



Musterlösung: Wortstatistik

```
▶ import java.io.*;

▶ class WordsStat
{
▶   static boolean isWhiteSpace(char c)
  {
    return c == ' ' || c == '\t' || c == '\n' || c == '\r';
  }
▶   static boolean isPunctuation(char c)
  {
    return c == '.' || c == '!' || c == '?' || c == ':' || c == ';' ||
           c == '"' || c == '(' || c == ')' || c == ',' || c == '-';
  }
▶   static boolean ignore(char c)
  {
    return isWhiteSpace(c) || isPunctuation(c);
  }
}
```



Musterlösung: Wortstatistik

```
▶ static String[] readWords(FileInputStream stream,int c,int index) throws IOException
{
▶ while(c != -1 && ignore((char) c))
    c = stream.read();
▶ if(c != -1)
    {
    String word = "";
    while(c != -1 && !ignore((char) c))
    {
    word += (char) c;
    c = stream.read();
    }
▶ String[] words = readWords(stream,c,index + 1);
words[index] = word.toLowerCase();
return words;
    }
▶ else
    return new String[index];}
```



Musterlösung: Wortstatistik

```
▶ static void count(String[] s)
{
    int i = 0;
▶ while(i < s.length)
    {
        String word = s[i++];
        int num = 1;
▶ while(i < s.length &&
        word.equals(s[i++]))
            ++num;
▶ System.out.println(num + "\t" + word);
    }
}
▶ static void swap(String[] s,int i, int j)
{
    String t = s[i]; s[i] = s[j]; s[j] = t;
}
```

```
▶ static void sort(String[] s)
{
    for(int i = 0; i < s.length-1; ++i)
        for(int j = i+1; j < s.length; ++j)
            if(s[j].compareTo(s[i]) < 0)
                swap(s,i,j);
}
▶ static void main(String[] args) throws
    FileNotFoundException,IOException
{
▶ FileInputStream stream =
    new FileInputStream(args[0]);
▶ String[] words =
    readWords(stream,stream.read(),0);
▶ sort(words);
▶ count(words);
}
```



Beispiel: Wortstatistik

```
1      terrific
4      that
17     the
1      then
1      they'll
2      thing
5      this
1      thought
11     to
1      trappings
1      turned
1      up
3      very
2      was
1      way
1      we'll
3      what
1      who
3      will
2      world
4      would
1      write
1      writers
2
```



Ausnahmen (Exceptions)

▶ Erzeugen (throw)

- ▶ *Exceptions* dienen zum Mitteilen von Fehlerzuständen
- ▶ An der Stelle, an der die Exception ausgelöst wird, wird die Ausführung des Programms unterbrochen und an anderer Stelle wieder aufgenommen

▶ Fangen (catch)

- ▶ Das Behandeln einer Exception bezeichnet man als *fangen*
- ▶ Wird eine Exception nicht gefangen, wird das Programm abgebrochen und eine Fehlermeldung ausgegeben

▶ Erzwingen der Behandlung

- ▶ Durch die Angabe einer *throws*-Klausel hinter dem Funktionskopf werden Aufrufer der Funktion gezwungen, die genannte Exception zu behandeln
- ▶ Allerdings kann der Aufrufer ebenfalls eine *throws*-Klausel hinter seinem Funktionskopf verwenden, um die Behandlung an seinen Aufrufer weiterzuleiten



