Wo sind wir?

- Java-Umgebung
- Lexikale Konventionen
- Datentypen
- Kontrollstrukturen
- Ausdrücke
- Klassen, Pakete, Schnittstellen
- JVM
- **Exceptions**
- Java Klassenbibliotheken
- Ein-/Ausgabe
- Collections
- Threads
- Applets, Sicherheit
- Grafik
- Beans
- Integrierte Entwicklungsumgebungen

Beispiel

```
class Test {
    public static void main(String[] args) {
        int[] a = new int[10];
        int i = 10;

        a[i] = 4711;
    }
}
```

Was ist falsch?

Erlaubte Feldindizes sind 0-9 und nicht 10

Was passiert?

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException
        at Test.main(Test.java:6)
```


Ausnahmebehandlung

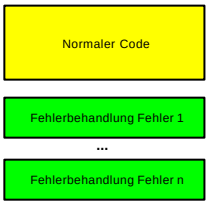
Der Java Compiler versucht zur Übersetzungszeit möglichst viele Fehler zu erkennen. Trotzdem bleiben Fehlerquellen übrig, die erst zur Laufzeit erkannt werden können (Indexverletzung wie letztes Beispiel; Downcasting,...).

Eine Exception ist ein abnormal Ereignis während der Programmausführung.

Tritt eine Exception während der Programmausführung auf (Fehler wird ausgelöst), so unterbricht die JVM den Programmablauf und sucht nach einem geeigneten Exception Handler, der diesen Fehler behandeln kann (Fehler wird abgefangen). Ein Exception Handler ist ein speziell markierter Programmbereich.

War für den Programmbereich, in dem der Fehler auftrat, kein Exception Handler angegeben, so wird die Exception an die aufrufende Methode weitergegeben und dort nach einem Exception Handler gesucht usw. Wird bei dieser Suche kein Exception Handler gefunden, so bewirkt die Exception die Beendigung der Ausführung des Threads (normalerweise Programmabbruch) durch die JVM.

Programmstruktur

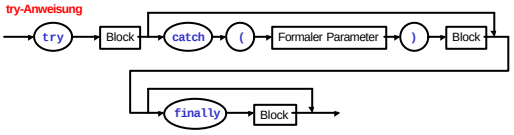


Der "normale" Code wird aus Gründen der Übersichtlichkeit (Fehler ist Ausnahme!) von der Fehlerbehandlung getrennt.

Tritt im normalen Code ein Fehler auf, so wird dieser Block verlassen und zur geeigneten Fehlerbehandlung verzweigt.

```
try { ... }                                 // normaler Code
catch (Exceptiontyp_1 name_1) { ... }       // Behandlung Fehlertyp 1
...
catch (Exceptiontyp_n name_n) { ... }       // Behandlung Fehlertyp n (n=0,1,...)
finally { ... }                             // wird immer durchlaufen (optional)
```


try-Anweisung



Entweder muss mindestens ein catch-Teil vorhanden sein oder ein finally-Teil oder beides. Ein formaler Parameter ist genau wie bei einer Methodendeklaration von der Form "Typ Bezeichner". Der Typ muss die Klasse `java.lang.Throwable` oder eine Unterklasse sein.

Semantik der try-Anweisung

Der try-Block wird ausgeführt. Tritt keine Exception auf, so wird, falls kein finally-Block vorhanden ist, die gesamte Anweisung normal beendet.

Wird während der Ausführung des try-Blocks (inkl. Methodenaufrufen) eine Exception ausgelöst (und damit eine Instanz der Klasse `java.lang.Throwable` erzeugt), so wird die Ausführung des Blocks sofort beendet und der erste catch-Fall, der diese Exception behandeln kann, ausgeführt. Nach Ausführung des catch-Falls ist auch die gesamte Anweisung beendet (aber s.u.) Damit ist diese Exception behandelt und wird nicht weitergereicht.
Wird kein catch-Fall gefunden, so wird die Anweisung beendet und in umschließenden dynamischen Blöcken und Methoden (d.h. Aufrufkette) nach einem geeigneten catch-Fall weitergesucht.

Ist ein finally-Block vorhanden, so wird dieser in jedem Fall (egal ob Exception ausgelöst oder nicht, Exception behandelt oder nicht) vor Verlassen der Anweisung ausgeführt.

Beispiel

