Exceptions

Lors de la création d'un programme, le principal souci du développeur est de savoir si son code est fonctionnel. Il va devoir donc prévoir les potentiels sources de problèmes.

Une exception est un événement inattendu qui se produit lors de l'exécution d'un programme. Elle peut être entrainer par différentes choses, une erreur de programmation, une erreur avec les données que l'on va introduire dans notre programme, une erreur matérielle (pas assez de mémoire vive), et d'autres types d'erreurs.

Pour limiter les événements qui vont interrompre ou modifier le fonctionnement de notre programme nous pouvons coder ce dernier de manière qu'il gère les erreurs potentielles, mais cela n'est pas tout le temps possible.

Nous pouvons notamment le faire en appliquant des conditions, du genre « si la chaine de caractères est composée de 10 caractères » fait cela ou affiche un message d'erreur.
On pourrait donc définir cela comme des erreurs gérées et les traiter dans le flux continu du programme.

Exemple

Prenons l'exemple du code suivant :

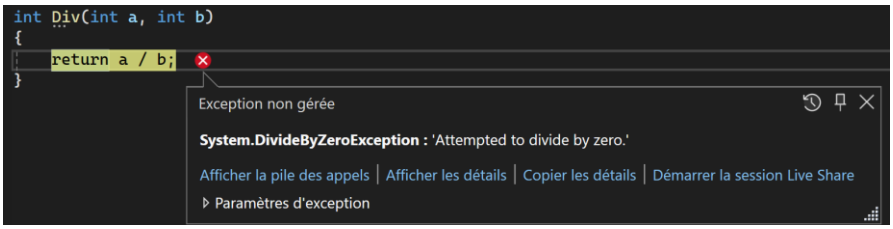
```
public class OperationMath
{
    /// <summary>
    /// Main
    /// </summary>
    static void Main(string[] args)
    {
        Result = Division(0,2)

    }

    /// <summary>
    /// Divide two numbers
    /// </summary>
    /// <param name="a">First number</param>
    /// <param name="b">Second number</param>
    public int Division(int a, int b)
    {
        return a / b;
    }
}
```

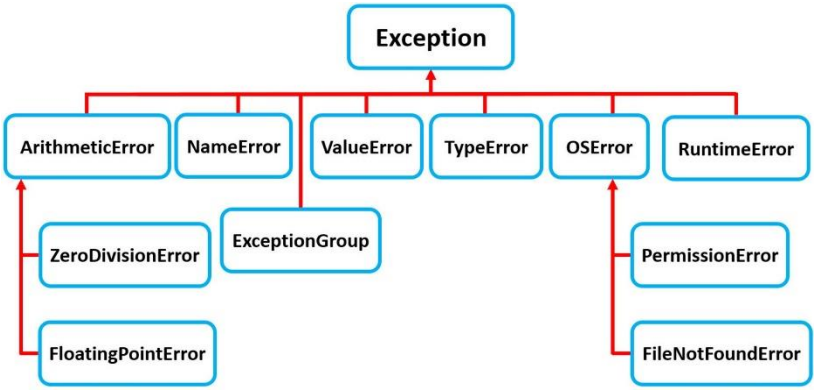

Dernière modification le 19 septembre 2024
par : Anass Benfares

Mon IDE va lever une exception et m'afficher sa nature :



Il est vrai que nous pourrions coder notre programme de manière à ne pas laisser la division être effectué si nous mettons comme diviseur un 0 mais d'autres Exceptions pourraient également se produire.

Nous avons donc un certain nombre d'exceptions existantes que nous pouvons rencontrer dans notre code.



Un exemple de différentes exceptions existantes.

Source : https://realpython.com/python-catch-multiple-exceptions/

Pour isoler les exceptions nous pouvons nous appuyer sur des structures de contrôle comme le **Try**, **Catch** et **Finally**.

