

C# : Structure de base

```
using <notre espace de noms>
namespace <espace de noms facultatif>
{
    class Program
    {
        static void Main(string[] args)
        {
            //Code source principal.
        }
    }
}
```

C# : Structure de base

- Le mot clé **using**: est une directive qui indique quel espace de noms utiliser
- Le mot clé **namespace** défini lui un espace de nom facultatif.
- ensuite c'est la classe **Program** qui est le cœur du programme.
- **static void Main** indique la méthode principale (code qui sera exécuté en premier).

C# : Structure de base

Exemple 1 :

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

- WriteLine : affiche un message
- ReadLine : récupère une entrée au clavier (sert de pause)

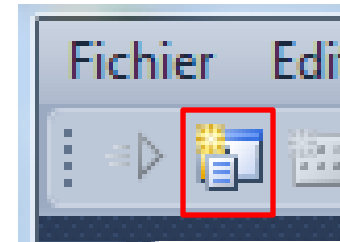
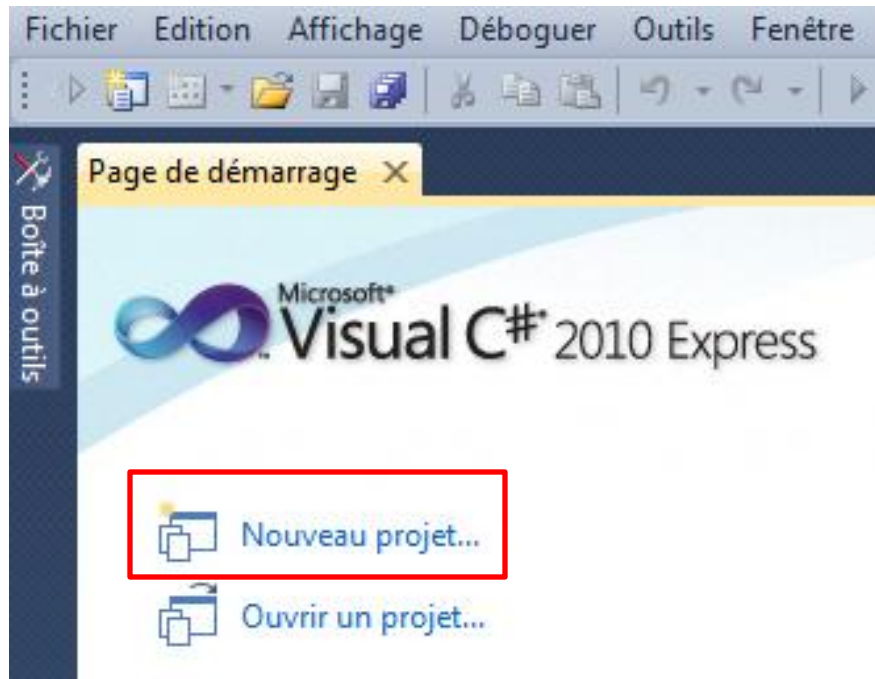
C# : Structure de base

Exemple 2 :

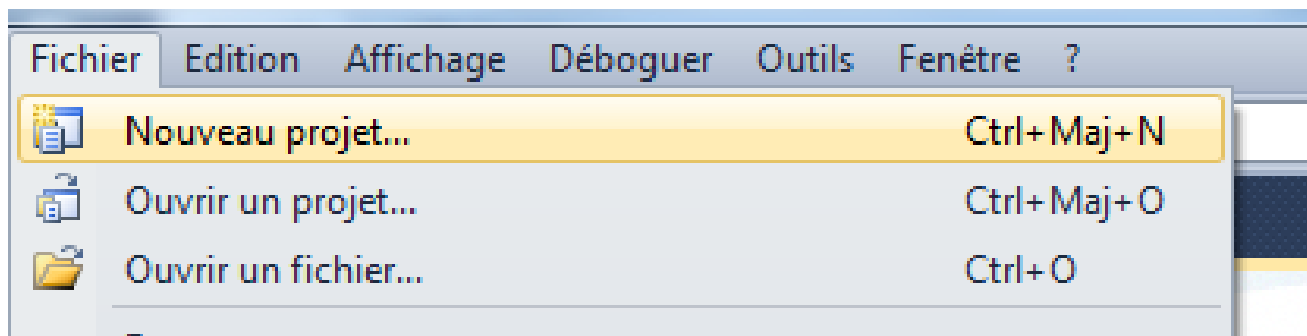
```
using System;
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello!");
        Console.ReadKey();
    }
}
```

