EML Research

http://www.eml-research.de/

Java - Part 6

Katja Wegner

katja.wegner@eml-r.villa-bosch.de

Überblick

⇒ Ausnahmebehandlung (Exceptions)

⇒ Input/Output

- Streams/Readers + Writers
- Lesen aus / Schreiben in Dateien

Exceptions

⇒ Behandeln von Ausnahmesituationen

- erlauben, das weitere Ausführen des Programms
- erlauben, das Behandeln von Fehlern (z.B. den Fehler zu reparieren)
- stellen Informationen für den Programmierer bereit (Bugs beheben)
- ermöglichen robustere Programme

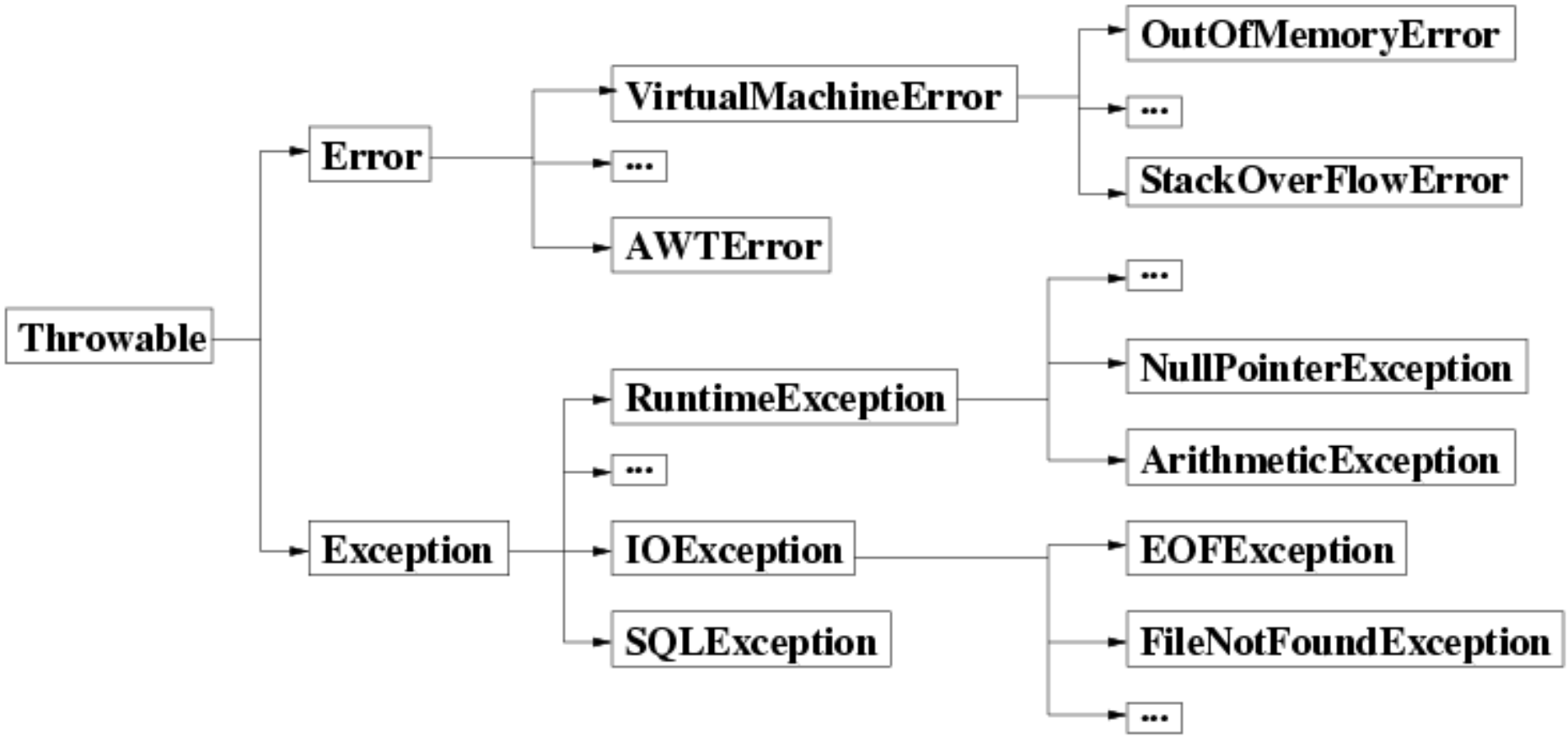
⇒ abgeleitet von der Klasse **Throwable** (oder einer ihrer Unterklassen)

⇒ ausgelöst durch bestimmte Befehle (z.B. Methodenausrufe)