Try, Catch & Finally

Il existe donc des cas ou les exceptions ne peuvent pas être gérés spécifiquement dans le code, cela signifie que lors de l'exécution de notre programme, il est possible qu'une Exception soit levée alors que nous pensions avoir pris en considération toutes les problématiques.
Ce que nous allons faire c'est utiliser la méthode **Try**,**Catch**.
Cela nous permettra d'isoler le code qui potentiellement va lever une exception.

La méthode **Try** indiquera au compilateur que ce qui se trouve à l'intérieur de la méthode peut être source de problème et qu'en cas d'erreur, il ne faut pas effectuer une interruption du programme, mais à la place effectuer le bloc **Catch**.

Cela peut être le cas pour cet exemple :

```
using System;
using System.IO;
public class Program
{
    /// <summary>
    /// Main
    /// </summary>
    static void Main(string[] args)
    {
        Console.WriteLine("Veuillez entrer le chemin complet du fichier :");
        string filePath = Console.ReadLine();

        try
        {
            // Ouvre le fichier en mode lecture texte
            Console (var reader = File.OpenText(filePath))
            {
                // Lit et affiche tout le contenu du fichier
                Console.WriteLine(reader.ReadToEnd());
            }
        }
        catch
        {
            // Gère toutes les exceptions possibles
            Console.WriteLine("il semblerait qu'une erreur se soit produite.");
        }
    }
}
```

Nous avons également la possibilité d'afficher le message d'erreur en écrivant le bloc catch de cette forme pour avoir un peu plus de détail quant à l'Exception survenue

```
catch (Exception ex)
        {
            // Gère toutes les exceptions possibles
        Console.WriteLine($"Une erreur est survenue : {ex.Message}");
        }
```

Cela permettra d'avoir le détail quant à l'exception lors de l'exécution du bloc Catch.

Nous pouvons reprendre l'exemple de la division pour voir en pratique les différents types d'Exceptions possible lors d'une saisie de valeur pour la division.

```
public class OperationMath
{
    /// <summary>
    /// Main
    /// </summary>
    static void Main(string[] args)
    {
        try
        {
        Console.WriteLine("Entrez un chiffre pour la divison" );
        a = Int32.Parse(Console.ReadLine());

        int result = division(a, 1);
        Console.WriteLine(result);

        }
        catch (DivideByZeroException ex)
        {
                Console.WriteLine(ex.Message);
        }
        catch (FormatException ex)
        {
                Console.WriteLine(ex.Message);
        }

    }

    /// <summary>
    /// Divide two numbers
    /// </summary>
    /// <param name="a">First number</param>
    /// <param name="b">Second number</param>
    public int Division(int a, int b)
    {
        return a / b;
    }
}
```

Dans ce cas, nous pourrons gérer l'erreur potentiel de la saisie de donnée et la gérer différemment que l'erreur de division par 0.

La méthode **Finally** quant a elle n'est pas obligatoire lors de l'utilisation du **Try**, **Catch.**

Elle s'exécutera simplement de toute façon, que l'Exception ou non soit levée.

```
public class OperationMath
{
    /// <summary>
    /// Main
    /// </summary>
    static void Main(string[] args)
    {
        try
        {
        Console.WriteLine("Entrez un chiffre pour la divison");
        a = Int32.Parse(Console.ReadLine());

        int result = division(a, 1);
        Console.WriteLine(result);

        }
        catch (DivideByZeroException ex)
        {
                Console.WriteLine(ex.Message);
        }
        catch (FormatException ex)
        {
                Console.WriteLine(ex.Message);
        }
        finally
        {
                Console.WriteLine("Éxecution du code terminé");
        }
    }

    /// <summary>
    /// Divide two numbers
    /// </summary>
    /// <param name="a">First number</param>
    /// <param name="b">Second number</param>
    public int Division(int a, int b)
    {
        return a / b;
    }
}
```

Voici la manière dont la console va s'afficher en fonction des Exceptions:

```
Entrez un chiffre pour la divison:
0
Attempted to divide by zero.
Éxecution du code terminé
```

Erreur de division par zéro

```
Entrez un chiffre pour la divison:
Bonjour
The input string 'Bonjour' was not in a correct format.
Éxecution du code terminé
```

Erreur de format