- `using System` **permet d'écrire** `Console.WriteLine` **au lieu de** `System.Console.WriteLine`
- `ReadKey`: **autre solution de pause**

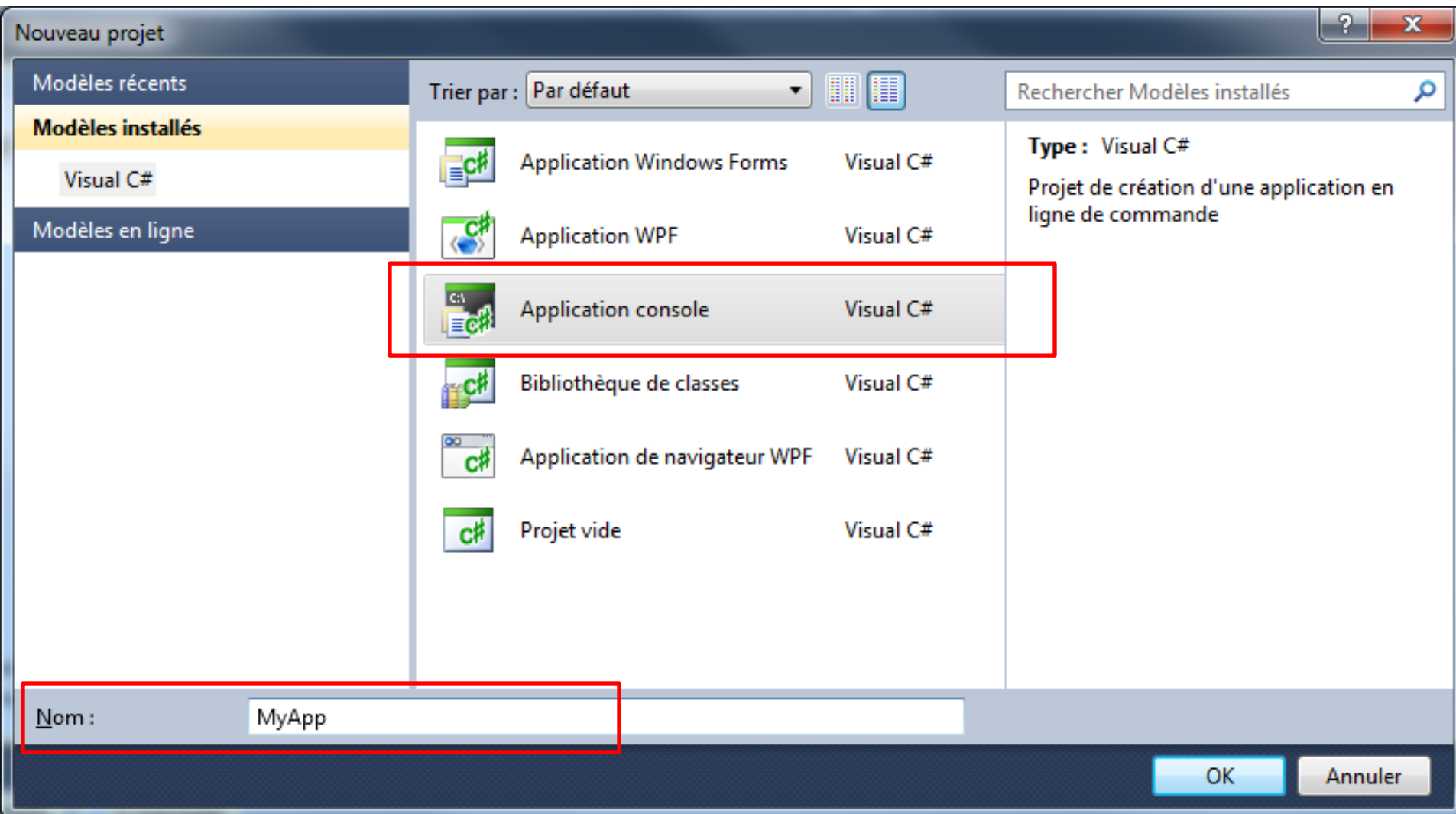
C# : mon premier programme



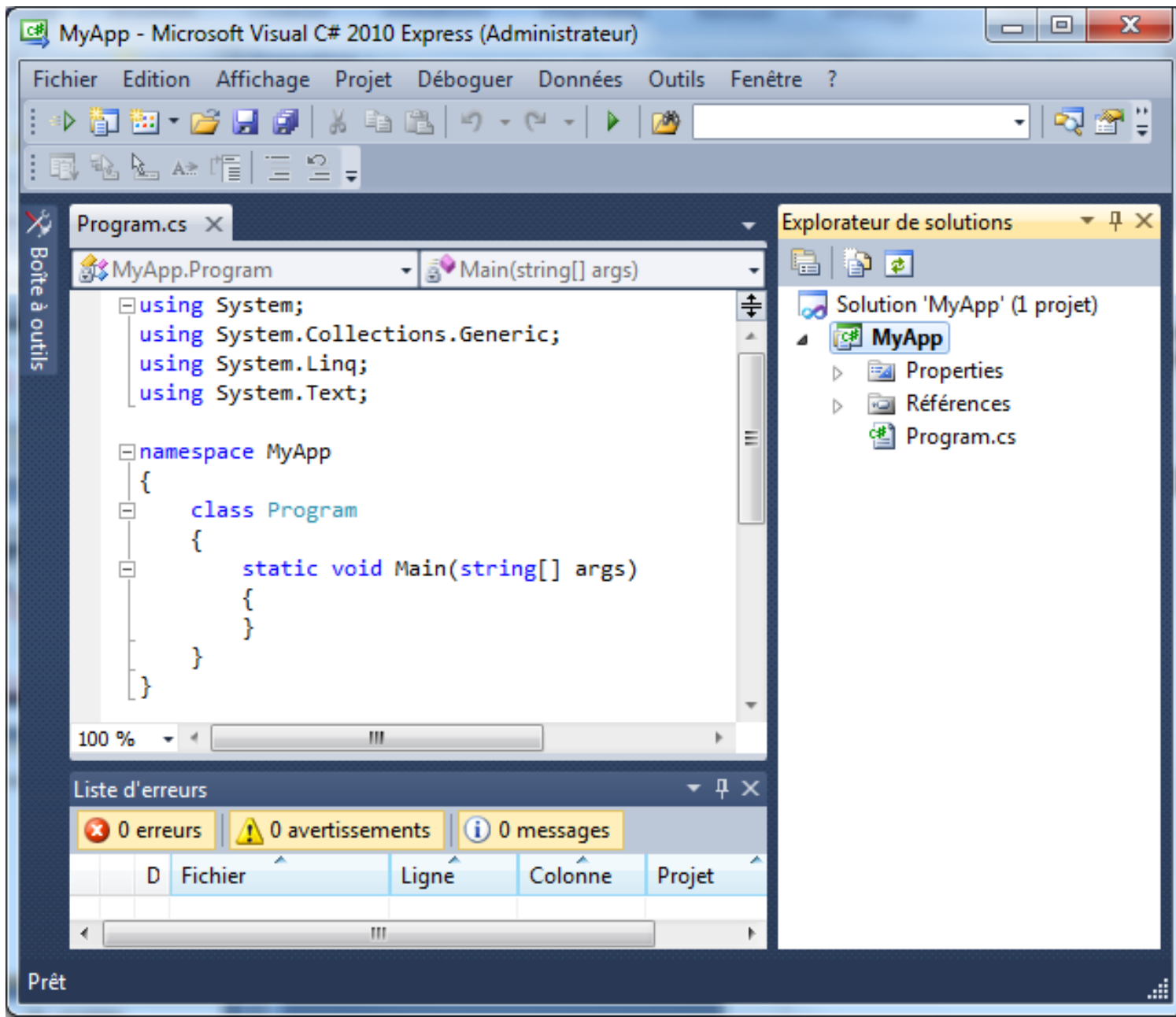
ou



C# : mon premier programme



C# : mon premier programme

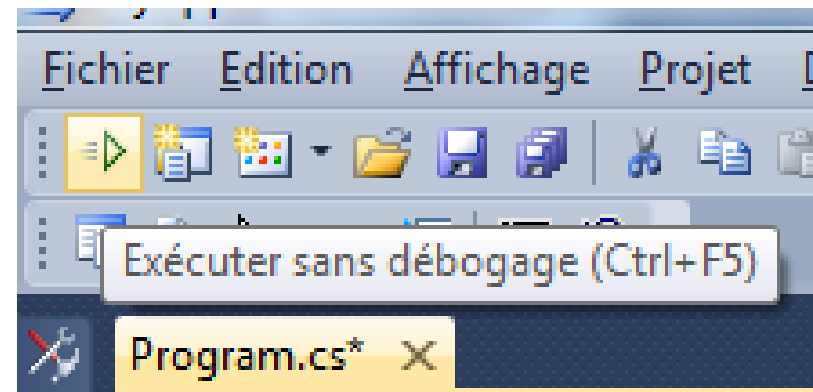
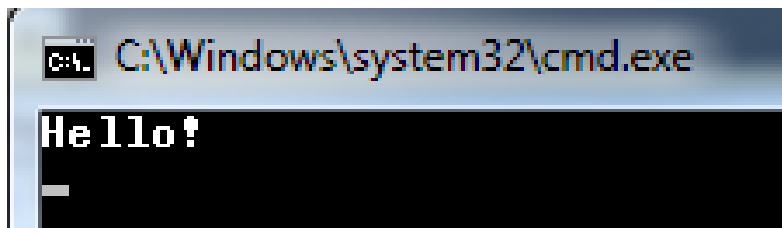


C# : mon premier programme

- Ajouter le corps du main

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello!");
        Console.ReadKey();
    }
}
```

- Sauvegarder et exécuter



C# : assembly?

- En C# on peut créer des **.exe** qui pourront directement être exécuté par le CLR, mais aussi des bibliothèques sous la forme d'un fichier possédant l'extension **.dll**.
- On appelle ces deux formes de programmes des assemblages « **assembly** ».
 - Les fichiers .exe sont des assemblys de processus
 - Les fichiers .dll sont des assemblys de bibliothèques
- Concrètement, le .exe servira à lancer une application alors qu'une dll pourra être partagée entre plusieurs applications .exe afin de réutiliser du code déjà écrit.
