



Service Web .Net

Cours n°1 : Introduction

Pierre-Loïc CHEVILLOT – Sébastien BEREIZIAT

Pierre-loic.chevillot@capgemini.com – sebastien.bereiziat@capgemini.com

Plan

- Généralités
 - Langage C#
 - Environnement .Net
 - Architecture
- Syntaxe & Coding Guidelines



Généralités

Microsoft .Net

- Ensemble de produits & technologies
- But : rendre les applications facilement portable sur internet

Basé sur :

- OS : Windows
- Framework : .net framework
 - Protocoles de communications
 - Ensemble de bibliothèque
 - Langages : C#, VB, F#, ASP
- Bibliothèque logicielle compatible
- Environnement d'exécution : CLI
- Gestion de compilation : MSBuild (Roselyn)
- IDE : VisualStudio (Community, Pro, Entreprise)
, VSCode

C#

- Langage de programmation orientée objet
- Commercialisé depuis 2002
- Dérivé du C++
- Langage de développement principal sur la plateforme .Net
- Typage fort
- Tout les types sont objets
- Garbage-Collector
- Système de gestion d'exceptions

Version

C# Version	.NET Framework	CLR Version	Major Features and Enhancement
C# 1.0	1.0	1.0	Class, Structs, Interfaces, Events, Properties, Delegates, Expressions, Statements, Attributes, Literals
C# 1.2	1.1	1.1	Enhancements and call of Dispose on an IEnumerator for-each call.
C# 2.0	2.0	2.0	Generics, Partial types, Anonymous Methods, Iterators, Nullable types, Getter/Setter separate accessibility, Delegates, Static classes, Delegate inference.
C# 3.0	3.5	2.0	Implicitly typed local variables, Object and collection initializers, Auto-Implemented properties, Anonymous types, Extension methods, Query expressions, Lambda expressions, Expression trees, Partial methods.
C# 4.0	4.0	4.0	Dynamic binding, Named and optional arguments, Covariance and Contravariance for generic delegate and interfaces, Embedded interop types (NoPIA)
C# 5.0	4.5	4.0	Asynchronous methods support and Caller info attributes.