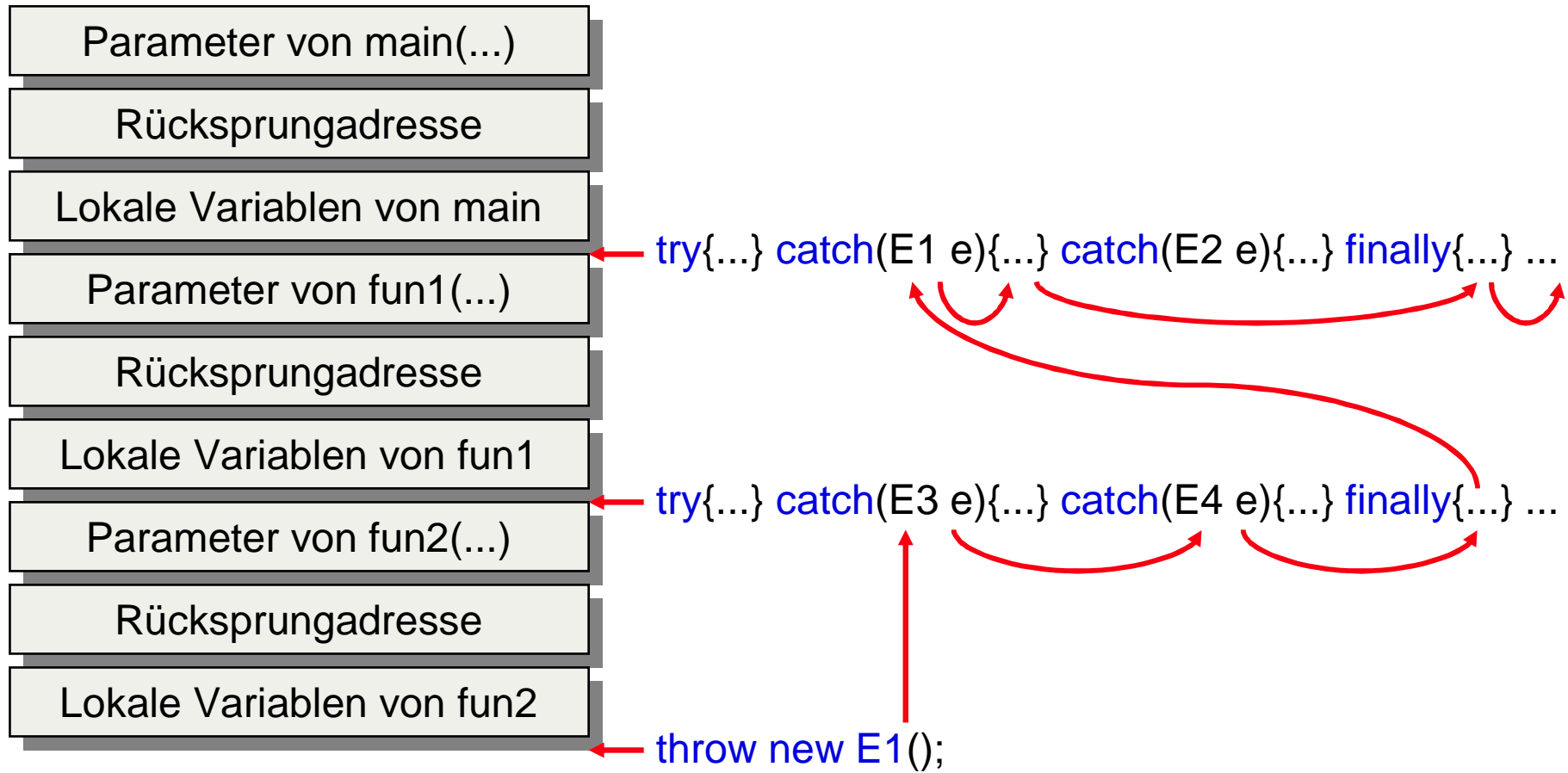
Ausnahmen: Syntax

```
funktionskopf [ throws exceptiontyp { , exceptiontyp } ]
```

```
throw new exceptiontyp ( [ ausdrück { , ausdrück } ] );
```

