1. Introduction
2. Créer votre premier projet C#
3. Expliquer la structure de la classe
4. L'explication de la structure du Projet
5. La note importante avec le programme C #
6. Ajouter la nouvelle classe
7. Les types de données dans C#
8. Des variables et la déclaration de variable
9. Des déclarations Branch
   1. La déclaration If-else
   2. La déclaration Switch-Case
10. La boucle dans C#
    1. La boucle for
    2. La boucle while
    3. La boucle do-while
    4. La commande break dans la boucle
    5. La commande continue dans la boucle
11. Des tableaux dans C#
    1. Tableau à une seule dimension (uni-dimensionnel)
    2. Tableau à deux dimensions
    3. Tableau d'un tableau
12. Classe, Constructeur et des instances
13. Le champ (Field)
14. La méthode (Method)
15. L'héritage dans C#
16. L'héritage et le polymorphisme en C #

1- Introduction

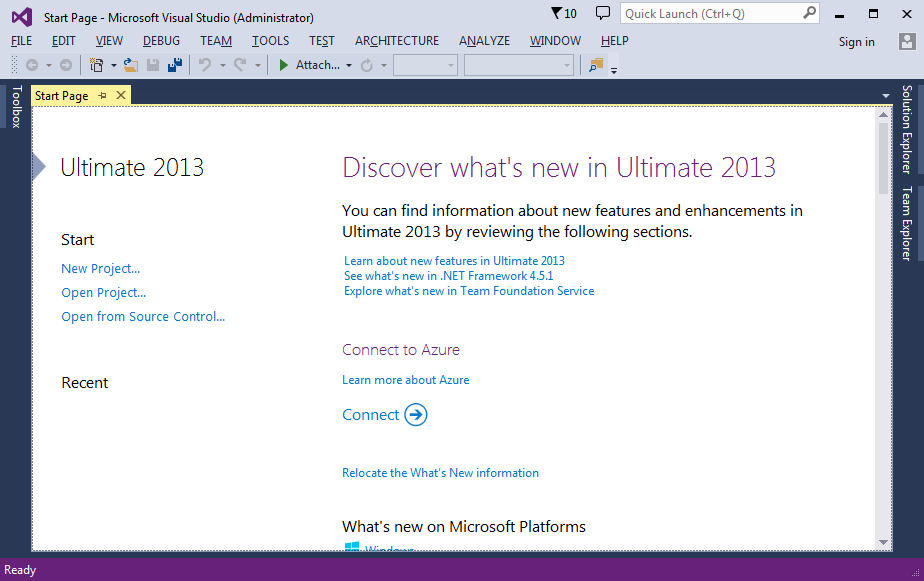
Ceci est un guide de C# pour les débutants. Pour programmer C#, vous devez installer des outils de programmation Visual Studio. Vous pouvez voir les instructions pour télécharger et installer à:

* [De quoi avez-vous besoin pour démarrer avec C#?](https://devstory.net/10331/de-quoi-avez-vous-besoin-pour-demarrer-avec-csharp)

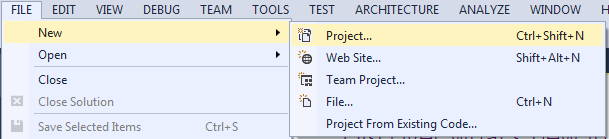
Si vous êtes nouveau sur C#, vous devriez lire ce document du début à la fin, ce qui vous donnera une vue d'ensemble avant d'apprendre d'autres documents plus détaillés.

2- Créer votre premier projet C#

C'est la première image lors que vous ouvrez Visual Studio.

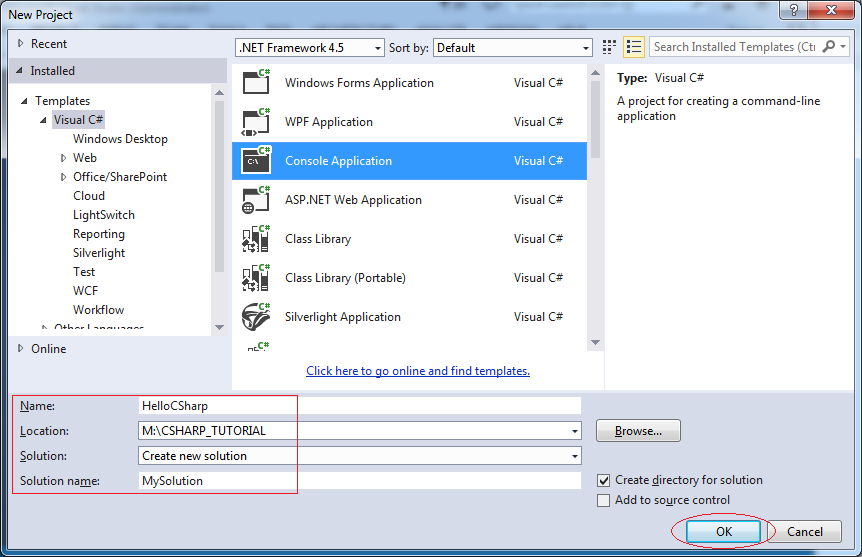


Créer un nouveau projet:

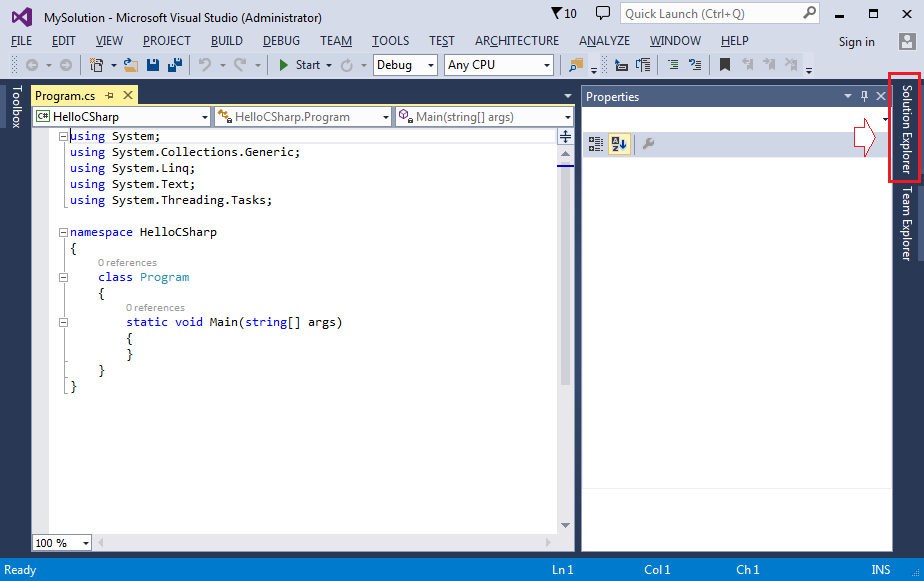


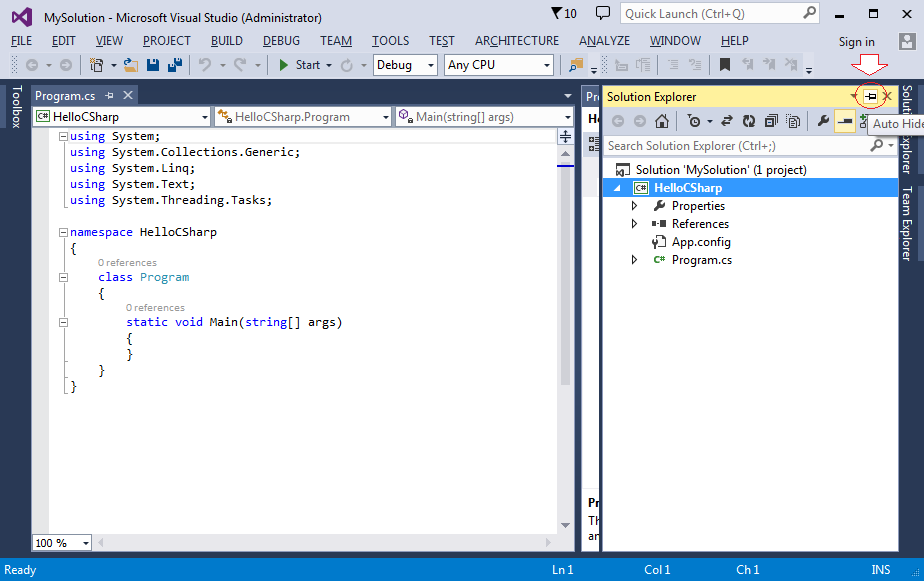
Nous créons un simple projet (L'application Console, est une application qui n'a pas de l'interface). Saisissez:

* Name: HelloCSharp
* Solution: Create new solution
* Solution Name: MySolution



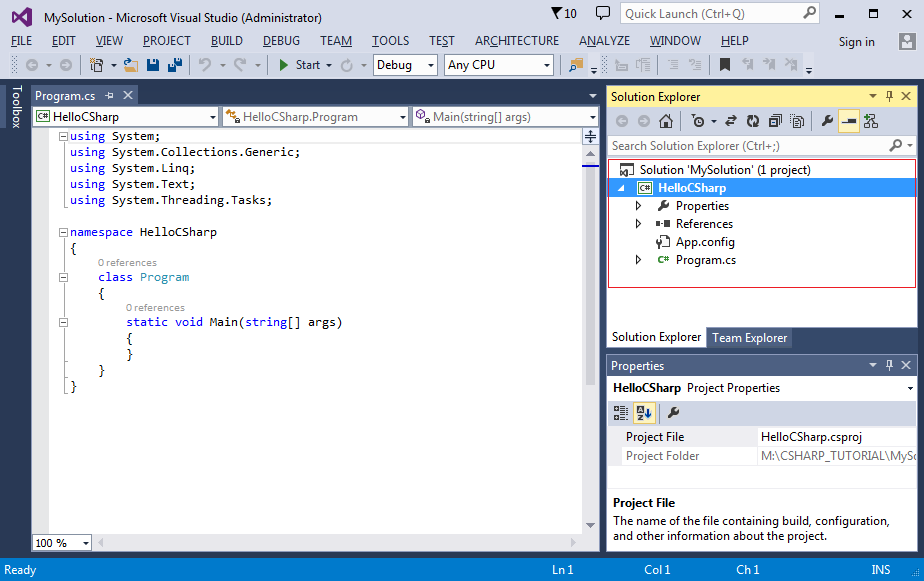
C'est une image de votre projet qui a été créée. Vous devez cliquer sur Solution Explorer pour afficher la structure de ce projet créé





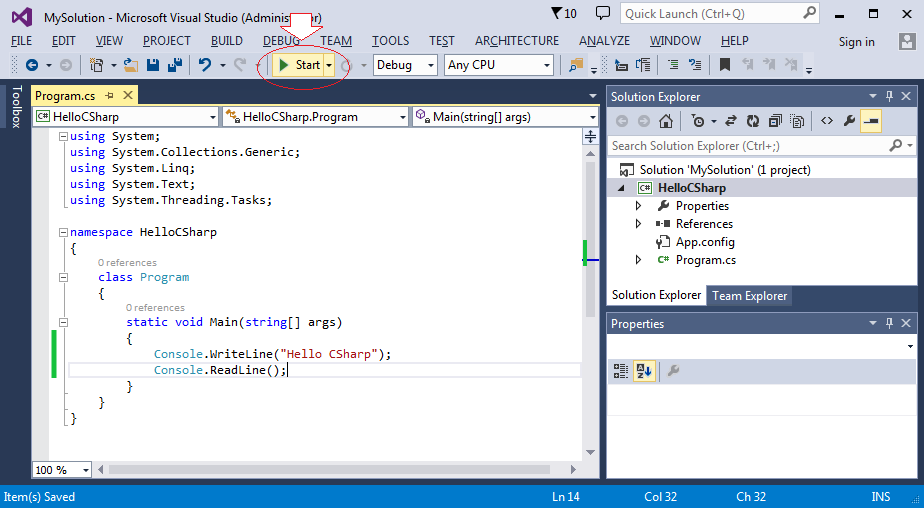
Visual Studio créé une Solution (la solution) nommée MySolution et comprend un Project nommé HelloCSharp. Et crée par défat une classe baptisée Program (correspondant au fichier *Program.cs*).

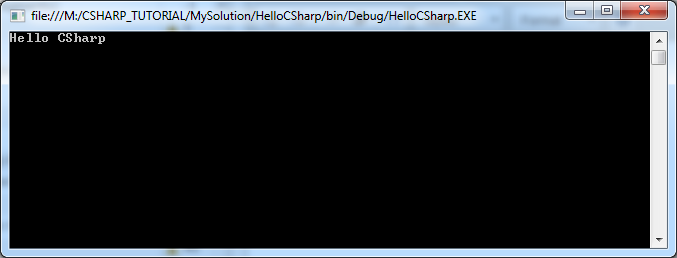
*Remarque: Une Solution (Solution) peut contenir un ou plusieurs Project.*



Éditez le code de la classe Program, alors, lors de l'exécution, il apparaîtra sur l'écran de Console une ligne de lettres "Hello CSharp", et en attendant que l'utilisateur entre tout texte avant de finir.

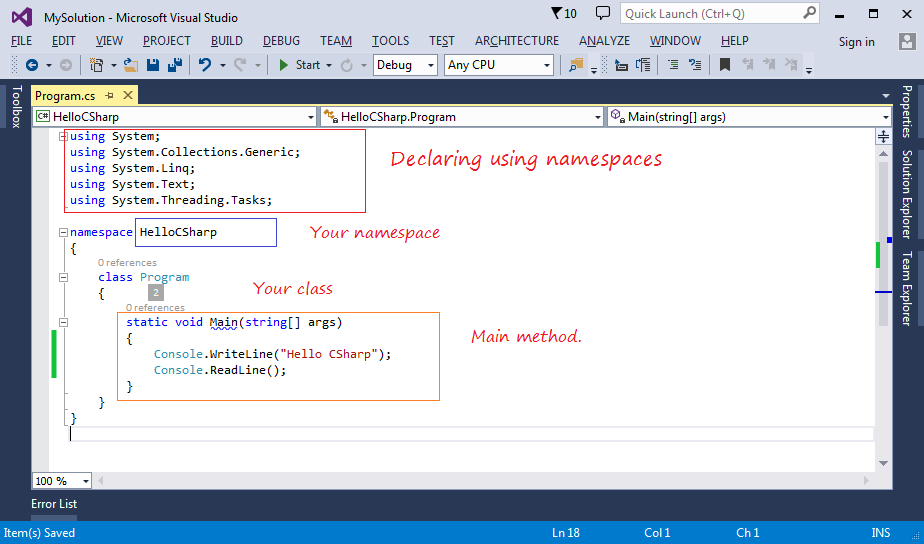
Cliquez sur Start pour exécuter la classe Program.





3- Expliquer la structure de la classe

L'illustration ci-dessous est la structure de la classe Program, qui se trouve dans l'espace de noms (namespace) HelloCSharp. Un espace de noms peut contenir une ou plusieurs classes.



Si vous souhaitez utiliser une certaine classe, vous devez déclarer utiliser cette classe ou déclarer utiliser un espace de noms (namespace) contenant cette classe.

// Déclarer d'utiliser l'espace de noms (namespace) System.

// (Cela signifie que vous pouvez utiliser toutes les classes, .. dans cet espace de noms).

**using** System;

Lors que le programme exécute, la méthode Main(string[]) sera appelée pour exécuter.

1. static est le mot-clé informé qu'il s'agit d'une méthode statique.
2. void est le mot clé informé que la méthode ne renvoie rien.
3. args est le paramètre de la méthode, un ensemble de string (chaînes) - string[].

**static** **void** **Main**(string[] args)

{

// Imprimez le texte sur la fenêtre de la Console.

Console.WriteLine("Hello CSharp");

// Attendez que l'utilisateur saisit quelque chose et appuyez sur Enter avant de terminer le programme.

Console.ReadLine();

}

Après avoir déclaré l'utilisation de l'espace (namespace) du System, vous pouvez utiliser la classe Console dans cet namespace. *WriteLine(string)* est une méthode statique de la classe Console, elle permet d'imprimer une chaîne sur l'écran.

// Déclarez l'utilisation de l'espace de noms System (System namespace).

// (Il contient la classe Console).

**using** System;

// Et vous pouvez utiliser la classe Console:

Console.WriteLine("Hello CSharp");

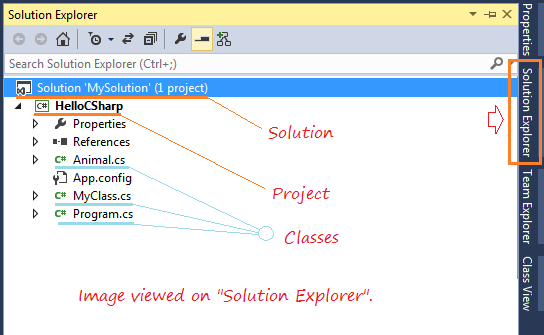
// Si vous ne voulez pas déclarer l'espace de noms System.

// Mais vous voulez utiliser la classe Console, vous devez écrire la totalité:

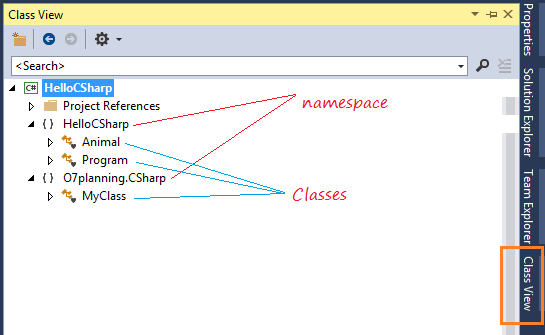
System.Console.WriteLine("Hello CSharp");

4- L'explication de la structure du Projet

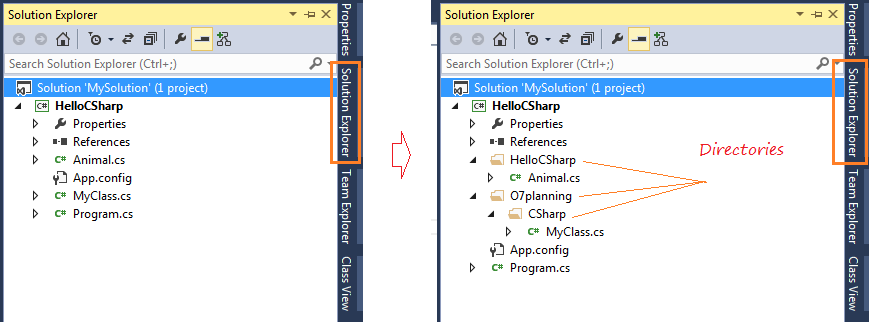
Une solution (Solution) peut comprendre plusieurs Project. Dans ces Project ils comprennent des classes (class).



