Programmierung 2
Studiengang MI / WI

- Dipl.-Inf., Dipl.-Ing. (FH) Michael Wilhelm
- Hochschule Harz
- FB Automatisierung und Informatik
- mwilhelm@hs-harz.de
- http://mwilhelm.hs-harz.de
- Raum 2.202
- Tel. 03943 / 659 338

Inhalt der Vorlesung

Überblick:

- Objekte und Methoden
- Swing
- **Exception**
- **I/O-Klassen / Methoden**
- Algorithmen (Das Collections-Framework)
- Design Pattern
- Graphentheorie
- JUnit

Java: Exception

```
public class exception1 {

  public static void main(String args[]) {
    int j,k;
    j=0;
    k=3 / j;
  }  // main
}
```

Java: Exception

```
D:\HS-Harz\Prog2>java exception1
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at exception1.main(exception1.java:8)

D:\HS-Harz\Prog2>
```

Java: Exception

Ein perfektes Programm mit perfekten Anwendern benötigt keine Fehlerbehandlung. Alle Daten werden korrekt eingegeben, jede Datei existiert und alle Module sind richtig programmiert.

Reale Welt:
Beim Absturz eines Programms entstehen:
•Datenverluste
•Unstimmigkeiten in einer Datenbank
•Neueingaben
•Ärger des Anwenders

Abhilfe:
•Benachrichtigung an den Anwender
•Sicherung der Daten
•Sicheres Beenden des Programms

Java: Exception

Definition:
public class Exception extends Throwable

The class Exception and its subclasses are a form of Throwable that indicates conditions that a reasonable application might want to catch.

Since:
JDK1.0

Java benutzt für die Fehlerbehandlung ein „error trapping“ bzw. „exception handling“ (Delphi, C++). Diese Fehlerbehandlung ist weit flexibler als die VB Basicvariante „On error goto“.

Java: Exception

Ursachen für eine Exception:

•Fehlerhafte Eingabe (Numerische Eingabe, URL)
•Gerätfehler (Drucker, Webseite, Diskette)
•Physikalische Grenzen (mangelnder Speicher, Freier Speicherplatz)
•Programmierfehler (Arrayindex, Stackfehler, Overflow, Underflow)

Exception:

Bei einer Exception wird

- die aktuelle Prozedur sofort verlassen
- es wird ein Exceptionobjekt erzeugt
- es wird kein Returncode erzeugt
- der Aufrufcode wird nicht weiterabgearbeitet
- es beginnt die Suche nach einem „exception handler“, der für diese Exception zuständig ist

Java: Exception-Arten

Standard Runtime Exceptions:

ArithmeticException:	An exceptional arithmetic situation has arisen, such as an integer division operation with a zero divisor.
ArrayStoreException:	An attempt has been made to store into an array component a value whose class is not assignment compatible with the component type of the array
ClassCastException:	An attempt has been made to cast a reference to an object to an inappropriate type.
IllegalArgumentException:	A method was passed an invalid or inappropriate argument or invoked on an inappropriate object. Subclasses of this class include:
IllegalThreadStateException:	A thread was not in an appropriate state for a requested operation.
NumberFormatException:	An attempt was made to convert a String to a value of a numeric type, but the String did not have an appropriate format.
IllegalMonitorStateException:	A thread has attempted to wait on or notify other threads waiting on an object that it has not locked.
IndexOutOfBoundsException:	Either an index of some sort (such as to an array, a string, or a vector) or a subrange, specified either by two index values or by an index and a length, was out of range.

Java: Exception

Package java:

java.io.IOException:	Oberklasse alle IO-Exception A requested I/O operation could not be completed normally. Subclasses of this class include:
java.io.EOFException:	End of file has been encountered before normal completion of an input operation.
java.io.FileNotFoundException:	A file with the name specified by a file name string or path was not found within the file system.
java.io.InterruptedIOException:	The current thread was waiting for completion of an I/O operation, and another thread has interrupted the current thread, using the interrupt method of class Thread (§ 20.20.31).
java.io.UTFDataFormatException:	A requested conversion of a string to or from Java modified UTF-8 format could not be completed (§ 22.1.15, § 22.2.14) because the string was too long or because the purported UTF-8 data was not the result of encoding a Unicode string into UTF-8.

Standard Runtime Exceptions:

NullPointerException:	An attempt was made to use a null reference in a case where an object reference was required.
SecurityException:	A security violation was detected (§ 20.17).

