Ausnahmen

Gerd Bohlender

Institut für Angewandte und
Numerische Mathematik

Vorlesung: Einstieg in die Informatik mit Java

27.6.07

Übersicht

Einführung

Ausnahmen *sind Ausnahmesituationen in Programmen und werden in Java durch spezielle Klassen dargestellt.*

Im folgenden eine Liste der häufigsten Ausnahmefehler:

- falscher Feldindex (index out of bounds),

Einführung

Ausnahmen *sind Ausnahmesituationen in Programmen und werden in Java durch spezielle Klassen dargestellt.*

Im folgenden eine Liste der häufigsten Ausnahmefehler:

- falscher Feldindex (index out of bounds),
- unzulässige Zeichen beim Lesen einer Zahl,

Einführung

Ausnahmen *sind Ausnahmesituationen in Programmen und werden in Java durch spezielle Klassen dargestellt.*

Im folgenden eine Liste der häufigsten Ausnahmefehler:

- falscher Feldindex (index out of bounds),
- unzulässige Zeichen beim Lesen einer Zahl,
- erfolglose Dateisuche,

Einführung

Ausnahmen *sind Ausnahmesituationen in Programmen und werden in Java durch spezielle Klassen dargestellt.*

Im folgenden eine Liste der häufigsten Ausnahmefehler:

- falscher Feldindex (index out of bounds),
- unzulässige Zeichen beim Lesen einer Zahl,
- erfolglose Dateisuche,
- Division durch Null.

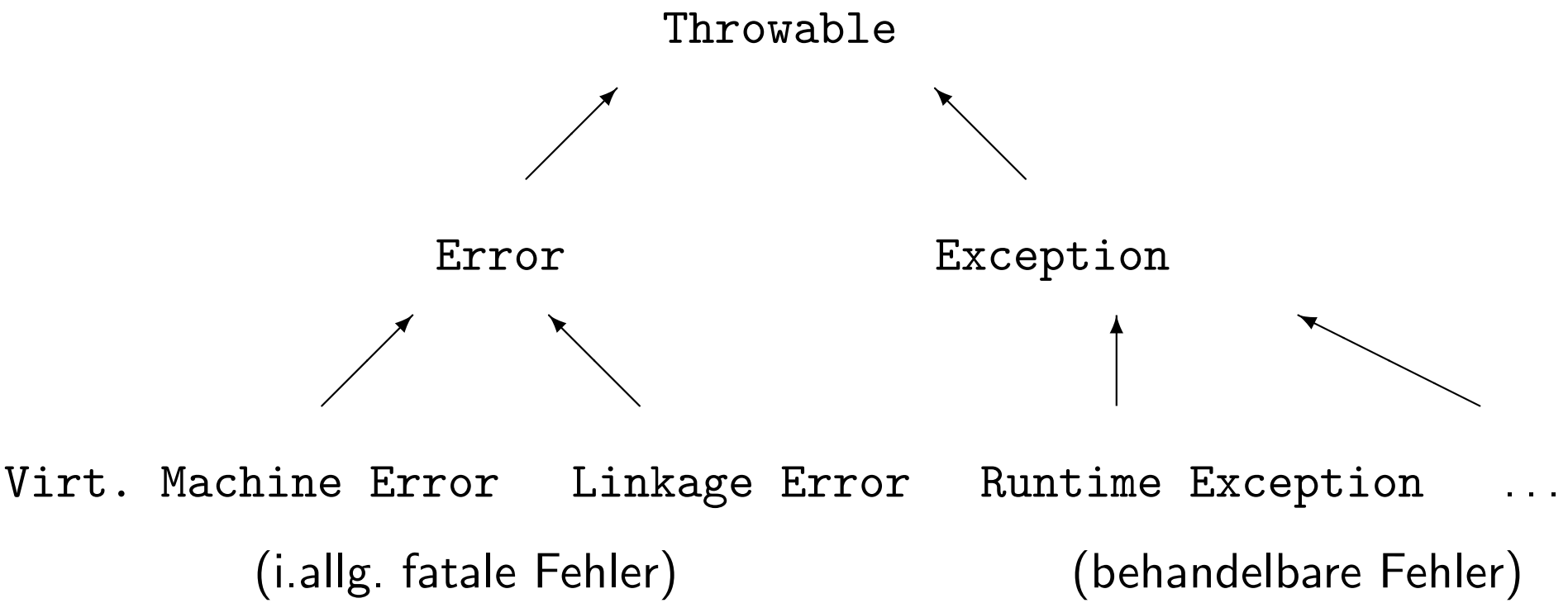
Einführung

Ausnahmen *sind Ausnahmesituationen in Programmen und werden in Java durch spezielle Klassen dargestellt.*

Im folgenden eine Liste der häufigsten Ausnahmefehler:

- falscher Feldindex (index out of bounds),
- unzulässige Zeichen beim Lesen einer Zahl,
- erfolglose Dateisuche,
- Division durch Null.