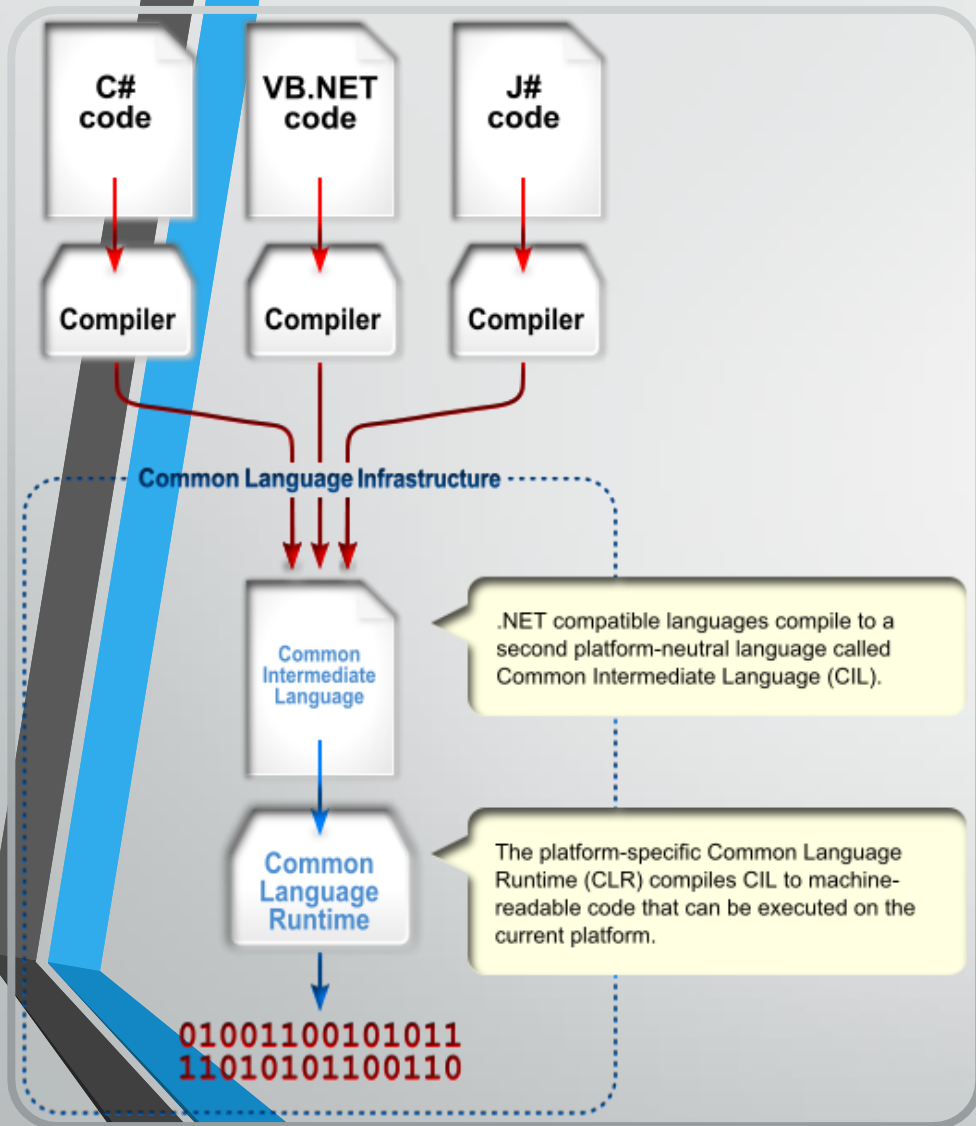
Version

C# 6.0	4.6	4.0	Compiler-as-a-Service (Roslyn), Import of static type members into the namespace, Exception filters, Await in catch/finally blocks, Auto property initializer, the default value for a property getter, Expression-bodied members, Null propagator (Null-conditional operator, succinct null checking), string interpolation, new nameof operator and Dictionary initializers.
C# 7.0	4.6.2	4.0	Out variables ,Pattern matching ,Tuples ,Deconstruction ,Discards ,Local Functions ,Binary Literals ,Digit Separators ,Ref returns and locals ,Generalized async return types ,More expression-bodied members ,Throw expressions
C# 7.1	4.7	4.0	Async main ,Default expressions ,Reference assemblies ,Inferred tuple element names ,Pattern-matching with generics
C# 7.2	4.7.1	4.0	Span and ref-like types, In parameters and readonly references, Ref conditional, Non-trailing named arguments, Private protected accessibility, Digit separator after base specifier
C# 7.3	4.7.2	4.0	System.Enum, System.Delegate and unmanaged constraints, Ref local re-assignment, Stackalloc initializers, Indexing movable fixed buffers, Custom fixed statement, Improved overload candidates, Expression variables in initializers and queries, Tuple comparison, Attributes on backing fields
C# 8.0	4.8	4.0	Nullable reference types, default interface members, recursive patterns, async streams, additional using usage, range, and indexes, null-coalescing assignment, static local functions, unmanaged generic structs, readonly members, stackalloc in nested contexts, alternative interpolated verbatim strings support and Obsolete on attribute accessors.

