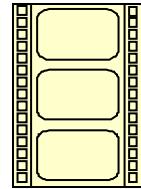


Initiation au langage C#

**Abdallah MOUJAHID
PMP®, COBIT® V5 , ITIL® V3, ISO 27002**



Plan

- ✓ **Eléments de base**
- ✓ Structures de contrôle
- ✓ Classes & objets
- ✓ Héritage, polymorphisme & interfaces
- ✓ Autres éléments de C#

Eléments de base

Introduction

- Langage de programmation créée par Anders Hejberg (Microsoft) en 2001
- Langage orienté objet, inspiré de C++ et Java
- Permet d'utiliser le framework .NET de manière optimale
- Dernière version: 4.6 (Framework) & 6.0 (Langage – 07/2015)
- Classement TIOB des langages les plus utilisés au monde: 4^{ème} en Septembre 2015 derrière Java, C et C++.

Eléments de base

Déclaration & Assignment

- Déclaration d'une variable

```
int a;  
int b = 5;  
double c = a + b;
```

- Déclaration d'une constante

```
const float PI = 3.14;
```

- Assignation d'une valeur

```
a = 6;  
c = a + 4  
c += 9;
```

- Commentaire

// Commentaire sur une ligne

```
/*  
 * Commentaires sur plusieurs lignes  
 */
```

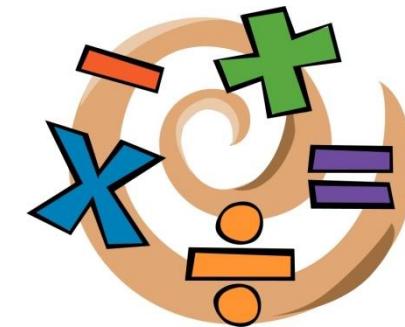
Eléments de base

Exemples de types existants

C#	C / C++
int	int
long	long
char	char
string	String
bool	bool
double	double
float	float
decimal	float

Eléments de base

Opérateurs



- Arithmétiques

$a + b$	Addition
$a - b$	Soustraction
a / b	Division
$a \% b$	Modulo
$a += b$	a est égal à la somme de a et de b
$a -= b$	a est égal à la soustraction entre a et b

- Comparaison

$a == b$	Egal
$a != b$	Différent
$a > b$	Supérieur
$a >= b$	Supérieur ou égal
$a < b$	Inférieur
$a <= b$	Inférieur ou égal

Eléments de base

La méthode Main() (1/2)

- Chaque assemblage exécutable a au moins une méthode statique Main() dans une de ses classes.
- Cette méthode représente le *point d'entrée du programme*, c'est-à-dire que le thread principal créé automatiquement lorsque le processus se lance va commencer par exécuter le code de cette méthode.
- Il peut y avoir éventuellement plusieurs méthodes Main() (chacune dans une classe différente) par programme. Le cas échéant, il faut préciser au compilateur quelle méthode Main() constitue le point d'entrée du programme.

Eléments de base

La méthode Main() (2/2)

- Le programme suivant ajoute les nombres passés en arguments en ligne de commande, et affiche le résultat. S'il n'y a pas d'argument, le programme le signale :

```
using System;
class Prog
{
    static void Main(string[] args)
    {
        if (args.Length == 0)
            Console.WriteLine("Entrez des nombres à ajouter.");
        else
        {
            long result = 0;
            foreach( string s in args )
                result += System.Int64.Parse(s);
            Console.WriteLine("Somme de ces nombres
:{0}",result);
        }
    }
}
```

Eléments de base

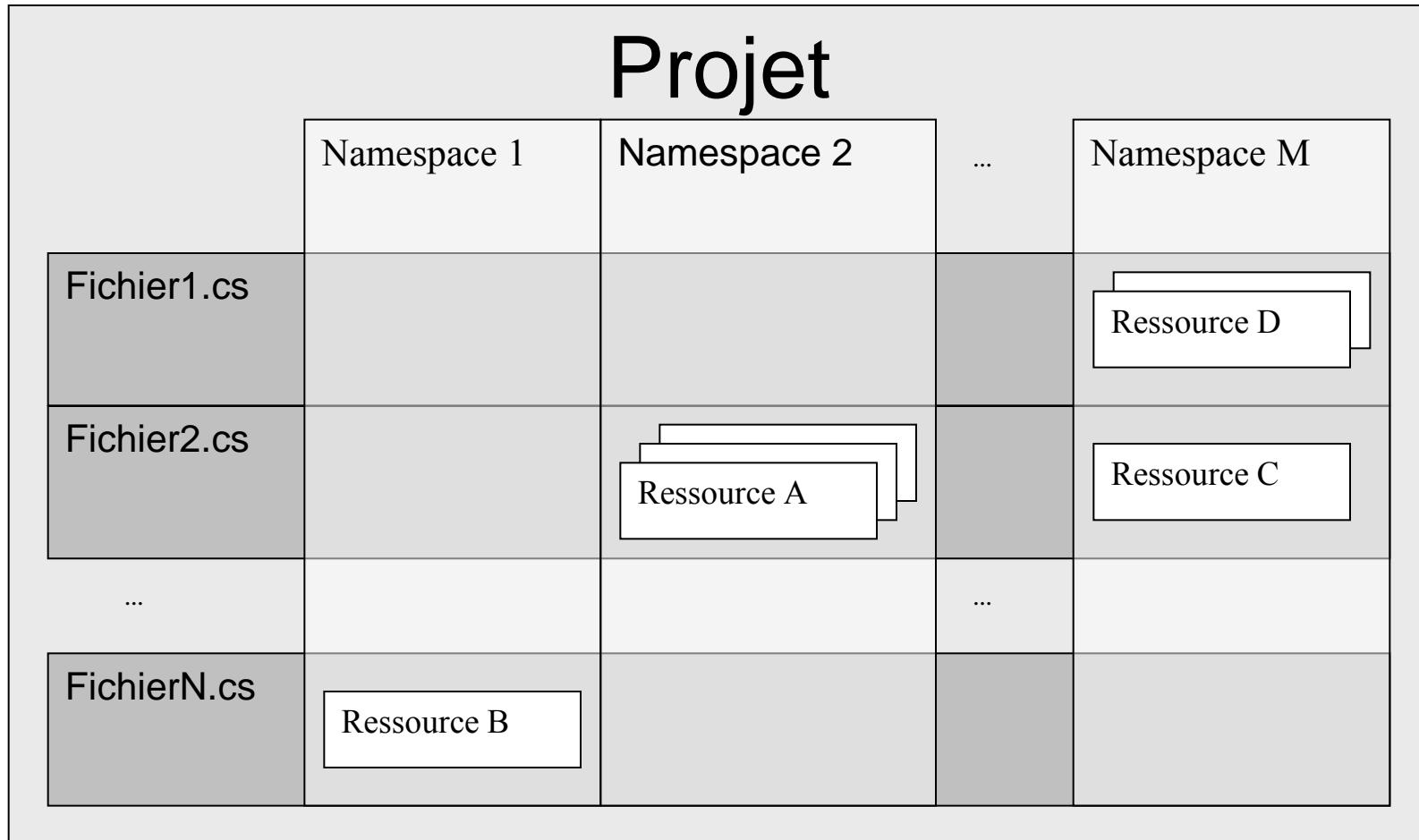
Les espaces de noms

- Une ressource dans le code source a la possibilité d'être déclarée et définie à l'intérieur d'un espace de noms.
- Si une ressource n'est déclarée dans aucun espace de noms elle fait partie d'un espace de noms global et anonyme.

```
using Foo1;
using Foo1.Foo2;
using System;
namespace Foo1
{
    // ici les ressources de l'espace de noms
    Foo1
    namespace Foo2
    {
        // ici les ressources de l'espace de noms
        Foo1.Foo2
    }
    // ici les ressources de l'espace de noms
    Foo1
}
```

Eléments de base

Organisation du code source

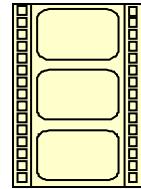


Eléments de base

Hello World !!!

