Les Exceptions

Julien CONDOMINES | StJoSup | 18 mars 2022

Table des matières

I-Introduction

Les exceptions sont mises en place pour gérer les situations inhabituelles. Les exceptions sont des objets qui sont créés lors de situations d'erreurs et qui sont traités dans des sections réservées à cet effet. Enfin, le caractère objet des exceptions autorise la création de hiérarchies d'exception. Leur traitement peut ainsi être réalisé au degré de finesse souhaité. Concrètement, quand le programme détecte une erreur anormale ou une erreur il lève ou lance (throw en anglais) une exception. La levée d'une exception est nécessairement « inscrite dans » un bloc d'essai (try), elle provoque instantanément la sortie du bloc concerné qui doit être capturée dans un bloc catch.

Comme beaucoup d'autres objets du framework .NET, les exceptions spécifiques dérivent d'une classe de base, à savoir la classe Exception. Il existe une hiérarchie entre les exceptions. Globalement, deux grandes familles d'exceptions existent : ApplicationException et SystemException. Elles dérivent toutes les deux de la classe de base Exception. La première est utilisée lorsque des erreurs récupérables sur des applications apparaissent, la seconde est utilisée pour toutes les exceptions générées par le framework .NET. Par exemple, l'exception que nous avons vue, FormatException dérive directement de SystemException qui dérive elle-même de la classe Exception. Le framework .NET dispose de beaucoup d'exceptions correspondant à beaucoup de situations. Notons encore au passage une autre exception bien connue des développeurs qui est la NullReferenceException. Elle se produit lorsqu'on essaie d'accéder à un objet qui vaut null.

II-Try Catch

L'intérêt des exceptions est qu'elles sont typées. Finis les codes d'erreurs incompréhensibles. C'est le type de l'exception, c'est-à-dire sa classe, qui va nous permettre d'identifier le problème. Pour éviter le plantage de l'application, nous devons gérer ces cas limites et intercepter les exceptions. Pour ce faire, il faut encadrer les instructions pouvant atteindre des cas limites avec le bloc d'instruction try...catch, par exemple.

Nous allons chercher à « attrapé » l'exception levée par la méthode de conversion grâce au mot-clé catch. Cette construction nous permet de surveiller l'exécution d'un bout de code, situé dans le bloc try et s'il y a une erreur, alors nous interrompons son exécution pour traiter l'erreur en allant dans le bloc catch.

```
try
            {
                string s = null;
                ProcessString(s);
            }

            catch (ArgumentNullException e)
            {
                Console.WriteLine("{0} Exception la plus spécifique.", e);
            }

            catch (SystemException e)
            {
               Console.WriteLine("{0} Exception un peu plus générale.", e);
            }

            catch (Exception e)
            {
                Console.WriteLine("{0} Exception la plus générale", e);
            }
```

III-Finally

On peut compléter une instruction try...catch avec une instruction appellée Finally. Le bloc Finally est utilisé pour exécuter un ensemble donné d'instructions, qu'une exception soit levée ou non. Par exemple, si vous ouvrez un fichier, il doit être fermé, qu'une exception soit déclenchée ou non.

```
Try
            ' permet a la commande sql de se lancer '
            MaCommandeSql1.ExecuteNonQuery()

            ' indique que la matière a été ajouté a la bdd'
            MessageBox.Show("Ajouter avec succès !")

            ' marque la fin de la commande et déconnecte de la bdd '
            MaConnection.Close()

        Catch ex As Exception

            ' Show the exception's message.
            MessageBox.Show("impossible d'ajouter a la bdd ")

        Finally


            ' marque la fin de la commande et déconnecte de la bdd '
            MaConnection.Close()
            MessageBox.Show("Déconnecter de la bdd")
            cbMatiere.Items.Clear()
            lblConnection.Text = MaConnection.State.ToString


        End Try
```

IV-Racine carré

Au travers du TP racine carré, nous cherchons à explorer l'instruction Throw. Grâce à elle, il est possible de déclencher soi-même la levée d'une exception, si nous considérons que notre code atteint un cas limite, qu'il soit fonctionnel ou technique. Il faut donc utiliser le mot-clé throw, suivi d'une instance d'une exception.

```
public static double RacineCarree(double valeur)
            {
                if (valeur <= 0)

                    throw new ArgumentOutOfRangeException("valeur", "Le
paramètre doit être positif");

                return Math.Sqrt(valeur);
            }
```

V-Propagation de l'exceptions

Une exception est d'abord levée depuis le haut de la pile et si elle n'est pas interceptée, elle descend la pile d'appels vers la méthode précédente. Après qu'une méthode lève une exception, le système d'exécution tente de trouver quelque chose pour la gérer. L'ensemble des « quelque chose » possibles pour gérer l'exception est la liste ordonnée des méthodes qui ont été appelées pour accéder à la méthode où l'erreur s'est produite. La liste des méthodes est connue sous le nom de pile d'appels et la méthode de recherche est la propagation d'exceptions.

Ici, nous allons chercher à enregistrer une erreur dans un fichier de log, lequel sera ensuite envoyé par mail à l'administrateur.

```
public static void MaMethode()
        {
            try
            {
                Console.WriteLine("Veuillez saisir un entier :");
                string chaine = Console.ReadLine();
                int entier = Convert.ToInt32(chaine);
            }
            catch (Exception ex)
            {
                throw;
            }
        }
        static void Main(string[] args)
        {
            try
            {
                MaMethode();
            }
            catch (Exception ex)
            {
                // ici, on intercepte toutes les erreurs possibles en
indiquant qu'un problème inattendu
```

```
                // s'est produit
                Console.WriteLine("L'application a rencontré un problème,
un mail a été envoyé à l'administrateur ...");

                EnvoyerExceptionAdministrateur(ex);
            }
        }

        private static void EnvoyerExceptionAdministrateur(Exception ex)
        {
            string log = @"C:\Users\jcond\Desktop\log.txt";
            /////////////// LOG ////////////
            if (File.Exists(log))
            {
                File.Delete(log);
            }
            using (FileStream fileStr = File.Create(log))
            {
                // Ajouter du texte au fichier
                Byte[] text = new
UTF8Encoding(true).GetBytes(DateTime.Now.ToString() + ":" + ex);
                fileStr.Write(text, 0, text.Length);
            }

            MailMessage mail = new MailMessage();
            SmtpClient SmtpServer = new SmtpClient("smtp.gmail.com");
            mail.From = new MailAddress("troubadouroufo@gmail.com");
            mail.To.Add("pmercy@stjosup.com");
            mail.Subject = "Fichier log Julien CONDOMINES";
            mail.Body = "Bonjour, \n\nVoici le mail demandé envoyé à 11h02
le 07/03/2022. \n\nCordialement \n\nJulien CONDOMINES";

            System.Net.Mail.Attachment attachment;
            attachment = new
System.Net.Mail.Attachment(log,MediaTypeNames.Application.Octet);
            mail.Attachments.Add(attachment);

            SmtpServer.Port = 587;
```

```
            SmtpServer.Credentials = new
System.Net.NetworkCredential("troubadouroufo@gmail.com",
"tp9FmM9PfsTUEKt");
            SmtpServer.EnableSsl = true;

            SmtpServer.Send(mail);
        }
    }
```

VI-Conclusion

Nous avons donc vu ici les exceptions, c'est une notion essentiels pour tout développeur, celle-ci va permettre de fluidifier la gestion des erreurs traditionnels. En effet, dans celle-ci, il existe toujours des conditions if-else pour gérer les erreurs. Ces conditions et le code pour gérer les erreurs sont mélangés avec le flux normal. Cela rend le code moins lisible et maintenable. Avec la gestion d'erreurs, le code de gestion des erreurs est séparé du flux normal.

VII-Annexes

TRY CATCH

Try Catch

```
using System;

namespace Try_Catch
{
    class Program
    {



        static void Main(string[] args)
        {
            // exerice 1

            try
            {
                Console.WriteLine("Entrez une chaîne");
                string chaine = Console.ReadLine() ;
                int valeur = Convert.ToInt32(chaine);
            }
            catch (Exception ex)
```

```
            {
                Console.WriteLine("Il y a un eu une erreur : " +
               ex.ToString());
            }


        }
    }
}
```

Multiple Try Catch

```
using System;

namespace Multiple_Catch
{
    class Program
    {
        // exercice 2
        static void ProcessString(string s)
        {
            if (s == null)
            {
                throw new ArgumentNullException(paramName: nameof(s),
message: "Ne peut pas être nul");
                throw new SystemException();
                throw new Exception();
            }
        }
        static void Main(string[] args)
        {
            try
            {
                string s = null;
                ProcessString(s);
            }

            catch (ArgumentNullException e)
            {
                Console.WriteLine("{0} Exception la plus spécifique.", e);
```

```
            }

            catch (SystemException e)
            {
                Console.WriteLine("{0} Exception un peu plus générale.",
e);
            }

            catch (Exception e)
            {
                Console.WriteLine("{0} Exception la plus générale", e);
            }
        }
    }
}
```

RACINE CARRÉ

```
      using System;


namespace Multiple_Catch
{
    class Program
    {
        // exercice 2
        static void ProcessString(string s)
        {
            if (s == null)
            {
                throw new ArgumentNullException(paramName: nameof(s),
message: "Ne peut pas être nul");
                throw new SystemException();
                throw new Exception();
            }
        }
        static void Main(string[] args)
        {
            try
```

```
            {
                string s = null;
                ProcessString(s);
            }

            catch (ArgumentNullException e)
            {
                Console.WriteLine("{0} Exception la plus spécifique.", e);
            }

            catch (SystemException e)
            {
                Console.WriteLine("{0} Exception un peu plus générale.",
e);
            }

            catch (Exception e)
            {
                Console.WriteLine("{0} Exception la plus générale", e);
            }
        }
    }
}
```

PROPAGATION

```
using Microsoft.AspNetCore.Mvc.Formatters;
using System;
using System.IO;
using System.Net;
using System.Net.Mail;
using System.Net.Mime;
using System.Text;

namespace Propagation
{
    class Program
    {
```

```
        public static void EnregistrerErreurDansUnFichierDeLog(Exception
ex)
        {


        }
        public static void MaMethode()
        {
            try
            {
                Console.WriteLine("Veuillez saisir un entier :");
                string chaine = Console.ReadLine();
                int entier = Convert.ToInt32(chaine);
            }
            catch (Exception ex)
            {
                EnregistrerErreurDansUnFichierDeLog(ex);
                throw;
            }
        }
        static void Main(string[] args)
        {
            try
            {
                MaMethode();
            }
            catch (Exception ex)
            {
                // ici, on intercepte toutes les erreurs possibles en
indiquant qu'un problème inattendu
                // s'est produit
                Console.WriteLine("L'application a rencontré un problème,
un mail a été envoyé à l'administrateur ...");

                EnvoyerExceptionAdministrateur(ex);
            }
        }

        private static void EnvoyerExceptionAdministrateur(Exception ex)
        {
```

```
            string log = @"C:\Users\jcond\Desktop\log.txt";
            ////////////// LOG //////////// 
            if (File.Exists(log))
            {
                File.Delete(log);
            }
            using (FileStream fileStr = File.Create(log))
            {
                // Ajouter du texte au fichier
                Byte[] text = new
UTF8Encoding(true).GetBytes(DateTime.Now.ToString() + ":" + ex);
                fileStr.Write(text, 0, text.Length);
            }

            MailMessage mail = new MailMessage();
            SmtpClient SmtpServer = new SmtpClient("smtp.gmail.com");
            mail.From = new MailAddress("troubadouroufo@gmail.com");
            mail.To.Add("pmercy@stjosup.com");
            mail.Subject = "Fichier log Julien CONDOMINES";
            mail.Body = "Bonjour, \n\nVoici le mail demandé envoyé à 11h02
le 07/03/2022. \n\nCordialement \n\nJulien CONDOMINES";

            System.Net.Mail.Attachment attachment;
            attachment = new
System.Net.Mail.Attachment(log,MediaTypeNames.Application.Octet);
            mail.Attachments.Add(attachment);

            SmtpServer.Port = 587;
            SmtpServer.Credentials = new
System.Net.NetworkCredential("troubadouroufo@gmail.com",
"tp9FmM9PfsTUEKt");
            SmtpServer.EnableSsl = true;

            SmtpServer.Send(mail);
        }
    }
}
```