

Module 9

"Structured Exception Handling"



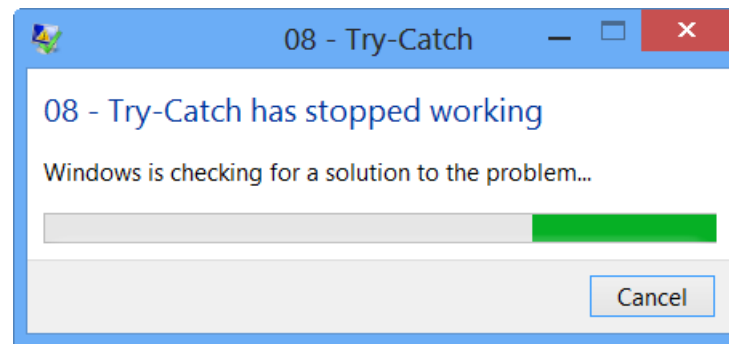
TEKNOLOGISK
INSTITUT

Agenda

- ▶ **Introducing Exceptions**
- ▶ Catching Exceptions
- ▶ Throwing Exceptions
- ▶ Defining Custom Exceptions
- ▶ Lab 9
- ▶ Discussion and Review

What are Exceptions?

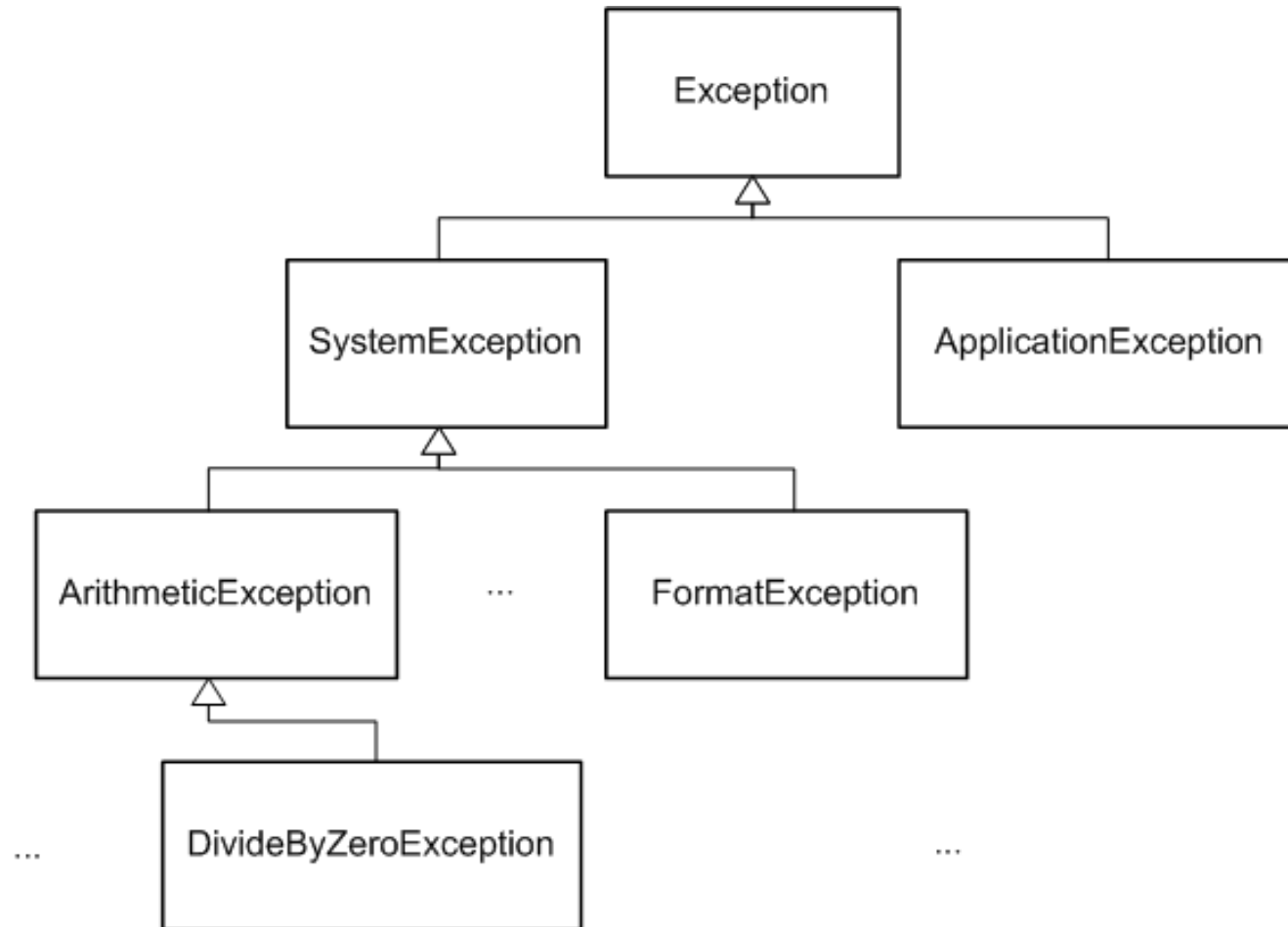
- ▶ Well...