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

namespace ConsoleApplication1
{
    class HelloWorld
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```



Plan

- ✓ Eléments de base
- ✓ **Structures de contrôle**
- ✓ Classes & objets
- ✓ Héritage, polymorphisme & interfaces
- ✓ Exceptions
- ✓ Autres éléments de C#

Structures de contrôle

Définition

- Une structure de contrôle est un élément du programme qui change le comportement par défaut de l'unité d'exécution (du thread). Rappelons que ce comportement par défaut est d'exécuter les instructions les unes à la suite des autres.
- Trois familles de structures de contrôles:
 - Les *conditions*, qui exécutent (ou pas) un bloc de code qu'à une certaine condition, portant généralement sur les états de variables et d'objets.
 - Les *boucles*, qui exécutent en boucle un bloc d'instructions. Le programmeur a le choix entre terminer de boucler après un certain nombre d'itérations, ou terminer de boucler à une certaine condition.
 - Les *branchements* ou *sauts*, qui permettent de rediriger directement vers une instruction particulière l'unité d'exécution.

Structures de contrôle

Les conditions

- Trois types de condition: if/else, switch, l'opérateur ternaire ?:

```
if( b == true )                  // si b vaut true alors...
if( i >= 4 && i<= 6)          // si i dans l'intervalle fermé [4, 6]
    alors...
if( i < 4 || i > 6)           // si i hors de l'intervalle fermé
    [4, 6] alors...
...
// s="bonjour" si i strictement inférieur à j, sinon s="hello"
string s = i<j ? "bonjour" : "hello";
...
switch(i)
{
    case 1:                  Console.WriteLine("i vaut 1");
        break;
    case 6:                  Console.WriteLine("i vaut 6");
        break;
    default:Console.WriteLine("i ne vaut ni 1 ni 6");
        break;
}
```

Structures de contrôle

Les boucles

LOOPS

- Quatre types de boucle: while, do/while, for, foreach

```
int i=0;int j=8;
while( i < 6 && j > 9 )
{
    i++;j--;
}
```

```
for(int i = 1; i<=6 ; i++) ...
int i = 3;int j=5;
for(; i<7&& j>1 ;i++ , j--) ...
for( int i =6 ; i<9 ; ) { ... i++; }
```

```
int i=0;int j=8;
do
{
    i++;j--;
}
while( i < 6 && j > 9)
```

```
int [] tab = {1,3,4,8,2};
// Calcul de la somme des éléments du tableau.
int Somme = 0;
foreach(int i in tab )
Somme += i;
// Somme vaut : 1+3+4+8+2 = 18
```

Structures de contrôle

Instructions conditionnelles

if / then / else

- Permet d'exécuter une instruction si une condition est vraie

```
if (a == 5)
    Console.WriteLine("a == 5");
else if (a == 10)
    Console.WriteLine("a == 10");
else
    Console.WriteLine("a != 5 && a != 10");
```

Structures de contrôle

Instructions conditionnelles

if / then / else

- Une instruction peut être remplacée par un bloc

```
if (a == 5 || a == 10)
{
    Console.WriteLine("a == 5...");
    Console.WriteLine("ou a == 10");
}
```

Structures de contrôle

Instructions conditionnelles

Opérateur ternaire

- Exemple :

```
a = (b > 0 ? 1 : 2);
```

- Équivalent à :

```
if (b > 0)
    a = 1;
else
    a = 2;
```

Structures de contrôle

Instructions conditionnelles

switch

- Exécute des instructions en fonction de la valeur d'une variable

```
switch(a)
{
    case 1:
        Console.WriteLine("a == 1");
        break;
    case 2:
        Console.WriteLine("a == 2");
        break;
    default:
        Console.WriteLine("a != 1 && a != 2");
        break;
}
```

Structures de contrôle

Instructions itératives

- Permettent d'exécuter une instruction plusieurs fois
 - for
 - foreach
 - while
 - do / while

Structures de contrôle

Instructions itératives

Boucle for

- Exemple :

```
// Affiche les nombres de 1 à 10
for (int i = 1; i <= 10; i++)
{
    Console.WriteLine(i);
}
```

Structures de contrôle

Instructions itératives

Boucle foreach

- Permet d'itérer dans un ensemble de données
- Exemple :

```
// Affiche les nombres de 1 à 10
int[] array = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
foreach (int i in array)
{
    Console.WriteLine(i);
}
```

Structures de contrôle

Instructions itératives

Boucle while

- Exemple :

```
// Affiche les nombres de 1 à 10
int a = 1;
while (a <= 10)
{
    Console.WriteLine(i);
    i++;
}
```

Structures de contrôle

Instructions itératives

Boucle do / while

- Exemple :

```
// Affiche les nombres de 1 à 10
int i = 1;
do
{
    Console.WriteLine(i);
    i++;
} while (i <= 10);
```

- Contrairement à while, le bloc do / while est exécuté au moins 1 fois

Création du projet console

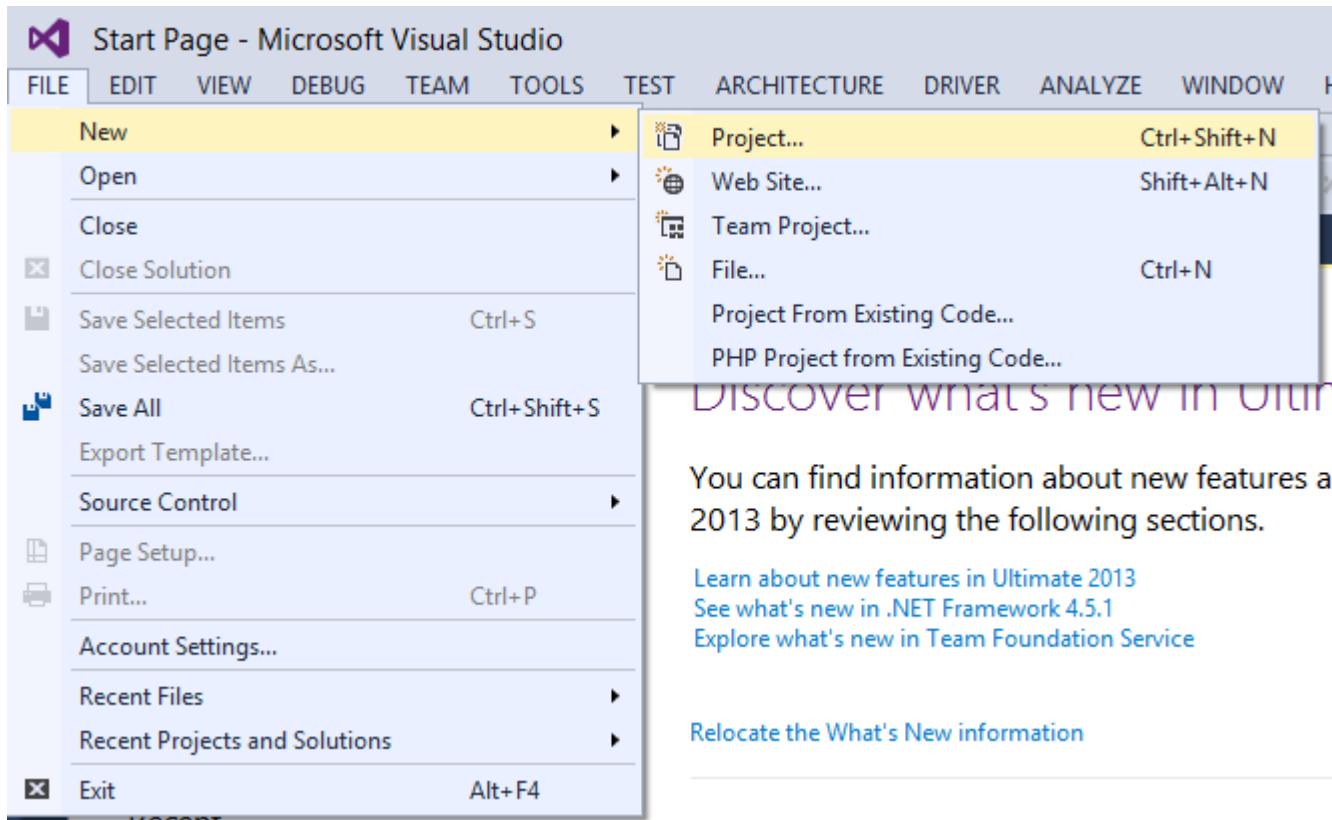
Démarrer Visual Studio

The screenshot shows the Microsoft Visual Studio 2013 Start Page. At the top, there's a navigation bar with FILE, EDIT, VIEW, DEBUG, TEAM, TOOLS, TEST, ARCHITECTURE, DRIVER, ANALYZE, WINDOW, and HELP. Below the navigation bar is a toolbar with icons for Attach, Suspend, Thread, and Stack Frame. The main content area has several sections:

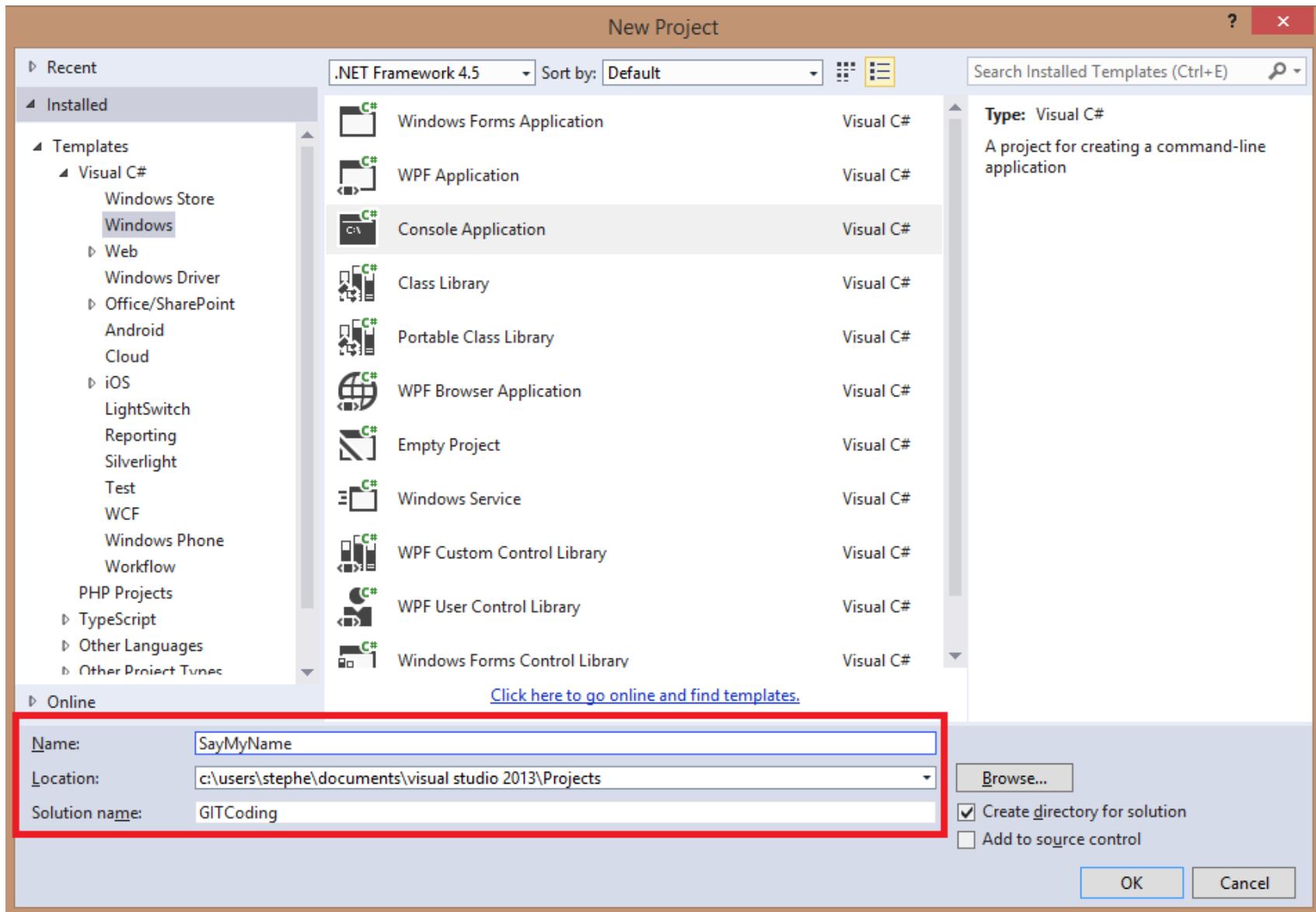
- Ultimate 2013**: A section about the new features in Visual Studio Ultimate 2013, including links to Start, New Project..., Open Project..., and Open from Source Control... . It also includes a "Discover what's new in Ultimate 2013" section with links to Learn about new features in Ultimate 2013, See what's new in .NET Framework 4.5.1, and Explore what's new in Team Foundation Service.
- Product Videos**: A section showing five video thumbnails with titles: "Intro to Windows Azure with Scott Hanselman" (5:46), "Testing Windows Store Applications" (3:00), "Go To Code From Production" (2:51), "Peek Definition" (1:48), and "Managed Memory Analysis".
- Announcements**: A section with three items:
 - Visual Studio 2013.1 (Update 1) is available** (dimanche, décembre 22, 2013). Description: Read Brian Harry's note on the release of the first Visual Studio 2013 Update.
 - Rename Refactor Helper for Visual Studio 2013** (mardi, décembre 3, 2013). Description: Check out this new DevLabs prototype, a "rename" refactoring tool for Visual C++.
 - MVPs Explore Visual Studio 2013** (mardi, décembre 3, 2013). Description: Read MVP posts diving into the features of Visual Studio 2013.
- What's new on Microsoft Platforms**: A list of platforms with icons:
 - Windows
 - Windows Azure
 - ASP.NET vNext and Web
 - Windows Phone
 - Microsoft Office
 - SharePoint Development

On the right side, there are two panes: **Solution Explorer** and **Properties**. The Solution Explorer pane is mostly empty. The Properties pane shows a list of properties for a selected item, with tabs for Solution Explorer, Team Explorer, and Class View. The bottom of the screen shows the status bar with Ln 1, Col 1, Ch 1, and INS.

