

**Ecole Nationale des Sciences Appliquées  
Al Hoceima**

# **DÉVELOPPEMENT D'APPLICATIONS .NET**

---

M. Tarik BOUDAA

Année universitaire: 2020/2021

## Enseignant : Tarik BOUDAA

- Ingénieur & docteur en Informatique
- Ingénieur Principal à ENSAH
- Auparavant Concepteur et Développeur Java/JEE à Atos
- Chercheur en traitement automatique de la langue naturelle
- Email : [t.boudaa@uae.ac.ma](mailto:t.boudaa@uae.ac.ma)

## Codes sources

- Les codes sources associés à cette présentation seront publiés sur la plateforme *e-services*

1

# **Introduction à la plateforme Microsoft .NET**

## C'est quoi la plateforme .NET?

- ❑ Microsoft .NET est **une plateforme de développement d'usage général** pour **tout type d'applications** ou de charge de travail, qui offre des fonctionnalités essentielles pour la création d'applications de grande qualité s'exécutant dans un **environnement dit « managé »**, qui permet de gérer tous les aspects de l'exécution d'une application notamment la gestion automatique de la mémoire.
- ❑ .NET permet :
  - de développer des applications
  - de déployer des applications
  - d'exécuter des applications
- ❑ Plusieurs types d'applications peuvent être développées avec .NET :
  - Applications Web
  - Web services
  - Applications Windows
  - Services Windows
  - Applications Console
  - Applications Mobile
  - Bibliothèques de classes
  - IoT / AI
  - Microservices, Cloud
  - etc..



**Du point de vue du programmeur, .NET peut être compris comme un environnement d'exécution et une bibliothèque de classes de base complète.**

## Langages supportés par .NET

❑ .NET Framework supporte different langages de programmation:

➤ **VB .NET**

➤ **F#**

➤ **C#**

➤ C++

➤ etc.

C#

VB

C++

F#

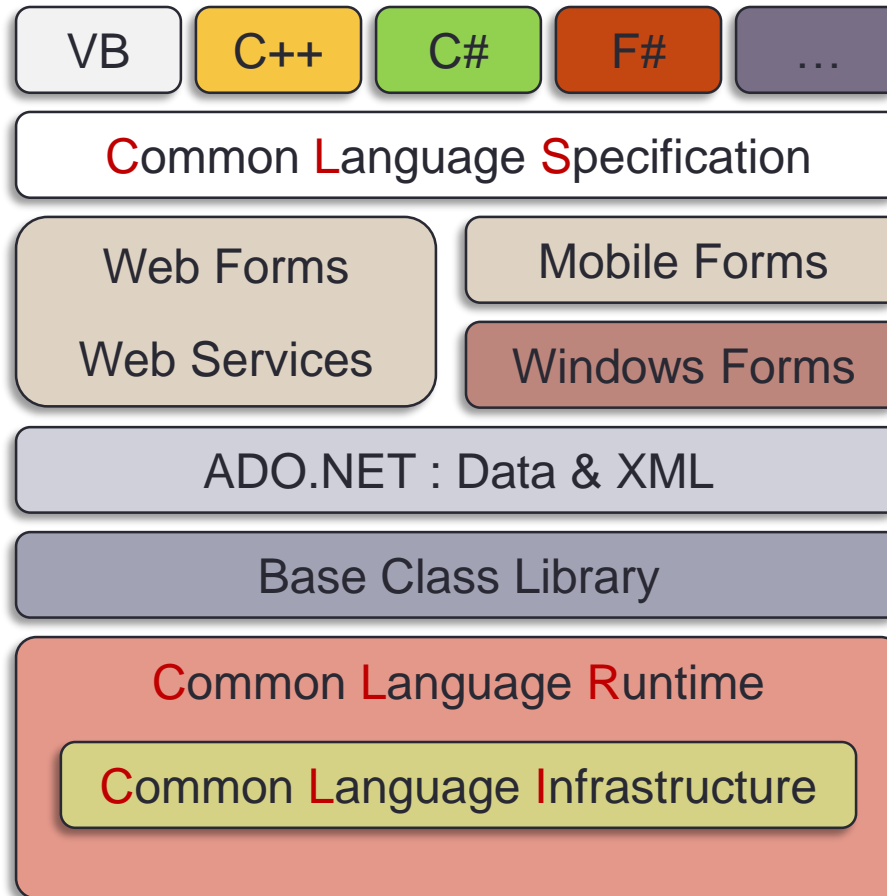
## Microsoft Visual Studio .NET

Visual Studio est un ensemble complet d'outils de développement permettant de créer des applications, des Services Web, des applications bureautiques et des applications mobiles. Visual Basic, Visual C# et Visual C++ utilisent tous le même environnement de développement intégré (IDE), qui permet le partage d'outils et facilite la création de solutions à plusieurs langages. Par ailleurs, ces langages utilisent les fonctionnalités du .NET Framework, qui fournit un accès à des technologies clés simplifiant le développement d'applications Web, Services Web et REST par exemple.



# L'architecture du Framework .NET<sup>7</sup>

## Architecture globale (simplifiée) du Framework



# L'architecture du Framework .NET<sup>8</sup>

## Le Framework .Net supporte l'interopérabilité entres les langages

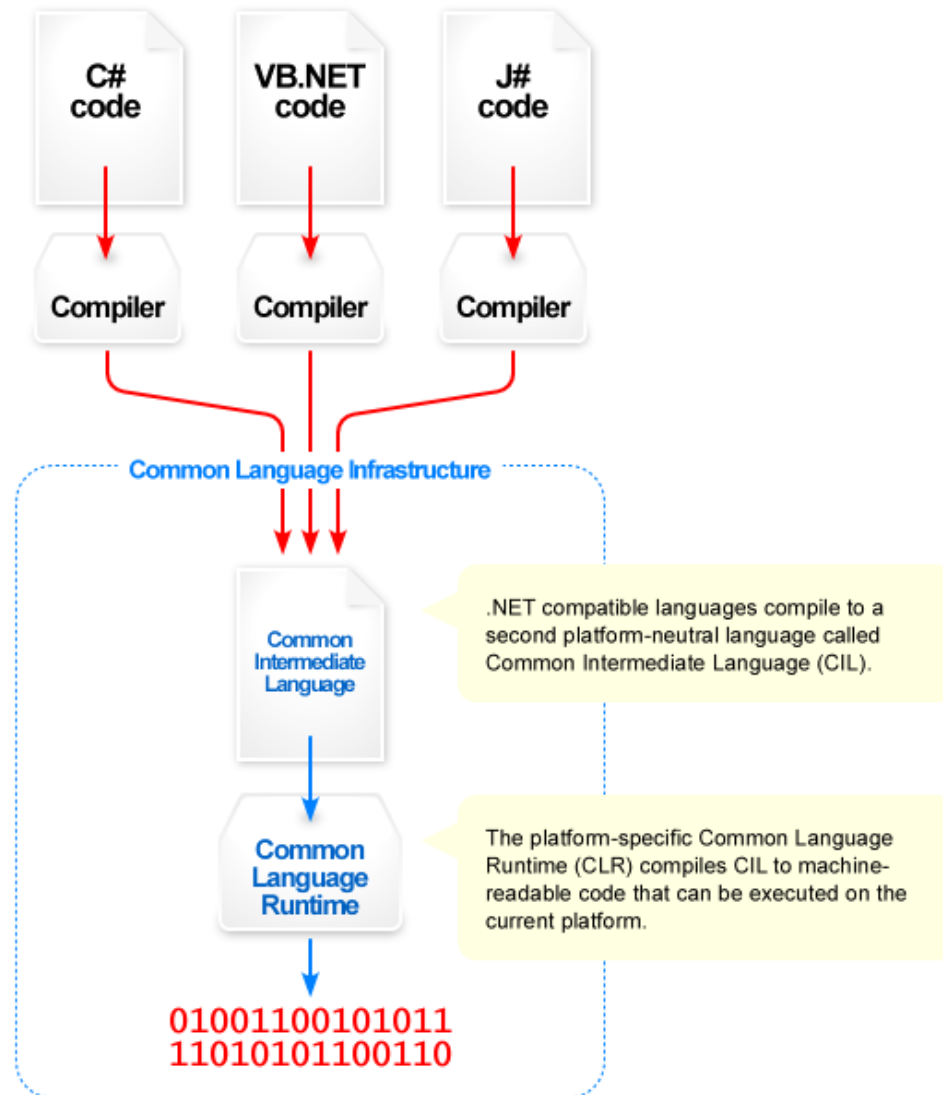
- ❑ L'interopérabilité est la capacité que possède un système, dont les interfaces sont intégralement connues, à fonctionner avec d'autres systèmes.
- ❑ Une implémentation .NET est indépendante du langage. Cela ne signifie pas seulement qu'un programmeur peut écrire son code dans n'importe quel langage qui peut être compilé en *Common Intermediate Language (Bytecode)*. Mais également utiliser des bibliothèques/classes écrites par un langage .Net quelconque dans un autre langage .NET.
- ❑ **interopérabilité entres les langages** (cross-language interoperability): Par exemple .NET prend en charge l'héritage entre des types définis dans des langages .Net différents. Ainsi vous pouvez définir une classe de base en C # et étendre ce type dans Visual Basic.



# L'architecture du Framework .NET<sup>9</sup>

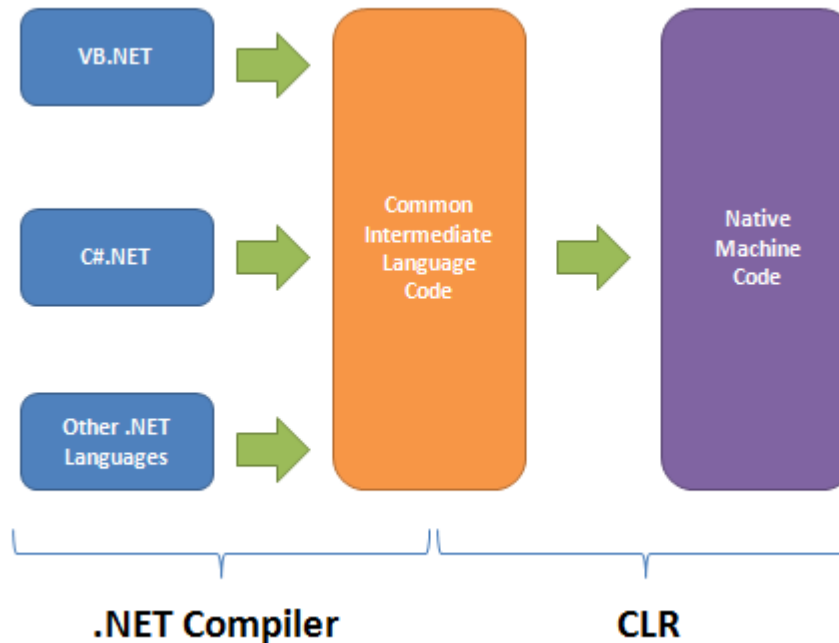
## La Common Language Infrastructure (CLI)

- ❑ La Common Language Infrastructure (**CLI**) est une **spécification** ouverte développée par Microsoft pour sa plateforme .NET qui décrit l'environnement d'exécution de la machine virtuelle basé sur CIL. La spécification définit un environnement qui permet d'utiliser de nombreux langages de haut niveau
- ❑ Le code répondant aux spécifications CLI est dit « **managed code** »
- ❑ L'**implémentation** de la CLI inclut des fonctions pour gérer les erreurs, le ramasse-miettes, la sécurité et l'interopérabilité avec le système d'exploitation



# L'architecture du Framework .NET<sup>10</sup>

## Common Language Runtime (CLR)

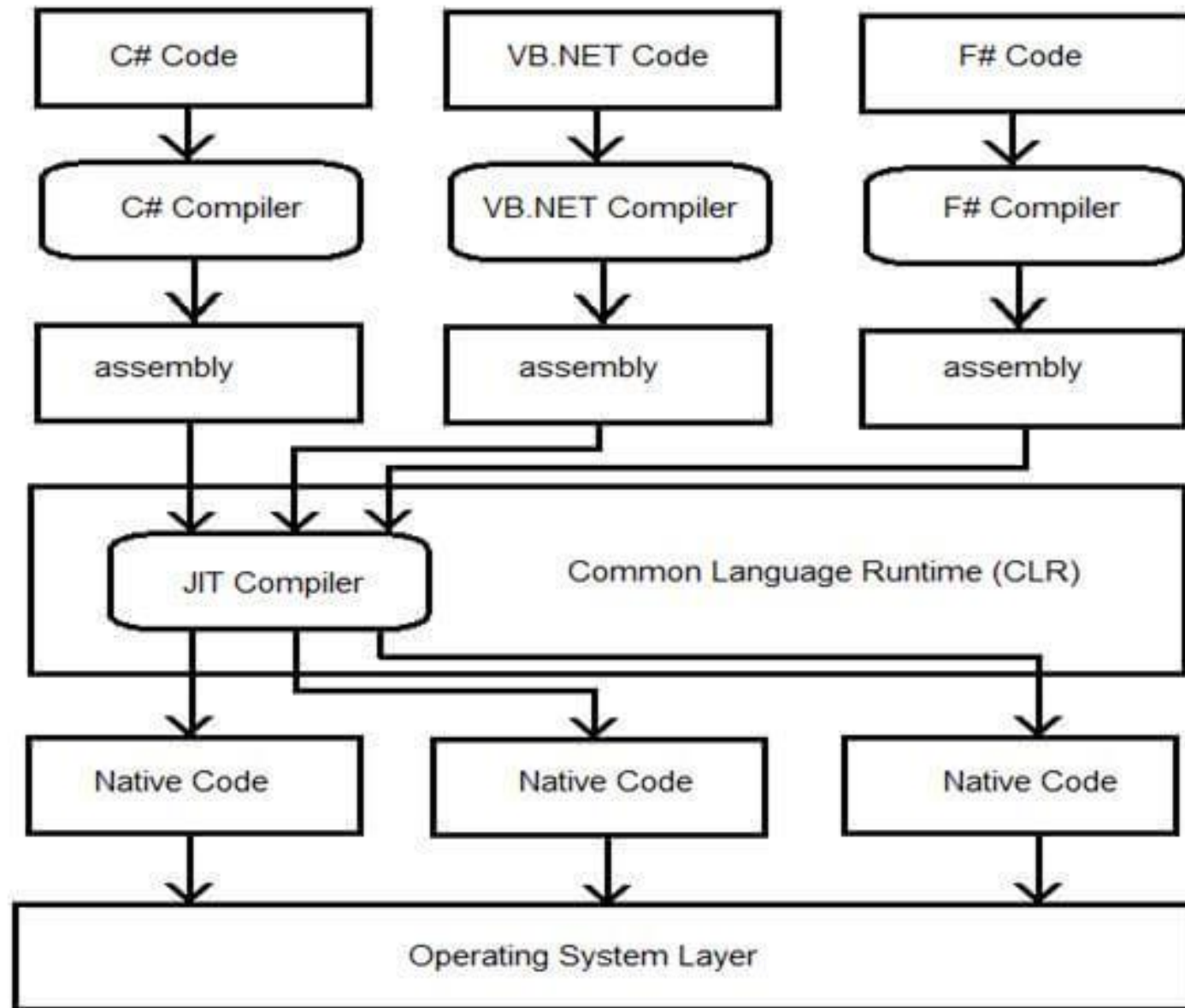


## Common Language Runtime

- ☐ L'implémentation de Microsoft pour le Common Language Infrastructure (CLI)
- ☐ Elle permet d'exécuter les applications .NET
- ☐ Conceptuellement le CLR est similaire à JVM dans le monde JAVA

# L'architecture du Framework .NET<sup>11</sup>

## Common Language Runtime (CLR)



# L'architecture du Framework .NET<sup>12</sup>

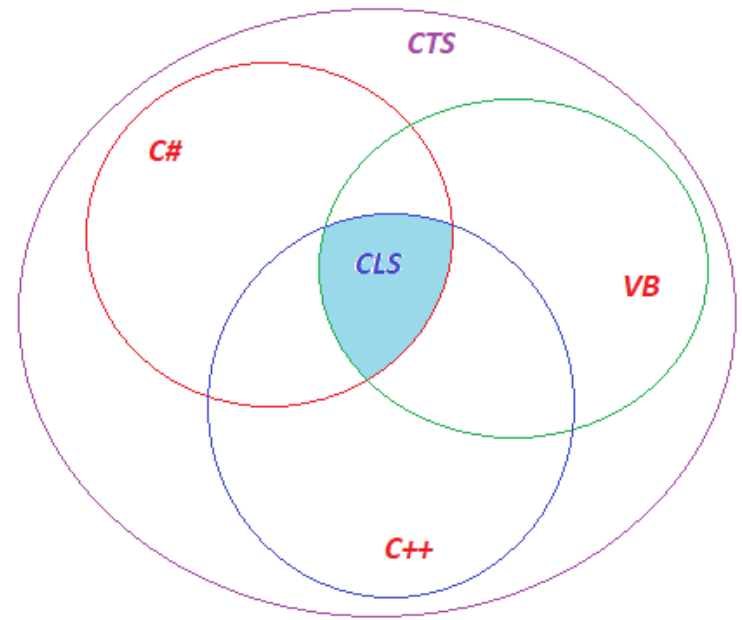
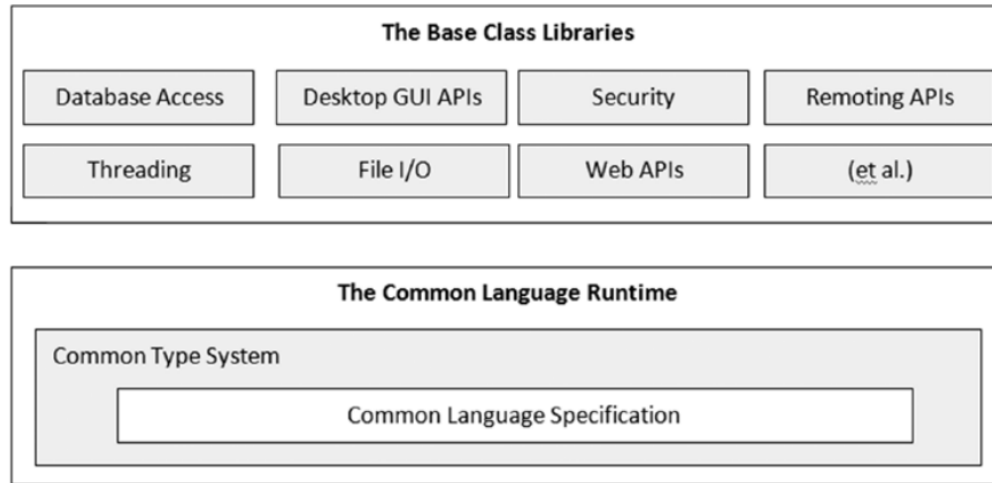
## Common Language Runtime (CLR)

Le **CLR** est l'environnement d'exécution des applications, l'équivalent de la Machine Virtuelle Java (JVM). Il est composé d'un ensemble de services qui sont indépendants du langage de programmation utilisé. Parmi ces services on distingue:

- La gestion de l'exécution du code
- La gestion de la mémoire (GC)
- La gestion des threads
- La gestion des exceptions (erreurs)
- La gestion de la sécurité du code
- ...

