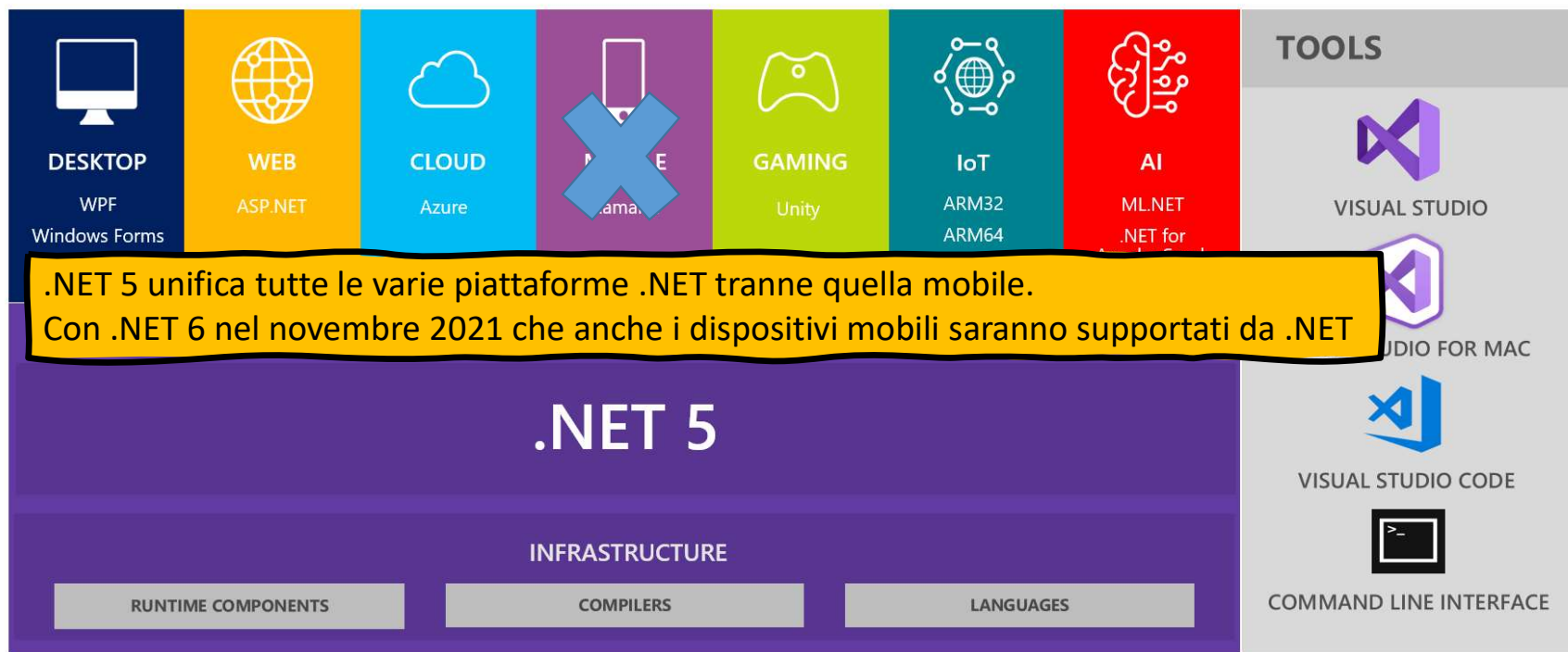
.NET 5 (Nov 2020) and beyond

.NET

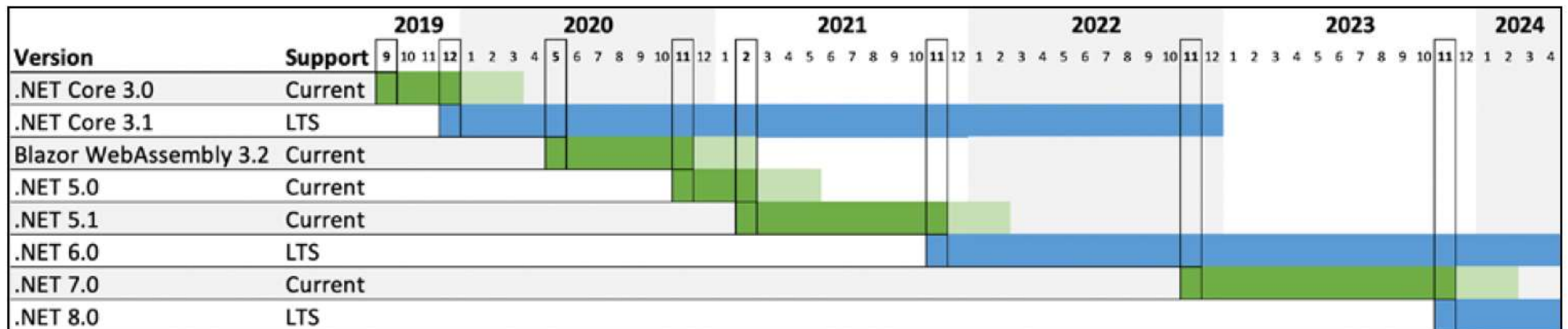


.NET 5 (Nov 2020) and beyond

.NET



.NET 5 (Nov 2020) and beyond



Demo

.NET Core CLI



C#



- Value types VS Reference Types
- Classi e interfacce (classi astratte, enum)
- Incapsulamento, Ereditarietà, Polimorfismo
- Eccezioni
- Gestione degli eventi, delegates
- Lettura e Scrittura su File
- Multithreading
- LINQ e Lambda



Value Type

Contengono direttamente il dato nell'ambito dello stack del thread.
Una copia di un Value Type implica la copia dei dati in esso contenuti.
Le modifiche hanno effetto solo sull'istanza corrente.
Contengono sempre un valore (null **non è** direttamente ammesso).

I Value Type comprendono:

- i tipi primitivi come int, byte, bool, ecc.
- **enum, struct** (definiti dall'utente).

```
int i = 1;  
bool b = true;  
double d = 0d;  
DateTime a = DateTime.Now;
```

