

## 13. Robuste Programme durch Ausnahmebehandlung

### 13.1 Grundlagen

### 13.2 Typen von Ausnahmen

### 13.3 Behandlung von Ausnahmen

### 13.4 Weitergabe von Ausnahmen

### 13.5 Zusammenfassung

## 13. Robuste Programme durch Ausnahmebehandlung

### 13.1 Grundlagen

### 13.2 Typen von Ausnahmen

### 13.3 Behandlung von Ausnahmen

### 13.4 Weitergabe von Ausnahmen

### 13.5 Zusammenfassung

- Eine *Ausnahme* ist ein Ereignis, das zu einem Laufzeitfehler führt.
- Ausnahmen können dabei unterschiedliche Ursachen haben:
  - ① Programmierfehler, z.B. der Zugriff auf ein Array außerhalb der definierten Grenzen.
  - ② Anwenderfehler, z.B. die Angabe eines falschen Dateinamens, was z.B. dazu führt, dass eine Datei, die geöffnet werden soll, nicht gefunden wird.
- In den älteren Programmiersprachen führen Ausnahmen meist zum Absturz des Programms, d.h. das Programm wird einfach beendet.
- Neuere Programmiersprachen wie Java erlauben die systematische Behandlung von Ausnahmen und ermöglichen dadurch robustere Programme.

- Eine Ausnahme kann also durch ein Programm zur Laufzeit verursacht werden. In Java spricht man von einer *Exception*.
- Beim *Auslösen* einer Exception spricht man in Java auch von *Werfen* oder *Throwing*.
- Das *Behandeln* einer Ausnahme, also eine explizite Reaktion auf das Eintreten einer Exception, wird im Java-Sprachgebrauch auch als *Catching* bezeichnet.

- Ein Laufzeitfehler oder eine vom Programmierer gewollte Bedingung löst eine Exception aus (Throwing).
- Diese kann entweder von dem Programmteil, in dem sie ausgelöst wurde, behandelt werden (Catching), oder sie kann weitergegeben werden (an andere Programmteile).
- Die Regel, dass alle Exceptions entweder behandelt oder weitergegeben werden müssen, bezeichnet man auch als *catch-or-throw-Regel*.
- Wird die Exception weitergegeben, so hat der Empfänger der Ausnahme erneut die Möglichkeit, sie entweder zu behandeln oder selbst weiterzugeben.
- Wird die Exception von keinem Programmteil behandelt, so führt sie zum Abbruch des Programms und zur Ausgabe einer Fehlermeldung.

