What is an Exception in C#?

A runtime error is known as an exception in C#. The exception will cause the abnormal termination of the program execution. So these errors (exceptions) are very dangerous because whenever the exception occurs in the programs, the program gets terminated abnormally on the same line where the error gets occurred without executing the next line of code.

Who is responsible for abnormal termination of the program whenever runtime errors occur?

Objects of exception classes are responsible for abnormal termination of the program whenever runtime errors (exceptions) occur. These exception classes are predefined under BCL (Base Class Libraries) where a separate class is provided for each and every different type of exception like

1.IndexOutOfRangeException
2.FormatException
3.NullReferenceException
4.DivideByZeroException
5.FileNotFoundException
6.SQLException,
7.OverFlowException, etc

Each exception class provides a specific exception error message. All the above exception classes are responsible for abnormal termination of the program as well as after abnormal termination of the program they will be displaying an error message which specifies the reason for abnormal termination i.e. they provide an error message specific to that error.

Exception handling in C# using the Try Catch implementation

To implement the try-catch implementation .NET framework provides three keywords

1.Try

2.Catch

3.Finally

try:

The try keyword establishes a block in which we need to write the exception causing and its related statements. That means exception-causing statements must be placed in the try block so that we can handle and catch that exception for stopping abnormal termination and to display end-user understandable messages.

Catch:

The catch block is used to catch the exception that is thrown from its corresponding try block. It has the logic to take necessary actions on that caught exception. The Catch block syntax in C# looks like a constructor. It does not take accessibility modifier, normal modifier, return type. It takes the only single parameter of type Exception. Inside catch block, we can write any statement which is legal in .NET including raising an exception.

Finally:

The keyword finally establishes a block that definitely executes statements placed in it. Statements that are placed in finally block are always going to be executed irrespective of the way the control is coming out from the try block either by completing normally or throwing an exception by catching or not catching.

Once we use the try and catch blocks in our code the execution takes place as follows:

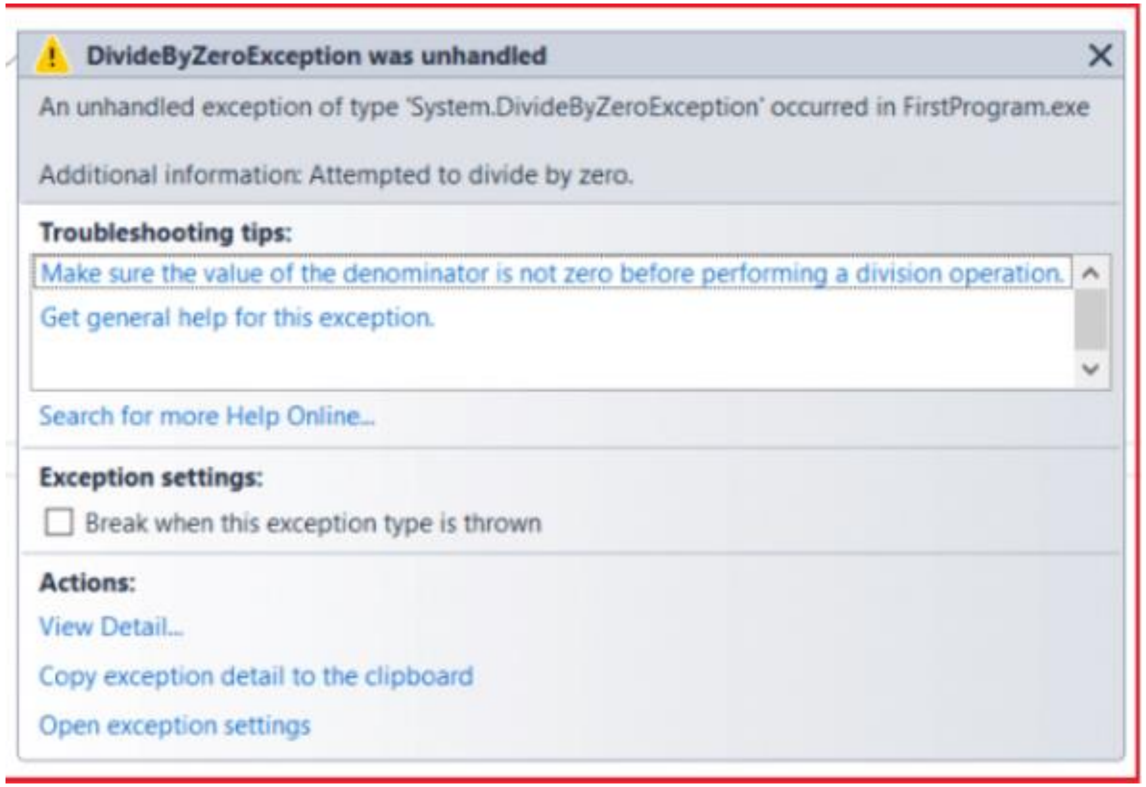
1.If all the statements under try block are executed successfully, from the last statement of the try block, the control directly jumps to the first statement that is present after the catch block (after all catch blocks) without executing catch block (it means there is no runtime error in the code at all).

