Ausnahmebehandlung - Exceptions

Ingo Köster

Diplom-Informatiker (FH)

Ausnahmen

- Fehler in einem Programm können zu Ausnahmen führen
  - Division durch 0
  - Fehler beim Parsen von Zeichenketten
  - Fehlerhafte Typumwandlung
  - Ungültiger Arrayzugriff
  - etc.

Ausnahmebehandlung

- Die Ausnahmebehandlung soll für Trennung zwischen normalem Programmablauf und der Behandlung von Fehlern sorgen
- In einem Fehlerfall werden Ausnahmen ausgelöst, auf die reagiert werden kann
  - Informationen zum Fehler können abgerufen werden
- Keine ständige Überwachung notwendig

Ausnahmetypen

- Arten der auftretenden Ausnahmen von Klassen und Methoden des .NET Frameworks über die Dokumentation ermitteln

- Beispiele
  - `ArgumentOutOfRangeException`
  - `DivideByZeroException`
  - `IndexOutOfRangeException`
  - `NullReferenceException`
  - `FileNotFoundException`
  - etc.

Ausnahmen erzeugen

- Mittels des Schüsselwortes `throw` wird ein Fehler der Laufzeitumgebung mitgeteilt
  - Beispiel:
  - `throw new NotImplementedException();`
- Erzeugt ein Fehlerobjekt
- Ist ein Objekt der Klasse `Exception` oder davon abgeleitet

Exception Handling

- Fängt Fehler-Objekt auf
- Führt Fehlerbehandlung durch
- Blöcke:
- `try`
  - Programmteil in dem Exceptions ausgelöst werden können
- `catch`
  - Programmteil in dem Exceptions behandelt werden können

Exception Handling

```
try
{
 // Anweisungen die Ausnahmen auslösen können
}
catch (Exception e)
{
 // Behandlung des Fehlers
}
```

Exception Handling

- Auf `try` muss mindestens ein `catch`-Block oder ein `finally`-Block (oder beides) folgen
- Der `finally`-Block wird immer ausgeführt
  - Ideal für Aufräumarbeiten
- Der `catch`-Block kann Zugriff auf ein Exception-Objekt mit Informationen zum Fehler erhalten
- Der Fehler wird im `catch`-Block behandelt oder weitergeworfen

`catch`-Block

- In den runden Klammern wird Typ der Exception angegeben der abgefangen werden soll
- Es können mehrere `catch`-Blöcke für einen `try`–Block angegeben werden
  - mit verschiedenen Exception-Typen
- Die Execptions sollten dann immer ungenauer werden
- Mittels
  - `catch { … }` (ohne runde Klammern) werden alle Fehler abgefangen, ohne dass ein Fehlerobjekt zur Verfügung steht

Fehlerfall

- Ablauf
  - Fehlerobjekt wird mittels `throw` erzeugt
  - `try`-Block wird verlassen
  - Alle lokale Variablen und Objekte im `try`-Block werden zerstört
  - Stack wird wieder auf den Zustand vor Betreten des `try`-Blocks zurückgesetzt

- Dadurch wird der Zustand vor dem Auftreten des Fehlers wiederhergestellt

Handler Suche

› Suche beginnt beim ersten `catch`-Block

› Typ der Exception im `catch`-Block entscheidet, ob es zur Ausführung kommt oder nicht

› `catch` wird aufgerufen wenn Typ in den runden Klammern:
  › Identisch mit dem Typ der geworfenen Exception ist
  › Eine Basisklasse der Klasse der geworfenen Exception ist

› `catch {…}` muss daher immer am Schluss stehen

Nach der Behandlung

- Nach Ausführung des passenden `catch`-Blocks wird der `finally`-Block ausgeführt (falls vorhanden)

- Danach wird die Ausführung nach dem letzten `catch`-Block bzw. der `finally`-Anweisung fortgesetzt

try-catch-finally

```
try {
  // Anweisungen die Ausnahmen auslösen können
}
catch (Exception e) {
  // Behandlung des Fehlers
}
finally {
  // Aufräumen
}
```

Warum finally-Block?

```
public bool MachWas() {
  try
  {
     // Anweisung(en), die eine Exception
     // auslösen könnten
  }
  catch (Exception e) {
    return false;
  }
  // Aufräumarbeiten
  …
  return true;
}
```

Warum finally-Block? - Problem

```
public bool MachWas() {
  try
  {
     // Anweisung(en), die eine Exception
     // auslösen könnte(n)
  }
  catch (Exception e) {
    return false;          // tritt dieser Fall ein…
  }
  // Aufräumarbeiten
  …                        // …wird nicht aufgeräumt!!!
  return true;
}
```

Warum finally-Block ?

```
public bool MachWas() {
  try {
     // Anweisung(en), die eine Exception
     // auslösen könnte(n)
  }
  catch (Exception e) {
    return false;
  }
  finally {
    // Aufräumarbeiten
    …                          // wird immer ausgeführt
  }
  return true;
}
```

Schachteln von `try`-Blöcken

› try-catch Anweisungen können geschachtelt werden

› Innere Blöcke können spezielle Probleme behandeln, äußere den Rest

› Innere Exceptions können mittels `throw` an äußere try-catch Anweisungen weitergeleitet werden

Schachteln von `try`-Blöcken

```
try
{
  try
  {
        // Innere Exception
  }
  catch(ExceptionTyp1 ex1) {
    throw;   // Weiterleiten
  }
}
catch (ExceptionTyp2 ex2) {
      // Hier behandeln
}
catch { … }
```

Schachteln von `try`-Blöcken

```
try {
  try {
                  // Innere Exception
  }
  catch(Typ 1) {
    throw;        // Weiterleiten zum Aufrufer
                  // (ähnlich zu return)
  }
  finally {
                  // wird immer ausgeführt
  }
  // Wird nicht ausgeführt, wenn eine Exception auftritt!!!
}
```

Exceptions in Methoden

- Methoden können unabhängig vom Rückgabewert Ausnahmen werfen

- Aufrufer der Methode muss sich um das Behandeln der Ausnahme kümmern bzw. diese mittels `throw` weiterwerfen

Exceptions in Methoden

```
void MachWas()
{
 try
 {
  MachWasAnderes(-5);
 }
 catch
 {
   throw; // Exception wird
          weitergeworfen
 }
}
```

```
int MachWasAnderes(int i)
{
 if (i < 0)
 {
  throw new Exception("i
kleiner 0");
 }
 return 42;
}
```

Eigene Exceptions

› Von Basisklasse `Exception` ableiten

› Konstruktor und überschriebenen Getter `Message` anlegen

Eigene Exceptions

```
class MyException : Exception
{
  string message;

  public MyException(string message)
  {
    this.message = message;
  }

  public override string Message
  { get { return message; } }
}
```

Eigene Exceptions auslösen

```
int x = -5;

if (x < 0)
{
  throw new MyException("Wert kleiner 0");
}
```

Exception Filter

› Es ist möglich die Catch-Blöcke an Bedingungen zu knüpfen

› Mittels des when Schlüsselwortes wird der Catch-Block nur dann ausgeführt, wenn die Bedingung wahr ist
  › Ähnlich wie ein `if`

```
try { … }
catch (Exception e) when (
  DateTime.Now.DayOfWeek == DayOfWeek.Saturday
  || DateTime.Now.DayOfWeek == DayOfWeek.Sunday)
{ … }
```