**Gestion des exceptions**

**Introduction**

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.

*Exemple d’erreur (Division par zéro)*

Créez un nouveau projet avec seulement la classe **main**, et mettez-y le code suivant :

static void Main(string[] args)

 {

 int j = 20, i = 0;

 Console.WriteLine(j/i);

 Console.WriteLine("Fin du programme !");

 Console.Read() ;

 }

Vous devriez avoir un message d'erreur comme celui-ci :



Mais surtout, vous devez vous rendre compte que lorsque l'exception a été levée, **le programme s'est arrêté !**

.NET Framework fournit une classe spéciale appelée **Exception**. Une fois que le compilateur rencontre une erreur, la classe **Exception** vous permet d'identifier le type d'erreur et prendre une mesure appropriée.

Normalement, E**xception** sert la plupart du temps de classe générale des exceptions. Prévoyant divers types de problèmes qui peuvent se produire dans un programme, Microsoft a dérivé diverses classes E**xception** pour rendre cette question plus convivial. En conséquence, presque n'importe quel type d'exception que vous pouvez avoir déjà rencontré a une classe créée pour la traiter. Par conséquent, quand votre programme fait face à une exception, vous pouvez facilement identifier le type d'erreur.

Dans ce message, le nom de l'exception qui a été déclenchée est DevideByZeroException. Nous savons donc maintenant qu'une division par zéro est une DevideByZeroException. Nous allons pouvoir la capturer, et réaliser un traitement en conséquence.

**Clause try/catch**

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*.

Les exceptions ont les propriétés suivantes :

* Lorsque votre application rencontre une circonstance exceptionnelle, comme une division par zéro, une exception est générée.
* Délimitez toute instruction susceptible de lever des exceptions par un bloc **try**.
* Lorsqu'une exception se produit au sein d'un bloc **try**, le flux de contrôle est immédiatement transmis à un gestionnaire d'exceptions associé (bloc catch) s'il en existe un.
* Les erreurs sont traitées dans la section **catch**. Pour ce faire, catch s'utilise comme si c'était une méthode. Ceci signifie que, du côté droit **catch**, ouvrez une parenthèse, déclarez une variable de type exception que vous voulez traiter. Par défaut, une exception est de type **Exception**.
* Si aucun gestionnaire d'exceptions pour une exception donnée n'est présent (absence du bloc catch), le programme cesse de s'exécuter avec un message d'erreur.

Ce que je vous propose maintenant, c'est de capturer l'exception de notre division par zéro, et d'afficher un message décrivant l’erreur. Pour ce faire, tapez le code suivant dans votre **main** :

static void Main(string[] args)

 {

 int j = 20, i = 0;

 try

 {

 Console.WriteLine(j / i);

 Console.WriteLine("Fin du programme !");

 }

 catch(Exception e)

 {

 Console.WriteLine(e.Message);

 Console.WriteLine("division par zero");

 }

 Console.Read();

 }

Quand une exception se produit dans la section **try**, la compilation du code est transférée à la section **catch**. Si vous déclarez l'exception comme type **Exception**, cette classe identifiera l'erreur. Une des propriétés de la classe **Exception** s'appelle **Message**. Cette propriété contient une chaîne de caractères qui décrit le type d'erreur qui s'est produite. Vous pouvez alors accéder à cette propriété **Exception.Message** pour afficher un message d'erreur si vous voulez. Vous pouvez aussi, créer votre propre message et l'afficher à l'utilisateur.

Voici quelques une des propriétés de la classe System.Exception :

* HelpLink : fournit un lien à un fichier d'aide fournissant plus d'informations sur l'exception
* Message : c'est le texte décrivant la condition ayant générée l'erreur
* Source : contient le nom de l'application ou de l'objet qui a causé l'exception

**Quelques classes d'exceptions**

Le Framework .Net offre beaucoup de classes gérant les exceptions dont :

* [IndexOutOfRangeException](http://msdn.microsoft.com/fr-fr/library/system.indexoutofrangeexception%28v%3Dvs.90%29.aspx) : un index supérieur à la taille d'un tableau ou d'une collection est utilisé.
* [NullReferenceException](http://msdn.microsoft.com/fr-fr/library/system.nullreferenceexception%28v%3Dvs.90%29.aspx) : une propriété ou méthode d'une référence a été utilisée avant que cette référence ait eu pour valeur une instance valide.
* [ArithmeticException](http://msdn.microsoft.com/fr-fr/library/system.arithmeticexception%28v%3Dvs.90%29.aspx) : une opération produit un dépassement de capacité positif ou négatif.
* [FormatException](http://msdn.microsoft.com/fr-fr/library/system.formatexception%28v%3Dvs.90%29.aspx) : un argument ou un opérande est de format incorrect.

La classe Exception hérite de la classe Object et toutes les autres classes d'exceptions héritent directement ou indirectement de la classe Exception.