- Par abus **assembly** (tout court) sert à désigner uniquement les bibliothèques dont l'extension est .dll.

C# : variables

Type	Description	Espace mémoire
bool	Valeur booléenne	8 bits
char	Caractère Ascii	16 bits
string	Chaîne de caractères	n
sbyte	De -128 à 127	8 bits
byte	De 0 à 255	8 bits
short	De -32768 à 32767	16 bits
ushort	De 0 à 65535	16 bits
int	De -2147483648 à 2147483647	32 bits
uint	De 0 à 4294967295	32 bits
long	De -9223372036854775808 à 9223372036854775807	64 bits
ulong	De 0 à 18446744073709551615	64 bits
float	De 1.5×10^{-45} à 3.4×10^{38}	32 bits
double	De 5.0×10^{-324} à 1.7×10^{308}	64 bits
decimal	De 1.0×10^{-28} à 7.9×10^{28}	128 bits

C# : variables

Exemple de déclaration :

```
bool exempleBooleen = true; //true ou false
char exempleChar = 'w';
string exempleString = "une chaîne de
    caractères!";
sbyte exempleSbyte = 8;
byte exempleByte = 139;
short exempleShort = -345;
ushort exempleUshort = 35000;
int exempleEntier = 40000;
uint exempleUint = 79500;
```

C# : variables

Exemple de déclaration :

```
long exempleLong = 402340540607;
ulong exempleUlong = 345678905768;
float exempleFloat = 1.72f;
//C'est un point et pas une virgule !(f à la
    fin)
double exempleDouble =
    8.76543987652354235525e12d;
//(d à la fin)
decimal exempleDecimal =
    1.2456723425734738590342849249e-4m;
// nombres à très grande précision (m à la fin)
//déclaration d'une constante
const float g = 9.80665f;
```

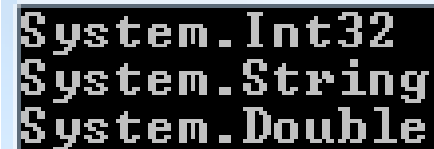
C# : variables

- Le type **var** est un type générique qui permet de stocker tout type de variable
 - inférence de type : le compilateur va rechercher le meilleur type pour stocker cette variable
- Attention : il faut que la variable soit locale et doit être initialisée au moment de la déclaration.
- Une variable définie à l'intérieur d'un bloc de code aura pour portée ce bloc de code et ses sous blocs.
 - Un bloc de code permet de regrouper des instructions qui commencent par `\{` et qui finissent par `\}`.

C# : variables

- Exemple :

