Software Development

Quellenangaben:

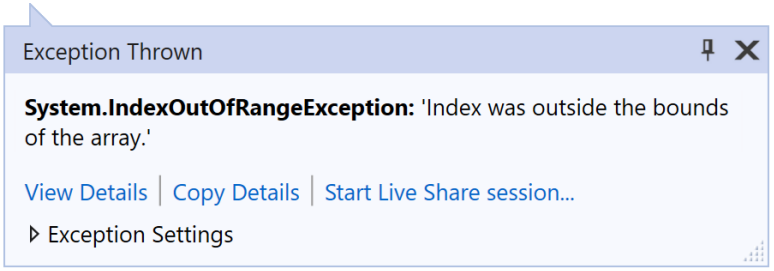
- Jürgen Kotz, München, Visual C# 2019, Carl Hanser Verlag München
- Michael Bonacina, C# Programmieren für Einsteiger (2. Aufl.), BMU Verlag
- https://docs.microsoft.com/de-de/dotnet/csharp/programming-guide

Ausnahmen
Exceptions

- Definition
- Ausnahmen erzeugen
- Ausnahmen abfangen
- Ausnahmen weiterleiten

Definition

- In C# werden Fehler, die zur Laufzeit im Programm auftreten, über Ausnahmen (Exception) übermittelt.
- Die können Fehler aus dem System, (Benutzer-)Eingabefehler oder Laufzeitfehler sein.
- Sobald eine Ausnahme ausgelöst wird, wird sie in der Aufrufliste so lange aufwärts weitergeleitet, bis eine Fehlerbehandlung (catch-Anweisung) für die Ausnahme gefunden wird.
- Nicht abgefangene Ausnahmen werden vom System (bzw. der Laufzeit-Umgebung) in einem Dialogfeld als "nicht behandelte Ausnahme" angezeigt.



Definition

- Alle Ausnahmen werden durch Fehler-Klassen dargestellt.
- Alle Ausnahmen werden von der Basisklasse System.Exception abgeleitet
- Die Fehler-Klasse bestimmt den Typ/die Art der Ausnahme und enthält Eigenschaften mit Details über die Ausnahme.
- Ausnahmen werden mit dem **`throw`** Schlüsselwort "geworfen"

Werfen einer allgemeinen Ausnahme

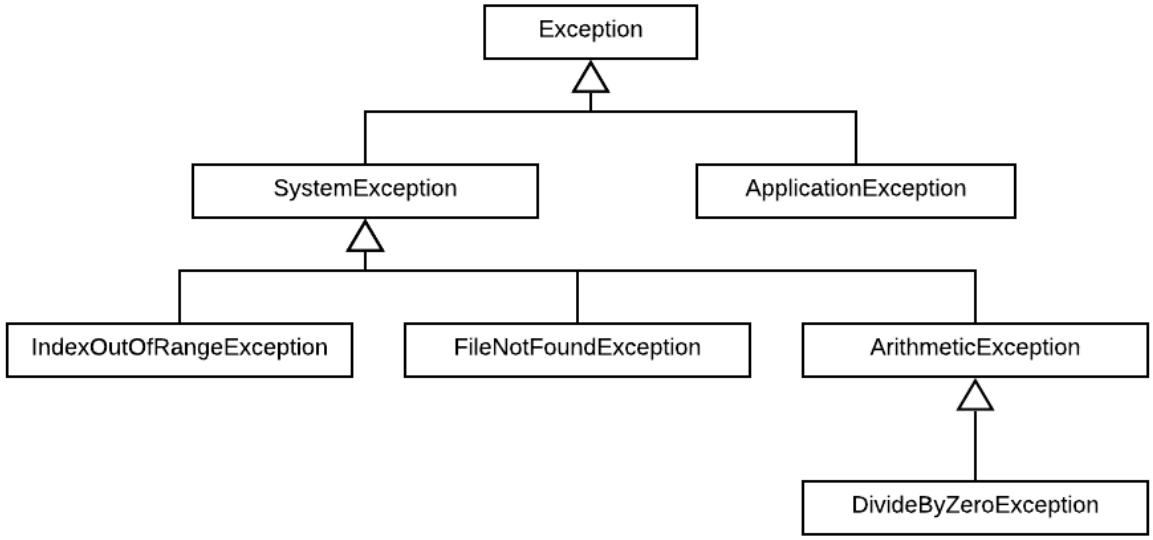
C#

```
throw new System.Exception();
```