Java: Exception (Anfangsbeispiel)

```
import java.io.*;

public class exception1 {

  public static void main(String argv[]) {
    int j,k;
    j=0;
    k=3 / j;
  }  // main
}
```

Java: Exception (interne Abfrage)

```
import java.io.*;

public class exception2 {

  public static void main(String argv[]) {
    int j,k;
    j=0;
    try {          // Excecption Block
      k=3 / j;
      j=k*k;
      save (j)   // speichern in Datei
    }
    catch (ArithmeticException f) {
        System.err.println(" ArithmeticException : " + f);
    }
  }  // main
}
```

Kein Absturz für den Anwender

exception2

Java: Exception (externe Abfrage)

```
import java.io.*;

public class exception3 {

  public static double loadDouble(String sFileName) {
     // hier könnte ein Fehler auftreten
    return 1.0;   //  Platzhalter
  }

  public static void main(String argv[]) {
    double d;
    d = loadDouble("c:\\1.dat");
   }  // main
}
```

exception3

Java: Exception (externe Abfrage)

```
import java.io.*;

public class exception4 {

  public static double loadDouble(String sFileName) throws IOException {
    return 1.0;
  }

  public static void main(String argv[]) {
    double d;
    d = loadDouble("c:\\1.dat");  // Zwang zur Benutzung einer Exception
  }  // main
}
```

Compilermeldung:

Nicht bekannte java.IO.IOException; muss abgefangen werden oder zum Auslösen deklariert werden

Java: Exception

```
import java.io.*;

public class exception4 {

  public static double loadDouble(String sFileName) throws IOException {
    return 1.23;
  }

  public static void main(String argv[]) {
  double d;
   try {
       d = loadDouble("c:\\1.dat");
   }
    catch (IOException f) {
         System.err.println("IOException: " + f);
    }
  }  // main
} // class
```

Exception4

Java: Throw Exception

```
public static double loadDouble(String sFileName) throws IOException {
  int i=1;
  if (i==1) throw new EOFException("Modul loadDouble (Header hat eine falsche Version)");
  return 1.234;
}  // loadDouble

public static void main(String argv[]) {
  double d;
  try {
    d = loadDouble("c:\\1.dat");
  }
  catch (EOFException f) {
       System.err.println("main: " + f);
  }
  catch (IOException f) {
       System.err.println("main: " + f);
  }
}  // main
```

Java: Throw Exception

Ergebnis:

```
D:\HS-Harz\GUI>java exception5
main: java.io.EOFException: Modul loadDouble (Header)

D:\HS-Harz\GUI>
```

Java: Eigene Exceptionklasse

- Exception sind in eine Klassenhierarchie eingebettet.
- Sie dienen unterschiedlicher Aufgaben.
- Wünschenswert ist es manchmal, eine eigene Klasse für die Abarbeitung zu definieren.
- Eigene Hierarchie
- Debug Informationen

Java: Eigene Exceptionklasse

Definition:

```
class HeaderFalseVersion extends IOException {
        int version;
        public HeaderFalseVersion() {
                super();
        }

        public HeaderFalseVersion(String error) {
                super(error);
        }
        public HeaderFalseVersion(String error, int version) {
                super(error);
                this.version = version;
        }

}  // HeaderFalseVersion
```

Java: finally Klausel

Beim Auslösen einer Exception wird sofort das jeweilige Modul sofort verlassen.

Problem:

- Lokale Resourcen werden nicht sauber entfernt:
- Eine Datei ist geöffnet, wird nicht geschlossen

Abhilfe:

finally Klausel

Java: finally Klausel

```
try {
  Resource resource = ...;
  resource.methodThrowsException();
}
catch (Exception e) {
 ...
}
finally {
   resource.close();
}
```

Java: finally Klausel

```
void createGraphic(String sFileName) {

  Graphics g = image.getGraphics(sFileName);
  try {
    code mit der Möglichkeit einer Exception
    g.dispose();  // Speicher freigeben
    ok = true;
  }
  catch(IOException e) {
    ok = false;
  }

}
```

Java: finally Klausel

```
void createGraphic(String sFileName) {
  Graphics g = image.getGraphics(sFileName);
  try {
     code mit der Möglichkeit einer Exception
     ok = false;
  }
  catch(IOException e) {
     ok = false;
  }
  finally {
     g.dispose();  // Speicher freigeben
  }

}
```

```
 FileInputStream fin=null;
 DataInputStream din=null;
 byte b;
 try {
   fin = new FileInputStream( sFilename );
   din = new DataInputStream(fin);
   while  ( din.available()>0 ) {
     b  = din.readByte();
     System.out.println((char) (b) );
   }
 }
catch ( FileNotFoundException e )  {
  System.err.println( "Die Datei ist nicht vorhanden!" );
}
catch ( IOException e )  {
  System.err.println( "Schreib-/Leseprobleme!" );
}
finally {
  if ( fin != null )
   try { fin.close(); }
   catch ( IOException e ) {
    e.printStackTrace();
   }
}
```

Java: Anwendung von Exceptions

Vier Regeln:

1) Exception-Handling ersetzt nicht das Testen.

   Das Auslösen einer Exception kostet sehr viel Zeit.

2) Splitten von Exceptions

   Wenn überhaupt nur eine try Klammer mit mehreren catch-Klauseln.

3) Kein „Verstecken“ von Exceptions

   Eine catch-Anweisung ohne eine Meldung ist nicht sinnvoll.
   Tritt der Fehler auf, erhält der Anwender keine Mitteilung.