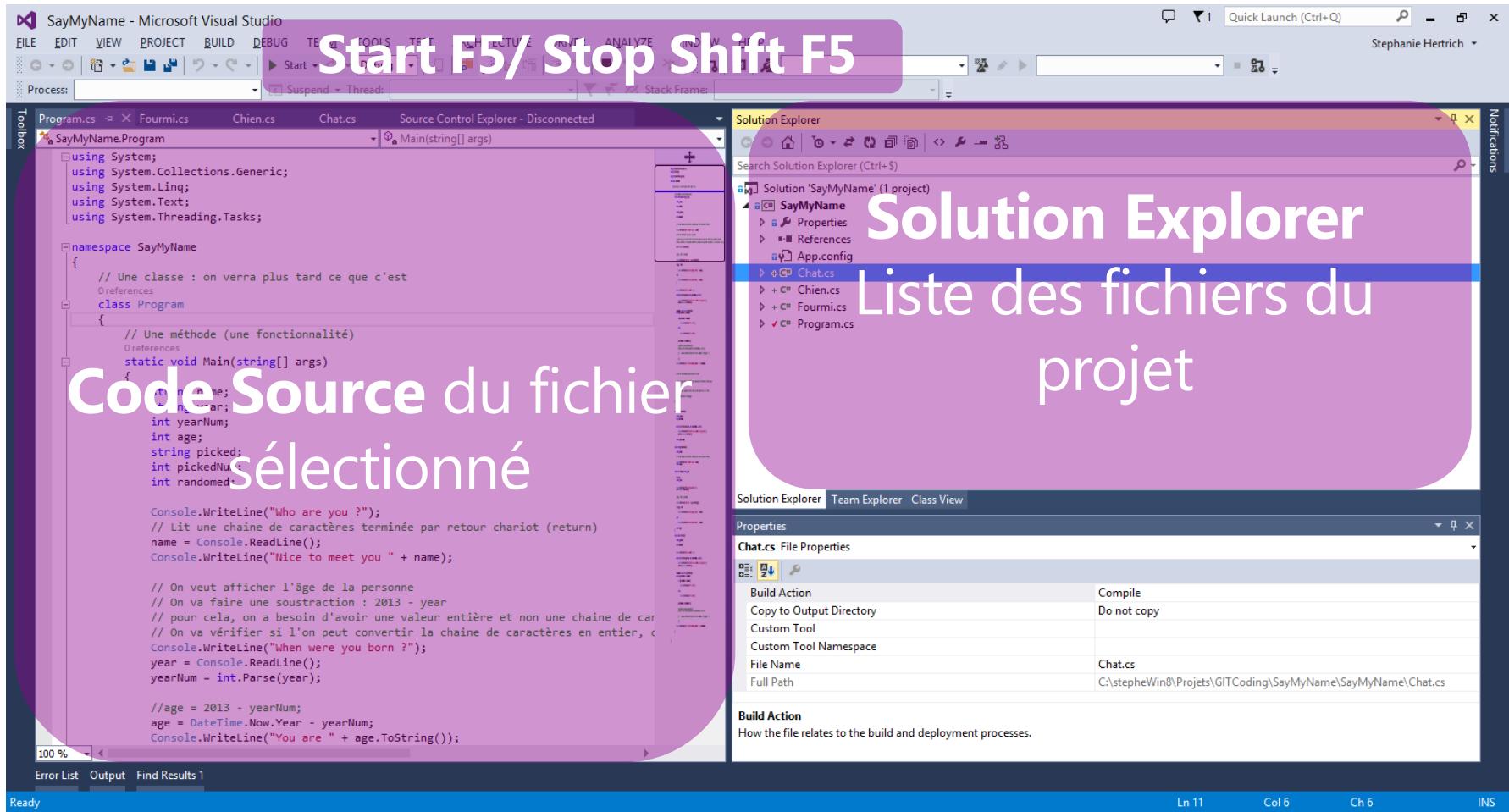
Création du projet console 1/2



Création du projet console 2/2



Présentation de l'outil : la base

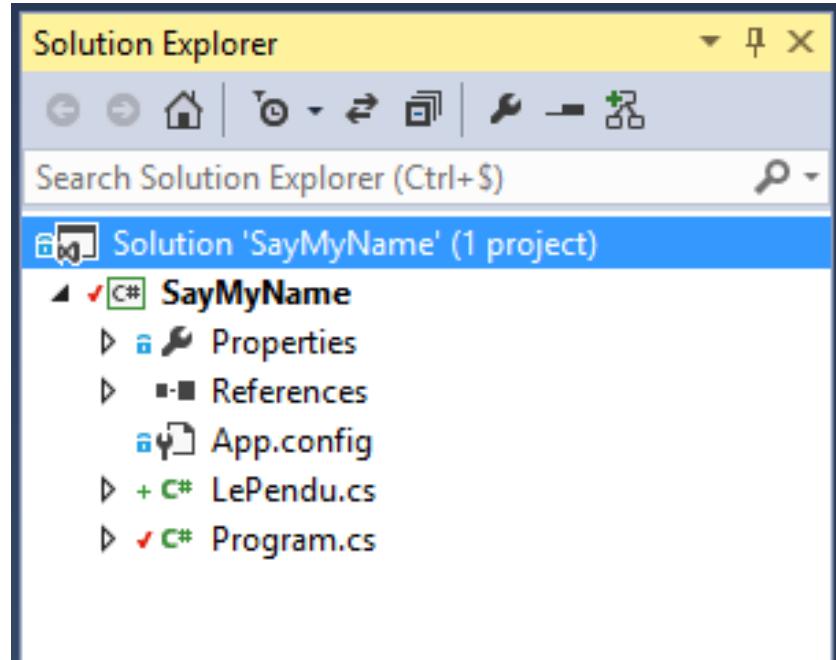


Au secours y'a plein de menus/boutons !

Anatomie d'une solution Visual Studio

Solution

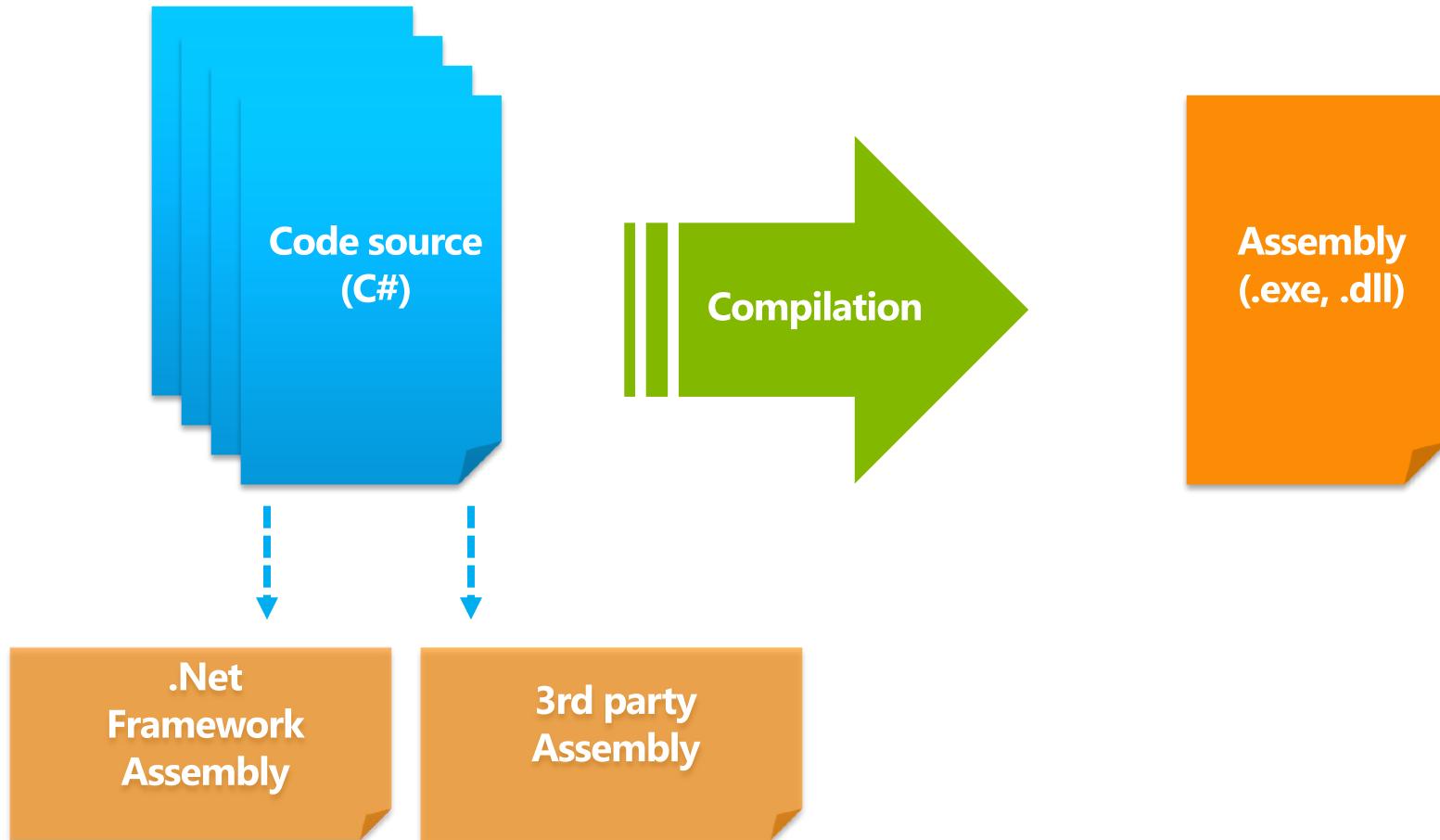
- Projet 1
 - Fichier1.cs
 - Fichier2.cs
 - ...
- Projet 2
 - Fichier1.cs
 - Fichier2.cs
 - ...