```
static void Main(string[] args)
{
    var age1 = 17;
    var age2 = "Quarante ans";
    var age3 = 32.5;
    Console.WriteLine(age1.GetType());
    Console.WriteLine(age2.GetType());
    Console.WriteLine(age3.GetType());
    Console.ReadLine();
}
```

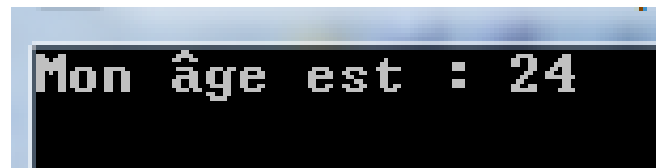


```
System.Int32
System.String
System.Double
```

C# : Conversions de données

- Le cast implicite : se fait si pas de perte de données.
 - exemple : convertir un type «int» en un type «long» (le type «long» englobe le type «int»).

```
int age = 24;  
long longAge = age;  
Console.WriteLine("Mon âge est : {0}", longAge);
```

A screenshot of a console window with a black background and white text. The text displayed is "Mon âge est : 24".

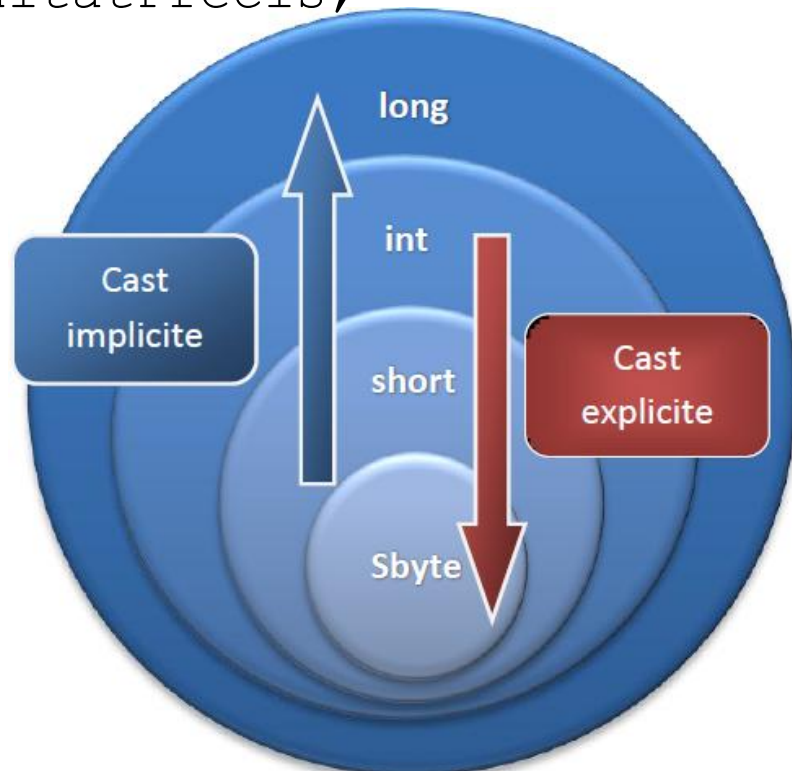
Mon âge est : 24

C# : Conversions de données

- Le cast explicite:
 - exp : convertir un « float » en «int»

```
float resultatPrecis = 17.83f;  
int resultat = (int)resultatPrecis;
```

Ce type de cast se fait
à la responsabilité du
programmeur



C# : Conversions de données

Conversion d'une chaîne en nombre

- On utilise la fonction « Parse » précédée du type voulu.
- Exp : convertir « x » en entier se fait par «int.Parse(x)»
- Il existe aussi des méthodes comme Convert.ToInt16 mais elles font appel au « parse » adéquat

```
string A = "42";  
String B = "23";  
int addition = int.Parse(A) + int.Parse(B);
```

C# : Conversions de données

Conversion d'un nombre en chaine :

- s'effectue grâce à la fonction « ToString() » que l'on place à la suite de la variable à convertir.