⇒ Beispiele:

- IndexOutOfBoundException
- NullPointerException
- FileNotFoundException

Klassenhierarchie



Ausnahmebehandlung

```
try {
        ...;    // Anweisungen, welche u.U. zu einer Ausnahme führen
}
catch (Exception1 e) {
        ...;    // Ausnahme vom Typ Exception1 behandeln
}
catch (Exception2 e) {
        ...;    // Ausnahme vom Typ Exception2 behandeln
}
finally {
        ...;    // Anweisungen zum Beenden des try-Blocks
                // (müssen auf jeden Fall ausgeführt werden)
}
```

Mit Ausnahmen arbeiten

⇒ Erzeugen von Ausnahmen: **throw**

throw new Exception1(“Die Ausnahme 1 ist aufgetreten.”)

Beispiel:

```
if (x == null) throw new IllegalArgumentException(``null value'')
```

⇒ Deklarieren von Ausnahmen: **throws**

typ methodenName() throws Exception1, Exception2

Beispiel:

```
public int convertStringToInt(String number) throws NumberFormatException {
        int x = Integer.parseInt(number);
        return x;
}
```

Mit Ausnahmen arbeiten (2)

⇒ oft benutzte Methoden:

- $getMessage()$
- $printStackTrace()$

⇒ alle Ausnahmen müssen explizit abgefangen werden (außer: **RuntimeException** und die dazugehörigen Unterklassen, z.B. IllegalArgumentException)

Beispiel

```
public static void main(String args []) {
    int i=0;
    String[] names = new String[5];
    names[0] = "Alma Mater";
    names[1] = "Peter Pan";
    while (i < 5) {
        try { // Nachnamen ausgeben
            int index = names[i].indexOf(" ");
            System.out.println(names[i].substring(index));
        }
        catch (NullPointerException e) {
            names[i] = "John Doe";
            i--; }
        finally {
            i++; }
    } }
```

6.1 Übung

Schreiben Sie eine Klasse MyExceptionTest, die den Code des Beispiels enthält.

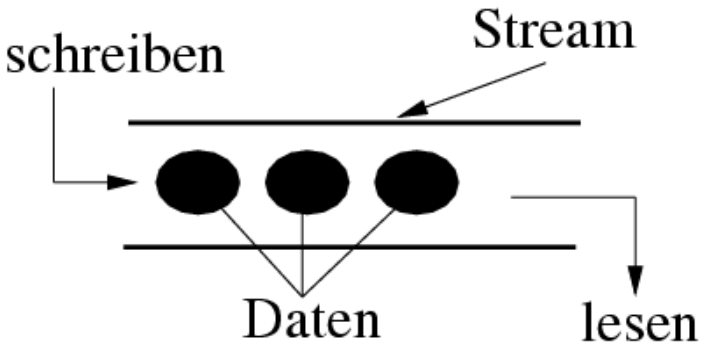
Fügen Sie bei der Initialisierung der String-Elemente einen Namen ohne ein Leerzeichen hinzu und gucken Sie was passiert.

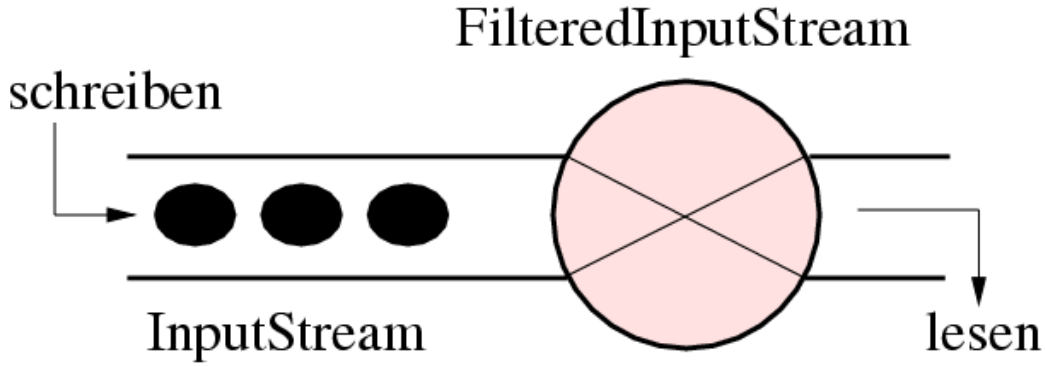
Benutzen Sie eine zweite catch-Anweisung, um diese Problem zu beheben.

