

BASES DU C#

Et utilisation de visual studio

Le C# : C Sharp

- Développé par Microsoft pour le framework .net :
 - Librairie de fonctions très nombreuses
 - Ensemble de règles (pattern) pour développer une application
- Langage objet très proche du java et du C++
- Langage à typage fort

Outil de « dév » : Visual Studio

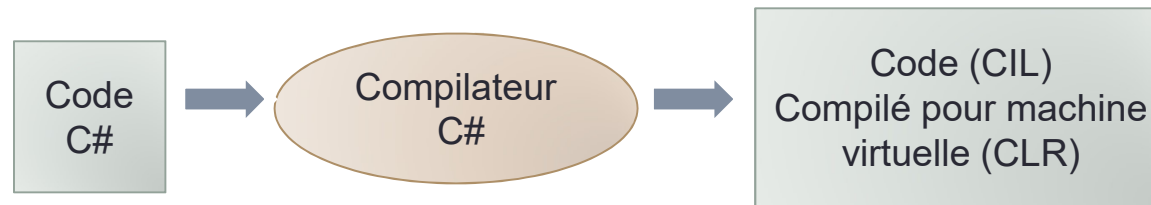
- Visual studio ultimate : ensemble complet d'outils pour développer :
 - Des applications consoles
 - Des applications fenêtrées
 - Des applications WEB
 - Des services WEB XML
 - Des applications mobiles

Visual Studio

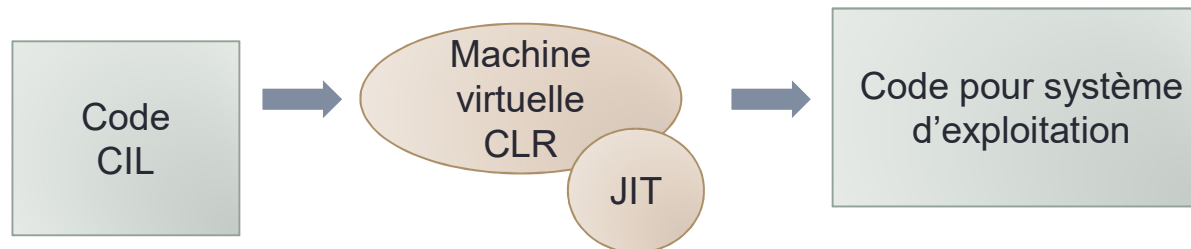
- Avec Visual studio ultimate : on peut faire du :
 - C#
 - C++
 - Visual Basic
 - ...
- Visual studio express (version allégée, 1 seul langage, gratuite pour tous dans un but d'apprentissage)

C# : langage compilé puis interprété

- **Phase de compilation:** code compilé et traduit en langage commun (CIL) pour la machine virtuelle CLR.



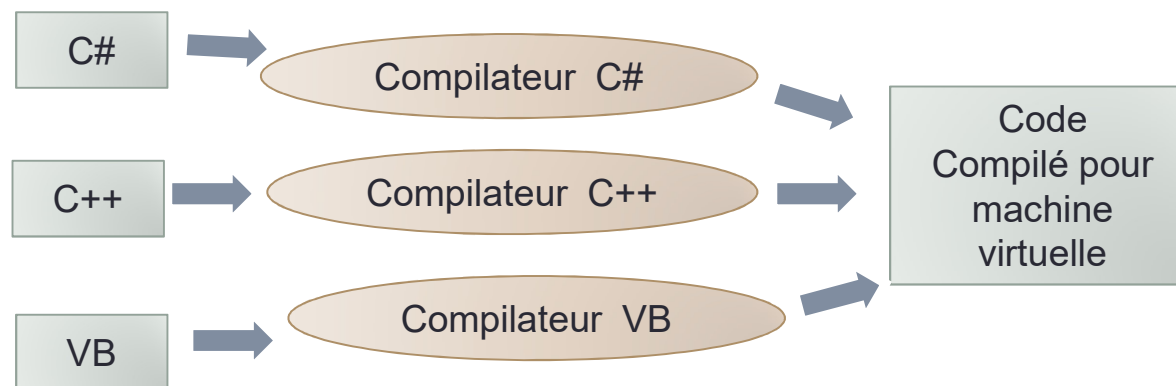
- **Phase d'interprétation :** code traduit à la volé par le compilateur JIT (Just In Time) de la machine virtuelle pour l'OS sur lequel elle est installée



Optimisation de la mémoire : garbage collector
Accélération de l'exécution du code en réutilisant
dès que possible le cache