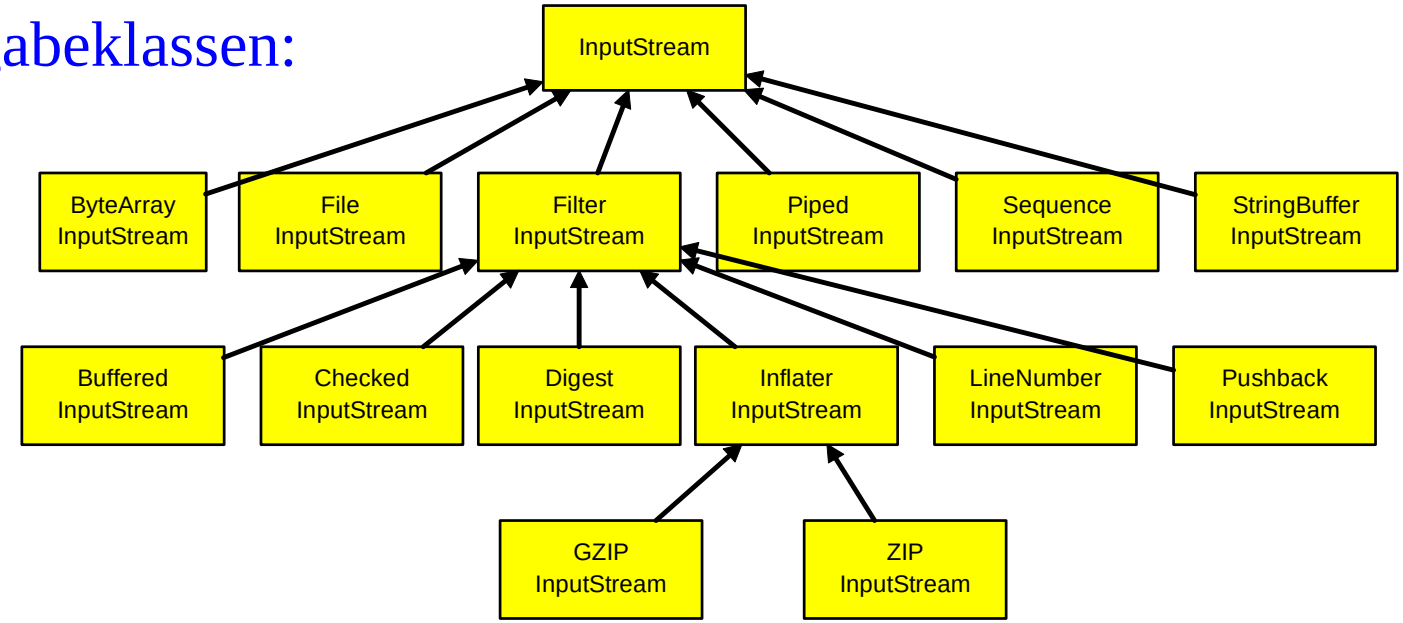
4) Also weiterreichen von Exceptions

   Das Verstecken einer Exception liefert „einfacheren Code“ für den „Aufrufer“. Der Aufrufer sollte aber von der Exception erfahren.
   Man ist kein Hellseher beim Programmieren.

