Clase 10
Excepciones

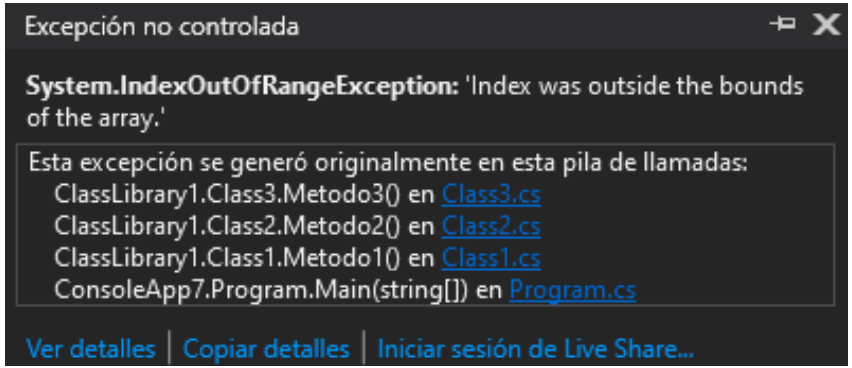
¿Qué es una Excepción?

Una excepción está definida como un evento inesperado que ocurre durante la ejecución de un programa.

Estas pueden ser causadas por el usuario, lógica o errores del sistema.

Excepciones no controladas

Las excepciones pueden estar causadas por un usuario, un problema en la lógica o un error del sistema. Si el desarrollador no provee un mecanismo para manejarlas adecuadamente, el programa finalizará su ejecución abruptamente.



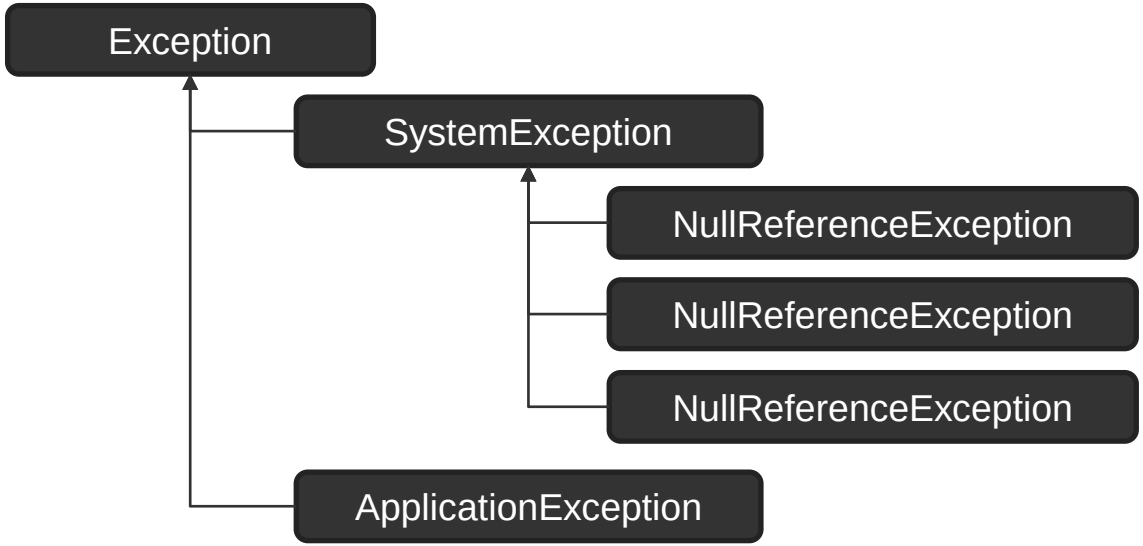
Objeto Exception

Objeto Exception

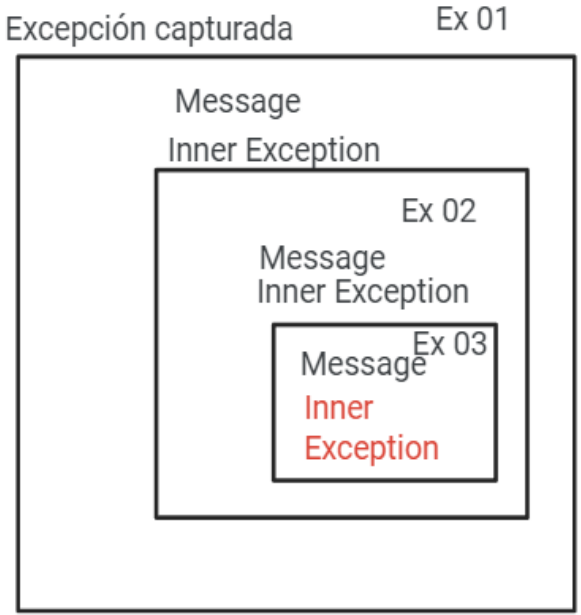
Todas las excepciones derivan de la clase Exception, que es parte de la base class library de .NET. Esto permite que todas las excepciones contengan mensajes descriptivos y distintas propiedades útiles para analizar los errores y prevenirlos.

