**Exceptions und Exceptionhandling – Ausnahmen und Ausnahmebehandlung**

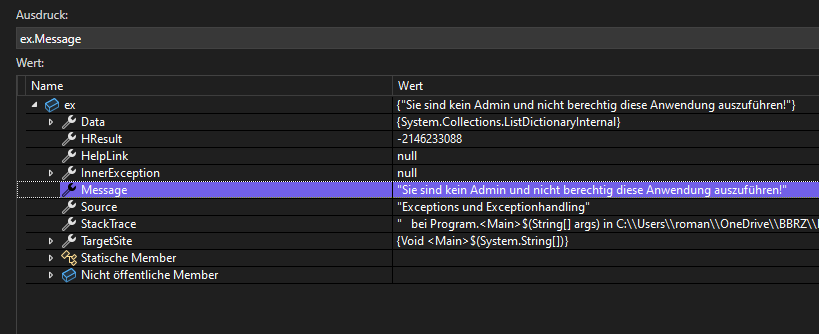
**Einführung**

**Definition**: Was sind **Exceptions (Ausnahmen)?**   
Eine Exception ist eine **Laufzeitabweichung**, die die normale Programmausführung **unterbricht**.

**Was kann eine Exception hervorrufen?** Ganz viel. Z.B.:

* + Division durch Null (**DivideByZeroException**)
  + Zugriff auf nicht vorhandene Array-Indizes (**IndexOutOfRangeException**)
  + Umgang mit null-Werten (**NullReferenceException**)
  + U.V.M.

Was beinhaltet ein Exception-Objekt?



**Behandlung von Exceptions**

**try**

{

// Code, der eine Ausnahme **auslösen könnte**

}

**catch** (Exception ex)

{

// Code zur **Fehlerbehandlung**

}

**finally**

{

// Code, der **immer** ausgeführt wird (auch bei Exceptions)

}

**Erklärung in einfachen Try-Catch-Finally-Block**

* + **try**: Hier befindet sich der "**riskante**" Code.
  + **catch**: Hier wird die Ausnahme abgefangen und ggf. behandelt. Objekt Exception ex enthält Informationen zur Exception. Diese könnten von da aus könnten **geloggt** werden. Es auch der Platz für die Aktivierung entsprechender Alarme an den System- oder Applikation-Administrator (Alerting).
  + **finally**: wird immer ausgeführt, unabhängig davon, ob eine Ausnahme auftritt oder nicht (z. B. um Ressourcen freizugeben). Es ist optional, d.h. man kann es im Gegensatz zu try und catch weg lassen.

**Mehrere catch-Blöcke mit Unterscheidung von Exception-Typen**

**try**

{

// Riskanter Code

}

**catch** (**DivideByZeroException** ex)

{

Console.WriteLine("Division durch Null!");

}

**catch** (**IndexOutOfRangeException** ex)

{

Console.WriteLine("Array-Index außerhalb des Bereichs!");

}

**catch** (**Exception** ex)

{

Console.WriteLine("Ein anderer Fehler ist aufgetreten: " + ex.Message);

}

**Mit throw Exception neu auslösen**

Eine Exception kann (z.B. nach deren Behandlung) auch **neu geworfen** und so **weitergereicht** werden können.

Diese Vorgangsweise ist besonders wichtig in **schichtenorientierten Applikationen**, wo jedes Layer „für sich lebt“, seine Exceptions selbst behandelt **aber auch** solche mittels **throw** neu wirft, damit **der Klient der jeweiligen Schicht** (die aufrufende Unterkomponente), die Chance hat, auf die Exception in **eigener Art und Weise** zu reagieren.

static void MethodeA()

{

MethodeB();

}

static void MethodeB()

{

try

{

MethodeC();

}

catch (Exception ex)

{

Console.WriteLine("Fehler in MethodeB: " + ex.Message);

**throw**; // Exception weitergeben

}

}

static void MethodeC()

{

throw new InvalidOperationException("Fehler in MethodeC");

}

**Eigene Exception-Klassen für den Bedarf einer benutzerdefinierten Ausnahme erstellen**

Mit benutzerdefinierten Exceptions können Fehler klar und aussagekräftig benannt werden, was die **Lesbarkeit des Codes** verbessert. Sie ermöglichen es, unterschiedliche Fehlerarten **präzise zu unterscheiden** und entsprechend zu behandeln.  
Die Klasse einer benutzerdefinierten Exception muss **von** Exception **erben**.

**try**

{

Console.WriteLine("Wie heißt Ihre Systemrolle?");

var antwort = Console.ReadLine();

if (antwort != "Admin")

throw new WrongRuleFrirstNameException();

Console.WriteLine("Willkomen " + antwort + "!");

}

**catch** (**WrongRuleFrirstNameException** ex)

{

Console.WriteLine(ex.Message);

}

public class **WrongRuleException** : Exception

{

public WrongRuleException(): base("Sie sind kein Admin und nicht berechtig diese Anwendung auszuführen!")

{

}

}

**Hinweis:**

Mit Exceptions sollte man sparsam umgehen, weil deren Erzeugung recht viel Ressourcen (RAM, CPU) auf jener Maschine, auf der sie geworfen werden, erzeugt.

Innerhalb einer langen Iteration (Schleife) sollte man gar intelligente Logik einbauen, damit dieselbe Exception nicht bei jedem fehlerhaften Durchlauf auftritt bzw. künstlich (per throw) geworfen wird.

**Aufgabe**

Schritt 1.)

Erstellen Sie eine Bankkonto-Klasse mit Methoden zum **Einzahlen** und **Abheben**. Erstellen Sie eine eigene **BankingException**, die auftritt, wenn jemand versucht, mehr Geld abzuheben, als auf dem Konto verfügbar ist.

Schritt 2.)

Erstellen Sie mindestens eine weitere, sinnvolle Exception-Klasse (z.B. Wenn der einzuzahlende Betrag kleiner null ist oder/und wenn man einen Betrag über eine zulässige Höhe von 440 Euro abheben will) und implementieren Sie diese an passenden Stellen.

Schritt 3.)

Der Admin Ihrer Anwendung möchte über die Lauffähigkeit und Abstürze Ihrer Anwendung informiert werden.  
Erstellen Sie eine Logging-Komponente, deren Log-Methode die Fähigkeit hat, Infos, Warnings und Errors zu loggen.  
Die Log-Methode speichert die Einträge in ein JSON-File, das stündlich (nach Bedarf) neu angelegt wird.  
Beachten Sie bei Logs von Type Error auf deren starke Aussagekraft, indem sie neben ErrorMessage auch den StackTrace Ihres Exception-Objekts mitspeichert.