Datenverarbeitung in Java

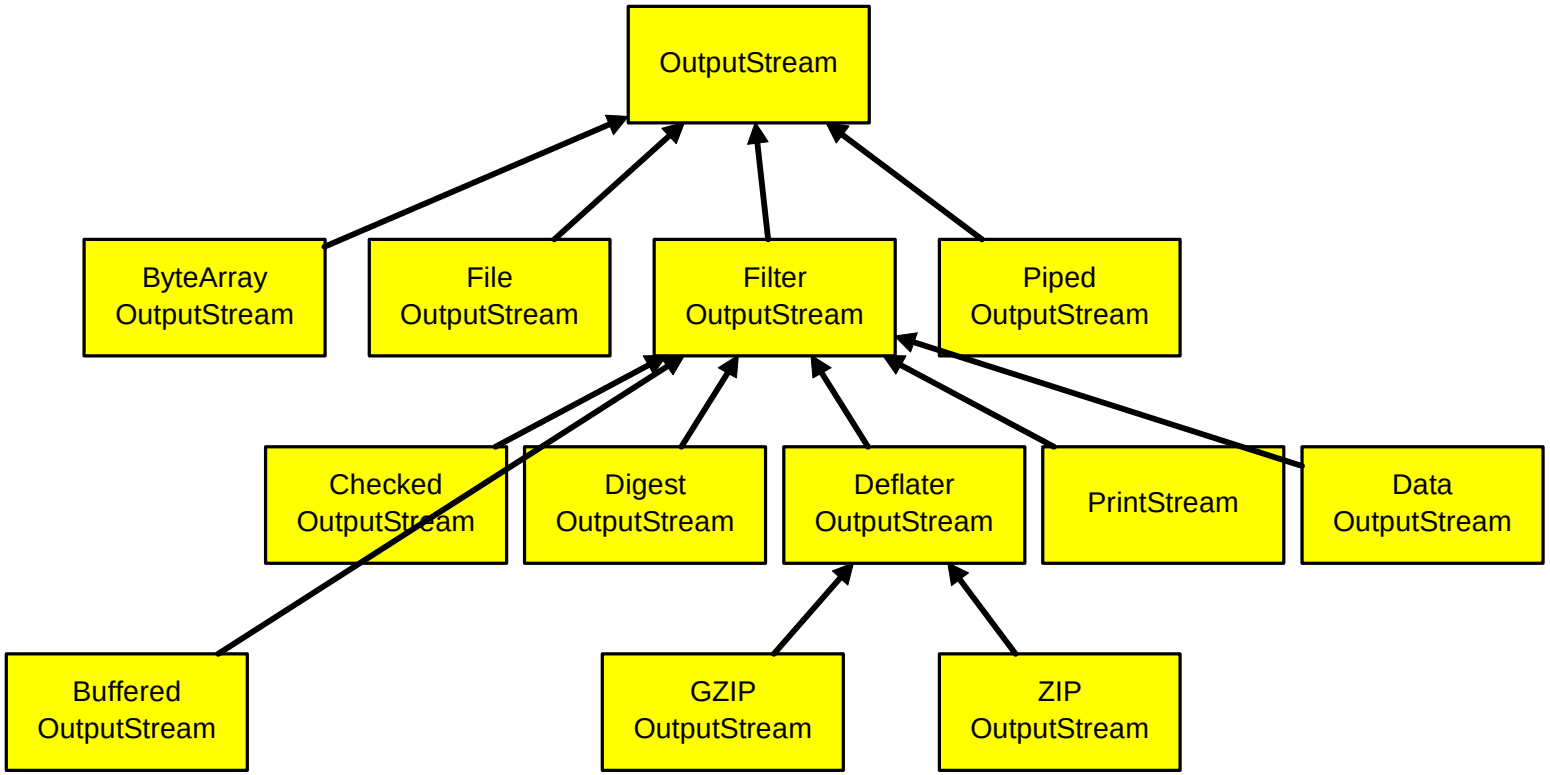
Es steht eine Vielzahl von Klassen/Modulen zur Verfügung (io:58):

Eingabeklassen:



- gepuffert
- Filter für Dateinamen
- für Zeichen, Zeichenketten, Objekte, Token
- mit Pipe-Verfahren
- Package: io, nio, nio2

Ausgabe in eine Datei



- gepuffert
- Filter für Dateinamen
- für Zeichen, Zeichenketten, Objekte, Token
- mit Pipe-Verfahren
- Package: io, nio, nio2

Stream-Unterscheidung

- ASCII:
  - PrintStream — print, println
  - DataOutputStream — writeBytes, writeUTF
  - LineNumberReader — readLine
  - DataInputStream — readLine
  - FileReader — read mit Array
- Binär
  - DataInputStream — readInt …
  - DataOutputStream — writeInt …
- ZIP
  - GZIPInputStream — read Array
  - GZIPOutputStream — write Array
- Serialize
  - ObjectInputStream — Object
  - ObjectOutputStream — Object
- XML
  - XMLInputFactory / XMLStreamReader — lesen
  - XMLEncoder — schreiben

Java: Ausgabe in eine ASCII-Datei

```
import java.io.*;
public class writeASCII01 {
  public static void main(String argv[]) {
    try {
        FileOutputStream Fout = new FileOutputStream("1.dat");
                // „Druck“ausgabe in eine ASCII-Datei
        PrintStream p = new PrintStream(Fout);
        p.print("Hallo");
        p.println("Hallo");
        p.println("Hallo Studenten");
        p.close();
      }
      catch (IOException e) {
         System.err.println("IOException: " + e);
      }
  }  // main
}
```

writeASCII02 arbeitet mit BufferedOutputStream

Stream-Unterscheidung

- ASCII:
  - PrintStream — print, println
  - DataOutputStream — writeBytes, writeUTF
  - LineNumberReader — readLine
  - DataInputStream — readLine
  - FileReader — read mit Array
- Binär
  - DataInputStream — readInt …
  - DataOutputStream — writeInt …
- ZIP
  - GZIPInputStream — read Array
  - GZIPOutputStream — write Array
- Serialize
  - ObjectInputStream — Object
  - ObjectOutputStream — Object
- XML
  - XMLInputFactory / XMLStreamReader — lesen
  - XMLEncoder — schreiben

Java: Lesen einer ASCII-Datei

```
public static void main(String argv[])  throws IOException {
   FileInputStream fin;
   InputStreamReader iin;
   LineNumberReader din;
   String sLine;
   try {
     fin = new FileInputStream("readASCII01.java");
     iin = new InputStreamReader(fin);
     din = new LineNumberReader(iin);  // oder BufferedReader br = new BufferedReader(ir);
     while  ( din.ready() ) {
       sLine = din.readLine();
       System.out.println(sLine);
     }
   }
    catch (FileNotFoundException e1) {
        System.err.println("Datei war nicht vorhanden!");
   }
   catch (IOException ee) {
         System.err.println("IOException: " + ee);
   }  }
```