# L'architecture du Framework .NET<sup>13</sup>

## Spécifications pour l'intégration multi-langages



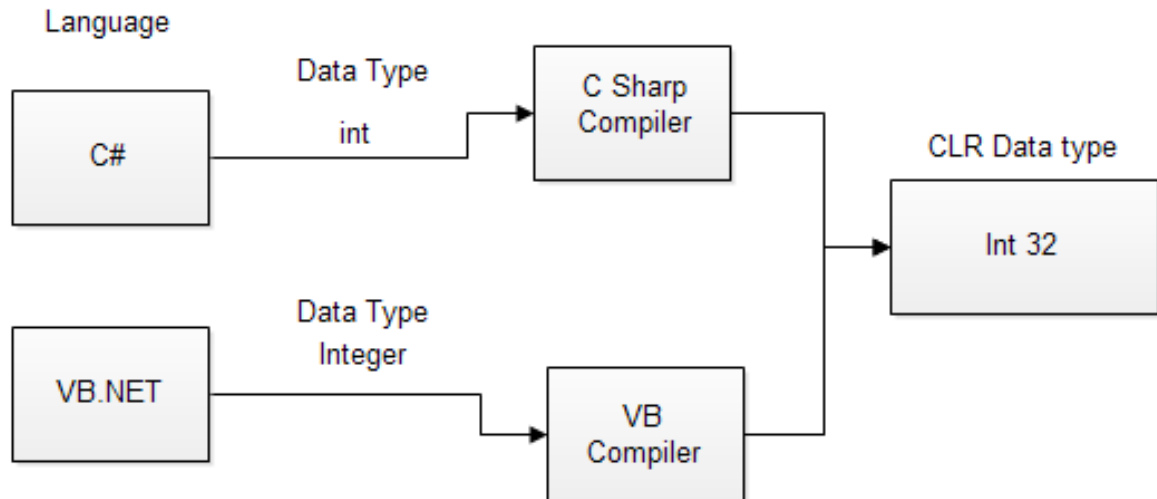
- **CTS (Common Type System)** définit la façon dont les types sont déclarés, utilisés et gérés dans le Common Language Runtime, et constitue également une partie importante de la prise en charge de l'intégration inter-langue
- **CLS signifie Common Language Specification** et il s'agit d'un sous-ensemble de CTS. Elle définit un ensemble de règles et de restrictions à suivre pour assurer une intégration ou une interopérabilité entre les langages.
- Une compréhension approfondie des spécifications CTS et CLS n'est généralement d'intérêt que pour les constructeurs d'outils / compilateurs.

# L'architecture du Framework .NET<sup>14</sup>

## Common Type System (CTS)

CTS définit plusieurs catégories de types, chacune avec sa sémantique et son utilisation spécifiques:

- Classes
- Structures
- Enums
- Interfaces
- Delegates



# L'architecture du Framework .NET<sup>15</sup>

## Base Class Library (BCL)

- La bibliothèque des classes de base utilisée par tous les langages basés sur Framework .NET, elle encapsule un grand nombre de fonctions communes nécessaires pour les tâches de routine (de bas niveau) comme la manipulation de fichiers , le rendu graphique, l'interaction avec une base de données, ....

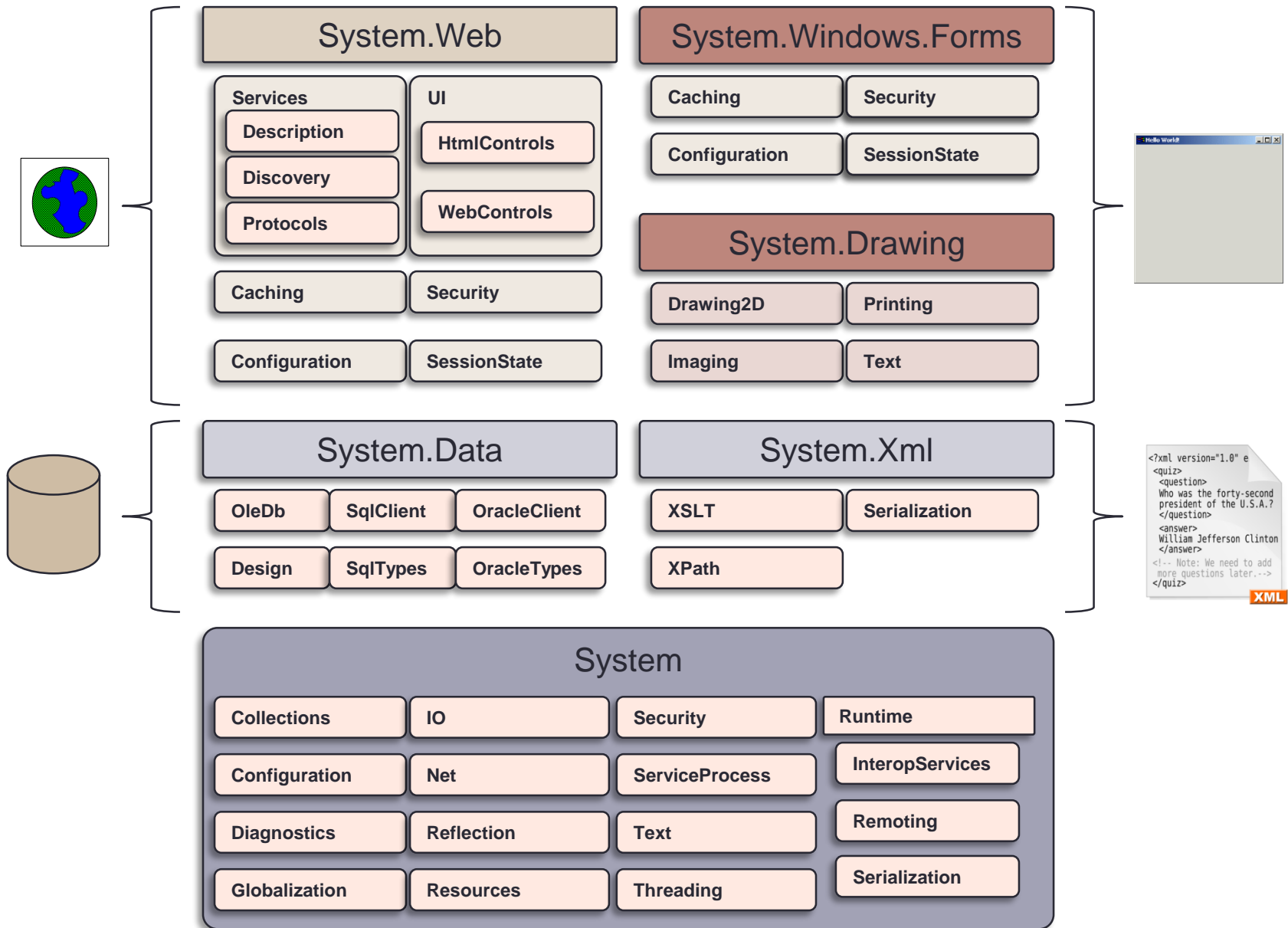
## Framework Class Library (FCL):

La bibliothèque complète livrée avec la plate-forme. Elle définit un ensemble de technologies spécifiques (ADO.NET pour l'accès aux données, ASP.NET pour les applications Web, ...). La BCL est un sous-ensemble de celle-ci.

*“The Base Class Library (BCL) is the core of the FCL and provides the most fundamental functionality”*

# L'architecture du Framework .NET<sup>16</sup>

## FCL & BCL





# Rappel : Les différents types des langages de programmation

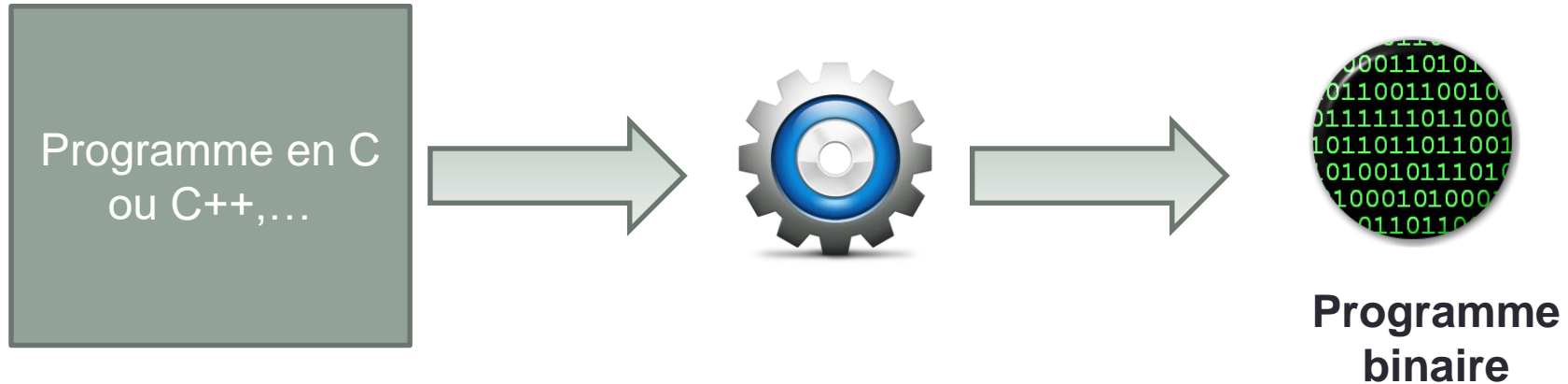
Un langage informatique est par définition différent du langage machine. Il faut donc le traduire pour le rendre intelligible du point de vue du processeur.

- **Langages compilés** : Un programme écrit dans un langage dit « compilé » se traduit une fois pour toutes par un programme annexe (compilateur) afin de générer un nouveau fichier autonome (n'a plus besoin d'un autre programme pour s'exécuter), on dit d'ailleurs que ce fichier est exécutable.
- **Langages interprétés** : Un programme écrit dans un langage interprété a besoin d'un programme auxiliaire (l'interpréteur) pour traduire au fur et à mesure les instructions du programme.
- **langages compilés en langage intermédiaire bytecode** : Avec les besoins de l'interopérabilité, une troisième voie s'ouvre : celle des langages compilés en langage intermédiaire *bytecode*, lui-même interprété (ou compilé) au sein d'une machine virtuelle.

# Rappel : Compilation traditionnelle d'un programme en binaire

18

*Compilation : traduire le code source en binaire exécutable*



## Avantages :

- Exécutable autonome
- Rapidité d'exécution : les programmes compilés sont réputés toujours plus rapides que ceux interprétés
- Un programme compilé a pour avantage de garantir la sécurité du code source. En effet, un langage interprété, étant directement intelligible (lisible), permet à n'importe qui de connaître les secrets de fabrication d'un programme et donc de copier le code voire de le modifier

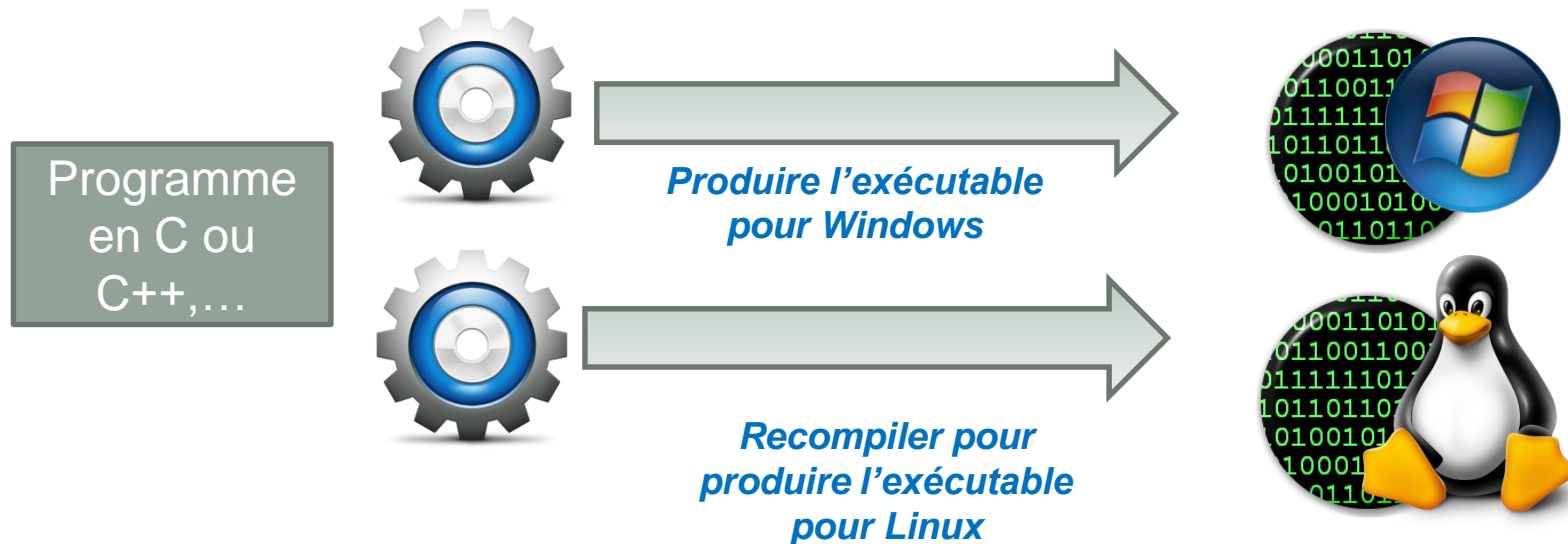
# Rappel : Compilation traditionnelle d'un programme en binaire

19

## Inconvénient :

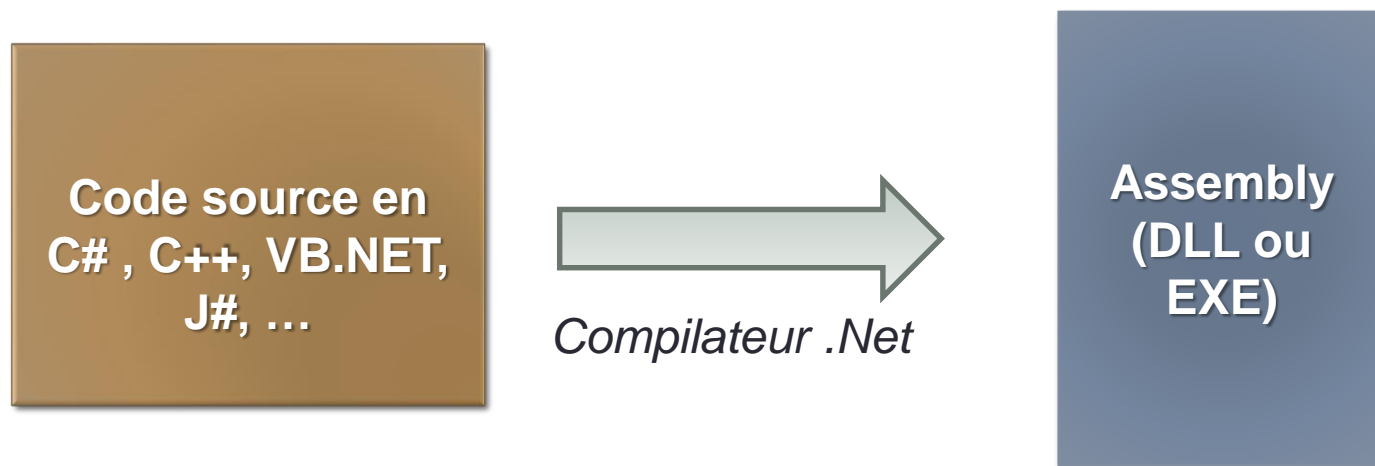
L'exécutable n'est pas portable (il faut faire des compilations multiples pour cibler toutes les plateformes) :

Un programme compilé (binaire) ne fonctionne que sur la plateforme pour laquelle il a été compilé, Cela veut dire que si on compile sous Windows, le programme obtenu ne fonctionnera que sous Windows (et sur un type de processeur particulier). Ainsi, pour qu'il fonctionne sous Linux par exemple il faut le recompiler sous Linux et d'effectuer au passage quelques modifications.



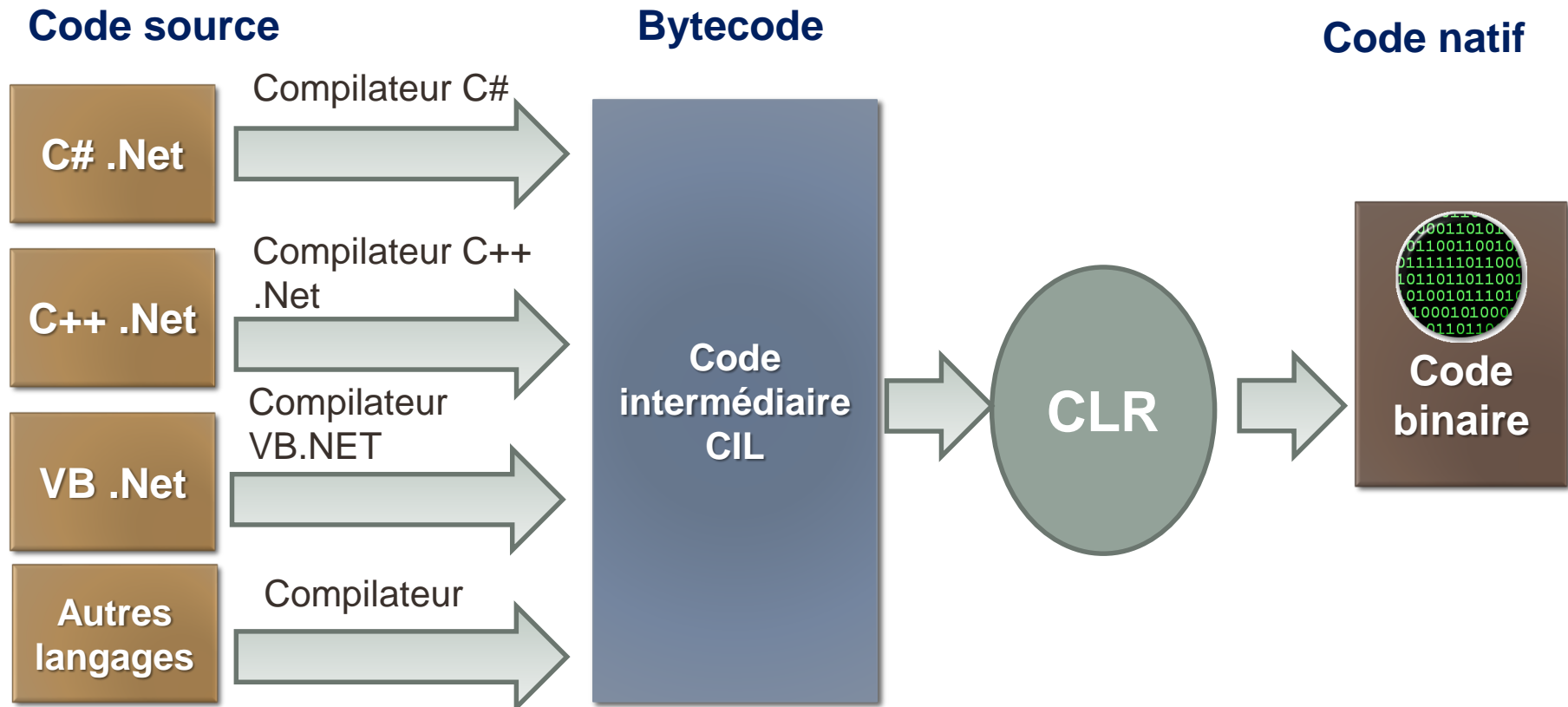
# Le code CIL (ou MSIL)

La compilation d'un programme écrit en .NET conduit vers la création d'un fichier exécutable (fichier .exe). Cet exécutable n'est pas écrit en code machine comme les exécutables classiques mais avec un langage intermédiaire appelé CIL (ou MSIL). Bien entendu, l'exécution du code CIL ne peut pas être directement assurée par les services du système d'exploitation.



# Etapes de compilation des langages .NET<sup>21</sup>

Le **CLR** fait tourner une sorte de bytecode (Le code CIL). Le compilateur à la volée (*JIT compiler*) **transforme le code CIL en code natif spécifique au système d'exploitation**. Le CLR fonctionne sur des systèmes d'exploitation Microsoft Windows. Le CLR est à .NET ce que la JVM est à Java, c'est-à-dire une machine virtuelle, sans laquelle l'exécution de code .NET ne serait pas possible. À l'inverse de son concurrent, le *framework.NET* a été conçu pour permettre l'interopérabilité entre différents langages.



## Windows Forms (WinForms)

Winforms est une bibliothèque de classes du framework .Net pour le développement d'interfaces graphiques pour les applications desktop.

Winforms commence à être en retard techniquement par rapport à d'autre Framework concurrent du marché.

