

# PlateForme C#.Net

**Ing. Meryem OUARRACHI**

# Plan

- ☐ **L'environnement .Net**
- ☐ **Initiation à la programmation C#**
- ☐ **Programmation Orienté Objet C#**
- ☐ **Programmation avancée en C#**

## CHAPITRE 1:

# L'environnement .Net

# Environnement .Net

-.Net est une plateforme élaborée par l'entreprise *Microsoft* – (Juin 2002) afin de fournir un ensemble des technologies informatiques, pour le développement d'applications d'entreprises multi-niveaux.

# Environnement .Net

Les applications d'entreprises doivent être :

**-portables**

**-fiables et sécurisées**

**-maintenables et flexibles**

**- performantes**

**- Nécessité d'intégrer à un système d'information existant**

-Toutes ces considérations rendent ces applications complexes → Solution → .Net

-Microsoft .NET constitue ainsi la réponse de Microsoft à la plateforme J2ee de Sun.

# Historique .Net

- 1995:lancement de java
- 1996:Microsoft a acheté une licence java de Sun pour pouvoir utiliser les applets java en Internet Explorer
- 1997:Action en justice de Sun contre Microsoft. Sun estime que Microsoft n'a pas respecté la licence en développant une version de java avec les spécifications de Sun.
- 1998:Sun obtient une injonction qui empêche MS de vendre tout produit java incompatible avec ses spécifications.

# Historique .Net

- 1999: Sun annonce le lancement de J2ee
- 2000: Microsoft lance .Net, avec un nouveau langage de programmation semblable à java appelé C#.
- Sun et Microsoft se mettent d'accord que Microsoft ne va pas développer des produits incompatibles avec les spécifications de Sun.
- 2001: Microsoft annonce que sa machine virtuelle pour java ne sera pas compatible avec son nouveau système d'exploitation windows XP

# Historique .Net

-2002: MS a lancé .Net framework SDK

-2002: action en justice de Sun contre Microsoft. Environ 1 milliard de dollars est demandé, l'inclusion de Java en Windows XP

-La plateforme .Net est lancée officiellement après que SUN remporte le procès de l'obligeant de MS d'arrêter le développement de sa propre machine virtuelle de Java

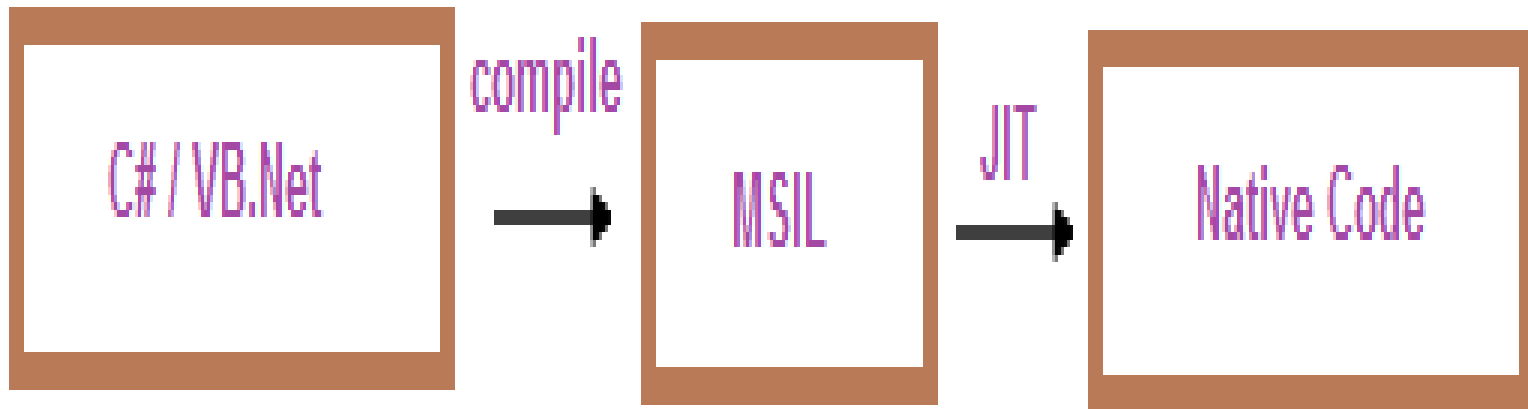


# .Net Framework

- Framework= cadre de développement
- Le Framework .Net a 3 roles
  - Build**: Vérifier le code (compilation)+exécution via le CLR
  - Manage**: outils facilitant la programmation
  - Deploy**: outil de déploiement de l'application pour la mise en production

# Common Langage Runtime (CLR)

-C'est le nom choisi par Microsoft pour le composant de machine virtuelle du framework .NET



# Class Library

- Bibliothèques de classe est une bibliothèque de types des classes, interfaces, qui permettent d'accéder aux fonctionnalités du système. Elle ont une extension .dll
- Elles représentent le fondement sur lequel les applications .NET Framework, les composants et les contrôles sont construits.