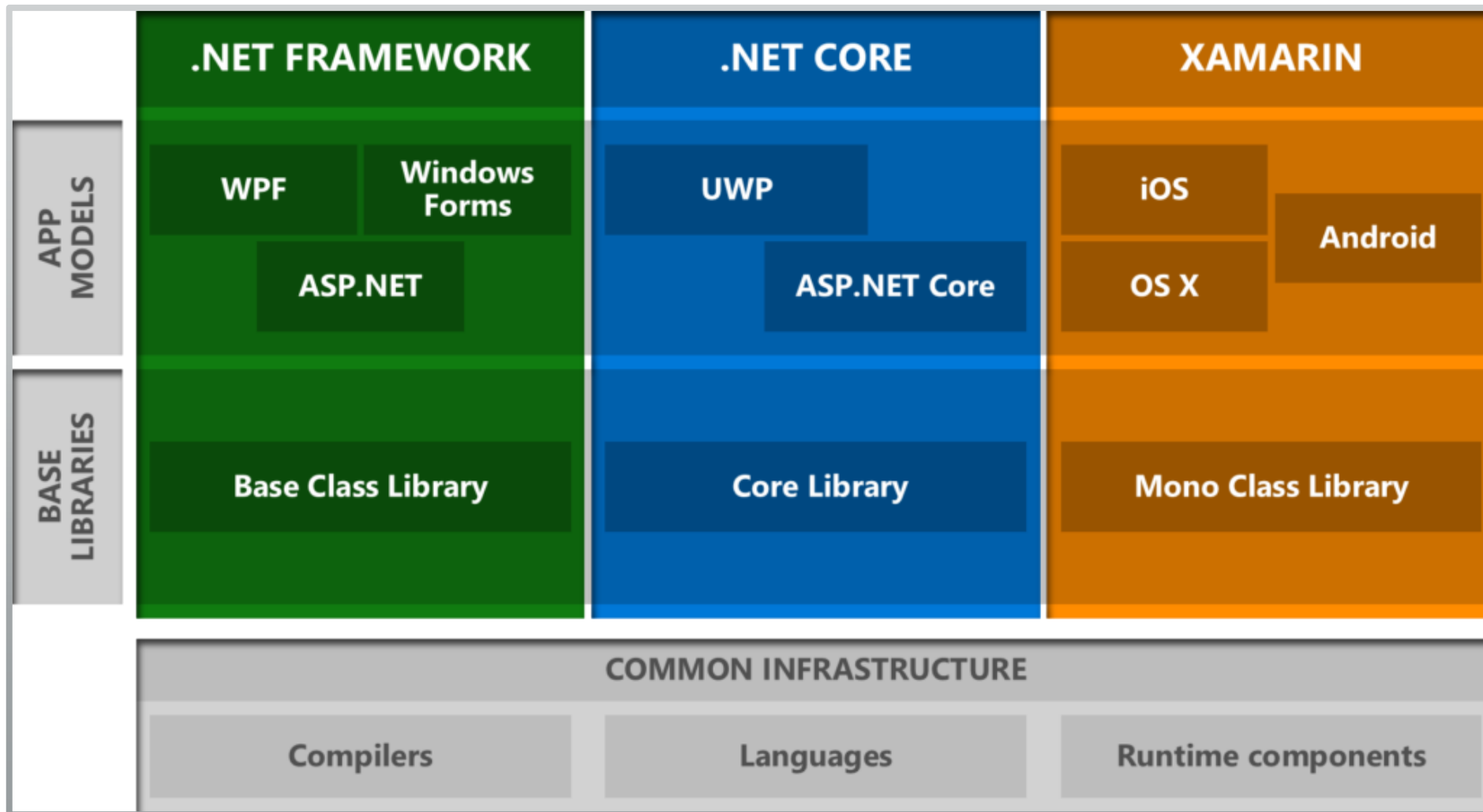


Architecture .net

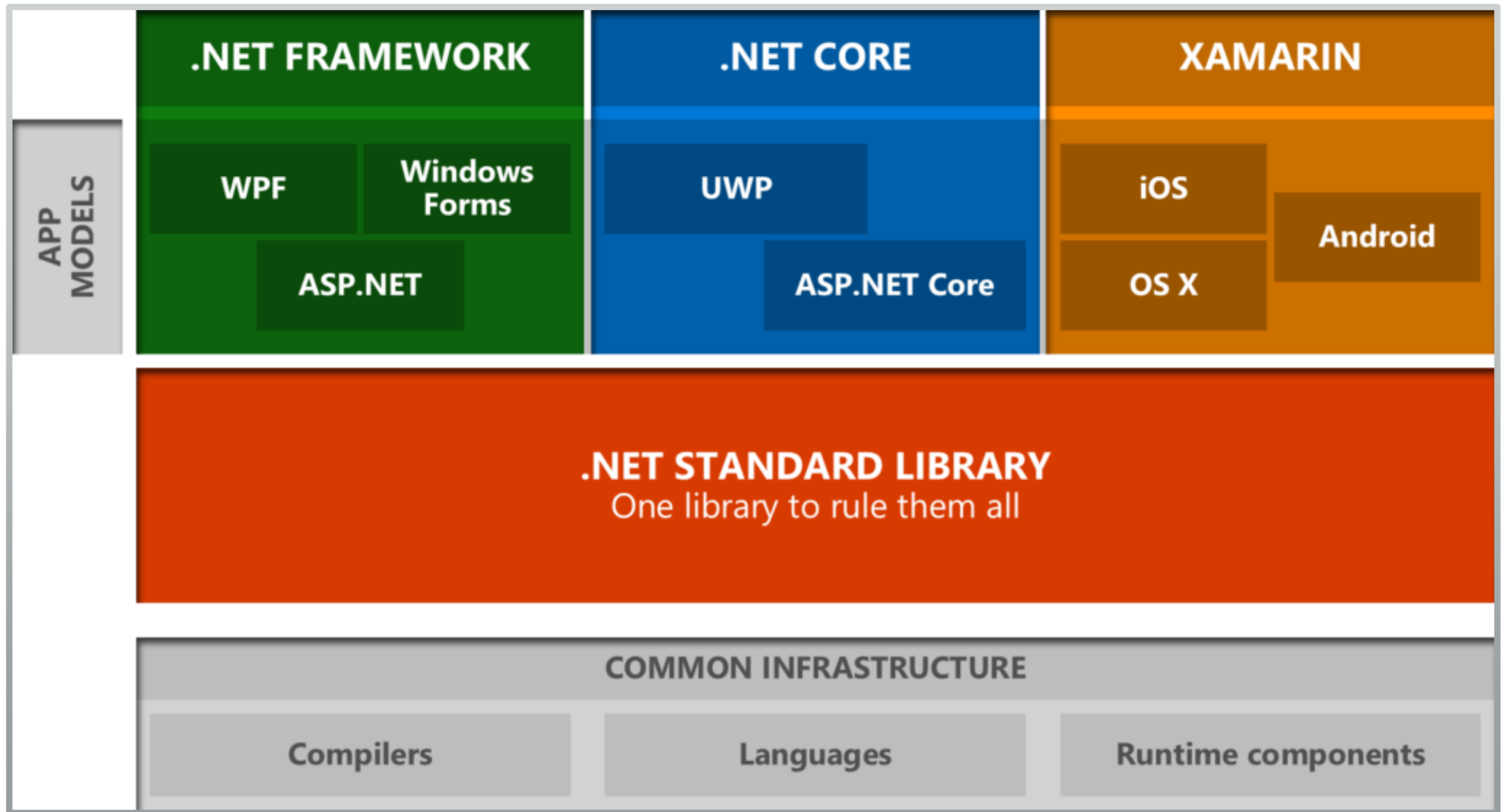
CLI / CIL / CLR



- CLI (Common Language Infrastructure) : fournir un langage indépendant de la plate-forme, aussi bien pour le développement que pour l'exécution. Elle inclut des fonctions pour gérer les erreurs, le [ramasse-miettes](#), la sécurité et l'interopérabilité avec les objets COM.
- CLR (Common Language Runtime) : composant de machine virtuelle du framework .NET. Il s'agit de l'implémentation par Microsoft du standard Common Language Infrastructure (CLI) qui définit l'environnement d'exécution des codes de programmes. Le compilateur à la volée transforme le code CIL en code natif spécifique au système d'exploitation.
- CIL (Common Intermediate Language) : langage de programmation de plus bas niveau qui peut être lu par un humain. Le code de plus haut niveau dans l'environnement .NET est compilé en code CIL qui est assemblé dans un code dit bytecode. CIL est un code assembleur orienté objet et pile



Jusqu'en 2014/2015
(ouverture en opensource)



.Net Framework

- Framework : ensemble de composants structurels (bibliothèques) qui permettent de créer les fondations d'un logiciel (architecture)
- Basé sur CLI dans un environnement windows, donc indépendante du langage de programmation utilisé (c#, vb.net, f#, ASP, etc..)
- But de faciliter les développements en proposant une approche unifiée à la conception d'application web & windows.
- Faciliter le déploiement et la maintenance des applications.
- Besoin d'être installée sur la machine de l'utilisateur final. (Uniquement Windows)
- Framework propriétaire jusqu'à 2014. Open-Source avec mise à disposition sur Github.
- Rétrocompatibilité descendante obligatoire.

