



Java Einführung

Exception Handling

Kapitel 17

Inhalt

- Was sind Exceptions?
- Wie werden sie ausgelöst?
- Wie kann man Exceptions behandeln?
- Erweiterung von Exceptions
- Spezialfall IO

Ausnahmezustände

Im Ablauf eines Programms können unvorhergesehen Probleme auftreten. Die JavaVM führt dann eine Ausnahmebehandlung (Exception) durch.

Ausnahmen sind in Java als Instanzen definiert (siehe: `java.lang.RuntimeException`, `java.io.IOException`, ...)

Wichtige Ausnahmezustände:

- Arithmetic Exceptions (bei Division durch 0 bei int)
- NullPointerException (Instanzname enthält keine Instanz)
- ArrayOutOfBoundsException (unmöglicher Index bei Arrays)
- IOException (Ein/Ausgabeproblem)

Beispiel für eine RuntimeException

```
class SimpleClass { int s; }

class Killer {
    public static void main (String [] args) {
        SimpleClass mySC = new SimpleClass();
        mySC= null;
        mySC.s=10;
    }
}
```

Fehler beim Ausführen und Programm bricht ab:

```
Exception in thread "main"
    java.lang.NullPointerException:
        at Killer.main(Killer.java:7)
```

Beispiel für eine RuntimeException II

```
class Killer2 {  
    public static void main (String [] args) {  
        int n=0;  
        int e = 10/n;  
        System.out.println(e);  
    }  
}
```

Fehler beim Ausführen und Programm bricht ab:

```
Exception in thread "main"  
    java.lang.ArithmeticException: / by zero  
        at Killer2.main(Killer2.java:4)
```

Ausnahmebehandler

try & catch

```
try {  
    Anweisungen;  
} catch (AException e) {  
    Behandlung;  
} catch (BException e) {  
    Behandlung;  
} catch (..  
..  
} finally {  
    Behandlung;  
}
```

- try-Block mit Anweisungen die die Exception auslösen kann
- catch-Blöcke mit den Fehlerbehandlungsanweisungen (wird nach Typ des Parameters bestimmt!)
- finally-Block wird bei jeder Exception am Schluss ausgeführt
- Nach der Behandlung der Exception wird das Programm **fortgesetzt!**

Beispiel zu try & catch

```
class Killer3 {
    public static void main (String [] args) {
        double myArray[] = new double[5]; // Index 0..4
        double v; int i=5;

        try {
            v= myArray[i]; // Indexfehler 5>4!
        } catch (ArrayIndexOutOfBoundsException e) {
            System.err.println("Folgende Exception "
                + " ist aufgetreten: " + e);
        }

        System.out.println("Weiter geht's ...");
    }
}
```

Exceptions in Methoden

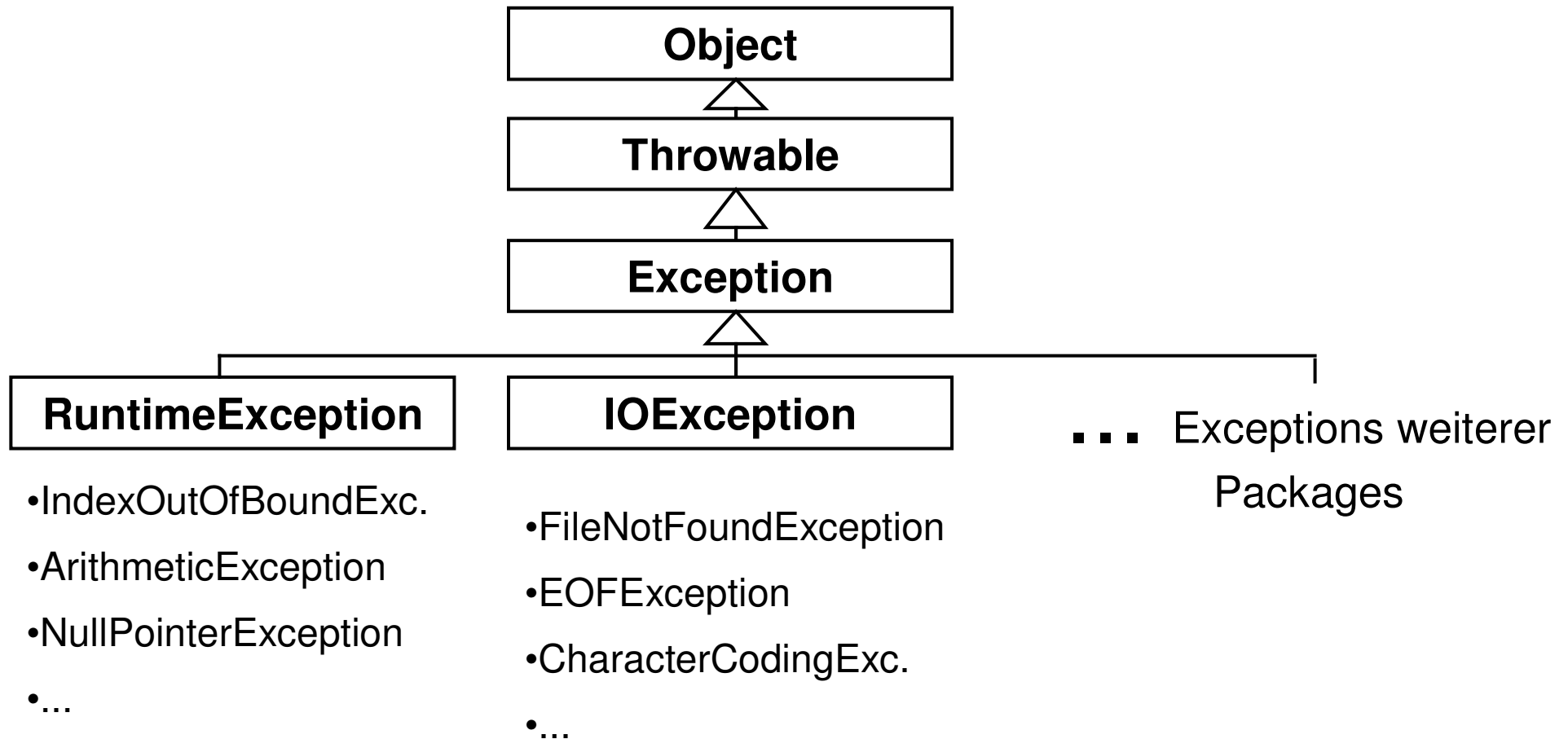
```
class Killer5 {  
    static int divide(int a, int b) throws ArithmeticException {  
        return (a/b);  
    }  
}
```