# Framework de développement

.Net framework offre aux développeurs un ensemble des framework qui leur facilitent la création de plusieurs types de projets:

- WindowsForm

- WPF

- ASP.net

- WCF

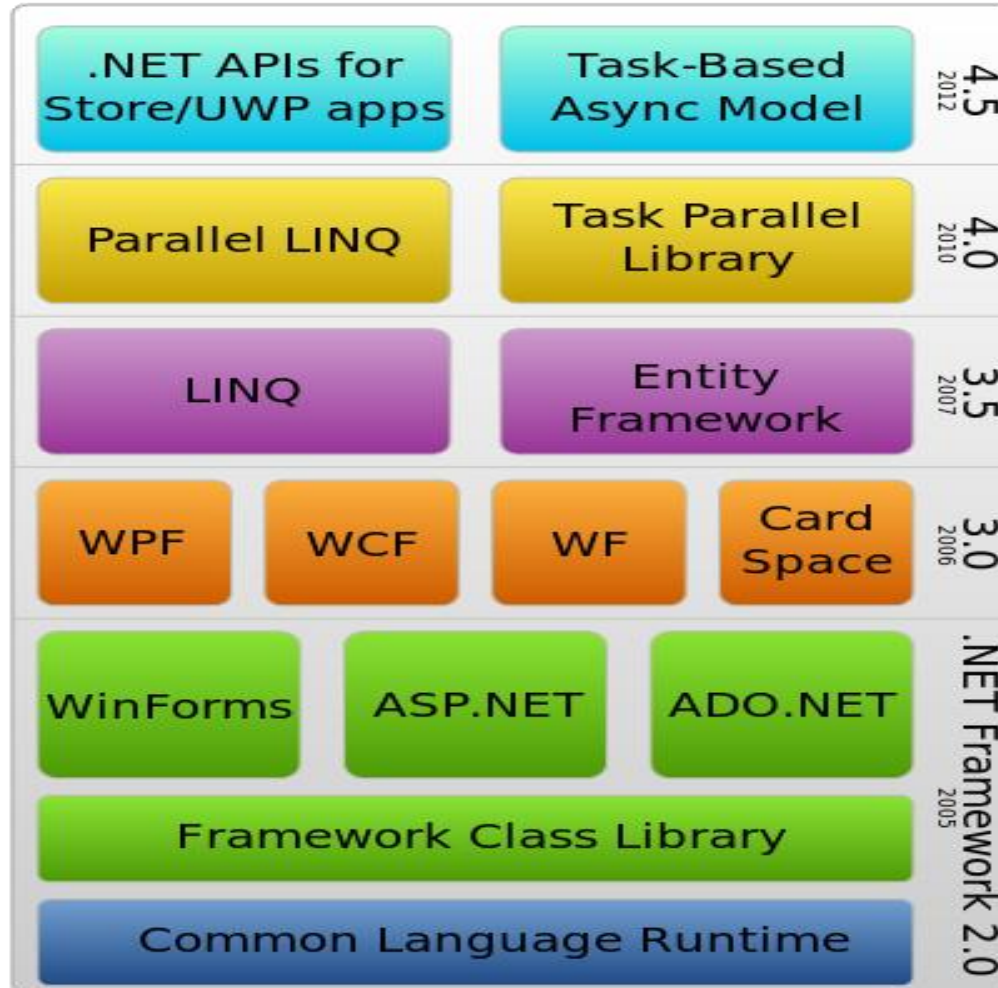
- etc

# Les versions de .Net framework/Net core

	.Net Framework	.Net Core
<b>Portabilité</b>	Il est principalement conçu pour Windows	multiplateforme
<b>Licence et Open Source</b>	propriétaire	Open source
<b>Taille de l'Application</b>	volumineuse	légère

# Les versions de .Net framework/Net core

## ❑ .Net Framework:



# Les versions de .Net framework/.Net Core

## ❑ .Net Core:

**.NET Core 1.0 à 3.1** : Une version modulaire, légère et multiplateforme de .NET.

**.NET 5.0** : Unification de .NET Core avec .NET Framework, marquant le début de la transition vers une seule plateforme .NET unifiée.

• **.NET 6** : Elle vise à fournir une expérience de développement unifiée pour divers scénarios, notamment les applications web, les applications mobiles.

• **.NET 7 et au-delà** : Ces versions futures peuvent apporter de nouvelles fonctionnalités, améliorations de performances et support pour davantage de scénarios.

# Les langages du .Net

Grâce au CLR, la plate-forme .NET est indépendante de tout langage de programmation et supporte nativement un grand nombre de langages de programmation, parmi lesquels :