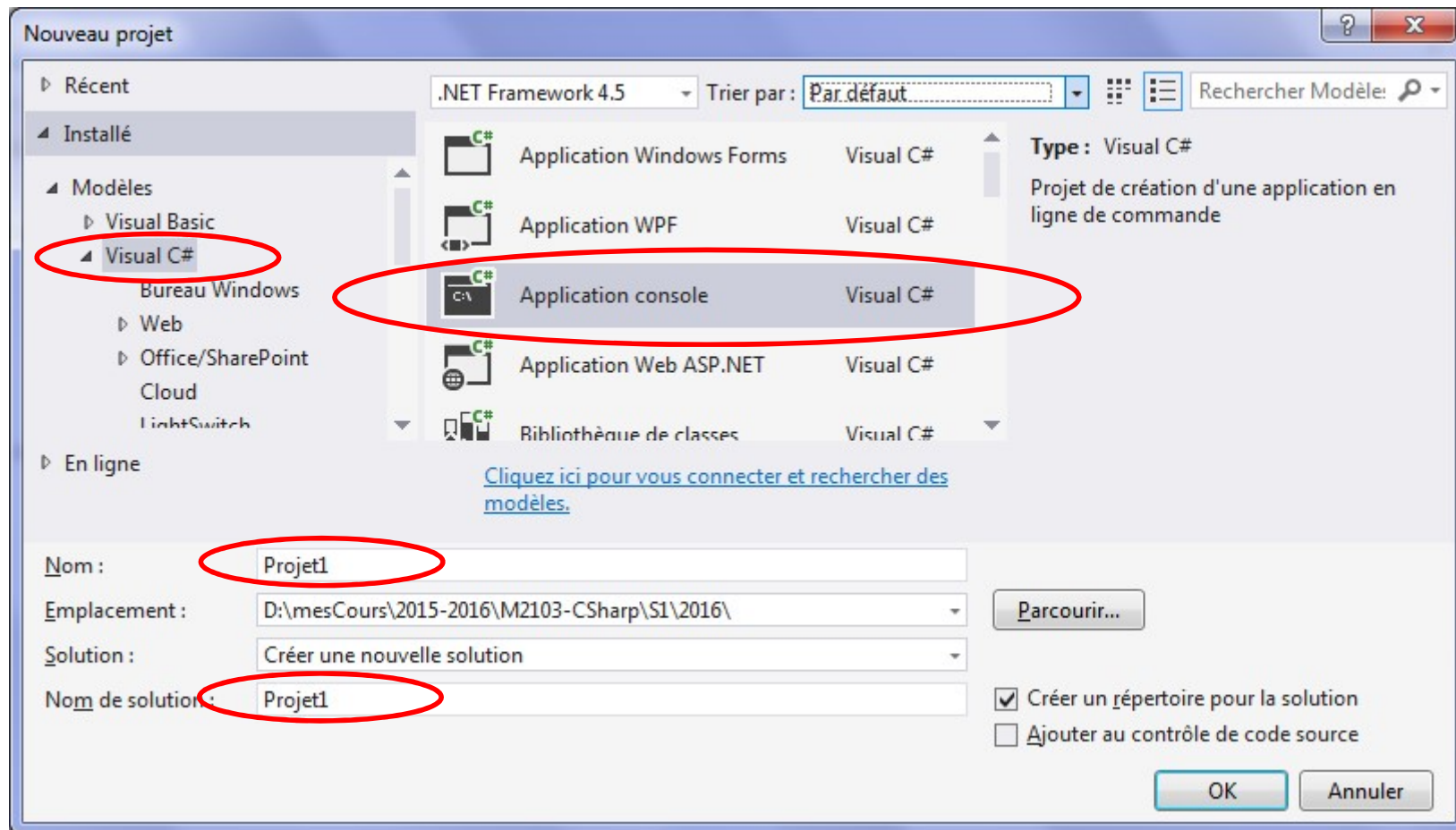
C# : langage compilé puis interprété

- Conclusion : Compiler : ne génère pas un fichier executable par un système d'exploitation, mais un fichier executable par une machine virtuelle.
- Spécificité de .net : hétérogénéité des langages



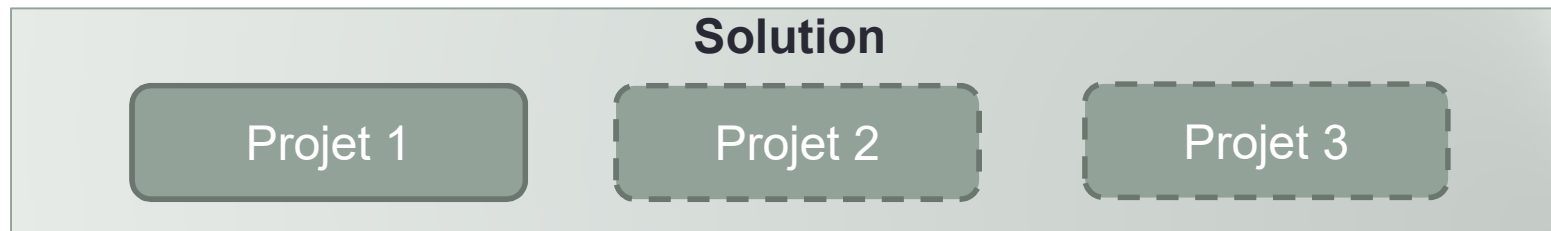
Organisation du code sous visual studio

- Il faut créer un projet.
- Par défaut, une solution de même nom est créée.