Java: Lesen einer ASCII-Datei

```
private static void read(String sFilename){
   FileReader fr;
   File file = new File(sFilename);
   char ch;
   byte b;
   try {
     fr = new FileReader( sFilename ); //file );
     String gelesen; // Der String, der am Ende ausgegeben wird
         // char-Array als Puffer fuer das Lesen. Die
        // Laenge ergibt sich aus der Groesse der Datei
        char[] temp = new char[(int) file.length()];
        fr.read(temp); // Lesevorgang
        gelesen = new String(temp); // Umwandlung des char-Arrays in einen String
        System.out.println("Ergebnis:\n"+gelesen);

  } catch (FileNotFoundException e1) {
       System.err.println("Datei nicht gefunden: " + file);
   }
   catch (IOException ee) {
         System.err.println("IOException: " + ee);
   }
}
```

Java: Lesen einer ASCII-Datei

```
private static void read(String sFilename){
    FileInputStream fin;
    DataInputStream din;
    File file = new File(sFilename);
    try {
      fin = new FileInputStream( sFilename ); //file );
      din = new  DataInputStream(fin);
      String gelesen; // Der String, der am Ende ausgegeben wird
         // char-Array als Puffer fuer das Lesen. Die
         // Laenge ergibt sich aus der Groesse der Datei
         byte[] temp = new byte[(int) file.length()];

         din.read(temp); // Lesevorgang
          // Umwandlung des char-Arrays in einen String
         gelesen = new String(temp);
         System.out.println("Ergebnis:\n"+gelesen);
   } catch (FileNotFoundException e1) {
         System.err.println("Datei nicht gefunden: " + file);
    }
    catch (IOException ee) {
          System.err.println("IOException: " + ee);
    }
  }
```

Java: Einlesen aus einer HTML-Datei

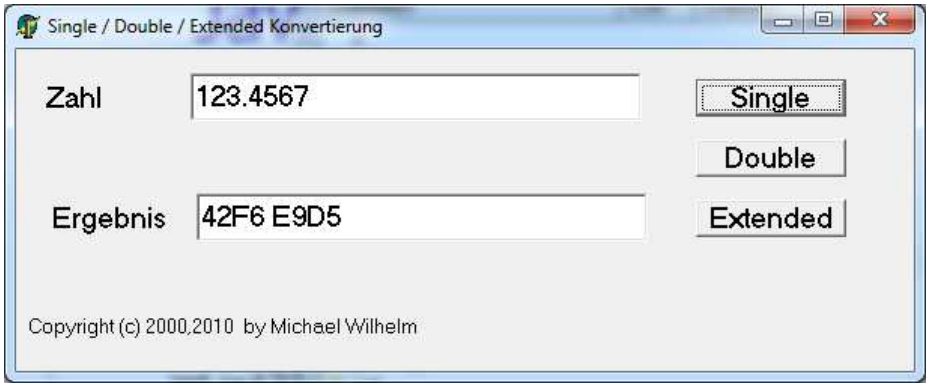
```
public static void main(String argv[])  throws IOException {
    URL u = new URL("http://www.hs-harz.de");
    InputStream in;
    DataInputStream din;
    String s;
    try {
    in = u.openStream();  // URL als Stream öffen
    din = new DataInputStream(in);
    while (  (s=din.readLine()) != null) {
      System.out.println(s);
    }
     catch (FileNotFoundException e1) {
         System.err.println("Datei war nicht vorhanden!");
    }
    catch (IOException ee) {
          System.err.println("IOException: " + ee);
    }
  }
```

Stream-Unterscheidung

- ASCII:
  - PrintStream — print, println
  - DataOutputStream — writeBytes, writeUTF
  - LineNumberReader — readLine
  - DataInputStream — readLine
  - FileReader — read mit Array
- Binär
  - DataInputStream — readInt …
  - DataOutputStream — writeInt …
- ZIP
  - GZIPInputStream — read Array
  - GZIPOutputStream — write Array
- Serialize
  - ObjectInputStream — Object
  - ObjectOutputStream — Object
- XML
  - XMLInputFactory / XMLStreamReader — lesen
  - XMLEncoder — schreiben

Java: Ausgabe in eine Datei (int , float, double)

```
private void write1() {
   try {
       FileOutputStream fout = new FileOutputStream("readwrite1.bin");  // byteweise
         // Umbau zur Ausgabe nun mittels primitiver Datentypen
       DataOutputStream dout = new DataOutputStream(fout);
       int i=1234;
       dout.writeInt(i);  // 4 Bytes
       float f = 123.4567f;
       dout.writeFloat(f);
       double d = 123.4567;
       dout.writeDouble(d);
       System.out.println("int i: "+i);
       System.out.println("double d: "+d);
       dout.close();
    }
    catch (IOException e) {
        System.err.println("IOException: " + e);
    }
  }  // write1
```