- Visual basic

- C#

- Pascal

- .-Etc



# Environnement de travail en .Net

1.Installer Microsoft .Net

2.Avoir un environnement de développement des applications .Net:Visual studio

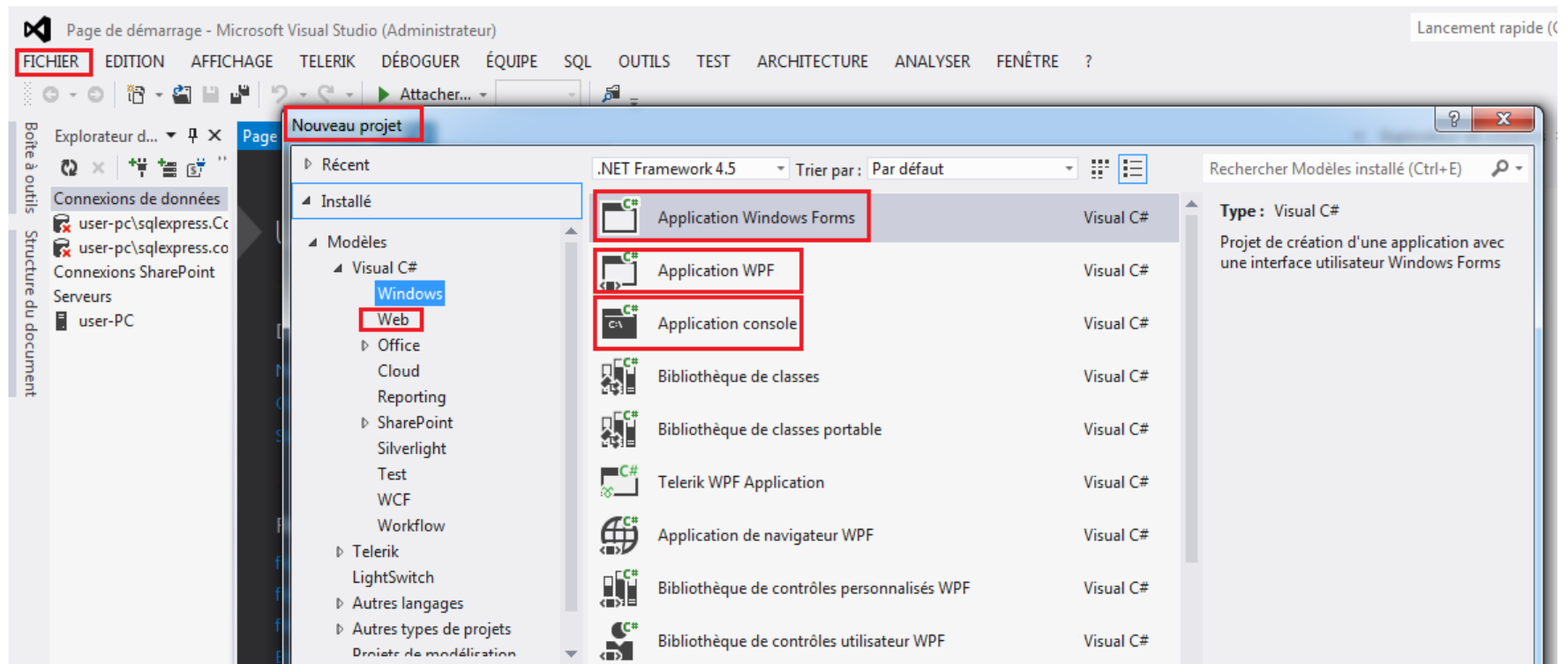
-Version gratuite:VS Community

-Version payante: entreprise,professional ...

*Remarque:* si vous travaillez en linux utilisez visual studio code

# Visual Studio

-Ouvrir Visual studio → Fichier → nouveau projet



# Visual Studio

**Remarque:** Lors de première démarrage de Visual Studio, il me demande le langage de programmation à utiliser (C#, VB...), par la suite, ça sera le langage par défaut pour les autres démarrages de Visual Studio. Si on veut modifier ce langage:

Outils → Importation et exportation des paramètres → Réinitialiser tous les paramètres > choisir le langage



Choisir une collection de paramètres par défaut

Quelle collection de paramètres voulez-vous rétablir ?

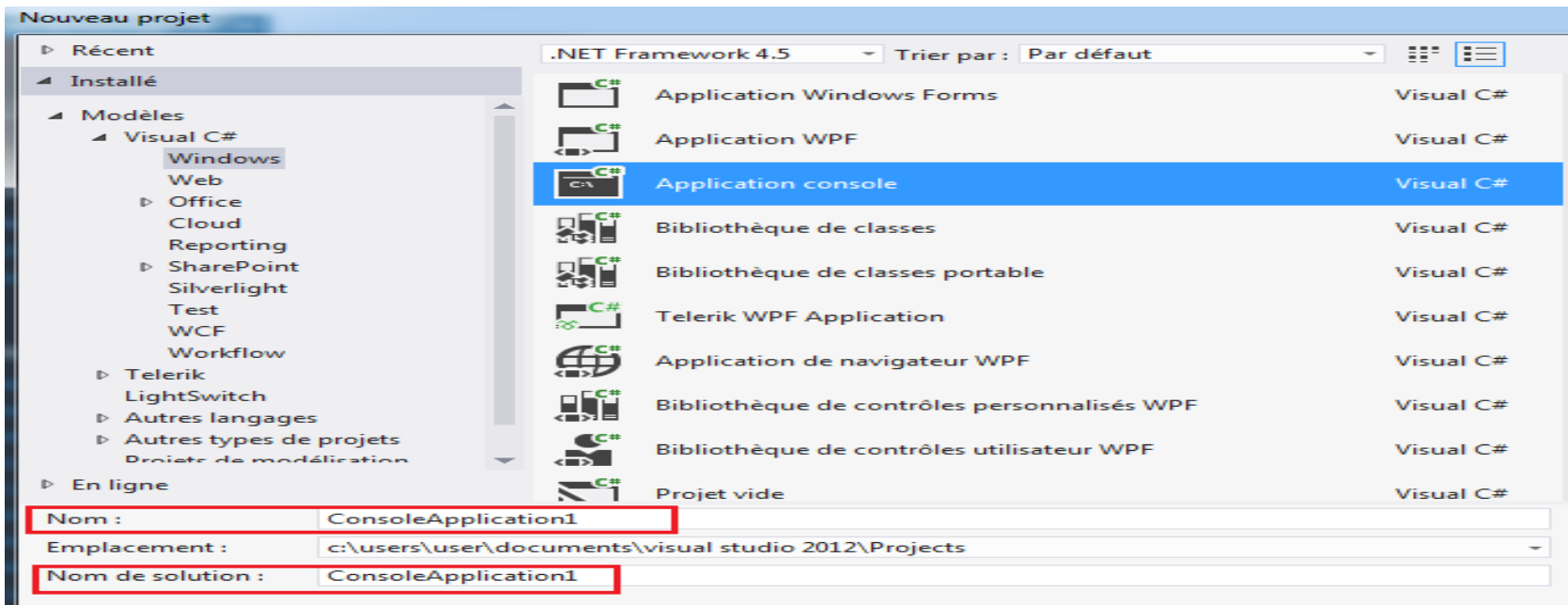
- ⚙ Développement Web
- ⚙ Développement Web (code uniquement)
- ⚙ JavaScript pour applications Windows 8
- ⚙ Paramètres de développement généraux
- ⚙ Paramètres de développement LightSwitch
- ⚙ Paramètres de développement SQL Server
- ⚙ Paramètres de développement Visual Basic
- ⚙ Paramètres de développement Visual C#**
- ⚙ Paramètres de développement Visual C++
- ⚙ Paramètres de développement Visual F#

