
II.4. Erweiterungen von Klassen und fortgeschrittene Konzepte

- 1. Unterklassen und Vererbung
- 2. Abstrakte Klassen und Interfaces
- 3. Modularität und Pakete
- 4. Ausnahmen (Exceptions)
- 5. Generische Datentypen
- 6. Collections

Ausnahmen (Exceptions)

Treten auf, wenn zur Laufzeit semantische Restriktionen nicht erfüllt werden, z.B.

Arithmetic Exception

■ Arithmetische Ausnahmen:

z.B. Division durch 0, Wurzel aus negativer Zahl, Overflow

■ Unzulässiger Zugriff auf Datenstrukturen:

ArrayIndexOutOfBoundsException

z.B. Zugriff auf Array-Element mit negativem Index oder Index größer als length-1.

z.B. Zugriff auf Eigenschaften eines Objekts über einen Verweis, der `null` ist

NullPointerException

■ Infrastrukturelle Ausnahmen:

z.B. Lesen aus einer Datei, die nicht existiert

IOException

z.B. Fehlschlag bei expliziter Datentypkonvertierung von Ober- zu Unterklasse

ClassCastException

Exception Handling

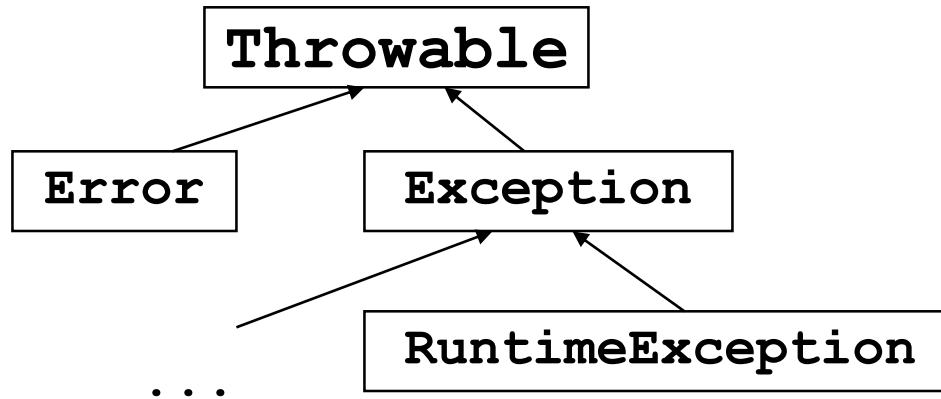
- *Wenn in einem Programmblock Ausnahmen auftreten, wird ein Exception Handler aufgerufen*

```
try { ... Normalblock ... }  
catch (AusnahmeArt1 Parameter1)  
    { .. Exception Handler1 .. }  
catch (AusnahmeArt2 Parameter2)  
    { .. Exception Handler2 .. }  
...  
finally { .. Abschließende Anweisungen .. }
```

- Bei Auftreten einer Ausnahme im Normalblock wird zu dem entsprechenden Exception Handler gesprungen.
- Der **finally** – Block ist optional und wird auf jeden Fall am Ende ausgeführt.

Exception Objekte

Eine Ausnahme ist ein Objekt der Klasse **Throwable**:



- Ausnahmeobjekte werden implizit erzeugt, wenn eine Ausnahme auftritt.
- **Throwable** hat den Konstruktor **Throwable(String m)** und die Methoden **getMessage()**, **printStackTrace()**, **toString()**, etc.
- **Error** und **RuntimeException**: unchecked exceptions, müssen nicht „gefangen“ werden

Beispiele von Exception Klassen

- **IOException** und **IOError**
Fehler in Ein- oder Ausgabe
← keine Runtime Exc. - U-Klasse v. Error
 - **ArithmeticException**
z.B. $x/0$ für int x
 - **ArrayIndexOutOfBoundsException**
Überschreiten des Indexbereichs eines Arrays
 - **ClassCastException**
Fehlschlag bei expliziter Konversion von Ober- zu Unterklasse
 - **NumberFormatException**
Versuch, String, der keine gültige Zahl enthält, in Zahl umzuwandeln
z.B. Integer.parseInt("hallo")
 - **NullPointerException**
Versuch, auf Objektvariable über **null**-Verweis zuzugreifen
- Runtime Exception*