Lors de la visualisation en "Class view", vous pouvez voir vos classes appartiennent à quel espace de noms (namespace).



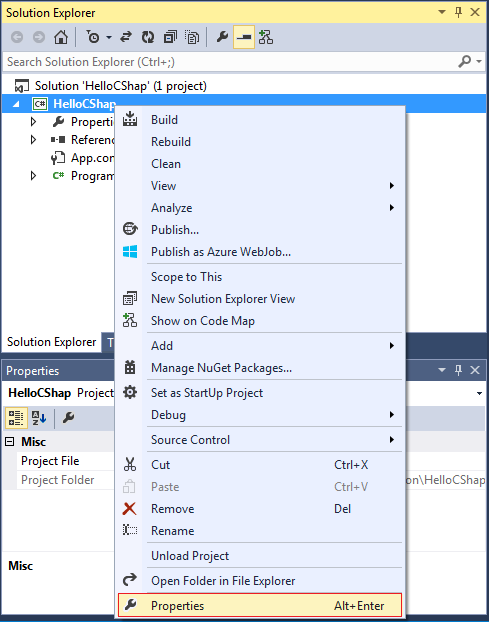
Dans CSharp, vous créez une classe Animal avec un espace nommé HelloCSharp, cette classe sera située dans le répertoire racine du projet par défaut. Vous créez une autre classe est MyClass avec l'espace de noms nommé O7planning. La classe CSharp s'est également située dans le répertoire racine du projet. Avec un grand projet avec de nombreuses classes, organisez les fichiers en tant que tels vous rendent difficile. Vous pouvez créer différents dossiers pour stocker les fichiers de classe, la règle est à votre décision, mais il est préférable de créer un dossier nommé comme le namespace. Apprenons comment organiser Java.



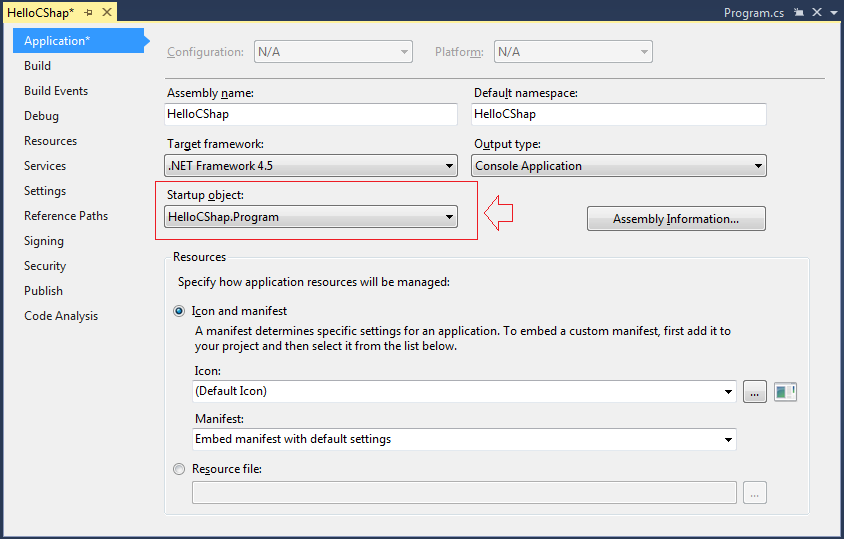
5- La note importante avec le programme C #

Dans l'application C#, vous devez déclarer explicitement une classe avec la méthode Main(string []) comme un point de départ pour exécuter votre application, ceci n'est pas obligatoire si l'entité de votre application n'a qu'une seule classe avec la méthode Main(string[]), Cependant, si vous avez 2 classes avec la méthode Main, si vous ne spécifiez pas, un message d'erreur apparaîtra dans le processus de compilation.  
  
Par conséquent, vous devez de préférence déclarer clairement la classe avec la méthode Main(string[]), vous pouvez déclarer une autre fois pour une autre classe si vous le souhaitez.

Cliquez sur le bouton droit du projet HelloCSharp, sélectionnez Properties:



Sélectionnez "Startup object" qui est une classe qui possède la méthode Main(string[]) et Save.

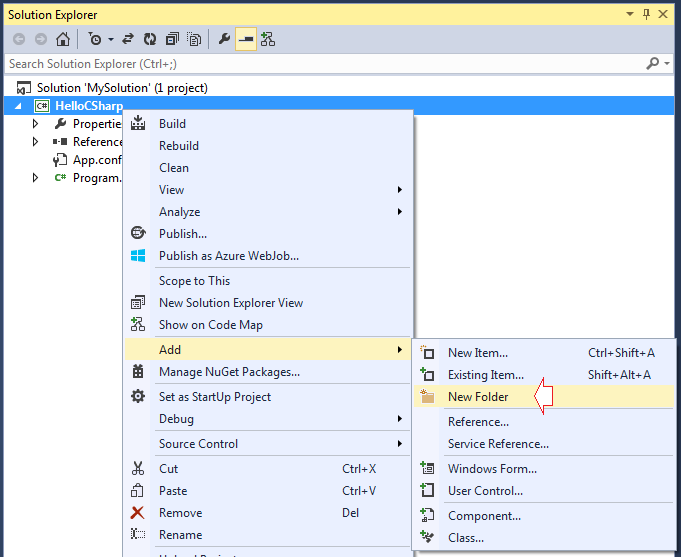


6- Ajouter la nouvelle classe

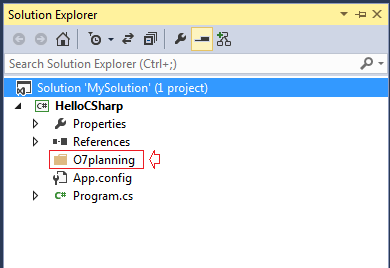
Maintenant, j'ajoute une classe MyClass avec un espace nommé O7planning.CSharp.

Dans "Solution Explorer" cliquez sur le bouton droit du projet, sélectionnez:

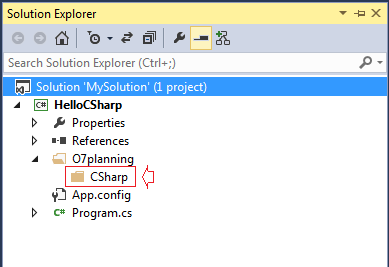
* Add/New Folder



Le nom du dossier est O7planning.

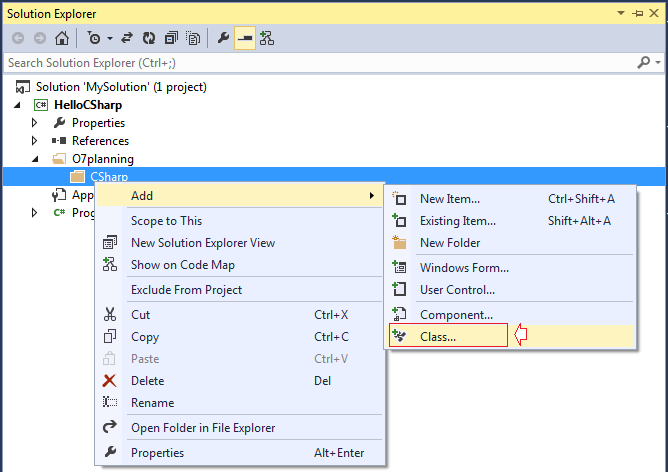


Ensuite, créez un dossier "CSharp" qui est sous- dossier de "O7planning".

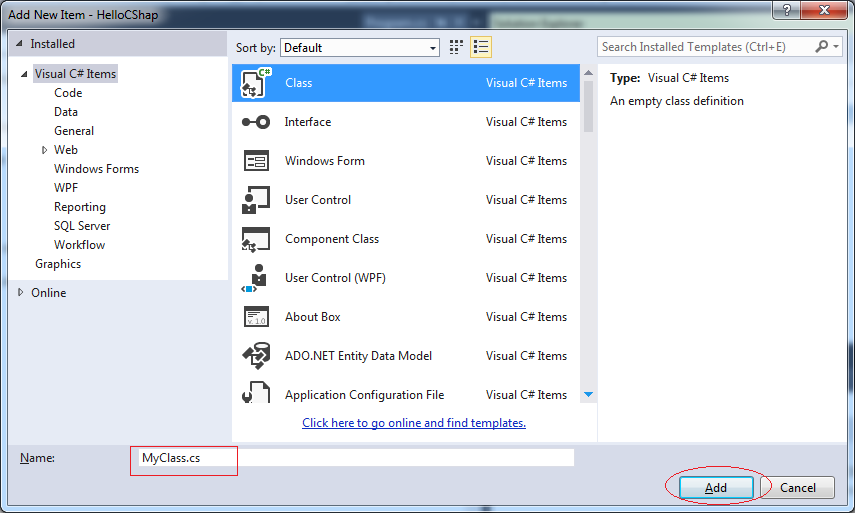


Cliquez sur le bouton droit du dossier "CSharp" et sélectionnez:

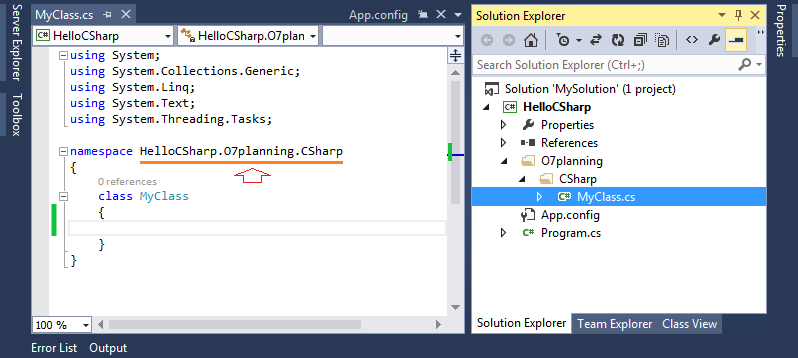
* Add/Class



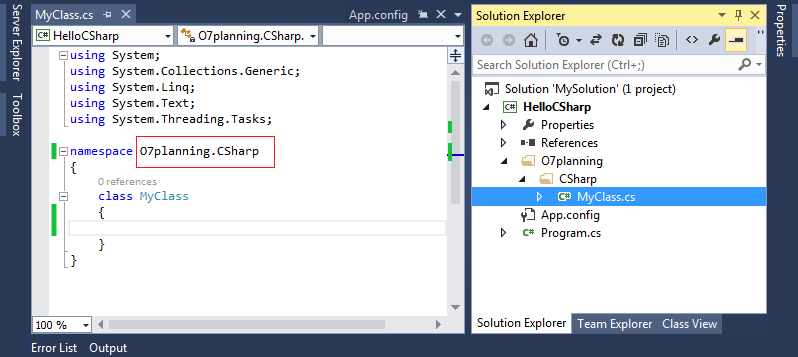
Sélectionnez le type de l'élément Class, et saisissez le nom de la classe.



La classe a été créée, elle se trouve dans le nom de l'espace "HelloCSharp.O7planning.CSharp". Vous pouvez renommer du namespace en "O7planning.CSharp".



Renommez namespace en "O7planning.CSharp".



Vous pouvez changer le contenu de la classe:

MyClass.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **O7planning.CSharp**

{

**class** **MyClass**

{

**static** **void** **Main**(string[] args)

{

Console.WriteLine("Hello from MyClass");

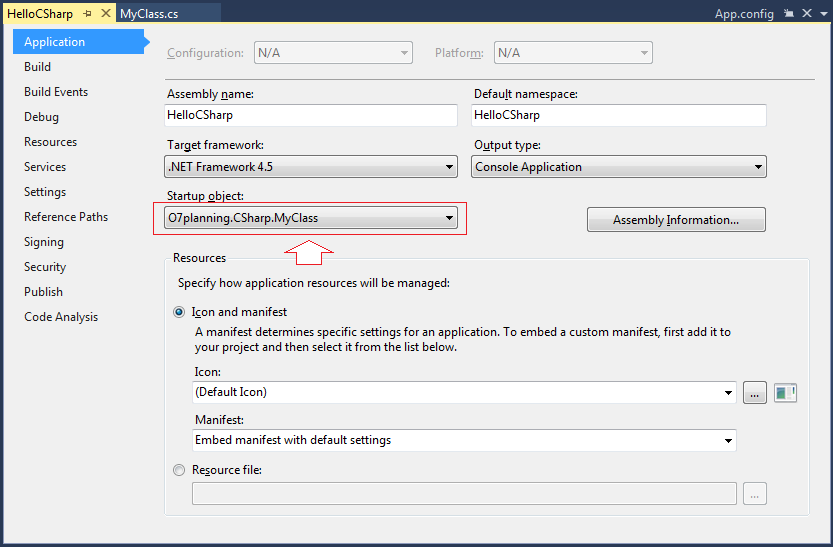
Console.ReadLine();

}

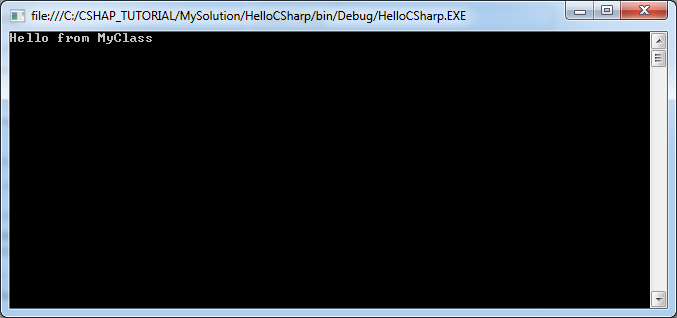
}

}

La déclaration de la classe MyClass est le point de départ de l'exécution. Cliquez sur le bouton droit du Project et sélectionner Properties



L'exécution de l'exemple:



|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Represents** | **Range** | **Default Value** |
| **bool** | **Boolean value** | **True or False** | **False** |
| **byte** | **8-bit unsigned integer** | **0 to 255** | **0** |
| **char** | **16-bit Unicode character** | **U +0000 to U +ffff** | **'\0'** |
| **decimal** | **128-bit precise decimal values with 28-29 significant digits** | **(-7.9 x 1028 to 7.9 x 1028) / 100 to 28** | **0.0M** |
| **double** | **64-bit double-precision floating point type** | **(+/-)5.0 x 10-324 to (+/-)1.7 x 10308** | **0.0D** |
| **float** | **32-bit single-precision floating point type** | **-3.4 x 1038 to + 3.4 x 1038** | **0.0F** |
| **int** | **32-bit signed integer type** | **-2,147,483,648 to 2,147,483,647** | **0** |
| **long** | **64-bit signed integer type** | **-923,372,036,854,775,808 to 9,223,372,036,854,775,807** | **0L** |
| **sbyte** | **8-bit signed integer type** | **-128 to 127** | **0** |
| **short** | **16-bit signed integer type** | **-32,768 to 32,767** | **0** |
| **uint** | **32-bit unsigned integer type** | **0 to 4,294,967,295** | **0** |
| **ulong** | **64-bit unsigned integer type** | **0 to 18,446,744,073,709,551,615** | **0** |
| **ushort** | **16-bit unsigned integer type** | **0 to 65,535** | **0** |

7- Les types de données dans C#

8- Des variables et la déclaration de variable

Une variable est identifiée par un nom, pour une zone de stockage de données que votre programme peut être manipulé. Chaque variable en C# a un type spécifique, qui détermine la taille et la disposition de la mémoire de la variable de l'amplitude de valeurs qui peuvent être stockées dans cette mémoire et l'ensemble des opérations qui peuvent être appliquées à la variable.

La variable peut changer de valeur au cours de son existence dans le programme. Les variables ont une valeur fixe appelée constantes. L'utilisation du mot-clé const pour déclarer une variable constante.

Déclarer une variable:

// Déclarez une variable

// Data Type: Type de données.

// Variable Name: Nom de variable.

<Data Type> <Variable name>;

// Déclarez une variable et lui attribuez une valeur.

<Data Type> <Variable name> = <**value**>;

// Déclarez une constante (constants)

**const** <Data Type> <Variable name> = <**value**>;

VariableExample.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **HelloCSharp**

{

**class** **VariableExample**

{

**static** **void** **Main**(string[] args)

{

// Déclarez un constant du type int.

// Vous ne pouvez pas attribuer de nouvelles valeurs pour les constantes.

**const** int MAX\_SCORE = 100;

// Déclarez une variable du type int.

int score = 0;

// Attribuez une nouvelle valeur pour la variable score.

score = 90;

// Déclarez une chaîne (string).

string studentName = "Tom";

// Affichez la valeur de la variable dans l'écran Console.

Console.WriteLine("Hi {0}", studentName);

Console.WriteLine("Your score: {0}/{1}", score, MAX\_SCORE);

// Attendez que l'utilisateur saisit quelque chose et appuyez sur Enter avant de terminer le programme.

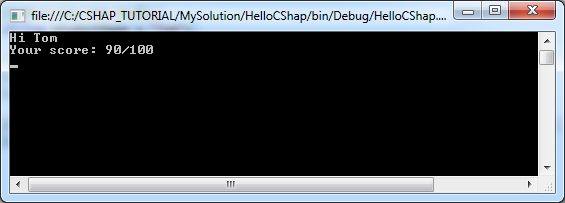
Console.ReadLine();

}

}

}

L'exécution de l'exemple:



9- Des déclarations Branch

9.1- La déclaration If-else

if est une déclaration qui vérifie une certaines conditions dans C#. Par exemple: Si a > b faites quelques choses ....

Les opérateurs de comparaison communs:

|  |  |  |
| --- | --- | --- |
| Opérateur | Signification | Exemple |
| > | Plus grand que | 5 > 4 est vrai (true) |
| < | Plus petit que | 4 < 5 est vrai (true) |
| >= | Plus grand que ou égale à | 4 >= 4 est vrai (true) |
| <= | Plus petit que ou égale à | 3 <= 4 est vrai (true) |
| == | Égale à | 1 == 1 est vrai (true) |
| != | Différent de | 1 != 2 est vrai (true) |
| && | Et | a > 4 && a < 10 |
| || | Ou | a == 1 || a == 4 |

// Syntaxe

// condition: La condition de vérification.