También es posible crear nuestros propios tipos de excepciones heredando de la clase Exception

Objeto Exception



Propiedad InnerException



Existen casos donde se quiere lanzar una excepción distinta a la original pero sin perder ese contexto original.

La propiedad InnerException nos permite almacenar una excepción dentro de otra, de esta forma una nueva excepción puede contener información acerca de la anterior.

Para cargar la propiedad se debe pasar la instancia como argumento del constructor de la nueva excepción

Propiedad StackTrace

- El stackTrace representa una pila de llamadas en un cierto tiempo.

- Una pila es un tipo de datos que contiene una colección de elementos que trabaja con el formato LIFO
- last-in,first-out, cada elemento en la colección representa una llamada a función que contiene lógica.

- Cuando una llamada a función lanza un error, vamos a tener una colección de estas llamadas que nos van a llevar hasta la llamada que causó el problema.

- El stack trace primero imprime la llamada que causó el error y luego va imprimiendo las llamadas previas que llevaron a la llamada que causó el error, de esta forma leyendo la primer línea del stacktrace vamos a encontrar llamada exacta donde se lanzó el error.

Propiedad StackTrace

```
   at ClassLibrary1.Class3.Metodo3() in C:\Users\lauta\source\repos
\ConsoleApp7\ClassLibrary1\Class3.cs:line 15
   at ClassLibrary1.Class2.Metodo2() in C:\Users\lauta\source\repos
\ConsoleApp7\ClassLibrary1\Class2.cs:line 13
   at ClassLibrary1.Class1.Metodo1() in C:\Users\lauta\source\repos
\ConsoleApp7\ClassLibrary1\Class1.cs:line 13
   at ConsoleApp7.Program.Main(String[] args) in C:\Users\lauta\source
\repos\ConsoleApp7\ConsoleApp7\Program.cs:line 13
```

← Ejemplo de una pila de llamadas.

Manejo de Excepciones

Bloque Try - Catch

Bloque Try – Catch

C# provee las palabras reservadas Try - Catch para implementar el manejo de excepciones.

El bloque Try encapsula las instrucciones que podrían lanzar una excepción mientras que el bloque Catch maneja la excepción si alguna ocurre.

Si ninguna excepción ocurrió dentro del bloque Try, el bloque catch nunca se ejecuta y el programa continua su flujo.

Bloque Try – Catch

Cuando ocurre una excepción dentro del bloque **try**, automáticamente se le transfiere el control al bloque **catch** más apropiado.

Ejemplo de un bloque **Try - Catch:**

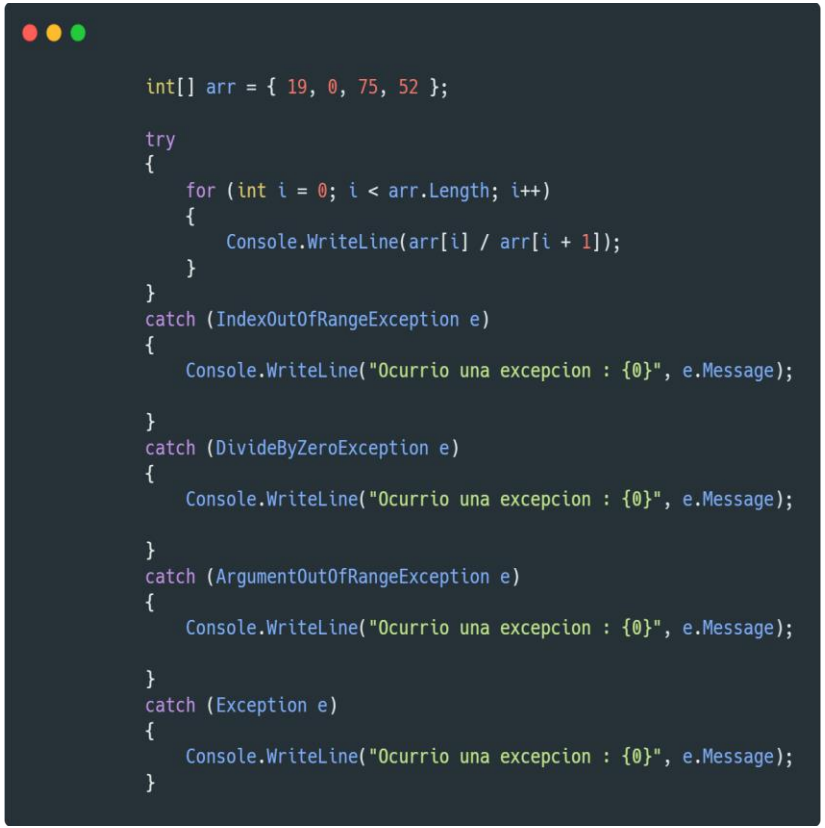
```
int[] arr = { 1, 2, 3, 4, 5 };

try
{
    arr[arr.Length] = 8;
}
catch (Exception)
{
    Console.WriteLine("Ocurrio una excepcion");
}
```

En este caso una vez capturada la excepción se imprimirá un mensaje por consola.