2.Then if any of the statements in the try block causes an error, from that statement without executing any other statements in the try block, the control directly jumps to the catch blocks which can handle that exception.

3.If a proper catch block is found that handles the exception thrown by the try block, then the abnormal termination stops there, executes the code under the catch block, and from there again it jumps to the first statement after all the catch blocks.

4.If a matching catch is not found then abnormal termination occurs.

```
int a = 20;
int b = 10;
int c;
Console.WriteLine("A VALUE = " + a);
Console.WriteLine("B VALUE = " + b);
c = a / b;
Console.WriteLine("C VALUE = " + c);
Console.ReadKey();
```


DivideByZeroException was unhandled

An unhandled exception of type 'System.DivideByZeroException' occurred in FirstProgram.exe

Additional information: Attempted to divide by zero.

Troubleshooting tips:

Make sure the value of the denominator is not zero before performing a division operation.

Get general help for this exception.

Search for more Help Online...

Exception settings:

☐ Break when this exception type is thrown

Actions:

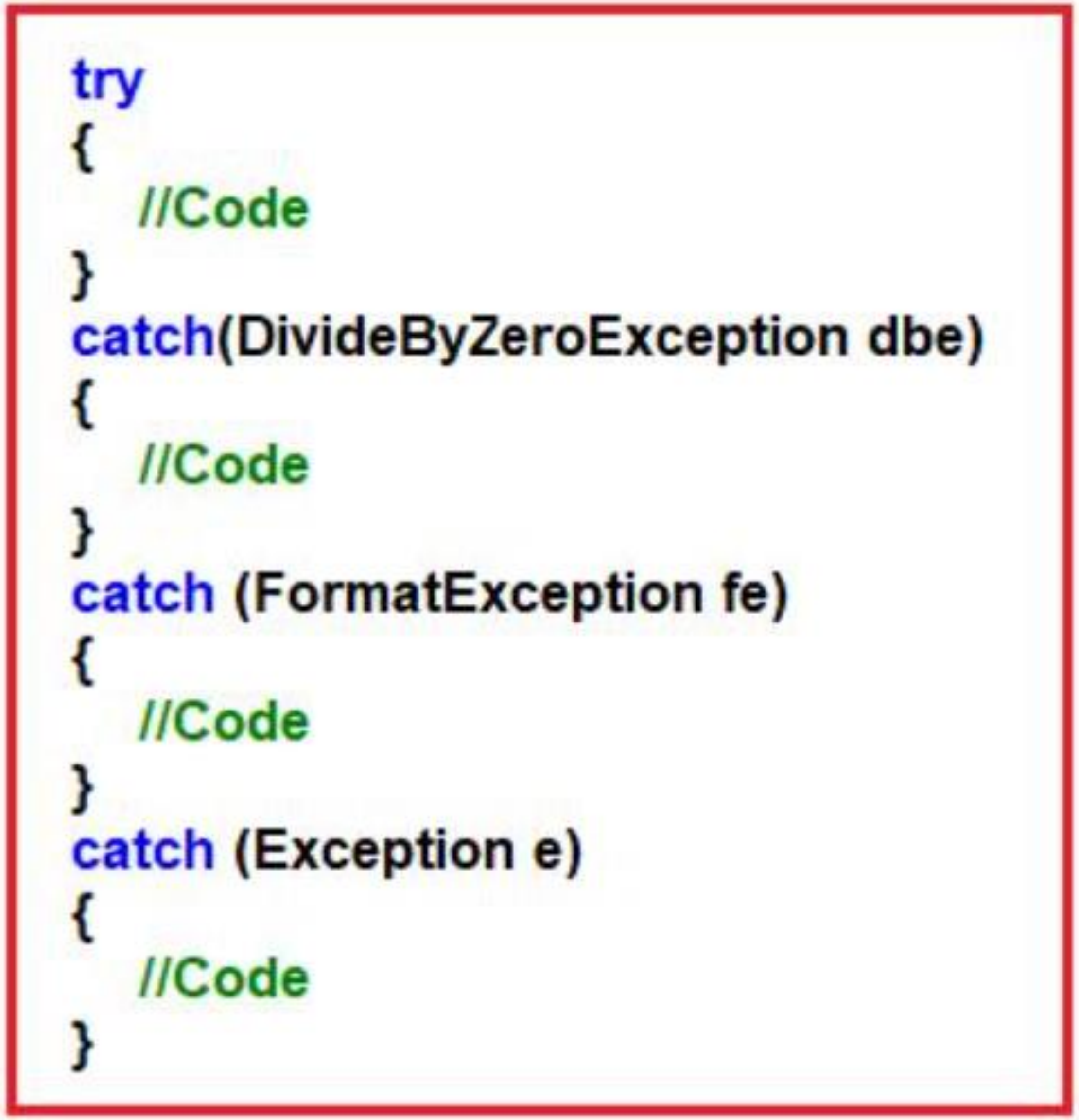
View Detail...