Description :

Personnalise l'environnement pour optimiser l'espace à l'écran de l'éditeur de code et rendre les commandes C# plus visibles. Améliore la productivité grâce à des raccourcis clavier conçus pour être faciles à apprendre et à utiliser.

# Visual Studio

- Pour organiser les classes, Java utilise le concept de packages et C# utilise le concept de Namespaces
- Nom de solution:conteneur d'un ensemble de projet
- Nom:le nom de projet



# Comparaison entre J2ee et .Net



**J2EE**



Microsoft®  
**.NET**

# Comparaison entre J2ee et .Net

- Langage de programmation

- J2ee utilise un seul langage Java

- .Net multilangage

- Portabilité

- Les deux sont portables maintenant

- Documentation

- Mieux en J2ee par rapport à .Net

# Comparaison entre J2ee et .Net

- Application de bureau

J2EE	.Net
Swing Awt JavaFX	Windows Forms WPF

# Comparaison entre J2ee et .Net

- Application Serveur

	J2EE	.Net
<b>Serveur Web , serveur d'Application</b>	Apache tomcat,Jboss,GlassFish	IIS Nginx
<b>Application Web</b>	JSP / Servlet JSF /Struts Spring MVC	ASP WebForms ASP MVC ASP.net core
<b>Gestion de base de données</b>	JDBC	ADO.net



# Comparaison entre J2ee et .Net

	J2EE	.Net
<b>Framework ORM</b>	Hibernate JPA	Nhibernate Linq Entity Framework
<b>Framework pour Injection de dépendance(Inversion de controle)</b>	Spring	Microsoft dependancy Injection Spring.Net

# Comparaison entre J2ee et .Net

	J2EE	.Net
<b>Programmation distribué</b>	RMI Web Services	.Net Remoting WCF
<b>BI</b>	Talend	Power BI
<b>Application mobile</b>	Android	-Xamarin -MAUI (Multi-platform App UI)

# Comparaison entre J2ee et .Net

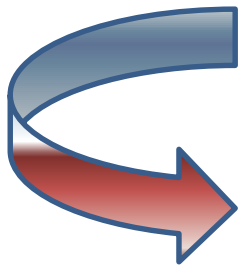
## ❑ Le marché du travail:

- Le langage de programmation Java est l'un des plus célèbres, langage qui sont toujours demandé sur le marché de l'emploi
- Ces dernières années,nombreuses entreprises de toutes tailles et de toutes industries utilisent .NET, car il offre des fonctionnalités robustes pour le développement.
- C# occupe la 4ème place dans l'indice TIOBE en Mai 2021. Il fait partie des langages de programmation les mieux payés en 2023.

# Comparaison entre J2ee et .Net

- Récapitulatif

**Tout ce qu'on peut faire en J2ee, on peut le faire en .Net et vice versa mais la façon qui se change.**



**En tant que ingénieur en développement informatique, il faudra avoir une base solide dans les deux environnements**

## CHAPITRE 2:

# **Initiation à la programmation C#**

# Introduction

- Le langage C# est le langage le plus utilisé par les professionnels dans les projets .Net
- Il peut être considéré comme une synthèse entre le langage C++ et java. Il a pris leurs principales caractéristiques en ajoutant d'autres spécifications pour rendre la programmation OO plus performantes

# Structure Typique D'un FichierSource

Program.cs

```
using System; //l'importation des namespaces nécessaires.  
namespace Com.Ensas //déclaration d'un namespace  
{  
    class Personne //déclaration d'une classe  
    {  
        //... Corps de la classe  
    }  
}
```