**if** ( condition )

{

// Faites quelques choses ici.

}

Exemple:

// Exemple 1:

**if** ( 5 < 10 )

{

Console.WriteLine( "Five is now less than ten");

}

// Exemple 2:

**if** ( true )

{

Console.WriteLine( "Do something here");

}

La structure complềt de la déclaration if - else if - else:

// Notez qu'au maximum un bloc de code sera exécuté.

// Le programme vérifie les conditions de haut en bas.

// Lors qu'une condition est vraie, ce bloc sera exécuté.

// Les conditions restantes seront ignorées.

...

// Si la condition 1 est vraie alors ..

**if** (condition1 )

{

// Faites quelque chose si la condition1 est vraie.

}

// Sinon si la condition2 est vraie alors ....

**else** **if**(condition2 )

{

// ​​​​​​​Faites quelque chose ici si la condition 2 est vraie

// (condition1 est fausse).

}

// Sinon si la condition N est vraie alors ...

**else** **if**(conditionN )

{

// Faites quelque chose si la condition N est vraie

// (Toutes les conditions ci-dessus sont fausses)

}

// Sinon (Toutes les conditions ci-dessus sont fausses).

**else** {

// Faites quelque chose.

}

IfElseExample.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **HelloCSharp**

{

**class** **IfElseExample**

{

**static** **void** **Main**(string[] args)

{

// Déclarez un entier du type int, décrit votre âge.

int age;

Console.WriteLine("Please enter your age: \n");

// Déclarez une variable pour stocker l'extrait du texte que l'utilisateur a saisi à partir du clavier.

string inputStr = Console.ReadLine();

// Int32 est une classe qui se trouve dans l'espace de noms System.

// Utilisez la méthode statique Parse de la classe Int32 pour convertir une chaîne en nombre

// Et attribuez à la variable age.

// (Remarque: Si 'inputStr' n'est pas de chaîne en nombre, une erreur de programme se produira ici).

age = Int32.Parse(inputStr);

Console.WriteLine("Your age: {0}", age);

// Vérifiez si l'âge est inférieur à 80 alors ...

**if** (age < 80)

{

Console.WriteLine("You are pretty young");

}

// Sinon, si l'âge entre 80 et 100 ans, alors

**else** **if** (age >= 80 && age <= 100)

{

Console.WriteLine("You are old");

}

// Sinon, (Les cas restants)

**else**

{

Console.WriteLine("You are verry old");

}

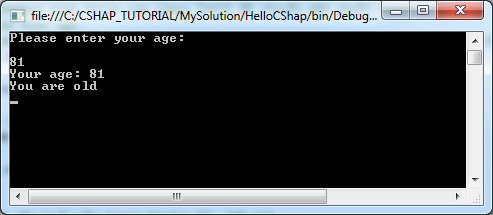
Console.ReadLine();

}

}

}

L'exécution de l'exemple, et le saisissement 81, et consulter les résultats:



9.2- La déclaration Switch-Case

La syntaxe de *switch*:

// Utilisez le mot-clé 'switch' afin de vérifier la valeur d'une variable.

**switch** ( <variable> )

{

**case** value1:

// Faites quelque chose ici si la valeur de variable égale à value1.

**break**;

**case** value2:

// Faites quelque chose ici si la valeur de variable égale à value2.

**break**;

...

default:

// Les cas restants.

**break**;

}

BreakExample.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **HelloCSharp**

{

**class** **BreakExample**

{

**static** **void** **Main**(string[] args)

{

// Demandez aux utilisateurs de sélectionner une option.

Console.WriteLine("Please select one option:\n");

Console.WriteLine("1 - Play a game \n");

Console.WriteLine("2 - Play music \n");

Console.WriteLine("3 - Shutdown computer \n");

// Déclarez une variable 'option'.

int option;

// La chaîne que l'utilisateur a saisie à partir du clavier

string inputStr = Console.ReadLine();

// Convertissez en entier.

option = Int32.Parse(inputStr);

// Vérifiez la valeur de 'option'.

**switch** (option)

{

**case** 1:

Console.WriteLine("You choose to play the game");

**break**;

**case** 2:

Console.WriteLine("You choose to play the music");

**break**;

**case** 3:

Console.WriteLine("You choose to shutdown the computer");

**break**;

default:

Console.WriteLine("Nothing to do...");

**break**;

}

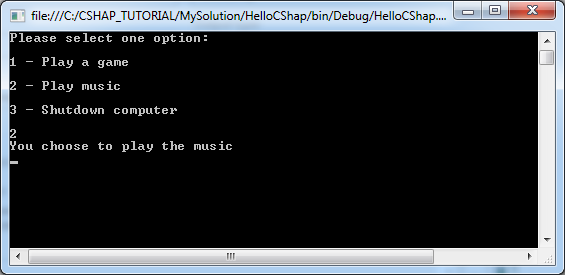
Console.ReadLine();

}

}

}

L'exécution de l'exemple et le saisissement 2:



*Dans ce cas, la commande break a dit au programme de sortir switch.*

Vous pouvez inclure plusieurs (case) ensemble pour exécuter un même bloc.

BreakExample2.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **HelloCSharp**

{

**class** **BreakExample2**

{

**static** **void** **Main**(string[] args)

{

// Déclarez une variable et lui attribuez la valeur de 3.

// ​​​​​​​@@ (2)

// Vérifiez la valeur de 'option'.

int option = 3;

Console.WriteLine("Option = {0}", option);

// (2)

**switch** (option)

{

**case** 1:

Console.WriteLine("Case 1");

**break**;

// Si valeur dans (2,3,4,5)

**case** 2:

**case** 3:

**case** 4:

**case** 5:

Console.WriteLine("Case 2,3,4,5!!!");

**break**;

default:

Console.WriteLine("Nothing to do...");

**break**;

}

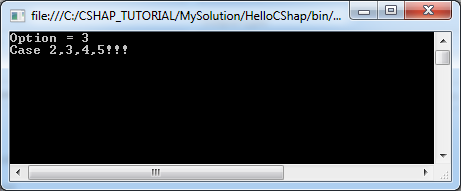
Console.ReadLine();

}

}

}

L'exécution de l'exemple:



10- La boucle dans C#

La boucle (loop) est utilisée pour exécuter de manière répétitive un bloc de déclaration. Cela permet à votre programme d'exécuter à plusieurs reprises un bloc de déclaration à plusieurs reprises, c'est l'une des tâches de base de la programmation.

C# fournit 3 types différents de boucles:

* FOR
* WHILE
* DO WHILE

10.1- La boucle for

La syntaxe de la boucle for:

// initialize variable: initialiser la variable.

// condition: La condition.

// updates new value for variable: Mettre à jour de la nouvelle valeur pour variable.

**for** (initialize variable ; condition ; updates **new** **value** **for** variable )

{

// Exécutez le bloc de commande dans le cas où la condition est vraie (true).

}

Exemple:

// Exemple 1:

// Déclarez une variable x et lui attribuez la valeur de 0.

// La condition est x <5

// Si x <5 est vraie alors exécutez le bloc.

// Après chaque itération (iteration), la valeur de x est augmentée de 1.

**for** (int x = 0; x < 5 ; x = x + 1)

{

// Faites quelque chose ici si la condition x < 5 est vraie (true).

}

// Exemple 2:

// Déclarez la variable x et lui attribuez la valeur de 2

// La condition est x <15

// Si x <15 est vraie alors exécutez le bloc.

// Après chaque itération (iteration), la valeur de x est augmentée de 3.

**for** (int x = 2; x < 15 ; x = x + 3)

{

// Faites quelque chose ici si la condition x < 15 est vraie (true).

}

ForLoopExample.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **HelloCSharp**

{

**class** **ForLoopExample**

{

**static** **void** **Main**(string[] args)

{

Console.WriteLine("For loop example");

// Déclarez la variable x et lui attribuez la valeur de 2

// La condition est x <15

// Si x <15 est vraie alors exécutez le bloc.

// Après chaque itération (iteration), la valeur de x est augmentée de 3.

**for** (int x = 2; x < 15; x = x + 3)

{

Console.WriteLine( );

Console.WriteLine("Value of x = {0}", x);

}

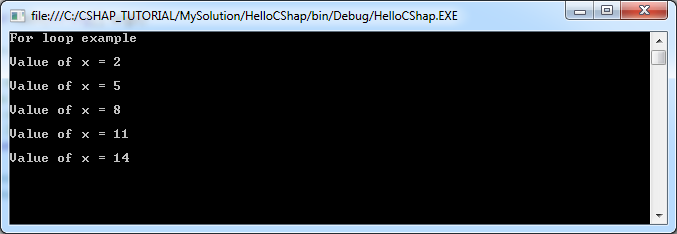
Console.ReadLine();

}

}

}

L'exécution de l'exemple:



10.2- La boucle while

La syntaxe de la boucle while:

// condition: La condition

**while** (condition)

{

// Alors que la 'condition' est vraie, exécutez le bloc.

}

Exemple:

// Déclarez une variable x.

int x = 2;

**while** ( x < 10)

{

// Faites quelque chose ici si la condition x < 10 est encore vraie.

...