Java: Ausgabe in eine Datei (String)

```
private void write2() {
    System.out.println("\n\nwrite2");
   try {
        FileOutputStream fout = new FileOutputStream("readwrite1.bin");  // byteweise
                                  // Ausgabe nun mittels primitiver Datentypen
        DataOutputStream dout = new DataOutputStream(fout);

        String s = "abcdef";
        dout.writeUTF(s);

        System.out.println("String s: "+s);
        dout.close();
    }
     catch (IOException e) {
        System.err.println("IOException: " + e);
     }
  }  // write2
```

Java: Einlesen aus einer Datei (int , float, double)

```
private void read1() {
    System.out.println("\n\nread1");
   try {
        FileInputStream fin = new FileInputStream("readwrite1.bin");  // byteweise
                                  // Ausgabe nun mittels primitiver Datentypen
        DataInputStream din = new DataInputStream(fin);
        int i;
        i = din.readInt();  // 4 Bytes
        float f = din.readFloat();
        double d = din.readDouble();
        System.out.println("int i: "+i);
        System.out.println("float f: "+f);
        System.out.println("double d: "+d);
        din.close();
      }
      catch (IOException e) {
          System.err.println("IOException: " + e);
      }
 }  // read1
```

Java: Einlesen aus einer Datei (int , float, double)

```
private void read1() {
    System.out.println("\n\nread1");
   try {
        FileInputStream fin = new FileInputStream("readwrite1.bin");  // byteweise
                                    // Ausgabe nun mittels primitiver Datentypen
        DataInputStream din = new DataInputStream(fin);
        int i;
        i = din.readInt();  // 4 Bytes
        float f = din.readFloat();
        double d = din.readDouble();
        System.out.println("int i: "+i);
        System.out.println("float f: "+f);
        System.out.println("double d: "+d);
        din.close();
      }
      catch (IOException e) {
          System.err.println("IOException: " + e);
      }
 }  // read1
```

Java: Einlesen aus einer Datei (String)

```
private void read2() {
    System.out.println("\n\nread2");
   try {
        FileInputStream fin = new FileInputStream("readwrite1.bin");  // byteweise
                                  // Ausgabe nun mittels primitiver Datentypen
        DataInputStream din = new DataInputStream(fin);

        String s="";
        s = din.readUTF();
        System.out.println("String s: "+s);

        din.close();  // es reicht din
       }
       catch (IOException e) {
          System.err.println("IOException: " + e);
       }
  }  // read2
```

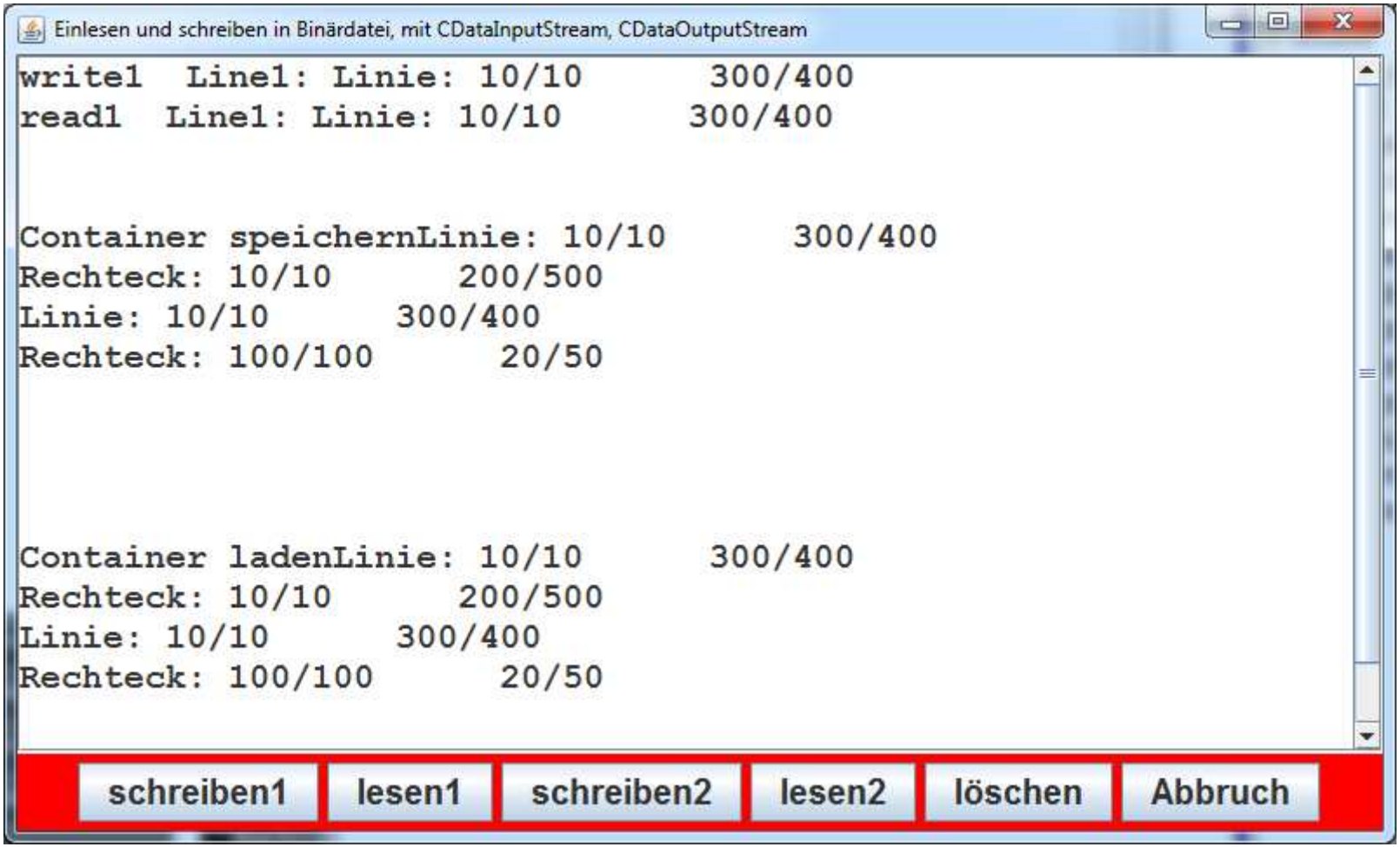
Java: Ausgabe in eine Datei (char)

```
private void write1() {
    try {
        System.out.println("write1");
        FileOutputStream fout = new FileOutputStream("readwrite1.bin");  // byteweise
                                    // Ausgabe nun mittels primitiver Datentypen
        DataOutputStream dout = new DataOutputStream(fout);
        int i = 12345;
        long ii=1234;
        dout.writeInt(i);  // 4 Bytes
        dout.writeLong(ii);  // 8 Bytes
        dout.writeChar('a');
        dout.writeChar('b');
        dout.writeChar('c');
        // dout.writeChars(s); // ??
        String s = "abcdef";
        dout.writeUTF(s);
        dout.writeDouble(123.4567);
        dout.close();
     }
     catch (IOException e) {
         System.err.println("IOException: " + e);
     }
} // write1
```