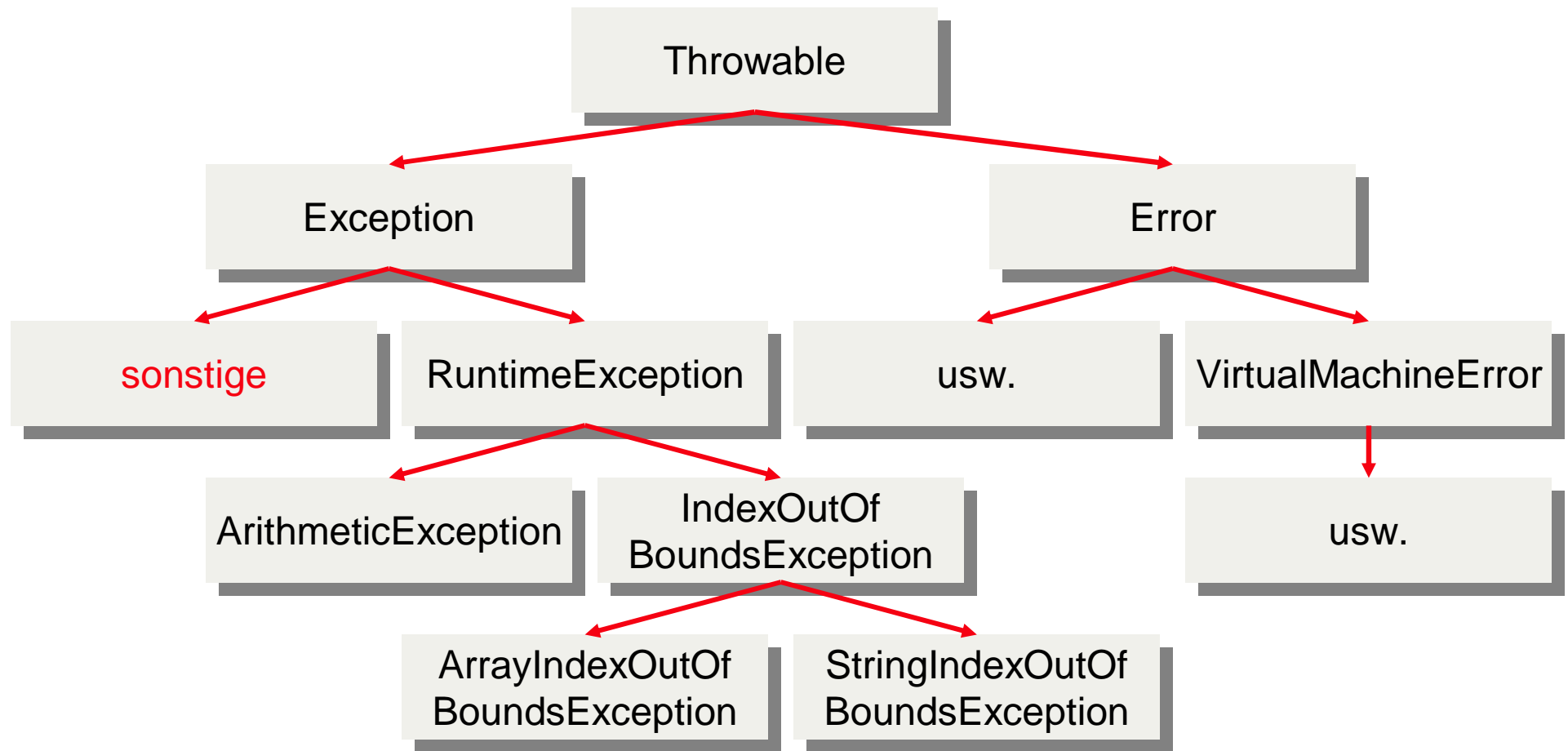
```
try  
{  
    { anweisung }  
}  
{ catch ( exceptiontyp bezeichner )  
{  
    { anweisung }  
}}  
[ finally  
{  
    { anweisung }  
}]
```

Der Weg einer Ausnahme





Die Exception-Hierarchie





Wortstatistik mit Exception-Handling

```
▶ static void main(String[] args)
{
▶   if(args.length != 1)
      System.out.println(
        "usage:\tjava WordsStat file");
▶   else
      try
      {
▶     FileInputStream stream =
        new FileInputStream(args[0]);
        String[] words =
          readWords(stream,stream.read(),0);
          sort(words);
          count(words);
      }
}
```

```
▶   catch(FileNotFoundException e)
      {
        System.out.println(args[0] +
          ": no such file or directory");
      }
▶   catch(IOException e)
      {
        System.out.println(args[0] +
          ": file is corrupted");
      }
    }
}
```