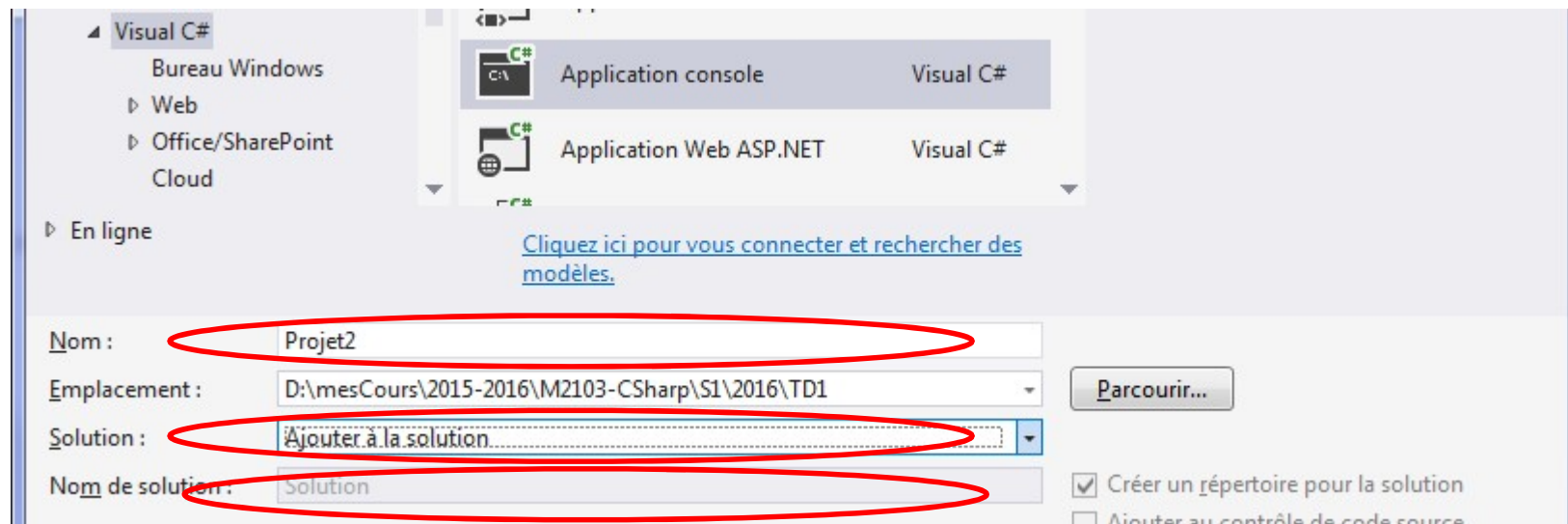


Organisation du code sous visual studio

- Solution = un ou plusieurs projets

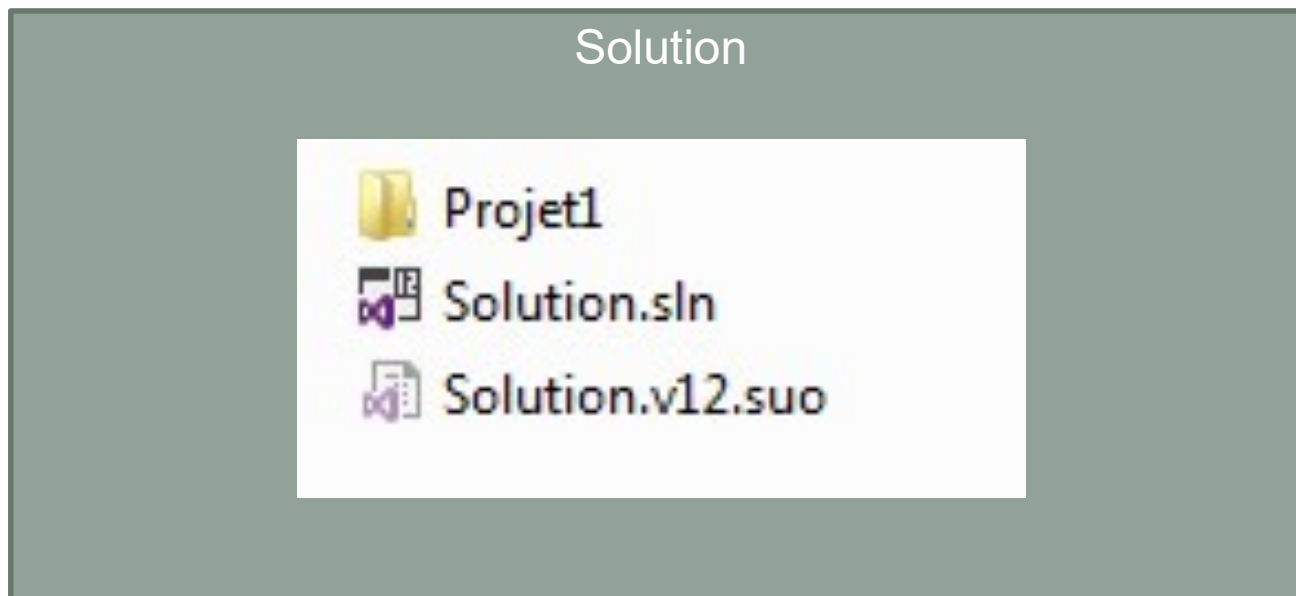


- On peut créer un projet et l'ajouter à une solution existante



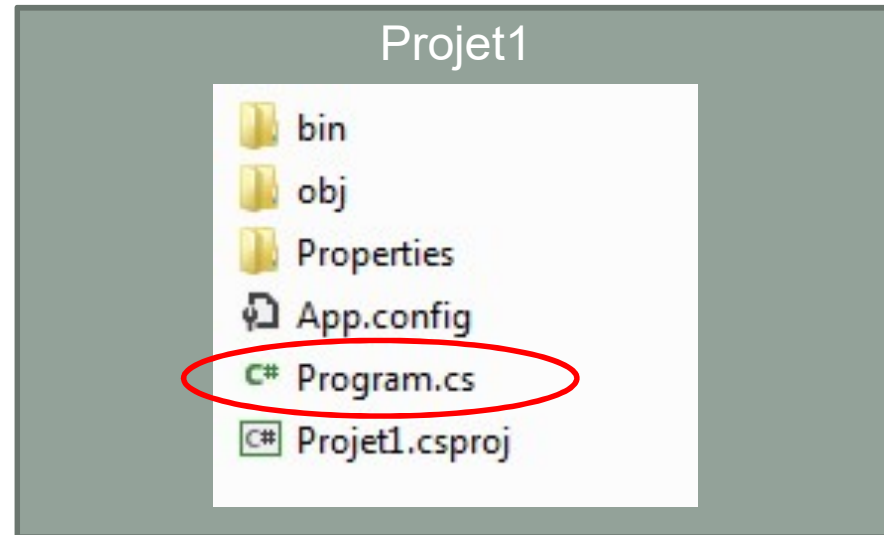
Contenu d'une solution

- Répertoire(s) : un par projet
- Sln : fichier de configuration (liste des projets)
- Suo : fichier contenant les options de la solution



Contenu d'un projet

- Répertoires
 - bin : fichiers exécutables
 - obj : fichiers de debug
 - Properties: paramètres de compilation