Dans notre projet console il y a...

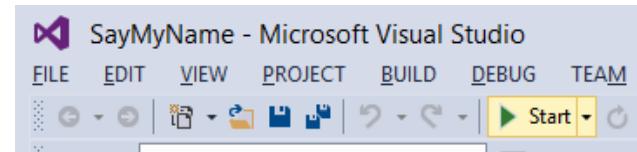
- SayMyName.sln *Solution*
- SayMyName.csproj *Projet*
 - Program.cs *Fichier source C#*
- Un fichier source « Program.cs » avec ce qu'il faut pour créer une application console vide
- D'autres éléments dont on ne se préoccupera pas pour l'instant

Du code source à l'application



La compilation, l'exécution 1/2

- Compilation : Ctrl + Shift + B
- Compilation + Exécution : Start F5 / Stop Shift-F5



Que constatez-vous à l'exécution ?

-> La fenêtre apparaît et disparaît immédiatement

La compilation, l'exécution 2/2

- La compilation du code source a généré un fichier exécutable (.exe)

Il se trouve sur votre disque, dans un sous-répertoire de votre Solution/Projet

Trouvez-le sur le disque et exécutez-le

Le « main » (*fichier program.cs*)

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

namespace SayMyName
{
    // Une classe : on verra plus tard ce que c'est
    class Program
    {
        // Une méthode (une fonctionnalité)
        static void Main(string[] args)
        {
            ENTREZ VOTRE CODE ICI
        }
    }
}
```

C# = Langage orienté objet

- La suite d'instructions qui permet de réaliser une fonctionnalité est regroupée dans une « fonction » (on dit « **méthode** » en OO).
→ Cela permet de réutiliser facilement une fonctionnalité dans une app en invoquant le nom de la méthode

Ex:

- *Afficher du texte à l'écran suivi d'un retour chariot : « **Writeline** »*
- *Lire un texte tapé au clavier « **ReadLine** »*

- Les méthodes qui se réfèrent au même « acteur » sont regroupées dans une classe

*Ex : classe « **Console** » pour toutes les fonctionnalités de la fenêtre d'exécution de l'application console:*

Console.WriteLine, Console.ReadLine

Utilisation des méthodes de la classe console

`Console.ReadKey();`

A screenshot of a code editor showing a C# program. The code is as follows:

```
namespace SayMyName1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine();
        }
    }
}
```

The cursor is positioned at the end of the first line of the Main() method. A tooltip is displayed over the `Console.WriteLine()` call, containing the following information:

▲ 11 of 19 ▼ void Console.WriteLine(string value)
Writes the specified string value, followed by the current line terminator, to the standard output stream.
value: The value to write.

// Les paramètres

`Console.WriteLine("Who are you ?");`
`Console.ReadLine();`

Les variables

```
// Lit une chaîne de caractères terminée par  
// retour chariot (return)  
string name = Console.ReadLine();
```

```
// A vous ! Affichez "Nice to meet you xxx"  
Console.WriteLine("Nice to meet you ");  
Console.WriteLine(name);
```

```
// ou  
Console.WriteLine("Nice to meet you " + name);
```

```
// ou  
// ...
```

Les types de variables

```
// On veut afficher l'âge de la personne en le déduisant à partir  
de son année de naissance  
Console.WriteLine("When were you born ?");  
// A vous : Récupérez l'année de naissance saisie dans une variable  
string year = Console.ReadLine();  
  
// On va faire une soustraction : 2013 - year  
// pour cela, on a besoin d'avoir une valeur entière et non une  
chaine de caractères (texte)  
int yearNum = int.Parse(year);  
  
// Essayez de saisir une valeur non numérique...quel est le résultat  
? On verra tout à l'heure comment résoudre ce problème  
  
//int age = 2013 - yearNum;  
// Mieux :  
int age = DateTime.Now.Year - yearNum;  
// A vous : Affichez l'âge de la personne  
Console.WriteLine("You are " + age.ToString());
```

If, else

```
// Afficher un message différent selon que l'on soit jeune
// ou vieux
// (à vous de décider l'âge à partir duquel on est vieux !)

if(age < 40)
{
    // A vous : petit message pour les jeunes
    // mentionnant le nom de la personne
    Console.WriteLine("You are young, lucky " + name);
}
else
{
    // A vous : petit message pour les vieux,
    // mentionnant le nom de la personne
    Console.WriteLine("You are experienced, " + name);
}
```

Le mode pas à pas

- F9 : Ajouter/Supprimer un point d'arrêt (Breakpoint)
- F10 : Step over l'instruction en cours
- F11 : Step in l'instruction en cours
- Quickwatch

Quizz

Parmi les propositions ci-dessous, laquelle permet la lecture et l'écriture à partir de la console?

- a. System.Array
- b. System.Output
- c. System.ReadLine
- d. System.Console

Quizz

La CLR est l'équivalent de _____.

1. a. Java Virtual Machine
2. b. Common Language Runtime
3. c. Common Type System
4. d. Common Language Specification

Quizz

..... Permet de traduire le code MSIL en code natif

a. JIT

b. CLS

c. IL

d. CTS

Quizz

- Lequel des éléments suivants est une autre façon de réécrire l'extrait de code donné ci-dessous?

```
int a = 1, b = 2, c = 0;  
if (a < b) c = a;
```

A.

```
int a = 1, b = 2, c = 0;  
c = a < b ? a : 0;
```

B.

```
int a = 1, b = 2, c = 0;  
a < b ? c = a : c = 0;
```

C.

```
int a = 1, b = 2, c = 0;  
a < b ? c = a : c = 0 ? 0 : 0;
```

D.

```
int a = 1, b = 2, c = 0;  
a < b ? return (c) : return (0);
```

E.

```
int a = 1, b = 2, c = 0;  
c = a < b : a ? 0;
```

Le code complet à ce stade

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

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Who are you ?");
            string name = Console.ReadLine();
            Console.WriteLine("Nice to meet you " + name);

            Console.WriteLine("Entrez votre année de naissance:");
            string annee = Console.ReadLine();
            int anneeInt = int.Parse(annee);
            int age = DateTime.Now.Year - anneeInt;
            Console.WriteLine("You are " + age);

            if (age < 40)
            {
                // A vous : petit message pour les plus jeunes mentionnant le nom de la personne
                Console.WriteLine("You are young, lucky " + name);
            }
            else
            {
                // A vous : petit message pour les plus agés, mentionnant le nom de la personne
                Console.WriteLine("You are experienced, " + name);
            }
            Console.ReadKey();
        }
    }
}
```

C# Exercice

Écrire un programme qui saisit une valeur n et qui affiche le carré suivant ($n = 5$ dans l'exemple) :

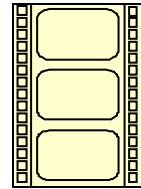
```
using System;

namespace Carré
{
    class MainClass
    {
        public static void Main (string[] args)
        {
            Console.Write ("Saisir n : ");
            int n = int.Parse(Console.ReadLine());

            for (int i = 1 ; i <= n ; i++)
            {
                for (int j = 1 ; j <= n ; j++)
                    Console.Write("X ");
                Console.WriteLine();
            }
        }
    }
}
```

A 5x5 grid of 'X' characters, representing a square matrix. Each row and column contains exactly five 'X' characters.