.Net Framework

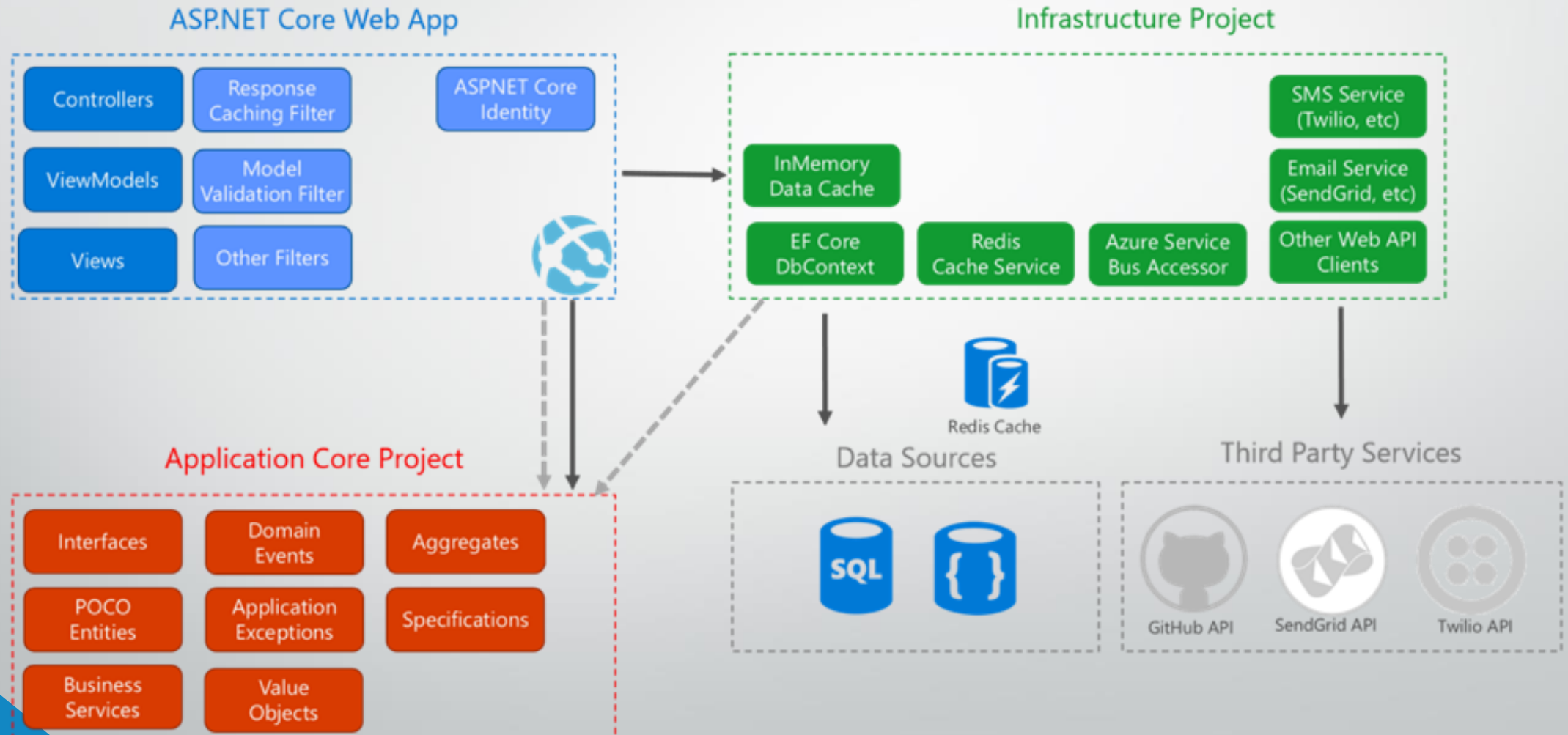


.Net Core

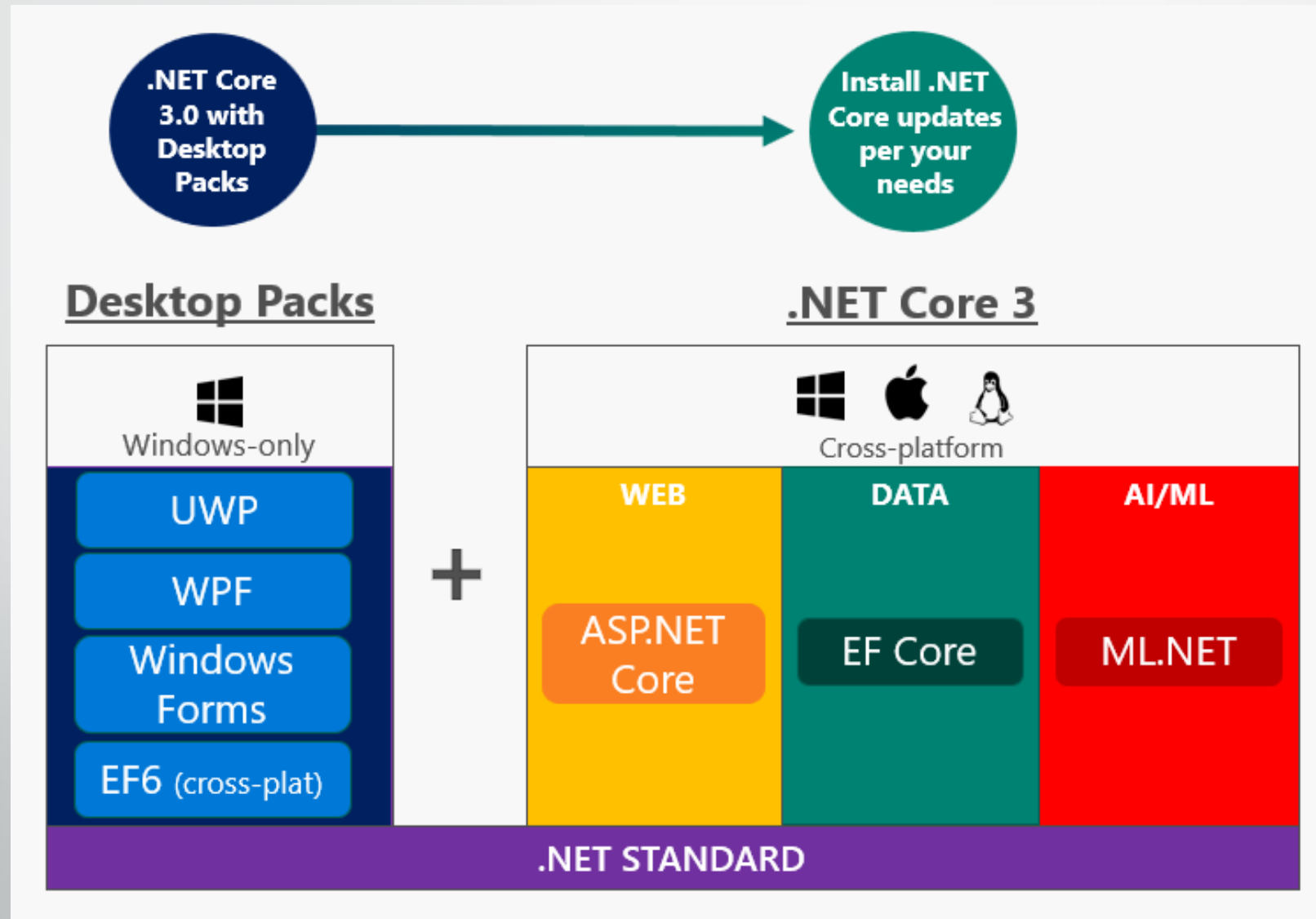
- Framework basé sur CoreCLR Libre et Open-Source capable d'exécuter des applications .net sur Windows, Linux, MacOS.
- CoreCLR est un fork du CLR. Il ne fait pas partie de .Net Framework.
- Pas de rétrocompatibilité descendante. NetCore 3.X n'exécute pas du code en NetCore 2.X
- Lightweight based & Cloud Native => meilleurs performances.
 - On embarque uniquement les packages (Nuget) dont l'application a besoin.
 - Docker native
 - Pas d'application « lourde » dans les premières versions

ASP.NET Core Architecture

-----> Compile Time Dependency
—————> Run Time Dependency



.Net Core 3.X



.Net Standard

- Spécification **formelle** des API .net qui sont destinées à être disponibles sur toutes les implémentations de .net.
- .NET Standard consiste à établir une meilleure uniformité dans l'écosystème .NET.