// Mettez à jour de nouvelle valeur de la variable x.

x = x + 3;

}

WhileLoopExample.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **HelloCSharp**

{

**class** **WhileLoopExample**

{

**static** **void** **Main**(string[] args)

{

Console.WriteLine("While loop example");

// Déclarez une varible x et lui attribuez la valeur de 2.

int x = 2;

// La condition est x < 10.

// Si x < 10 est vrai alors exécutez le bloc.

**while** (x < 10)

{

Console.WriteLine("Value of x = {0}", x);

x = x + 3;

}

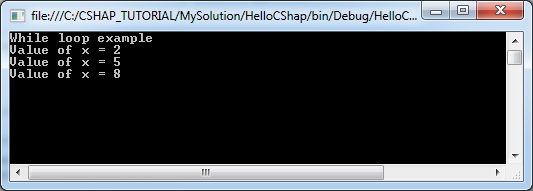
Console.ReadLine();

}

}

}

L'exécution de l'exemple:



10.3- La boucle do-while

La syntaxe de la boucle do-while

// Les caractéristiques de la boucle 'do-while' sont qu'il exécutera le bloc au moins 1 fois.

// Après chaque itération (iteration), il vérifie la condition,

// Si la condition est vraie, le bloc sera exécuté.

**do** {

// Faites quelque chose ici.

} **while** (condition);

// Remarque : il devrait y avoir un point-virgule (;) ici.

DoWhileLoopExample.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **HelloCSharp**

{

**class** **DoWhileLoopExample**

{

**static** **void** **Main**(string[] args)

{

Console.WriteLine("Do-While loop example");

// Déclarez une variable x et lui attribuez une valeur de 2.

int x = 2;

// Exécutez le bloc au moins une fois.

// Après chaque itération (iteration), il vérifie la condition,

// Si la condition est vraie, le bloc sera exécuté.

// ​​​​​​​

**do**

{

Console.WriteLine("Value of x = {0}", x);

x = x + 3;

} **while** (x < 10);

// Remarque : il devrait y avoir un point-virgule (;) ici.

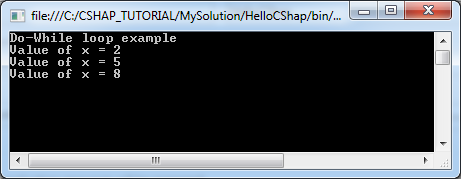
Console.ReadLine();

}

}

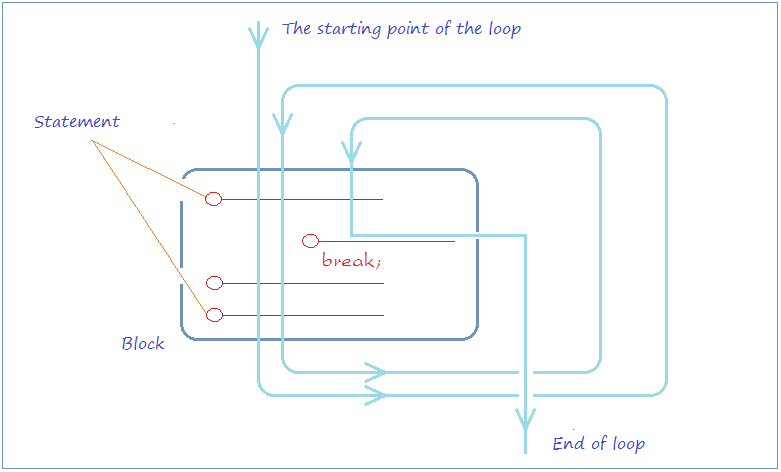
}

L'exécution de l'exemple:



10.4- La commande break dans la boucle

break est une commande qui peut être située dans un bloc de boucle. Cette commande met fin à la boucle inconditionnellement.



LoopBreakExample.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **HelloCSharp**

{

**class** **LoopBreakExample**

{

**static** **void** **Main**(string[] args)

{

Console.WriteLine("Break example");

// Déclarez une variable x et lui attribuez une valeur de 2.

int x = 2;

**while** (x < 15)

{

Console.WriteLine("----------------------\n");

Console.WriteLine("x = {0}", x);

// Vérifiez si x = 5, quittez la boucle.

**if** (x == 5)

{

**break**;

}

// Augmentez la valeur de x par 1 (Écrit en bref x = x + 1;).

x++;

Console.WriteLine("x after ++ = {0}", x);

}

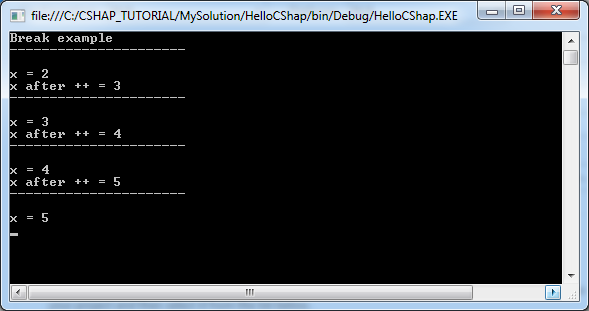
Console.ReadLine();

}

}

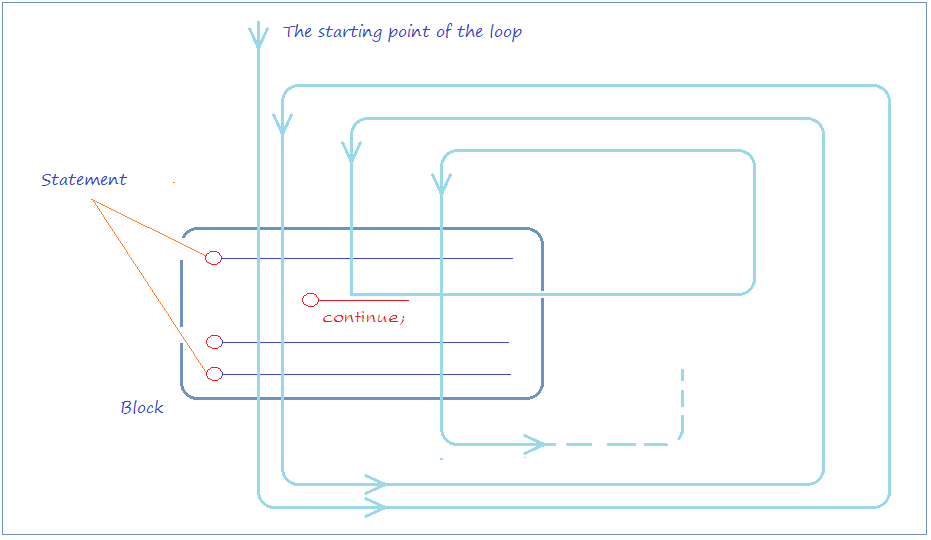
}

L'exécution de l'exemple:



10.5- La commande continue dans la boucle

*continue*est une déclaration qui peut être située dans une boucle. Lorsqu'on saisit la déclaration continue, le programme ignorera les lignes de commande en bloc, en dessous de la commande continue et le début d'une nouvelle boucle.



LoopContinueExample.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **HelloCSharp**

{

**class** **LoopContinueExample**

{

**static** **void** **Main**(string[] args)

{

Console.WriteLine("Continue example");

// Déclarez une valeur x et lui attribuez une valeur de 2.

int x = 2;

**while** (x < 7)

{

Console.WriteLine("----------------------\n");

Console.WriteLine("x = {0}", x);

// L'opérateur % sert à calculer le reste.

// Si x est pair, ignorez la ligne de commande ci-dessous de 'continuer',

// et lancez un nouvel itérateur (si la condition est toujours vraie).

**if** (x % 2 == 0)

{

// Augmentez la valeur de x par 1. (écrit en bref x = x + 1;).

x++;

**continue**;

}

**else**

{

// Augmentez la valeur de x par 1. (écrit en bref x = x + 1;).

x++;

}

Console.WriteLine("x after ++ = {0}", x);

}

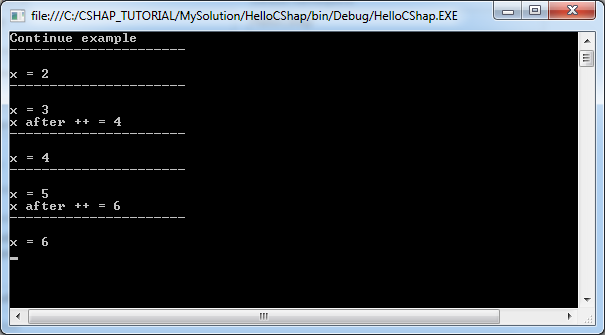
Console.ReadLine();

}

}

}

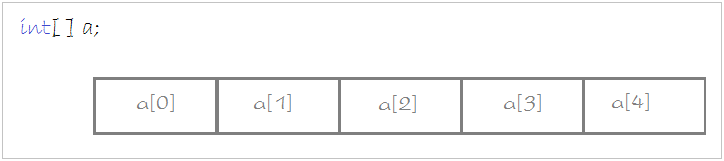
L'exécution de l'exemple:



11- Des tableaux dans C#

11.1- Tableau à une seule dimension (uni-dimensionnel)

Ce sont des illustrations du tableau à une seule dimension avec 5 éléments indexés de 0 à 4.



La syntaxe de déclaration un tableau à une seule dimension:

// Méthode 1:

// Déclarez un tableau de int, spécifiez les éléments.

int[] years = { 2001, 2003, 2005, 1980, 2003 };

// Méthode 2:

// Déclarez un tableau de float en spécifiant les éléments.

