Traitements d'erreur et exceptions

I. Introduction : dans nos programmes, nous essayons d'éviter que des erreurs se produisent au « Runtime » en utilisant diverses stratégies :
   - Analyse du projet.
   - Utilisation de l'algorithmique.
   - Utilisation de modèle mathématique prouvant le bon fonctionnement.
   - Code « anti-idiot ».

Malgré tout, des erreurs peuvent se produire. Certaines de ces erreurs peuvent provoquer un dysfonctionnement du système tel que une fermeture prématuré du programme. (Ce sont les moins graves !)

NB : un plantage (pe boucle sans fin ou deadlock) ne provoque pas d'erreur au runtime. C'est une erreur algorithmique.

II. Exemple de la division par zéro :
    (du processeur à l'utilisateur)

interruption DividebyZero :

Créer un projet en mode Console avec le code :

```
int i;
Console.WriteLine("Entrée un nombre :");
i = Convert.ToInt32( Console.ReadLine());
Console.WriteLine(" 100 / i = {0:D}", 100 / i);
Console.ReadKey();
```

Tester le programme en donnant le nombre 0.
Que se passe-t-il ?

Cette opération provoque une interruption du processeur : DivideByZero.
Cette interruption est traité par le gestionnaire d'erreur par défaut de la bibliothèque Visual. Mais ce gestionnaire arrête notre programme.

Notre objectif : éviter que les erreurs arrêtent prématurément notre programme.

III. Que faire pour que éviter d'arrêter le programme :

1. Prévenir : code anti-idiot.

```
int i;
 Console.WriteLine("Entrée un nombre :");
 i = Convert.ToInt32(Console.ReadLine());
 if (i!=0)    // prévenir
    Console.WriteLine(" 100 / i = {0:D}", 100 / i);
 else
    Console.WriteLine(" erreur de division par zéro, le nom entrée est incorrect
                                        !");  // si i == 0
 Console.ReadKey();
```

nécessite d'énumérer tout les cas qui peuvent produire l'erreur.

2. Ajouter un gestionnaire d'erreur qui remplacera le gestionnaire par défaut.

```
int i;
 Console.WriteLine("Entrée un nombre :");
 i = Convert.ToInt32( Console.ReadLine());
 try
 {  // essayer
    Console.WriteLine(" 100 / i = {0:D}", 100 / i);
 }
 catch (DivideByZeroException e)  // si cette exception se produit quelque part
                    dans l'essai
 {Console.WriteLine(" erreur de division par zéro, le nombre entrée est
                    incorrect !");  // ce que l'on fait en cas
                                                d'erreur
 }
 Console.ReadKey();
```

DivideByZeroException est une classe dérivant de la classe Exception.
Catch change la destination de l'exception.

L' avantage est que l'exception peut se produire à l'intérieure d'un appel de fonction :

```
static int SousFonction(int i)
        {
           return 100 / i;   // si l'erreur se produit ici le programme sort des fonctions
// imbriquées et va à l'instruction contenu dans le premier catch trouvé
// et ce “sans passer par la case return”.
        }
        static int Fonction(int i)
        {
           return SousFonction(i);
        }
     static void Main(string[] args)
        {
           int i;
           Console.WriteLine("Entrée un nombre :");
           i = Convert.ToInt32( Console.ReadLine());
           try
           {  // essayer
              Console.WriteLine(" 100 / i = {0:D}", Fonction( i));
           }
           catch (DivideByZeroException e)  // si cette exception se produit quelque part
                                                 dans l'essai
           {
              Console.WriteLine(" erreur de division par zéro, le nombre entrée est
                                  incorrect !");  // ce que l'on fait en cas d'erreur
           }
           finally
           {
              Console.WriteLine("Toujours exécuté, En Fin de Compte");
           }
           Console.ReadKey();
        }
```

Le programme reprend à l'endroit où est placé le catch, ce qui est impossible avec un simple “if”.

IV. Interception d'une exception :

1. try, catch, finally.

Try indique la portée dans laquelle l'exception définie dans le catch sera gérée.
Finally permet d'exécuter un bloc d'instruction dans tout les cas.

```
static void Main(string[] args)
    {
      int i;
      Console.WriteLine("Entrée un nombre :");
      i = Convert.ToInt32( Console.ReadLine());
      try
      {  // essayer
        Console.WriteLine(" 100 / i = {0:D}", 100 / i);
      }
      catch (DivideByZeroException e)  // si cette exception se produit quelque part
                                          dans l'essai
      {
        Console.WriteLine(" erreur de division par zéro, le nombre entrée est
                            incorrect !");  // ce que l'on fait en cas d'erreur
      }
      finally
      {
        Console.WriteLine(" En Fin de Compte");
      }
      Console.ReadKey();
    }
```

2. Classe Exception : se placer sur DivideByZeroException et faire atteindre la définition.

3. l'instruction « throw » permet de provoquer une exception.

Modifier SousFonction pour déclencher l'exception sans division.

```
static int SousFonction(int i)
{
  throw new DivideByZeroException();
  return i;
}
```

4. Plusieurs catch pour gérer plusieurs exceptions :

```
try
{  // essayer
   Console.WriteLine(" 100 / i = {0:D}", Fonction(i));
}
catch (DivideByZeroException e)  // si cette exception se produit quelque part
         dans l'essai
{
   Console.WriteLine(" erreur de division par zéro, le nombre entrée est incorrect !"
      + e.Message);  // ce que l'on fait en cas d'erreur
}
catch (DllNotFoundException e)
{
   Console.WriteLine(e.Message);
}
```

5. « using » : Utiliser de l'instruction C# using pour gérer correctement les objets systèmes tel que les flux (Fichiers). Ainsi, dans tout les cas, vous êtes sure que les ressources utilisées sont libérées même si une exception se produit.
   Les objets utilisés doivent implémenter l'interface IDisposable.

Exemple :

```
try {
    using (StreamReader fichier = new StreamReader("fichier.txt"))
        {
          while (!fichier.EndOfStream)
          {
            ... traitement utilisant le fichier
          }
        }
    }
     catch (FileNotFoundException e)
     {
       MessageBox.Show(" fichier.txt non trouvé !");
     }
```

ici “Using” garantie l'appel de la méthode “Dispose()” des “StreamReader” qui appelle “close” et libère les ressources.

Sans “using” il faudrait écrire :

```
    StreamReader fichier;
      try
      {
       fichier = new StreamReader("fichier.txt");
       while (!fichier.EndOfStream)
         {// lecture
         }
      }
      catch (FileNotFoundException e)
      {
        MessageBox.Show(" Fichier \"fichier.txt\" non trouvé ");
      }
      finally
      {
        if (fichier != null)
          fichier.Dispose();
      }
```

Dans TP4 :

1. Identifier les sources possibles d'erreurs.
2. Gérer ces erreurs.