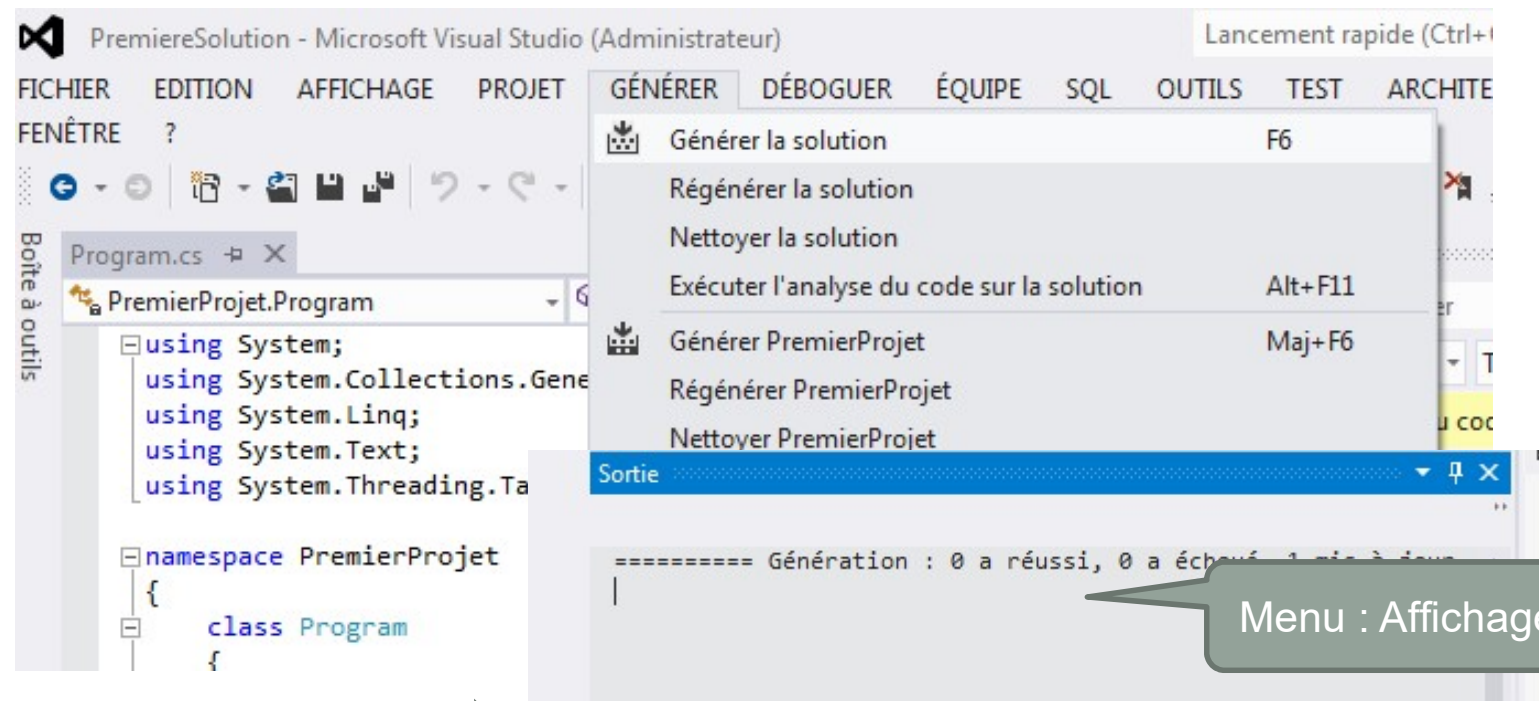


- **.config: fichier pour définir des variables « paramètres »**
utilisables dans le code source : à utiliser mais plus tard
- .csproj: fichier de configuration du projet (liste des fichiers, options de compilation....)

- **.cs: fichier de code source**

1ere compilation

- Générer la solution

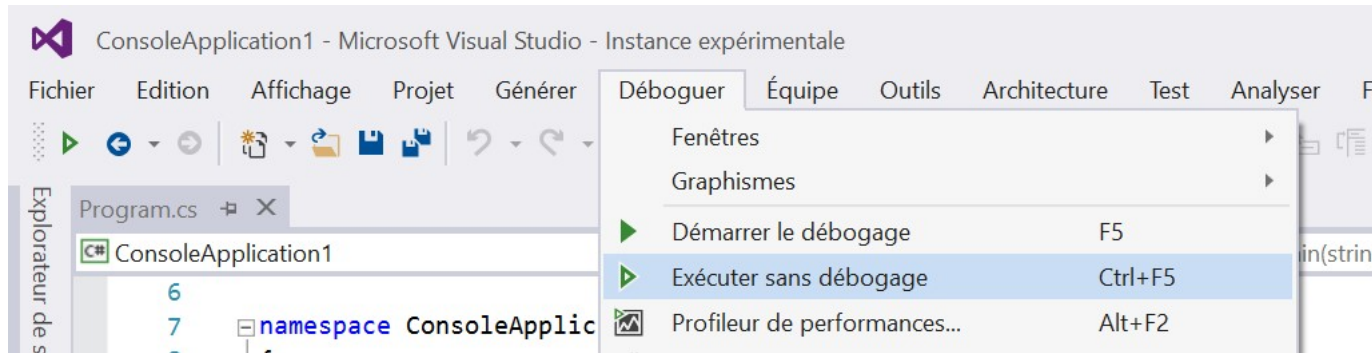


Génère un .exe

PremierProjet.exe
PremierProjet.pdb

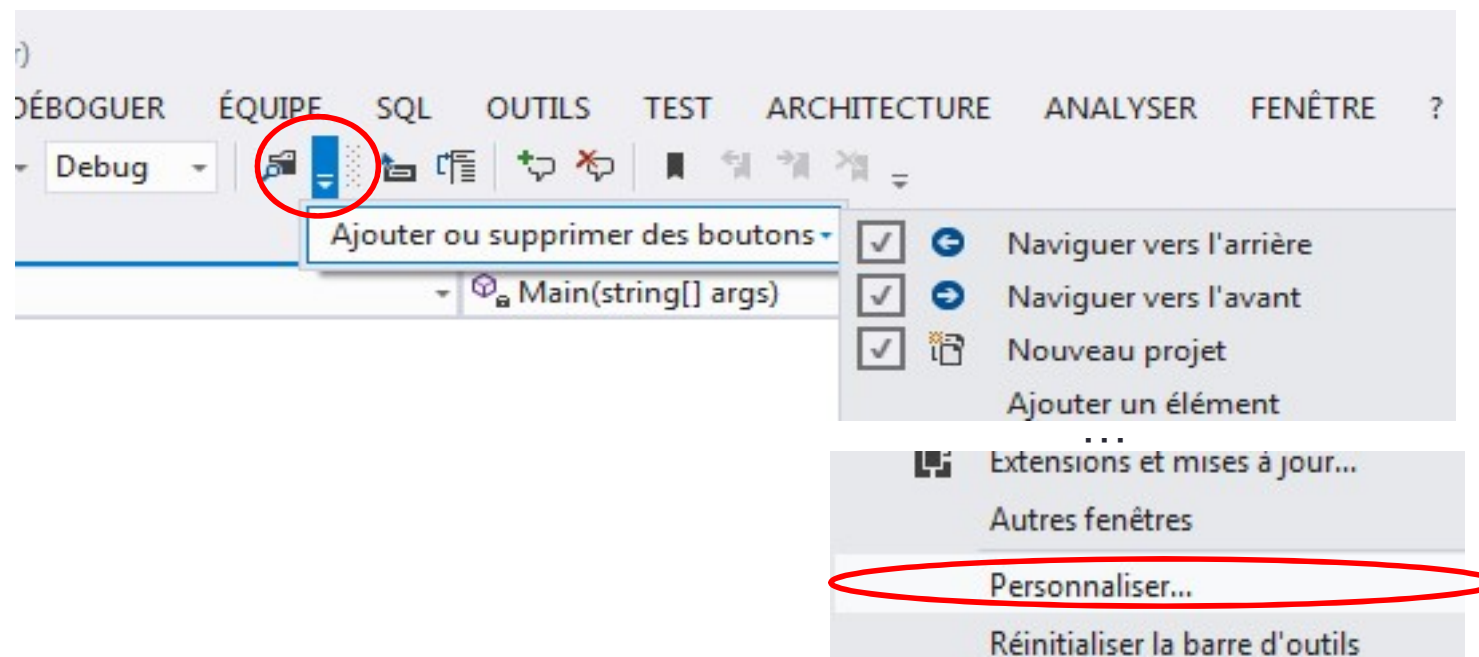
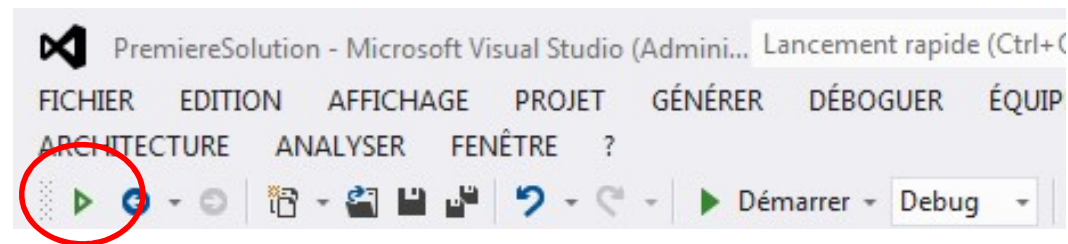
1ere execution

