Handling Exceptions

Gill Cleeren
CTO Xpirit Belgium

@gillcleeren | xpirit.com/gill

Agenda

- Understanding exceptions in code
- Using a try/catch block
- Catching several types of exceptions
- Using finally

Understanding Exceptions in Code

Errors will occur!

Errors are problems that will occur while our application is executing.

Exceptions Will Occur

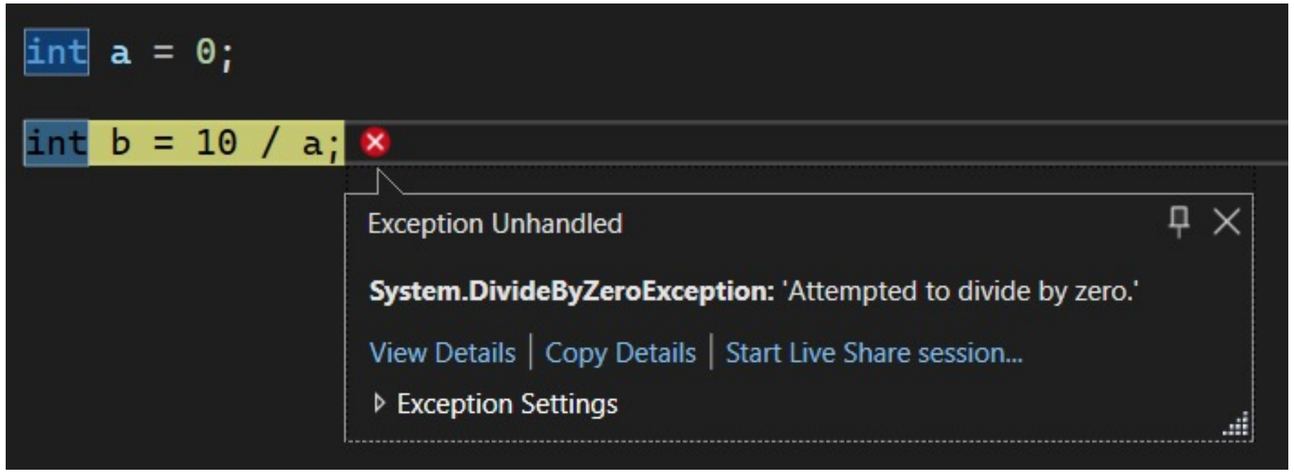
Divide by zero

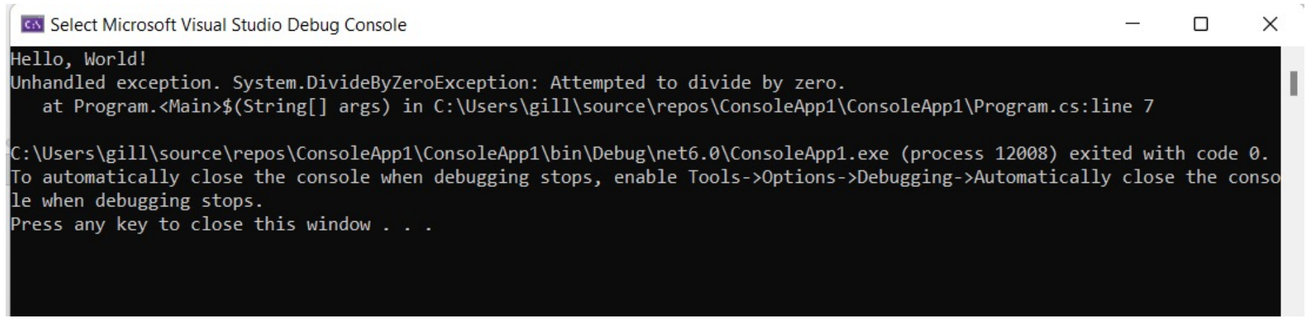
File not accessible

Incorrect permissions

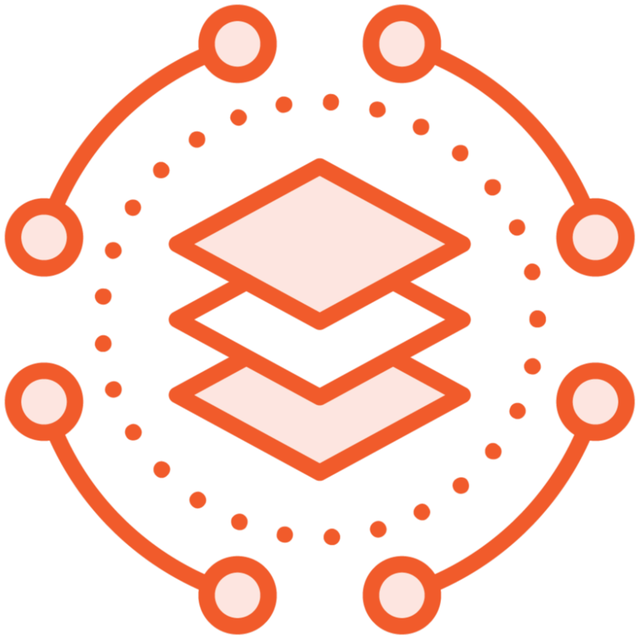
Database unreachable

The Default Handling of the Exception





Using a try/catch Block



In C#, error handling code can be written separately

- Application code executes
- Covered by exception handling code
- Exceptions that occur can be handled

Based on

- try
- catch

Using a try/catch block

```
try
{
    //code goes here
}
catch (Exception ex)
{
    //Here we can handle the exception
    throw;
}
```

The try Block

```
try
{
    //code goes here
}
catch (Exception ex)
{
    throw;
}
```

Code in try block is attempted

If all goes well, will execute regularly

If execution fails, jumps to catch block

The catch Block

```
try
{
    //code goes here
}
catch (Exception ex)
{
    throw;
}
```

Exceptions will be caught here

Handles specific type of exception

- Here we have a base exception
- Multiple types of exceptions can be handled differently

Using a try/catch Block

```
try
{