Why Exceptions?

```
int rc = 0;
Start:
rc = OpenFile( "Hello.txt" );
if( rc == -1 ) goto FileNotFound;
rc = GetFileContents();
if( rc == 87 ) goto FileEmpty;
rc = ConvertContents();
if( rc == 112 ) goto IllegalContents;
goto Succeeded;
FileNotFound: goto Start;
FileEmpty: goto Failed;
IllegalContents: goto Failed;
Succeeded: Console.WriteLine( "Yeah!" );
Failed: ...;
```

The Hierarchy of Exceptions



Agenda

- ▶ Introducing Exceptions
- ▶ **Catching Exceptions**
- ▶ Throwing Exceptions
- ▶ Defining Custom Exceptions
- ▶ Lab 9
- ▶ Discussion and Review

The try-catch Construct

- ▶ Perform application logic in a **try**-block
- ▶ Deal with exceptions in a **catch**-block

```
try
{
    Console.WriteLine( "Enter a number: " );
    int i = int.Parse( Console.ReadLine() );
    Console.WriteLine( "You entered the number {0}", i );
}
catch( FormatException exception )
{
    Console.WriteLine( exception );
}
```

- ▶ The exception variable is scoped to the **catch**-block only





Understanding Exceptions: The Call Stack

```
static void Main()  
{  
    A();  
}  
static void A()  
{  
    B();  
}  
static void B()  
{  
    C();  
}  
static void C()  
{  
    Console.WriteLine(" In C" );  
}
```



Program.C()
Line 21

Program.B()
Line 17

Program.A()
Line 13

Program.Main()
Line 9

Call Stack



Core Members of System.Exception

Name	Characteristics
Data	Additional data in the shape of key/value pairs
HelpLink	URL to file or website
InnerException	Previous exceptions
Message	Textual description
Source	Name of assembly throwing the exception
StackTrace	Call stack
TargetSite	Details of the method throwing the exception



Multiple **catch**-blocks

- ▶ Multiple **catch**-blocks can be applied when distinct types of exceptions need to be treated

```
try
{
    Console.WriteLine( "Enter a number: " );
    int i = int.Parse( Console.ReadLine() );
    Console.WriteLine( "You entered the number {0}", i );
}
catch( OverflowException caught ) { ... }
catch( FormatException exception ) { ... }
```

- ▶ Blocks must be ordered from most specific first to most general last
- ▶ Only catch exceptions that you can do something about...



The Generic catch

- ▶ A generic **catch**-block anonymously catches all exceptions

```
try
{
    Console.WriteLine( "Enter a number: " );
    int i = int.Parse( Console.ReadLine() );
    Console.WriteLine( "You entered the number {0}", i );
}
catch
{
    Console.WriteLine( "Something went haywire..!" );
}
```

- ▶ This is equivalent to

```
catch( Exception exception )
{
    Console.WriteLine( "Something went haywire..!" );
}
```



Which Classes Throw Which Exceptions?

```
try
{
    Console.WriteLine( "Enter a number: " );
    int i = int.Parse( Console.ReadLine() );
    Console.Writ
}
catch( OverflowE
{
    Console.Writ
}
catch( FormatExc
{
    Console.WriteLine( exception );
}
catch
{
    Console.WriteLine( "Something went haywire..!" );
}
```

int int.Parse(string s) (+ 3 overload(s))

Converts the string representation of a number to its 32-bit signed integer equivalent.

