La gestion des exceptions

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
}
```

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. int[] tab = { 0, 1, 2, 3 };

2. foreach (int élmt in tab) {
3. Console.WriteLine(élmt);
4. }

5. ////try-catch ////génération d'une exception
6. try {
7. tab[100] = 6;
8. } catch (Exception e) {
9. Console.Error.WriteLine("L'erreur suivante s'est produite : " +e);
10.   }
11.   finally {
12.   Console.WriteLine("finally ...");
13.   }
14.   Console.ReadKey();
```

Ci-dessus, la ligne 7 va générer une exception parce que le tableau *tab* n'a pas d'élément n° 100. L'exécution du programme donne les résultats suivants :

```
0
1
2
3
L'erreur suivante s'est produite : System.IndexOutOfRangeException: L'index se trouve en dehors des limites du tableau.
   à LesExceptions.Program.Main(String[] args) dans c:\users\othman\documents\visual studio 2010\Projects\LesExceptions\LesExceptions\Program.cs:ligne 20
finally ...
```

ligne 8 : l'exception [System.IndexOutOfRangeException] s'est produite
ligne 12 : 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. string nom = "";
2. int age = 0;
3. try
4. {
5. Console.WriteLine("Donner le Nom : ");
6. nom = Console.ReadLine();
7. Console.WriteLine("Donner l'âge : ");
8. age = int.Parse(Console.ReadLine());
9. Console.WriteLine("Votre Nom est: {0} et Votre age est: {1}an(s)",
   nom, age);

10.      }
11.      catch (Exception e)
12.      {
13.      Console.WriteLine("ERREUR :: " + e.Message);
14.      }
```

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 :

Définir l'exception AgeNonValide

```
1.      class AgeNonValide :Exception
2.      {
3.          public AgeNonValide(string notremessage)
4.              : base(notremessage)
5.          {
6.          }
7.      }
```

2. Ecrire une fonction qui lève cette exception

```
void testage(int ageentre)
{
    if ((ageentre < 1) || (ageentre > 100))
        throw new AgeNonValide("Intervale d'age non valide");
}
```

3. Traiter l'exception

```
1. string nom = "" ;
2. int age = 0 ;
3. try
4. {
5. Console.WriteLine("Donner le Nom : ");
6. nom = Console.ReadLine();
7. Console.WriteLine("Donner l'âge : ");
8. age = int.Parse(Console.ReadLine());
9. if ((age < 1) || (age > 100))
10.   throw new AgeNonValide("Intervale d'age non valide");
11.   Console.WriteLine("Votre Nom est: {0} et Votre age est: {1}an(s)",
   nom, age);
12.   }
13.   catch (AgeNonValide a)
14.   {
15.   Console.WriteLine(a.Message);
16.   }
17.   catch(FormatException fe)
18.   {
19.   Console.WriteLine( fe.Message);
20.   }
21.   catch (Exception autreexception)
22.   {
23.   Console.WriteLine(autreexception.Message);
24.   }
```