# Structure Typique D'un FichierSource

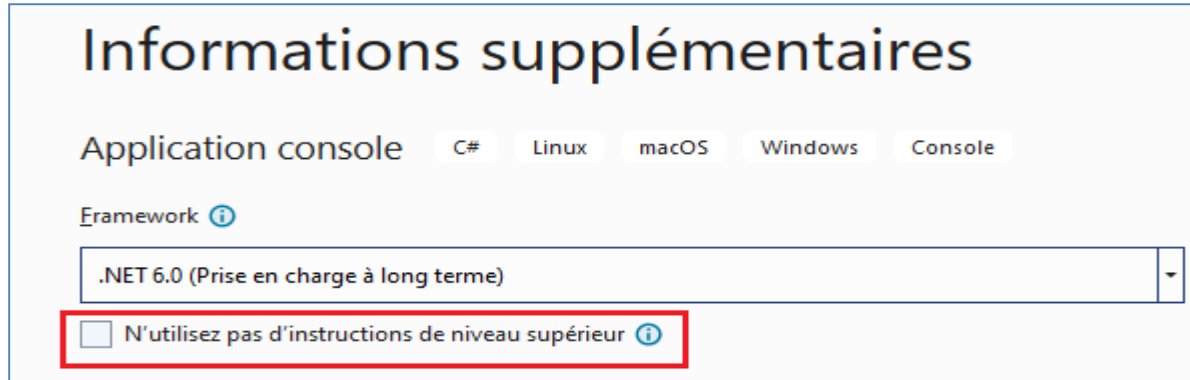
```
using System;

namespace MyApp // Note: actual namespace depends on the project name.
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```



# Structure Typique D'un FichierSource

Lors de création de projet dans une version qui dépasse la version .Net 5



Informations supplémentaires

Application console C# Linux macOS Windows Console

Framework ⓘ

.NET 6.0 (Prise en charge à long terme)

☒ N'utilisez pas d'instructions de niveau supérieur ⓘ

Voilà le résultat si on utilise le niveau supérieur

```
// See https://aka.ms/new-console-template for more information
Console.WriteLine("Hello, World!");
```

On n'aura pas besoin d'inclure les autres éléments de programme, le compilateur les génère pour nous.

# Structure Typique D'un FichierSource

- En C# ,on peut avoir plusieurs classes publiques dans le même fichier.
- Les importations peuvent être mises à l'intérieur de la déclaration du namespace, dans ce cas les namespaces importés sont reconnus seulement dans le namespace où ils sont déclarés.
- Utilisation du UpperCamelCase pour les noms des classes, interfaces, namespaces et pour les méthodes

# Structure Typique D'un FichierSource

Program.cs

```
namespace Com.Ensas.Ns1
{
    using System;

    class MainClass
    {public static void Main()
        {
            Console.WriteLine
                ("Bonjour");
        }
    }
}
```

```
namespace Com.Ensas.Ns2
{
    class Personne
    {
        public void SayHello(String
            name)
        {
            //ligne suivante : Compilation
            Error,
            //classe Console est non reconnue
            Console.WriteLine
                ("Bonjour" +
                name);
        }
    }
}
```

# Alias et Noms Pleinement Qualifiés

Program.cs

```
namespace Com.Ensas.Ns1
{
    using pNS = Com.Ensas.Ns2; //Alias d'un namespace
    using pClass = Com.Ensas.Ns2.Personne; //Alias d'une
    classe
    class MainClass
    {
        public static void Main()
        {
            new Com.Ensas.Ns2.Personne().SayHello(); //nom
            pleinement qualifié

            new pNS.Personne().SayHello(); //alias du namespace

            new pClass().SayHello(); //alias de la classe elle-
            même
        } } }
```

# Déclaration des Variables

- La déclaration des variables en C# se fait avec la syntaxe

`<type> <nom_var> = <valeur>;`

*Exemple* : `int x=5;`

- En C#, on possède également le mot clé `var` pour déclarer une variable avec un type implicite, le type de la variable est donc déterminé depuis la valeur affectée :

*Exemple* : `var y=6.10`

# Types de Données

- Tous les types primitifs en C# sont des **objets** dans le namespace System.
- Pour chacun de ces types on a un nom plus court ou un alias afin de simplifier l'utilisation.

*Exemple:* `int` pour représenter le type `System.Int32`

- Vu que ces types sont des objets, ils possèdent des méthodes qu'on peut appeler;

```
int x = 4;  
Console.WriteLine(x.CompareTo(12));
```

# Types de Données

- Le type entier:

Short Name	.NET Class	Type	Number of Bits
byte	<a href="#">Byte</a>	Unsigned integer	8
sbyte	<a href="#">SByte</a>	Signed integer	8
int	<a href="#">Int32</a>	Signed integer	32
uint	<a href="#">UInt32</a>	Unsigned integer	32
short	<a href="#">Int16</a>	Signed integer	16
ushort	<a href="#">UInt16</a>	Unsigned integer	16
long	<a href="#">Int64</a>	Signed integer	64
ulong	<a href="#">UInt64</a>	Unsigned integer	64

# Types de Données

- Le type réel:

Short name	.Net class	Number of bits	<i>Exemple</i> : float x=22.18f; double y=9.5;
<b>float</b>	<b>Float</b>	32 bits	
<b>double</b>	<b>Double</b>	64 bits	

- Type caractère

Short name	.Net class	Number of bits
<b>char</b>	<b>Char</b>	<b>16bit</b>

- Type boolean

Short name	.Net class	Number of bits
<b>bool</b>	<b>Boolean</b>	<b>1 bit</b>



# Types de Données

- Le type String:

Short name	.Net class
<b>string</b>	<b>String</b>

- Type object

Short name	.Net class
<b>object</b>	<b>Object</b>

# Les Types Nullable

- Les types qu'on vient de voir ne peuvent pas prendre **null** comme valeur. C# par contre offre une version nullable de ces types.
- Par exemple pour le type **int** (System.Int32), on possède le type nullable équivalent suivant :

Short name	.Net class
<b>int ?</b>	<b>Nullable&lt;Int32&gt;</b>

# Les Types Nullables

- Pour accéder à la valeur de ces types nullable, C# offre les moyens suivants :

- L'utilisation d'un cast explicite.
- La propriété Value.