die Methode kann eine Exception auslösen

```
public static void main (String [] args) {  
    try {  
        System.out.println ("1/0 = "+divide(1,0));  
    } catch(Exception e) {  
        System.out.println("I caught the Exception:"  
            +e.getMessage());  
    }  
    System.out.println("Life goes on!");  
}
```

die aufrufende Methode kümmert sich um
die Behandlung

Klassenhierarchie von Exceptions



Erzeugen einer neuen Exception

```
class FunnyException extends Exception {  
    String joke;  
    FunnyException(String in) { joke=in; }  
    public String getMessage() {  
        return ("I was caused by the joke: "+joke);  
    }  
}  
  
class Killer4 {  
    public static void main (String [] args) {  
        try {  
            throw new FunnyException("Yesterday I was...");  
        } catch(Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

Jede Exceptionklasse kann durch extends spezialisiert werden.

IOException

IO geht in Java prinzipiell über die abstrakten Klassen:

- `InputStream`
- `OutputStream`

Streams können von Tastatur, zu Bildschirm und von/zur Datenträger, Netzwerk usw. gehen

Streams sind `byte`-orientiert und verfügen über:

- einen Konstruktor (=open)
- `int read(byte[] b)` bzw. `void write(byte[] b)`
- `close()`

Als Vereinfachung bei der Eingabe gibt es `BufferedReader` mit der Methode: `String readLine()`

IOException II

Bsp: Einlesen von der Tastatur

```
try {
    /* kann IOExceptions auslösen (siehe BufferedReader und
    * und InputStreamReader in API) */
    BufferedReader inKeyboard = new BufferedReader(new
        InputStreamReader(System.in));
    fileName= inKeyboard.readLine();
} catch (Exception e) {
    System.err.println("Fehler bei der Eingabe: "
        + e.getMessage());
    System.exit(1); // JVM mit Fehlercode beenden
}
```

IOException II

Bsp: Einlesen von der Datei (Ausgabe funktioniert analog mit
FileWriter/BufferedWriter)

```
try {
    /* kann IOExceptions auslösen (siehe BufferedReader und
    * und FileReader in API) */
    FileReader inFile = new FileReader ("Datei.txt");
    BufferedReader in = new BufferedReader (inFile);
    while((buffer = in.readLine()) != null) {
        System.out.println(buffer);
    }
    in.close();
} catch (Exception e) {
    System.err.println("Fehler beim öffnen/lesen: "
        + e.getMessage());
    System.exit(1); // JVM mit Fehlercode beenden
}
```

Was sie nach dieser Einheit wissen sollten...

- die Funktionsweise von Exceptions verstehen.
- wissen wie man in der API Spezifikation Exceptions einzelner Pakete/Klassen findet.
- Exceptions richtig behandeln können.
- das Prinzip von Ein-/Ausgabe in Java verstehen.