Beispiel: Exception-Handling

```
3      very
2      was
1      way
1      we'll
3      what
1      who
3      will
2      world
4      would
1      write
1      writers

>java WordsStat
usage: java WordsStat file

>java WordsStat
usage: java WordsStat file

>java WordsStat djfhds
djfhds: no such file or directory

>java WordsStat d:layout.bin
d:layout.bin: file is corrupted

>
```



BinTree mit Exceptions

```
▶ BinTree left() throws Exception
{
▶   if(empty())
      throw new Exception("Zugriff auf leeren Baum!");
      else
      return leftBranch;
}
▶ BinTree right() throws Exception
{
      :
}
▶ int value() throws Exception
{
  if(empty())
    throw new Exception("Zugriff auf leeren Baum!");
  else
    return val;
}
```



BinTree mit Exceptions

```
▶ static public void main(String[] args)
{
▶ try
{
    BinTree b1 = new BinTree();
    BinTree b2 = new BinTree();
    System.out.println(new BinTree(b1,42,b2).left() == b1);
                        :
    System.out.println(new BinTree().left());
}
▶ catch(Exception e)
{
    System.out.println(e.getMessage());
}
}
```



Beispiel: BinTree mit Exceptions

```
>javac BinTree.java  
  
>java BinTree  
true  
true  
true  
true  
true  
true  
Zugriff auf leeren Baum!  
  
>_
```



Dokumentieren von Quelltexten

▶ **Klassen**

- ▶ Allgemeine Beschreibung, was die Aufgabe der Klasse ist

▶ **Member-Variablen**

- ▶ Was wird in der Variablen gespeichert?
- ▶ Haben bestimmte Belegungen besondere Bedeutungen?

▶ **Member-Funktionen**

- ▶ Was tut die Funktion?
- ▶ Welche Parameter hat sie und was bedeuten sie?
- ▶ Welches Ergebnis liefert die Funktion?
- ▶ Welche Seiteneffekte hat die Funktion?
- ▶ Was sind die Vorbedingungen für die Anwendbarkeit der Funktion?
- ▶ Welche Nachbedingungen sind erfüllt?



Dokumentieren von Quelltexten

▶ In externen Dokumenten

- ▶ Wie ist die Gesamtarchitektur des Programms?
- ▶ Wie arbeiten die Klassen zusammen?
- ▶ Wie benutzt man das Programm?

▶ Kommentare in Java

- ▶ Zeilen-Kommentar: `// Dies ist ein Kommentar bis zum Zeilenende`
- ▶ Block-Kommentar: `/* Dies ist ein Kommentar bis zur Endmarkierung */`
- ▶ JavaDoc-Kommentar: `/** Dies ist ein JavaDoc-Kommentar */`

▶ Spezialitäten in JavaDoc-Kommentaren

- ▶ `@author` Hier steht der Author
- ▶ `@version` Hier wird die Versionsnummer angegeben
- ▶ `@param` Hier wird ein Funktionsparameter beschrieben
- ▶ `@return` Hier wird der Rückgabewert beschrieben
- ▶ `@throws` Hier wird eine möglicherweise von einer Funktion erzeugte Exception beschrieben



Stack mit JavaDoc-Kommentaren

```
▶ /**
 * The class Stack realizes an unlimited
 * stack for integer numbers. The current
 * implementation is based on a linked list.
 * @author Thomas R&ouml;fer
 */
▶ public class Stack
{
▶ /**
 * The local class StackElement represents
 * a single element on the stack. It consists
 * of the value to store and a reference to
 * the next element on the stack.
 */
▶ private class StackElement
{
    StackElement next;
    int value;
}
```

```
▶ /**
 * The private member first is a reference
 * to the first element on the stack. If it is
 * null, the stack is empty.
 */
    private StackElement first;
▶ /**
 * The only constructor of the class.
 */
    public Stack()
    {
        first = null;
    }
▶ /**
 * The function puts a number on top of
 * the stack.
 * @param i The value pushed onto the
 * stack.
 */
```