- Préférez le mode sans débogage (Ctrl + F5)

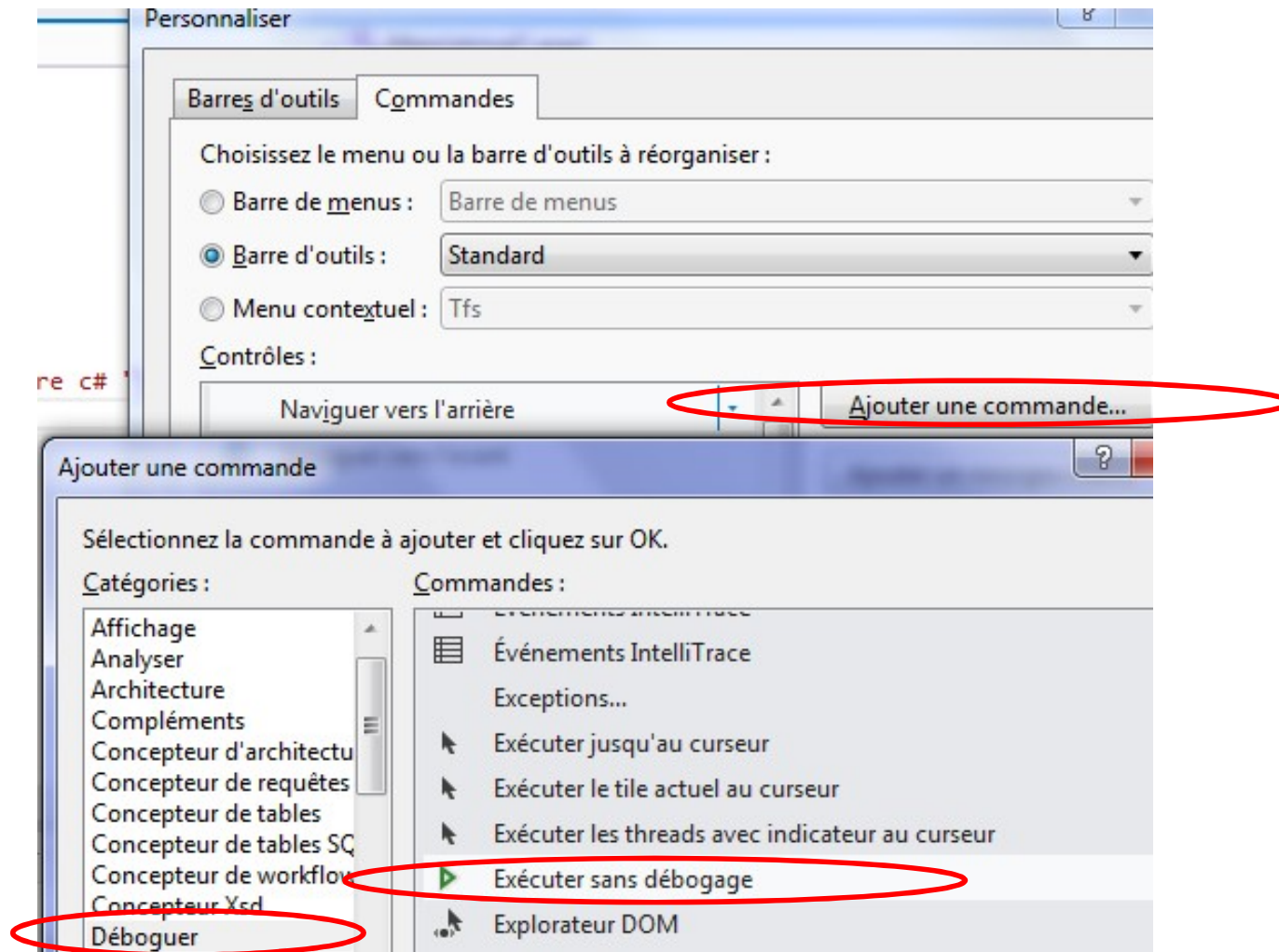


Paramétrez le bouton d'exécution

- Ajoutez un bouton de raccourci si nécessaire !



Paramétrez le bouton d'exécution



Squelette d'un programme

- Code généré automatiquement

The image shows a screenshot of a C# code editor with a project named 'Projet1'. The code is as follows:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Projet1
{
    // Références
    class Program
    {
        // Références
        static void Main(string[] args)
        {
        }
    }
}
```

Callouts explaining the code structure:

- using System; using System.Collections.Generic; using System.Linq; using System.Text; using System.Threading.Tasks;**: Liens vers les namespace (= répertoires) contenant les fonctions les plus utilisées
- namespace Projet1**: Namespace : répertoire pour organiser son code.
- class Program**: Class : fichier contenant le code.
- static void Main(string[] args)**: Main comme en C

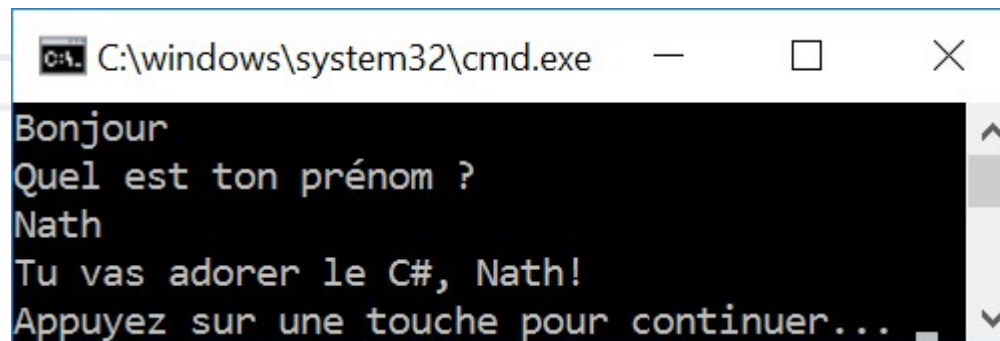
Un peu de vocabulaire

- Une classe = un fichier (c'est mieux)
- Une classe commence par une majuscule
- 3 sortes de classe :
 - 2 avec peu de concepts Objets :
 - La classe « Program » : contient le main d'un projet
 - Les classes « statics » : Console, Math : regroupent un ensemble de méthodes (fonctions) par thème.
 - Les « vraies » classes : elles définissent un nouveau type : sa structure, ses opérateurs, ses méthodes...
- Un espace de nom regroupe des classes par thème.

Ville	
Classe	
+ Champs	
- Propriétés	
🔑	NbHabitants : int
🔑	Nom : string
🔑	Superficie : double
- Méthodes	
🔑	CalculDensite() : double
🔑	Equals() : bool

1er programme

