Module 05

"Structured Exception Handling"

TEKNOLOGISK INSTITUT

TEKNOLOGISK
INSTITUT

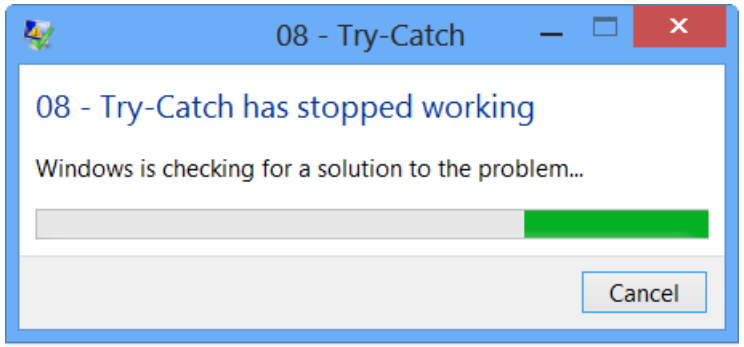
Agenda

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

What are Exceptions?

TEKNOLOGISK INSTITUT

- Well…

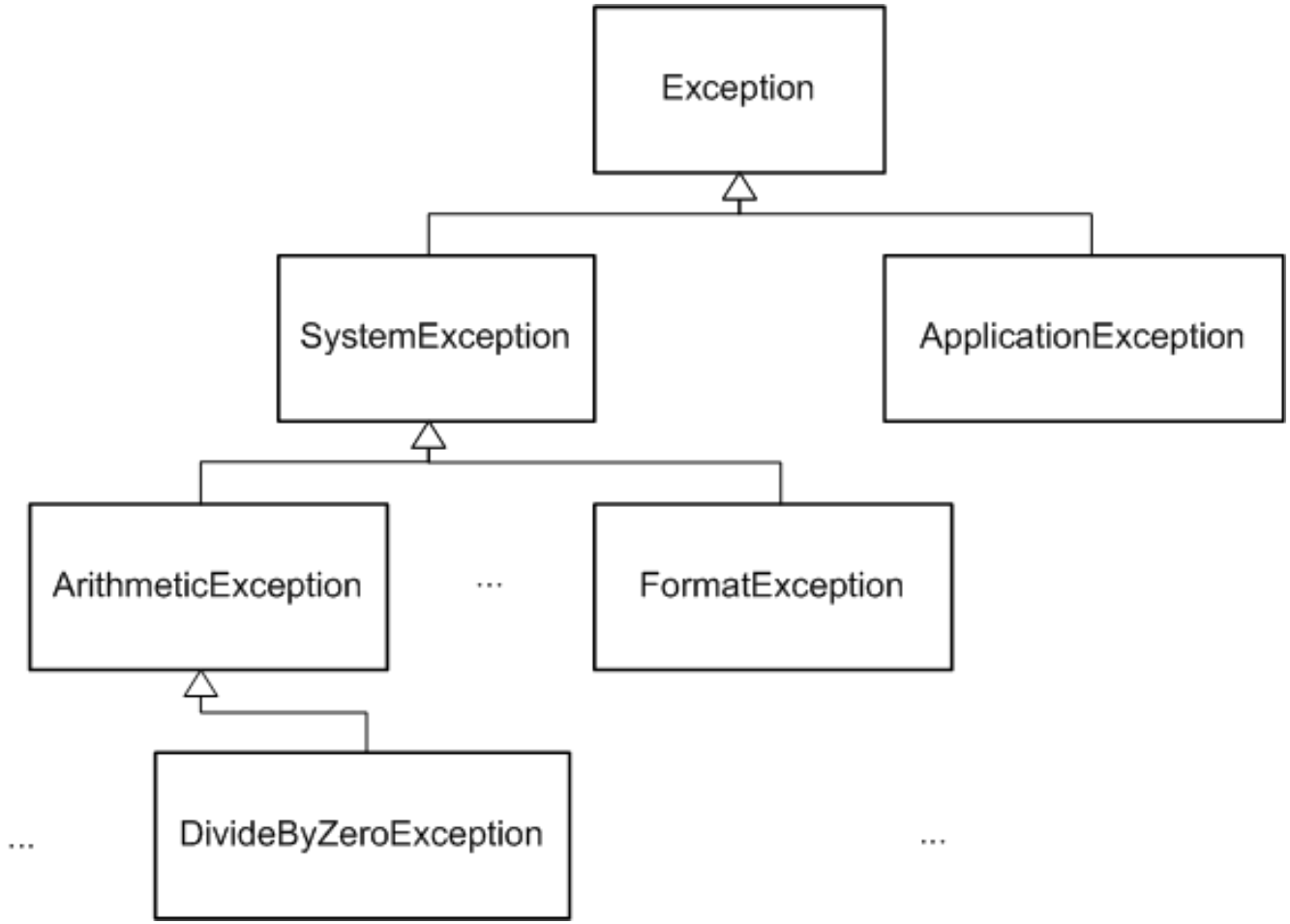


Why Exceptions?

TEKNOLOGISK
INSTITUT

```
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



TEKNOLOGISK
INSTITUT

Agenda

- Introducing Exceptions
- **Catching Exceptions**
- Throwing Exceptions
- Defining Custom Exceptions

The `try-catch` Construct

TEKNOLOGISK INSTITUT

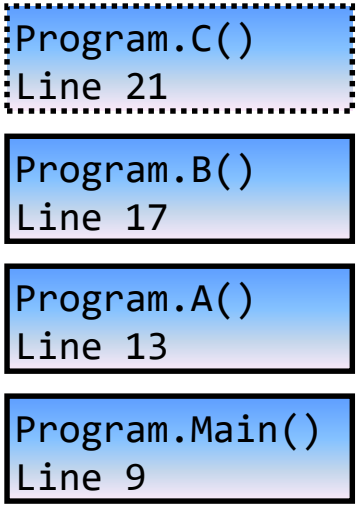
- Perform application logic in a **`try`**-block
- Deal with exceptions in a **`catch`**-block

```
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.Write(" In C" );
}
```



Call Stack

Core Members of `System.Exception`

TEKNOLOGISK INSTITUT

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

TEKNOLOGISK INSTITUT

- 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`

TEKNOLOGISK INSTITUT

- 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?

TEKNOLOGISK INSTITUT

```
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

TEKNOLOGISK INSTITUT

- Use the tools of Visual Studio to both inspect, detect, and prevent exceptions!
- Use the Exception Window to break when certain exceptions are thrown
  - Locate this at "Debug" -> "Exceptions…"
- 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

TEKNOLOGISK
INSTITUT

Agenda

- Introducing Exceptions
- Catching Exceptions
- **Throwing Exceptions**
- Defining Custom Exceptions

The `throw` Keyword

TEKNOLOGISK INSTITUT

- 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!" );
}
```

TEKNOLOGISK INSTITUT

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