Beispiele von Ausnahmen

Ausnahme	Beschreibung
ArithmeticException	Ausnahmen, die während arithmetischer Operationen ausgelöst werden, z.B. DivideByZeroException oder OverflowException
DivideByZeroException	Der Nenner in einem Integer-oder Decimal Division-Vorgang ist 0 (null).
IndexOutOfRangeException	Ein Index liegt außerhalb der Grenzen eines Arrays oder einer Auflistung.
NullReferenceExeption	Wird ausgelöst, wenn ein Objekt referenziert wird, dessen Wert NULL ist.
InvalidCastException	Wird ausgelöst, wenn eine explizite Konvertierung von einem Basistyp ein einen Schnittstellentyp oder in einen abgeleiteten Typ zur Laufzeit fehlschlägt.
StackOverflowException	Wird ausgelöst, wenn der Ausführungsstapel durch zu viele anstehende methodenaufrufe ausgelastet ist. Im Normalfall deutet dies auf eine zu tiefe Rekursion oder eine Endlosschleife hin.

Ableitungshierarchie der Exception



Ausnahmen erzeugen

Methode mit Ausnahme

Diese Methode wirft eine Exception (mit `throw`) wenn der Parameter `p` negativ ist

C#

```
public void MethodeMitAusnahme(int p)
{
    Console.WriteLine($"Beginn der Methode mit Parameter {p}");
    if (p <0)
    {
        throw new System.Exception();
    }
    Console.WriteLine($"Ende der Methode mit Parameter {p}");
}
```

s. Beispiel: Beispiele\Kurseinheit 8\Exceptions\

Ausnahmen erzeugen
Methode mit Ausnahme

Falls eine Exception geworfen wird, wird der Programm-Verlauf abgebrochen und die Methode sofort verlassen.

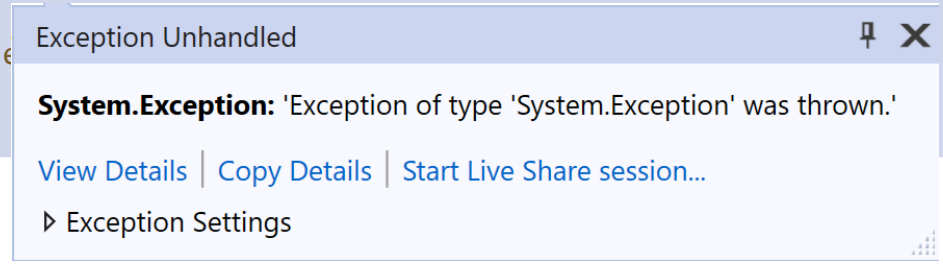
Falls die aufrufende Methode die Exception nicht behandelt, wird das Programm mit einer Fehlermeldung (Laufzeitfehler) abgebrochen.

C #

```
public void MethodeMitAusnahme(int p)
{
    Console.WriteLine($"Beginn der Methode mit Parameter {p}");
    if (p <0)
    {
        throw new System.Exception();
    }
    Console.Write
}
```

p>=0

Ausnahme bricht den Verlauf hier ab



Ausnahmen behandeln
try und catch

Zum Abfangen von (evtl. auftretenden) Fehlern wird der Bereich, in welchem die Ausnahme entstehen könnte, mit Hilfe von `try` eingeschlossen.
`catch` "fängt" eine so geworfene Exception auf und behandelt diese, zum Beispiel durch Ausgabe einer Fehlermeldung auf die Console

C#

```
try
{
   mache etwas
   mache etwas was evtl. eine Exception wirft
   mache noch etwas, was evtl. eine Exception wirft
}
catch (die geworfene Exception)
{
   behandle die Exception
}
```

Ausnahmen behandeln

Beispiel: Division mit Exception

Aufruf: `int resultat = Division(12, 0);`

```
Division von 12 durch 0 ist nicht erlaubt.
```

C #

```
public static int Division(int a, int b)
{
    int resultat = 0;
    try
    {
        resultat = a / b;
    }
    catch (DivideByZeroException ex)
    {
        Console.WriteLine($"Division von {a} durch {b} ist nicht erlaubt.");
    }
    return resultat;
}
```