```
int n = 256;  
string ChaineNombre = n.ToString();  
Console.WriteLine(ChaineNombre);
```

C# : Conversions de données

Conversion en objet

- La conversion d'une variable en objet est appelée « Boxing » (inversement Unboxing)
- Exemple:

```
int nombre = 12;  
//Boxing  
object MonObjet = nombre;  
//Unboxing  
int j = (int)MonObjet;
```

C# : valeur et référence

Valeur	Référence
Tous les types Numériques	Les String
Les caractères	Les tableaux
Les booléens	Les classes
Les structures	Les délégués
Les énumérations	

- Les données de type valeur sont stockés dans la pile.
- Les données de type référence sont stocké dans une partie de la mémoire appelée le tas tandis qu'une référence pointant vers celle-ci se trouve dans la pile.

C# : opérateurs

- Opérateurs arithmétiques : +, -, *, / et %
- Opérateurs de test: >, <, >=, <=, !=, ==
- Opérateurs conditionnels : !, ||, &&, ? :
(Condition ? valeur_si_Vrai : valeur_si_Faux) et ?? (X ??
valeur si X est null)
- Opérateurs d'attribution : =, +=, -=, *=, /= et
%=
- Opérateurs d'incrémentement : ++ et --

C# : condition

```
if (condition)
{
    instructions;
}
else if (condition) //on en mettre autant
{
    instructions;
}
else
{
    instructions;
}
```

C# : condition

```
switch (expression)
{
case a :
    instruction_a;
    break;
case b :
    Instruction_b;
    break;
... // on peut mettre autant de case
default:
    instructionParDefaut;
    break;
}
```

C# : condition

- Exemple

```
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;
}
```


C# : boucle

- `while (expression)`
 {
 instructions;
 }
- `do`
 {
 instructions;
 }`while (expression);`
- `for (initialisation; expression; pas)`
 {
 instructions;
 }

C# : boucle

- Exemple

```
int x;  
x = 1;  
while (x!=0)  
{  
    Console.WriteLine("Entrez 0 pour sortir  
de la boucle");  
    x = Convert.ToInt16(Console.ReadLine());  
}
```

- `Convert.ToInt16()` transforme la chaîne de caractère récupérée par `ReadLine` en entier.

C# : boucle

- Exemple:

```
for (int i = 1; i <= 10; i++)  
{  
    Console.WriteLine("Bonjour " +  
        i + " fois");  
}
```

- + est l'opérateur de concaténation

C# : Les tableaux

- Un tableau permet de stocker plusieurs variables de même type à la suite dans des cases contigües de la mémoire.
- Exp : un tableau de 7 entiers
 - Déclaration puis initialisation

```
int[] monTableau = new int[7];
```

```
int[] monTableau = new int[] { 345, 0, -25, 7, 42, 23, 1337 };
```

- En un coup

```
int[] monTableau = { 345, 0, -25, 7, 42, 23, 1337 };
```

C# : Les tableaux

- Parcourir un tableau à l'aide de for :

```
int i;  
for (i = 0; i < monTableau. Length; i++)  
{  
    Console.WriteLine(monTableau[i]);  
}
```

- Parcourir un tableau grâce à foreach :

```
foreach (int a in monTableau)  
{  
    Console.WriteLine(a);  
}
```

C# : Les tableaux

Dimension multiples

- Pour déclarer un tableau à 2 dimensions il faut rajouter une virgule entre les crochets (2 virgules pour un tableau à 3 dimensions, ...)
- Les dimensions sont entre accolades et celles-ci sont séparées entre elles par une virgule, le tout étant lui aussi compris entre accolades.
- Exp

```
int[,] monTableau = { { 11, 12, 22, 33 } ,  
                      { 21, 22, 23, 24 } };
```

C# : Les tableaux

Parcourir un tableau à dimensions multiples (for)

```
int j; //indice de la dimension verticale.  
int i; //indice de la dimension horizontale  
  
for (j = 0; j < monTableau.GetLength(0); j++)  
{  
    for (i = 0; i < monTableau.GetLength(1); i++)  
    {  
        Console.Write("j={0} i={1} : ", j, i);  
        Console.WriteLine(monTableau[j, i]);  
    }  
}
```

C# : Les tableaux

Parcourir un tableau à dimensions multiples
(foreach)

```
foreach (int a in monTableau)
{
    Console.WriteLine(a);
}
```


C# : Les Listes

- Une liste peut être une alternative intéressante au tableau lorsque l'on ne connaît pas le nombre d'éléments à stocker à l'avance.
- Pour créer une liste rien de plus simple : on crée un objet de type liste pour le type désiré
- Exp :

```
List<int> MaListe1 = new List<int>();  
List<string> MaListe2 = new List<string>();
```

C# : Les Listes

- **Add()** : permet d'ajouter des éléments à la liste

```
MaListe1.Add(11);
```

- **Parcours avec for**

```
int i;  
for (i = 0; i < MaListe1.Count(); i++)  
{  
    Console.WriteLine(MaListe1[i]);  
}
```

- **Parcours avec foreach**

