C#

Erreurs et Exceptions

SOMMAIRE

LE TRAITEMENT DES ERREURS

Les objectifs

- Repérer les instructions qui sont susceptibles de générer des exceptions
- Capturer des exceptions pour afficher les messages d'erreur adéquats.
- Générer des exceptions

Vue d'ensemble

Des erreurs d'exécution peuvent se produire lors de l'exécution d'une instruction : Si aucun traitement particulier n'est effectué, l'application s'arrête brutalement, ce qui en général, n'est guère apprécié des utilisateurs, et peut provoquer des pertes de données.

Il faut donc identifier les parties de programme ou des erreurs d'exécution peuvent se produire, et écrire du code spécifique pour les traiter.

Les techniques de traitement d'erreurs permettent en cours d'exécution de programme de détecter et de réagir à :

- Des erreurs générées par le système, comme les divisions par 0, l'utilisation d'un index de tableau invalide ou la non correspondance d'une valeur d'une énumération.
- Des erreurs générées par une fonction du programme, par exemple une valeur négative ou trop élevée pour l'âge d'une personne.

Exemple :

```
static void Main(string[] args)
{
 int a = 10, b = 0, c;
 Console.WriteLine("La division");
 c= a/b;
 Console.WriteLine(c);
}
```

Le programme est interrompu par une erreur,

L'exception DivideByZeroException n'a pas été gérée.

Le programme s'arrête brutalement, et le deuxième affichage n'est jamais exécuté.

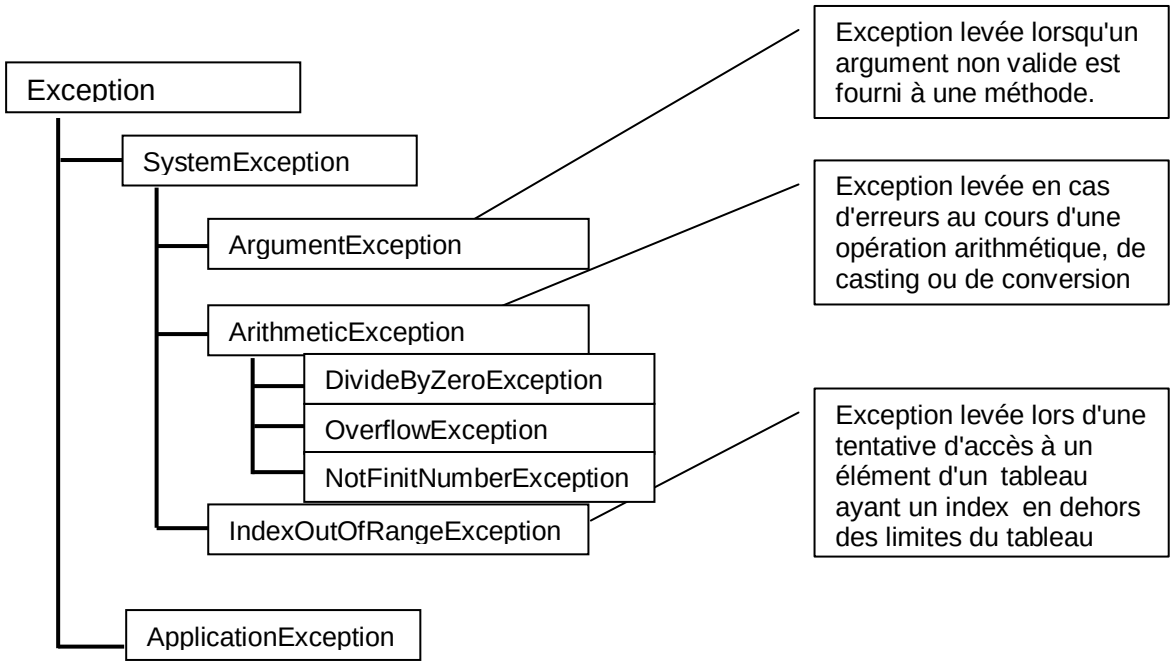
Les objets Exception

Le framework définit une hiérarchie de classes d'exception, dont la classe de base est la classe **Exception**.

Il existe deux catégories d'exceptions sous la classe de base **Exception** :

- les classes d'exceptions prédéfinies du Common Language Runtime, dérivées de **SystemException**.
- les classes d'exceptions de l'application définies par l'utilisateur, dérivées de **ApplicationException**.

Chaque classe d'exception décrit clairement l'erreur qu'elle représente.



Les clauses try et catch

L'idée est de séparer les instructions essentielles du programme, des instructions de gestion d'erreur.

Les parties de code susceptibles de lever des exceptions sont placées dans un bloc **try**, et le code de traitement de ces exceptions est placé dans un bloc **catch**.

A l'intérieur du groupe **try**, les instructions sont écrites, sans se soucier du traitement d'erreurs.

Dans notre exemple :

```
static void Main(string[] args)
{
 int a = 10, b = 0, c;
 try
 {
  Console.WriteLine("La division");
  c= a/b;
  Console.WriteLine(c);
 }
 catch (Exception)
 {
  Console.WriteLine("Erreur arithmétique");
 }
```

```
}
```

Le programme rentre dans le bloc **try**, comme si l'instruction **try** n'existait pas, et les instructions sont exécutées en séquence.

Dés qu'une erreur est détectée par le système, un objet de la classe Exception, ou classe dérivée est automatiquement créé, et le programme se débranche à la première instruction du bloc **catch** correspondant.

A l'intérieur des parenthèses de la clause catch, un nom d'objet est souvent spécifié, de manière à obtenir des informations supplémentaires au sujet de l'erreur, par exemple le champ Message de la classe Exception, qui délivre en clair le message de l'erreur.

```
catch (Exception e)
{
 Console.WriteLine("Erreur arithmétique \n" + e.Message);
}
```

Le message affiché sera :

Erreur arithmétique
Tentative de division par zéro.

Nous nous sommes servi de la classe générale d'exception pour traiter l'erreur ; de ce fait, toutes les erreurs détectées par n'importe quelle instruction afficheront ce même message… ce qui parait absurde !

Il est possible, et recommandé d'enchaîner les blocs catch chacun correspondant à un type d'exception traité.

```
static void Main(string[] args)
{
 int a = 10, b = 0, c;
 try
 {
  Console.WriteLine("La division");
  c= a/b;
  Console.WriteLine(c);
 }
 catch (ArithmeticException ae)
 {
  Console.WriteLine("Erreur arithmétique\n" + ae.Message);
 }
 catch (Exception e)
 {
  Console.WriteLine("Erreur générale\n" + e.Message);
 }
}
```

De ce fait, si aucun bloc catch spécifique n'existe, l'exception est interceptée par un bloc catch général, le cas échéant.

Il faut veiller à toujours **classer les exceptions dans les blocs catch de la plus spécifique à la plus générale**.

Cette technique permet de gérer l'exception spécifique avant qu'elle ne passe à un bloc catch plus général.

Attention : vous vous apercevrez rapidement que la mise en place d'une gestion d'erreurs affecte les performances de manière considérable …. Réservez ces traitements aux erreurs imprévisibles (vous préférerez ainsi contrôler préventivement la saisie d'un nombre en mémoire plutôt que de mettre en place un bloc **try** … **catch**).

Le groupe finally

C# fournit la clause **finally** pour entourer les instructions qui doivent être exécutées, quoi qu'il arrive : ces instructions seront donc exécutées, si le programme se termine normalement, à l'issue du bloc **try**, ou anormalement, à l'issue d'un **catch.**

Le bloc f**inally** est utile dans deux cas :

- éviter la duplication d'instructions

  des instructions devant s'exécuter à la fois en cas de fin normale et anormale trouveront leur place dans un bloc finally
- libérer des ressources après la levée d'une exception.

De manière générale, et sur un exemple :

```
using System;
…
try
{
// Séquence d'instructions correspondant à la saisie de données
// numériques, à la gestion de tableau et à l'exécution
// d'opérations arithmétiques.
}
catch (FormatException fe)
{
  Console.WriteLine("Erreur de saisie \n" + fe.Message");
}
catch (ArithmeticException ae)
{
  Console.WriteLine("Erreur arithmétique \n" + ae.Message);
}
catch (IndexOutOfRangeException ioore)
{
  Console.WriteLine("Index de tableau non valide \n" + ioore.Message).
}
catch (Exception e)
{
  // Ce message est affiché pour toutes les autres
  // exceptions susceptibles d'être générées.
  Console.WriteLine("Erreur générale \n" + e.Message);
}
finally
{
  // instructions à exécuter quelque soit l'issue
  // du programme
}
```

L’instruction throw

Dans le traitement d'une fonction, il peut être intéressant de générer une exception. Par exemple, dans l’utilisation d’un tableau, on peut générer exception lorsque l’index champ est inférieur à 1 ou supérieur au nombre de postes contenus dans le tableau.

Pour générer une exception, on utilise le mot-clé **throw** :

```
Console.WriteLine("Entrez un chiffre compris entre 1 et 9:");
if (i <0 || i>=9)
  throw new IndexOutOfRangeException(i + "n'est pas un choix
  valide");
```

L’exécution séquentielle normale du programme est interrompue, et le contrôle d’exécution est transféré sur le premier bloc **catch** capable de gérer l’exception, en fonction de sa classe.

Les instructions qui suivent le **throw** ne seront jamais exécutées.

Le choix de l'exception dépend bien sur du type de problème à traiter.

Exercices

Reprendre l’exercice 1 (manipulations de base) Les tableaux, et sécuriser le code.