### 13. Robuste Programme durch Ausnahmebehandlung

#### 13.1 Grundlagen

#### 13.2 Typen von Ausnahmen

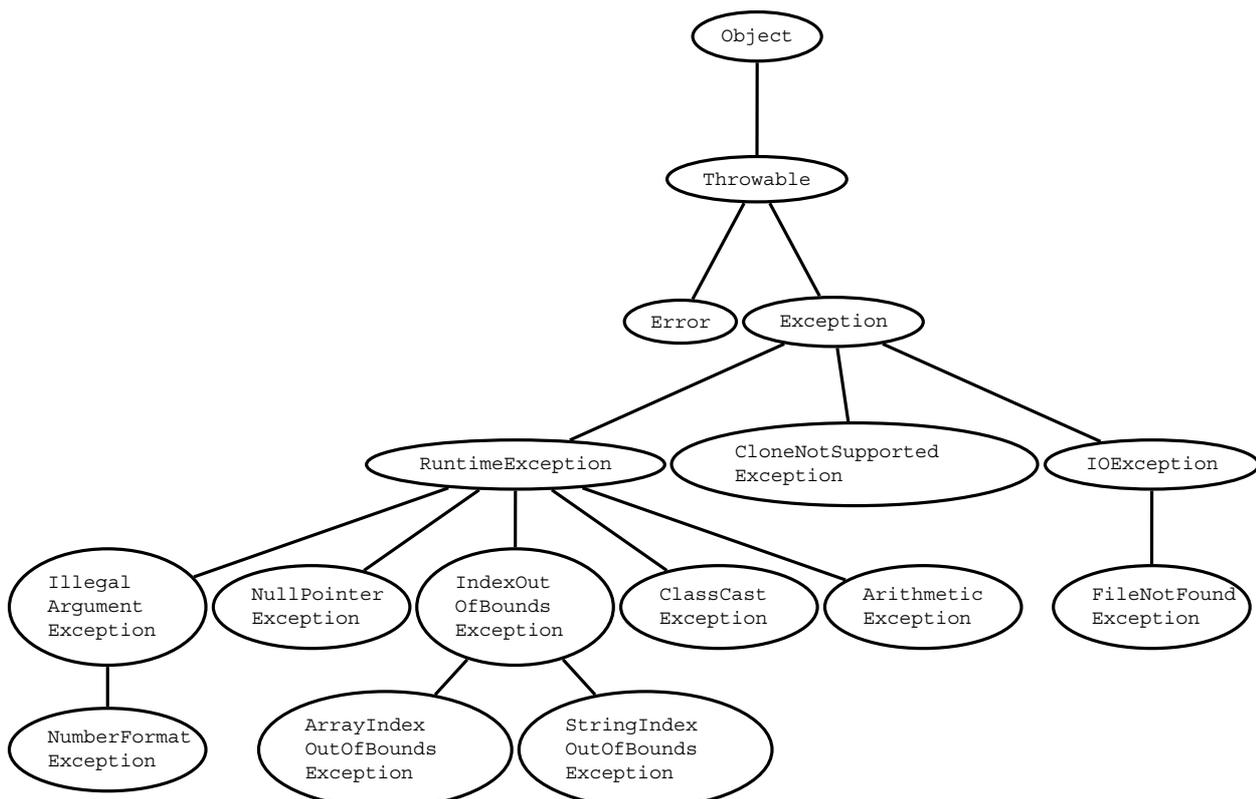
#### 13.3 Behandlung von Ausnahmen

#### 13.4 Weitergabe von Ausnahmen

#### 13.5 Zusammenfassung

- In Java sind Exceptions als normale Klassen realisiert, die instanziiert werden können. Insbesondere kann man auch eigene Exceptions definieren.
- Für die wichtigsten Exceptions gibt es bereits vordefinierte Klassen (im Package `java.lang`, d.h. sie sind ohne `import`-Anweisung implizit eingebunden und verwendbar).
- Oberklasse aller Exceptions ist die Klasse `Throwable`, die keine explizite Oberklasse besitzt und damit implizit von `Object` abgeleitet ist.
- Die Klasse `Throwable` stellt wichtige Konstruktoren und Methoden zum Exception-Handling bereit.

## Wichtige Ausnahmen in ihrer Vererbungshierarchie



- Java unterscheidet zwei Arten von Exceptions:
  - ① “Allgemeine” Exceptions (Unterklassen von `Exception`, aber nicht von `RuntimeException`) modellieren Fehler, die im Routine-Betrieb des Programms entstehen können, z.B. auf Grund falsch eingegebener Daten. Diese Exceptions müssen *immer* behandelt oder weitergegeben werden.
  - ② “Spezielle” Laufzeit-Ausnahmen (Unterklassen von `RuntimeException`) modellieren dagegen Fehler, die nicht im Routine-Betrieb des Programmes auftreten sollten, und daher auf Programmierfehler hindeuten, z.B. der Zugriff auf ein Array außerhalb der definierten Grenzen. Diese Exceptions müssen *nicht* behandelt oder explizit weitergegeben werden, bilden also eine Ausnahme zur catch-or-throw-Regel.
- Definiert man eigene Exception-Klassen, sollte man dies im Blick haben, auch wenn beide Arten von Exceptions nicht immer klar zu trennen sind.

- Zusätzlich unterscheidet Java noch “schwere” Exceptions (oder: Fehler).
- Schwere Fehler modellieren hauptsächlich Probleme, die in der virtuellen Maschine ausgelöst wurden.
- Beispiel: `OutOfMemoryError` tritt auf, wenn die JVM ein neues Objekt allozieren soll, aber keinen Speicher mehr zur Verfügung hat.
- Fehler der Klasse `Error` oder deren Unterklassen müssen ebenfalls nicht behandelt oder weitergegeben werden.
- Es wird empfohlen, dies auch tatsächlich nicht zu tun.