Java: Einlesen aus einer Datei (char)

```
…
i = din.readInt();  // 4 Bytes
System.out.println("i: "+i);
ii=din.readLong();  // 8 Bytes
System.out.println("long: "+ii);
ch=din.readChar();   // a
System.out.println("char ch: "+ch);
ch=din.readChar();
System.out.println("char ch: "+ch);
ch=din.readChar();
System.out.println("char ch: "+ch);
String s;
//din.readChars();   / / gibt es nicht
s = din.readUTF();
System.out.println("String s: "+s);
d = din.readDouble();
System.out.println("double d: "+d);
din.close();
…
```

Objekte speichern und laden:Readwrite_int_double_3



Readwrite_int_double_3

```
private void bn_write1() {
   try {
      FileOutputStream fout = new FileOutputStream("grafik.bin");
      DataOutputStream dout = new DataOutputStream(fout);
      Line line1 = new Line(10,10,300,400);
      line1.save(dout);
      dout.close();
      editor.append("write1  Line1: "+line1+"\n");
   }
    catch (IOException ee) {
       System.err.println("IOException: " + ee);
    }
 }  // write1
 private void bn_read1() {
   try {
      FileInputStream fin = new FileInputStream("grafik.bin");
      DataInputStream din = new DataInputStream(fin);
      Line line1 = new Line(0,0,0,0);
      line1.load(din,true);
      din.close();
      editor.append("read1  Line1: "+line1+"\n");
   }
    catch (IOException ee) {
       System.err.println("IOException: " + ee);
    }
 }  // read1
```

Readwrite_int_double_3

```
class Line extends Grafik {
  private final int version=1;
   private int x1=0;
   private int y1=0;
   private int x2=0;
   private int y2=0;

public void save(DataOutputStream dout ) {
    try {
     dout.writeInt(Grafik.LINE);
     dout.writeInt(version);
     dout.writeInt(x1);
     dout.writeInt(y1);
     dout.writeInt(x2);
     dout.writeInt(y2);
    }
    catch (IOException ee) {
        System.err.println("IOException: " + ee);
     }
   }
```

Readwrite_int_double_3

```
class Line extends Grafik {
  private final int version=1;
   private int x1=0;
   private int y1=0;
   private int x2=0;
   private int y2=0;

   public void load(DataInputStream din, boolean mitTyp ) {
    try {
       int typ;
       if (mitTyp)  typ = din.readInt();  // besser mit unread: PushbackInputStream
                                          // Readwrite_int_double_4
       int version=din.readInt();
       x1=din.readInt();
       y1=din.readInt();
       x2=din.readInt();
       y2=din.readInt();
    }
    catch (IOException ee) {
         System.err.println("IOException: " + ee);
     }
   }
```