- .NET Standard permet les scénarios clés suivants :
 - Définit un ensemble uniforme d'API de bibliothèque de classes de base pour toutes les implémentations de .NET à implémenter, indépendamment de la charge de travail.
 - Permet aux développeurs de générer des bibliothèques portables utilisables sur toutes les implémentations de .NET, à l'aide de ce même ensemble d'API.
 - Réduit ou même élimine une compilation conditionnelle de source partagée résultant des API .NET, uniquement pour les API de système d'exploitation.

[illegible]

.Net5.0

.NET – A unified platform





Syntaxe & Coding guidelines

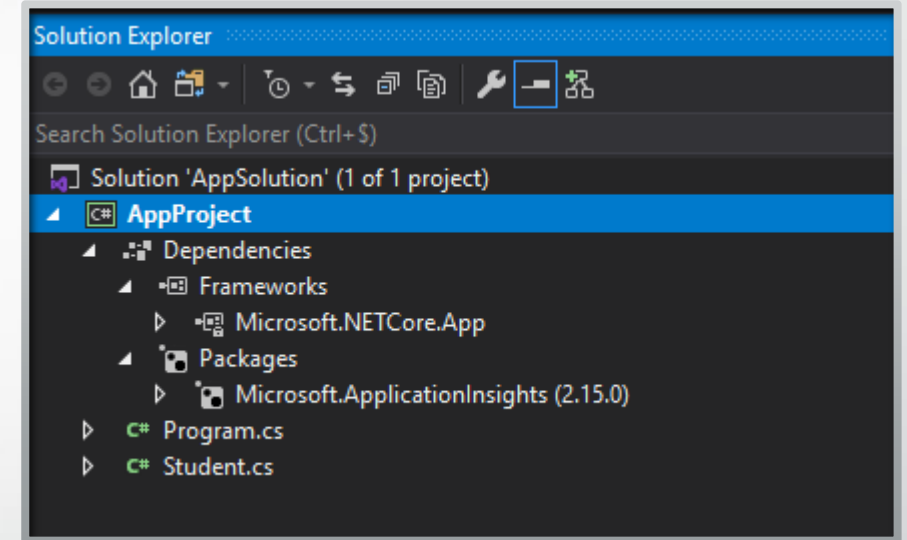
Vocabulaire VisualStudio

Solution : Ensemble de projets composant une application

Projet : brique d'une application comprenant votre code source. Un projet peut être une librairie ou un exécutable.