- **Exam tip:**
  - **The difference between throw and throw exception is the call stack!**

The `finally`-block

TEKNOLOGISK INSTITUT

- 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

TEKNOLOGISK
INSTITUT

Quiz: `try-catch-finally`

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

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

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

TEKNOLOGISK
INSTITUT

Agenda

- Introducing Exceptions
- Catching Exceptions
- Throwing Exceptions
- **Defining Custom Exceptions**

A Custom Exception Class

TEKNOLOGISK INSTITUT

- 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

TEKNOLOGISK INSTITUT

- 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.

TEKNOLOGISK
INSTITUT

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

TEKNOLOGISK
INSTITUT

Summary

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

TEKNOLOGISK
INSTITUT

Question

You are developing a program using structured exception handling. Your application defines a class called Logger containing the following method.

```
public static class Logger
{
   public static void LogToFile( Exception exception ) { ... }
}
```

You need to ensure that

1. All exceptions are logged using Logger.LogToFile()
2. Exceptions are rethrown using the entire call stack.

Which code fragment should you use?

TEKNOLOGISK
INSTITUT

Question (Cont'd)

a)
```
catch( Exception e )
{
   Logger.LogToFile( e );
   throw e;
}
```

b)
```
catch( Exception e )
{
   Logger.LogToFile( e );
   throw;
}
```

c)
```
catch
{
   Exception e = new Exception();
   Logger.LogToFile( e );
}
```

d)
```
catch
{
   Exception e = new Exception();
   throw e;
}
```

TEKNOLOGISK
INSTITUT