X	X	X	X	X
X	X	X	X	X
X	X	X	X	X
X	X	X	X	X
X	X	X	X	X

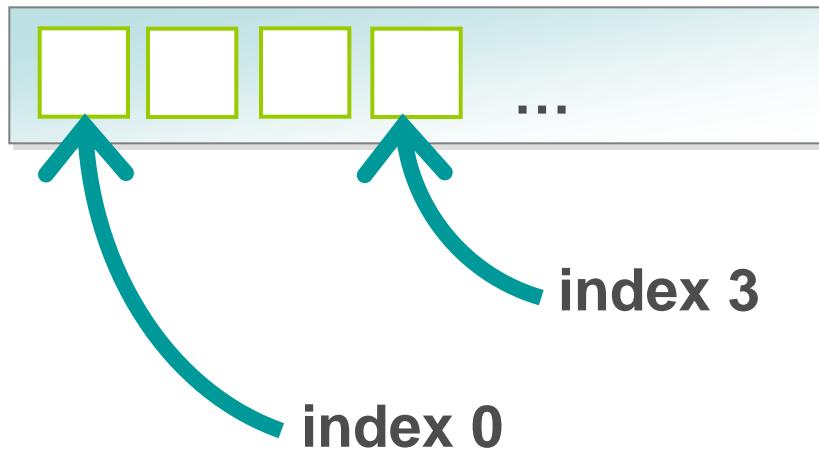


Plan

- ✓ Eléments de base
- ✓ Structures de contrôle
- ✓ **Tableaux**
- ✓ Classes & objets
- ✓ Héritage, polymorphisme & interfaces
- ✓ Autres éléments de C#

Les tableaux

- Suite d'éléments du même type
- Accessibles par un index



- Type référence

A photograph of a blackboard titled "PROBLEM" at the top right. The board contains a table with data. The columns are labeled "GROUP", "WT Kg", "COST", and "LOAD". The "COST" column has sub-labels "MAT", "LAB", and "TOTAL". The "LOAD" column has sub-labels "AWK" and "COLLAPSE". The data rows are as follows:

GROUP	WT Kg	MAT	LAB	TOTAL	AWK	COLLAPSE
14	1.90	135.5	554	679.5	✓	1.86
19	349	259.8	282	541.8	✗	0.31
24	3.12	202.2	332	534.2	✓	2.28
15	2.17	152.3	370	522.3	✓	2.81
27	2.22	148.2	266	414.2	✓	2.26
23	1.63	113.2	292	405.2	✓	1.44
26	2.35	165.4	174	399.4	✓	2.26
17	279	206.6	158	394.6	✓	3.64

Les tableaux

Tableaux à 1 dimension

■ Déclaration :

```
// Crée un tableau de 10 entiers
int[] array = new int[10];
```

■ Utilisation :

```
for (int i = 0; i < array.Length; i++)
{
    array[i] = i * i;
}

// Affiche 25
Console.WriteLine(array[5]);
```

Les tableaux

Tableaux à 2 dimensions

■ Déclaration :

```
// Crée un tableau de 10x10 entiers  
int[,] array = new int[10,10];
```

■ Utilisation :

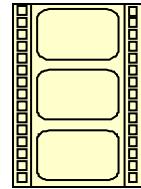
```
for (int i = 0; i < array.GetLength(0); i++)  
    for (int j = 0; j < array.GetLength(1); j++)  
    {  
        array[i, j] = i * j;  
    }
```

Les tableaux

- Les types de tableaux sont tous des types référence. En fait chaque type de tableau est une classe qui hérite de la classe abstraite `System.Array`.
- C# oblige tous les éléments d'un tableau à avoir le même type. Cette contrainte peut être facilement contournée. Il suffit de spécifier que les éléments sont des références vers une classe de base, pour qu'en fait, chaque élément puisse être une référence vers un objet de n'importe quelle classe dérivée.

C# Exercice

- Lire la dimension N d'un tableau T du type int (dimension maximale: 20 éléments),
- Remplir le tableau par des valeurs entrées au clavier et affiche le tableau.
- Copier ensuite toutes les composantes strictement positives dans un deuxième tableau TPOS et toutes les valeurs strictement négatives dans un troisième tableau TNEG.
- Afficher les tableaux TPOS et TNEG.



Plan

- ✓ Eléments de base
- ✓ Structures de contrôle
- ✓ Tableaux
- ✓ **Classes & objets**
- ✓ Héritage, polymorphisme & interfaces
- ✓ Exceptions
- ✓ Autres éléments de C#

Notions fondamentales de la Programmation Orientée Objet

• **Classes et Objets**

- Une classe comporte un ensemble de variables (appelés **attributs**) et de procédures/fonctions (appelés **méthodes**). La classe représente le moule de l'objet.
- Un objet est une instance créée à partir d'une classe.
- La POO permet de se rapprocher de la réalité. En effet, les variables représentent l'état de l'objet, alors que les méthodes définissent son comportement.

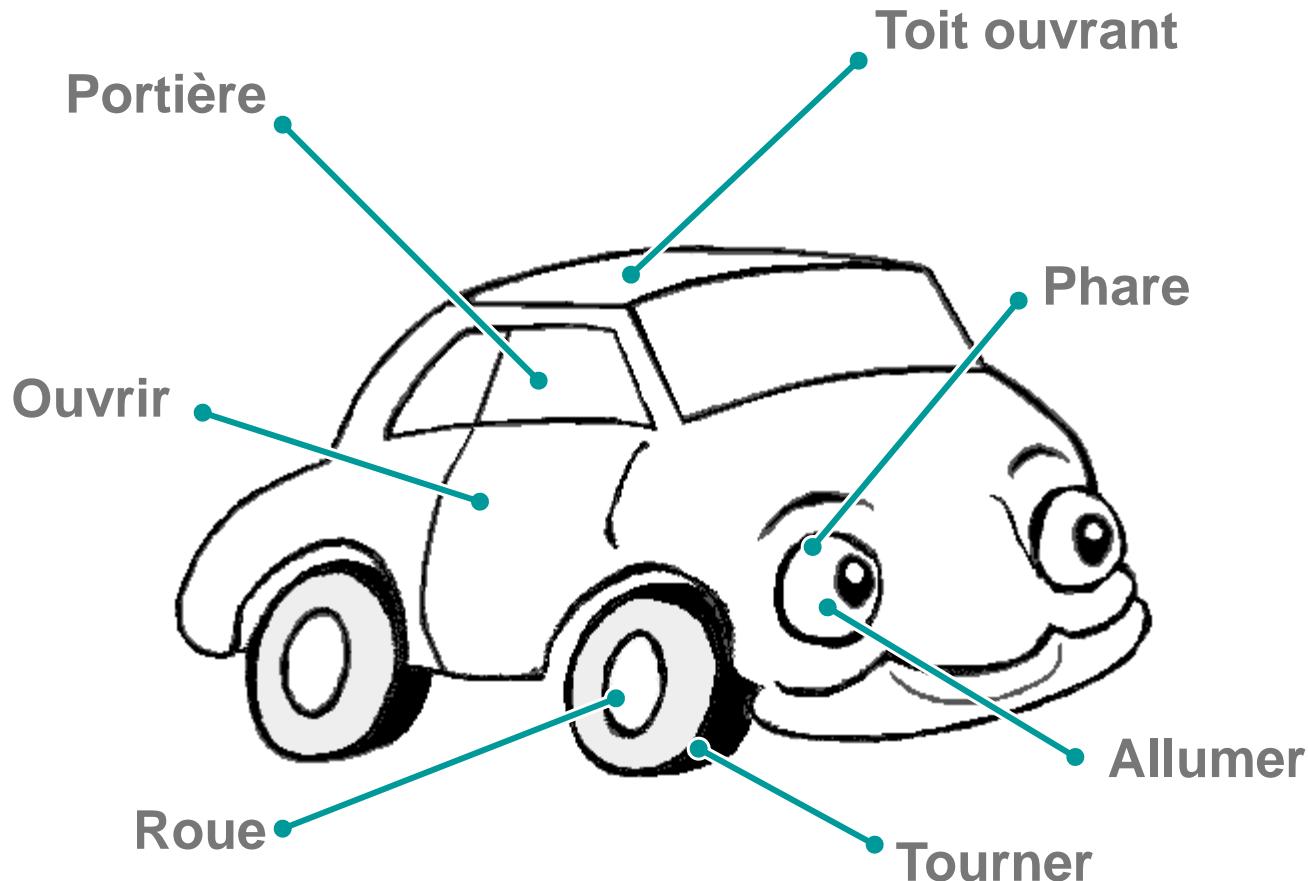
Notions fondamentales de la Programmation Orientée Objet

- **Exemple 1:**
 - Une classe: Une personne
 - Des objets: Zakaria, Ghita, Sabrine...
 - Attributs: Couleur des cheveux, Taille, Poids, Sexe...
 - Comportements: Parler, Courir, Marcher, Sourir...
- **Exemple 2:**
 - Une classe: Une voiture
 - Des objets: Ford Fiesta, Peugeot 208, Audi R8...
 - Attributs: Couleur, Type de carburant, Nombre portes...
 - Comportements: Démarrer, Accélérer, Freiner, Tourner, Stop...