## 13. Robuste Programme durch Ausnahmebehandlung

### 13.1 Grundlagen

### 13.2 Typen von Ausnahmen

### 13.3 Behandlung von Ausnahmen

### 13.4 Weitergabe von Ausnahmen

### 13.5 Zusammenfassung

- Das Behandeln von Exceptions erfolgt mit der `try-catch`-Anweisung:

```
try
{
    <Anweisung>
    ...
}
catch (<ExceptionTyp> e)
{
    <Anweisung>
    ...
}
```

- Der `try`-Block enthält eine oder mehrere Anweisungen, bei deren Ausführung eine Exception vom Typ `ExceptionTyp` auftreten kann (dabei wird dann ein Objekt vom Typ `ExceptionTyp` erzeugt).
- Tritt bei einer der Anweisungen eine entsprechende Exception auf, wird die normale Programmausführung unterbrochen.
- Der Programmablauf fährt mit der ersten Anweisung im `catch`-Block fort.
- Im `catch`-Block kann Code untergebracht werden, der eine angemessene Reaktion auf die Exception realisiert.
- Nach einer `try-catch`-Anweisung wird mit der ersten Anweisung nach dem `catch`-Block normal fortgefahren. Dies geschieht entweder wenn das Ende des `try`-Blocks erreicht wurde (und dabei keine Exception aufgetreten ist) oder wenn das Ende des `catch`-Blocks nach einer Exception erreicht wurde.

- Im folgenden Beispiel soll der String "40" aus verschiedenen Zahlensystemen in ein `int` konvertiert und dann als Dezimalzahl ausgegeben werden.
- Dazu verwenden wir die statische Methode `Integer.parseInt(String, int)`, die einen String `s` und ein `int` `i` (Basis des Zahlensystems) als Parameter erwartet.

```
public class KeineBehandlung
{
    public static void main(String[] args)
    {
        int i = 0;

        for(int base=10; base >=2; --base)
        {
            i = Integer.parseInt("40", base);
            System.out.println("40 zur Basis "+base+" = "+i);
        }
    }
}
```

Bei Ausführung stürzt das Programm mit folgender Ausgabe ab:

```
40 zur Basis 10 = 40
40 zur Basis 9 = 36
40 zur Basis 8 = 32
40 zur Basis 7 = 28
40 zur Basis 6 = 24
40 zur Basis 5 = 20
Exception in thread "main" java.lang.NumberFormatException:
For input string: "40"
    at java.lang.NumberFormatException.forInputString
(NumberFormatException.java:48)
    at java.lang.Integer.parseInt(Integer.java:447)
    at de.lmu.ifi.dbs.info2.ss07.skript.exceptions.KeineBehandlung.main
(KeineBehandlung.java:21)
```

```
public class MitBehandlung
{
    public static void main(String[] args)
    {
        int i = 0;
        int base = 0;

        try
        {
            for(base=10; base >=2; --base)
            {
                i = Integer.parseInt("40", base);
                System.out.println("40 zur Basis "+base+" = "+i);
            }
        }
        catch (NumberFormatException e)
        {
            System.out.println("40 ist keine Zahl zur Basis "+base);
        }
    }
}
```

Zwar ist 40 immer noch keine Zahl zur Basis 4, aber das Programm fängt diesen Fehler nun ab und gibt eine geeignete Fehlermeldung aus:

```
40 zur Basis 10 = 40
40 zur Basis 9 = 36
40 zur Basis 8 = 32
40 zur Basis 7 = 28
40 zur Basis 6 = 24
40 zur Basis 5 = 20
40 ist keine Zahl zur Basis 4
```

- Im Kopf der `catch`-Klausel wird die Art des abzufangenen Fehlers definiert (in unserem Beispiel `NumberFormatException`).
- Das Objekt `e` vom Typ `NumberFormatException`, das im `try`-Block erzeugt wurde, wird dem `catch`-Block übergeben.
- Da alle Exception-Klassen von der Klasse `Throwable` abgeleitet sind, ist `e` auch vom Typ `Throwable` und erbt alle Methoden aus dieser Vaterklasse.