Java: Methoden der Klasse DataOutputStream

DataOutputStream:

writeBoolean

writeByte (Schreiben einer 8-Bit Vorzeichenzahl)

writeChar (Schreiben einer 16-Bit vorzeichenlosen Zahl)

writeDouble (Schreiben einer Double-Zahl)

writeFloat (Schreiben einer Single-Zahl)

writeInt (Schreiben einer 32-Bit Vorzeichenzahl)

writeLong (Schreiben einer 64-Bit Vorzeichenzahl)

writeShort (Schreiben einer 16-Bit Vorzeichenzahl)

writeUTF (Schreiben eines Strings)

writeBytes (Schreiben eines Strings)

writeChars

Java: Methoden der Klasse DataInputStream

DataInputStream:

readBoolean

readByte (Einlesen 8-Bit Vorzeichenzahl)

readChar (Einlesen 16-Bit Vorzeichenzahl)

readDouble (Einlesen einer Double-Zahl)

readFloat (Einlesen Single-Zahl)

readInt (Einlesen 32-Bit Vorzeichenzahl)

readLong (Einlesen 64-Bit Vorzeichenzahl)

readShort (Einlesen 16-Bit Vorzeichenzahl)

readUTF (Einlesen eines Strings)

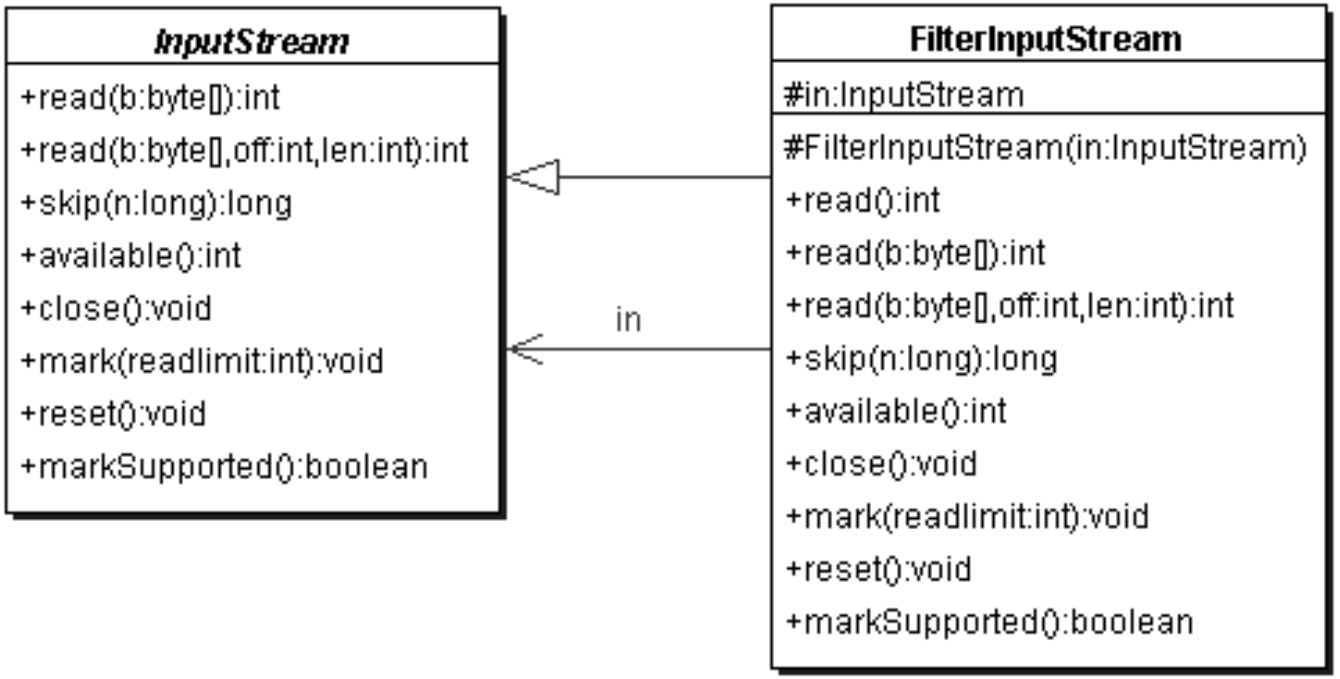
readByte (byte Array)

Int-Datentypen in Java

byte
from -128 to 127
short
from -32768 to 32767
int
from -2147483648 to 2147483647
long
from -9223372036854775808 to 9223372036854775807
char
from '\u0000' to '\uffff‘, =>, from 0 to 65535

Java Beispiele

IOtest.java	Laufzeit ASCII vs. Binär
ReadASCII01.java bis ReadASCII04.java	
WriteASCII01.java bis WriteASCII02.java	Schreiben ASCII
readwriteStr1.java bis readwriteStr3.java	Lesen und Schreiben ASCII
Readwrite_int_double_1.java bis Readwrite_int_double_4.java	
Read_gz.java	Lesen gz-Format
Write_gz.java	Schreiben gz-Format
Read_Write_zip.java	Lesen und Schreiben zip-Format (kein echtes zip-Format)
Xml1.java bis Xml3.java	Lesen einer XL-Datei



- Die UML-Diagramme zeigen, dass jeder Filter zum einen selbst ein Stream ist und zum anderen einen Stream verwaltet.
- Damit nimmt er Daten entgegen und leitet sie gleich weiter.
- Das ist ein bekanntes Design-Pattern und nennt sich ***Dekorator***.