```
class Test {
    public static void main(String[] args) {
        int[] a = new int[2];

        try {
            int i = (int)java.lang.StrictMath.log(4711.0); // ist 8
            a[i] = 31415;          // hier rappelt es im Karton
        } catch(ClassCastException e1) {
            System.out.println("Klasse falsch");
            e1.printStackTrace();
        } catch(ArrayIndexOutOfBoundsException e2) {
            System.out.println("Index falsch");
            e2.printStackTrace();
        } catch (Exception e3) {
            System.out.println("Weiss nicht");
            e3.printStackTrace();
        } finally { System.out.println("Das hätten wir geschafft"); }
    }
}
```

Ausgabe:

```
Index falsch
java.lang.ArrayIndexOutOfBoundsException
        at Test.main(Test.java:7)
Das hätten wir geschafft
```


throw-Anweisung

throw-Anweisung

An jeder Stelle, an der eine Anweisung erlaubt ist, kann eine throw-Anweisung eingesetzt werden, um ein sofortiges Auslösen einer Exception des im Ausdruck angegeben Fehlertyps (Objekt vom Typ `java.lang.Throwable` oder Unterklasse) zu bewirken. Die JVM sucht dann (falls vorhanden) den catch-Block, der diese Exception behandelt.

Dies macht meistens Sinn mit eigenen Fehlerkennungen (später).

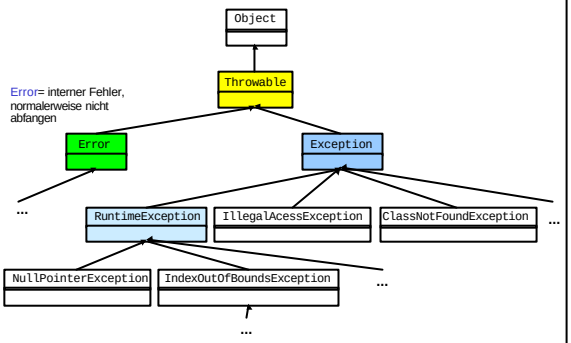
Beispiel

```
class Oberklasse { int x; }
class Unterklasse1 extends Oberklasse { }
class Unterklasse2 extends Oberklasse { }

class Test {
    static void methode(Oberklasse o) throws ClassCastException {
        if(!(o instanceof Unterklasse1))
            throw new ClassCastException();          // hier wird geworfen
        System.out.println("alles ok");
    }

    public static void main(String[] args) {
        try {
            methode(new Unterklasse1());             // sollte ok sein
            methode(new Unterklasse2());             // sollte nicht gehen
        } catch (Exception e) {                      // Exception behandeln
            System.out.println(e.toString());        // Fehlerart ausgeben
        }
    }
}
```

Ausgabe:

```
alles ok
java.lang.ClassCastException
```


Baum der Fehlerklassen



Exceptions bei der Auswertung von Ausdrücken

- `OutOfMemoryError`, wenn bei der Erzeugung von Klasseninstanzen, Feldern oder Strings kein Speicherplatz mehr vorhanden ist
- `ArrayNegativeSizeException`, wenn Feldgrößenangabe kleiner 0
- `NullPointerException`, wenn Zugriff über Referenzen mit Wert null
- `IndexOutOfBoundsException`, wenn Feldindex kleiner 0 oder größer als Felddimension
- `ClassCastException`, wenn explizite Typumwandlung nicht möglich ist
- `ArithmeticException`, wenn in Division/Modulo der rechte Operand 0 ist (nur bei ganzzahliger Operation)
- `ArrayStoreException`, wenn bei Zuweisung an Feldelement Typinkompatibilitäten auftreten

Eigene Fehlerobjekte

Da Throwable eine nicht-finale Klasse ist, lassen sich von ihr auch Klassen ableiten, um z.B. eigene Fehlerkennungen zu erzeugen.

Beispiel:

```
// Dies ist meine Fehlerklasse
class MeineException extends Exception {
    MeineException() {
        // Exception hat u.a. einen Konstruktor mit String-Argument
        super("Meine Fehlerkennung");
    }
}

class Test {
    public static void main(String[] args) {
        try {
            throw new MeineException();              // Exception erzeugen
        } catch (MeineException e) {                 // Exception behandeln
            System.out.println(e.getMessage());      // "Meine Fehlerkennung"
        }
    }
}
```

Folgerung aus Baumhierarchie

Als Folgerung aus der Baumhierarchie und der Semantik der try-catch-Anweisung sollte man spezialisierte Fehler (weit unten in der Baumhierarchie) zuerst mit einem catch-Block abfragen und zuletzt die allgemeinste Fehlerart (oben in der Baumhierarchie). Ansonsten würde durch den allgemeinen Fehler der spezialisierte Fehler schon abgedeckt.

Beispiel:

```
try { a[4711] = 31415;
    } catch(ArrayIndexOutOfBoundsException e1) {
        System.out.println("Index falsch");
    } catch (Exception e2) {
        System.out.println("Weiss nicht");
    }
}
```

Richtig: Zuerst der spezialisierte Fall.