```
foreach (int elm in MaListe1)  
{  
    Console.WriteLine(elm);  
}
```

C# : Les Listes

- Les objets de type «List» possèdent de nombreuses méthodes :
 - Clear() : Supprime toute les valeurs
 - Insert(place,nb) : Insère notre nombre à la place désirée
 - Reverse() : Range la liste dans le sens inverse.
 - Count() : Count retourne le nombre d'éléments
 - Max() : Renvoie la valeur maximale de la liste
 - Min() : Renvoie la valeur minimale de la liste
 - Sum() : Renvoie la somme des éléments
 - BinarySearch(nombre) : Renvoie l'indice de l'objet a trouver
 - Contains(nombre) : Renvoie vrai si le nombre
- Utilisation : `MaListe1.Nom_méthode()`

C# : Les énumérations

- Une énumération permet de créer un type de variable afin d'en restreindre les valeurs possibles.

- Exp : jours de la semaine

```
enum Jours { lun, mar, mer, jeu,  
            ven, sam, dim };
```

- Ce type s'utilise dans la déclaration de variables

```
Jours jourDeStage;
```

```
jourDeStage = Jours.jeu;
```

C# : Les énumérations

- Remarques :
 - la déclaration d'une énumération ne peut s'effectuer à l'intérieur d'une fonction.
 - les membres d'une énumération peuvent également être appelés avec leur numéro

```
class Program
{
    enum Jours { lun, mar, mer, jeu, ven, sam, dim };
    static void Main(string[] args)
    {
        Jours jourDeStage; //variable de type "Jours"
        jourDeStage = (Jours)3; // = Jours.jeu;
        Console.WriteLine(jourDeStage);
        Console.ReadLine();
    }
}
```

C# : Les structures

- Une structure permet de rassembler plusieurs types de donnée dans un groupe plus grand.
- Une structure ressemble un peu à une classe mais elle est de type valeur. Une classe est de type référence.
- La déclaration se fait en dehors du main mais si les différents champs sont déclarés en «public» ils seront accessibles dans le main.

C# : Les structures

- Exp : un élève

```
class Program
{
    struct Eleve
    {
        public string nom;
        public int numIns;
    }
    static void Main(string[] args)
    {
        Eleve alami;
        alami.nom = "ALAMI Alem"; alami.numIns = 1234;
        Console.WriteLine("Nom : " + alami.nom
                           + " N° Inscription : " + alami.numIns);
        Console.ReadLine();
    }
}
```

C# : Les méthodes

- Une méthode se présente comme ceci :

```
Type_Retour Nom_Methode (type parametre)
{
    //Ici on met le code de la méthode
    return variableRetournee;
}
```

- void signifie que la méthode ne renvoie rien

C# : Les méthodes

- Exemple

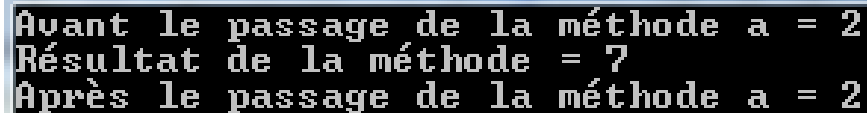
```
static int CalculAire (int longueur, int largeur)
{
    int aire = longueur * largeur;
    return aire;
}
static void Main(string[] args)
{
    int coteA = 3;
    int coteB = 7;

    Console.WriteLine(CalculAire(coteA, coteB));
    Console.ReadLine();
}
```

C# : Les méthodes

- Passage de paramètres par valeur :

```
static int notre_methode(int x)
{
    x = 7;
    return x;
}
static void Main()
{
    int a = 2;
    Console.WriteLine("Avant le passage de la méthode a = " + a);
    Console.WriteLine("Résultat de la méthode = " + notre_methode(a));
    Console.WriteLine("Après le passage de la méthode a = " + a);
    Console.ReadLine();
}
```

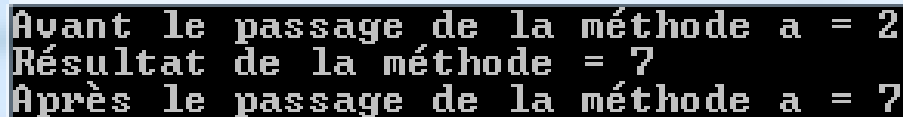


```
Avant le passage de la méthode a = 2
Résultat de la méthode = 7
Après le passage de la méthode a = 2
```

C# : Les méthodes

- Pour passer un paramètre par référence, il suffit d'ajouter le mot clé «ref» devant le paramètre, dans la fonction et dans l'appel.