Klassenhierarchie



Try-Catch

Ähnlich wie in C++ werden Ausnahmen mittels `try-catch`-Blöcken behandelt.

Syntax

```
try Block_1
catch (Parameter) Block_2
...
finally Block_n
```

Try-Catch

Ähnlich wie in C++ werden Ausnahmen mittels `try-catch`-Blöcken behandelt.

Syntax

```
try Block_1
catch (Parameter) Block_2
...
finally Block_n
```

- Es wird zunächst versucht, die Anweisungen in *Block 1* auszuführen.

Try-Catch

Ähnlich wie in C++ werden Ausnahmen mittels `try-catch`-Blöcken behandelt.

Syntax

```
try Block_1
catch (Parameter) Block_2
...
finally Block_n
```

- Es wird zunächst versucht, die Anweisungen in *Block 1* auszuführen.
- Tritt eine Ausnahme ein, so wird *Block 1* abgebrochen und *Block 2* ausgeführt, falls die Ausnahme mit dem Parameter übereinstimmt.

Try-Catch

Ähnlich wie in C++ werden Ausnahmen mittels `try-catch`-Blöcken behandelt.

Syntax

```
try Block_1
catch (Parameter) Block_2
...
finally Block_n
```

- Es wird zunächst versucht, die Anweisungen in *Block 1* auszuführen.
- Tritt eine Ausnahme ein, so wird *Block 1* abgebrochen und *Block 2* ausgeführt, falls die Ausnahme mit dem Parameter übereinstimmt.
- In allen Fällen wird am Ende *Block n* ausgeführt.

Try-Catch

Ähnlich wie in C++ werden Ausnahmen mittels `try-catch`-Blöcken behandelt.

Syntax

```
try Block_1
catch (Parameter) Block_2
...
finally Block_n
```

- Es wird zunächst versucht, die Anweisungen in *Block 1* auszuführen.
- Tritt eine Ausnahme ein, so wird *Block 1* abgebrochen und *Block 2* ausgeführt, falls die Ausnahme mit dem Parameter übereinstimmt.
- In allen Fällen wird am Ende *Block n* ausgeführt.

Beispiel

Einlesen einer Zahl von der Kommandozeile

Mögliche Ausnahmen

(a) keine Eingabe von Parametern an der Kommandozeile,

(b) keine legale Zahl in der Parameterzeichenkette.

Beispiel

Einlesen einer Zahl von der Kommandozeile

Mögliche Ausnahmen

(a) keine Eingabe von Parametern an der Kommandozeile,

(b) keine legale Zahl in der Parameterzeichenkette.

```
class Catch {
   public static void main (String [] args) {
      try {
         int i = Integer.parseInt (args[0]);
         System.out.println (i);
      }
      catch (ArrayIndexOutOfBoundsException e) {
         System.out.println
            ("Aufruf: java Catch <Zahl>");
      }
      catch (NumberFormatException e) {
         System.out.println
            ("Fehlerhafte ganze Zahl");
} } }
```

Beispiel

Einlesen einer Zahl von der Kommandozeile

Mögliche Ausnahmen

(a) keine Eingabe von Parametern an der Kommandozeile,

(b) keine legale Zahl in der Parameterzeichenkette.

```
class Catch {
   public static void main (String [] args) {
      try {
         int i = Integer.parseInt (args[0]);
         System.out.println (i);
      }
      catch (ArrayIndexOutOfBoundsException e) {
         System.out.println
            ("Aufruf: java Catch <Zahl>");
      }
      catch (NumberFormatException e) {
         System.out.println
            ("Fehlerhafte ganze Zahl");
} } }
```

Try-Catch

- Weitere Ausnahmen sind:
  `ArithmeticException` (z.B. Division durch Null)
  `NullPointerException` (z.B. Zugriff auf Objekt über `null`)
- Die `catch`-Klauseln werden in der angegebenen Reihenfolge geprüft. Dabei wird nur die erste passende ausgeführt, d.h. zunächst müssen spezifische und erst danach allgemeine Ausnahmen geprüft werden (zunächst abgeleitete Klassen, dann erst die Basisklasse).