Exceptions:

System.ArgumentNullException

System.FormatException

System.OverflowException

Visual Studio is Your Friend

- ▶ Use the tools of Visual Studio to both inspect, detect, and prevent exceptions!
- ▶ Use the Exception Settings Window to break when certain exceptions are thrown
 - Locate this at "Debug" -> "Windows" -> "Exception Settings..."
- ▶ Step through the code with F5, F10, F11
- ▶ Set breakpoints to pause execution at specific points
 - These can be parameterized by conditions
 - The items at the bottom of the "Debug" menu
 - You can even change values
- ▶ Note: Visual Studio treats exceptions in two different ways
 - With debugger: Breaks at original exception location
 - Without debugger: Displays OS's exception dialog



Agenda

- ▶ Introducing Exceptions
- ▶ Catching Exceptions
- ▶ **Throwing Exceptions**
- ▶ Defining Custom Exceptions
- ▶ Lab 9
- ▶ Discussion and Review

The throw Keyword

- ▶ It is possible to directly throw exceptions via the **throw** keyword

```
Console.WriteLine( "Enter a number: " );  
int i = int.Parse( Console.ReadLine() );  
Console.WriteLine( "Enter another number: " );  
int j = int.Parse( Console.ReadLine() );  
  
if( i == j )  
{  
    throw new ArgumentException( "Identical numbers entered!" );  
}
```



Re-throwing Exceptions

```
try
{
    Console.WriteLine( "Enter a number: " );
    int i = int.Parse( Console.ReadLine() );
    Console.WriteLine( "You entered the number {0}", i );
}
catch( OverflowException )
{
    Console.WriteLine( "Cannot deal with overflow here! :-( " );
    throw;
}
```

- ▶ The variable in the **catch** can be omitted if not used

The **finally**-block

- ▶ The statements in a **finally** block are always executed – even in the presence of exceptions!

```
SqlConnection connection = new SqlConnection();  
try  
{  
    connection.Open();  
}  
finally  
{  
    connection.Close();  
}
```

- ▶ The **catch** blocks are optional when there is a **finally**-block

Quiz: try-catch-finally

```
try
{
    Console.WriteLine( "try" );

    throw new Exception();
}
catch( Exception )
{
    Console.WriteLine( "catch" );

    return;
}
finally
{
    Console.WriteLine( "finally" );
}
```



Agenda

- ▶ Introducing Exceptions
- ▶ Catching Exceptions
- ▶ Throwing Exceptions
- ▶ **Defining Custom Exceptions**
- ▶ Lab 9
- ▶ Discussion and Review

A Custom Exception Class

- ▶ You can easily create your own custom exceptions

```
public class TicketSalesException : Exception
{
    public TicketSalesException()
    {
    }
    public TicketSalesException( string message )
        : base( message )
    {
    }
    public TicketSalesException( string message,
                                Exception inner )
        : base( message, inner )
    {
    }
}
```



Inner Exceptions

- ▶ If specified, the **InnerException** property supplies the original exception

```
try
{
    Console.WriteLine( "{0} was just requested", EnterTickets() );
}
catch( TicketSalesException e )
{
    Console.WriteLine( e );
    if( e.InnerException != null )
    {
        Console.WriteLine( "Technical info: " + e.InnerException );
    }
}
```

- ▶ In fact, the exception given by **InnerException** may in turn have inner exceptions.



Exception Filters

- ▶ New in C# 6.0: Exception Filters!
 - facilitate the handling of exceptions matching a specific type and/or condition

```
try
{
    Console.WriteLine( "{0} was just requested", EnterTickets() );
}
catch( TicketSalesException e ) when ( e.InnerException != null )
{
    // Catch only when there is an inner exception present

    Console.WriteLine( e );
    Console.WriteLine( "Technical info: " + e.InnerException );
}
```

- ▶ Distinct clauses can match same exception type but with different conditions



Best Practices for Exceptions

- ▶ Use exceptions only for *exceptional* cases
- ▶ Only catch exceptions you can handle
- ▶ Don't let any exceptions slip out of your program or component
 - Create an exception filter!
- ▶ Throw exception objects of a type as specific as possible
- ▶ Never throw a generic **Exception**
- ▶ Make your own exception classes public
- ▶ Implement the three "standard" constructors (*)
- ▶ The "**Exception/ApplicationException**" Controversy

Lab 9: Implementing Exception Handling



Discussion and Review

- ▶ Introducing Exceptions
- ▶ Catching Exceptions
- ▶ Throwing Exceptions
- ▶ Defining Custom Exceptions



WINCUBATE

Jesper Gulmann Henriksen

PhD, MCT, MCSD, MCPD

Phone : +45 22 12 36 31

Email : jgh@wincubate.net

WWW : <http://www.wincubate.net>

Hasselvangelen 243

8355 Solbjerg

Denmark