



# Base du Language

## C#

Mme Ben bouzid NOURHENE



# Plan:

- Déclaration des variables et Types
- Cast et conversions de types
- La Classe Console
- Les operateurs du langage : Les opérateurs de calcul ,de Test et Logiques ...
- Les instructions conditionnelles et itératives
- La gestion des exceptions

# Déclaration des variables et Types

- Vérifiés par CLR, par Common Type System (CTS)
- CTS définit les types de données que le Runtime .net comprend et que les applications .net peuvent utiliser.
- CLS est un sous ensemble de la CTS
- Un programme qui utilise des types de la CLS peut interagir avec un autre programme .net écrit dans un autre langage.
- Exemple: une classe c# peut heriter d'une classe VB .net

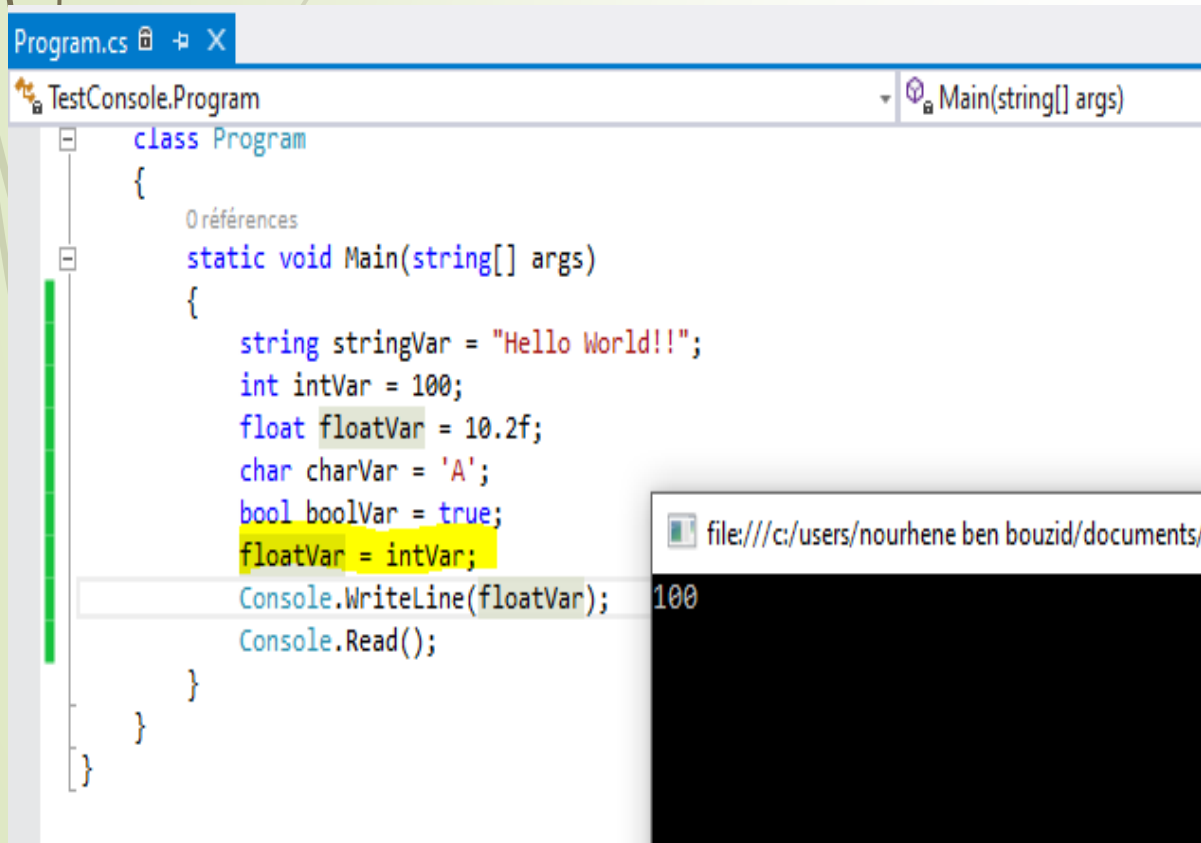
# Déclaration des variables et Types

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	$(-7.9 \times 10^{28} \text{ to } 7.9 \times 10^{28}) / 10^{28-29}$	0.0M
double	64-bit double-precision floating point type	$(+/-)5.0 \times 10^{-324} \text{ to } (+/-)1.7 \times 10^{308}$	0.0D
float	32-bit single-precision floating point type	$-3.4 \times 10^{38} \text{ to } +3.4 \times 10^{38}$	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