```
double? x=7;  
double y=(double) x; //cast explicite  
double z= x.Value;   //propriété Value  
double v=x;          //erreur
```

# Value Types Vs. Reference Types

Type Valeur	Type Référence
<ul style="list-style-type: none"><li>• Un type de données est dit « Type valeur » s'il possède la donnée dans son propre espace de mémoire.</li></ul> <p>-----</p> <ul style="list-style-type: none"><li>• Les types «Type valeur » sont :<ul style="list-style-type: none"><li>-Tous les types numériques.</li><li>-Boolean, Char et Date</li><li>-Les énumérations.</li></ul></li></ul>	<ul style="list-style-type: none"><li>• Un type de données est dit « Type référence » s'il possède une référence vers un autre emplacement de la mémoire où la donnée est stockée.</li></ul> <p>-----</p> <ul style="list-style-type: none"><li>• Les types «Type référence» sont :<ul style="list-style-type: none"><li>-String. (mais se comporte comme un Value Type!!)</li><li>-Tous les tableaux.</li><li>-Les classes.</li><li>-Les Delegates.</li></ul></li></ul>

# L'affichage en C#

1. `Console.WriteLine("Expression à afficher");`

2. `Console.Write("Expression à afficher");`

3. `int x=6;`

`Console.Write("Expression à afficher"+x);`

4. Format string

`int x=6;`

`int y=9;`

`Console.Write("{0} et {1}",x,y);` → 6 et 9

# La saisie en C#

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

-le résultat de la méthode Console.ReadLine( ) est toujours une chaîne de caractère.

-Pour pouvoir récupérer la valeur saisie dans une variable de type approprié, il faut effectuer une conversion par le composant (la classe) Convert et la méthode correspondante au type voulu.

Convert.ToInt32(v);

Convert.ToDouble(v);

Convert.ToChar(v);

etc

# La saisie en C#

## Exemple

```
string v;  
int nb;  
Console.WriteLine("tapez un nombre entier ");  
v = Console.ReadLine(); //Saisie dans une variable de type string  
nb = Convert.ToInt32(v);
```

# Les boucles

- Boucle while

```
while (/* Condition */)  
{ //Instructions à répéter  
}
```

- Boucle do ... While

**Do**

```
{ //Instructions à répéter  
}while(condition);
```

- La boucle for

```
for (Compteur =Initial ; Final ; ValeurDuPas )  
{ //Instructions à répéter  
}
```



# Les boucles

- Boucle foreach

```
foreach (int i in t1)  
{ Console.WriteLine(i); }
```

# Les tableaux

- On peut déclarer un tableau en suivant l'une de ces méthodes.

```
int [ ] t1 = {1, 2, 3} ;
```

```
int [ ] t2= new int [3];
```

```
int [ ] t3= new int [ ] { 1, 2, 3};
```

```
int [ ] t4= new int [3] { 1,2,3} ;
```

- Pour savoir la taille d'un tableau on utilise la propriété Length.

# Le passage de paramètres

- Lors de l'appel d'une méthode prenant en paramètre des **types valeur**, les paramètres sont passés par valeur
- Lors de l'appel d'une méthode prenant en paramètre des **types références**, les paramètres sont passés par référence
- Pour contrôler ce comportement par défaut, C# propose des **modificateurs des paramètres**.

# Modificateurs des Paramètres

- Le modificateur out :

Le modificateur **out** *exige* que la méthode modifie la valeur de la variable concernée, sinon on aura une erreur de compilation.

```
public static void somme(int a, int b, out int c)
{ c = a + b; }

public static void somme(int a, int b, out int c)
{a = a + b; //erreur de compilation, on doit affecter
  //une valeur à c dans le corps de cette méthode
}

static void Main(string[] args)
{   int y;
    somme(2, 3, out y); //Appel de méthode
}
```

# Modificateurs des Paramètres

- **Le modificateur ref:**

Le modificateur **ref** permet la modification de la variable dans le corps de la méthode, mais ne l'exige pas

```
public static void somme(int a, int b, ref int c)
{ c = a + b; }

public static void somme(int a, int b, ref int c)
{a = a + b; // pas d'erreur ici
}

static void Main(string[] args)
{   int y;
    somme(2, 3, ref y); //Appel de méthode
}
```

# Paramètres Nommés

- On peut appeler une méthode sans respecter l'ordre de ses paramètres, ceci en précisant le nom du paramètre avant la valeur qui va prendre.