Packages : librairies annexes ajoutées a votre projet afin d'avoir des fonctionnalités du Framework en plus.

Fichier Cs : Fichier C# compréhensible par le compilateur.



Vocabulaire VisualStudio

Class (Pascal Case) : Nom de la classe qui sera exposé en public / privée / interne

Propriété (Pascal Case) : Accès a des variables de la classe

Variable (camel Case) : variable interne a votre classe. Peut etre globale a la classe (privée) ou uniquement dans une méthode.

Méthode (Pascal Case): Exécution d'un algorithme afin de calculer, retourner un résultat.

```
namespace AppProject
{
    Oreferences
    public class Student
    {
        1reference
        public string Firstname { get; set; }
        1reference
        public string Surname { get; set; }

        private int _age;

        1reference
        public int Age
        {
            get { return _age; }
            set { _age = value; }
        }

        Oreferences
        public string ConcatNames(int age)
        {
            _age = age; //Age = age; IDENTICAL
            string namesConcat = $"{Firstname} {Surname} {Age}";
            return namesConcat;
        }
    }
}
```

Vocabulaire

Convention de nommage

- Pascal Case : Démarre par une majuscule
 - Class
 - Propriétés
 - Méthode
- Camel Case : Démarre par une minuscule
 - Variable
 - Privée de globalité a une class, elle doit etre préfixée d'un _ (Exemple : _age)
 - Utilisation a l'intérieur d'une méthode, ou paramètre d'une méthode, celle-ci reste en camelCase classique

Typage de données

Valeur

- Type simples
 - Entier : int, long, uint, ulong, decimal
 - Caractère : char
 - Virgule flottante : float, double
 - Booleen : bool
- Enum & struct
 - enum E {...}
 - struct S {...}
- Nullable : tous les types de valeurs avec une valeur null

Référence

- Class
 - Tous les types object
 - Chaînes de caractères : string (= String)
 - Class définies : class C {...}
- Interfaces
 - Interface I {...}
- Tableaux
 - Int[], int[,]
 - List<>, Array<>, ... (dérivés de object)
- Délégués
 - delegate int D(...)

Transtypage

- Transtyper = changer une variable de type
- Transtypage implicite
 - `Int n = 12; float f = n;`
- Transtypage explicite
 - `Float f = 12f; n = (int) f;`
- Conversion > Transtypage
 - `Convert.To...(obj)`

Héritage / Polymorphisme

- Comme n'importe quel langage orienté object C# implémente les notions d'héritage et d'encapsulation
- Polymorphisme : prendre « plusieurs formes »
 - Au moment de l'exécution, les objets d'une classe dérivée peuvent être traités comme des objets d'une classe de base dans les paramètres de méthode et les collections ou les tableaux. Lorsque ce polymorphisme se produit, le type déclaré de l'objet n'est plus identique à son type au moment de l'exécution.
 - Les classes de base peuvent définir et implémenter des méthodes virtuelles, et les classes dérivées peuvent les substituer, ce qui signifie qu'elles fournissent leur propre définition et implémentation. Au moment de l'exécution, quand le code client appelle la méthode, le CLR recherche le type au moment de l'exécution et appelle cette substitution de la méthode virtuelle. Dans votre code source, vous pouvez appeler une méthode sur une classe de base et provoquer l'exécution de la version d'une classe dérivée de la méthode.

Polymorphisme

```
public class Shape
{
    // A few example members
    public int X { get; private set; }
    public int Y { get; private set; }
    public int Height { get; set; }
    public int Width { get; set; }
    // Virtual method
    public virtual void Draw()
    {
        Console.WriteLine("Performing base class drawing tasks");
    }
}

public class Circle : Shape
{
    public override void Draw()
    {
        // Code to draw a circle...
        Console.WriteLine("Drawing a circle");
        base.Draw();
    }
}

public class Rectangle : Shape
{
    public override void Draw()
    {
        // Code to draw a rectangle...
        Console.WriteLine("Drawing a rectangle");
        base.Draw();
    }
}

public class Triangle : Shape
{
    public override void Draw()
    {
        // Code to draw a triangle...
        Console.WriteLine("Drawing a triangle");
        base.Draw();
    }
}
```