Try-Catch

- Weitere Ausnahmen sind:
  `ArithmeticException` (z.B. Division durch Null)
  `NullPointerException` (z.B. Zugriff auf Objekt über `null`)
- Die `catch`-Klauseln werden in der angegebenen Reihenfolge geprüft. Dabei wird nur die erste passende ausgeführt, d.h. zunächst müssen spezifische und erst danach allgemeine Ausnahmen geprüft werden (zunächst abgeleitete Klassen, dann erst die Basisklasse).
- Definitionen eigener Ausnahmen sind ebenfalls möglich. Diese werden von der Klasse `Throwable` abgeleitet und erzeugt durch

  `throw new` *Ausnahme* (*Fehlertext*)

Weiter Punkte auf der nächsten Seite

Try-Catch

- Weitere Ausnahmen sind:
  `ArithmeticException` (z.B. Division durch Null)
  `NullPointerException` (z.B. Zugriff auf Objekt über `null`)
- Die `catch`-Klauseln werden in der angegebenen Reihenfolge geprüft. Dabei wird nur die erste passende ausgeführt, d.h. zunächst müssen spezifische und erst danach allgemeine Ausnahmen geprüft werden (zunächst abgeleitete Klassen, dann erst die Basisklasse).
- Definitionen eigener Ausnahmen sind ebenfalls möglich. Diese werden von der Klasse `Throwable` abgeleitet und erzeugt durch

  `throw new` *Ausnahme* (*Fehlertext*)

Weiter Punkte auf der nächsten Seite

Try-Catch

- Ausnahmeblöcke können auch geschachtelt werden, d.h. Ausnahmen, die vom inneren Ausnahmeblock nicht behandelt werden, werden an den äußeren weitergereicht.
- Falls in einer Methode oder einem Konstruktor Ausnahmen auftreten können, die dort nicht lokal behandelt werden, müssen diese hinter dem Methodenkopf aufgelistet werden.

Try-Catch

- Ausnahmeblöcke können auch geschachtelt werden, d.h. Ausnahmen, die vom inneren Ausnahmeblock nicht behandelt werden, werden an den äußeren weitergereicht.
- Falls in einer Methode oder einem Konstruktor Ausnahmen auftreten können, die dort nicht lokal behandelt werden, müssen diese hinter dem Methodenkopf aufgelistet werden.

Beispiel

```
boolean testeKlasse (Class e, String t)
   throws ClassNotFoundException {
      ...  // bearbeite c, aber c existiert evtl. nicht.
   }
```

Try-Catch

- Ausnahmeblöcke können auch geschachtelt werden, d.h. Ausnahmen, die vom inneren Ausnahmeblock nicht behandelt werden, werden an den äußeren weitergereicht.
- Falls in einer Methode oder einem Konstruktor Ausnahmen auftreten können, die dort nicht lokal behandelt werden, müssen diese hinter dem Methodenkopf aufgelistet werden.

Beispiel

```
boolean testeKlasse (Class e, String t)
   throws ClassNotFoundException {
      ...  // bearbeite c, aber c existiert evtl. nicht.
   }
```

Davon ausgenommen sind die *ungeprüften Ausnahmen* und deren Subtypen: `Error`, `RuntimeException`

Try-Catch

- Ausnahmeblöcke können auch geschachtelt werden, d.h. Ausnahmen, die vom inneren Ausnahmeblock nicht behandelt werden, werden an den äußeren weitergereicht.
- Falls in einer Methode oder einem Konstruktor Ausnahmen auftreten können, die dort nicht lokal behandelt werden, müssen diese hinter dem Methodenkopf aufgelistet werden.

Beispiel

```
boolean testeKlasse (Class e, String t)
   throws ClassNotFoundException {
      ...  // bearbeite c, aber c existiert evtl. nicht.
   }
```

Davon ausgenommen sind die *ungeprüften Ausnahmen* und deren Subtypen: `Error`, `RuntimeException`