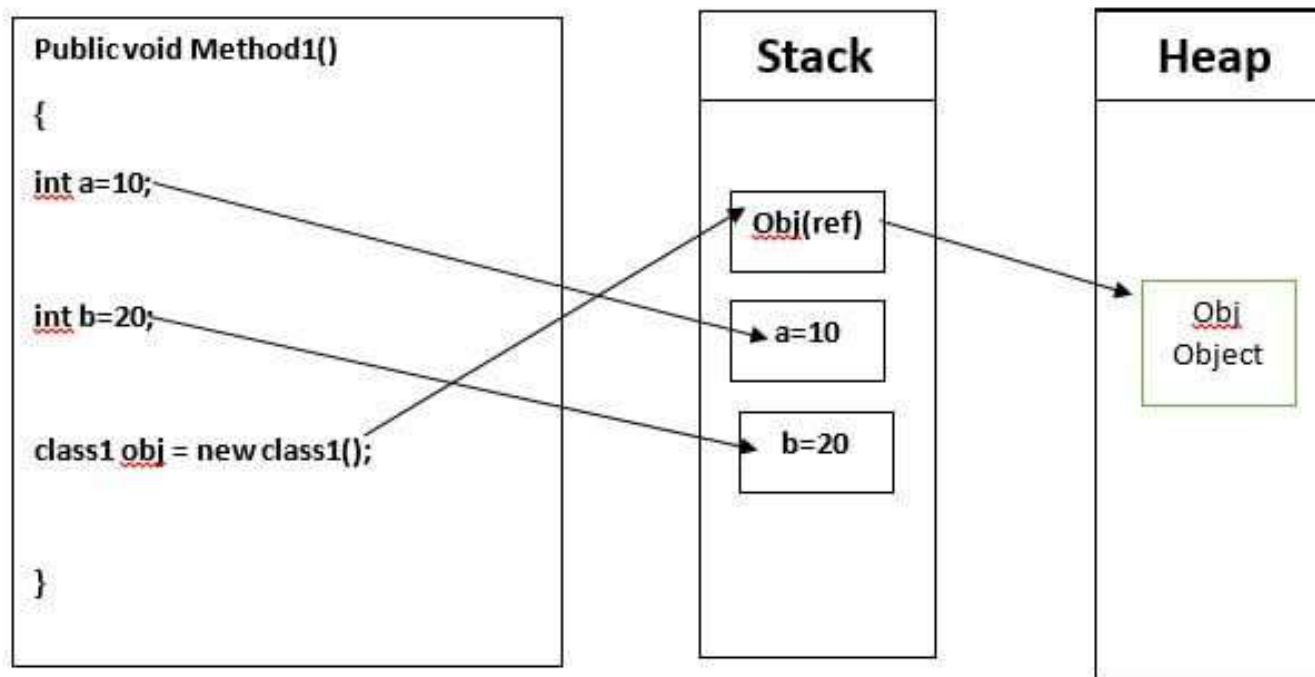
Reference Type



- Contengono solo un riferimento ad un oggetto nell'ambito dell'heap
- La copia di un Reference Type implica la duplicazione del solo reference
- Le modifiche su due reference modificano l'oggetto a cui puntano
- Il reference che non referencia nessuna istanza vale **null**
- **Tutte le classi sono Reference Type!**

```
//Attenzione: il tipo string è un caso particolare perché è immutabile.  
string s = "C#";  
DataSet ds = New DataSet();  
Person p = New Person();
```


Value Type & Reference Type



Enumerazioni



Un **enum** è una "classe" speciale che rappresenta un gruppo di costanti (variabili non modificabili / di sola lettura).

```
public enum TimeOfDay
{
    Morning = 0,
    Afternoon = 1,
    Evening = 2
}
```

Possiamo **accedere ad un valore** utilizzando:

```
var timeOfDay = TimeOfDay.Morning;
```

Classi



Una classe è come un costruttore di oggetti o un "blueprint" per la creazione di oggetti.

```
public class MyClass {  
    //...  
}
```

Una classe può contenere ed eventualmente esporre una sua interfaccia:

- Dati (**campi** e **proprietà**)
- Funzioni (**metodi**)

Classi, campi e proprietà



Campo

```
public class MyClass
{
    public string Name;
}

MyClass c = new MyClass();
c.Name = "C#";
Console.WriteLine(c.Name);
```

Classi e proprietà



Proprietà

```
public class MyClass
{
    public string Name { get; set; }
}

MyClass c = new MyClass();
c.Name = "C#";
Console.WriteLine(c.Name);
```

Classi e proprietà



Proprietà 'condensata'

```
public class MyClass
{
    public string Name { get; set; }
}

MyClass c = new MyClass();
c.Name = "C#";