Notions fondamentales de la POO

- **Classes et Objets**

Exemple : Cas d'une voiture



Notions fondamentales de la POO

- **L'encapsulation**

Mécanisme permettant de rassembler les attributs et méthodes propres à un type donné afin d'en restreindre l'accès et/ou d'en faciliter l'utilisation et la maintenance.

Notions fondamentales de la Programmation Orientée Objet

• L'encapsulation

- Dans une classe, les éléments ont 3 niveaux de visibilité : **Privé** (Private), **Protégé** (Protected) et **Public**.
- Les attributs sont toujours **PRIVES**. Pour lire ou modifier une valeur, on créer une méthode publique **Get...()** et une autre méthode publique **Set...()**.
- En C#, les classes peuvent comporter également des « propriétés ». Elles permettent d'alléger la syntaxe et ainsi finalement de faire *maVoiture.Couleur = Rouge*.

Notions fondamentales de la Programmation

Orientée Objet

- **L'encapsulation**

- Déclaration d'une classe sans propriétés

```
class Personne
{
    // Attribut
    private string _nom;

    // Getter
    public string GetNom()
    {
        return _nom;
    }

    // Setter
    public void SetNom(string nom)
    {
        _nom = nom;
    }

    // Méthode
    public void Parler()
    {
        Console.WriteLine("{0} dit bonjour.", _nom);
    }
}
```

Notions fondamentales de la Programmation Orientée Objet

• L'encapsulation

- Simplification en utilisant les propriétés

```
class Personne
{
    // Propriété
    public Nom { get; set; }

    // Méthode
    public void Parler()
    {
        Console.WriteLine("{0} dit bonjour.", Nom);
    }
}
```

Notions fondamentales de la POO

• L'encapsulation

- Sans propriétés

```
class Personne
{
    private int _age;

    public int GetAge()
    {
        return _age;
    }

    public void SetAge(int age)
    {
        _age = age;
    }
}
```

- Avec propriétés

```
class Personne
{
    private int _age;

    public int Age
    {
        get { return _age; }
        set
        {
            if(value > 0)
            {
                age = value;
            }
        }
    }
}
```

Notions fondamentales de la POO

• L'encapsulation

L'encapsulation regroupe :

- **Modificateurs d'accès**

- private, public, etc.

- **Accessors**

- Permettent de récupérer la valeur d'un champ
 - GetData() { ... }

- **Mutators**

- Permettent de définir la valeur d'un champ
 - SetData() { ... }

Notions fondamentales de la POO

Encapsulation et niveau de visibilité

	public	internal protected	internal	protected	private
Méthodes de la classe A	OK	OK	OK	OK	OK
Méthodes d'une classe dérivée de A, dans le même assemblage	OK	OK	OK	OK	
Méthodes d'une autre classe dans le même assemblage	OK	OK	OK		
Méthodes d'une classe dérivée de A dans un assemblage différent	OK	OK			
Méthodes d'une autre classe dans un assemblage différent	OK				

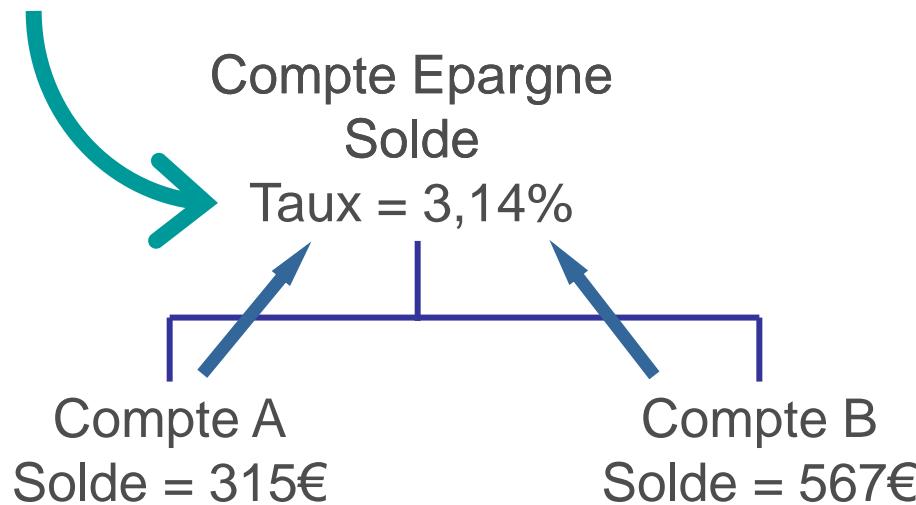
Notions fondamentales de la POO

- **Données de l'objet, données statiques et méthodes statiques**
 - Chaque objet possède des **données propres**
 - **Données statiques**
 - Propres à toutes les instances de la classe
 - **Méthodes statiques**
 - Accessibles par toutes les instances
 - Peuvent utiliser des données statiques

Notions fondamentales de la POO

- **Données de l'objet, données statiques et méthodes statiques**

Exemple: Cas d'un compte épargne



Notions fondamentales de la POO

Modificateurs d'accessibilité

La modification de l'accessibilité permet de définir le comportement et la porté d'un élément avec son environnement (dans la classe ou dans les classes dérivées)

- Tous les éléments sont définis avec des modificateurs d'accessibilités.
- Si aucun modificateur n'est précisé l'élément est défini comme '**privé**'.
- Les modificateurs disponibles sont:
 - **Virtual**, // Indique que l'élément surcharge une définition de la classe mère.
 - **Override**, // Indique que l'élément peut être surchargé dans une classe fille.
 - **Abstract**, // Seul le prototype de l'élément est définie il doit être surchargé dans la classe fille.
 - **Sealed**, // Interdit la surcharge de l'élément (par défaut pour les méthodes, mais peut être utilisé dans la surcharge d'une méthode virtual).
 - **Public**, // L'élément est accessible de partout.
 - **Private**, // L'élément n'est accessible qu'en interne du type dans lequel il est défini (par défaut).
 - **Internal**, // Accessible uniquement depuis le même assemblage que l'élément.
 - **Protected** // Accessible uniquement au type qui le définit mais aussi aux types dérivés.
- Certain modificateur ne s'utilise pas avec tout les types.

```
using System;
class A {
    public virtual void Test() {
        Console.WriteLine("A.Test");
    }
}

class B : A {
    public override void Test() {
        Console.WriteLine("B.Test");
    }
}

class Program {
    static void Main() {
        A ref1 = new A(); ref1.Test();
        A ref2 = new B(); ref2.Test();
    }
}
```

Output A.Test B.Test

```
abstract class Test {  
    public int _a;  
    public abstract void A(); }
```

```
class Example1 : Test {  
    public override void A() {  
        Console.WriteLine("Example1.A");  
        base._a++; } }
```

```
class Example2 : Test {  
    public override void A() {  
        Console.WriteLine("Example2.A"); base._a--; } }
```

```
class Program {  
    static void Main() {  
        t1 = new Example1();  
        test1.A();  
        Test test2 = new Example2();  
        test2.A(); } }
```

Example1.A
Example2.A

Notions fondamentales de la POO

Les constructeurs

- Une méthode est automatiquement appelée lorsque l'objet est construit. On appelle ce type de méthode un *constructeur* (*ctor* en abrégé).
- En C#, syntaxiquement, une méthode constructeur porte le nom de la classe, et ne retourne rien (même pas le type void).
- Pour chaque classe il peut y avoir :
 - Aucun constructeur : dans ce cas le compilateur fournit automatiquement un *constructeur par défaut*, qui n'accepte aucun argument.
 - Un seul constructeur : dans ce cas, ce sera toujours lui qui sera appelé. Le compilateur ne fournit pas de constructeur par défaut.