Stack mit JavaDoc-Kommentaren

```
▶ public void push(int i)
  {
    StackElement s = new StackElement();
    s.value = i;
    s.next = first;
    first = s;
  }
▶ /**
 * The function removes the topmost element from the stack
 * and returns it. <b>Note:</b> The stack must not be empty!
 * @return The value removed from the stack.
 */
▶ public int pop()
  {
    int i = first.value;
    first = first.next;
    return i;
  }
}
```



Beispiel: Dokumentieren mit JavaDoc

```
>javadoc stack.java
Loading source file stack.java...
Constructing Javadoc information...
Building tree for all the packages and classes...
Building index for all the packages and classes...
Generating overview-tree.html...
Generating index-all.html...
Generating deprecated-list.html...
Building index for all classes...
Generating allclasses-frame.html...
Generating index.html...
Generating packages.html...
Generating Stack.html...
Generating serialized-form.html...
Generating package-list...
Generating help-doc.html...
Generating stylesheet.css...

>_
```



Beispiel: Dokumentieren mit JavaDoc

The screenshot shows a JavaDoc page for the `Stack` class. On the left, there is a sidebar titled "All Classes" with a link to `Stack`. The main content area has a navigation bar with links for "Class", "Tree", "Deprecated", "Index", and "Help". Below this, there are links for "PREV CLASS", "NEXT CLASS", "FRAMES", and "NO FRAMES", and "DETAIL: FIELD", "CONSTR", and "METHOD". The main content is titled "Class Stack" and shows the class hierarchy: `java.lang.Object` is the superclass, and `Stack` is the subclass. The class signature is `public class Stack extends java.lang.Object`. A description follows: "The class Stack realizes an unlimited stack for integer numbers. The current implementation is based on a linked list."

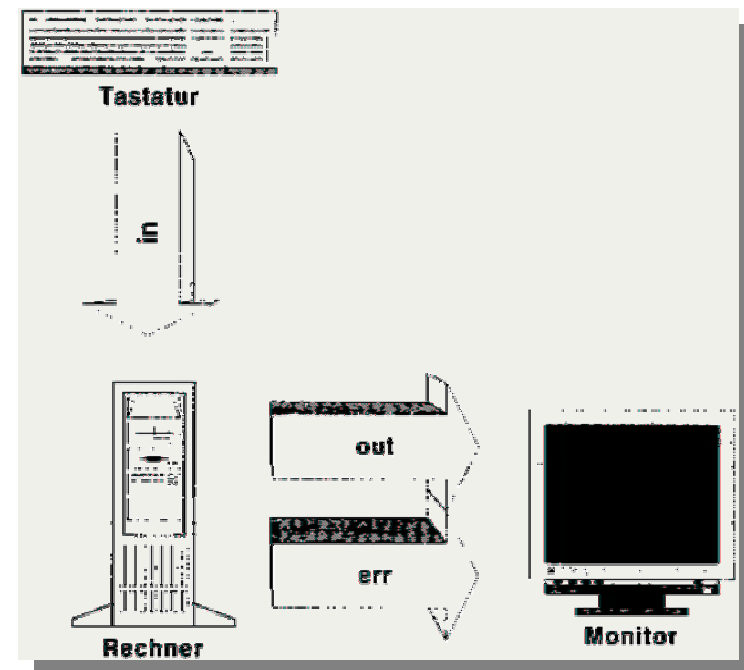
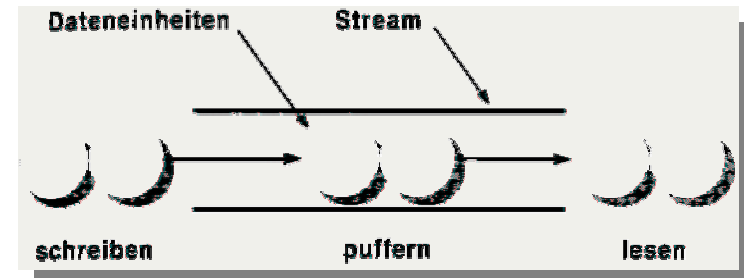
Datenströme