Console.WriteLine(c.Name);
```

Proprietà tradizionale

```
public class MyClass
{
    private string _name;

    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }
}

MyClass c = new MyClass();
c.Name = "C#";
```

Classi e proprietà



- È il modo migliore per soddisfare uno dei pilastri della programmazione OOP: *incapsulamento*
- Una proprietà può provvedere accessibilità in lettura (**get**) scrittura (**set**) o entrambi
- Si può usare una proprietà per ritornare valori calcolati o eseguire una validazione



Metodi di una classe

- In sostanza la dichiarazione di un metodo è composta di:
 - zero o più keyword
 - il tipo di ritorno del metodo oppure **void**
 - il nome del metodo
 - l'elenco dei parametri tra parentesi tonde
- La *firma (signature)* di un metodo è rappresentata dal nome, dal numero dei parametri e dal loro tipo; il valore ritornato **non** fa parte della firma.

```
void MyMethod(string str) {  
    // ...  
}
```

```
int MyMethod(string str) {  
    int a = int.Parse(str);  
    return a;  
}
```


Accessibilità



- I tipi definiti dall'utente (classi, strutture, enum) e i membri di classi e strutture (campi, proprietà e metodi) possono avere accessibilità diversa (*accessor modifier*):
 - **public** Accessibile da tutte le classi
 - **protected** Accessibile solo dalle classi derivate
 - **private** Non accessibile dall'esterno
 - **internal** Accessibile all'interno dell'assembly
 - **internal protected** Combinazione delle due
- Differenziare l'accessibilità di un membro è fondamentale per realizzare l'*incapsulamento*.
- L'insieme dei membri esposti da una classe rappresenta la sua *interfaccia*.