```
namespace ConsoleApplication1
{
    0 références
    class Program
    {
        0 références
        static void Main(string[] args)
        {
            Console.WriteLine("Bonjour");
            Console.WriteLine("Quel est ton prénom ?");
            String prenom = Console.ReadLine();
            Console.WriteLine("Tu vas adorer le C#, " + prenom + "!");
        }
    }
}
```



C:\windows\system32\cmd.exe

```
Bonjour
Quel est ton prénom ?
Nath
Tu vas adorer le C#, Nath!
Appuyez sur une touche pour continuer...
```

Complétion et détection d'erreurs

- Sans compilation : certaines erreurs auto détectées

```
namespace PremierProjet
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(" Premier ordre c# ");
        }
    }
}
```

100 %

Liste d'erreurs

1 erreur 0 avertissements 0 messages

	Description	Fichier	Ligne
1	'System.Console' ne contient pas de définition pour 'WriteLine'	Program.cs	13

Menu : Affichage/ liste d'erreurs

Structure conditionnelle : if

- Idem C, PHP, ...
- Rappel : attention à la portée des variables

Pas besoin
de { } s'il n'y
a qu'une
seule
instruction

```
if (condition)
{
    instructions;
}
```

```
if ( condition )
{
    instructions ;
}
else
{
    instructions;
}
```

```
if (condition)
{
    instructions ;
}
else if (condition)
{
    instructions;
}
else
{
    instructions;
}
```

Structure conditionnelle : switch

- Identique au C : ne teste que l'égalité !
- La variable peut être une chaîne

```
switch ( variable)
{
    case valeur1 : instructions;
                    break;
    case valeur2 : instructions ;
                    break;

    ...
    default : instructions;
              break;
}
```

Boucles : For, while, do ... while

- On peut déclarer le compteur au sein du for
- Rappel : attention à la portée des variables

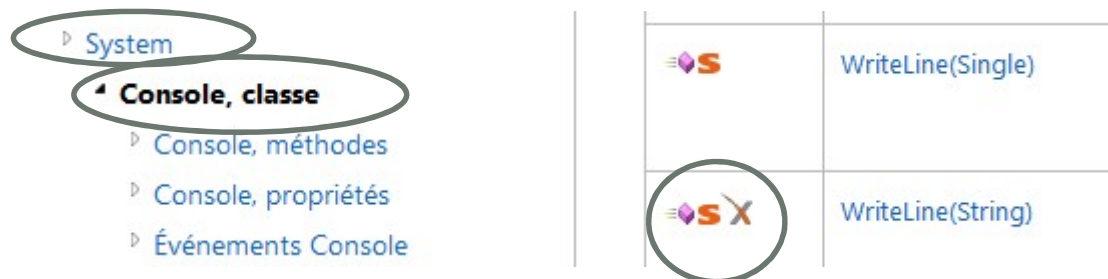
```
for (int i=0;i<5;i++)  
    // instruction à répéter
```

```
int i =0 ;  
do  
{  
    // instructions à répéter  
    i ++;  
} while (i<5);
```

Afficher : Console.WriteLine()

Write, WriteLine (avec saut de ligne) sont des méthodes :

- Définies dans la classe Console
- Rangée dans l'espace de nom System
- Statiques : leur appel doit être précédé de la classe
- Surchargées : plusieurs signatures



DOC MSDN

Et encore bien d'autres :

WriteLine(char)

WriteLine(double)

....

Afficher : Console.WriteLine()

C# C++ F# VB

DOC MSDN

```
[HostProtectionAttribute(SecurityAction.LinkDemand, UI = true)]  
public static void WriteLine(  
    string value  
)
```

Informatif : à ne pas recopier !

Exemple :

```
// si using System; en début de fichier  
Console.WriteLine("Bonjour !");  
  
// sinon  
// System.Console.WriteLine("Bonjour !");
```

WriteLine est une méthode statique de classe Console rangée dans l'espace de nom System

Récupérer une saisie utilisateur : Console.ReadLine()

ReadLine est une méthode :

- Définie dans la classe Console
- Rangée dans l'espace de nom System
- Statique : son appel doit être précédé de la classe

Console, classe

- Console, méthodes
- Console, propriétés
- Événements Console



ReadLine

L'utilisateur a appuyé. La touche enfoncée s'ajoute à titre dans la fenêtre de console.



ResetColor

Définit les couleurs de premier plan et d'arrière-plan de l'écrit avec leurs valeurs par défaut.

Attention : elle n'est pas surchargée, elle ne fait que renvoyer du texte.

Récupérer une saisie utilisateur : Console.ReadLine()

C# C++ F# VB DOC MSDN

```
[HostProtectionAttribute(SecurityAction.LinkDemand, UI = true)]  
public static string ReadLine()
```

Informatif : à ne pas recopier !

Exemple :

```
Console.WriteLine("Votre prénom : " );  
  
String prenom = Console.ReadLine();
```

Convertir la saisie utilisateur : TryParse()

- **ReadLine()** : renvoie une chaîne : string
- **A convertir si l'information n'est pas textuelle.**

```
Console.WriteLine("Quel age ?");  
String sAge = Console.ReadLine();  
sbyte age;  
while (SByte.TryParse(sAge, out age) == false)  
{  
    Console.WriteLine("Erreur de saisie. Veuillez saisir un nombre entre 0 et 255");  
    sAge = Console.ReadLine();  
}
```


out age ⇔ &age

TryParse : essaie de convertir sAge sous forme de sbyte, affecte le résultat de la conversion à la variable age passée par adresse si cela est possible puis renvoie le fait qu'elle y soit arrivée ou non.

Convertir la saisie utilisateur : TryParse()

TryParse() est une méthode :

- **définie pour chaque type primitif dans sa structure associée :**
 - Int16 pour les short, Int32 pour les int, Double,
- **statique : son appel doit donc être précédé par la structure appropriée : Int16, Int32,...**

	<code>TryParse(String, Int16)</code>	Convertit la représentation d'un nombre sous forme de chaîne en entier 16 bits équivalent. Une valeur de retour indique si la conversion a réussi ou échoué.
---	--------------------------------------	--

C#	C++	F#	VB
-----------	------------	-----------	-----------

```
public static bool TryParse(  
    string s,  
    out short result  
)
```

OUT : passage par adresse (& en C et PHP)
Pour permettre la modification de la valeur

Convertir une saisie utilisateur

Il existe d'autres méthodes :

- Parse() au sein de chaque structure : Int16, Int32,...
- Des méthodes statiques de la classe Convert :
 - ToDouble ()
 - .ToInt16 ()
 - ...

Mais elles peuvent provoquer des exceptions (erreurs), si jamais le texte ne peut être converti dans le type demandé !