- Die Klasse `Throwable` definiert wichtige Methoden, die bei der Behandlung von Exceptions sehr hilfreich sind, z.B.
  - Die Methode `getMessage` liefert einen Fehlertext.
  - Die Methode `printStackTrace` druckt einen Auszug aus dem aktuellen Laufzeit-Keller.
- Wird eine Laufzeit-Exception nicht behandelt, ruft die JRE automatisch die Methode `printStackTrace` auf, bevor es das Programm beendet.

- Die Reaktion auf eine Exception muss nicht zwangsweise darin bestehen, das Programm zu beenden.
- Stattdessen kann versucht werden, den Fehler zu beheben oder zu umgehen um dann mit dem Programm fortzufahren.
- Im obigen Programm führt die Behandlung der Exception zum Programmende, da nach dem `catch`-Block keine Anweisungen mehr stehen.
- Soll nach jedem Fehler die Konvertierung mit der nächsten Basis fortgesetzt werden, muss die `try-catch`-Anweisung in die `for`-Schleife platziert werden.
- Ob es sinnvoller ist, nach einem Fehler abubrechen oder fortzufahren, hängt von der Art des Fehlers ab.

```
public class NachFehlerFortsetzen
{
    public static void main(String[] args)
    {
        int i = 0;

        for(int base=10; base >=2; --base)
        {
            try
            {
                i = Integer.parseInt("40", base);
                System.out.println("40 zur Basis "+base+" = "+i);
            }
            catch (NumberFormatException e)
            {
                System.out.println("40 ist keine Zahl zur Basis "+base);
            }
        }
    }
}
```

Die Ausführung des Programms ergibt nun folgende Ausgabe:

```
40 zur Basis 10 = 40
40 zur Basis 9 = 36
40 zur Basis 8 = 32
40 zur Basis 7 = 28
40 zur Basis 6 = 24
40 zur Basis 5 = 20
40 ist keine Zahl zur Basis 4
40 ist keine Zahl zur Basis 3
40 ist keine Zahl zur Basis 2
```

- Innerhalb eines `try`-Blocks können natürlich auch mehrere Exceptions (unterschiedlichen Typs) auftreten.
- Daher ist es möglich, zu einem `try`-Block mehrere `catch`-Klauseln bzw. -Blöcke anzugeben.
- Jede `catch`-Klausel fängt den Fehler ab, die zum Typ des angegebenen Fehlerobjekts zuweisungskompatibel ist, d.h. alle Fehler der angegebenen Exception-Klasse und all ihrer Unterklassen.
- Die einzelnen `catch`-Klauseln werden in der Reihenfolge ihres Auftretens abgearbeitet.

- Die `try-catch`-Anweisung enthält einen optionalen Bestandteil, den wir noch nicht erläutert haben: die sogenannte `finally`-Klausel.
- Im Block der `finally`-Klausel kann Code plaziert werden, der immer dann ausgeführt wird, wenn die zugehörige `try`-Klausel betreten wurde.
- Dieser Code in der `finally`-Klausel wird grundsätzlich ausgeführt, *unabhängig* davon, welches Ereignis dazu führte, dass die `try`-Klausel verlassen wurde.
- Die `finally`-Klausel ist der ideale Ort für Aufräumarbeiten wie z.B. Dateien zu speichern/schließen, Ressourcen freizugeben, etc.

```
public class NachFehlerFortsetzenPlusFinally
{
    public static void main(String[] args)
    {
        int i = 0;

        for(int base=10; base >=2; --base)
        {
            try
            {
                i = Integer.parseInt("40", base);
                System.out.println("40 zur Basis "+base+" = "+i);
            }
            catch (NumberFormatException e)
            {
                System.out.println("40 ist keine Zahl zur Basis "+base);
            }
            finally
            {
                System.out.println("Ein bloedes Beispiel!");
            }
        }
    }
}
```

Die Ausführung des Programms ergibt nun folgende Ausgabe:

```
40 zur Basis 10 = 40
Ein bloedes Beispiel.
40 zur Basis 9 = 36
Ein bloedes Beispiel.
40 zur Basis 8 = 32
Ein bloedes Beispiel.
40 zur Basis 7 = 28
Ein bloedes Beispiel.
40 zur Basis 6 = 24
Ein bloedes Beispiel.
40 zur Basis 5 = 20
Ein bloedes Beispiel.
40 ist keine Zahl zur Basis 4
Ein bloedes Beispiel.
40 ist keine Zahl zur Basis 3
Ein bloedes Beispiel.
40 ist keine Zahl zur Basis 2
Ein bloedes Beispiel.
```