Ereditarietà



- Si applica quando tra due classi esiste una relazione “è un tipo di”. Esempio: **Customer** è un tipo di **Person**.
- Consente di specializzare e/o estendere una classe.
- Si chiama *ereditarietà* perché la classe che deriva (**classe derivata**) può usare tutti i membri della classe ereditata (**classe base** – keyword **base**) come se fossero propri, ad eccezione di quelli dichiarati privati.

```
public class Person {  
    protected string name;  
}  
  
public class Customer : Person {  
  
    public void ChangeName(string newName) {  
        base.name = newName;  
    }  
}
```

Polimorfismo

- Il *polimorfismo* è la possibilità di trattare un'istanza di un tipo come se fosse un'istanza di un altro tipo.
- Il polimorfismo è subordinato all'esistenza di una relazione di derivazione tra i due tipi.
- Affinchè un metodo possa essere polimorfico, deve essere marcato come **virtual** o **abstract**.



```
public class Strumento
{
    public virtual void Accorda() { }
}

public class Violino : Strumento
{
    public override void Accorda()
    {
        base.Accorda();
    }
}

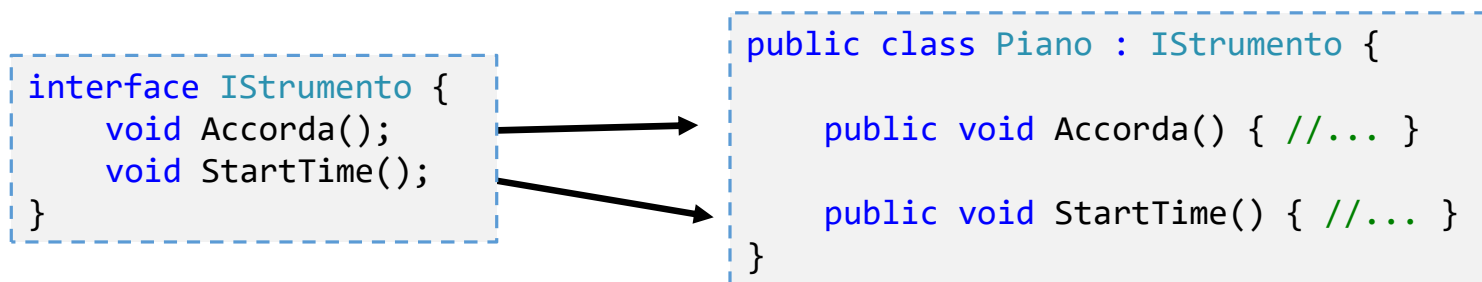
public class Orchestra
{
    public Strumento violino, chitarra, pianoforte;

    public Orchestra()
    {
        violino = new Violino();
        violino.Accorda();
    }
}
```

Interfacce



- Un'interfaccia definisce un contratto che la classe che la implementa deve rispettare
- Un'interfaccia è priva di qualsiasi implementazione e di modificatore di accessibilità (**public**, **private**, ecc.)
- Una classe può implementare più interfacce contemporaneamente.



Esercitazione 1

Realizzare una gerarchia di classi per rappresentare forme geometriche:

- Tutte le classi avranno una proprietà Nome (stringa), un metodo per il calcolo dell'area e un metodo che disegni la forma (è sufficiente stampare nella console i dettagli delle proprietà e dell'Area)
- Realizzare le classi che rappresentano:
 - Un cerchio, con le proprietà aggiuntive per le coordinate del centro (int) e per il Raggio (tutte double)
 - Un rettangolo, con le proprietà aggiuntive per Larghezza e Altezza (tutte double)
 - Un triangolo, con le proprietà aggiuntive per Base e Altezza (double)
- Tutte le classi dovranno implementare la propria versione del metodo di calcolo dell'area, perimetro e di disegno

Per verificare il corretto funzionamento delle classi realizzate, creare una istanza di ognuna nel metodo Main e visualizzare il risultato dell'esecuzione dei metodi di calcolo dell'area e di disegno.



Esercitazione 2

Riprendere l'esercizio precedente:

- Aggiungere una interfaccia IFileSerializable che include i metodi
 - SaveToFile, con un parametro fileName (string)
 - LoadFromFile con un parametro fileName (string)
- Implementare l'interfaccia su tutte le classi realizzate

Per verificare il corretto funzionamento delle classi realizzate, creare una istanza di ognuna nel metodo Main (usando IFileSerializable come tipo), e visualizzare il risultato dell'esecuzione dei metodi SaveToFile e LoadFromFile.

**I metodi devono solo stampare un messaggio a video.
Li implementeremo veramente in seguito.**