Las excepciones son manejadas por el namespace **System.Exception**, donde vamos a utilizar el objeto base **Exception**.

Bloque Try – Catch

Un **bloque try** puede tener más de un **bloque catch**, Usamos múltiples b**loques catch** cuando no estamos seguros del tipo de excepción que se puede generar, para esto vamos a ir generando bloques desde el más específico al más genérico.



```
int[] arr = { 19, 0, 75, 52 };

try
{
    for (int i = 0; i < arr.Length; i++)
    {
        Console.WriteLine(arr[i] / arr[i + 1]);
    }
}
catch (IndexOutOfRangeException e)
{
    Console.WriteLine("Ocurrio una excepcion : {0}", e.Message);

}
catch (DivideByZeroException e)
{
    Console.WriteLine("Ocurrio una excepcion : {0}", e.Message);

}
catch (ArgumentOutOfRangeException e)
{
    Console.WriteLine("Ocurrio una excepcion : {0}", e.Message);

}
catch (Exception e)
{
    Console.WriteLine("Ocurrio una excepcion : {0}", e.Message);
}
```

Bloque Catch Genérico

- Un bloque **Catch** general(**Exception**), puede capturar cualquier excepción independientemente de su clase y se utiliza con frecuencia para capturar cualquier posible excepción que se pudiera producir por la falta de un controlador adecuado.

- Un bloque try no puede tener más que un bloque catch general.

- En caso de existir, **un bloque Catch general debe ser el último bloque Catch en el programa**.

```
int[] arr = { 19, 0, 75, 52 };

try
{
    for (int i = 0; i < arr.Length; i++)
    {
        Console.WriteLine(arr[i] / arr[i + 1]);
    }
}
catch (Exception e)
{
    Console.WriteLine("Ocurrio una excepcion : {0}", e.Message);
}
```

Bloque Finally

El bloque finally es la parte del código que debe ser ejecutada tanto si ocurrio una excepción como si no.

Independientemente de si el bloque try se ejecuto sin errores o si ocurrió una excepción el bloque Finally siempre se va a ejecutar.

Esto nos puede ser útil para evitar repetir instrucciones y para liberar nuestros recursos luego de lanzar una excepción.

```
int[] arr = { 19, 0, 75, 52 };
try
{
    for (int i = 0; i < arr.Length; i++)
    {
        Console.WriteLine(arr[i] / arr[i + 1]);
    }
}
catch (Exception e)
{
    Console.WriteLine("Ocurrio una excepcion : {0}", e.Message);
}
finally
{
    for (int i = 0; i < arr.Length; i++)
    {
        Console.Write(" {0}", arr[i]);
    }
}
```

En nuestro ejemplo los elementos de nuestro array siempre van a ser mostrados aunque ocurra una excepción.

Instrucción Throw

Una excepción puede ser lanzada manualmente usando la palabra reservada **throw, toda excepción derivada de la clase base Exception puede ser lanzada de esta forma**.

Esto lo que hace es interrumpir la secuencia de ejecución del programa y transferir el control al primer bloque catch que pueda hacerse cargo de esta nueva excepción.

Throw

```
catch (ArgumentException e)
{
   throw new ArgumentNullException();
}
```

Tenemos que tener en cuenta que de esta forma toda la información contenida dentro del objeto ArgumentException se pierde al ser Re-Lanzada como un objeto ArgumentNullException.

```
catch (ArgumentException e)
{
   throw new ArgumentNullException("Ocurrio un error", e);
}
```

Usando la propiedad **InnerException** nos podemos asegurar de no perder esa información.

Throw

```
catch (ArgumentException e)
{
   throw e;
}
```

← También podemos relanzar la misma excepción hacia otro bloque Catch pero tenemos que tener en cuenta que si lo hacemos de esta forma perdemos el **Stack Trace o pila de llamadas.**

```
catch (ArgumentException)
{
   throw;
}
```

← De esta forma nos aseguramos de no perder el **Stack Trace.**

Summary

```
/// <summary>
/// Descripcion de la funcion
/// </summary>
/// <param name="param 1">Descripcion param 1</param>
/// <param name="param 2">Descripcion param 2</param>
/// <returns>Retorno de la funcion en caso que tenga</returns>
/// <exception cref="DivideByZeroException">Descripcion de la Excepcion</exception>
```