```
sbyte age = SByte.Parse(sAge);
```

```
Quel age ?  
AZE
```

```
Exception non gérée : System.FormatException: Le format de la chaîne  
'entrée est incorrect.  
à System.Number.StringToNumber(String str, NumberStyles options, N  
umberBuffer& number, NumberFormatInfo info, Boolean parseDecimal)  
à System.Number.ParseInt32(String s, NumberStyles style, NumberFor  
matInfo info)
```

Langage typé

- Il existe 2 grandes catégories de type :

- Les types primitifs. Ex : int, float, double,...

```
int age = 18 ;
```

Pour variable contenant 1 info. On dispose alors de qq opérateurs

- Les types complexes :

- Structures ou classes Ex : DateTime, Point, Int16, String ...

```
DateTime remiseDiplome = new DateTime (2017,6,30);
```

Pour variable contenant plusieurs infos. On dispose alors de qq opérateurs et de bons nombres de méthodes supplémentaires.

- Une structure a été mise en place pour chaque type primitif afin de l'enrichir.

Types primitifs : Les entiers

Type C#	valeurs	Structure
sbyte (byte)	-128 à 127 (0 à 255)	System.Sbyte (System.Byte)
short (ushort)	-32768 à 32767 (0 à 65 535)	System.Int16 (System.UInt16)
int (uint)	-2147483648 à 2147483647 (0 à 4 294 967 295)	System.Int32 (System.UInt32)
long (ulong)	-9223372036854775808 à 9223372036854775807 (0 à 18 446 744 073 709 551 615)	System.Int64 (System.UInt64)

Exemple :
byte age = 25;

Types primitifs : Les réels

Type C#	valeurs	précision	Structure
float	$\pm 1.5 \times 10^{-45}$ à $\pm 3.4 \times 10^{38}$	7 chiffres	System.Single
double	$\pm 5.0 \times 10^{-324}$ à $\pm 1.7 \times 10^{308}$	15-16 chiffres	System.Double
decimal	$\pm 1.0 \times 10^{-28}$ à $\pm 7.9 \times 10^{28}$	28-29 chiffres significatifs	System.Decimal

Exemples :

double prix = 39.99;

float txRemise = 0.50F;

Attention : on met suffixe F pour différencier un float d'un double, car par défaut double

Conversion de types primitifs proches

- Implicite

Exemple :

```
int iAge = 25 ;  
float fAge = iAge ;
```

- Explicite :

Exemple :

```
float fAge = 35.5F;  
int iAge = (int) fAge;
```

- Remarque : la conversion peut être source d'erreur si la donnée n'est pas adaptée au nouveau type.

Les autres types primitifs

- Le type caractère : char – Structure associée [System.Char](#)
 - Attention : char et String sont <>

Exemples :

```
char c_categorie='A';  
String s_categorie ="A" ;
```

- Le type booléen : bool – Structure associée [System.Boolean](#)

Exemples :

```
bool trouve = true;
```

Opérations sur les variables

- Opérateurs mathématiques: + - * / %

Exemple :

```
int age = 30;  
age = age - 5 ; // ou age -= 5;  
age = age / 2 ;
```

25/2 = 12 !
Rappel : / entiers

- Opérateurs d'incrémentation ou de décrémentation: ++, --

Exemple :

```
int age = 10 ;  
age++;  
char c = 'a';  
c ++ ;
```

Constante

- Une constante :

Exemple :

const int majorite =18 ;

String : une classe

- La classe String :
 - Permet de définir des variables de type chaîne de caractères
 - Permet d'avoir des infos sur la chaîne via des propriétés :
 - Length
 - Chars[int]
 - Permet d'avoir des méthodes pour travailler avec la chaîne :
 - public string ToUpper()
 - public string ToLower()
 - public bool StartsWith(string value)
 - public bool EndsWith(string value)
 - public int IndexOf(string value)
 -

Remarque : public signifie que c'est utilisable au sein de vos programmes.

String : une classe

- Pour en savoir plus : documentation MSDN

- › MSDN Library
- › Développement .NET
- › .NET Framework 4.5
- › Bibliothèque de classes .NET Framework
- › System
 - ▾ **String, classe**
 - › String, constructeur
 - › Champs String
 - › Méthodes String
 - › Opérateurs String
 - › Propriétés String

▲ Propriétés

Afficher: ☒ Hérité ☒ Protégé

	Nom	Description
	Chars	Obtient l'objet <code>Char</code> à une position de caractère spécifiée dans l'objet String en cours.
	Length	Obtient le nombre de caractères dans l'objet String actuel.

Début

▲ Méthodes

Afficher: ☒ Hérité ☒ Protégé

	Nom	Description
	Clone	Retourne une référence à cette instance de String.
	Compare(String, String)	Compare deux objets String spécifiés et retourne un entier qui indique leur position relative dans

Variable de classe String => Objet

- Classe = Type complexe
- Objet = Variable complexe
- Un objet String est une exception, car on affecte un String comme on affecte une variable primitive

Exemple :



```
String prenom = "julien";
```

```
int age = 20 ;
```

string alias de String : on peut donc omettre la majuscule

Manipuler les propriétés d'un String

- Pour utiliser les propriétés : `objet.Propriété`

	Nom	Description
	<code>Chars</code>	Obtient l'objet <code>Char</code> à une position de caractère spécifiée da
	<code>Length</code>	Obtient le nombre de caractères dans l'objet <code>String</code> actuel.

Exemple :

```
String prenom = "julien";
```

```
Console.WriteLine("Nombre de lettres :" + prenom.Length);
```

```
Console.WriteLine("1ere lettre :" + prenom.Chars[0] );
```



Utiliser les méthodes d'un String

- Pour utiliser les méthodes : objet.methode()
- Objet = variable complexe qui peut exécuter une méthode (traitement)

Exemple :

```
String prenom = "julien";
```



```
String prenomEnMaj = prenom.ToUpper();  
Console.WriteLine("prenom :" + prenomEnMaj);
```

Et non ToUpper (prenom) !!!

Concaténation

- Opérateur : +

Exemple :

```
int age = 25;  
String msg = "J'ai " + age + " ans";
```

Remarque : on peut concaténer tout type de données...

Attention : Méthodes statiques


- Certaines méthodes (résidus de la programmation procédurale) s'exécutent à partir de la classe et non de l'objet !



`Compare(String, String)`

Compare deux objets String spécifiés et retourne un entier qui indique leur position relative dans