    string input = Console.ReadLine();
    int a = int.Parse(input);
}
catch (FormatException ex)
{
    //Here we can handle the exception
}
```

Should all code go in a try/catch block?

Exceptions in the Documentation

Exceptions

UnauthorizedAccessException

The caller does not have the required permission.

-or-

`path` specified a file that is read-only.

-or-

`path` specified a file that is hidden.

ArgumentException

.NET Framework and .NET Core versions older than 2.1: `path` is a zero-length string, contains only white space, or contains one or more invalid characters. You can query for invalid characters by using the GetInvalidPathChars() method.

ArgumentNullException

`path` is `null`.

PathTooLongException

The specified path, file name, or both exceed the system-defined maximum length.

DirectoryNotFoundException

The specified path is invalid (for example, it is on an unmapped drive).

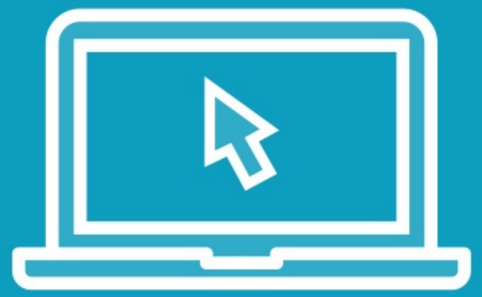
IOException

An I/O error occurred while creating the file.

NotSupportedException

`path` is in an invalid format.

Demo



Adding exception handling

Inspecting the Exception Details

Message

InnerException

StackTrace

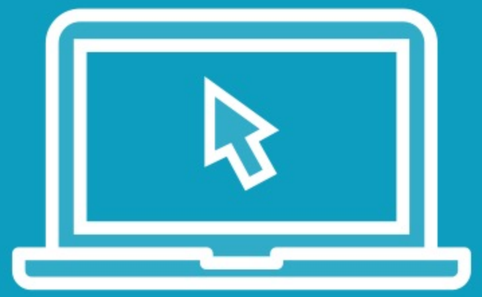
HelpLink

Inspecting the Exception Details

```
try
{
    string input = Console.ReadLine();
    int a = int.Parse(input);
}
catch (FormatException ex)
{
    Console.WriteLine(ex.Message);
    Console.WriteLine(ex.StackTrace);

}
```