Copy exception detail to the clipboard

Open exception settings

Example: Implementing Multiple Catch Blocks in C#.

```
int a, b, c;
Console.WriteLine("ENTER ANY TWO NUBERS");
try
{
a = int.Parse(Console.ReadLine());
b = int.Parse(Console.ReadLine());
c = a / b;
Console.WriteLine("C VALUE = " + c);
}
catch (DivideByZeroException dbze)
{
Console.WriteLine("second number should not be
zero");
}
catch (FormatException fe)
{
Console.WriteLine("enter only integer numbers");
}
Console.ReadKey();
```



```
try
{
    //Code
}
catch(DivideByZeroException dbe)
{
    //Code
}
catch (FormatException fe)
{
    //Code
}
catch (Exception e)
{
    //Code
}
```

Create Custom Exception in C#?

```
public class OddNumberException : Exception
{
//Overriding the Message property
public override string Message
{
get
{
return "divisor cannot be odd number";
}
}
}
```

```
class Program
{
static void Main(string[] args)
{
int x, y, z;
Console.WriteLine("ENTER TWO INTEGER NUMBERS:");
x = int.Parse(Console.ReadLine());
y = int.Parse(Console.ReadLine());
try
{
if (y % 2 > 0)
{
//OddNumberException ONE = new OddNumberException();
//throw ONE;
throw new OddNumberException();
}
z = x / y;
Console.WriteLine(z);
}
catch (OddNumberException one)
{
Console.WriteLine(one.Message);
}
Console.WriteLine("End of the program");
Console.ReadKey();
}
```

Inner Exception Example in C#:

Let us say we have an exception inside a try block which is throwing **DivideByZeroException** and the catch block catches that exception and then tries to write that exception to a file. However, if the file path is not found, then the catch block is also going to throw **FileNotFoundException**.

Let's say the outside try block catches this **FileNotFoundException** exception, but how about the actual **DivideByZeroException** that was thrown? Is it lost? No, the **InnerException** property of the Exception class contains the actual exception.

```
using System;
using System.IO;
namespace ExceptionHandlingDemo
{
    class Program
    {
        public static void Main()
        {
            try
            {
                try
                {
                    //throw new ArgumentException();
                    Console.WriteLine("Enter First Number");
                    int FN = Convert.ToInt32(Console.ReadLine());

                    Console.WriteLine("Enter Second Number");
                    int SN = Convert.ToInt32(Console.ReadLine());

                    int Result = FN / SN;
                    Console.WriteLine("Result = {0}", Result);
                }

                catch (Exception ex)
                {
                    //make sure this path does not exist
                    string filePath = @"C:\LogFile\Log.txt";
                    if (File.Exists(filePath))
                    {
                        StreamWriter sw = new StreamWriter(filePath);
                        sw.Write(ex.GetType().Name + ex.Message + ex.StackTrace);
                        sw.Close();
                        Console.WriteLine("There is a problem! Plese try later");
                    }
                    else
                    {
                        //To retain the original exception pass it as a parameter
                        //to the constructor, of the current exception
                        throw new FileNotFoundException(filePath + " Does not Exist",
ex);
                    }
                }
            }

            catch (Exception e)
            {
                //e.Message will give the current exception message
                Console.WriteLine("Current or Outer Exception = " + e.Message);

                //Check if inner exception is not null before accessing Message property
                //else, you may get Null Reference Excception
                if (e.InnerException != null)
                {
                    Console.Write("Inner Exception : ");
```