// (3 éléments).

float[] salaries = **new** float[3];

ArrayExample1.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **HelloCSharp**

{

**class** **ArrayExample1**

{

**static** **void** **Main**(string[] args)

{

// Méthode 1:

// Déclarez un tableau de int, spécifie les éléments.

int[] years = { 2001, 2003, 2005, 1980, 2003 };

// Length est une propriété du tableau, elle renvoie le nombre d'éléments.

Console.WriteLine("Element count of array years = {0} \n", years.Length);

// Utilisez une boucle for pour imprimer les éléments du tableau.

**for** (int i = 0; i < years.Length; i++) {

Console.WriteLine("Element at {0} = {1}", i, years[i]);

}

// Méthode 2:

// Déclarez un tableau de flot, en spécifiant le nombre d'éléments.

// (3 éléments)

float[] salaries = **new** float[3];

// Attribuez des valeurs aux éléments.

salaries[0] = 1000;

salaries[1] = 1200;

salaries[2] = 1100;

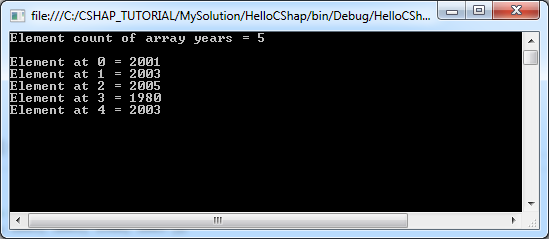
Console.ReadLine();

}

}

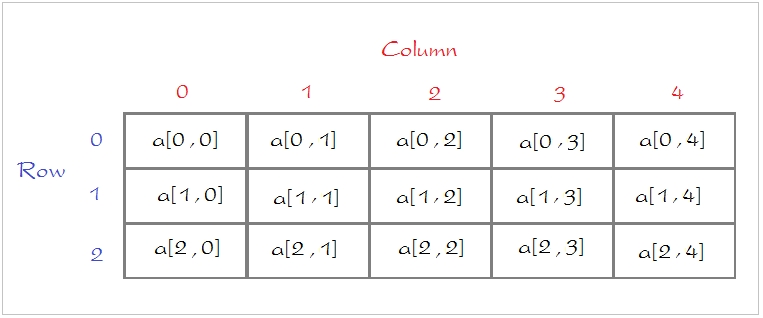
}

L'exécution de l'exemple:



11.2- Tableau à deux dimensions

Ceci est l'illustration d'un tableau bidimensionnel



La syntaxe pour déclarer un tableau bidimensionnel:

// Déclarez un tableau à 2 dimensions, 3 lignes et 5 colonnes.

// spécifiez les éléments.

int[,] a = **new** int[,] {

{1 , 2 , 3, 4, 5} ,

{0, 3, 4, 5, 7},

{0, 3, 4, 0, 0}

};

// Déclarez un tableau à 2 dimensions, 3 lignes et 5 colonnes

// L'élément n'a pas reçu de valeur.

int[,] a = **new** int[3,5];

ArrayExample2.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **HelloCSharp**

{

**class** **ArrayExample2**

{

**static** **void** **Main**(string[] args)

{

// Déclarez un tableau à 2 dimensions, 3 lignes et 5 colonnes.

// Spécifiez les éléments.

int[,] a = {

{ 1, 2, 3, 4, 5 },

{ 0, 3, 4, 5, 7 },

{ 0, 3, 4, 0, 0 }

};

// Utilisez une boucle for pour imprimer les éléments du tableau.

**for** (int row = 0; row < 3; row++) {

**for** (int col = 0; col < 5; col++) {

Console.WriteLine("Element at [{0},{1}] = {2}", row, col, a[row,col]);

}

Console.WriteLine("-------------");

}

// Déclarez un tableau à 2 dimensions, 3 lignes, 5 colonnes

// L'élément n'a pas reçu de valeur.

int[,] b = **new** int[3, 5];

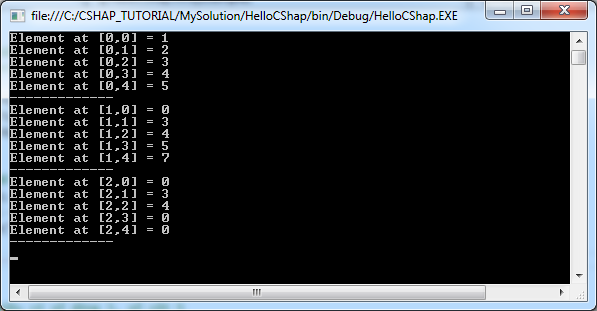
Console.ReadLine();

}

}

}

L'exécution de l'exemple:



11.3- Tableau d'un tableau

ArrayOfArrayExample.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **HelloCSharp**

{

**class** **ArrayOfArrayExample**

{

**static** **void** **Main**(string[] args)

{

// Déclarez un tableau de 3 éléments.

// Chaque élément est un autre tableau.

string[][] teams = **new** string[3][];

string[] mu = { "Beckham","Giggs"};

string[] asenal = { "Oezil", "Szczęsny", "Walcott" };

string[] chelsea = {"Oscar","Hazard","Drogba" };

teams[0] = mu;

teams[1] = asenal;

teams[2] = chelsea;

// Utilisez une boucle for pour imprimer les éléments du tableau.

**for** (int row = 0; row < teams.Length; row++)

{

**for** (int col = 0; col < teams[row].Length ; col++)

{

Console.WriteLine("Element at [{0}],[{1}] = {2}", row, col, teams[row][col]);

}

Console.WriteLine("-------------");

}

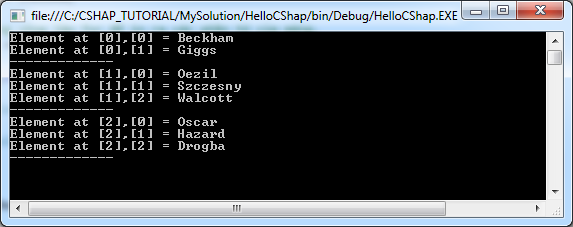
Console.ReadLine();

}

}

}

L'execution de l'exemple:

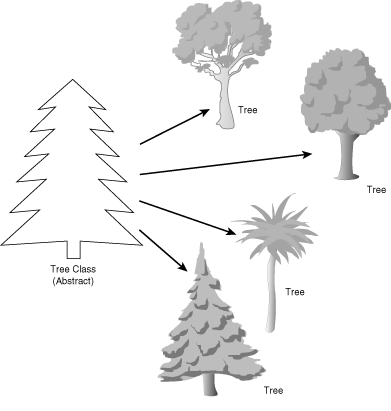


12- Classe, Constructeur et des instances

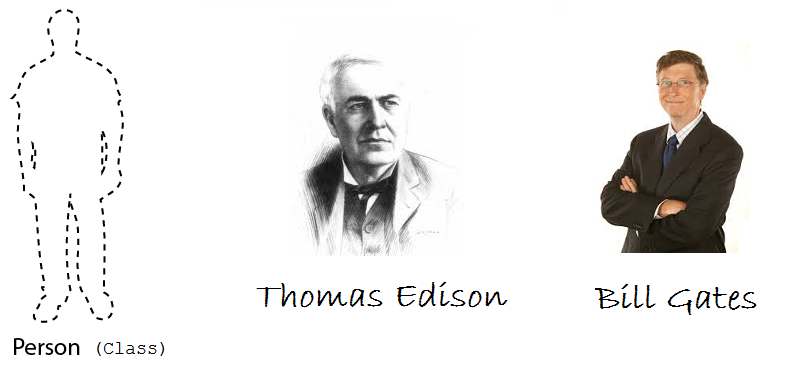
You need to have a distinction between three concepts:

* Class
* Constructor
* Instance

When we talk about the tree, it is something abstract, it is a class. But when we pointed to a specific tree, it was clear, and that is the instance



Or when we talk about the Person, that's abstract, it is a class. But when pointed at you or me, it is two different instances, the same class of People.



Person.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **HelloCSharp**

{

**class** **Person**

{

// Ceci est un champ (field).

// Pour stocker le nom de la personne.

**public** string Name;

// Ceci est un constructeur,

// Utilisé pour initialiser l'objet.

// Ce constituant a un paramètre.

// Le constructeur a toujours le même nom que celui de la classe

**public** **Person**(string persionName)

{

// Attribuez la valeur du paramètre au champ name.

**this**.Name = persionName;

}

// C'est un type de retour de méthode qui renvoie du type string.

**public** string **GetName**()

{

**return** **this**.Name;

}

}

}

Person class without Main method. Next in PersonTest, we create Person instance from its Constructor

PersonTest.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **HelloCSharp**

{

**class** **PersonTest**

{

**static** **void** **Main**(string[] args)

{

// Créez un objet de classe Person

// Initialisez cet objet à partir du constructeur de la classe Person.

Person edison = **new** Person("Edison");

// La classe Person a la méthode GetName().

// L'utilisation de l'objet pour appeler la méthode GetName().

String name = edison.GetName();

Console.WriteLine("Person 1: " + name);

// Créez un objet de classe Person.

// Initialisez cet objet à partir de la classe Constructor of Person.

Person billGate = **new** Person("Bill Gates");

// La classe Person dont le champ 'name' est public (public). Accédez au domaine public de la personne.