# Cast et conversions de types

## ➤ Conversions implicites:

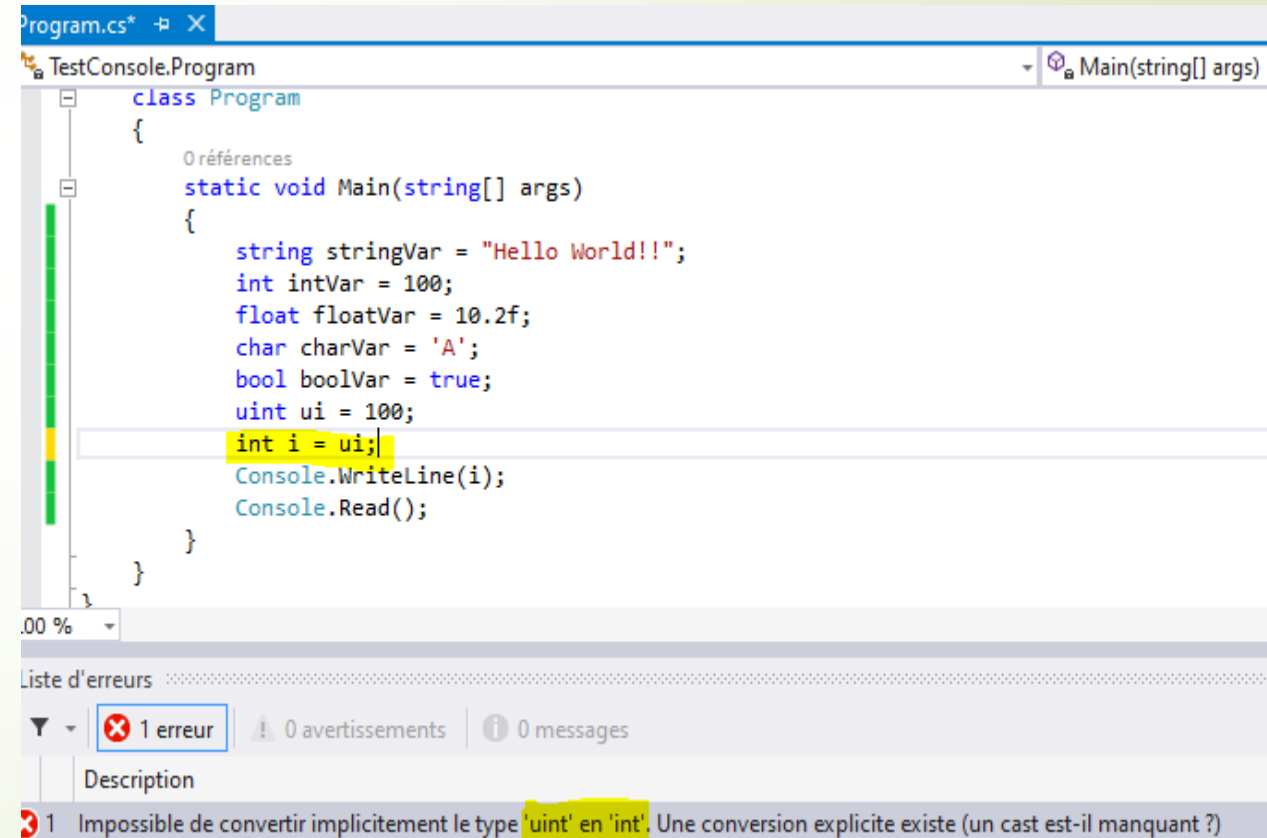
Pour les types numériques intégrés, une conversion implicite peut être effectuée quand la valeur à stocker peut tenir dans la variable sans être tronquée ni arrondie.



```
Program.cs
TestConsole.Program
Main(string[] args)
class Program
{
    // Références
    static void Main(string[] args)
    {
        string stringVar = "Hello World!!";
        int intVar = 100;
        float floatVar = 10.2f;
        char charVar = 'A';
        bool boolVar = true;
        floatVar = intVar;
        Console.WriteLine(floatVar);
        Console.Read();
    }
}
```

file:///c:/users/nourhene ben bouzid/documents/

100



```
Program.cs*
TestConsole.Program
Main(string[] args)
class Program
{
    // Références
    static void Main(string[] args)
    {
        string stringVar = "Hello World!!";
        int intVar = 100;
        float floatVar = 10.2f;
        char charVar = 'A';
        bool boolVar = true;
        uint ui = 100;
        int i = ui;
        Console.WriteLine(i);
        Console.Read();
    }
}
```

0.00 %

Liste d'erreurs

1 erreur 0 avertissements 0 messages

Description

1 Impossible de convertir implicitement le type 'uint' en 'int'. Une conversion explicite existe (un cast est-il manquant ?)

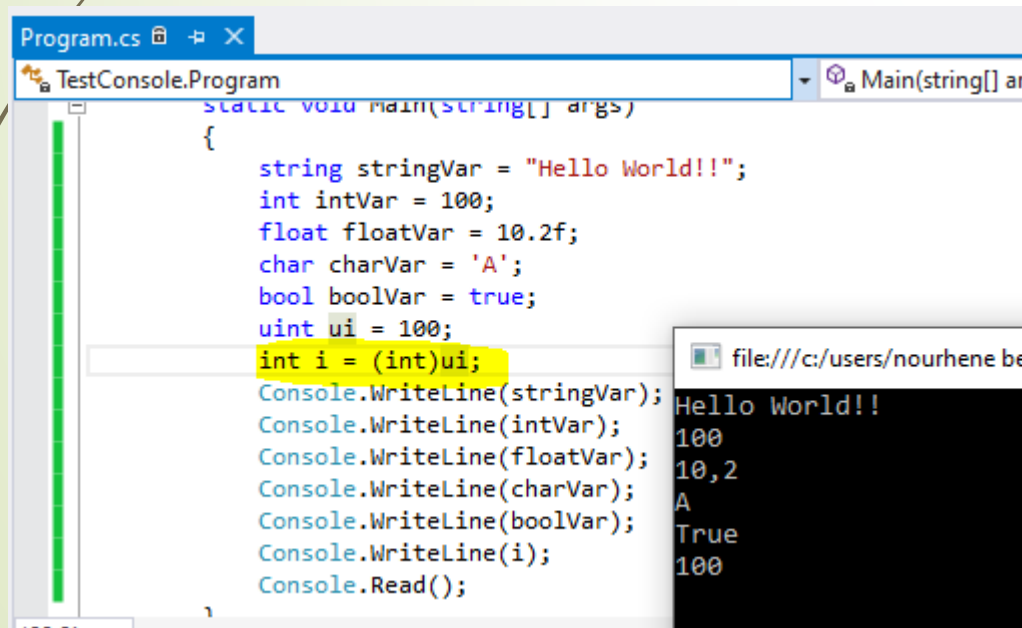
# Cast et conversions de types

## ➤ Conversions explicites: CAST

Un cast est un moyen d'informer explicitement le compilateur que vous avez l'intention d'effectuer la conversion.

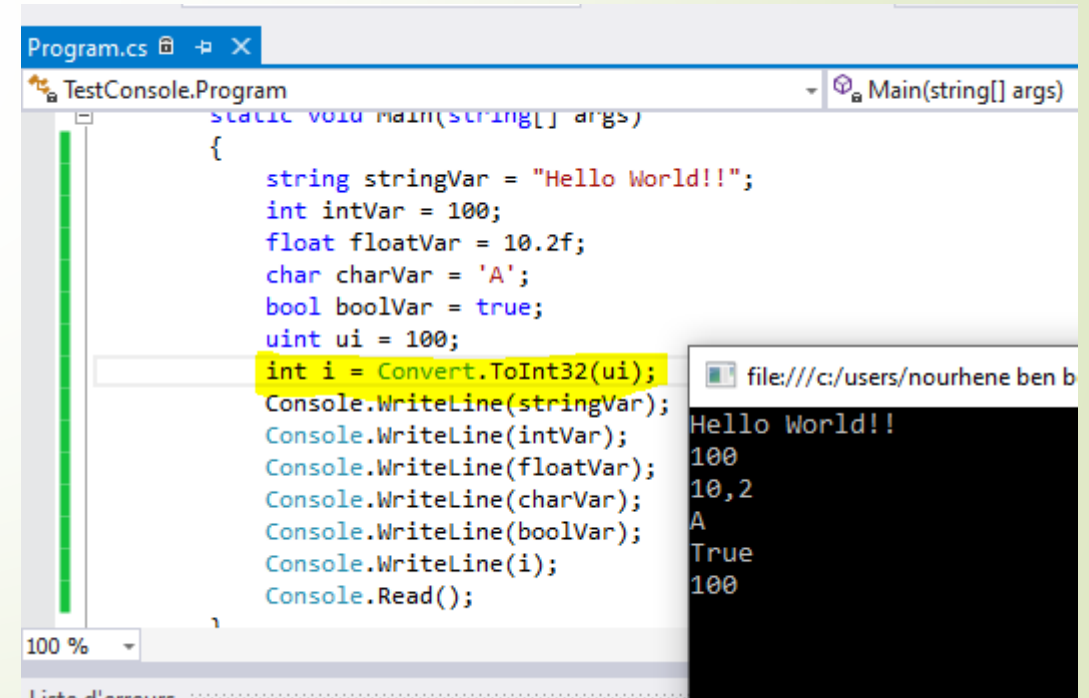
Pour effectuer un cast, spécifiez le type voulu entre parenthèses devant la valeur ou la variable à convertir.

Parse() method/ Convert class/ TryParse() method



```
Program.cs
TestConsole.Program
Main(string[] args)
{
    string stringVar = "Hello World!!";
    int intVar = 100;
    float floatVar = 10.2f;
    char charVar = 'A';
    bool boolVar = true;
    uint ui = 100;
    int i = (int)ui;
    Console.WriteLine(stringVar);
    Console.WriteLine(intVar);
    Console.WriteLine(floatVar);
    Console.WriteLine(charVar);
    Console.WriteLine(boolVar);
    Console.WriteLine(i);
    Console.Read();
}
```

file:///c:/users/nourhene be  
Hello World!!  
100  
10,2  
A  
True  
100



```
Program.cs
TestConsole.Program
Main(string[] args)
{
    string stringVar = "Hello World!!";
    int intVar = 100;
    float floatVar = 10.2f;
    char charVar = 'A';
    bool boolVar = true;
    uint ui = 100;
    int i = Convert.ToInt32(ui);
    Console.WriteLine(stringVar);
    Console.WriteLine(intVar);
    Console.WriteLine(floatVar);
    Console.WriteLine(charVar);
    Console.WriteLine(boolVar);
    Console.WriteLine(i);
    Console.Read();
}
```

file:///c:/users/nourhene ben b  
Hello World!!  
100  
10,2  
A  
True  
100

# Déclaration des variables et Types

Pour les chaînes de caractères, il existe des caractères spéciaux dont celui du retour à la ligne et de la tabulation:

- ❖ `\n`, `\r` : retour à la ligne
- ❖ `\t` : tabulation (pratique pour les alignements verticaux)
- ❖ `\\` : permet d'afficher un antislash
- ❖ `\"` : permet d'afficher un guillemet
- ❖ `@` : pour échapper les caractères spéciaux
  
- ❖ Concaténation des chaînes avec l'opérateur « + »



# Classe Console

- Représente les flux standard d'entrée, de sortie et d'erreur pour les applications console. Cette classe ne peut pas être héritée.
- La console est une fenêtre du système d'exploitation dans laquelle les utilisateurs interagissent avec le système d'exploitation ou avec une application de console textuelle en entrant une entrée de texte via le clavier de l'ordinateur et en lisant la sortie du texte à partir du terminal de l'ordinateur
- La sortie:
  - ❖ Avec un saut de ligne : `Console.WriteLine(expression) ;`
  - ❖ Passer une ligne : `Console.WriteLine() ;`
  - ❖ La sortie sans saut de ligne : `Console.Write(expression) ;`
- L'entrée :
  - ❖ `nom-var = Console.ReadLine () ; // nom-var est obligatoirement de type string.`
- Effacer l'écran : `Console.Clear();`



# Les operateurs du langage

## ➤ Les opérateurs de Test :

< Inferieur à (<= Inferieur ou égal )

> Supérieur à (>= Supérieur ou égal)

!= Différent de

== Est égal à

## ➤ Les opérateurs Logiques :

&& ET logique (vrai si tous les membres sont vrai)

|| OU logique (vrai si au moins un des membres est vrai)

## ➤ Attribution d'une valeur à une variable :

= Attribution

# Les instructions alternatives en C#

## ■ Instruction conditionnelle :

```
if (condition)
{ instruction;
instruction;
...}
```

## ■ Else

```
{instruction;
instruction;
...}
```

S'il n'y a qu'une instruction, les deux accolades { } ne sont pas obligatoires.

Les parenthèses autour de la condition sont obligatoires.

# Les instructions alternatives en C#

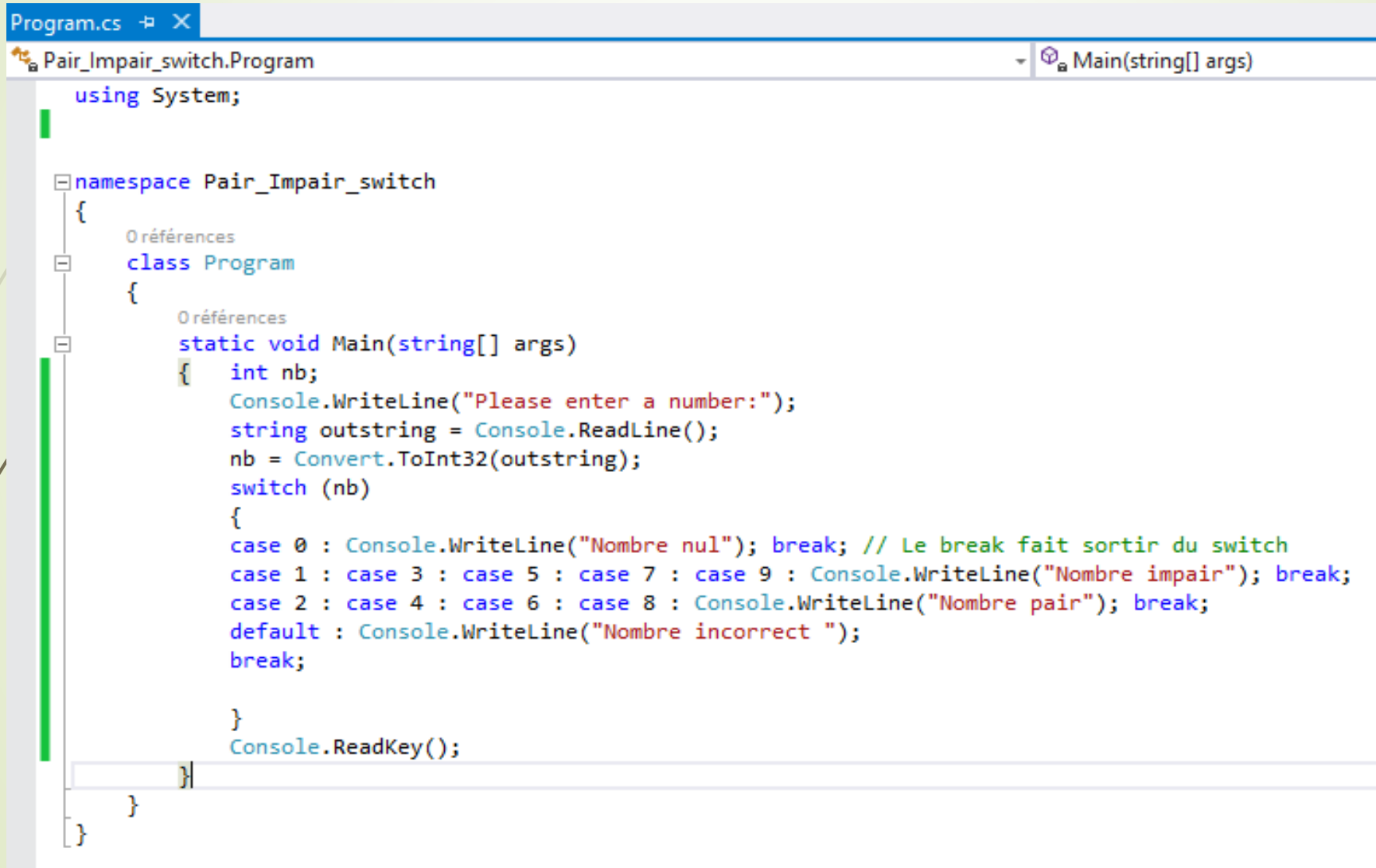
## ■ Instruction à choix multiples :

```
switch ( variable )  
{  
    case valeur 1 : instruction; instruction; ...break;  
    case valeur 2 : instruction; instruction; ...break;  
    case valeur n : instruction; instruction; ...break;  
    default : instruction; instruction; ... break;  
}
```

L'instruction `break;` est obligatoire sinon les instructions de tous les cas suivants seront exécutées.

Les parenthèses autour de la variable sont obligatoires.

# Les instructions alternatives en C#



```
Program.cs [X]
Pair_Impair_switch.Program
Main(string[] args)

using System;

namespace Pair_Impair_switch
{
    class Program
    {
        static void Main(string[] args)
        {
            int nb;
            Console.WriteLine("Please enter a number:");
            string outstring = Console.ReadLine();
            nb = Convert.ToInt32(outstring);
            switch (nb)
            {
                case 0 : Console.WriteLine("Nombre nul"); break; // Le break fait sortir du switch
                case 1 : case 3 : case 5 : case 7 : case 9 : Console.WriteLine("Nombre impair"); break;
                case 2 : case 4 : case 6 : case 8 : Console.WriteLine("Nombre pair"); break;
                default : Console.WriteLine("Nombre incorrect ");
                break;
            }
            Console.ReadKey();
        }
    }
}
```

# Les instructions itératives

## ➤ **For** loop

```
For(int i=0; i<10;i++) {}
```

## ➤ **Foreach** loop

```
string[] names=new string[10];  
Foreach(string name in names){}
```

## ➤ **While** loop:

```
Bool dataToEnter= CheckIfUserWantsToEnterData  
While (dataToEnter)  
{.....  
dataToEnter= CheckIfUserWantsToEnterData  
}
```

## ➤ **Do** loop:

```
Do{  
} While (condition);
```

# Les instructions itératives

- L'instruction `goto` transfère le contrôle du programme directement à une instruction étiquetée.
- Une utilisation courante de `goto` consiste à transférer le contrôle à une étiquette `switch-case` ou à l'étiquette par défaut d'une instruction `switch`.
- L'instruction `goto` sert aussi à quitter des boucles fortement imbriquées.

```
int b=2;
Operation1: MessageBox.Show(b.ToString());
while (b>0)
{ b--;
  goto Operation1;
}
```



# La gestion des exceptions

- un évènement qui apparaît pendant le déroulement d'un programme et qui empêche la poursuite normale de son exécution.
- Une exception représente un problème qui survient dans un certain contexte : base de données inaccessible, mauvaise saisie utilisateur, fichier non trouvé... → une exception traduit un évènement exceptionnel et a priori imprévisible. Pour essayer de répondre à ce problème, le programmeur doit mettre en place un mécanisme de gestion des exceptions.
- La classe la plus élevée est Exception
- Deux sous-classes directes sont SystemException et ApplicationException

# La gestion des exceptions

- Les Exceptions disponibles dans CSharp sont généralement dérivées de `SystemException`. Alors que les Exceptions des utilisateurs (programmeur) devraient hériter de `ApplicationException` ou de ses sous-classes.
- En C#, nous utilisons 4 mots-clés pour gérer les exceptions:

`try`

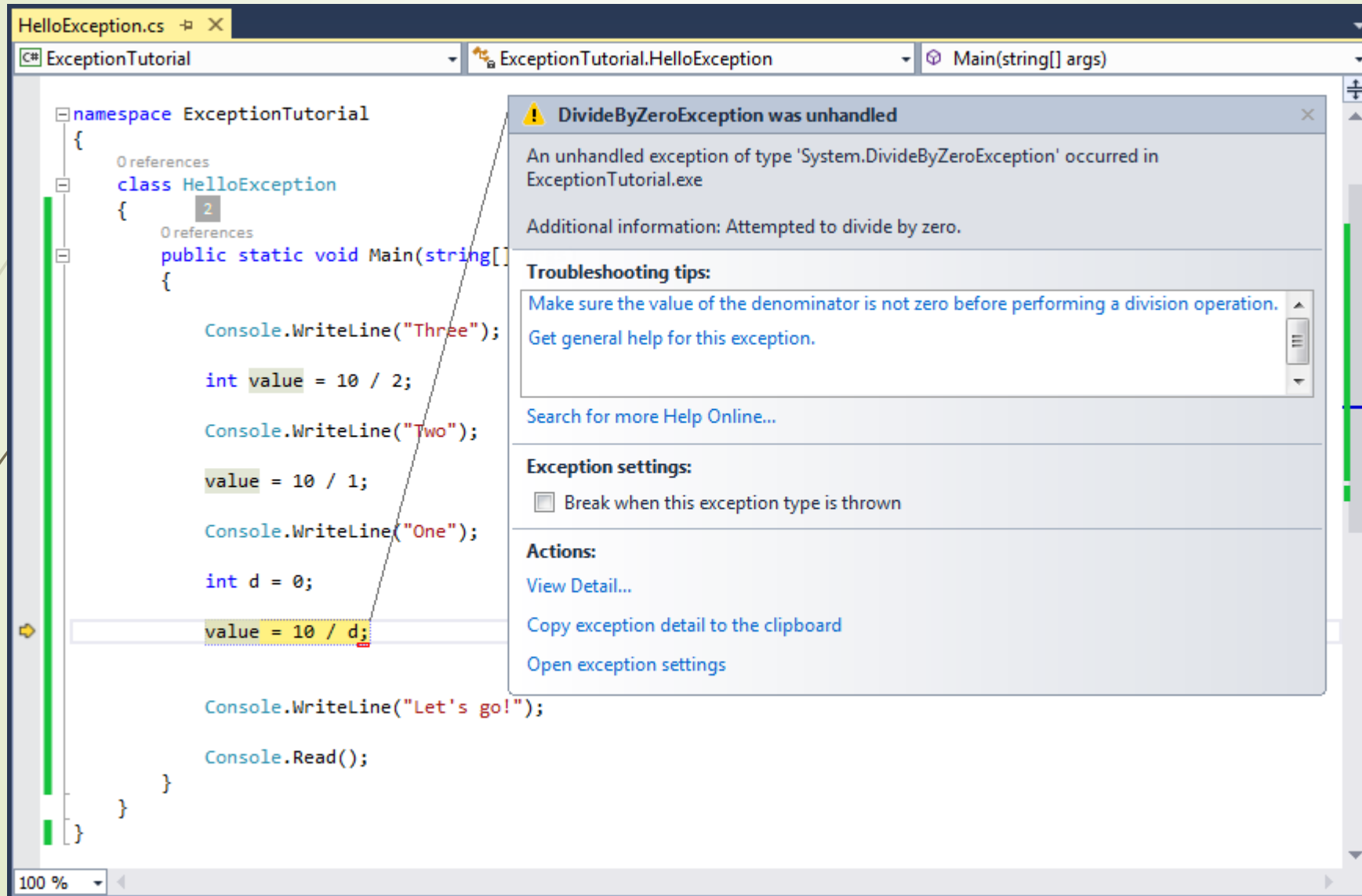
`catch`

`finally`

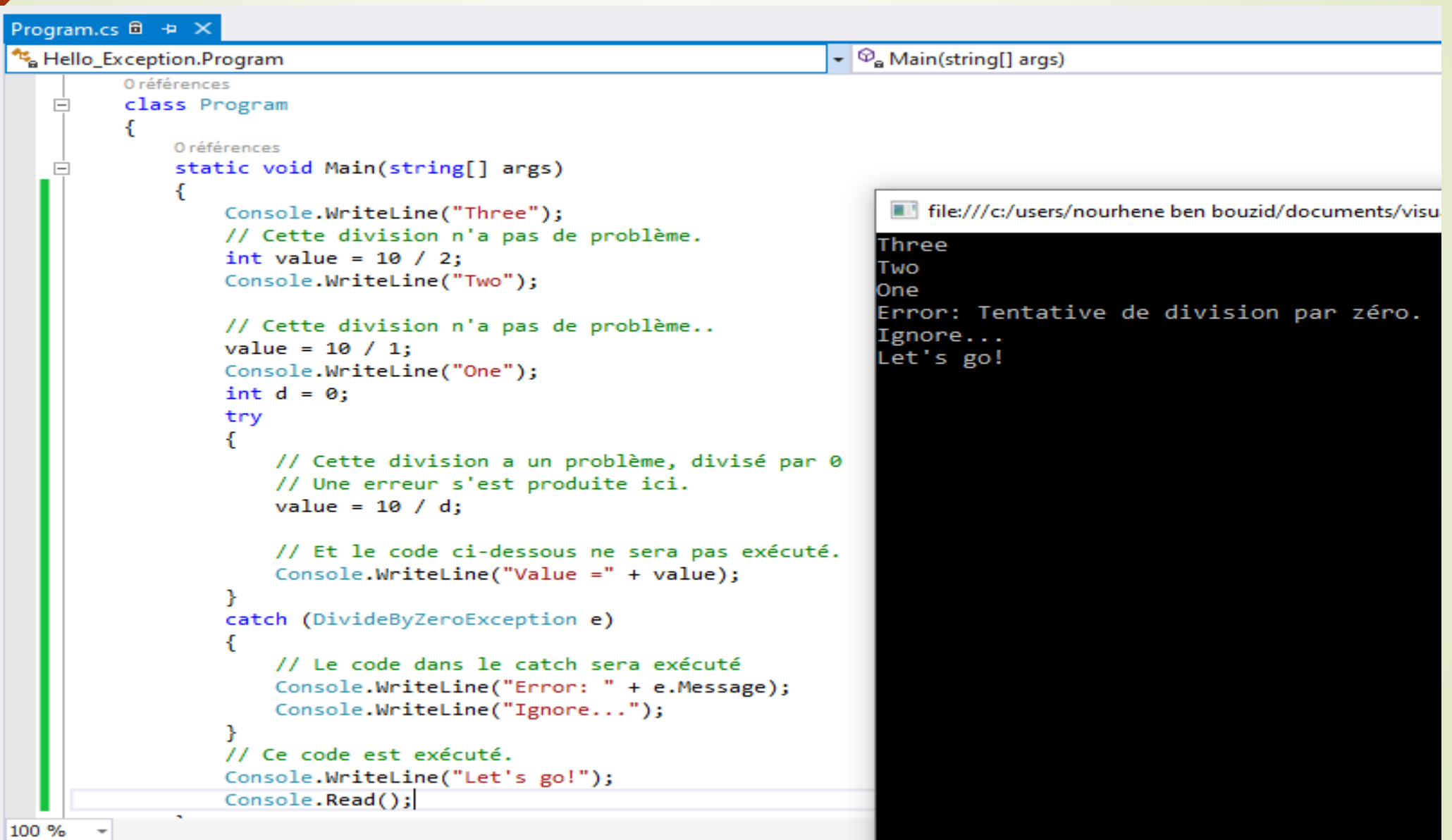
`throw`

Exception	Description
<code>System.IO.IOException</code>	Gère les erreurs d'entrée-sortie.
<code>System.DivideByZeroException</code>	Gère l'erreur générée en divisant un nombre par zéro.
<code>System.NullReferenceException</code>	Gère l'erreur générée en référençant un objet null.
<code>System.InvalidCastException</code>	Gère l'erreur générée par une conversion de type non valide.
<code>System.FieldAccessException</code>	gère l'erreur générée par un accès invalide à un attribut privé ou protégé.