### Quelques inconvénients de WinForms :

- Très limité dans la personnalisation de ses différents contrôles
- L'intégration d'éléments et effet 3D est très difficile
- Il n y a pas de séparation entre l'aspect design de l'interface et les traitements associés, ainsi le développeur et le Designer effectuent tous les deux leurs travaux sur le même code ce qui rendait très compliqué le travail collaboratif.

## .NET

### Windows Presentation Foundation (WPF)

Windows Presentation Foundation (WPF) dans Visual Studio 2015 fournit aux développeurs un modèle de programmation unifié pour créer des applications métier de bureau modernes sur Windows. (Source : MSDN).

Le **WPF** va introduire **XAML** eXtensible Application Markup Language (qui s'agit d'un dialecte XML) qui est un complément au langage .NET (C#, VB,...) et qui s'occupe de la partie design des contrôles, permettant ainsi une séparation entre le design et les traitements.



### ADO.Net

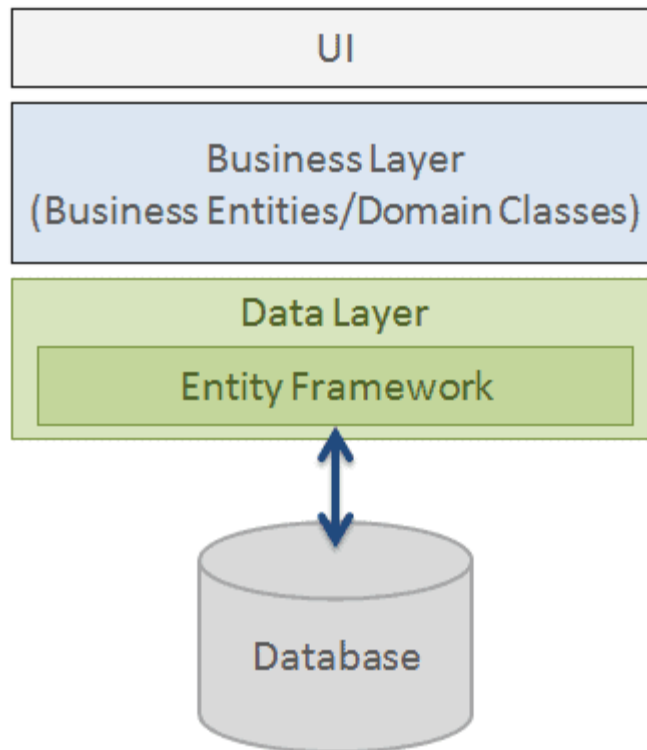
ADO.NET est un ensemble de classes qui exposent les services d'accès aux données pour les programmeurs .NET Framework. ADO.NET fournit un ensemble complet de composants pour créer des applications distribuées de partage de données. Il fait partie intégrante du .NET Framework, fournissant un accès aux données relationnelles, XML et applicatives. ADO.NET prend en charge une variété de besoins de développement, y compris la création de clients de base de données frontaux et d'objets métier de niveau intermédiaire utilisés par des applications, des outils, des langages ou des navigateurs Internet.



## .NET

### Entity Framework

Entity Framework est un mappeur relationnel objet (O / RM) qui permet aux développeurs .NET de travailler avec une base de données à l'aide d'objets .NET. Il élimine le besoin de la plupart du code d'accès aux données que les développeurs ont généralement besoin d'écrire

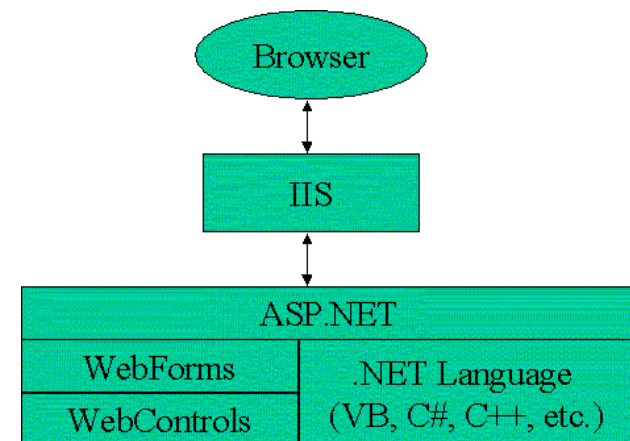


## ASP.NET

« ASP.NET est un modèle de développement Web unifié qui offre les services dont vous avez besoin pour générer des applications Web d'entreprise avec un minimum de codage. ASP.NET fait partie du .NET Framework et, lors du codage des applications ASP.NET, vous avez accès aux classes du .NET Framework. Vos applications peuvent être créées dans tout langage compatible avec le Common Language Runtime (CLR), comme Microsoft Visual Basic .NET, C#, Jscript .NET et J#. (Source : MSDN) »

ASP.NET offre trois Framework pour la création d'applications Web:

- **Web ASP.NET Web Forms**
- **ASP.NET MVC**
- **ASP.NET Web Pages**



# <sup>27</sup>Inconvénients de la Plateforme .NET classique

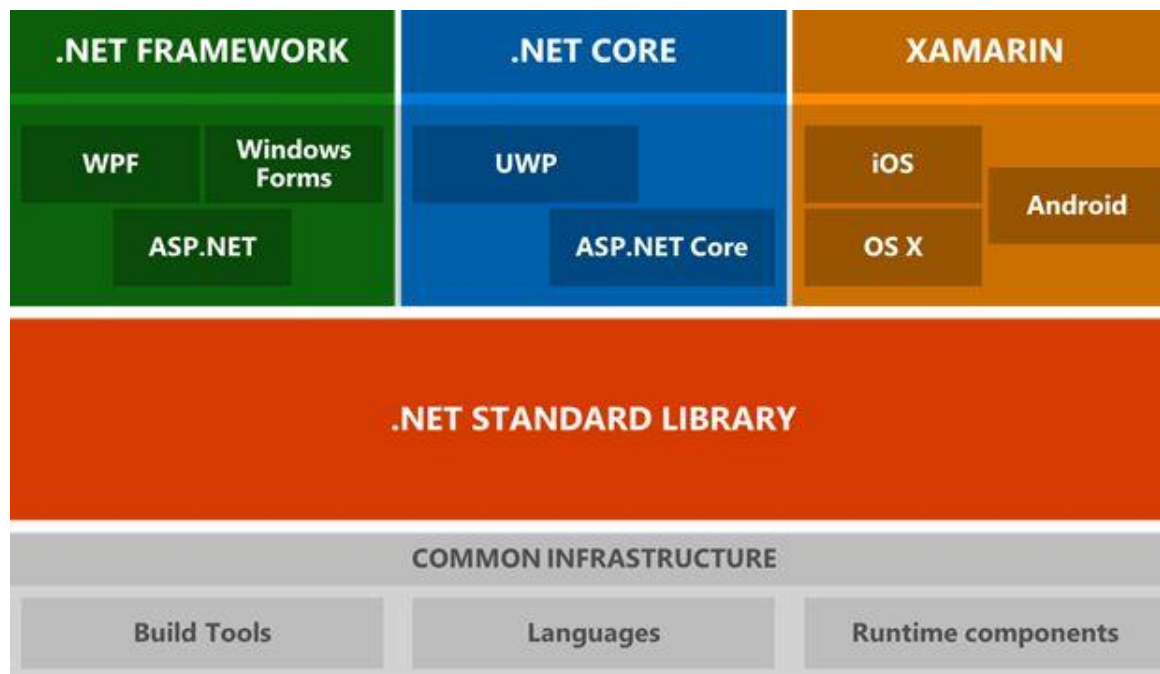
- ❑ Bien que conceptuellement le framework .Net supporte l'exécution sur plusieurs systèmes d'exploitation mais pratiquement parlant .Net Framework est une plateforme destinée et optimisée par Microsoft pour windows.
- ❑ Toutes les applications sur un ordinateur écrites pour .NET Framework partagent la même version du CLR et des bibliothèques stockées dans le Global Assembly Cache (GAC), ce qui peut entraîner des problèmes si certaines d'entre elles ont besoin d'une version spécifique pour la compatibilité.
- ❑ .Net Framework n'est pas recommandé actuellement à l'utiliser pour développer de nouvelles applications. Il peut être utilisé pour maintenir les applications existantes dans des projets de migration des applications Legacy.
- ❑ .Net Core est l'avenir

# Projet Mono

- ❑ Des tiers ont développé une implémentation .NET Framework appelée le projet Mono. Mono est multiplateforme, mais il n'implémente pas toutes les fonctionnalités de l'implémentation officielle de .NET Framework.
- ❑ Mono a trouvé une niche en tant que base de la plate-forme mobile Xamarin ainsi que des plates-formes de développement de jeux multiplateformes telles que Unity.
- ❑ Le projet Mono est une distribution open source de la CLI qui cible diverses distributions Linux, macOS, les appareils iOS (iPad, iPhone), les appareils Android et Windows.

# Différentes plateformes .NET

Actuellement il existe 3 .NET. Et Microsoft travaille actuellement pour les unifier



D'ici la fin de 2021, Microsoft promet qu'il y aura une seule plate-forme .NET. .NET 6 est prévu d'avoir un seul BCL et deux environnements d'exécution: l'un optimisé pour les scénarios de serveur ou de bureau tels que les sites Web et les applications de bureau Windows basés sur le runtime .NET Core, et l'autre optimisé pour les applications mobiles basées sur le runtime Xamarin.

# Différentes plateformes .NET

## Les 3 plateformes actuelles

Technology	Description	Host OSes
.NET 5	Modern feature set, full C# 9 support, port existing and create new Windows and Web apps and services.	Windows, macOS, Linux
.NET Framework	Legacy feature set, limited C# 8 support, no C# 9 support, maintain existing applications.	Windows only
Xamarin	Mobile and desktop apps only.	Android, iOS, macOS

- ❑ Le nouveau produit .NET Core (ou .Net 5) comprend une implémentation multiplateforme du CLR connue sous le nom de **CoreCLR** (*alternative de CLR*)
- ❑ et une bibliothèque rationalisée de classes connue sous le nom de **CoreFX** (*un fork partiel de FCL*)
- ❑ Xamarin est né du projet Mono et permet de développer des applications d'interface graphique multiplateformes pour les appareils mobiles

# .Net Core

## Différences majeures entre .Net Framework et .Net Core

- .Net Core est plus adapté aux applications modernes, notamment les applications micro-services et cloud.
- .Net est cross-plateforme.
- .NET Core est plus petit que la version actuelle de .NET Framework en raison du fait que les technologies héritées et non multiplateformes ont été supprimées. Par exemple, Windows Forms et Windows Presentation Foundation (WPF)
- ASP.NET Web Forms et Windows Communication Foundation (WCF) sont d'anciennes technologies d'application et de service Web que moins de développeurs choisissent d'utiliser pour de nouveaux projets de développement aujourd'hui, elles ont donc également été supprimées de .NET 5. Au lieu de cela, les développeurs préfèrent utiliser ASP .NET MVC et API Web ASP.NET. Ces deux technologies ont été refactorisées et combinées dans une plate-forme qui s'exécute sur .NET 5, nommée **ASP.NET Core**.

# .Net Core

## Différences majeures entre .Net Framework et .Net Core

- En plus de supprimer de gros morceaux de .NET Framework afin de créer .NET Core, Microsoft a intégré .NET dans des **packages NuGet**, qui sont de petits morceaux de fonctionnalités qui peuvent être déployés indépendamment. L'objectif principal de Microsoft n'est pas de rendre .NET plus petit que .NET Framework. L'objectif est de décomposer .NET en un ensemble de composants (*to componentize*) pour prendre en charge les technologies modernes et pour avoir moins de dépendances, de sorte que le déploiement ne nécessite que les packages dont votre application a besoin.
- Entity Framework (EF) 6 est une technologie de mappage relationnel objet conçue pour fonctionner avec des données stockées dans des bases de données relationnelles telles qu'Oracle et Microsoft SQL Server. Elle a pris de l'ampleur au fil des ans, de sorte que l'API multiplateforme a été allégée, a été prise en charge pour les bases de données non relationnelles a été renommée **Entity Framework Core**.



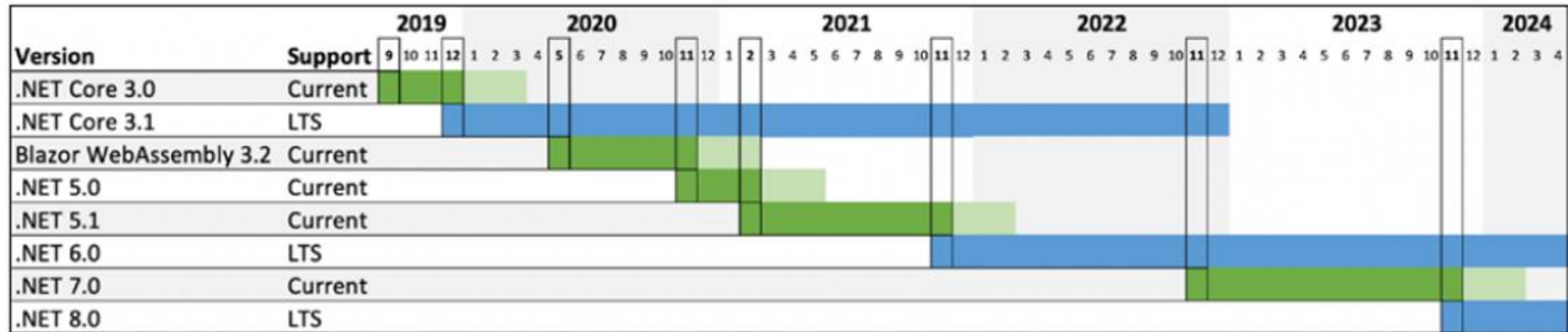
# .Net Core

## Versions de .NET Core

Version	Released	Edition	Published
.NET Core RC1	November 2015	First	March 2016
.NET Core 1.0	June 2016		
.NET Core 1.1	November 2016		
.NET Core 1.0.4 and .NET Core 1.1.1	March 2017	Second	March 2017
.NET Core 2.0	August 2017		
.NET Core for UWP in Windows 10 Fall Creators Update	October 2017	Third	November 2017
.NET Core 2.1 (LTS)	May 2018		
.NET Core 2.2 (Current)	December 2018		
.NET Core 3.0 (Current)	September 2019	Fourth	October 2019
.NET Core 3.1 (LTS)	December 2019		
.NET 5.0 (Current)	November 2020	Fifth	November 2020
.NET 6.0 (LTS)	November 2021	Sixth	November 2021

# .Net Core

## Support des versions .NET Core



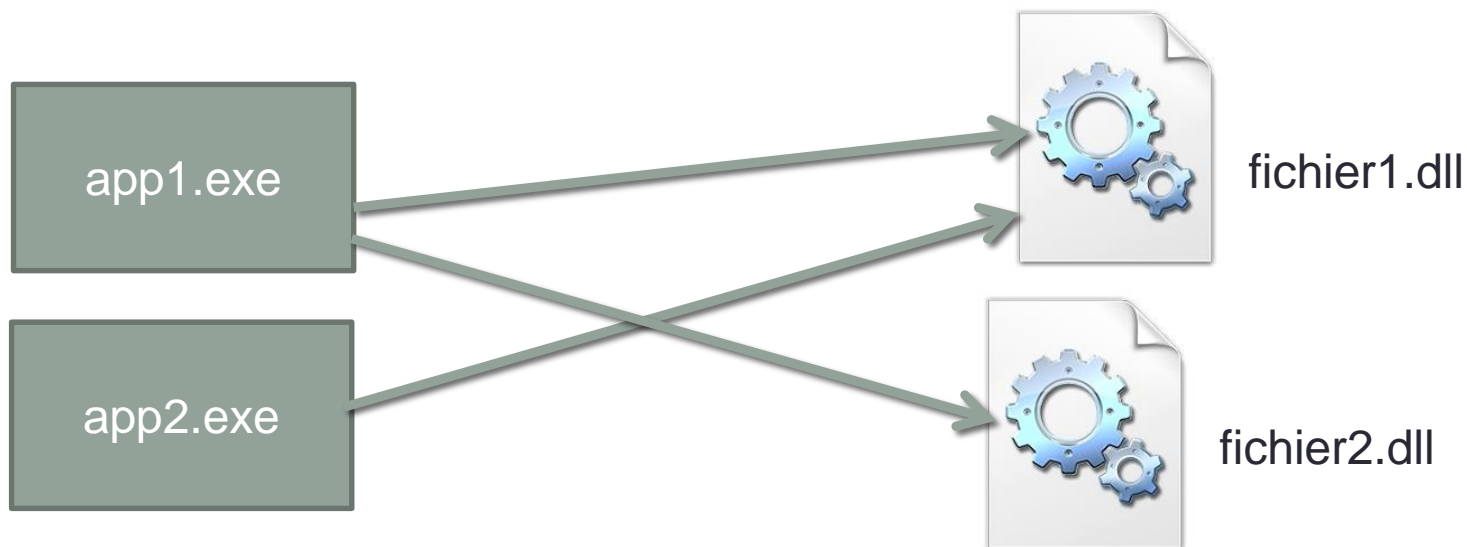
- Pour un support à long terme de Microsoft, actuellement .NET Core 3.1 est mieux que .NET 5.0. Une fois que .NET 6.0 sortira en novembre 2021, Il y aura encore plus d'un an de support avant de devoir mettre à niveau le projet vers .NET 6.0. Toutes les versions de .NET Core sont arrivées en fin de vie, à l'exception des versions LTS qui arriveront en fin de vie, comme indiqué dans la liste suivante:
  - ✓ .NET Core 2.1 arrivera en fin de vie le 21 août 2021.
  - ✓ .NET Core 3.1 arrivera en fin de vie le 3 décembre 2022.
  - ✓ .NET 6.0 arrivera en fin de vie en novembre 2024 s'il sort comme prévu en novembre 2021.

# .Net Native

- ❑ Une autre initiative .NET est appelée .NET Native. Cela compile le code C # en instructions CPU natives à l'avance (AoT), plutôt que d'utiliser le CLR pour compiler le code IL avec JIT en code natif plus tard.
- ❑ .NET Native améliore la vitesse d'exécution et réduit l'encombrement mémoire des applications car le code natif est généré au moment de la génération, puis déployé à la place du code IL.