```
public static void Afficher(int a, int b)
{ Console.WriteLine("Valeur1="+a+"Valeur2="+b); }
```

```
static void Main(string[] args)
{ Afficher(b: 5, a: 7); }
```

# Nombre Variable de Paramètres

- C# offre le mot clé `params` pour avoir la possibilité de passer un nombre variable de paramètres à une méthode

```
//paramètres a et b sont obligatoires, optParams sont optionnels.  
public static void Somme(int a, int b, params int[] optParams)  
{  
    var som = a + b;  
    foreach (var x in optParams)  
    { som += x; }  
    Console.WriteLine("La somme des valeurs="+som);  
}  
static void Main(string[] args)  
{  
    Somme(5);    // Erreur de Compilation ! Besoin d'au moins 2 paramètres  
    Somme(1,8); //OK  
    Somme(1,6,3); // OK  
}
```

# Les Enumérations

-Les énumérations sont un type de données qui se compose d'un ensemble de constantes, appelé liste d'énumérateurs, il est sous la forme des noms/valeurs.

```
static void Main(string[] args)
{
    const string todaysDay = "samdi";
    if (IsWeekendDay(todaysDay))
        Console.WriteLine("Youpi c'est le week-end !");
    else
        Console.WriteLine("Métro boulot dodo...");
}
```

```
private static bool IsWeekendDay(string day)
{
    if (day.ToLower() == "samedi" || day.ToLower() == "dimanche")
        return true;

    return false;
}
```



# Les Enumérations

## -Avantages:

Les énumérations permettent d'organiser des valeurs arbitraires afin d'avoir plus de contrôle sur celles-ci et d'avoir également une meilleure lisibilité du code.

## -Création

```
public enum JoursDeSemaine
{
    Lundi,          //valeur par défaut=0
    Mardi,          //valeur par défaut=1
    Mercredi,       //valeur par défaut=2
    Jeudi,          //valeur par défaut=3
    Vendredi,       //valeur par défaut=4
    Samedi,         //valeur par défaut=5
    Dimanche        //valeur par défaut=6
}
```

# Les Enumérations

-Appel

```
static void VerifierSemaine(JoursDeSemaine j)
{
    if (j == JoursDeSemaine.Samedi || j == JoursDeSemaine.Dimanche)
        Console.WriteLine("C'est la fin de semaine");
    else
        Console.WriteLine("Ce n'est pas la fin de semaine");
}
static void Main(string[] args)
{
    VerifierSemaine(JoursDeSemaine.Jeudi);
}
```

## CHAPITRE 3:

# **Programmation O.O en C#**

# Les Classes

- Création et instantiation d'une classe

```
class Personne
{
    private string nom;
    public void Bienvenue()
    {
        Console.WriteLine("Bonjour"+nom);
    }
    static void Main(string[] args) {
        Personne p = new Personne(); //Création d'une instance
        p.Bienvenue();
    }
}
```

# Les Classes Partielles

-C# offre la possibilité de partitionner une classe sur plusieurs fichiers sources. Chaque fichier source contient une section de la définition de type ou de méthode, et toutes les parties sont combinées lorsque l'application est compilée

```
class Personne
{
    private string nom;
    private string prenom;
    public void Bienvenue()
    {
        Console.WriteLine("Bonjour"+nom);
    }
}
```



```
partial class Personne
{
    private string nom;
    private string prenom;
}
```



```
partial class Personne
{
    public void Bienvenue()
    {
        Console.WriteLine("Bonjour"+nom);
    }
}
```

# Constructeurs

- C'est une ou plusieurs méthode(s) permettant d'initialiser les objets.
- Le constructeur est appelé lors de la création de l'objet
- Le constructeur a le même nom que la classe
- Il n'a pas de valeur de retour
- Le constructeur peut être surchargé

# Constructeurs

- Un constructeur qui ne prend aucun paramètre est dit un constructeur par défaut
- Si on déclare un constructeur dans une classe, le constructeur par défaut implicite n'existera plus et on devra le déclarer explicitement si on souhaite instancier la classe sans paramètres.

# Constructeurs

```
class Personne {  
    string nom;  
    int age;  
    public Personne() { nom = "ahmed"; }  
    public Personne(string n,int a)  
    { nom = n;  
      age=a;}  
}
```



# Le mot clé « this »

- « this » = référence sur l'objet courant
- Représente l'instance courante en train de s'exécuter
- Il peut être utilisé pour distinguer entre les champs de la classe et les paramètres qui portent le même nom.

```
class Personne {  
    string nom;  
    public Personne(string nom)  
    { this.nom= nom;  
    }  
}
```

# Le mot clé « this »

- Il peut être utilisé aussi pour appeler un constructeur à partir d'un autre :

```
class Personne {  
    string nom;  
    //appel du 2ème constructeur à partir du constructeur par  
    //défaut.  
    public Personne() : this("sara")  
    { }  
  
    public Personne(string nom )  
    { this.nom = nom ;  
    }  
}
```

```
Personne p = new Personne();
```

# Attribut statique

- La valeur d'un attribut déclaré comme « static » est partagée par toutes les instances (objets) de la classe
- Elles sont utilisables sans avoir besoin d'instancier la classe.