Regeln für Ausnahmen

- Verwenden Sie einen try-Block um Anweisungen herum, welche eventuell eine Ausnahme generieren kann.
- Sobald eine Ausnahme im try-Block auftritt, springt die Ablaufsteuerung zur nächsten passenden Ausnahme-Behandlung.
- In C# wird die Ausnahme-Behandlung mit dem catch-Schlüsselwort definiert.
- Falls für eine bestimmte Ausnahme keine Ausnahme-Behandlung vorhanden ist, wird die Ausführung des Programms mit einer Fehlermeldung (Dialog-Fenster) angehalten.

Eigene, spezifische Ausnahme-Klassen

- Das Fangen von Exception (der Basisklasse aller Exceptions) behandelt alle Fehler. Allerdings ist dann nicht klar, welche Art von Exception aufgetreten ist und wie diese behandelt werden kann.
- Durch das Definieren von möglichst spezifischen (eigenen) Exception-Klassen, kann man exakte Informationen über den Typ des Fehlers erhalten.
- Eigene Exceptions können ausführliche Informationen zum Fehler, z.B. den Zustand der Aufrufliste (wo genau ist der Fehler passiert) und eine Textbeschreibung des Fehlers enthalten.

Eigene, spezifische Ausnahme-Klassen

- Die Klasse ParameterFehler ist von Exception abgeleitet und implementiert eine eigene, spezifische Ausnahme, die für Fehleingaben von Benutzern verwendet werden kann.
- Der Basiskonstruktor `Exception(String message)` speichert die Fehlermeldung.

C#

```
public class ParameterFehler :Exception
{
    public ParameterFehler(string message): base($"Ungültige Parameter: {message}")
    {
        // Fehlermeldung abspeichern (nicht zwingend)
    }
}
```

Eigene, spezifische Ausnahme-Klassen

- Die Methode Divison wirft bei der Übergabe von negativen Werten einen ParameterFehler (mit entsprechender Fehlermeldung)

C #

```
public static int DivisionMitParameterUeberpruefung(int a, int b)
{
    if (a <= 0 || b <= 0)
    {
        string meldung = $"Falsche Inputparameter: a={a}, b={b}";
        throw new ParameterFehler(meldung);
    }

    return a / b;
}
```

Eigene, spezifische Ausnahme-Klassen

Abfangen der eigenen Exception-Klasse

C #

```
class Program
{
    static void Main(string[] args)
    {
        try
        {
            resultat = DivisionMitParameterUeberpruefung(-2, -4);
        }
        catch (ParameterFehler ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
}
```

Der catch-Block fängt die Ausnahme auf und gibt die Meldung auf die Console aus

```
Ungültige Parameter: Falsche Inputparameter: a=-2, b=-4
```

Ausnahmen behandeln
finally

- Code in einem finally-Block wird immer ausgeführt, egal ob eine Exception ausgelöst wurde oder nicht
- Verwenden Sie einen finally-Block
  - um den Ablauf sauber zu beenden
  - um Ressourcen freizugeben
  - um lokale Informationen auszugeben

Ausnahmen behandeln
finally

C #

```
public class TestKlasseFinally
{
    public void DivisionMethode(string s)
    {
        try
        {
            int i = int.Parse(s);
            Console.WriteLine($"Division: 2/{i}={2 / i}");
        }
        finally
        {
            Console.WriteLine($"Die Eingabe war: {s}");
        }
    }
}
```

wird auch bei Fehlerfall ausgeführt

Ausnahmen behandeln
finally

C #

```
class Program
{
    static void Main(string[] args)
    {
        try
        {
            var meineTestObjektFinally = new TestKlasseFinally();
            meineTestObjektFinally.DivisionMethode("a");
        }
        catch (FormatException ex)
        {
            Console.WriteLine($"Formatfehler: {ex.Message}");
        }
        catch (DivideByZeroException ex)
        {
            Console.WriteLine($"DivisionDurch0: {ex.Message}");
        }
    }
}
```

Erst nachdem der finally Block ausgeführt wurde, wird die Exception im passenden catch-Block behandelt.

```
Die Eingabe war: a
Formatfehler: Input string was not in a correct format.
```

Übungen

Uebungsaufgaben

`Projektarbeit.pdf`