▶ Allgemein

- ▶ Datenströme leiten Daten seriell in ein Programm hinein bzw. wieder heraus
- ▶ Datenströme können umgeleitet werden, so kann die Eingabe aus einer Datei kommen bzw. die Ausgabe in eine Datei geschrieben werden

▶ Standard-Datenströme

- ▶ Standardeingabe
 - ▶ *System.in* (Instanz von *InputStream*)
- ▶ Standardausgabe
 - ▶ *System.out* (Instanz von *PrintStream*)
- ▶ Standardfehlerausgabe
 - ▶ *System.err* (Instanz von *PrintStream*)





Datenströme in Java

▶ Byte-orientierte Datenströme

▶ InputStream

▶ *FileInputStream, FilterInputStream, StringBufferInputStream ...*

▶ OutputStream

▶ *FileOutputStream, FilterOutputStream, PrintStream ...*

▶ Unicode-orientierte Datenströme

▶ Reader

▶ *BufferedReader, FilterReader, InputStreamReader, StringReader ...*

▶ Writer

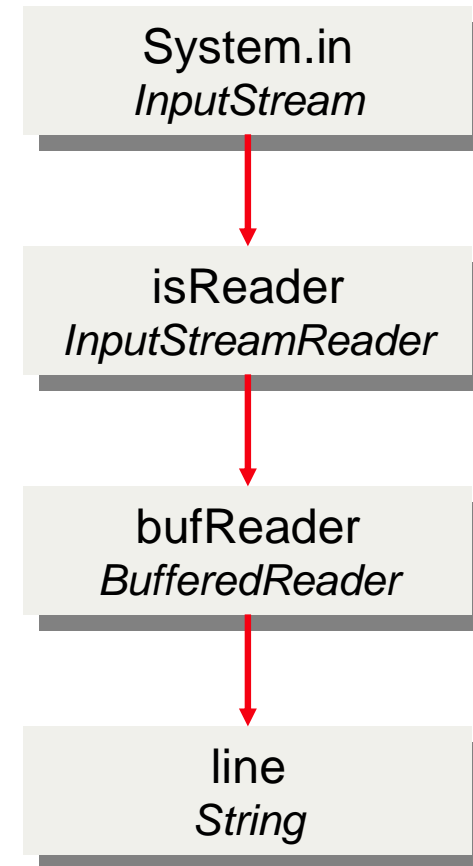
▶ *BufferedWriter, FilterWriter, OutputStreamWriter, PrintWriter, StringWriter ...*



Datenströme in Java

```
▶ import java.io.*;

▶ class More
{
    static void main(String[] args) throws IOException
    {
        ▶ InputStreamReader isReader =
            new InputStreamReader(System.in);
        ▶ BufferedReader bufReader =
            new BufferedReader(isReader);
        ▶ String line = bufReader.readLine();
        ▶ while(line != null)
        {
            System.out.println(line);
            line = bufReader.readLine();
        }
    }
}
```





Beispiel: More

```
>javac More.java  
>java More <Text  
-
```



Übungsaufgaben

▶ Aufgabe 9

- ▶ Mindestens eines von den bisher erstellten Programmen „wasserdicht“ machen, d.h. alle zu erwartenden Exceptions auffangen und entsprechende Fehlermeldungen ausgeben

▶ Aufgabe 10

- ▶ Mindestens eines von den bisher erstellten Programmen JavaDoc-konform dokumentieren und mit JavaDoc eine Online-Dokumentation erstellen

▶ Aufgabe 11

- ▶ Erstellen von zwei Programmen *Zip* und *UnZip*, die eine Datei komprimieren bzw. eine komprimierte Datei wieder auspacken

▶ Tipp

- ▶ Es gibt zwei Klassen `GZIPInputStream` und `GZIPOutputStream`, die das Ein- und Auspacken übernehmen