```
class Personne {  
    public static int age;  
    static void Main(string[] args){  
        Personne p = new Personne();  
        p.age = 18; //Erreur de compilation  
        Personne.age = 20; //OK  
    }  
}
```

# Méthode statique

- Une méthode statique est une méthode qui n'a accès qu'aux membres static de la classe.
- L'appel d'une méthode statique ne se fait pas sur un objet, mais sur une classe

```
class Personne
{
    public static void Bienvenue()
    { Console.WriteLine("salut");}

    static void Main(string[] args)
    {
        Personne p = new Personne();
        p.Bienvenue(); // Erreur de compilation!
        Personne.Bienvenue(); //ok
    }
}
```

# Constructeur statique

- Problème:** Pour initialiser un attribut statique, on peut le faire lors de la déclaration si la valeur à affecter est connue

```
class Personne {  
    public static int age = 20;  
}
```

- Mais si la valeur doit être lue à partir d'un fichier ou d'une base de données on est devant un problème : on ne peut pas écrire le traitement de la lecture dans la déclaration car on aura une erreur de compilation.

# Constructeur statique

•**Solution:** Utilisation de constructeur statique, il permet d'initialiser n'importe quelle attribut statique. Il s'exécute dans deux cas :

- lorsqu'une propriété/méthode statique est appelée
- lorsqu'une instance de notre type est créée

# Constructeur statique

```
class Personne {  
    static int age;  
    static Personne () {  
        //On peut faire des traitements plus complexes pour  
        // initialiser la variable static age  
        age=20;  
        Console.WriteLine("Je suis dans le constructeur.");  
    }  
  
    public static void Bienvenue() {  
        Console.WriteLine("Salut!");  
    }  
  
    static void Main(string[] args)  
    { Personne.Bienvenue();  
    }  
}
```

- Résultat:

Je suis dans le constructeur.  
Salut!

# Constructeur statique

- On ne peut déclarer qu'un seul constructeur statique dans une classe.
- Un constructeur statique ne prend aucun paramètre.
- Un constructeur statique n'accepte aucun modificateur d'accès (public, private etc.)
- Le constructeur statique s'exécute avant tous les constructeurs d'instances.



# Classe statique

- C'est une classe qui contient que des membres statiques.
- Elle ne peut pas être instanciées.

```
static class Personne
{
    static int age;
    string nom; // erreur cette classe doit contenir juste des éléments statiques
    static Personne() { }
    static void Bienvenue() { }
}
```

# Constante

- On utilise le mot **const** pour déclarer une constante çàd un attribut dont la valeur ne peut pas être modifiée.

```
class Personne
{
    const int age = 22;
}
```

- La constante doit être initialiser obligatoirement lors de la déclaration
- La constante est implicitement statique
- Une constante ne peut être initialisé que par des values types

# readonly

- Le mot clé **readonly** indique qu'un champ d'une classe est en mode lecture seule après la première affectation d'une valeur.

```
class Personne {  
    readonly int age; //peut être initialisé ici également.  
    public Personne()  
    {age = 22;  
    }  
}
```

- champ readonly doit être initialisé soit lors de la déclaration soit dans un constructeur.
- Champ readonly n'est pas statique. (peut être déclaré static explicitement)
- Un champs readonly peut être initialisé par un Reference Type.

# Encapsulation

Modificateur	Description
public	Accessible partout
private	Accessible seulement dans la classe où ils sont définis.
protected	Accessible dans la classe où ils sont définis et aussi dans les classes filles.
internal	Accessible partout dans l'assembly où ils sont définis.
protected internal	Accessible partout dans l'assembly où ils sont définis, dans la classe mère et dans les classes filles.

-Par défaut accessibilité est private

# Les propriétés

-En C# on peut utiliser les propriétés au lieu d'utiliser des méthodes comme getters et setters, afin d'accéder aux champs privées.

```
class Personne
{
    private int age;
    public int Age
    {
        get
        { return age; } //getter de la propriété
        set           //setter de la propriété
        {
            if (age < 0)
                Console.WriteLine("Erreur : age non valide!");
            else
                age = value;
        }
    }
}

static void Main(string[] args)
{
    Personne p = new Personne();
    p.Age=18; // au lieu de p.setAge(18);
    Console.WriteLine("age=" + p.Age); //au lieu de p.getAge();
}
```

# Les propriétés

- **get=** Lecture seule     /     **Set=** Ecriture seule

```
class Personne {  
    private int age;  
    public int Age {//pas de 'get' → écriture seule  
        set { age = value; }  
    }  
  
    private string nom;  
    public int Nom {//pas de 'set' → lecture seule  
        get { return nom; }  
    }  
}
```

# Les propriétés statiques

- On peut ajouter le mot clé **static** pour déclarer une propriété statique

```
class Personne {  
    private static int age;  
    public static int Age {  
        get { return age; }  
        set { age = value; }  
    }  
}
```

# Les propriétés automatiques

-S'il n'y a pas de traitement spécifique à effectuer lors de l'écriture ou la lecture (validation etc.), on peut utiliser les propriétés automatiques:

```
class Personne {  
    private int age;  
  
    //propriété automatique  
  
    public int Age { get; set; } }
```

-**Remarque:** on ne peut pas déclarer des propriétés automatiques en écriture seules!

```
class Personne {  
    public int Name { get; } // OK  
    public int Age { set; } //Erreur! }
```