

# Kapitel 11: Ausnahmebehandlung

- **Ausnahmen (Exceptions)**
  - Signalisierung von Fehlern und unerwünschten Situationen zur Laufzeit.
  - Übersichtliche Programmstrukturierung zur Behandlung von Fehlern.
- **Ansatz**
  - Idee: Trennung der Abläufe von Normalfall und Ausnahmefällen.
  - Normalablauf einer Methode besteht oft nur aus wenigen Anweisungen.
  - Programmtext kann durch integrierte Behandlung von Fehlerfällen unübersichtlich werden; dadurch ist es ggf. schwierig, den regulären Ablauf nachzuvollziehen.
  - Ausnahmen automatisieren zum Teil die Fehlerbehandlung, die sonst oft unterlassen wird (z.B. aus Zeitknappheit).

# Laufzeitfehler

- **Auftreten unerwünschter Fälle**
  - Arithmetische Fehler: Division durch Null, Wurzel aus negativer Zahl, etc.
  - Überschreiten der Indexgrenzen bei Zugriffen auf Arrays.
  - Zugriff auf eine undefinierte Objektreferenz, z.B. `p.toString()` für `p == null`
  - Erreichen der physischen Beschränkung des Hauptspeichers.
  - Fehler beim Öffnen, Lesen und Schreiben von Dateien.
- **Gebräuchliche Reaktionen auf Laufzeitfehler**
  - Weiterarbeiten auf fehlerhaften Zuständen → völlig inakzeptabel.
  - abrupter Abbruch des Programmes → meist ohne Information.
  - Rückgabe von Signalwerten (z.B. `-1`, `null`) → Prüfung unterbleibt oft.
  - Auslösen und Behandeln von Ausnahmen → klares Konzept.

## Beispiel (ohne Ausnahmebehandlung)

- Beispiel: Einlesen einer Menge von Punkten

```
static readPoints (String filename, Point3[] points) {
    java.io.FileReader fr = new java.io.FileReader (filename);
    if (fr == null) {
        System.err.println („cannot read " + filename);
        System.exit(1);
    }
    for (int i = 0; i < points.length; ++i) {
        points[i].read(fr);
        if (points[i] == null) {
            System.err.println („wrong number format in " + filename);
            fr.close();
            System.exit(1);
        }
    }
    fr.close();
}
```

## Ausnahmebehandlung allgemein

- Auftreten von Ausnahmen
  - Ausnahmen werden an Fehlerstellen ausgelöst.
  - Ausnahme werden dann an (höher liegenden) Auffangstellen behandelt.
- Ausnahmebehandlung in Methoden
  - Auffangen und unmittelbar behandeln: **try – catch**
  - Nicht behandeln, sondern in der Aufrufhierarchie nach oben weiterreichen: **throws**
  - Automatische Standardbehandlung bestimmter Laufzeitfehler: Ausgabe einer Beschreibung der Ausnahme, dann Programmabbruch.

# Auffangen von Ausnahmen (**try – catch**)

## Blockstruktur

```
try {                                // Normalablauf, in dem Ausnahmen
    ...                               auftreten können.
} catch (ExceptionType1 e) {        // Behandlung von ExceptionType1
    ...
} catch (ExceptionType2 e) {        // Behandlung von ExceptionType2
    ...
} finally {                           // optional; wird immer ausgeführt
    ...                               (unabhängig von Ausnahmen)
}
```

# Aufbau von **try – catch – finally**

- **try-Block**
  - Wird im try-Block eine Ausnahme ausgelöst, so wird geprüft, welcher der catch-Blöcke diese Ausnahme behandeln kann.
  - Gibt es für eine bestimmte Ausnahme keinen geeigneten catch-Block, so wird diese Ausnahme an den Aufrufer der Methode weitergereicht.
- **catch-Blöcke** (einer oder mehrere)
  - beginnen mit der Deklaration des jeweils behandelten Ausnahmetyps.
  - Ausnahmen bilden in Java eine Vererbungshierarchie → ein catch-Block behandelt gleichzeitig alle abgeleiteten Ausnahmen (Bsp. *IOException*).
- **finally-Block** (optional)
  - wird immer ausgeführt, unabhängig vom Auftreten einer Ausnahme.
  - Typische Aufgaben: Zurückgabe von Betriebsmitteln, Schließen von Dateien und Kommunikationskanälen.