// Vous pouvez utiliser l'objet pour accéder au champ 'name'.

String name2 = billGate.Name;

Console.WriteLine("Person 2: " + name2);

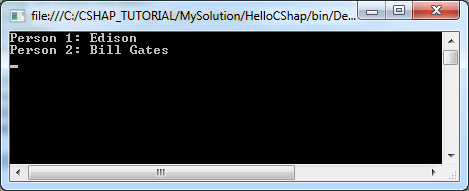
Console.ReadLine();

}

}

}

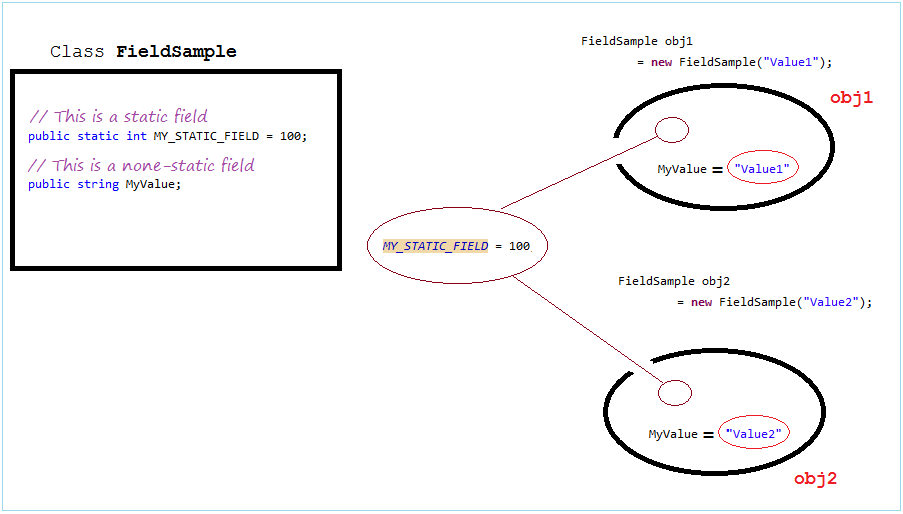
Running the example:



13- Le champ (Field)

Dans cette section, nous aborderons certains des concepts :

* Field
  + Normal field
  + static Field
  + const Field
  + readonly Field



FieldSample.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **HelloCSharp**

{

**class** **FieldSample**

{

// Ceci est un champ statique (static field).

**public** **static** int MY\_STATIC\_FIELD = 100;

// Ceci est un champ.

**public** string MyValue;

// Voici est un Constructeur de la classe FieldSample.

**public** **FieldSample**(string **value**)

{

**this**.MyValue = **value**;

}

}

}

FieldSampleTest.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **HelloCSharp**

{

**class** **FieldSampleTest**

{

**static** **void** **Main**(string[] args)

{

// Imprimez la valeur du champ statique (static field).

// Avec le champ statique, vous devez l'accéder via le nom de la classe.

Console.WriteLine("FieldSample.MY\_STATIC\_FIELD= {0}", FieldSample.MY\_STATIC\_FIELD);

// Vous pouvez changer la valeur du champ statique.

FieldSample.MY\_STATIC\_FIELD = 200;

Console.WriteLine(" ------------- ");

// Créez le premier objet.

FieldSample obj1 = **new** FieldSample("Value1");

// Avec le champ non-statique, vous devez y accéder via l'objet.

Console.WriteLine("obj1.MyValue= {0}", obj1.MyValue);

// Créez le deuxième objet :

FieldSample obj2 = **new** FieldSample("Value2");

Console.WriteLine("obj2.MyValue= {0}" , obj2.MyValue);

// Vous pouvez changer la valeur du champ.

obj2.MyValue = "Value2-2";

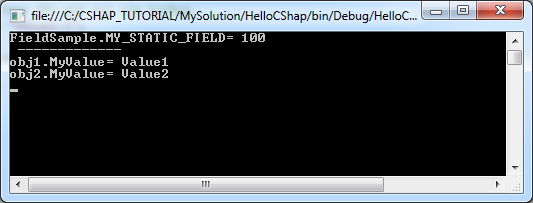
Console.ReadLine();

}

}

}

Running the example:



Example: readonly & static readonly.

ConstFieldExample.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **HelloCSharp**

{

**class** **ConstFieldExample**

{

// Un champ constant, sa valeur est déterminée au moment de la compilation.

// Vous ne pouvez pas attribuer de nouvelles valeurs pour les constantes

// Remarque: la constante est toujours statique.

**public** **const** int MY\_VALUE = 100;

// Un champ statique (static field) et en lecture seule.

// Sa valeur est attribuée une seule fois lors de la première exécution dans le Constructeur.

**public** **static** **readonly** DateTime INIT\_DATE\_TIME1 = DateTime.Now;

// Un champ en lecture seule et non-statique (none-static)

// Sa valeur est attribuée une seule fois dans Constructeur.

**public** **readonly** DateTime INIT\_DATE\_TIME2 ;

**public** **ConstFieldExample**()

{

// Sa valeur est attribuée une seule fois lors de la première exécution.

INIT\_DATE\_TIME2 = DateTime.Now;

}

}

}

14- La méthode (Method)

Method

* Method.
* static Method
* • Méthode scellée. (Sera mentionné dans l'héritage de la classe).

MethodSample.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **HelloCSharp**

{

**class** **MethodSample**

{

**public** string text = "Some text";

// Un Constructeur par défaut.

// Cela signifie que le constructeur est sans paramètres.

**public** **MethodSample**()

{

}

// Ceci est une méthode qui renvoie le type de chaîne.

// Cette méthode est sans paramètre.

**public** string **GetText**()

{

**return** **this**.text;

}

// C'est une méthode avec un paramètre String.

// Retour vide (ne retourne rien).

**public** **void** **SetText**(string text)

{

// this.text: référence vers le champ texte.

// Pour distinguer au paramètre du texte.

**this**.text = text;

}

// Ceci est une méthode statique.

// Renvoie le type int et a 3 paramètres.

**public** **static** int **Sum**(int a, int b, int c)

{

int d = a + b + c;

**return** d;

}

}

}

MethodSampleTest.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **HelloCSharp**

{

**class** **MethodSampleTest**

{

**static** **void** **Main**(string[] args)

{

// Créez l'objet MethodSample

MethodSample obj = **new** MethodSample();

// Appelez la méthode GetText().

// La méthode non statique doit être appelée à travers l'objet.

String text = obj.GetText();

Console.WriteLine("Text = " + text);

// Appelez la méthode SetText (chaîne).

// La méthode non-statique doit être appelée à travers l'objet.

obj.SetText("New Text");

Console.WriteLine("Text = " + obj.GetText());

// La méthode statique doit être appelée dans la classe.

int sum = MethodSample.Sum(10, 20, 30);

Console.WriteLine("Sum 10,20,30= " + sum);

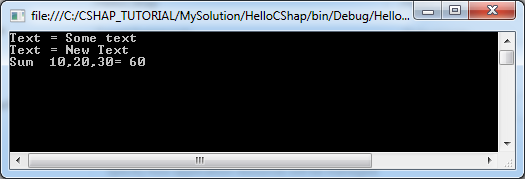
Console.ReadLine();

}

}

}

Running the example:



15- L'héritage dans C#

Java autorise les classes qui s'étendent d'une autre classe. La classe étend une autre classe appelée sous-classes. Les sous-classes ont la possibilité d'hériter des attributs de champs, des méthodes de la classe parente.

Voir un exemple d'héritage dans CSharp :

Animal.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **HelloCSharp**

{

// Simule une classe d'animaux.

**class** **Animal**

{

**public** **Animal**()

{

}

**public** **void** **Move**()

{

Console.WriteLine("Move ...!");

}

}

}

Cat.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **HelloCSharp**

{

**class** **Cat** : **Animal**

{

**public** **void** **Say**()

{

Console.WriteLine("Meo");

}

// Une méthode de la classe Cat.

**public** **void** **Catch**()

{

Console.WriteLine("Catch Mouse");

}

}

}

Ant.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **HelloCSharp**

{

// Une fourmi

**class** **Ant** : **Animal**

{

}

}

AnimalTest.cs

**using** System;

**using** System.Collections.Generic;

**using** System.Linq;

**using** System.Text;

**using** System.Threading.Tasks;

**namespace** **HelloCSharp**

{

**class** **AnimalTest**

{

**static** **void** **Main**(string[] args)

{

// Créer un objet Cat.

Cat tom = **new** Cat();

// Vérifiez si 'tom' est Animal

// Le résultat est vrai

bool isAnimal = tom **is** Animal;

// ==> true

Console.WriteLine("tom is Animal? " + isAnimal);

// Appelez la méthode Catch.

tom.Catch();

// Appelez la méthode Say()

// ==> Meo

tom.Say();

Console.WriteLine("--------------------");

// Déclarez un objet Animal.

// Créez-le à partir du constructeur de Cat.

Animal tom2 = **new** Cat();

// Appelez la méthode Move().

tom2.Move();

Console.WriteLine("--------------------");

// Créez un objet Ant.

Ant ant = **new** Ant();

// Appelez la méthode Move(), héritée de Animal.

ant.Move();

Console.ReadLine();

}

}

}

Le résultat de l'exécution de la classe AnimalTest:

