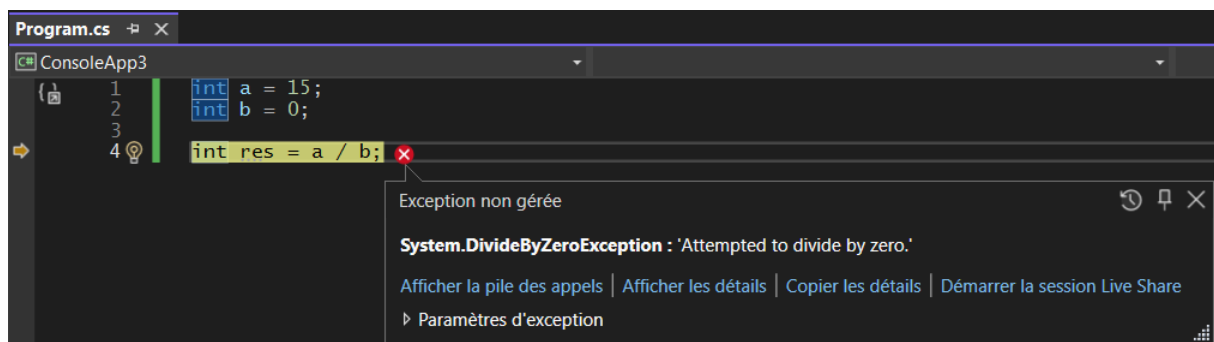


Exceptions

Une exception est un événement imprévisible qui se produit lors de l'exécution d'un programme et perturbe le flux normal des instructions. Les exceptions peuvent être générées par le runtime, le programme ou manuellement via des instructions spécifiques. Elles permettent de gérer les erreurs de manière structurée et de les traiter de façon appropriée.

Exemple de base

Prenons un exemple simple d'une exception de division par zéro :



Le système ne peut pas calculer une division par 0. Le programme « se plante », « crashe » sur cette instruction et ne peut pas continuer.

Lorsque l'on exécute notre programme dans Visual Studio comme ci-dessus, nous pouvons voir un message d'erreur explicite qui nous permet de comprendre le problème. Ce n'est pas forcément le cas quand le programme est publié sous forme d'application (exécutable).

Il est donc important de prévoir les situations exceptionnelles possibles et de prévoir du code qui les gère.

L'exemple ci-dessus est (intentionnellement) très simple. Mais il n'est pas représentatif d'une situation exceptionnelle. En effet, le code met explicitement la valeur 0 dans la variable b avant de l'utiliser pour la division. Le crash est donc prévisible à la lecture.

Cela n'est plus vrai quand notre code devient :

```
int a = 15;  
Console.WriteLine("Valeur de b:");  
int b = Convert.ToInt32(Console.ReadLine());  
  
int res = a / b;
```

Il est impossible de prévoir ce que l'utilisateur va introduire. Il y a là trois cas possibles :

1. L'utilisateur introduit une valeur entière comme attendu (p.ex : « 5 »). La conversion fonctionne et la division aussi

2. L'utilisateur introduit une valeur qui ne peut pas être convertie en entier. On a alors :

```
int a = 15;
Console.Write("Valeur de b:");
int b = Convert.ToInt32(Console.ReadLine());
int res = a / b;
```

Exception non gérée
System.FormatException : 'The input string '5,6' was not in a correct format.'

[Afficher la pile des appels](#) | [Afficher les détails](#) | [Copier les détails](#) | [Démarrer la session Live Share](#)
▸ Paramètres d'exception

3. L'utilisateur introduit la valeur « 0 ». La conversion fonctionne, mais pas la division. On a alors la même erreur qu'au début

On sera naturellement tenté de faire un test pour éviter la division par 0 :

```
int a = 15;
Console.Write("Valeur de b:");
int b = Convert.ToInt32(Console.ReadLine());

if (b != 0)
{
    int res = a / b;
}
else
{
    Console.WriteLine("Division par 0 impossible !!!");
}
```

Très bien, mais cela ne résoud pas le cas de la conversion !

Qu'à cela ne tienne ! .NET fournit des outils pour tester avant de convertir :

```
int a = 15;
Console.Write("Valeur de b:");
int b;
string input = Console.ReadLine();
if (int.TryParse(input, out b))
{
    if (b != 0)
    {
        int res = a / b;
    }
    else
    {
        Console.WriteLine("Division par 0 impossible !!!");
    }
}
else
{
    Console.WriteLine("Valeur incorrecte");
}
```

Voilà du code à toute épreuve !

Mais quelle quantité de code pour gérer un cas relativement simple et, qui plus est, normalement exceptionnel.

De plus à chaque possibilité d'erreur supplémentaire, on va devoir ajouter un niveau de « if ».

Le mécanisme d'exception permet d'adopter une approche optimiste en se disant que tout va bien se passer et en regroupant la gestion des erreurs.

Structure de contrôle d'une exception

Une exception se gère au moyen de la structure dite « try/catch », qui comporte trois blocs :

1. **Try** : Le bloc try contient le code « optimiste », il représente le déroulement normal, attendu du programme. Il n'a donc plus besoin de certains tests comme vu plus haut.
2. **Catch** : Un bloc catch capture les exceptions et permet de traiter les erreurs d'un type choisi. Il peut y en avoir plusieurs, chacun traitant un type d'exception.
3. **Finally** : Ce bloc – s'il est présent – est toujours exécuté, qu'une exception ait été levée ou non, ce qui permet d'effectuer des nettoyages nécessaires (fermeture de fichiers, libération de ressources, etc.).

En code :

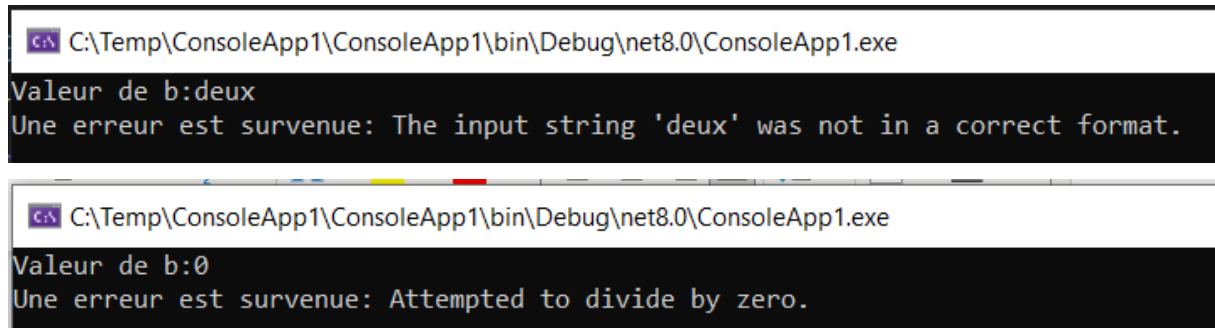
```
int a = 15;
Console.Write("Valeur de b:");
int b;

try // code optimiste
{
    string input = Console.ReadLine();

    // On prend le pari que la valeur est au bon format
    b = int.Parse(input) ;

    // On prend le pari que la valeur est non nulle
    int res = a / b;
}
catch (Exception e)
{
    Console.WriteLine($"Une erreur est survenue: {e.Message}");
}
```

Résultats d'exécution:



```
C:\Temp\ConsoleApp1\ConsoleApp1\bin\Debug\net8.0\ConsoleApp1.exe
Valeur de b:deux
Une erreur est survenue: The input string 'deux' was not in a correct format.

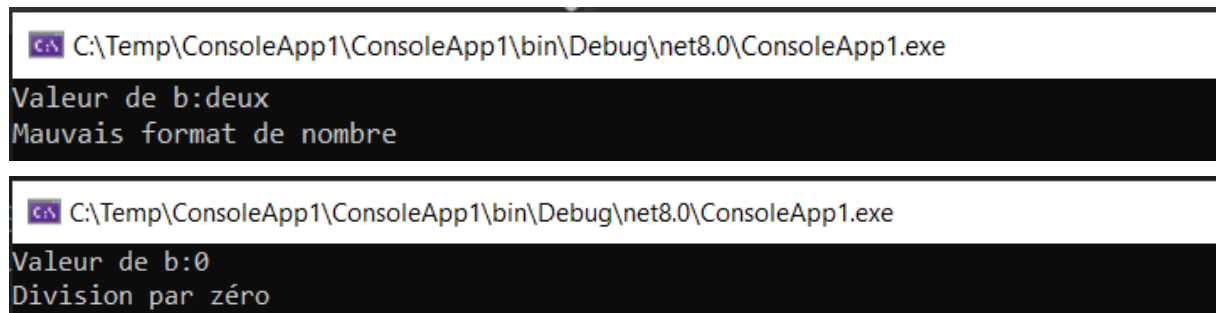
C:\Temp\ConsoleApp1\ConsoleApp1\bin\Debug\net8.0\ConsoleApp1.exe
Valeur de b:0
Une erreur est survenue: Attempted to divide by zero.
```

Si on veut exécuter des actions spécifiques en fonction du type d'erreur, on multiplie les blocs catch :

```
int a = 15;
Console.WriteLine("Valeur de b:");
int b;

try // code optimiste
{
    string input = Console.ReadLine();
    b = int.Parse(input);           // On prend le pari que la valeur
    est dans le bon format         est dans le bon format
    int res = a / b;               // On prend le pari que la valeur
    est non nulle                  est non nulle
}
catch (FormatException e)
{
    Console.WriteLine("Mauvais format de nombre");
}
catch (DivideByZeroException e)
{
    Console.WriteLine("Division par zéro");
}
```

Résultats d'exécution :



```
C:\Temp\ConsoleApp1\ConsoleApp1\bin\Debug\net8.0\ConsoleApp1.exe
Valeur de b:deux
Mauvais format de nombre

C:\Temp\ConsoleApp1\ConsoleApp1\bin\Debug\net8.0\ConsoleApp1.exe
Valeur de b:0
Division par zéro
```

Exceptions courantes en C#

- `NullReferenceException` : Tentative d'accès à un objet `null`.
- `IndexOutOfRangeException` : Tentative d'accès à un indice de tableau hors des limites.
- `DivideByZeroException` : Tentative de division par zéro.
- `InvalidOperationException` : Opération illégale dans l'état actuel d'un objet.

Lancer des exceptions

Votre code va devoir prendre en charge des exceptions produites (« lancées » ou « levées ») par des composants que vous utilisez.

Mais votre code peut aussi lancer des exceptions à son tour !

Cela se fait grâce au mot-clé « throw ».

Exemple : on décide que ce n'est pas à nous (=notre code) de décider ce qu'il faut faire si l'utilisateur ne rentre pas des valeurs acceptables. On pourra donc faire :

```
class Blob
{
    private int _a = 15;

    float UserInput()
    {
        Console.WriteLine("Valeur de b:");
        int b;

        try // code optimiste
        {
            string input = Console.ReadLine();
            b = int.Parse(input);
            return _a / b;
        }
        catch (Exception e)
        {
            throw new Exception("Pas content!");
        }
    }
}
```

Ce sera maintenant au code qui utilise la classe Blob de faire du try/catch !