Streams - Datenströme

⇒ sind Quelle oder Ziel einer Reihe von **bytes**

⇒ haben immer eine Richtung





Streams (2)

⇒ Basistypen:

- $InputStream$
- $OutputStream$

⇒ Standardtypen:

- $InputStream\ System.in$

  (Standardinput, z.B. von der Shell)
- $PrintStream\ System.out$

  (Standardausgabe, z.B. in die Shell)
- $PrintStream\ System.err$

  (Standardfehlerausgabe, z.B. in die Shell)

Basismethoden für Input Streams

⇒ Lesen:

int read()	liest ein byte
int read(byte[] b)	liest ein Feld von bytes
int read(byte[] b, int i, int n)	liest maximal n bytes in Feld b beginnend am Index i

⇒ Andere Methoden:

void close()	schliest den Stream
int available()	gibt die Anzahl der bytes im Stream zurück
long skip(longn)	überspringt die nächsten n bytes

Basismethoden für Output Streams

⇒ Lesen:

int write()	schreibt ein byte
int write(byte[] b)	schreibt ein Feld von bytes
int write(byte[] b, int i, int n)	schreibt maximal n bytes in Feld b beginnend am Index i

⇒ Andere Methoden:

void close()	schliest den Stream
void flush()	schreibt alle (eventuell gepufferte) bytes im Stream werden raus geschrieben

Spezielle Streams

⇒ **FileInputStream / FileOutputStream**

Klassen zum Lesen aus und Schreiben in Dateien

⇒ **BufferedInputStream / BufferedOutputStream**

Klassen, die die Effizienz erhöhen, wenn mehrere bytes gelesen oder geschrieben werden (z.B. $readLine()$)

⇒ **PipedInputStream / PipedOutputStream**

Klassen, die für die Kommunikation ziwschen Prozessen benutzt werden können

Spezielle Streams (2)

⇒ **DataInputStream / DataOutputStream**

- Klassen zum Lesen und Schreiben von primitiven Datentypen (boolean, char, short, int, long, float, double, ...)

boolean readBoolean()	*void writeBoolean()*
int readInt()	*void writeInt()*
double readDouble()	*void writeDouble*
...	...

Readers und Writers

⇒ JDK 1.1

⇒ lesen und schreiben Unicode (16 bit) Zeichen (keine bytes wie Streams)

⇒ mehr Funktionalität

⇒ auch für plattformabhängige Zeichensätze

```
InputStreamReader in = new InputStreamReader(System.in, "latin-1");
OutputStreamWriter out = new OutputStreamWriter(System.out, "latin-1");
```

Streams vs. Readers / Writers

Reader	**InputStream**
BufferedReader	BufferedInputStream
FileReader	FileInputStream
FilterReader	FilterInputStream
StringReader	StringBufferInputStream
PipedReader	PipedInputStream
Writer	**OutputStream**
BufferedWriter	BufferdOutputStream
FileWriter	FileOutputStream
FilterWriter	FilterOutputStream
PrintWriter	PrintStream
...	...

Beispiel - Lesen einer Datei

```
File myInfile = new File("name-of-file"); // erzeugt einen abstrakten Pfadname
try {
  FileReader fileRead = new FileReader(myInfile); // erzeugt Reader

  BufferedReader bufRead = new BufferedReader(fileRead); // puffert die Daten

  String line = bufRead.readLine();  // liest die erste Zeile der Datei
  while (line != null) {
        .....
        line = bufRead.readLine();
  }    //end while
  bufRead.close();
}     //end try
catch (IOException e) { ... }
```

Beispiel - Schreiben in eine Datei

```
File myOutfile = new File("name-of-file");      // erzeugt ein Dateiobjekt

try {
        FileWriter fw = new FileWriter(myOutfile);
        String s;
        while (  <some-condition> ) {
                s = ....;
                fw.write(s);
                fw.write(System.getProperty("line.separator"));
        } // end while
        fw.close();
}  // end try
catch (IOException e)  { ...}
```

Beispiel mit Ausnahmebehandlung

```
Vector objVec = new Vector(10,5);
try {
      FileInputStream fileInStr = new FileInputStream(filename);
      ObjectInputStream objInStr = new ObjectInputStream(fileInStr);

      while ( )
          objVec.add (objInStr.readObject());
}  // end try
catch(EOFException eofe) {
     objVec.trimToSize();
     System.out.println("Number of Objects read: " + objVec.size());
}
catch(IOException) {...}
catch (ClassNotFoundExcepetion cnfe) {....}
finally { return objVec; }
```

6.2 Übung

Schreiben Sie eine Klasse mit einer grafischen Oberfläche, die einen $FileDialog$ enthält. Der Benutzer soll sich eine Datei aussuchen können, die entweder in eine neue Datei kopiert oder auf dem Bildschirm anzeigt wird.