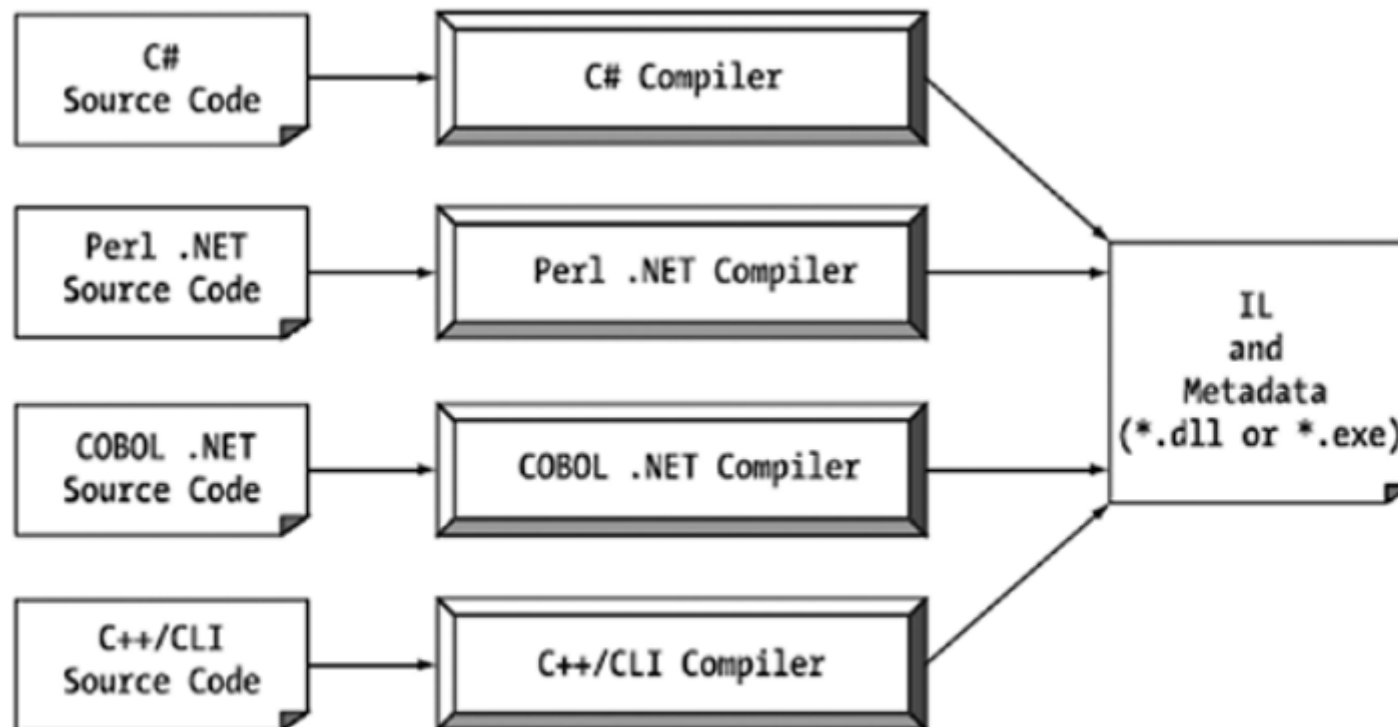
## Les .EXE et les DLL

- L'assemblage est le fichier exe ou dll produit par la compilation d'un code .NET. Un assemblage contient trois types de données : Le code CIL qui résulte de la compilation; les méta données et les ressources.
- Les .exe pourront directement être exécutés par le CLR
- Les .dll représente des bibliothèques de codes et pourront être partagées entre plusieurs applications .exe



## Les .EXE et les DLL

- Quel que soit le langage .NET avec lequel vous choisissez de programmer, sachez que même si les binaires .NET prennent la même extension de fichier que les binaires Windows non gérés (\*.dll ou \*.exe), ils n'ont absolument aucune similitude interne. Plus précisément, les binaires .NET ne contiennent pas d'instructions spécifiques à la plate-forme, mais plutôt un langage intermédiaire (IL) indépendant de la plate-forme et des métadonnées de type

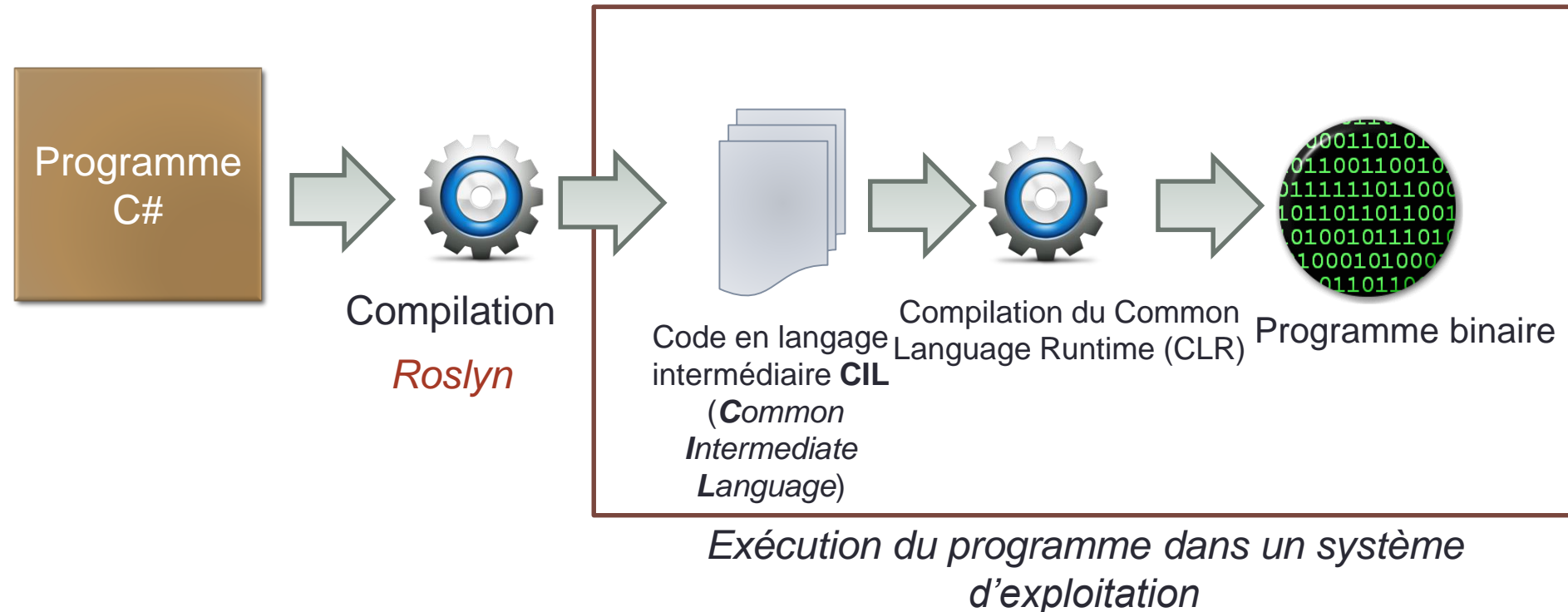


2

## **Notions de base du langage C#**

# Rappel : Compilation d'un programme C#<sup>39</sup>

## Compilation C# et exécution du CIL par le CLR



Le code CIL (ou MSIL) correspond au programme à distribuer, bien entendu le code CIL n'est pas exécutable lui-même, car l'ordinateur ne comprend que le binaire. Le code CIL sous Windows, il prend l'apparence d'un .exe comme les programmes habituels, mais il ne contient en revanche pas de binaire.

# Les variables

## Déclaration et initialisation

- Syntaxe de déclaration et initialisation d'une ou plusieurs variables est :  
**Type variable\_1[=valeur\_1], variable\_2=[valeur\_2],...;** (Syntaxe analogue à Java)
- On peut également ne pas préciser le type exact d'une variable en utilisant le **mot clé var** au lieu de *Type*:  
**var variable\_1=valeur\_1,variable\_2=valeur\_2,...;**
  - ✓ Dans ce cas la variable *variable\_i* prendra le type de la donnée *valeur\_i* qui lui est affectée.
  - ✓ L'initialisation dans ce cas est **obligatoire** afin que le compilateur puisse en déduire le type de la variable.
  - ✓ Une variable typée implicitement par le mot clé *var* ne peut pas ensuite changer de type
- La syntaxe de déclaration d'une **constante** est la suivante :  
**const type nom=valeur;**



# Les variables

## Type de variables (liste non exhaustive)

Tous les types de données primitifs en C# sont des objets dans l'espace de noms System. Pour chaque type de données, un nom court, ou alias, est fourni. Par exemple, **int** est le nom court pour **System.Int32** et **double** est l'abréviation de **System.Double**.

Nom court	.NET Class	Type	Largeur
<b>byte</b>	Byte	Entier non signé	8
<b>sbyte</b>	SByte	Entier signé	8
<b>int</b>	Int32	Entier signé	32
<b>short</b>	Int16	Entier signé	16
<b>long</b>	Int64	Entier signé	64
<b>float</b>	Single	Type virgule flottante à simple précision	32
<b>double</b>	Double	Type virgule flottante à double précision	64
<b>char</b>	Char	Caractère Unicode unique	16
<b>bool</b>	Boolean	Type booléen logique	8
<b>object</b>	Object	Type de base de tous les autres types	
<b>string</b>	String	Séquence de caractères	
<b>decimal</b>	Decimal	Type fractionnaire ou intégral précis qui peut représenter des nombres décimaux avec 29 bits significatifs	128

# Conversion chaîne de caractères <-> nombre<sup>42</sup>

nombre ➔ chaîne : *nombre.ToString()*

chaîne ➔ int : *int.Parse(chaine)*

chaîne ➔ long : *long.Parse(chaine)*

chaîne ➔ double : *double.Parse(chaîne)*

chaîne ➔ float : *float.Parse(chaîne)*

## Afficher sur la console

***Console.WriteLine(expression)*** ou ***Console.Write (expression)*** (*pas de retour à la ligne*)

où *expression* est tout type de donnée qui puisse être converti en chaîne de caractères pour être affiché à l'écran. Comme en JAVA tous les objets de C# ont une méthode *ToString()* qui est utilisée pour faire cette conversion.

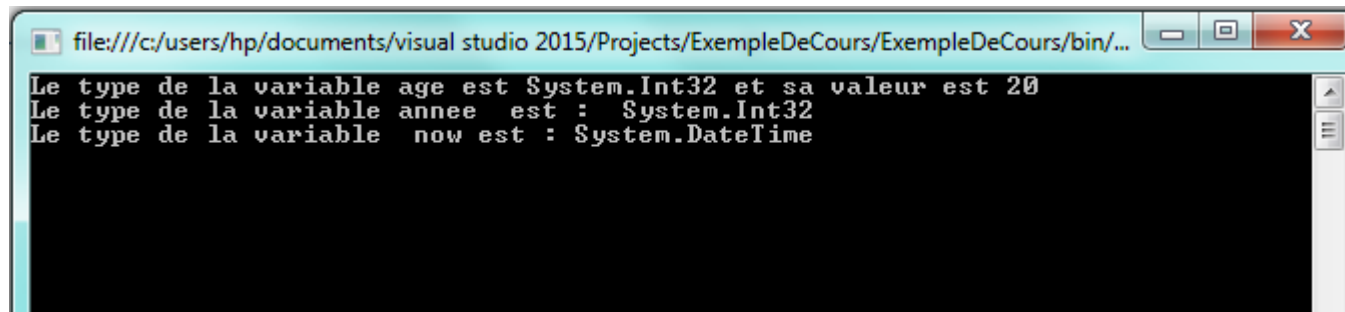
## Lecture de données tapées au clavier

```
string ligne = Console.ReadLine();
```

# Affichage sur la console et lecture au clavier<sup>44</sup>

## Exemple : initialisation des variables et affichage sur l'écran

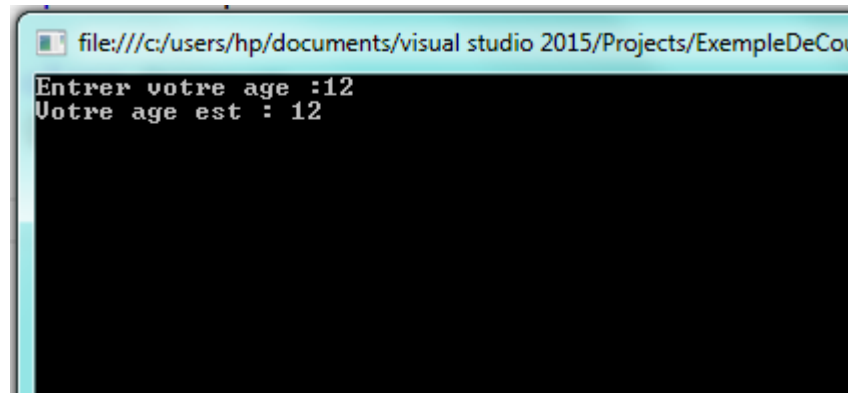
```
7 namespace ExempleDeCours
8 {
9     class Program
10    {
11        static void Main(string[] args)
12        {
13            //Déclaration et initialisation des variables
14            int age = 20;
15            Console.WriteLine("Le type de la variable age est {0} et sa valeur est {1}", age.GetType(), age);
16            //Déclaration d'une variable le type sera déduit par le compilateur
17            var annee = 2016;
18            //Affichage avec une syntaxe comme celle du Java
19            Console.WriteLine("Le type de la variable annee est : " + annee.GetType() );
20            var now = DateTime.Now;
21            Console.WriteLine("Le type de la variable now est : {0}", now.GetType());
22            Console.Read();
23        }
24    }
```



```
file:///c:/users/hp/documents/visual studio 2015/Projects/ExempleDeCours/ExempleDeCours/bin/...
Le type de la variable age est System.Int32 et sa valeur est 20
Le type de la variable annee est : System.Int32
Le type de la variable now est : System.DateTime
```

## Exemple : Lecture au clavier

```
7 namespace ExempleDeCours
8 {
9     class Program
10    {
11        static void Main(string[] args)
12        {
13            int age;
14            Console.Write("Entrer votre age :");
15            //Lecture et conversion string->int car ReadLine retourne string
16            age = int.Parse( Console.ReadLine() );
17            Console.WriteLine("Votre age est : " + age);
18
19            Console.Read();
20        }
21    }
22 }
23 }
```



The screenshot shows a console window with the following text:

```
file:///c:/users/hp/documents/visual studio 2015/Projects/ExempleDeCours/
Entrer votre age :12
Votre age est : 12
```

# Opérateurs et quelques fonctions utiles

## Opérateurs de comparaison

Opérateur	Description
==	Égalité
!=	Différence
>	Supérieur à
<	Inférieur à
>=	Supérieur ou égal
<=	Inférieur ou égal
&&	ET logique
	OU logique
!	Négation

## L'opérateur ternaire ?

Expresion\_condition ? expr1:expr2

## Quelques fonctions mathématiques (class Math):

*double Sqrt(double x) : racine carrée*

*double Cos(double x) : Cosinus*

*double Sin(double x) : Sinus*

*double Tan(double x) : Tangente*

*double Pow(double x,double y) : x à la puissance y (x>0)*

*double Exp(double x) : Exponentielle*

*double Log(double x) : Logarithme népérien*

*double Abs(double x) : valeur absolue*

## Cas des chaînes de caractères:

Comme en Java les chaînes de caractères sont des objets → utilisez **Equals** et **CompareTo** pour la comparaison (== compares les références).

### Exemple :

```
int n = s1.CompareTo(s2);
bool egal = s1.Equals(s2);
```

## Opérateur Conversion forcée (cast)

Comme en JAVA :  
(type) valeur

### Exemple :

```
Int i = 1, j=2;
double d=(double)i/j ;
```

# Les structures conditionnelles

## If/Else

```
if (civilite == " Mme")  
    Console.WriteLine(" Vous êtes une femme ");  
else if (civilite == " Mlle ")  
    Console.WriteLine(" Vous êtes une femme non mariée ");  
else if (civilite == "M.")  
    Console.WriteLine(" Vous êtes un homme ");  
else  
    Console.WriteLine("Je ne sais pas");
```

# Les structures conditionnelles

## Switch / case

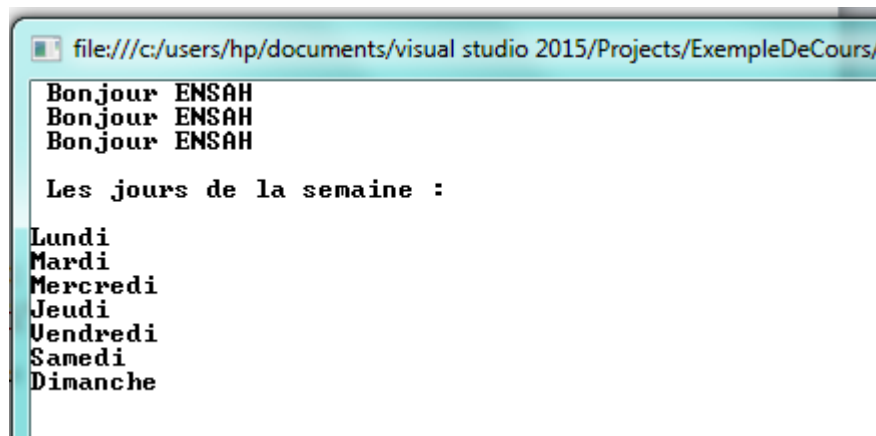
```
switch (civilite)
{
    case "M.":
        Console.WriteLine(" Bonjour monsieur ");
        break;
    case "Mme":
        Console.WriteLine(" Bonjour madame ");
        break;
    case " Mlle ":
        Console.WriteLine(" Bonjour mademoiselle ");
        break;
    default:
        Console.WriteLine(" Bonjour inconnu ");
        break;
}
```



# Les boucles

## Boucle for

```
int compteur;  
for (compteur = 0; compteur < 3; compteur++)  
{  
    Console.WriteLine(" Bonjour ENSAH");  
}  
Console.WriteLine("\n Les jours de la semaine : \n");  
string[] jours = new string[] { "Lundi", "Mardi",  
    "Mercredi ", "Jeudi", "Vendredi", "Samedi", "Dimanche" };  
int indice;  
for (indice = 0; indice < 7; indice++)  
{  
    Console.WriteLine(jours[indice]);  
}
```



```
file:///c:/users/hp/documents/visual studio 2015/Projects/ExempleDeCours/  
Bonjour ENSAH  
Bonjour ENSAH  
Bonjour ENSAH  
  
Les jours de la semaine :  
Lundi  
Mardi  
Mercredi  
Jeudi  
Vendredi  
Samedi  
Dimanche
```

## Boucle foreach

```
string[] jours = new string[] { " Lundi ", " Mardi ", " Mercredi ",  
    "Jeudi ", " Vendredi ", " Samedi ", " Dimanche " };
```

```
Console.WriteLine("\n ## avec un tableau : ## \n ");
```

```
foreach (string jour in jours)  
{  
    Console.WriteLine(jour);  
}
```

***Attention, la boucle foreach  
est une boucle en lecture  
seule!!***

```
Console.WriteLine("\n ## avec une liste : ## \n ");  
List<string> joursList = new List<string> { " Lundi ", " Mardi ",  
    "Mercredi ", " Jeudi ", " Vendredi ", " Samedi ", " Dimanche " };  
foreach (string jour in joursList)  
{  
    Console.WriteLine(jour);  
}
```



```
file:///c:/users/hp/documents/visual studio 2015/Projects/ExempleDeCours  
  
## avec un tableau : ##  
Lundi  
Mardi  
Mercredi  
Jeudi  
Vendredi  
Samedi  
Dimanche  
  
## avec une liste : ##  
Lundi  
Mardi  
Mercredi  
Jeudi  
Vendredi  
Samedi  
Dimanche
```

# Les boucles

## Boucle while

```
int i = 0;
while (i < 3)
{
    Console.WriteLine(" Bonjour Al Hoceima");
    i++;
}
```

## Boucle do...while

```
int i = 0;
do
{
    Console.WriteLine(" Bonjour ENSAH");
    i++;
}
while (i < 50);
```

## Les instructions break et continue

(Syntaxe analogue à Java)

## Arrêt d'exécution d'un programme :

Avec la méthode `Exit` de la classe *Environment*

## Tableaux à une dimension

**Déclaration** : `Type[] tableau=new Type[n]`

**Taille** : La taille est définie par la propriété **Length** du tableau.

**Initialisation à la déclaration (Même Syntaxe que JAVA):**

```
int[] entiers={0,10,20,30};
```

**Initialisation à n'import quel point dans le programme (Même Syntaxe que JAVA):**

```
string [] jours = new string [] { " Lundi ", " Mardi ", " Mercredi ", "
Jeudi ", " Vendredi ", " Samedi ", " Dimanche " };
```

Ou

```
string [] jours = new string [7];
```

```
jours [0] = " Lundi ";
```

```
jours [1] = " Mardi ";
```

```
jours [2] = " Mercredi ";
```

```
jours [3] = " Jeudi ";
```

```
jours [4] = " Vendredi ";
```

```
jours [5] = " Samedi ";
```

```
jours [6] = " Dimanche ";
```

**Comme Java C# dispose d'une classe Array** : elle offre un ensemble de méthodes de traitement des tableaux. Exemple : `Array.Sort ( tab);` pour le tri

## Tableaux à deux dimensions

### Déclaration :

***Type[,] tableau=new Type[n,m];***

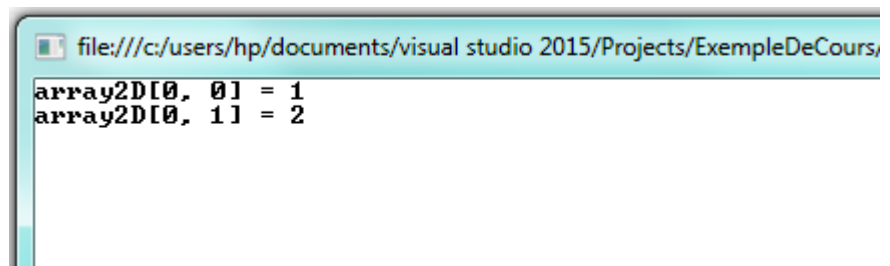
où *n* est le nombre de lignes, *m* le nombre de colonnes

- Le nombre d'éléments dans chacune des dimensions peut être obtenue par la méthode **GetLength(i)** où *i*=0 représente la dimension correspondant au 1er indice, *i*=1 la dimension correspondant au 2ième indice, ...
- Le nombre total de dimensions est obtenu avec la propriété **Rank**, le nombre total d'éléments avec la propriété **Length**.