Notions fondamentales de la POO

Les constructeurs

- Plusieurs constructeurs : dans ce cas ils diffèrent selon leurs signatures (i.e nombre et type des arguments). Le compilateur ne fournit pas de constructeur par défaut.
- Lorsqu'une méthode constructeur retourne, il est impératif que tous les champs de type valeur de la classe aient été initialisés.
- Un constructeur admet un niveau de visibilité.

Notions fondamentales de la POO

Le destructeur

- On nomme *destructeur* la méthode de la classe qui est appelée par le *ramasse-miettes* juste avant qu'il désalloue la zone mémoire du tas occupée par l'objet.
- Il ne peut y avoir plus d'un destructeur dans une classe, et si le développeur n'en écrit pas, le compilateur crée un destructeur par défaut.
- Un destructeur est une méthode non statique avec le même nom que sa classe, et commençant par le caractère tilde ‘~’.
- Syntaxiquement, un destructeur n'admet pas d'argument et ne retourne aucun type, même pas le type void.
- En interne, le compilateur C# fait en sorte qu'un destructeur est la réécriture de la méthode `Finalize()` de la classe `Object`.
- Un destructeur n'admet pas de niveau de visibilité.

Notions fondamentales de la POO

Exemple 1: Construction et destruction des objets

```
public class Article
{
    private int Prix;
    public Article(int Prix) { this.Prix = Prix; }      // ctor 1
    public Article(double Prix) { this.Prix = (int) Prix; } // ctor 2
    public Article() { Prix = 0; }                         // ctor 3
    ~Article() { /* rien à désallouer! */ } // dtor
}

class Prog
{
    static void Main(string[] args)
    {
        Article A = new Article(); // Appelle le ctor 3.
        Article B = new Article(6); // Appelle le ctor 1.
        Article C = new Article(6.3); // Appelle le ctor 2.
    }
    // le ramasse miettes appelle le destructeur sur les objets A B et C
    // avant la terminaison du programme
}
```

Notions fondamentales de la POO

Exemple 2: Construction et destruction des objets

```
class Personne
{
    // Constructeur par défaut
    public Personne()
    {
        Nom = « Dupont »;
        Age = 17;
    }

    // Constructeur spécialisé (avec un ou plusieurs paramètres)
    public Personne(string nom, int age)
    {
        Nom = nom;
        Age = age;
    }

    // Un seul et unique destructeur
    public ~Personne()
    {
        Console.WriteLine(«{0} est détruit.», Nom);
    }
}
```

Notions fondamentales de la POO

Instanciation d'un objet / Crédation d'une instance de classe

// Utilisation du constructeur par défaut

```
Personne p1 = new Personne();
```

// Utilisation du constructeur spécialisé

```
Personne p2 = new Personne("Jean");
```

```
p1.Nom = "Pierre";
```

```
p1._nom = "Paul"; // Attribut privé
```

```
p1.Parler();
```

```
p1.Age = 21;
```

Notions fondamentales de la POO

Héritage

- En POO, il est possible de créer des classes dérivant d'une autre. On appelle ça **l'héritage**.
- On appelle **classe mère** la classe de base. On appelle **classes filles** celles qui dérivent.
- Les classes filles possèdent les attributs protégés et publics de la classe mère et apportent chacune leur propre lot d'attributs et de méthodes à leur tour.
- En C#, une classe « fille » ne peut dériver que d'une seule classe « mère ».
- Une classe fille peut redéfinir une méthode qui existe déjà dans la classe mère.

Notions fondamentales de la POO

Héritage

- Exemple 1:
 - Classe mère: Vehicule
 - Classes filles: Voiture, Camion...
- Exemple 2:
 - Classe mère: Compte bancaire
 - Classes filles: Compte courant, Livret épargne...
- Exemple 3:
 - Classe mère: Forme
 - Classes filles: Rectangle, Cercle, Carré...

Notions fondamentales de la POO

Héritage

```
class Vehicule
{
    protected int _nombreCV;
    private string _couleur;
}

class Camion : Vehicule
{
    private double _longueurRemorque;

    public Voiture(int nombreCV, double longueurRemorque)
    {
        _longueurRemorque = longueurRemorque;
        _nombreCV = nombreCV; // Protégé
        _couleur = « Rouge »; // Privé
    }
}
```

Notions fondamentales de la POO

Surchage de méthodes

La surcharge de méthode est utilisée pour modifier le comportement par défaut d'une méthode définie dans une classe mère.

- Seule les méthodes ‘virtual’ des classes mère peuvent être Surchargées.
- La méthode de surcharge doit avoir le même nom et prototype que la méthode mère.
- La méthode de surcharge doit être déclarée comme ‘override’.
- Les éléments statiques ne peuvent pas être surchargés.

Notions fondamentales de la POO

Classe et méthode abstraite

- Une classe peut être définie comme **abstraite**, c'est-à-dire qu'il n'est pas possible de l'instancier. Il faut donc instancier une classe fille.
- Une méthode abstraite, est une méthode qui existe dans une classe mais qui ne possède pas de corps/d'implémentation. Le comportement est donc défini dans la/les classe(s) fille(s).

Notions fondamentales de la POO

Interfaces

- Une interface est un contrat de propriétés et/ou de méthodes qu'une classe s'engage à respecter.
- Le nom d'une interface commence par un « I » en C#.
- Pour une interface, on crée un fichier .cs
- Contrairement à une classe, une interface ne contient pas de corps pour les méthodes. Elle définit juste les propriétés ou les méthodes que la classe doit implémenter.

Notions fondamentales de la POO

Interfaces

- Déclaration

```
interface IAnimal
{
    int Age { get; set; }

    void Crier();
    void Courir(float vitesse);
}
```

Notions fondamentales de la POO

Interfaces

- Implémentation

```
class Lievre : IAnimal
{
    public int Age { get; set; }

    public void Crier()
    {
        Console.WriteLine("Le lièvre couine");
    }

    public void Courir(float vitesse)
    {
        Console.WriteLine("Le lièvre court à {0} km/h", vitesse);
    }
}
```

Notions fondamentales de la POO

Interfaces

- Implémentation

```
class Tortue : IAnimal
{
    public int Age { get; set; }

    public void Crier()
    {
        Console.WriteLine(" La tortue cracrapète ");
    }

    public void Courir(float vitesse)
    {
        Console.WriteLine(" La tortue marche à {0} km/h",
vitesse);
    }
}
```

Notions fondamentales de la POO

Interfaces

- Utilisation

```
IAnimal a1 = new Tortue();  
a1.Age = 200;  
a1.Courir(2); // Affiche « La tortue marche à 2 km/h »
```

```
a1 = new Lievre();  
a1.Age = 5;  
a1.Crier(); // Affiche « Le lièvre couine »
```

Notions fondamentales de la POO

Interfaces

- Si une classe implémente une interface, elle DOIT respecter les propriétés et les méthodes contenues dans celle-ci.
- Dans une classe qui implémente une interface, les éléments sont forcément publics.
- Les interfaces ont plusieurs utilités:
 - Niveau d'abstraction supplémentaire: il n'est pas nécessaire de savoir si l'animal est un lion ou une tortue. L'important est de savoir qu'il a un âge, qu'il peut crier et courir.
 - Filtrage: Si une classe implémente beaucoup de méthodes, on peut utiliser les interfaces pour « filtrer » ces méthodes afin que celles qui nous intéressent soient uniquement proposés par l'auto-complétion.
- Une interface peut implémenter une ou plusieurs autres interfaces.

Notions fondamentales de la POO

Interfaces

- Interfaces les plus fréquemment utilisées

- IDisposable
- IEnumerable
- ICollection
- IList
- IValueConverter
- IEqualityComparer
- ICommand
- ...