# La gestion des exceptions



# La gestion des exceptions



The screenshot displays a C# IDE with a file named `Program.cs` open. The code defines a `Program` class with a `Main` method. The `Main` method contains several lines of code that demonstrate exception handling. It starts by writing "Three" and "Two" to the console, then performs a division of 10 by 2. Next, it writes "One" and performs a division of 10 by 1. Then, it attempts a division of 10 by 0, which throws a `DivideByZeroException`. This exception is caught in a `catch` block, where the error message is written to the console, and the program continues to write "Ignore..." and "Let's go!". The console output on the right shows the sequence of these operations.

```
Program.cs [Icon] [Icon] X
Hello_Exception.Program
0 références
class Program
{
    0 références
    static void Main(string[] args)
    {
        Console.WriteLine("Three");
        // Cette division n'a pas de problème.
        int value = 10 / 2;
        Console.WriteLine("Two");

        // Cette division n'a pas de problème..
        value = 10 / 1;
        Console.WriteLine("One");
        int d = 0;
        try
        {
            // Cette division a un problème, divisé par 0
            // Une erreur s'est produite ici.
            value = 10 / d;

            // Et le code ci-dessous ne sera pas exécuté.
            Console.WriteLine("Value =" + value);
        }
        catch (DivideByZeroException e)
        {
            // Le code dans le catch sera exécuté
            Console.WriteLine("Error: " + e.Message);
            Console.WriteLine("Ignore...");
        }
        // Ce code est exécuté.
        Console.WriteLine("Let's go!");
        Console.Read();
    }
}
```

file:///c:/users/nourhene ben bouzid/documents/visu  
Three  
Two  
One  
Error: Tentative de division par zéro.  
Ignore...  
Let's go!

# La gestion des exceptions

The screenshot displays the Visual Studio IDE with a C# project named 'Create\_Exception'. The code in 'Program.cs' defines a 'Program' class with a 'Main' method. Inside 'Main', a string 'str' is assigned the value 'WayToLea'. An 'if' statement checks if 'str.Length' is greater than or equal to 10. If true, it prints 'str[10]' to the console. If false, it throws a new 'Exception' with the message 'L'index se trouve en dehors des limites du tableau'. The line where the exception is thrown is highlighted in yellow.

On the right side of the IDE, the 'IntelliTrace' window shows a list of events. The first event is 'Exception: Levée : "L'index se trouve en dehors des limites du tableau"', which is highlighted.

Below the IntelliTrace window, an error dialog box is open. It features a yellow warning icon and the title 'L'exception Exception n'a pas été gérée'. The main text states: 'Une exception non gérée du type 'System.Exception' s'est produite dans Create\_Exception.exe'. Below this, it provides 'Informations supplémentaires : L'index se trouve en dehors des limites du tableau'. Under the 'Conseils de débogage' section, there is a link: 'Obtenir une aide d'ordre général pour cette exception.'. At the bottom of the dialog, there is a link: 'Rechercher de l'aide en ligne complémentaire...'.

```
namespace Create_Exception
{
    Oréférences
    class Program
    {
        Oréférences
        static void Main(string[] args)
        {
            string str = "WayToLea";
            if (str.Length >= 10)
            {
                Console.WriteLine(str[10]);
            }
            else
            {
                throw new Exception("L'index se trouve en dehors des limites du tableau");
            }
        }
    }
}
```

IntelliTrace

Toutes les catégories

Rechercher

⊞ Débogueur: Début de

⚠ Exception: Levée : "L'index se trouve en dehors des limites du tableau"

⊞ Débogueur: Arrêt à l'e

➔ Événement réel: Exce

Une exception a été int

⚠ L'exception Exception n'a pas été gérée

Une exception non gérée du type 'System.Exception' s'est produite dans Create\_Exception.exe

Informations supplémentaires : L'index se trouve en dehors des limites du tableau

Conseils de débogage :

[Obtenir une aide d'ordre général pour cette exception.](#)

[Rechercher de l'aide en ligne complémentaire...](#)



**Merci pour votre attention**