### Exemples :

```
// Two-dimensional array.  
int[,] array2D = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };  
// The same array with dimensions specified.  
int[,] array2Da = new int[4, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
```

```
System.Console.WriteLine("array2D[0, 0] = {0} ", array2D[0, 0]);  
System.Console.WriteLine("array2D[0, 1] = {0} ", array2D[0, 1]);
```

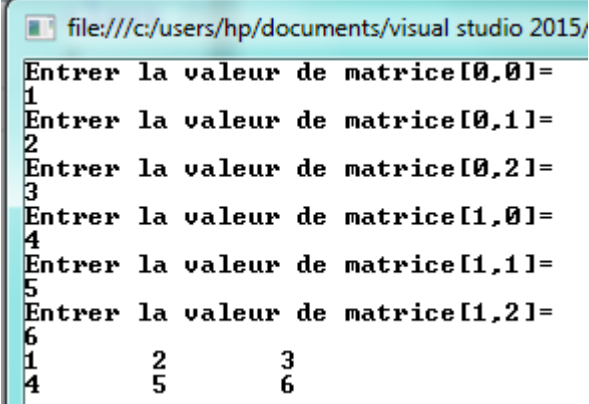


```
file:///c:/users/hp/documents/visual studio 2015/Projects/ExempleDeCours/  
array2D[0, 0] = 1  
array2D[0, 1] = 2
```

## Tableaux à deux dimensions

### Exemples :

```
// un tableau à 2 dim
double[,] matrice = new double[2,3];
//Initialisation
for (int i = 0; i < matrice.GetLength(0); i++)
{
    for (int j = 0; j < matrice.GetLength(1); j++)
    {
        Console.WriteLine("Entrer la valeur de matrice[{0},{1}]= ", i, j);
        matrice[i, j] = double.Parse(Console.ReadLine());
    }
}
//Affichage
for (int i = 0; i < matrice.GetLength(0); i++)
{
    for (int j = 0; j < matrice.GetLength(1); j++)
    {
        Console.Write("{0} \t", matrice[i, j]);
    }
    Console.WriteLine();
}
```



```
file:///c:/users/hp/documents/visual studio 2015/
Entrer la valeur de matrice[0,0]=
1
Entrer la valeur de matrice[0,1]=
2
Entrer la valeur de matrice[0,2]=
3
Entrer la valeur de matrice[1,0]=
4
Entrer la valeur de matrice[1,1]=
5
Entrer la valeur de matrice[1,2]=
6
1      2      3
4      5      6
```

## Tableaux à 3 dimensions

```
int[, ,] array1 = new int[4, 2, 3];
```

# Tableaux

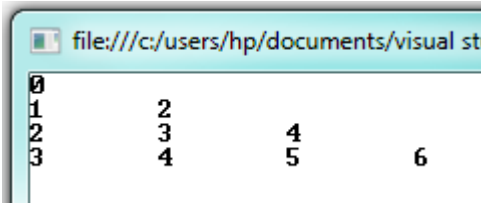
## Tableaux de tableaux

Comme en Java, C# dispose également des **tableaux de tableaux**

**Déclaration :** *Type[][] tableauDeTableau=new Type[n][];*

**Exemple :**

```
// un tableau de tableaux
int[][] TableauDeTableau = new int[4][];
for (int i = 0; i < TableauDeTableau.Length; i++)
{
    TableauDeTableau[i] = new int[i + 1];
}
// initialisation
for (int i = 0; i < TableauDeTableau.Length; i++)
{
    for (int j = 0; j < TableauDeTableau[i].Length; j++)
    {
        TableauDeTableau[i][j] = i+j;
    }
}
//Affichage
for (int i = 0; i < TableauDeTableau.Length; i++)
{
    for (int j = 0; j < TableauDeTableau[i].Length; j++)
    {
        Console.Write("{0} \t" , TableauDeTableau[i][j]);
    }
    Console.WriteLine();
}
```



0	1	2	3
1	2	3	4
2	3	4	5
3	4	5	6

# Les collections génériques

## Les collections génériques

Comme Java, C# dispose de plusieurs classes générique pour stocker des collections d'éléments. Il existe des versions génériques dans l'espace de noms *System.Collections.Generic* et des versions non génériques dans *System.Collections*. Exemple : **ArrayList** est une collection non générique et **List<T>** est une collection générique. *List<Object>* est fonctionnellement équivalente *ArrayList*.

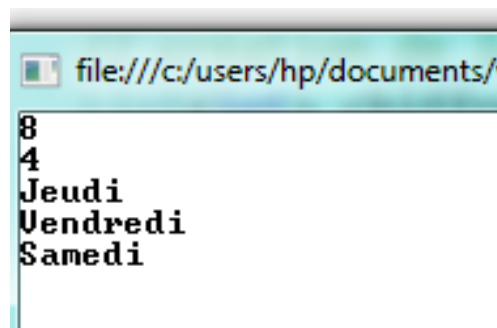
### Listes générique List<T> :

```
// création de la liste
List<int> chiffres = new List<int>();
chiffres.Add(8); // chiffres contient 8
chiffres.Add(9); // chiffres contient 8, 9
chiffres.Add(4); // chiffres contient 8, 9, 4
chiffres.RemoveAt(1); // chiffres contient 8, 4
```

```
foreach (int chiffre in chiffres)
{
    Console.WriteLine(chiffre);
}
```

```
List<string> jours = new List<string>();
jours.Add("Jeudi");
jours.Add("Vendredi");
jours.Add("Samedi");
```

```
foreach (string chiffre in jours)
{
    Console.WriteLine(chiffre);
}
```





# Les collections génériques

## Quelques Propriété et Méthodes de List<T>

**public int Count {get;}** ➔ nombre d'éléments de la liste

**public void Add(T item)** ➔ ajoute *item* à la liste

**public int BinarySearch<T>(T item)** ➔ rend la position de *item* dans la liste s'il s'y trouve sinon un nombre <0

**public void Clear()** ➔ supprime tous les éléments de la liste

**public bool Contains(T item)** ➔ rend True si *item* est dans la liste, False sinon

**public void CopyTo(T[] tableau)** ➔ copie les éléments de la liste dans *tableau*.

**public int IndexOf(T item)** ➔ rend la position de *item* dans *tableau* ou -1 si *valeur* n'est pas trouvée.

**public void Insert(T item, int index)** ➔ insère *item* à la position *index* de la liste

**public bool Remove(T item)** ➔ supprime *item* de la liste. Rend True si l'opération réussit, False sinon.

**public void RemoveAt(int index)** ➔ supprime l'élément n° *index* de la liste

**public void Sort()** ➔ trie la liste selon l'ordre défini par le type des éléments de la liste

**public T[] ToArray()** ➔ rend les éléments de la liste sous forme de tableau

# Les collections génériques

## Dictionnaire ( La classe Dictionary<TKey,TValue>)

Equivalent de Map en Java

**Création** : `Dictionary<TKey,TValue> D=new Dictionary<TKey,TValue>();`

**Accès à la valeur associée à la clé C dans le dictionnaire D** : `D[C]`.

**Ajouter une valeur dans le dictionnaire** : `D[Clé1] = Val1`.

### Quelques Propriété et Méthodes :

`public int Count {get;}` → taille de la collection

`public void Add(TKey key, TValue value)` → ajoute le couple (key, value)  
au dictionnaire

`public void Clear()` → supprime tous les couples du dictionnaire

`public bool ContainsKey (TKey key)` → rend True si *key* est une clé du dictionnaire, False sinon

`public bool ContainsValue (TValue value)` → rend True si *value* est une valeur du dictionnaire, False sinon

`public void CopyTo(T[] tableau)` → copie les éléments de la liste dans *tableau*.

`public bool Remove(TKey key)` → supprime du dictionnaire le couple de clé *key*. Rend True si l'opération réussit, False sinon.

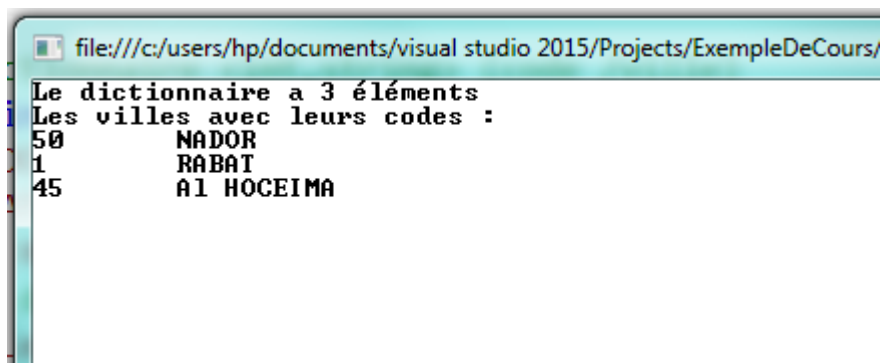
Dictionnaire

clé	valeur
Clé 1	Val 1
Clé 2	Val 3
Clé 3	Val 3

# Les collections génériques

## Exemple

```
// création d'un dictionnaire <int,string> (code /ville)
Dictionary<int, string> villeCode = new Dictionary<int, string>();
villeCode[50] = "NADOR";
villeCode[1] = "RABAT";
villeCode[45] = "Al HOCEIMA";
// nbre d'éléments dans le dictionnaire
Console.WriteLine("Le dictionnaire a " + villeCode.Count + " éléments");
Console.WriteLine("Les villes avec leurs codes :");
foreach (int code in villeCode.Keys)
{
    Console.WriteLine( code + " \t " + villeCode[code]);
}
```



```
file:///c:/users/hp/documents/visual studio 2015/Projects/ExempleDeCours/
Le dictionnaire a 3 éléments
Les villes avec leurs codes :
50      NADOR
1       RABAT
45      Al HOCEIMA
```

# Les énumérations

## Les énumérations

Comme en Java, C# dispose des énumérations. Une énumération est un type de données dont le domaine de valeurs est un ensemble de constantes entières.

**Exemple** : Considérons les mentions de la délibération : *Passable, Assez Bien, Bien, Très Bien, Excellent*.

On peut alors définir une énumération pour ces cinq constantes :

```
enum Mentions { Passable, Assez_Bien, Bien, Très_Bien, Excellent };
```

```
// une variable qui prend ses valeurs dans l'énumération Mention
```

```
Mentions maMention = Mentions.Passable;
```

```
// affichage valeur variable
```

```
Console.WriteLine("mention=" + maMention);
```

```
// test avec valeur de l'énumération
```

```
if (maMention == Mentions.Passable)
```

```
{
```

```
    Console.WriteLine("Vous pouvez faire mieux ...");
```

```
}
```

```
// liste des mentions sous forme de chaînes
```

```
foreach (Mentions m in Enum.GetValues(maMention.GetType()))
```

```
{
```

```
    Console.WriteLine(m);
```

```
}
```

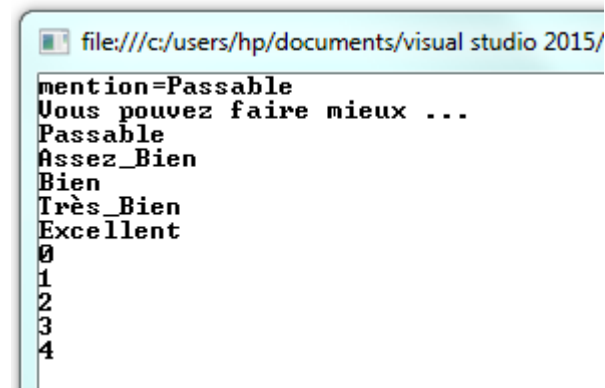
```
//liste des mentions sous forme d'entiers
```

```
foreach (int m in Enum.GetValues(typeof(Mentions)))
```

```
{
```

```
    Console.WriteLine(m);
```

```
}
```



```
file:///c:/users/hp/documents/visual studio 2015/
mention=Passable
Vous pouvez faire mieux ...
Passable
Assez_Bien
Bien
Très_Bien
Excellent
0
1
2
3
4
```

# Inclure un espace de nom (Instruction using)<sup>61</sup>

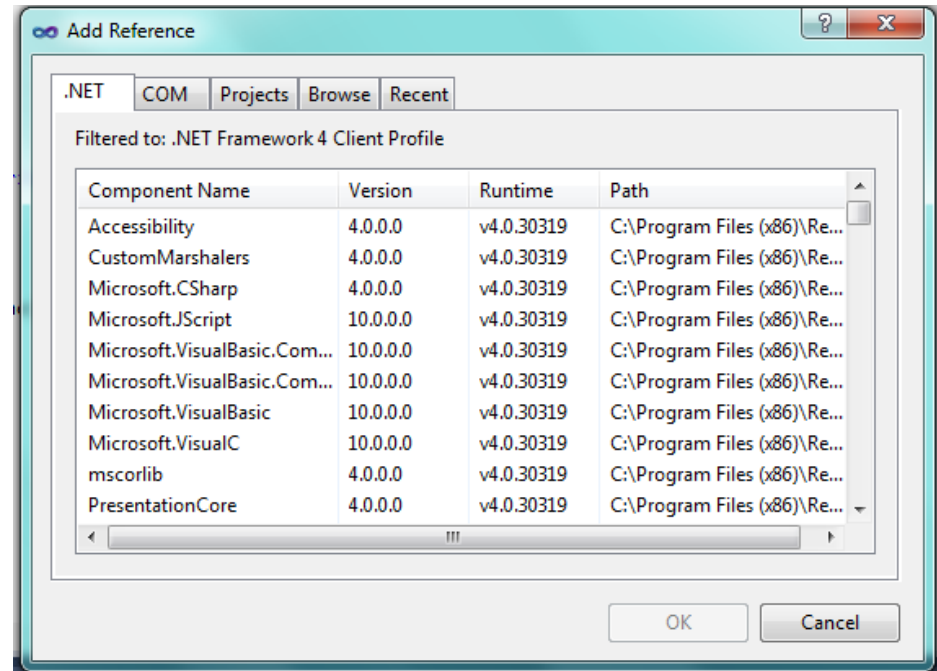
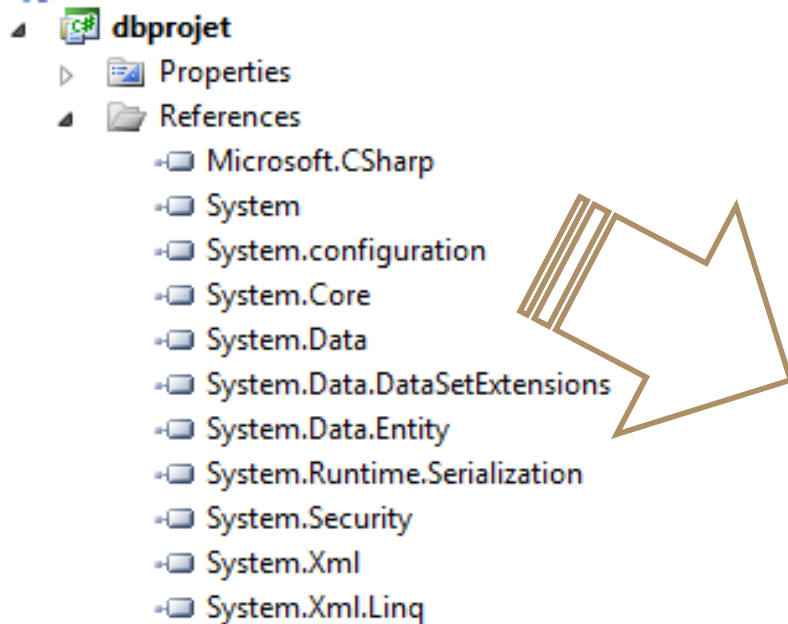
## Instruction using

On utilise le mot clé **using** pour inclure un espace de nom comme raccourci dans le programme, ce qui permet de ne pas avoir à préfixer les types de leurs espaces de noms complets

### Exemple :

**System.DateTime.Now**  **using System ;  
DateTime.Now**

## Référencer une assembly



les assemblages possèdent des fragments de code compilés en langage intermédiaire CIL. Ils sont réutilisables et se trouvent dans des fichiers dont l'extension est .dll.

Le framework .NET est composé d'une multitude d'assemblages qui sont installés sur le système d'exploitation

3

## La POO en langage C#

La programmation orientée objet en C# est très proche à celle du langage JAVA au niveau des concepts et notions et également au niveau syntaxe.

Cette section constitue donc un rappel des notions de la POO en mettant le point sur les petites différences qui existent entre la syntaxe du Java et celle du C#.



## Une première classe en C#

```
7 namespace ExempleDeCours
8 {
9     public class Etudiant
10    {
11        private string nom;
12        private string prenom;
13        private int age;
14        public Etudiant()
15        {
16            Console.WriteLine("Je suis le constructeur par défaut");
17        }
18        public Etudiant(string nom, string prenom, int age)
19        {
20            Console.WriteLine("Je suis le constructeur avec arguments");
21            InitialiseEtudiant(nom, prenom, age);
22        }
23        private void InitialiseEtudiant(string nom, string prenom, int age)
24        {
25            this.nom = nom;
26            this.prenom = prenom;
27            this.age = age;
28        }
29        public void afficheEtudiant()
30        {
31            Console.WriteLine(" Nom : {0} \n Prénom : {1} \n Age : {2} ", nom, prenom, age);
32        }
33
34        public bool plusAgeQue(int age)
35        {
36            if (this.age > age) return true;
37
38            return false;
39        }
40    }
41 }
42 }
```

# Les classes

## Une première classe en C#

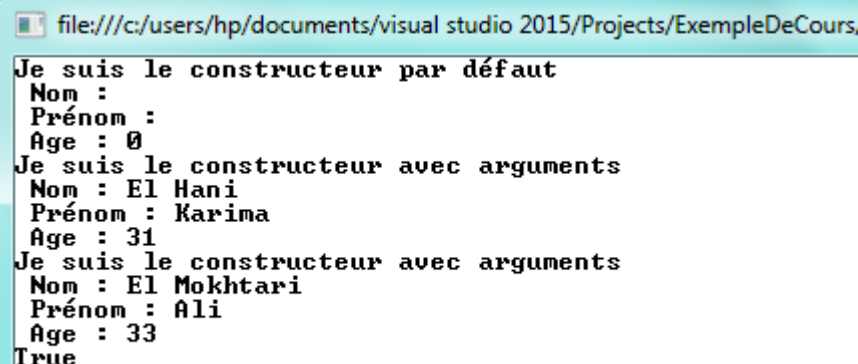
```
//Méthode de test
public static void Main(string[] args)
{
    Etudiant et0 = new Etudiant();
    et0.afficheEtudiant();

    Etudiant et1 = new Etudiant("El Hani", "Karima", 31);
    et1.afficheEtudiant();

    Etudiant et2 = new Etudiant("El Mokhtari", "Ali", 33);
    et2.afficheEtudiant();

    Console.WriteLine(et2.plusAgeQue(30));

    Console.Read();
}
```



```
file:///c:/users/hp/documents/visual studio 2015/Projects/ExempleDeCours
Je suis le constructeur par défaut
Nom :
Prénom :
Age : 0
Je suis le constructeur avec arguments
Nom : El Hani
Prénom : Karima
Age : 31
Je suis le constructeur avec arguments
Nom : El Mokhtari
Prénom : Ali
Age : 33
True
```

## Passage de paramètres à une méthode (cas des types simples)

```
7 namespace ExempleDeCours
8 {
9     public class Etudiant
10    {
11        private string nom;
12        private string prenom;
13        private int age;
14
15        public Etudiant()...
19        public Etudiant(string nom, string prenom, int age)...
24        private void InitialiseEtudiant(string nom, string prenom, int age)...
30        public void afficheEtudiant()...
34
35        public bool plusAgeQue(int age)...
42        public void changeAge(int page)
43        {
44            page = 12;
45            this.age = page;
46        }
47        //Méthode de test
48        public static void Main(string[] args)
49        {
50            int agetest = 36;
51            Etudiant et1 = new Etudiant("El Hani", "Karima", 31);
52            et1.afficheEtudiant();
53            et1.changeAge(agetest);
54            Console.WriteLine("\nAprès la modification :");
55            et1.afficheEtudiant();
56            Console.WriteLine("agetest=" + agetest);
57            Console.Read();
58        }
59    }
60 }
```

Par défaut le passage se fait  
par valeur



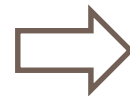
```
file:///c:/users/hp/documents/visual studio 2015/Projec
Je suis le constructeur avec arguments
Nom : El Hani
Prénom : Karima
Age : 31

Après la modification :
Nom : El Hani
Prénom : Karima
Age : 12
agetest=36
```

## Passage par référence avec le mot clé ref(cas des types simples)

```
public void changeAge(ref int page)
{
    page = 12;
    this.age = page;
}
//Méthode de test
public static void Main(string[] args)
{
    int agetest = 36;
    Etudiant et1 = new Etudiant("El Hani", "Karima", 31);
    et1.afficheEtudiant();
    et1.changeAge(ref agetest);
    Console.WriteLine("\nAprès la modification :");
    et1.afficheEtudiant();
    Console.WriteLine("agetest=" + agetest);
    Console.Read();
}
```

**Passage par référence  
et la variable *agetest* change de valeur**



```
file:///c:/users/hp/documents/visual studio 2015/Projects/
Je suis le constructeur avec arguments
Nom : El Hani
Prénom : Karima
Age : 31

Après la modification :
Nom : El Hani
Prénom : Karima
Age : 12
agetest=12
```

## Passage par référence avec le mot clé out (cas des types simples)

Le mot clé out entraîne le passage des arguments par référence. La situation est similaire à celle du mot clé ref, sauf que ref nécessite que la variable soit initialisée avant d'être transmise. Pour utiliser un paramètre out, la définition de la méthode et la méthode d'appel doivent utiliser explicitement le mot clé out.

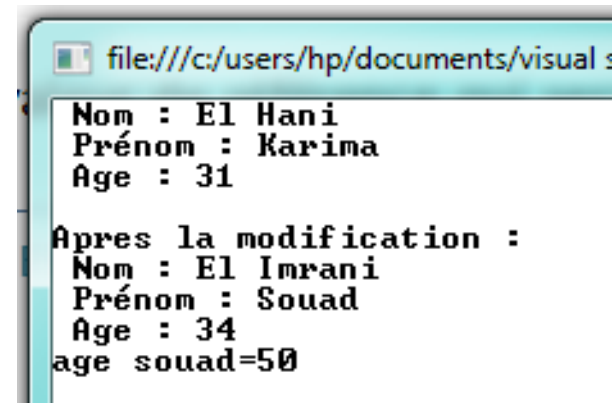
```
class OutExample {  
    static void Method(out int i)  
    {  
        i = 44;  
    }  
    static void Main()  
    {  
        int value;  
        Method(out value);  
        // value est maintenant = 44  
    }  
}
```

## Passage de paramètres à une méthode (cas des objets)

Comme en Java, dans ce cas c'est la valeur de la référence qui sera copiée dans le paramètre formel de la méthode

```
public void InitialiseEtudiant(Etudiant etd)
{
    this.nom = etd.nom;
    this.prenom = etd.prenom;
    this.age = etd.age;

    etd.age = 50;
}
//Méthode de test
public static void Main(string[] args)
{
    Etudiant et1 = new Etudiant("El Hani", "Karima", 31);
    Etudiant et2 = new Etudiant("El Imrani", "Souad", 34);
    et1.afficheEtudiant();
    et1.InitialiseEtudiant(et2);
    Console.WriteLine("\nAprès la modification :");
    et1.afficheEtudiant();
    Console.WriteLine("age souad=" + et2.age);
    Console.Read();
}
```



```
file:///c:/users/hp/documents/visual s
Nom : El Hani
Prénom : Karima
Age : 31

Après la modification :
Nom : El Imrani
Prénom : Souad
Age : 34
age souad=50
```

## Visibilité

Visibilité	Description
<b>public</b>	Accès non restreint
<b>protected</b>	Accès depuis la même classe ou depuis une classe dérivée
<b>private</b>	Accès uniquement depuis la même classe
<b>internal</b>	Accès restreint à la même assembly
<b>protected internal</b>	Accès restreint à la même assembly ou depuis une classe dérivée

# Les classes

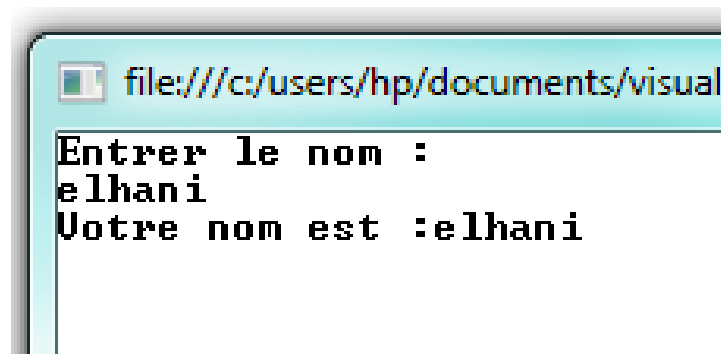
## Méthodes de lecture et d'écriture des attributs privés

Il existe deux méthodes :

- En utilisant les accesseurs et les modificateur (getters/setters) comme en Java
- En utilisant **les propriétés** : Celles-ci permettent de manipuler des attributs privés comme s'ils étaient publics.

```
public class Etudiant
{
    private string nomEtudiant;
    public string nom
    {
        get
        {
            return nomEtudiant;
        }
        set
        {
            nomEtudiant = value;
        }
    }
}

class Test
{
    //Méthode de test
    public static void Main(string[] args)
    {
        Etudiant et1 = new Etudiant();
        Console.WriteLine("Entrer le nom :");
        et1.nom = Console.ReadLine();
        Console.WriteLine("Votre nom est :" + et1.nom);
        Console.ReadLine();
    }
}
```



```
file:///c:/users/hp/documents/visual :
Entrer le nom :
elhani
Votre nom est :elhani
```



## Les propriétés

### *Autre syntaxe :*

```
public string nom {get; set;}  
public string nom {get; private set;}  
public string nom {private get; set;}
```

```
Voiture voiture = new Voiture ();  
voiture.Couleur = " Noir";  
voiture.Marque = " GOLF";  
voiture.Vitesse = 140;
```



```
Voiture voiture = new Voiture { Couleur =  
"Noir", Marque = " GOLF", Vitesse = 140};
```

**Astuce :** Pour générer les propriétés utiliser la « **snippet** » *prop*

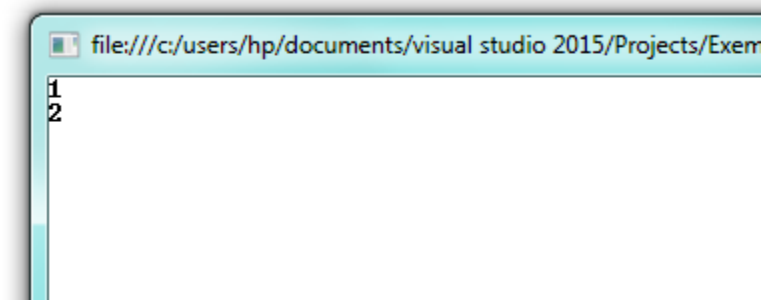
## Les méthodes et attributs de classe :

```
public class Etudiant
{
    private static int nbrEtd = 0;
    private string nomEtudiant;
    public Etudiant(string pNom)
    {
        nomEtudiant = pNom;
        nbrEtd++;
    }

    public static void getNbrEtd()
    {
        Console.WriteLine(nbrEtd);
    }
    //Méthode de test
    public static void Main(string[] args)
    {
        Etudiant e1 = new Etudiant("samir");
        Etudiant.getNbrEtd();
        Etudiant e2 = new Etudiant("mohamed");
        Etudiant.getNbrEtd();

        Console.Read();
    }
}
```

(Pas de différence avec Java)



```
class Animal
```

```
{
```

```
    public int NombreDePattes { get; set; }
```

```
    public void Respirer()
```

```
{
```

```
        Console.WriteLine("Je respire ");
```

```
}
```

```
}
```

*Animal*



*Chien*

```
public class Chien : Animal
```

```
{
```

```
    public void Aboyer()
```

```
{
```

```
        Console.WriteLine(" Haow !");
```

```
}
```

```
}
```

```
static void Main(string[] args)
{
```

```
    Animal animal = new Animal { NombreDePattes = 4 };
```

```
    animal.Respirer();
```

```
    Console.WriteLine();
```

```
    Chien chien = new Chien { NombreDePattes = 4 };
```

```
    chien.Respirer();
```

```
    chien.Aboyer();
```

```
    Console.Read();
```

```
}
```

file:///c:/users/h

Je respire

Je respire

Haow !

```
public class Animal
{
    protected string prenom;

    public int NombreDePattes { get; set; }

    public void Respirer()
    {
        Console.WriteLine("Je respire ");
    }
}

public class Chat : Animal
{
    public Chat(string prenomDuChat)
    {
        prenom = prenomDuChat;
    }

    public void Miauler()
    {
        Console.WriteLine(" Miaou ");
    }
}

public class Chien : Animal
{
    public Chien(string prenomDuChien)
    {
        prenom = prenomDuChien;
    }

    public void Aboier()
    {
        Console.WriteLine(" Haow !");
    }
}

class Program
{
    static void Main(string[] args)
    {
        List<Animal> animaux = new List<Animal>();
        Animal dingo = new Chien("Dingo");
        Animal tom = new Chat("Tom");

        animaux.Add(dingo);
        animaux.Add(tom);

        foreach (Animal animal in animaux)
        {
            animal.Respirer();
        }

        Console.Read();
    }
}
```



```
file:///c:/users/hp/document
Je respire
Je respire
```

## Mot clé base

équivalent du mot **super** en Java, dans Java super doit être la première instruction, dans C# le mot **base** dans être placé dans la signature de la méthode.

```
public class Animal
```

```
{
    protected string prenom;

    public int NombreDePattes { get; set; }

    public Animal(string prenomAnimal)
    {
        prenom = prenomAnimal;
    }

    public void Respirer()
    {
        Console.WriteLine("Je respire ");
    }
}
```

```
public class Chien : Animal
```

```
{
    public Chien(string prenomDuChien) : base ( prenomDuChien )
    {
    }

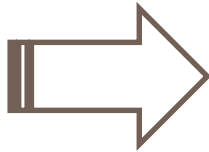
    public void Aboier()
    {
        Console.WriteLine(" Haow !");
    }
}
```

## Appeler un constructeur à partir d'un autre constructeur

```
public class Voiture
{
    private int vitesse;

    public Voiture()
    {
        vitesse = 10;
    }

    public Voiture(int vitesseVoiture)
    {
        vitesse = vitesseVoiture;
    }
}
```



```
public class Voiture
{
    private int vitesse;

    public Voiture() : this(10)
    {
    }

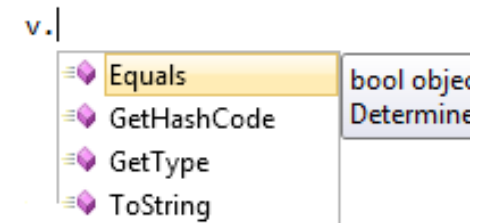
    public Voiture(int vitesseVoiture)
    {
        vitesse = vitesseVoiture;
    }
}
```

### Remarque :

comme en Java :

- Toutes les classes héritent de Object
- On retrouve également les méthodes Equals , GetHashCode (hashCode en Java) et ToString.
- Il est impossible de dériver de deux classes en même temps

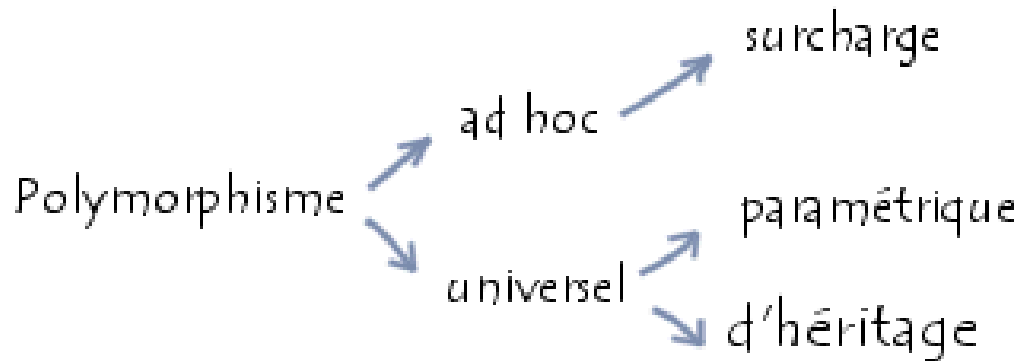
```
Voiture v = new Voiture();
```



# Polymorphisme

## Types de polymorphismes:

Il existe plusieurs sortes de polymorphismes fondamentalement différents :



On distingue généralement de trois types de polymorphisme :

- ❖ **Le polymorphisme ad hoc** (en programmation on parle de surcharge ou en anglais overloading)
- ❖ **Le polymorphisme paramétrique** (également généricité ou en anglais template)
- ❖ **Le polymorphisme d'héritage** (également redéfinition, spécialisation ou en anglais overriding)

## Redéfinition :

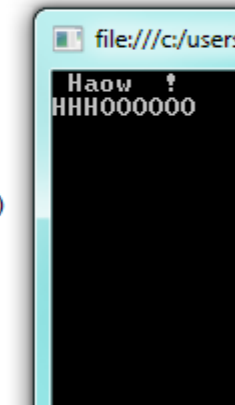
```
public class Chien : Animal
{
    public virtual void Aboyer()
    {
        Console.WriteLine(" Haow !");
    }
}
```

```
public class ChientMarocain : Chien
{
    public override void Aboyer()
    {
        Console.WriteLine("HHHOOOOOOO ");
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        List<Animal> animaux = new List<Animal>();
        Animal dingo = new Chien();
        Animal cm = new ChientMarocain();

        animaux.Add(dingo);
        animaux.Add(cm);

        foreach (Animal animal in animaux)
        {
            Chien c = (Chien)animal;
            c.Aboyer();
        }
    }
}
```





## Surcharge :

On applique les même règles que celles du Java

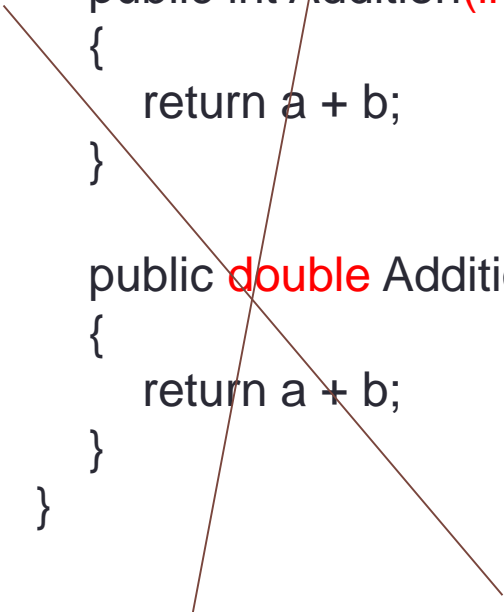
```
public class Math
{
    public int Addition(int a, int b)
    {
        return a + b;
    }

    public double Addition(double a, double b)
    {
        return a + b;
    }
}
```

Surcharge de la méthode **Addition**

```
public class Math
{
    public int Addition(int a, int b)
    {
        return a + b;
    }

    public double Addition(int a, int b)
    {
        return a + b;
    }
}
```



**Erreur de compilation**

## Conversion :

❖ Nous pouvons convertir un objet en un autre seulement s'il est une sorte de l'autre objet, Nous pourrions alors convertir le chien en animal dans la mesure où le chien est une sorte d'animal (car chien hérite de animal), Par contre, il est impossible de convertir un chien en voiture, car il n'y a pas de relation d'héritage entre les deux

❖ Mot clé **is** :

Il est cependant possible de tester si une variable correspond à un objet grâce au mot-clé **is** (équivalent de *instanceof* en Java) :

```
if ( animal is Chien )  
{  
    Chien c = ( Chien ) animal ;  
    c. Aboyer ();  
}
```

## Le cast dynamique

Nous pouvons aussi utiliser le cast dynamique. (le mot-clé **as**)

Ce cast dynamique vérifie que l'objet est bien convertible

Si c'est le cas, alors il fait un cast explicite pour renvoyer le résultat de la conversion, sinon, il renvoie une référence nulle.

```
Chien c1 = animal as Chien ;
```

```
if (c1 != null )  
{  
    c1. Aboyer ();  
}
```

# Comparaison des objets

L'opérateur « == » comme en Java compare les références

La comparaison d'égalité entre deux objets, il faut redéfinir la méthode **Equals()**

```
public class Voiture
{
    public string Couleur { get; set; }
    public string Marque { get; set; }
    public int Vitesse { get; set; }

    public override bool Equals(object obj)
    {
        Voiture v = obj as Voiture;
        if (v == null)
            return false;
        return Vitesse == v.Vitesse && Couleur == v.Couleur &&
            Marque == v.Marque;
    }
}
```

# Classes et méthodes abstraites

- ❖ Pour déclarer une méthode comme étant abstraite, il faut utiliser le mot-clé **abstract**
- ❖ Une classe qui contient au moins une méthode abstraite est forcément abstraite.

```
public abstract class Animal
{
    public abstract void SeDeplacer();

    public void Aboier()
    {
        Console.WriteLine("Hoaw");
    }
}
```

```
public class Chien : Animal
{
    public override void SeDeplacer()
    {
        Console.WriteLine("Je me déplace  
comme un chien");
    }
}
```

**il n'est pas possible d'instancier un objet Animal**

Une interface ne peut contenir que des méthodes et des propriétés. (Un contrat)

```
public interface IVolant
{
    int NombrePropulseurs { get; set; }
    void Voler ();
}
```

Les objets qui choisiront d'implémenter cette interface **seront obligés d'avoir une propriété entière *NombrePropulseurs* et une méthode *Voler()*** qui ne renvoie rien

Une classe peut implémenter plusieurs interfaces

```
public interface IVolant  
{  
    void Voler ();  
}
```

```
public interface IRoulant  
{  
    void Rouler ();  
}
```

```
public class Avion : IVolant , IRoulant  
{  
    public void Voler ()  
    {  
        Console . WriteLine ("Je vole ");  
    }  
  
    public void Rouler ()  
    {  
        Console . WriteLine ("Je Roule ");  
    }  
}
```

# Les classes partielles

Les classes partielles offrent la possibilité de définir une classe en plusieurs fois. En général, ceci est utilisé pour définir une classe sur plusieurs fichiers, si par exemple la classe devient très longue. Il pourra éventuellement être judicieux de découper la classe en plusieurs fichiers pour regrouper des fonctionnalités qui se ressemblent. On utilise pour cela le mot-clé **partial**

```
public partial class Voiture
{
    public string Couleur { get; set; }
    public string Marque { get; set; }
    public int Vitesse { get; set ; }
}
```

```
public partial class Voiture
{
    public string Rouler ()
    {
        return "Je roule à " + Vitesse + " km/h";
    }
}
```



## Les classes internes

C#, comme Java, dispose de la notion de classe interne, ainsi on peut avoir des classes définies à l'intérieur d'autres classes. Cela peut être utile si on souhaite restreindre l'accès d'une classe uniquement à sa classe mère. Dans l'exemple ci-dessous la classe Adresse n'est visible que pour la classe Personne

```
public class Personne
{
    private Adresse adresse = new Adresse{ville = "tanger", numRue = 12};
    public void showInfos()
    {
        Console.WriteLine(adresse);
    }

    private class Adresse
    {
        public int numRue { get; set; }
        public string ville { get; set; }
    }
}
```

# Les structures

- Une structure est un objet qui ressemble beaucoup à une classe, mais qui possède des restrictions.
- Les structures sont des types valeur (donc contrairement aux classes, les structures contiennent directement la valeur de l'objet) qui sont optimisés par le framework .NET.
- Il n'est pas possible d'utiliser l'héritage avec les structures.
- Comme pour les classes, les structures possèdent des propriétés, des méthodes.
- Comme pour les classes, il est possible d'avoir des constructeurs sur une structure, à l'exception du constructeur sans arguments qui est interdit.
- Les structures vont être utiles pour stocker de petits objets amenés à être souvent manipulés, comme les **int** ou les **bool** que nous avons déjà vus.
- Étant gérées en mémoire différemment, les structures sont optimisées pour améliorer les performances des petits objets.
- Comme il s'agit d'un type valeur, à chaque fois que nous passerons une structure en paramètres d'une méthode, une copie de l'objet sera faite. (Donc si la structure est « grosse » les performances peuvent se dégrader)

# Les structures

## Exemple

```
public struct PersonneStruct
{
    public string Prenom { get; set; }
    public int Age { get; set; }

    public override string ToString()
    {
        return Prenom + " a " + Age + " ans";
    }
    public static void Main(string[] args)
    {
        PersonneStruct p = new PersonneStruct();
        p.Prenom = " Karimi";
        p.Age = 22;
        Console.WriteLine(p.ToString());
        Console.Read();
    }
}
```

Une structure ne peut hériter que de Object, ainsi elle peut spécialiser ToString





4

## Notions avancées

Comme Java, C# dispose d'un mécanisme pour créer des type génériques. Avec les génériques.

On peut créer des méthodes ou des classes qui sont indépendantes d'un type. La syntaxe est légèrement différente de celle employée avec Java

## Exemple d'une méthode générique :

```
class ExempleGenerique
{
    //Cette méthode générique permet d'afficher
    //Les informations d'un objet de n'importe quel type
    public static void printObjectInfos<T>(T obj)
    {
        Console.WriteLine("Type :" + obj.GetType());
        Console.WriteLine("Représentation sous forme string :" + obj.ToString());
    }
    public static void Main(string[] args)
    {
        Prof prof = new Prof();
        prof.nom = "Boudaa";
        prof.prenom = "Tarik";
        printObjectInfos(prof);

        Etudiant etd = new Etudiant();
        etd.nom = "Eloualidi";
        etd.prenom = "Fikri";
        printObjectInfos(etd);

        Console.Read();
    }
}
```

*La méthodes générique accepte en paramètre un objet quelconque*

```
file:///C:/Users/hp/Documents/Visual Studio 2015/Projects/ExempleDeCours/ExempleDeCours/bin...
Type :ExempleDeCours.Prof
Représentation sous forme string :Nom Prof :Boudaa Prénom Prof :Tarik
Type :ExempleDeCours.Etudiant
Représentation sous forme string :Nom Etudiant :Eloualidi Prénom :Fikri
```

Nous pouvons avoir autant de paramètres génériques que nous le voulons :

## Exemple :

```
class ExempleGenerique
{
    //Cette méthode générique a deux paramètres
    public static string concatToString<T,U>(T obj1, U obj2)
    {
        return obj1.ToString() + " " + obj2.ToString();
    }
    public static void Main(string[] args)
    {
        Prof prof = new Prof { nom = "Boudaa", prenom = "Tarik" };
        Etudiant etd = new Etudiant { nom = "Eloualidi", prenom = "Fikri" };

        string contactRes = concatToString(prof,etd);
        Console.WriteLine(contactRes);

        Console.Read();
    }
}
```

Comme en Java on peut définir des classes et des interfaces génériques

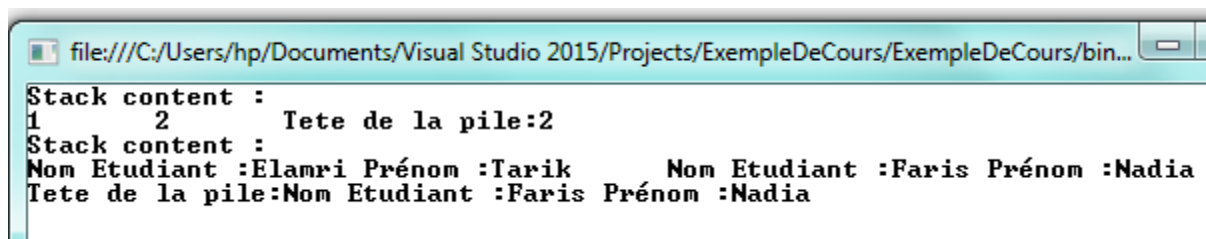
## Exemple classe générique

```
public class GenericStack<T>
{
    private int capacity;
    private int size;
    private T[] content;
    public GenericStack(int pCapacity)
    {
        size = 0;
        capacity = pCapacity;
        content = new T[pCapacity];
    }
    public void stacking(T o)
    {
        if (size >= capacity) throw new StackOverflowException();
        content[size++] = o;
    }
    public T getStackHead()
    {
        return content[size - 1];
    }
    public void print()
    {
        Console.WriteLine("Stack content :");
        for(int i = 0; i < size; i++)
        {
            Console.Write("{0}\t", content[i]);
        }
    }
}
```

## Exemple d'une classe générique

```
public static void Main(string[] args)
{
    //Pile d'entier
    GenericStack<int> IntegerStack = new GenericStack<int>(5);
    IntegerStack.stacking(1);
    IntegerStack.stacking(2);
    IntegerStack.print(); //on affiche le contenu de la pile
    Console.WriteLine("Tete de la pile:"+IntegerStack.getStackHead()); //On affiche la tete de la pile
    //Pile d'Etudiant
    GenericStack<Etudiant> studentStack = new GenericStack<Etudiant>(2);
    studentStack.stacking(new Etudiant {nom="Elamri",prenom="Tarik"});
    studentStack.stacking(new Etudiant { nom = "Faris", prenom = "Nadia" });
    studentStack.print(); //on affiche le contenu de la pile
    Console.WriteLine("Tete de la pile:" + studentStack.getStackHead()); //On affiche la tete de la pile

    Console.Read();
}
```



```
file:///C:/Users/hp/Documents/Visual Studio 2015/Projects/ExempleDeCours/ExempleDeCours/bin...
Stack content :
1      2      Tete de la pile:2
Stack content :
Nom Etudiant :Elamri Prénom :Tarik      Nom Etudiant :Faris Prénom :Nadia
Tete de la pile:Nom Etudiant :Faris Prénom :Nadia
```



## Exemple d'une interface générique

```
public interface GenericDAO<T, PK>
{
    void Save(T pObject);
    T findByIdentifier(PK pId);
}
```



On définit une interface générique

```
class GenericDAOImpl<T, PK> : GenericDAO<T, PK>
{
    public T findByIdentifier(PK pId)
    {
        //C'est juste un test
        T o = default(T) ;
        return o;
    }
    public void Save(T pObject)
    {
        //Implémentation de la méthode..
    }
}
```



Implémentation générique de l'interface générique

```
class GenericDAOImpl : GenericDAO<Etudiant, int>
{
    public Etudiant findByIdentifier(int pId)
    {
        throw new NotImplementedException();
    }

    public void Save(Etudiant pObject)
    {
        throw new NotImplementedException();
    }
}
```



Implémentation de l'interface générique en indiquant les valeurs des paramètres T et PK

```
class EtudiantDAOImpl : GenericDAOImpl<Etudiant,int>
{
}
```



On hérite de l'implémentation générique en indiquant les valeurs des paramètres T et PK

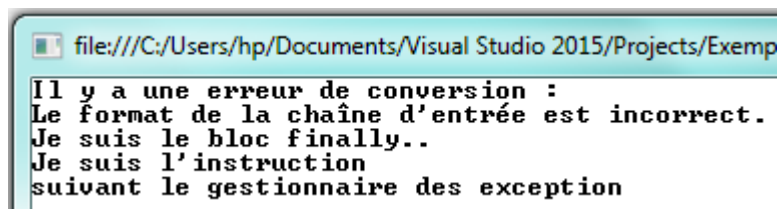
# Gestion des exceptions

Le mécanisme de gestion des exceptions en C# est presque identique à celui du Java:

```
public static void Main(string[] args)
{
    try
    {
        string chaine = "je ne suis pas un nombre";
        int valeur = Convert.ToInt32(chaine);
        Console.WriteLine(" Conversion OK");
    }
    catch (Exception ex)
    {
        Console.WriteLine("Il y a une erreur de conversion : {0}", ex.Message);
    }
    finally
    {
        Console.WriteLine("Je suis le bloc finally..");
    }

    Console.WriteLine("Je suis l'instruction suivant le gestionnaire des exception");

    Console.Read();
}
```



```
file:///C:/Users/hp/Documents/Visual Studio 2015/Projects/Exemp
Il y a une erreur de conversion :
Le format de la chaîne d'entrée est incorrect.
Je suis le bloc finally..
Je suis l'instruction
suivant le gestionnaire des exception
```

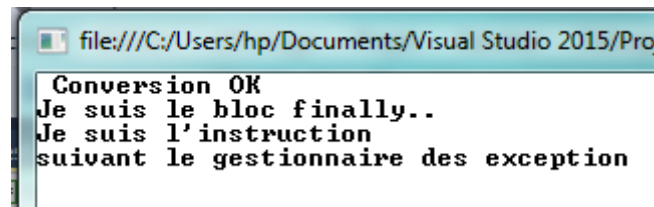
# Gestion des exceptions

```
public static void Main(string[] args)
{
    try
    {
        string chaine = "18";
        int valeur = Convert.ToInt32(chaine);
        Console.WriteLine(" Conversion OK");
    }
    catch (Exception ex)
    {
        Console.WriteLine("Il y a une erreur de conversion :\n{0}", ex.Message);
    }
    finally
    {
        Console.WriteLine("Je suis le bloc finally..");
    }

    Console.WriteLine("Je suis l'instruction \nsuivant le gestionnaire des exception");

    Console.Read();
}
```

***Comme en langage JAVA  
le bloc finally s'exécute toujours  
même si il n y a pas d'erreurs !!***



```
file:///C:/Users/hp/Documents/Visual Studio 2015/Pro
Conversion OK
Je suis le bloc finally..
Je suis l'instruction
suivant le gestionnaire des exception
```

# Gestion des exceptions

## Affichage des informations sur l'erreur

```
public static void Main(string[] args)
```

```
{
    try
    {
        string chaine = "je ne suis pas un nombre";
        int valeur = Convert.ToInt32(chaine);
        Console.WriteLine(" Conversion OK");
    }
```

```
catch (Exception ex)
```

```
{
    Console.WriteLine("Informations sur l'erreur :\n");
    Console.WriteLine(" Message d'erreur : {0}\n" , ex.Message);
    Console.WriteLine(" Pile d'appel : {0}\n", ex.StackTrace);
    Console.WriteLine(" Type de l'exception : {0}\n", ex.GetType());
}
```

```
Console.Read();
```

```
}
```

*Comme en Java on peut également afficher les informations liée à l'erreur avec la méthode ToString : ex.ToString ()*



```
file:///C:/Users/hp/Documents/Visual Studio 2015/Projects/ExempleDeCours/ExempleDeCours/bin...
Informations sur l'erreur :
Message d'erreur : Le format de la chaîne d'entrée est incorrect.
Pile d'appel : à System.Number.StringToNumber(String str, NumberStyles options, NumberBuffer& number, NumberFormatInfo info, Boolean parseDecimal)
à System.Number.ParseInt32(String s, NumberStyles style, NumberFormatInfo info)
à System.Convert.ToInt32(String value)
à ExempleDeCours.Etudiant.Main(String[] args) dans C:\Users\hp\Documents\Visual Studio 2015\Projects\ExempleDeCours\ExempleDeCours\Etudiant.cs:ligne 29
Type de l'exception : System.FormatException
```

# Gestion des exceptions

## Interceptor plusieurs exceptions

Pour intercepter plusieurs exceptions on utilise une syntaxe similaire à celle du Java

```
try
{
    //Code provoquant une exception
}
catch (FormatException ex)
{
    //Si une erreur de type FormatException
}
catch (NullReferenceException ex)
{
    //Si une erreur de
    //type NullReferenceException
}
catch (Exception ex)
{
    //Si l'erreur n'est pas de
    //type FormatException ni NullReferenceException
}
```

# Gestion des exceptions

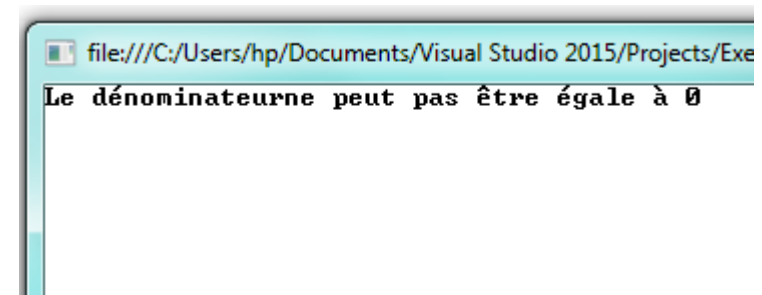
## Lever une exception

Il est possible de déclencher soi-même la levée d'une exception. La syntaxe à utiliser est comme celle du langage Java, sauf que en C# on indique pas l'exception dans la signature de la méthode avec throws.

```
public static double division(double numerateur, double denominateur)
{
    if (denominateur <= 0)
        throw new InvalidOperationException("Le dénominateur"+
            "ne peut pas être égale à 0");
    return numerateur / denominateur;
}

public static void Main(string[] args)
{
    try
    {
        division(12, 0);
    } catch (InvalidOperationException ex)
    {
        Console.WriteLine(ex);
    }

    Console.Read();
}
```



# Gestion des exceptions

## Propager une exception

Pour propager une exception on utilise la même syntaxe que celle du Java

```
public static double division(double numerateur, double denominateur)
{
    if (denominateur <= 0)
        throw new InvalidOperationException("Le dénominateur"+
            "ne peut pas être égale à 0");
    return numerateur / denominateur;
}

public static double methodQuiPropageException(double numerateur, double denominateur)
{
    double result = 0;
    try
    {
        result= division(12, 0);
    }
    catch (InvalidOperationException ex)
    {
        Console.WriteLine(ex.Message);
        //On propage l'exception
        throw ex; // ou tout simplement : throw;
        //On peut également "emballer" ex dans une autre exception
    }

    return result;
}
```

# Gestion des exceptions

## Créer une exception personnalisée

Comme en Java on peut créer des exceptions personnalisées par héritage

```
public class InvalideNombreSommetException : Exception
{
    public InvalideNombreSommetException()
    {
    }

    public InvalideNombreSommetException(string message) : base(message)
    {
    }

    public InvalideNombreSommetException(string message, Exception innerException) :
        base(message, innerException)
    {
    }

    protected InvalideNombreSommetException(SerializationInfo info, StreamingContext context) :
        base(info, context)
    {
    }
}
```



# Gestion des exceptions

## Hiérarchie des exceptions

*Pour plus d'informations sur les exceptions, leur hiérarchie et les bonnes pratiques:*

<https://msdn.microsoft.com/fr-fr/library/z4c5tckx.aspx>

[https://msdn.microsoft.com/en-US/library/seyhszts\(v=VS.80\).aspx](https://msdn.microsoft.com/en-US/library/seyhszts(v=VS.80).aspx)

- Les délégués permettent de créer des variables pointant vers des méthodes.
- Les délégués sont à la base des événements.
- On utilise les expressions lambdas pour simplifier l'écriture des délégués.
- Les événements sont un mécanisme du C# permettant à une classe d'être notifiée d'un changement.

# Les délégués (delegate)

- Les délégués en C# permettent de créer des variables spéciales qui référencent des méthodes. Ils sont proches de la notion de pointeurs de fonctions en C/C++,
- Le délégué permet de définir une signature de méthode et avec lui, on peut référencer n'importe quelle méthode qui respecte cette signature.
- En général, on utilise un délégué quand on veut passer une méthode en paramètre d'une autre méthode.

## Exemple

```
class TableSorter  
{  
  
    private delegate void DelegateSort(int[] tableau);  
  
}
```

DelegateSort est un délégué privé à la classe TableSorter qui permettra de pointer vers des méthodes qui ne retournent rien (*void*) et qui acceptent un tableau d'entiers en paramètre.

Oh..! delegate is a reference to a method

Delegate is just like functional pointer in C++  
But type Safe.. Oh..

But  
What is the purpose of Delegates?

Where can I use Delegates in our Daily  
projects?



# Les délégués (delegate)

## Exemple

```
class TableSorter
{
    private delegate void DelegateSort(int[] tableau);

    public void SorAscending(int[] tab)
    {
        Array.Sort(tab);
    }
    public void SorDescending(int[] tab)
    {
        Array.Sort(tab);
        Array.Reverse(tab);
    }
}
```

délégué qui permettra de pointer vers des méthodes qui ne retournent rien (void) et qui acceptent un tableau d'entiers en paramètre.

Deux méthodes qui peuvent être pointées par le délégué DelegateSort

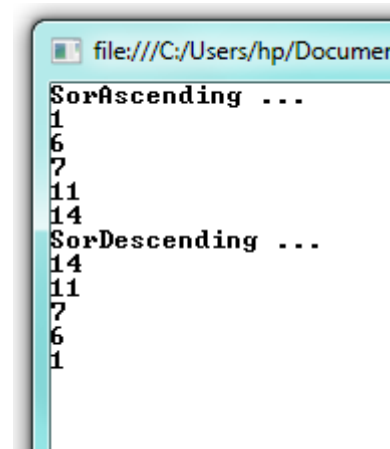
# Les délégués (delegate)

## Exemple

```
class TableSorter
{
    private delegate void DelegateSort(int[] tableau);
    public void SortAscending(int[] tab) ...
    public void SortDescending(int[] tab) ...
    public void TestMethod(int[] tab)
    {
        DelegateSort sort = SortAscending;
        Console.WriteLine("SorAscending ...");
        sort(tab);
        foreach (int i in tab)
        {
            Console.WriteLine(i);
        }
        Console.WriteLine("SorDescending ...");
        sort = SortDescending;
        sort(tab);
        foreach (int i in tab)
        {
            Console.WriteLine(i);
        }
    }
    public static void Main(string[] args)
    {
        int[] tag = new int[] { 14, 1, 6, 11, 7 };
        new TableSorter().TestMethod(tag);
        Console.Read();
    }
}
```

On déclare un délégué puis on le pointe sur la méthode SortAscending

On pointe maintenant le délégué sur la méthode SortDescending



```
file:///C:/Users/hp/Documer
SorAscending ...
1
6
7
11
14
SorDescending ...
14
11
7
6
1
```

# Les délégués (delegate)

## Exemple

```
class TableSorter
{
    public delegate void DelegateSort(int[] tableau);
    public void SortAscending(int[] tab) ...
    public void SortDescending(int[] tab) ...
    public void TestMethod(int[] tab)
    {
        SortAndPrint(tab, SortAscending);
        Console.WriteLine("SortDescending ...");
        SortAndPrint(tab, SortDescending);
    }
    public void SortAndPrint(int[] tab, DelegateSort pSortDelegate)
    {
        pSortDelegate(tab);
        foreach(int it in tab)
        {
            Console.WriteLine("{0} \t", it);
        }
    }
    public static void Main(string[] args)
    {
        int[] tag = new int[] { 14, 1, 6, 11, 7 };
        new TableSorter().TestMethod(tag);
        Console.Read();
    }
}
```

Dans cet exemple nous avons montré comment nous pouvons grandement simplifier la méthode TestMethod



```
file:///C:/Users/hp/Documen
SortAscending ...
1
6
7
11
14
SortDescending ...
14
11
7
6
1
```

## Diffusion multiple, le multicast

```
class TableSorter
{
    private delegate void DelegateSort(int[] tableau);
    public void SortAscending(int[] tab)
    {
        Console.WriteLine("SortAscending");
        Array.Sort(tab);
        foreach (int i in tab)
        {
            Console.WriteLine(i);
        }
    }
    public void SortDescending(int[] tab)
    {
        Console.WriteLine("SortAscending");
        Array.Sort(tab);
        Array.Reverse(tab);
        foreach (int i in tab)
        {
            Console.WriteLine(i);
        }
    }
    public void TestMethod(int[] tab)
    {
        DelegateSort tri = SortDescending;
        tri += SortDescending;
        tri(tab);
    }

    public static void Main(string[] args)
    {
        int[] tag = new int[] { 14, 1, 6, 11, 7 };
        new TableSorter().TestMethod(tag);
        Console.Read();
    }
}
```

On commence par créer un délégué puis on le pointe sur la méthode SortAscending.

Ensuite on ajoute à ce délégué (avec l'opérateur +=) une nouvelle méthode, à savoir SortDecending. Désormais, le fait d'invoquer le délégué va invoquer les deux méthodes



```
file:///C:/Users/hp/Docume
SortAscending
1
6
7
11
14
SortAscending
14
11
7
6
1
```

# Les délégués (delegate)

## Délégué et méthodes anonymes

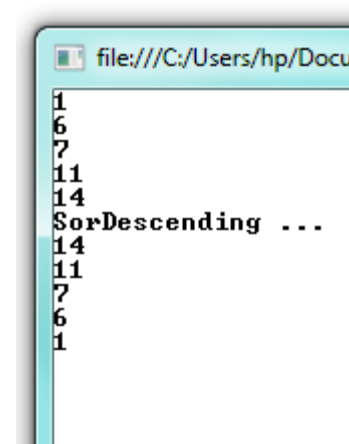
```
class TableSorter
{
    public delegate void DelegateSort(int[] tableau);
    public void TestMethod(int[] tab)
    {
        SortAndPrint(tab, delegate (int[] tableau)
        {
            Array.Sort(tab);
        });
        Console.WriteLine("SorDescending ...");
        SortAndPrint(tab, delegate (int[] tableau)
        {
            Array.Sort(tab);
            Array.Reverse(tab);
        });
    }

    public void SortAndPrint(int[] tab, DelegateSort pSortDelegate)
    {
        pSortDelegate(tab);
        foreach(int it in tab)
        {
            Console.WriteLine("{0} \t", it);
        }
    }

    public static void Main(string[] args)
    {
        int[] tag = new int[] { 14, 1, 6, 11, 7 };
        new TableSorter().TestMethod(tag);
        Console.Read();
    }
}
```

Définition d'une méthode anonyme (la méthode n'a pas de nom)

Évidemment, le délégué anonyme doit respecter la signature de DelegateSort



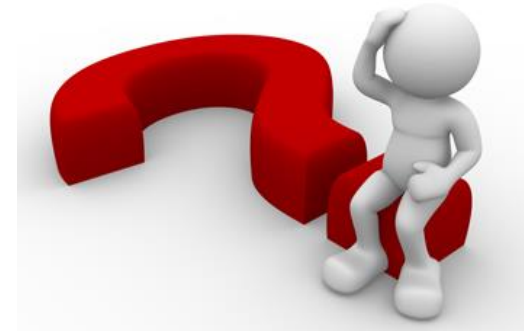
```
file:///C:/Users/hp/Docu
1
6
7
11
14
SorDescending ...
14
11
7
6
1
```



## Les délégués génériques Action et Func

**Question :** *Est-ce que à chaque fois on va avoir besoin d'utiliser un délégué, on doit créer un nouveau type en utilisant le mot-clé delegate ?*

**Réponse :** *non, C# dispose des délégués génériques Action et Func*



**Action<T,U,...>** ➡

Est un délégué générique qui permet de pointer vers une méthode qui ne renvoie rien

*(T,U,... sont les types de paramètres des méthodes à pointer par le délégué)*

**Func<T,U,...,V>** ➡

Lorsque la méthode dispose d'un retour, on peut utiliser le délégué générique **Func** sachant son dernier paramètre générique qui sera le type de retour du délégué

*(T,U,... sont les types de paramètres des méthodes à pointer et V leur type de retour)*

## Exemple avec le délégué générique Action

```
class ClassWithGenericDelegates
{
    //delegate generique pour les méthodes qui retourne void
    private Action<int, string> genericDelegateReturnVoid ;

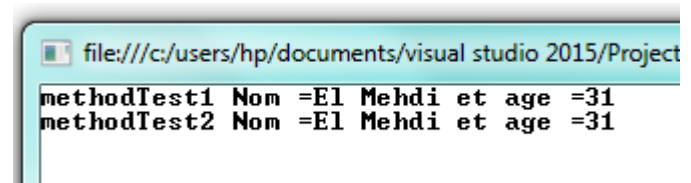
    public void methodTest1(int age, string nom)
    {
        Console.WriteLine("methodTest1 Nom ={0} et age ={1}", nom, age);
    }

    public void methodTest2(int age, string nom)
    {
        Console.WriteLine("methodTest2 Nom ={0} et age ={1}", nom, age);
    }

    public static void Main(string[] args)
    {
        ClassWithGenericDelegates c = new ClassWithGenericDelegates();
        c.genericDelegateReturnVoid = c.methodTest1;
        c.genericDelegateReturnVoid += c.methodTest2;

        c.genericDelegateReturnVoid(31, "El Mehdi");

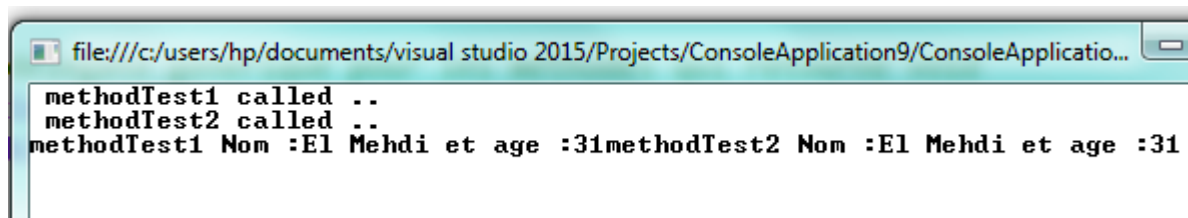
        Console.Read();
    }
}
```



```
file:///c:/users/hp/documents/visual studio 2015/Project
methodTest1 Nom =El Mehdi et age =31
methodTest2 Nom =El Mehdi et age =31
```

## Exemple avec le délégué générique Func

```
class ClassWithGenericDelegates
{
    //delegate generique pour les méthodes qui retourne string
    private Func<int, string, string> genericDelegateReturnString;
    public string methodTest1(int age, string nom)
    {
        Console.WriteLine(" methodTest1 called .. ");
        return "methodTest1 Nom :" + nom + " et age :" + age;
    }
    public string methodTest2(int age, string nom)
    {
        Console.WriteLine(" methodTest2 called .. ");
        return "methodTest2 Nom :" + nom + " et age :" + age;
    }
    public static void Main(string[] args)
    {
        ClassWithGenericDelegates c = new ClassWithGenericDelegates();
        c.genericDelegateReturnString = c.methodTest1;
        string res = c.genericDelegateReturnString(31, "El Mehdi");
        c.genericDelegateReturnString = c.methodTest2;
        res += c.genericDelegateReturnString(31, "El Mehdi");
        Console.WriteLine(res);
        Console.Read();
    }
}
```



```
file:///c:/users/hp/documents/visual studio 2015/Projects/ConsoleApplication9/ConsoleApplicatio...
methodTest1 called ..
methodTest2 called ..
methodTest1 Nom :El Mehdi et age :31methodTest2 Nom :El Mehdi et age :31
```

- Les expressions *Lambdas* permettent de simplifier l'écriture des délégués

```
DelegateSort tri = delegate (int [] tab)
```

```
{  
  Array .Sort (tab);  
};
```



Expression lambda équivalente

```
DelegateSort tri = (tab) =>
```

```
{  
  Array .Sort (tab);  
};
```



*L'expression lambda  
(tab) =>*

*se lit : «tab conduit à ».*



Lorsque la méthode ne contient qu'une seule instruction on peut omettre les parenthèses

```
DelegateSort tri = (tab) => Array .Sort  
(tab);
```

```
DelegateSort tri = tab => Array .Sort  
(tab);
```

S'il y a un seul paramètre à l'expression lambda, on peut omettre les parenthèses. Quand il y a plusieurs paramètres, on les sépare par une virgule

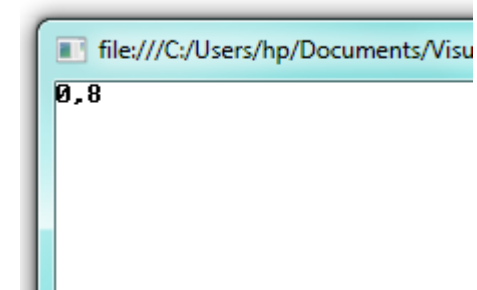
## Exemple

```
public static double Calcul(Func<int, int, double> methodeDeCalcul, int a, int b)
{
    return methodeDeCalcul(a, b);
}

public static void Main(string[] args)
{
    double division = Calcul((a, b) => (double)a / (double)b, 4, 5);

    Console.WriteLine(division);
    Console.ReadLine();
}
```

On appelle la méthode Calcul on lui passant en paramètre une expression lambda permettant de calculer la division



## Remarque :

$(a, b) \Rightarrow \{ \text{return } (\text{double})a / (\text{double})b; \}$



*On peut omettre return et les parenthèse lorsque la méthode ne contient qu'une seule instruction*

$a, b) \Rightarrow (\text{double})a / (\text{double})b$

## Les événements

- ❖ Les événements sont un mécanisme du C# permettant à une classe d'être notifiée d'un changement.
- ❖ Les événements sont déclarés à l'aide de délégués. Rappelez-vous qu'un objet délégué encapsule une méthode et qu'il peut donc être appelé de façon anonyme. Un événement est, pour une classe, le moyen d'autoriser les clients à fournir des délégués à ses méthodes qui doivent être appelées lorsque l'événement se produit. Lorsque l'événement se produit, les délégués qu'il a reçus de ses clients sont appelés.
- ❖ Un événement est défini grâce au mot-clé event.

## Exemple

```
class Vehicule
```

```
{
```

```
    public delegate void DelegatePositionChange(Object v);
```

```
    public event DelegatePositionChange changePositionEvent;
```

```
    public int x { get; set; }
```

```
    public int y { get; set; }
```

```
    public int z { get; set; }
```

```
    public Vehicule()
```

```
    {
```

```
        x = 0; y = 0; z = 0;
```

```
    }
```

```
    public void changePosition(int px, int py, int pz)
```

```
    {
```

```
        x = px; y = py; z = pz;
```

```
        if (changePositionEvent != null)
```

```
            changePositionEvent(this);
```

```
    }
```

```
    public override string ToString()
```

```
    {
```

```
        return "(" + x + "," + y + "," + z + ")";
```

```
    }
```

```
}
```



Déclaration d'un délégué qu'est la base de l'événement changePositionEvent



Lorsqu'il y a un changement des positions on notifie les abonnés, *this* est la référence sur l'état courant de l'objet (les valeurs de x, y et z)

## Exemple

```
class AlertSystem
{
    public AlertSystem(Vehicule pVehicule)
    {
        Vehicule.DelegatePositionChange
        delegeteChangePos = vehiculeBouge;
        pVehicule.changePositionEvent += delegeteChangePos;
        pVehicule.changePositionEvent += callThePolice;
    }
    private void vehiculeBouge(Object o)
    {
        Console.WriteLine("alert le véhicule change de"
            +"posistion, position actuelle = "+ o.ToString());
    }

    private void callThePolice(Object o)
    {
        Console.WriteLine("\a \a \a \a \a \a"+
            "Appel de la police en cours .. \n \n ");
    }
}
```



La classe AlertSystem indique à la classe Vehicule les délégués qui doivent être appelés lorsque l'événement « changement de position » se produit.



## Exemple

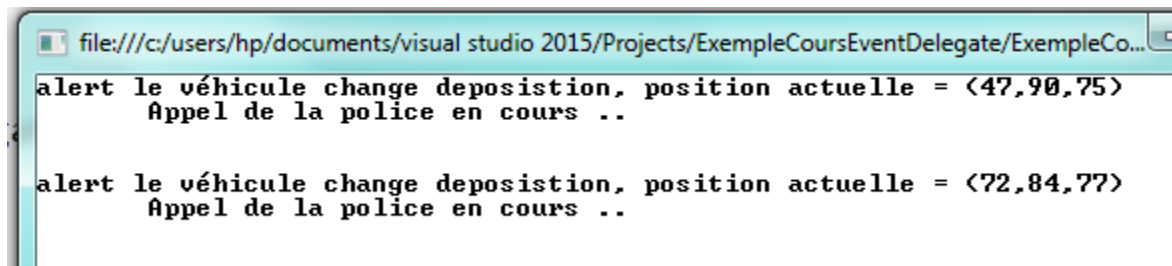
```
class Program
{
    public static void Main(string[] args)
    {
        Vehicule v = new Vehicule();
        AlertSystem alertSystem = new AlertSystem(v);

        Random random = new Random();
        do
        {
            v.changePosition(random.Next(100), random.Next(100), random.Next(100));

            System.Threading.Thread.Sleep(5000);

        } while (true);
    }
}
```

Chaque 5 secondes la position du véhicule change et les méthodes *vehiculeBouge* et *callThePolice* de la classe *Vehicule* s'exécute automatiquement.



```
file:///c:/users/hp/documents/visual studio 2015/Projects/ExempleCoursEventDelegate/ExempleCo...
alert le véhicule change de position, position actuelle = <47,90,75>
Appel de la police en cours ..
alert le véhicule change de position, position actuelle = <72,84,77>
Appel de la police en cours ..
```

*LINQ (Language INtegrated Query) est un composant du framework .NET qui ajoute des capacités d'interrogation sur des données aux langages .NET en utilisant une syntaxe proche de celle de SQL*

Il existe plusieurs domaines d'applications pour LINQ :

- Linq To Entities ou Linq To SQL qui utilisent ces extensions de langage sur les bases de données.
- Linq To XML qui utilise ces extensions de langage pour travailler avec les fichiers XML.
- Linq To Object qui permet de travailler avec des collections d'objets en mémoire.

*LINQ (Language INtegrated Query) est un composant du framework .NET qui ajoute des capacités d'interrogation sur des données aux langages .NET en utilisant une syntaxe proche de celle de SQL*

Il existe plusieurs domaines d'applications pour LINQ :

- **Linq To Entities** ou **Linq To SQL** qui utilisent ces extensions de langage sur les bases de données.
- **Linq To XML** qui utilise ces extensions de langage pour travailler avec les fichiers **XML**.
- **Linq To Object** qui permet de travailler avec des collections d'objets en mémoire.

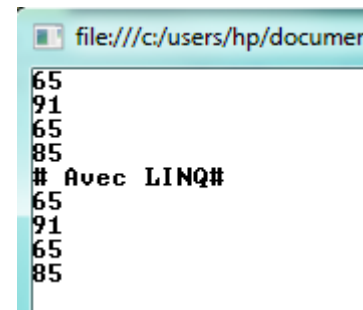
## Linq To Object.

Il s'agit d'extensions permettant de faire des requêtes sur les objets en mémoire et notamment sur tout ce qui implémente `IEnumerable<>` (comme les listes)

### Exemple 1

```
List<int> liste = new List<int> { 11, 65, 1, 91, 5, 65, 18, 85 };  
//Sans utiliser LINQ  
foreach (int i in liste)  
{  
    if (i > 60)  
    {  
        Console.WriteLine(i);  
    }  
}  
Console.WriteLine("# Avec LINQ#");  
//Avec LINQ  
IEnumerable<int> requeteFiltree = from i in liste  
                                where i > 60  
                                select i;  
  
foreach (int i in requeteFiltree)  
{  
    Console.WriteLine(i);  
}
```

Grâce à *Linq To Objet*, on a filtré la liste afin de ne parcourir que les entiers supérieurs à 60



```
file:///c:/users/hp/documer  
65  
91  
65  
85  
# Avec LINQ#  
65  
91  
65  
85
```

## Exemple 2

```
Console.WriteLine("# Avec LINQ 2 #");  
//Exemple LINQ 2  
IEnumerable<int> rqSortas = from i in liste  
                           where i > 60  
                           orderby i  
                           select i;  
  
foreach (int i in rqSortas)  
{  
    Console.WriteLine(i);  
}  
  
Console.WriteLine("# Avec LINQ 3 #");  
//Exemple LINQ 3  
IEnumerable<int> rqSortdes = from i in rqSortas  
                           orderby i descending  
                           select i;  
  
foreach (int i in rqSortdes)  
{  
    Console.WriteLine(i);  
}
```

*On sélectionne les entiers supérieurs à 60 et on ordonne le résultat*



*On réordonne les résultats précédents dans l'ordre décroissant*



```
file:///c:/users/hp/document  
# Avec LINQ 2 #  
65  
85  
91  
611  
# Avec LINQ 3 #  
611  
91  
85  
65
```

## Exemple 3

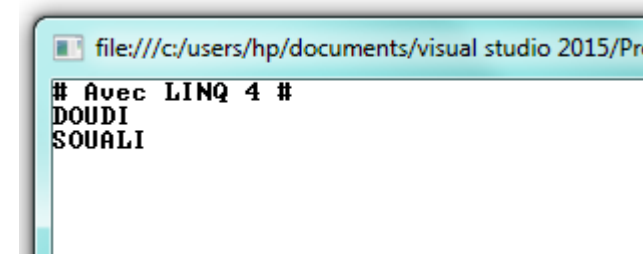
```
class Etudiant
{
    public int idEtudiant { get; set; }
    public int age { get; set; }
    public string nom { get; set; }
    public string prenom { get; set; }
}

Console.WriteLine("# Avec LINQ 4 #");
//Exemple LINQ 4
List<Etudiant> etdList = new List<Etudiant> {
    new Etudiant {idEtudiant=1,nom="BOUNDA", prenom="TARIK",age=18 },
    new Etudiant { idEtudiant = 1, nom = "DOUDI", prenom = "FADWA",age=22 },
    new Etudiant {idEtudiant=1,nom="SOUALI", prenom="FARID",age=31}
};

IEnumerable<string> requeteMajeur = from etd in etdList
                                   where etd.age > 18

                                   orderby etd.age , etd.nom
                                   select etd.nom;
foreach (string i in requeteMajeur)
{
    Console.WriteLine(i);
}

Console.ReadLine();
```

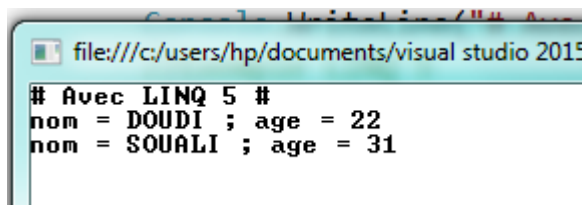


*On sélectionne les étudiants  
majeurs (>18 ans)*

## Exemple 4

```
Console.WriteLine("# Avec LINQ 5 #");  
//Exemple LINQ 5  
List<Etudiant> etdList2 = new List<Etudiant> {  
    new Etudiant {idEtudiant=1,nom="BOUNDA", prenom="TARIK",age=18 },  
    new Etudiant { idEtudiant = 1, nom = "DOUDI", prenom = "FADWA",age=22 },  
    new Etudiant {idEtudiant=1,nom="SOUALI", prenom="FARID",age=31}  
};  
  
var requeteMajeur2 = from etd in etdList2  
                     where etd.age > 18  
                     orderby etd.age, etd.nom  
                     select new { etd.nom, etd.age};  
  
foreach (var i in requeteMajeur2)  
{  
    Console.WriteLine("nom = {0} ; age = {1}",i.nom , i.age);  
}
```

*On peut retourner un objet anonyme juste avec deux propriétés*



```
file:///c:/users/hp/documents/visual studio 2015  
# Avec LINQ 5 #  
nom = DOUDI ; age = 22  
nom = SOUALI ; age = 31
```

# Annexe

- **Code source des exemples de cours : disponible sur e-services.**



# Référence