```
try { a[4711] = 31415;
    } catch (Exception e2) {
        System.out.println("Weiss nicht");
    } catch(ArrayIndexOutOfBoundsException e1) {
        System.out.println("Index falsch");
    }
}
```

Falsch: Exception enthält ArrayIndexOutOfBounds Exception.

Checked und Unchecked Exceptions

Bei Exceptions wird unterschieden zwischen:

- Checked Exceptions, die vom Programmierer abgefangen werden müssen, was der Compiler auch überprüft
- Unchecked Exceptions, die nicht vom Programmierer abgefangen werden müssen (aber können), was auch nicht vom Compiler überprüft wird

Alle Exceptions bis auf die Unterbäume `RuntimeException` und `Error` sind checked Exceptions.

Der Grund für diese Aufteilung liegt darin, dass z.B. mit jedem Zugriff auf ein Objekt über eine Referenzvariable ein Fehler auftreten kann, falls nämlich die Referenzvariable den Wert `null` hat. Falls dies eine checked Exceptions wäre, müsste jeder Zugriff überprüft werden, was Programme unlesbar machen würde.

Aus diesem Grunde kann der Programmierer solche Fehler abfangen, tut er dies nicht, so behandelt die JVM diesen Fehler.

Einige Checked und Unchecked Exceptions

Checked Exceptions:
Von `Exception` abgeleitet:

- `ClassNotFoundException` - Klasse wird beim Laden nicht gefunden
- `CloneNotSupportedException` - Objekt unterstützt keine clone-Operation
- `IllegalAccessException` - Klasse hat Methode aufgerufen auf die sie keinen Zugriff

Unchecked Exceptions:
Von `Error` abgeleitet:

- `AbstractMethodError` - Versuch, eine abstrakte Methode aufzurufen
- `InstantiationError` - Versuchtes Anlegen einer Instanz einer abstrakten Klasse
- `OutOfMemoryError` - Kein Speicher mehr verfügbar
- `StackOverflowError` - Stacküberlauf

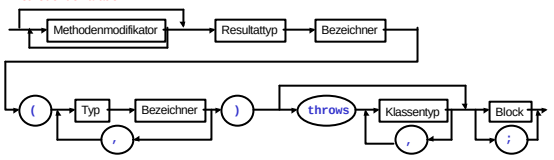
Von `RuntimeException` abgeleitet:

- `ArithmeticException` - int-Division durch 0
- `ArrayIndexOutOfBoundsException` - ungültiger Feldindex
- `ClassCastException` - Typunverträglichkeit bei cast
- `NullPointerException` - Datenfeldzugriff über null versucht

Propagieren von Exceptions

Kann eine Methode eine Exception auslösen, die nicht in der Methode selbst behandelt wird sondern an den Aufrufer weitergereicht (propagiert) wird, so muss (bei checked Exceptions) bzw. kann (bei unchecked Exceptions) im Methodenkopf eine throws-Klausel enthalten sein (bei checked Exceptions ansonsten Übersetzungsfehler).

Methodendeklaration



Beispiel

```
class Test {
    static void methode1(int[] a, int i) throws ArrayIndexOutOfBoundsException {
        a[i] = 4711;                                    // hier rappelt es wieder
    }

    static void methode2(int[] a, int i) {
        try {
            methode1(a, i);                             // hier wird aufgerufen
        } catch(ArrayIndexOutOfBoundsException e) {// hier wird abgefangen
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        int[] a = new int[2];
        methode2(a, 5);
    }
}
```

Ausgabe:

```
java.lang.ArrayIndexOutOfBoundsException
        at Test.methode1(Test.java:3)
        at Test.methode2(Test.java:8)
        at Test.main(Test.java:17)
```

Das Exception-Objekt enthält den Stack Trace vom Auslösen der Exception!