```
static int notre_methode( ref int x)
{
    x = 7;
    return x;
}
static void Main()
{
    int a = 2;
    Console.WriteLine("Avant le passage de la méthode a = " + a);
    Console.WriteLine("Résultat de la méthode = " + notre_methode(ref a));
    Console.WriteLine("Après le passage de la méthode a = " + a);
    Console.ReadLine();
}
```



```
Avant le passage de la méthode a = 2
Résultat de la méthode = 7
Après le passage de la méthode a = 7
```

C# : gestion d'exception

- Certaines actions imprévues peuvent compromettre la fiabilité du programme : tentative de division par zéro, écriture sur un fichier en lecture seule ou inexistant, ...
- Exp :

```
static void Main(string[] args)
{
    int age = 0;
    Console.WriteLine("Quel âge avez-vous?");
    age = int.Parse(Console.ReadLine());
    Console.WriteLine("Vous avez " + age + " ans !");
    Console.ReadLine();
}
```

Et si l'utilisateur donnait son âge en toutes lettres ?

➔ gérer cette exception en utilisant **try** et **catch**

C# : gestion d'exception

- Le bloc **try** on y place le code à risque.
- Le bloc **catch** on y place le code à exécuter en cas de problème

```
static void Main(string[] args)
{
    int age = 0;
    Console.WriteLine("Quel âge avez-vous?");
    try
    {
        age = int.Parse(Console.ReadLine());
        Console.WriteLine("Vous avez donc " + age + " ans");
    }
    catch
    {
        Console.WriteLine("Désolé vous devez écrire un nombre!");
    }
    Console.ReadLine();
}
```

C# : gestion d'exception

- Le bloc **finally** contient du code qui sera toujours exécuté, qu'il y est une exception levée ou non. Ce bloc est facultatif.

```
static void Main(string[] args)
{
    int age = 0;
    Console.WriteLine("Quel âge avez-vous?");
    try {
        age = int.Parse(Console.ReadLine());
        Console.WriteLine("Vous avez donc " + age + " ans");
    }
    catch {
        Console.WriteLine("Désolé vous devez écrire un nombre!");
    }
    finally {
        Console.WriteLine("\n*** Merci de votre visite ***");
    }
    Console.ReadLine();
}
```

C# : gestion d'exception

- En cas d'exceptions multiples, il suffit de placer plusieurs blocs «catch» en leur associant l'exception appropriée :

```
static void Main(string[] args)
{
    try {
    }
    catch (Exception) {
    }
    catch (Exception) {
    }
}
```

C# : gestion d'exception

- Exp :

```
static void Main(string[] args)
{
    Console.Write("\nEntrez un premier nombre : ");
    try {
        uint a = uint.Parse(Console.ReadLine());
        Console.Write("Entrez un deuxieme nombre : ");
        uint b = uint.Parse(Console.ReadLine());
        uint c = a / b;
        Console.WriteLine("Le résultat de la division est " + c);
    } catch (OverflowException) {
        Console.WriteLine("Entrez un nombre positif s'il vous plait");
    } catch (DivideByZeroException) {
        Console.WriteLine("Désolé vous ne pouvez pas diviser par zero");
    } catch (FormatException) {
        Console.WriteLine("Les chaine de caractères ne sont pas autorisées");
    } finally {
        Console.WriteLine("\n*** Au revoir! ***");
    }
    Console.ReadLine();
}
```


C# : gestion d'exception

Lever une exception

- Il est possible de déclencher soi-même la levée d'une exception.
 - cas limite, fonctionnel ou technique.
- Utiliser le mot-clé **throw**, suivi d'une instance d'une exception.
- Exp : racine carrée
 - un cas limite → paramètre négatif

C# : gestion d'exception

Lever une exception

```
class Program
{
    public static double RacineCarree(double valeur)
    {
        if (valeur <= 0)
            throw new Exception("Le paramètre doit être positif");
        return Math.Sqrt(valeur);
    }

    static void Main(string[] args)
    {
        try
        {
            double racine = RacineCarree(-5);
        }
        catch (Exception ex)
        {
            Console.WriteLine(" Calcul Impossible -->" + ex.Message);
        }
    }
}
```