```
String prenom = "nathalie";  
String prenomEnMaj = prenom.ToUpper();  
Console.Write(prenom + " et " + prenomEnMaj );
```



```
if ( String.Compare(prenom, prenomEnMaj)!=0)  
    Console.WriteLine( " ne sont pas identiques");  
else  
    Console.WriteLine( " sont identiques");
```

Structure/Classe

- Structure = Ancêtre des classes:
 - Fournit des constructeurs, des propriétés, des méthodes et des opérateurs
 - Ne permet pas l'héritage : concept POO étudié plus tard
 - Affectation par recopie de valeur (et non référence)

Structure/Classe - constructeur



- Initialisation d'une variable structurée ou objet avec :
 - Un constructeur : méthode conçue pour initialiser l'objet.
 - Le mot clef new

Exemple :

```
DateTime dRemise = new DateTime(2017, 6, 30);
```

Constructeurs

surcharge

	Nom	Description
	<code>DateTime(Int32, Int32, Int32)</code>	Initialise une nouvelle instance de la structure DateTime avec l'année, le mois et le jour spécifiés.
	<code>DateTime(Int32, Int32, Int32, Calendar)</code>	Initialise une nouvelle instance de la structure DateTime avec l'année, le mois et le jour spécifiés pour le calendrier spécifié.

Structure/Classe - propriétés

- Accès aux propriétés






Exemple :

```
DateTime dRemise = new DateTime(2017, 6, 30);  
Console.WriteLine("Ce sera le " + dRemise.DayOfYear + " eme jour  
de l'année");
```

Propriétés

Obtient ⇔ Propriété
avec un accès lecture
uniquement



	Nom	Description
	Date	Obtient le composant « date » de cette instance.
	Day	Obtient le jour du mois représenté par cette instance.
	DayOfWeek	Obtient le jour de semaine représenté par cette instance.
	DayOfYear	Obtient le jour de l'année représenté par cette instance.
	Hour	Obtient le composant « heure » de la date représentée par cette instance.

Structure/Classe - propriétés




- Accès aux propriétés

Autre Exemple avec la structure Point :

```
Point p = new Point( 6, 30);  
Console.WriteLine ( p.X);  
p.X=10;
```

Propriétés

Obtient ou définit ⇔
Propriété modifiable

	Nom	Description
	IsEmpty	Obtient une valeur indiquant si ce Point est vide.
	X	Obtient ou définit la coordonnée x de ce Point.
	Y	Obtient ou définit la coordonnée y de ce Point.

Structure/Classe - méthodes



- Accès aux méthodes

Exemple :

```
DateTime dRemise = new DateTime(2017, 6, 30);  
DateTime dLimite = dRemise.AddDays(7);
```

Méthodes



	Nom	Description
	Add(TimeSpan)	Retourne un nouveau DateTime qui ajoute la valeur du TimeSpan spécifié à la valeur de cette instance.
	AddDays(Double)	Retourne un nouveau DateTime qui ajoute le nombre de jours spécifié à la valeur de cette instance.
	AddHours(Double)	Retourne un nouveau DateTime qui ajoute le nombre d'heures spécifié à la valeur de cette instance.

Structure/Classe - opérateurs




- Observez la documentation pour connaître les opérateurs surchargés

Exemple :

```
DateTime d = new DateTime(2017, 6, 30);  
DateTime nd= d + new TimeSpan(7, 0, 0, 0);  
if (d == nd)  
    Console.WriteLine("Etrange !");  
TimeSpan temps = nd - d;
```

Opérateurs



	Nom	Description
	Addition(DateTime, TimeSpan)	Ajoute un intervalle de temps spécifié à une date et une heure spécifiées, générant une nouvelle date et heure.
	Equality(DateTime, DateTime)	Détermine si deux instances spécifiées de DateTime sont égales.
	Subtraction(DateTime, DateTime)	Soustrait une date et une heure spécifiées des autres date et heure spécifiées, et retourne un intervalle de temps.

Tableaux / listes

- Pour gérer un ensemble d'informations de même type
- Choix entre tableaux et listes:
 - Tableau : taille fixe et multidimensionnel

=> utile pour des constantes ou données paramètres dont le nombre est fixé. Par exemple : jour de la semaine, tarifs

- Liste : taille variable

=> utile pour des listes d'informations variables : liste de produits, de clients ...

Tableaux

- Comme en C : taille fixe
- Initialisation lors de la déclaration : pas de taille

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

- Déclaration puis initialisation

```
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";
```

new = allocation
mémoire contigue

Parcours de tableau

- Pour le lire :

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

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

- Pour le modifier :

```
for (int i=0 ; i < jours.Length ; i++)
{
    jours[i] = jours[i] + "s";
}
```

Méthodes pour les tableaux

Méthodes statiques de classe Array rangées dans l'espace de nom System :

- int IndexOf(Array array, Object value) : retourne la position de la 1ere occurrence de la valeur au sein du tableau
- int LastIndexOf(Array array, Object value) : retourne l'index de la dernière occurrence de la valeur au sein du tableau
- void Reverse (Array) : inverse
- void Sort (Array) : trie

```
String [ ] jours = new String [ ] { "Lundi", "Mardi", "Mercredi", ... "Dimanche" };  
int posMercredi = Array.IndexOf(jours, "Mercredi");  
Console.WriteLine("Mercredi c'est le jour " + (posMercredi+1));
```



```
Array.Sort(jours);
```


Listes : list <T>

- Initialisation lors de la déclaration : pas de taille

```
List<String> prenom = new List<String> { "Franck", "Aimerick", ... };
```

- Déclaration puis initialisation à l'aide de la méthode Add:

```
List<String> prenom = new List<String>(); // création de la liste
```



```
prenom.Add("Franck");  
prenom.Add("Aimerick");  
prenom.Add("Elodie");
```

Parcours de liste

- Pour le lire :

```
foreach (String prenom in prenoms)  
{  
    Console.WriteLine(prenom);  
}
```

- Pour le modifier :

```
for (int i = 0; i < prenoms.Count; i++)  
    prenoms[i] = prenoms[i] + "s";
```

Même syntaxe que
pour un tableau !

Méthodes pour les listes : list <T>

Méthodes pour les objets de classe List :

- int IndexOf(T value) : retourne la position de la 1ere occurrence de la valeur au sein du tableau
- int LastIndexOf(T value) : retourne l'index de la dernière occurrence de la valeur au sein du tableau
- void Reverse () : inverse
- void Sort () : trie
- void Insert(int index, T item)
- ...

```
List<String> prenom = new List<String>(); // création de la liste
prenom.Add("Franck");
prenom.Add("Aimerick");
prenom.Add("Elodie");
prenom.Sort();
```