Polymorphisme

```
// Polymorphism at work #1: a Rectangle, Triangle and Circle
// can all be used wherever a Shape is expected. No cast is
// required because an implicit conversion exists from a derived
// class to its base class.
var shapes = new List<Shape>
{
    new Rectangle(),
    new Triangle(),
    new Circle()
};

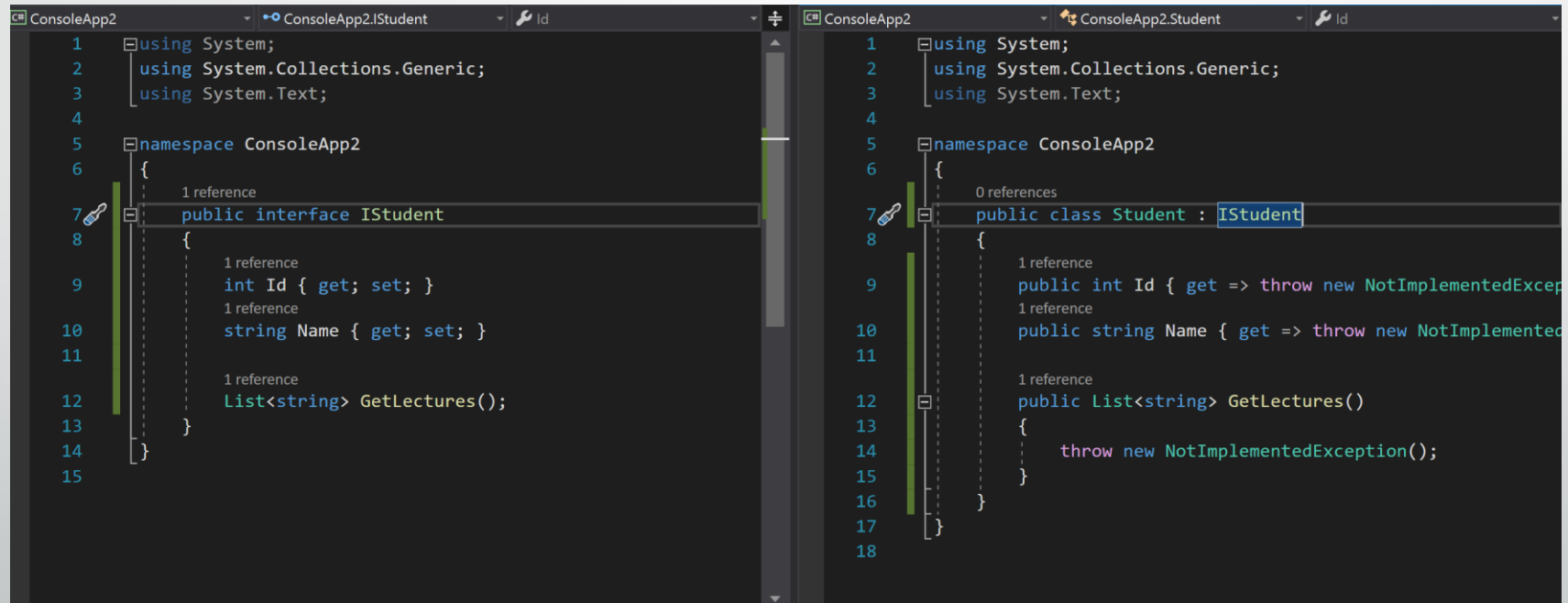
// Polymorphism at work #2: the virtual method Draw is
// invoked on each of the derived classes, not the base class.
foreach (var shape in shapes)
{
    shape.Draw();
}
/* Output:
    Drawing a rectangle
    Performing base class drawing tasks
    Drawing a triangle
    Performing base class drawing tasks
    Drawing a circle
    Performing base class drawing tasks
*/
```

Interfaces

- Aucune implémentation
- Ensemble de méthodes que l'objet doit suivre
= "contrat de service" du composant
- Code réalisé dans les classes qui l'implémente

- Sécurité
- Modularité

Interfaces



The image displays two side-by-side Visual Studio code windows, both titled 'ConsoleApp2'. The left window shows the 'ConsoleApp2.IStudent' file, which defines a public interface 'IStudent' within the 'ConsoleApp2' namespace. The interface includes three properties: 'int Id' with get and set accessors, 'string Name' with get and set accessors, and a method 'List<string> GetLectures()'. The right window shows the 'ConsoleApp2.Student' file, which implements the 'IStudent' interface. It defines a public class 'Student' that inherits from 'IStudent'. The 'Student' class implements the 'Id' and 'Name' properties by throwing a 'NotSupportedException' and implements the 'GetLectures()' method by throwing a 'NotSupportedException'.

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace ConsoleApp2
6 {
7     public interface IStudent
8     {
9         int Id { get; set; }
10        string Name { get; set; }
11
12        List<string> GetLectures();
13    }
14 }
15
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace ConsoleApp2
6 {
7     public class Student : IStudent
8     {
9         public int Id { get => throw new NotImplementedException(); }
10        public string Name { get => throw new NotImplementedException(); }
11
12        public List<string> GetLectures()
13        {
14            throw new NotImplementedException();
15        }
16    }
17 }
18
```

Abstract

- Classe incomplète, non instanciable
- Contient 1 ou plusieurs méthodes abstraites
- Code réalisé dans les classes qui l'hérite
- Possibilité de manipuler des ensembles
- Centralise la gestion des attributs communs

Abstract

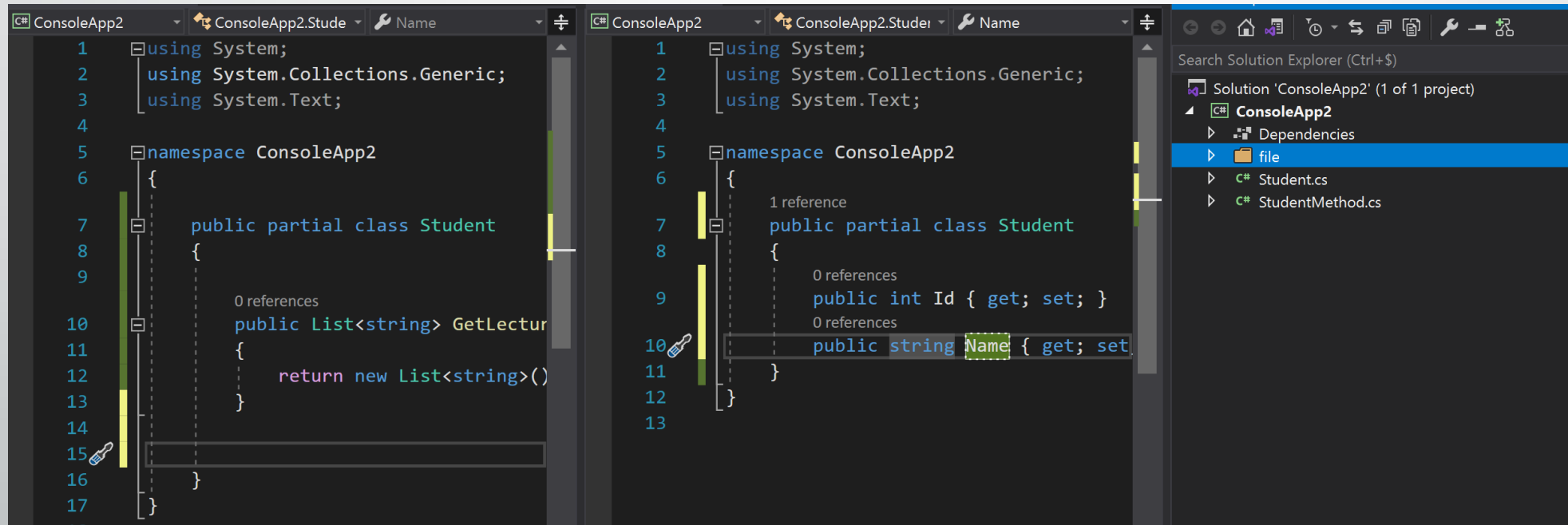
```
Person.cs | IStudent.cs
ConsoleApp2
  ConsoleApp2.Person
    Name
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace ConsoleApp2
6 {
7     1 reference
8     public abstract class Person
9     {
10         0 references
11         public int Id { get; set; }
12         0 references
13         public string Name { get; set; }
14         1 reference
15         public abstract List<string> GetLectures();
16     }
17 }

Student.cs* | Program.cs
ConsoleApp2
  ConsoleApp2.Student
    GetLectures()
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace ConsoleApp2
6 {
7     0 references
8     public class Student : Person
9     {
10         1 reference
11         public override List<string> GetLectures()
12         {
13             return new List<string>();
14         }
15     }
16 }
```


Partial

- Classe / méthode partielle
- Permet de splitter une classe dans de multiples fichiers
- Une seule obligation => Appartenir au même type

Partial



Opérateurs

- Arithmétiques
 - $+$ $-$ $*$ $/$
 - $\%$: Reste
- Incrémentation
 - Pré-incrémentation : $++var$
 - Post-incrémentation : $var++$
- Logiques
 - $!$: non
 - $\&$: et
 - $|$: ou
 - $\&\&$: et court-circuité
 - $||$: ou court-circuité

Conditionnel

- If
 - Forme :
if (cond) { traitement si cond vérifié } else { traitement sinon }
 - Version condensée :
Expression ? Valeur : valeur;
- Switch (cond)
 - {
 - case 1 : ... ; break;
 - default : ...; break;
 - }

Boucles

- While (cond) { traitement }
- For(int i = 0; i < 10, i++){ traitement }
- Foreach(int i in tab){traitement}

Collections

- Namespace : `System.Collections.Generic`
- Containers génériques
 - Avantage : Contient des objets typés dont le type est précisé dans l'instanciation
- Collections existantes
 - `List <Class>`
 - `Stack <Class>`
 - `Queue<Class>`
 - `Dictionary<keyClass, valueClass>`

Exemple List<>

- Implémente IList
- Non dimensionné
- Générique
- Gestion
 - Ajout : Add(<Class> o);
 - Suppression : Remove(<Class> o); RemoveAt(position);
 - Accès : MaListe[position]; MaListe.IndexOf(position)
 - Taille : Count();

Linq

- Language-Integrated Query, ou Requêtage intégré au langage = Un ajout marquant du Framework 3.5 et de C# 3.0
- Linq permet
 - L'interrogation uniforme quelque soit le type de données
 - Récupération et manipulation de données
 - Couche d'abstraction des données
- Linq se décompose en
 - **Linq To Objects** : manipulation des collections en .net
 - Manipulation des collections d'objets
 - Parcours, tri, filtres sur les collections
 - Remplacement des boucles while, foreach, ...
 - Linq To ADO.Net (ie Linq To SQL, Linq To DataSet et Linq To Entities) : Récupération et manipulation des données d'un SGBD
 - Linq To XML : Récupération et manipulation des données d'un modèle xml

Linq

- Exemple de requête Linq :

```
IEnumerable<string> results = from x in auteurs  
    where x.Prenom.StartsWith("Ai")  
    select x.Prenom + ' ' + x.Nom;
```

OU

```
IEnumerable<string> results = auteurs.Where(x => x.Prenom.StartsWith(« Ai »)).Select(x =>  
x.Prenom + ' ' + x.Nom);
```