**Part 40 C# Tutorial Exception Handling in C#**

**What is an exception?**  
An exception is an unforeseen error that occurs when a program is running.   
  
  
**Examples:**  
Trying to read from a file that does not exist, throws **FileNotFoundException**.  
Trying to read from a database table that does not exist, throws a **SqlException**.  
  
  
**Program without exception handling**  
using System;  
using System.IO;  
class ExceptionHandling  
{  
    public static void Main()  
    {  
        //This line will throw FileNotFoundException  
        StreamReader streamReader = new StreamReader("C:\\NonExistingFile.txt");  
        Console.WriteLine(streamReader.ReadToEnd());  
        //Closes the underlying stream and releases the system resources.  
        //If there is an exception before this line, the below line will never  
 //be executed and the resources are not relased  
        streamReader.Close();  
    }  
}

**Showing actual unhandled exceptions to the end user is bad for two reasons**  
**1.** Users will be annoyed as they are cryptic and does not make much sense to the end users.

**2.** Exceptions contain information, that can be used for hacking into your application  
  
  
using System;  
using System.IO;  
class ExceptionHandling  
{  
    public static void Main()  
    {  
        StreamReader streamReader = null;  
        try  
        {  
            // This line will throw FileNotFoundException  
            streamReader = new StreamReader("C:\\NonExistingFile.txt");  
            Console.WriteLine(streamReader.ReadToEnd());  
        }  
        // This catch block handles only FileNotFoundException  
        catch (FileNotFoundException fileNotFoundException)  
        {  
            // Log or email the exception  
            // Code to log or email exception details  
  
  
            // Display meaningful error message to the end user  
            Console.WriteLine("Please check if the file \"{0}\" is present", fileNotFoundException.FileName);  
        }  
        // This catch block handles all the other exceptions  
        catch (Exception exception)  
        {  
            Console.WriteLine(exception.Message);  
        }  
        finally  
        {  
            if (streamReader != null)  
            {  
                streamReader.Close();  
            }  
        }  
    }  
}

**An exception is actually a class that derives from System.Exception class. The System.Exception class has several useful properties, that provide valuable information about the exception.**  
**Message:** Gets a message that describes the current exception

**Stack Trace:** Provides the call stack to the line number in the method where the exception occurred.  
  
  
**We use try, catch and finally blocks for exception handling.**  
**try**- The code that can possibly cause an exception will be in the try block.  
**catch** - Handles the exception.  
**finally** - Clean and free resources that the class was holding onto during the program execution.  
  
  
Specific exceptions will be caught before the base general exception, so specific exception blocks should always be on top of the base exception block. Otherwise, you will encounter a compiler error.  
  
  
***Note:*** It is a good practice to always release resources in the finally block, because finally block is guarenteed to execute, irrespective of whether there is an exception or not.  
  
  
**Bad way of cleaning resources.**  
using System;  
using System.IO;  
class ExceptionHandling  
{  
    public static void Main()  
    {  
        StreamReader streamReader = null;  
        try  
        {  
            streamReader = new StreamReader("C:\\NonExistingFile.txt");  
            Console.WriteLine(streamReader.ReadToEnd());  
        }  
        catch (Exception)  
        {  
            throw new Exception("Intentional Exception");  
        }  
        //This code will never be executed, hence it is always a good  
        //practice to release resources in the finally block  
        streamReader.Close();  
    }  
}