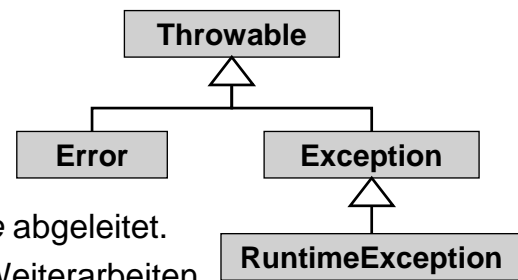
## Beispiel (mit Ausnahmebehandlung)

```
static void readPoints (String filename, Point3[] points) {
    java.io.FileReader fr;
    try {
        fr = new java.io.FileReader (filename);
        for (int i=0; i<points.length; ++i)
            points[i] = fr.read(in);
    } catch (NumberFormatException e) {
        System.err.println ("Wrong number format in " + filename);
        System.exit (1);
    } catch (java.io.IOException e) {
        System.err.println ("Cannot read " + filename);
        System.exit (1);
    } finally {
        fr.close();
    }
}
```

## Weiterreichen von Ausnahmen (**throws**)

- Motivation
  - Bisher: Methoden behandeln Ausnahmen selbst (**try – catch**).
  - Bsp: erneuter Dateizugriff, Meldung an Benutzer, Programmabbruch.
  - Neu: Falls in einer Methode nicht klar ist, wie eine Ausnahme behandelt werden soll, kann die Ausnahme an die aufrufende Methode weitergereicht werden.
- Schlüsselwort **throws**
  - Information für Programmierer sowie für Übersetzer, welche Ausnahmen auftreten können.
  - Deklaration im Methodenkopf durch **throws**-Klausel:  
void readPoints (...) **throws** IOException {...}
  - Aufrufende Methoden müssen dann ihrerseits mit diesen Ausnahmen umgehen, d.h. mit **catch** behandeln oder mit **throws** weiterreichen.

# Ausnahmetypen in Java



- Klassenhierarchie in Java
  - Alle Ausnahmetypen sind von *Throwable* abgeleitet.
  - *Exception* zeigt Ausnahmen an, für die Weiterarbeiten sinnvoll sein kann. Bsp: *CloneNotSupportedException*, *FileNotFoundException*, ...
  - *RuntimeException* für Ausnahmen, die nicht statisch geprüft werden. Bsp: *ArithmeticException*, *ClassCastException*, *NullPointerException*, *IndexOutOfBoundsException*, ...
  - *Error* umfasst Fehler, von denen sich ein Programm nicht erholen kann. Bsp: *OutOfMemoryError*, *StackOverflowError*, *InternalError*, ...
- Klassifikation zur statischen Prüfung
  - *unchecked exceptions*: keine statische Prüfung (*Error*, *RuntimeException*).
  - *checked exceptions*: Prüfung, ob Behandlung stimmt (alle anderen).

# Allgemeine Ausnahmebehandlung

- Allgemeine Konstruktion
 

```

try {
    ...
} catch (Exception e) {
    System.err.println (e.getMessage());
    e.printStackTrace();
    System.exit(1);
}
      
```
- Verwendete Methoden der Klasse *Throwable*
  - String getMessage()*
    - gibt eine erklärende Nachricht zur Ausnahme aus.
  - void printStackTrace()*
    - gibt aktuellen Zustand des Methodenaufkellers aus.

# Definition eigener Ausnahmetypen

- Vorgehensweise
  - Definition durch Erweiterung vorhandener Ausnahmetypen.
  - Möglichst nur „checked exceptions“ einführen (= mit statischer Prüfung).

```

class MyException extends Exception { }
class MyMessageException extends Exception {
    MyMessageException (String msg) {super(msg);}
}

```

- Auslösen eigener oder vordefinierter Ausnahmen mit **throw**
  - if** (*/\*error occurred\*/*) **throw new** MyException();
  - if** (*/\*special error\*/*) **throw new** MyMessageException ("special "+...);
  - if** (*/\*another error\*/*) **throw new** Exception ("particular error message");

# Wann benutzt man Ausnahmen?

- Klassifikation von Systemverhalten

	<i>Unerwartet</i>	<i>Erwartet</i>
<i>Erwünscht</i> (Normalfall)	„Wunder“	regulärer Ablauf
<i>Unerwünscht</i> (Ausnahmefall)	„Katastrophe“	Benutzer-/Systemfehler

- Entwurfsrichtlinien
  - Keine Rückgabe regulärer Ergebnisse durch Ausnahmen:
 

```
class SuccessMessage extends Exception { } // schlecht
```
  - Systemfehler sind immer Ausnahmesituationen.
  - Benutzereingaben können immer fehlerhaft sein.  
Bsp. Tippfehler, Verständnisfehler, Bösartigkeit, ...  
→ Behandlung als Normalfall oder als Ausnahmefall möglich.