Chapitre 4 : La gestion des exceptions

4.1 Les exceptions standards

De nombreuses fonctions C# sont susceptibles de générer des exceptions, c'est à dire des erreurs. Lorsqu'une fonction est susceptible de générer une exception, le programmeur devrait la gérer dans le but d'obtenir des programmes plus résistants aux erreurs : il faut toujours éviter le "plantage" sauvage d'une application.

La gestion d'une exception se fait selon le schéma suivant :

```
try{
code susceptible de générer une exception
} catch (Exception e){
traiter l'exception e
}
Instruction suivante
```

Si la fonction ne génère pas d'exception, on passe alors à instruction suivante, sinon on passe dans le corps de la clause catch puis à instruction suivante. Notons les points suivants :

- e est un objet de type Exception ou dérivé.
- On peut être plus précis en utilisant des types tels que IndexOutOfRangeException, FormatException, SystemException, etc… : il existe plusieurs types d'exceptions.
- En écrivant catch (Exception e), on indique qu'on veut gérer toutes les types d'exceptions. Si le code de la clause try est susceptible de générer plusieurs types d'exceptions, on peut vouloir être plus précis en gérant l'exception avec plusieurs clauses catch :

```
try{
code susceptible de générer les exceptions
} catch ( IndexOutOfRangeException e1){
traiter l'exception e1
}
} catch ( FormatException e2){
traiter l'exception e2
}
Instruction suivante
```

- On peut ajouter aux clauses try/catch, une clause finally :

```
try{
code susceptible de générer une exception
} catch (Exception e){
traiter l'exception e
}
finally{
code exécuté après try ou catch
}
```

Instruction suivante

Qu'il y ait exception ou pas, le code de la clause *finally* sera toujours exécuté.

- Dans la clause *catch*, on peut ne pas vouloir utiliser l'objet *Exception* disponible. Au lieu d'écrire *catch (Exception e){..}*, on écrit alors *catch(Exception){...}* ou plus simplement *catch {...}*.
-
- La classe *Exception* a une propriété **Message** qui est un message détaillant l'erreur qui s'est produite. Ainsi si on veut afficher celui-ci, on écrira :

catch (Exception ex){ Console.WriteLine("L'erreur suivante s'est produite : {0}",ex.Message); ... }//catch

- La classe *Exception* a une méthode **ToString** qui rend une chaîne de caractères indiquant le type de l'exception ainsi que la valeur de la propriété *Message*. On pourra ainsi écrire :

catch (Exception ex){ Console.WriteLine("L'erreur suivante s'est produite : {0}", ex.ToString()); ... }//catch

On peut écrire aussi :

catch (Exception ex){ Console.WriteLine("L'erreur suivante s'est produite : {0}",ex); ... }//catch Le compilateur va attribuer au paramètre {0}, la valeur *ex.ToString()*.

L'exemple suivant montre une exception générée par l'utilisation d'un élément de tableau inexistant :

```
1. using System ;

2. namespace Chapitre4 {
3. class Program {
4. static void Main(string[] args) {
5. // déclaration & initialisation d'un tableau
6. int[] tab = { 0, 1, 2, 3 };
7. int i;
8. // affichage tableau avec un for
9. for (i = 0; i < tab.Length; i++)
10.               Console.WriteLine("tab[{0}]={1}", i, tab[i]);
11.      // affichage tableau avec un for each
12.      foreach (int élmt in tab) {
13.               Console.WriteLine(élmt);
14.      }
15.      // génération d'une exception
16.      try {
17.               tab[100] = 6;
18.      } catch (Exception e) {
19.      Console.Error.WriteLine("L'erreur suivante s'est produite : " +
   e);
20.      }//try-catch
21.      finally {
22.             Console.WriteLine("finally ...");
23.             }
```

```
24.       }
25.       }
26.       }
```

Ci-dessus, la ligne 18 va générer une exception parce que le tableau *tab* n'a pas d'élément n° 100. L'exécution du programme donne les résultats suivants :
tab[0]=0
tab[1]=1
tab[2]=2
tab[3]=3
0
1
2
3
L'erreur suivante s'est produite : System.IndexOutOfRangeException: L'index se trouve en dehors
des limites du tableau.
à Chapitre.cs:ligne 7
finally ...

ligne 9 : l'exception [System.IndexOutOfRangeException] s'est produite
ligne 11 : la clause *finally* (lignes 22) du code a été exécutée.

Voici un autre exemple où on gère l'exception provoquée par l'affectation d'une chaîne de caractères à un variable de type entier lorsque la chaîne ne représente pas un nombre entier :

```
1. using System;
2.
3. namespace Chapitre4 {
4. class Program {
5.    static void Main(string[] args) {
8.    // On demande le nom
9.    Console.Write("Nom : ");
10.   // lecture réponse
11.   string nom = Console.ReadLine();
12.   // on demande l'âge
13.   int age = 0;
14.   bool ageOK = false;
15.   while (!ageOK) {
16.   // question
17.         Console.Write("âge : ");
18.   // lecture-vérification réponse
19.   try {
20.         age = int.Parse(Console.ReadLine());
21.         ageOK = age>=1;
22.   } catch {
23.         }//try-catch
24.   if (!ageOK) {
25.         Console.WriteLine("Age incorrect, recommencez...");
26.   }
27.   }//while
28.   // affichage final
29.   Console.WriteLine("Vous vous appelez {0} et vous avez {1}
an(s)",nom,age);
30. }
31. }
```

```
32. }
```

- lignes 24-26 : un message d'erreur est émis si l'âge est incorrect.

- lignes 15-27 : la boucle de saisie de l'âge d'une personne

- ligne 20 : la ligne tapée au clavier est transformée en nombre entier par la méthode int.Parse. Cette méthode lance une exception si la conversion n'est pas possible. C'est pourquoi, l'opération a été placée dans un try / catch.

- lignes 22-23 : si une exception est lancée, on va dans le catch où rien n'est fait. Ainsi, le booléen ageOK positionné à false, ligne 14, va-t-il rester à false.

- ligne 21 : si on arrive à cette ligne, c'est que la conversion string -> int a réussi. On vérifie cependant que l'entier obtenu est bien supérieur ou égal à 1

Quelques résultats d'exécution :

```
Nom : Slim
âge : 23
Vous vous appelez dupont et vous avez 23 an(s)
Nom : Slim
âge : x
Age incorrect, recommencez...
âge : -4
Age incorrect, recommencez...
âge : 12
Vous vous appelez Slim et vous avez 12 an(s)
```

4.2 Les exceptions personnalisées

Le langage C# offre la possibilité de définir des exceptions personnalisées et de les lever en utilisant le mot clé throw :

Exemple :

1. Définir l'exception AgeNonValideException

```
public class AgeNonValideException:Exception
{
    public AgeNonValideException(string s):base(s)
    {

    }
}
```

2. Ecrire une fonction qui lève cette exception

```
void testage(int a)
{
    if ((a < 1) || (a > 100))
        throw new AgeNonValideException("Age non valide");
}
```

3. Traiter l'exception

```
  private void button1_Click(object sender, EventArgs e)
  {
      int x=0;
      try
      {
          x = Int32.Parse(textBox1.Text);
          testage(x);
          MessageBox.Show(x.ToString());
      }
      catch (AgeNonValideException e2)
      {
          MessageBox.Show(e2.Message);
      }
      catch (Exception e1)
      {
          MessageBox.Show("L'erreur suivante est survenue : " + e1);
      }
      finally
      {
          MessageBox.Show("C'est la fin");
      }
}
```