Wo werden Exceptions behandelt

```
public int M1()  
{ .. M2(); ..  
..}  
  
public int M2()  
{ try { .. M3(); ..}  
  catch (A a) { .. } }  
  
public int M3()  
{ .. M4(); ..}  
  
public int M4()  
{ try { .. // Hier wird eine Exception vom Typ A erzeugt.  
  ..}  
  catch (B b) { .. } }
```

- Aufruf von M4 führt zu Exception A. Diese wird im Aufruf von M2 abgehandelt.

Wo werden Exceptions behandelt

```
public int M1()  
{ .. M2(); .. // Hier wird eine Exception vom Typ B erzeugt.  
..}
```

```
public int M2()  
{ try { .. M3(); ..}  
  catch (A a) { .. } }
```

```
public int M3()  
{ .. M4(); ..}
```

```
public int M4()  
{ try { .. // Hier wird eine Exception vom Typ A erzeugt.  
  ..}  
  catch (B b) { .. } }
```

- Aufruf von M4 führt zu Exception A. Diese wird im Aufruf von M2 abgehandelt.
- **Exception B im Aufruf von M1 wird vom Laufzeitsystem abgehandelt.**

Benutzerdefinierte Exceptions

```
public class NegativeNumberException extends Exception {  
    private int value;  
    public NegativeNumberException(int value) {this.value = value;}  
    public int getValue() {return value;}  
}
```

```
public static int fak (int x) throws NegativeNumberException{  
  
    if (x < 0)    throw new NegativeNumberException (x) ;  
  
    if (x > 1)    return x * fak (x - 1) ;  
    else        return 1;  
}
```

```
try{System.out.println ("Fakultaet von " + x + " ist " + fak(x));}  
catch (NegativeNumberException nne) {System.out.println ("Fehler! "  
                                                         + nne.getValue() + " < 0.");}
```

Benutzerdefinierte Exceptions

```
public class NegativeNumberException extends Exception {  
    private int value;  
    public NegativeNumberException(int value) {this.value = value;}  
    public int getValue() {return value;}  
}
```

```
public class TooBigNumberException extends Exception {  
    private int value;                ...  
}
```

```
public static int fak (int x) throws  
    NegativeNumberException, TooBigNumberException{  
    if (x < 0)    throw new NegativeNumberException (x) ;  
    if (x > 12)   throw new TooBigNumberException (x) ;  
    if (x > 1)    return x * fak (x - 1) ;  
    else         return 1;  
}
```

```
try{System.out.println ("Fakultaet von " + x + " ist " + fak(x));}  
catch (NegativeNumberException nne) {System.out.println ("Fehler! "  
    + nne.getValue() + " < 0.");}  
catch (Exception e) {System.out.println ("Fehler! Es trat die  
    folgende Ausnahme auf: " + e );}
```

Benutzerdefinierte Exceptions

```
public class NegativeNumberException extends Exception {...}
public class TooBigNumberException extends Exception {...}
```

```
public static void test() throws Exception {
    int x = SimpleIO.getInt("Bitte Zahl eingeben");
    try{System.out.println ("Fakultaet von " + x + " ist " + fak(x));}
    catch (NegativeNumberException nne) {System.out.println("Fehler! "
        + nne.getValue() + " < 0.");}

    finally {System.out.println ("Ende des try-catch-Blocks");}
    System.out.println ("Ende der Methode test.");}
```

```
public static int fak (int x) throws
    NegativeNumberException, TooBigNumberException{
    if (x < 0)    throw new NegativeNumberException (x) ;
    if (x > 12)   throw new TooBigNumberException (x) ;
    if (x > 1)    return x * fak (x - 1);
    else         return 1;}
```

```
try{test();}
catch (Exception e) {System.out.println ("Fehler! Es trat die
    folgende Ausnahme auf: " + e );}
```