### 13. Robuste Programme durch Ausnahmebehandlung

#### 13.1 Grundlagen

#### 13.2 Typen von Ausnahmen

#### 13.3 Behandlung von Ausnahmen

#### 13.4 Weitergabe von Ausnahmen

#### 13.5 Zusammenfassung

- Anstelle einer `try-catch`-Anweisung können Exceptions auch weitergegeben werden.
- In diesem Fall muss die Methode, in der die Exception auftreten kann, gekennzeichnet werden.
- Dazu wird am Ende des Methodenkopfes das Schlüsselwort `throws` mit einer Liste aller Ausnahmen, die auftreten können, angehängt.
- Beispiel:

```
public static double kehrWert(int i) throws ArithmeticException
{
    return 1.0/i;
}
```

- Im Sinne der catch-or-throw-Regel müssen also alle Exceptions (Ausnahme: Unterklassen von `RuntimeException`) entweder behandelt oder weitergegeben werden.

- Die `throws`-Klausel macht dem Compiler und auch allen Aufrufenden alle potentiellen Exceptions, die von der entsprechenden Methode verursacht werden können, bekannt.
- Dadurch kann sowohl der Compiler als auch der Aufrufende sicherstellen, dass bei jedem Aufruf dieser Methode wiederum die catch-or-throw-Regel eingehalten wird.

Wie funktioniert die Behandlung/Weitergabe von Exceptions?

- Tritt eine Exception auf, wird zunächst nach einem umgebenden **try-catch**-Block gesucht, der den Fehler behandelt.
- Ist kein solcher **try-catch**-Block vorhanden, wird die Suche sukzessive in allen umgebenden Blöcken wiederholt.
- Ist dies auch erfolglos, wird der Fehler an den Aufrufer der Methode weitergegeben, wo wiederum blockweise weitergesucht wird.
- Enthalten alle aufrufenden Methoden inklusive der `main`-Methode keinen Code, um den Fehler zu behandeln, bricht das Programm mit einer Fehlermeldung ab.

- Ein wichtiger Aspekt bei der Weiterleitung von Exceptions ist, dass man auch Ausnahmen explizit auslösen kann.
- Exceptions werden mit der Anweisung **throw** `<ExceptionObject>;` ausgelöst.
- Objekte von Ausnahme-Typen können wie alle anderen Objekte erzeugt und verwendet werden, also z.B. auch Variablen (des entsprechenden Typs) zugewiesen werden.

- Die Behandlung einer ausgelösten Exception folgt den vorher skizzierten Regeln.
- Gemäß der catch-or-throw-Regel müssen diese Exceptions also entweder behandelt (mit einer **try-catch**-Anweisung) oder weitergeleitet (mit einer **throws**-Klausel) werden.

```
public static boolean istPrim(int n) throws IllegalArgumentException
{
    if(n<=0)
    {
        throw new IllegalArgumentException("Parameter > 0 erwartet. Gefunden: "+n);
    }
    if(n==1)
    {
        return false;
    }
    for(int i=2; i<=n/2; ++i)
    {
        if(n % i == 0)
        {
            return false;
        }
    }
    return true;
}
```

## 13. Robuste Programme durch Ausnahmebehandlung

### 13.1 Grundlagen

### 13.2 Typen von Ausnahmen

### 13.3 Behandlung von Ausnahmen

### 13.4 Weitergabe von Ausnahmen

### 13.5 Zusammenfassung

Sie kennen jetzt

- das Konzept der Exceptions und deren Behandlung in Java,
- den Unterschied zwischen einer “allgemeinen” Java-Exception und einer Java-RuntimeException,
- einige wichtige vordefinierte Java-Exceptions und deren Vererbungshierarchie,
- die Behandlung von Exceptions mit **try-catch** und ggf. **finally**,
- die Weitergabe von Exceptions mit **throws**,
- das Auslösen von Exceptions mit **throw**.