A DEEP DIVE INTO C# ERRORS OR EXCEPTIONS HANDLING

Errors refer to the mistake or faults which occur during program development or execution. If you don't find them and correct them, they cause a program to produce wrong results.

Types of Errors

In programming language errors can be divided into three categories as given below-

1. Syntax Errors

Syntax errors occur during development, when you make type mistake in code. For example, instead of writing while, you write WHILE then it will be a syntax error since C# is a case sensitive language.

```
bool flag=true;

WHILE (flag) //syntax error, since c# is case sensitive
{
 //TO DO:
}
```

2. Runtime Errors (Exceptions)

Runtime errors occur during execution of the program. These are also called exceptions. This can be caused due to improper user inputs, improper design logic or system errors.

```
int a = 5, b = 0;
int result = a / b; // DivideByZeroException
```

Exceptions can be handled by using try-catch blocks.

3. Logical Errors

Logic errors occur when the program is written fine but it does not produce desired result. Logic errors are difficult to find because you need to know for sure that the result is wrong

```
int a = 5, b = 6;
double avg = a + b / 2.0; // logical error, it should be (a + b) / 2.0
```

Exception Handling

Exception handling is a mechanism to detect and handle run time errors. It is achieved by using Try-Catch-Finally blocks and throw keyword.

1. Try block

The try block encloses the statements that might throw an exception.

```
try
{
 // Statements that can cause exception.
}
```

2. Catch block

Catch block handles any exception if one exists.

```
catch(ExceptionType e)
{
 // Statements to handle exception.
}
```

3. Finally block

The finally block can be used for doing any clean-up process like releasing unused resources even if an exception is thrown. For example, disposing database connection.

```
finally
{
 // Statement to clean up.
}
```

4. Throw keyword

This keyword is used to throw an exception explicitly.

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

Key points about exceptions handling

1. Exceptions are types that all directly or indirectly derive from System.Exception class.
2. Exception objects contain detailed information about the error, such as the state of the call stack and a text description of the error.
3. A try block is used to throw multiple exceptions that can handle by using multiple catch blocks.
4. More specialized catch block should come before a generalized one. Otherwise specialized catch block will never be executed.
5. Exceptions can be explicitly generated by a program by using the throw keyword.

6. If no exception handling mechanism is used, the program stops executing with an error message.
7. By providing a catch block without a brackets or arguments, we can catch all exceptions occurred inside a try block.

   ```
   Catch
   {
    Console.WriteLine("oException" );
   }
   ```

8. By providing a catch block with an exception object, you can obtain more information about the type of exception that occurred.

   ```

   catch(Exception e)
   {
    Console.WriteLine(e.Message);
   }
   ```

9. The statements inside the finally block are always executed whether exception occurs or not in the program.
10. Also, before the termination of the program finally block is always executed.

Order of Try-Catch-Finally blocks

In the order of exceptions handling blocks try block comes first, after that catch block(s) come and in the last finally block come.

```
try
{
 // statements that can cause exception.
}
catch (MoreSpecificExceptionType e1)
{
 // error handling code
}
catch (SpecificExceptionType e2)
 {
  // error handling code
 }
 catch (GeneralExceptionType eN)
```

```
{
 // error handling code
}
finally
{
 // statement to clean up.
}
```

You can also skip finally block if required.

```
try
{
 // statements that can cause exception.
}
catch (MoreSpecificExceptionType e1)
{
 // error handling code
}
catch (SpecificExceptionType e2)
{
 // error handling code
}
catch (GeneralExceptionType eN)
{
 // error handling code
}
```

You can also skip catch block(s) if required.

```
try
{
 // statements that can cause exception.
}
finally
{
 // statement to clean up.
}
```

Hence a combination of try-catch-finally or try-catch or try-finally blocks is valid.

Source : http://www.dotnet-tricks.com/Tutorial/csharp/EJFQ291113-A-Deep-Dive-into-C#-Errors-or-Exceptions-Handling.html