Demo



Using the exception details

Catching Several Types of Exceptions

Using Multiple Exception Types

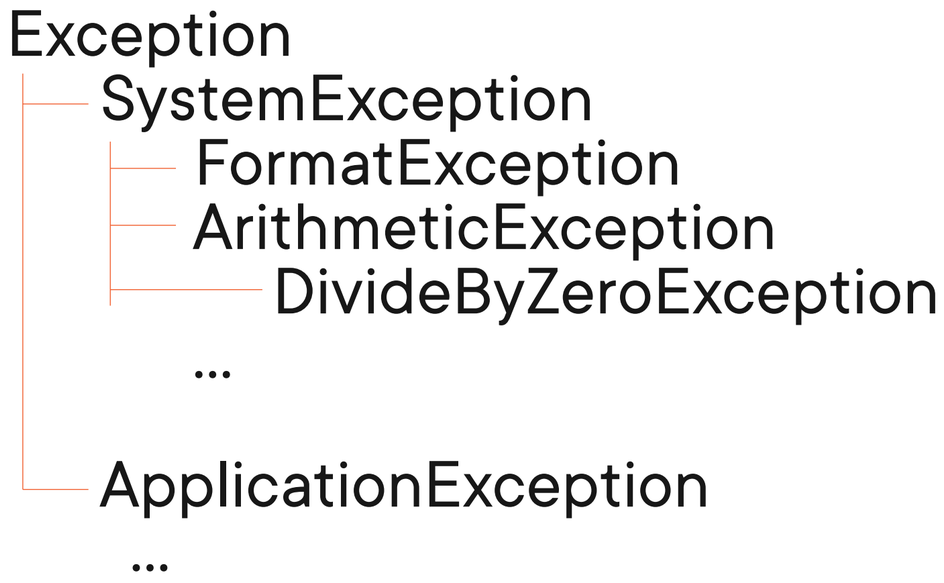
```
try
{
    string input = Console.ReadLine();
    int a = int.Parse(input);
    int b = 10 / a;
}
catch (FormatException fex)
{
    //Here we can handle the exception
}
catch (DivideByZeroException dbzex)
{
    //Here we can handle the exception
}
```

Handling different exception types

.NET comes with many exception types

Exceptions are classes and are part of a hierarchy

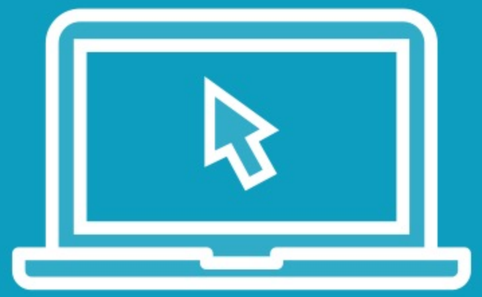
The Exception Hierarchy



Catching All Possible Exceptions

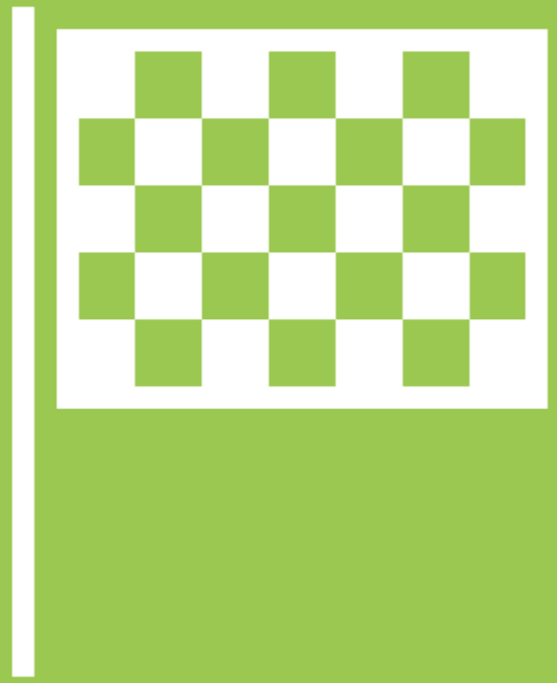
```
try
{
    string input = Console.ReadLine();
    int a = int.Parse(input);
    int b = 10 / a;
}
catch (FormatException fex)
{
    //Here we can handle the exception
}
catch (DivideByZeroException dbzex)
{
    //Here we can handle the exception
}
catch(Exception ex)
{
    //This will be invoked if another type of exception occurs
}
```

Demo



Catching multiple types of exceptions

Using finally



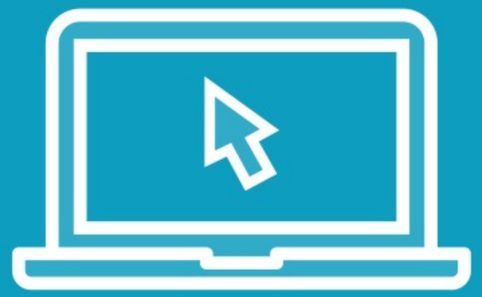
Finally!

If we have code that needs to run regardless if all went fine or not, we can add a finally block.

Using a finally Block

```
try
{
    //code goes here
}
catch (Exception ex)
{
    //Here we can handle the exception
    throw;
}
finally
{
    //This will always execute
}
```

Demo



Adding a finally block

Summary

A try/catch block makes our code more robust

The finally block will always execute

Up next:

Your next steps with C#