Exception Handling

Montag, 30. Januar 2017

11:20

Ziel: Behandlung von Ausnahmefehlern, Vermeidung von Programmabsturz

**Behandlung einer Ausnahme mit try / catch**

Beispiel:

using System.IO;

class Program {

static void Main(string[] args) {

StreamReader stream = new StreamReader(@"C:\Text.txt");

Console.WriteLine(stream.ReadToEnd());

stream.Close();

}

}

**Arten von Exceptions**

Auszug der Vererbungshierarchie

Exception 
SystemException 
SystemException 
DirectoryNotFoundException 
FileNotFoundException 
ApplicationException 
ArgumentException 
Argu mentN ull Exception 

Mehrere Exceptions-Typen können abgefangen werden um gesondert auf sie zu

reagieren

StreamReader stream = null;

Console.Write("Welche Datei soll geöffnet werden? ... ");

string path = Console.ReadLine();

try {

stream = new StreamReader(path);

Console.WriteLine(stream.ReadToEnd());

stream.Close();

}

catch (FileNotFoundException ex) {

Console.WriteLine(ex.Message);

}

catch (DirectoryNotFoundException ex) {

Console.WriteLine(ex.Message);

}

catch (ArgumentException ex) { //Pfadangabe leer

Console.WriteLine(ex.Message);

}

catch (Exception ex) {

Console.WriteLine(ex.Message);

}

...

**Ausnahmen werfen mit throw:** Zum Test von Code kann man auch selbst ausnahmen erzeugen (werfen)

throw AbcException;

**Finally Block**

Falls nach Behandlung einer Exception etwas unbedingt ausgeführt werden muss (z.B. Freigabe einer Datei, Schließen einer Datenbankverbindung,…)

try {…}

catch {…}

finally {…}

finally Block wird immer ausgeführt unabhängig ob

* Exception geworfen
* catch block ein return statement enthielt
* in catch noch eine Exception geworfen wurde

**Eigenschaften der Klasse Exception**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  | | --- | --- | | **Eigenschaft** | **Beschreibung** | | Data | Stellt zusätzliche Informationen zu der Ausnahme bereit. | | HelpLink | Verweist auf eine Hilfedatei, die diese Ausnahme beschreibt. | | InnerException | Falls bei der Behandlung einer Ausnahme eine weitere Exception ausgelöst wird, beschreibt diese Eigenschaft die neue (innere) Ausnahme. | | Message | Liefert eine Zeichenfolge mit der Beschreibung des aktuellen Fehlers. Die Information sollte so formuliert sein, dass sie auch von einem Anwender verstanden werden kann. | | Source | Liefert einen String zurück, der die Anwendung angibt, in der die Ausnahme ausgelöst worden ist. | | StackTrace | Beschreibt in einer Zeichenfolge die aktuelle Aufrufreihenfolge aller Methoden. | | TargetSite | Liefert zahlreiche Informationen zu der Methode, in der die Ausnahme ausgelöst worden ist. | | |

An nützlichsten sind Message und StackTrace, liefern beide einen string

**Innere Exceptions**

Tritt innerhalb einer Ausnahmebehandlung (catch) eine Exception auf so sollte auch diese Behandelt werden (innere Exception)

Beispiel: Was passiert hier ?

Was passiert wenn der (int) cast in calculateSomething weggelassen wird ?

public static void calculateSomething()

{

double x = 1 / 2 \* 2;

int y = 1 / (int) x ;

}

public static void Main(string[] args)

{

try

{

calculateSomething();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

try

{

StreamWriter file = new StreamWriter("error\_output.txt");

file.WriteLine("Fehler beim Aufruf von calculateSomething");

file.Close();

}

catch

{

Console.WriteLine("Fehler-Output konnte nicht geschrieben werden.");

}

}

}