La classe Exception étant une classe de base, celle-ci traite toutes les exceptions possibles.

On peut être plus précis en utilisant des types tels que *FormatException, OverflowException,* etc…. 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* :

static void Main(string[] args)

 {

 int nombre = 0;

 Console.Write("Entrez un nombre : ");

 string chaine = Console.ReadLine();

 try {

 nombre = Int32.Parse(chaine);

 }

catch (FormatException formEx)

{ Console.WriteLine(formEx.Message); }

 catch (OverflowException overEx)

{ Console.WriteLine(overEx.Message); }

 Console.Read();

 }

* FormatException : levée si le paramètre n'est pas un entier
* OverflowException : levée si le paramètre est un entier mais que celui-ci est trop grand (ou trop petit) pour être stocké dans un type Int32.

**Clause Try/catch/finally**

On peut ajouter aux clauses *try/catch*, une clause **finally** :Qu'il y ait exception ou pas, le code de la clause *finally* sera toujours exécuté.

***try****{*

*code susceptible de générer une exception*

*}*

***catch*** *(Exception e)*

*{ traiter l'exception e }*

***Finally*** *{*

*code exécuté après try ou catch }*

 *instruction suivante*

***Remarque***

* 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 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*

**Les exceptions personnalisées**

On reprend l’exemple de la classe Personne

class Personne

 {

 private string nom;

 private string prenom;

 private int age;

 private static int nbPersonnes;

 public Personne()

 {

 nbPersonnes++;

 }

 public Personne(string nom,string prenom, int age)

 {

 this.nom = nom;

 this.prenom = prenom;

 this.Age = age;

 nbPersonnes++;

 }

 }

 public Personne(Personne p)

 {

 nom = p.Nom;

 prenom = p.Prenom;

 Age = p.Age;

 nbPersonnes++;

 }

 public int Age

 {

 get { return this.age; }

 set {age = value; }

 }

 public string Nom

 {

 get { return this.nom; }

 set { this.nom=value; }

 }

 public string Prenom

 {

 get { return this.prenom; }

 set { this.prenom = value; }

 }

 public static int NbPersonnes

 {

 get { return nbPersonnes; }

 }

 public void afficher()

 {

 Console.WriteLine("nom: " + nom + "\tprenom: " + prenom + "\tage: " + Age);

 }

L’âge d’une personne doit être positif. Pourtant si vous attribuez une valeur négative à l’âge d’une personne, cette valeur sera accéptée.

Si vous souhaitez générer une exception lors de la saisie d’un âge négatif, vous devez créer vos propres classes d'exceptions par une dérivation à partir de la classe [Exception](http://msdn.microsoft.com/fr-fr/library/system.exception%28v%3Dvs.90%29.aspx). Par exemple, le code suivant crée une classe ***ErreurAgeException*** que vous pouvez lever si, par exemple, l’âge donné pour une nouvelle personne n'est pas valide. Le constructeur de classe pour votre exception affiche un message d’erreur indiquant le type d’erreur.

class ErreurAgeException:Exception

 {

 public ErreurAgeException()

 {

 Console.WriteLine("Erreur: Age négatif");

 }

 }

Il faut par la suite indiquer là où l’exception doit être levée en utilisant le mot clé throw. Dans notre cas, une exception de type ErreurAgeException doit être levée au niveau du constructeur de la classe personne (si l’utlisateur essaye de créer une personne avec un âge négatif) ou au niveau du setter de l’attribut âge (si l’utlisateur essaye d’attribuer un âge négatif à une personne)

Il faut donc modifier le constructeur comme suit :

public Personne(string nom,string prenom, int age)

 {

 // si l’age est négatif on declenche une exception

 if (age <= 0) throw new ErreurAgeException();

 else

 {

 this.nom = nom;

 this.prenom = prenom;

 this.Age = age;

 nbPersonnes++;

 }

 }

De même, il faut modifier la proprité Age comme suit :

public int Age

 {

 get { return this.age; }

 set

 {

 if (value <= 0) throw new ErreurAgeException();

 else age = value;

 }

 }

Testons maintenant la classe ErreurAgeException

static void Main(string[] args)

 {

 Personne p = new Personne("asri", "yassine",23);

 p.afficher();

 Console.WriteLine("donner l'age");

 p.Age = int.Parse(Console.ReadLine());

 p.afficher();

 Console.Read();

 }

On remarque que si l’utilisateur saisit un age négatif l’exception suivante se produit



Pour gérer cette exception, il faut utiliser la clause try/catch, et donc le code devient :

static void Main(string[] args)

 {

 Personne p = new Personne("asri", "yassine",23);

 p.afficher();

 Console.WriteLine("donner l'age");

 try

 {

 p.Age = int.Parse(Console.ReadLine());

 }

 catch (ErreurAgeException e)

 {

 Console.WriteLine(e.Message);

 }

 p.afficher();

 Console.Read();

 }