Grundlagen der Informatik 2:

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

Gliederung

1. Einführung
2. WWW / HTML
   Überblick über WWW
   Einführung in HTML
3. UNIX
4. Unix Kommandos
5. Unix Shellprogrammierung
6. Betriebssysteme (Prozess, Threads, Speicher, Dateiverwaltung)
7. **Java Ergänzung (Exception, Streams)**

Benutzeroberflächen: Die Java Klassenbibliothek

- Regelmäßige Erweiterung und Veränderung mit den Versionen des JDK
- Teile der Bibliothek werden in der Dokumentation zum JDK als packages bezeichnet
- Objektorientierte Vorteile der Bibliothek
  - Verfeinerung der allgemeine Klassen durch Vererbung der Methoden innerhalb eines package an spezielle Klassen
  - Bereitstellung generischer Datenstrukturen (Templates)

Die Java Klassenbibliothek

- Beispiele
  - java.lang: Sprachunterstüzung (Threads, String)
  - java.util: Hilfeklassen, allgemeine Datenstrukturen (Stack, Vector)
  - java.io: Ein/Ausgabe-Klassen (Dateien, Streams)
  - java.net: Sockets, URL
  - java.applets, java.awt: Java Windows Toolkit
- Erweiterungen (Auswahl)
  - java.swing: AWT mit anpassbaren „Look & Feel“)
  - speech: Srachein- und Sprachausgabe
  - JDBC: Datenbankanbindung
  - Beans: wiederverwendbare Komponenten
  - JMF: Java Media Foundation

Java: Exception

```
import java.io.*;

public class exception1 {

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

Java: Exception

```
D:\HS-Harz\Inf2>java exception1
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at exception1.main(exception1.java:8)

D:\HS-Harz\Inf2>
```

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 (§15.16.2) 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 (§10.10, §15.25.1).
ClassCastException:	An attempt has been made to cast (§5.4, §15.15) 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 (§20.1.6, §20.1.7, §20.1.8) or notify (§20.1.9, §20.1.10) 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-Arten

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

Package java:

java.io.IOException:	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.

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;
    }
    catch (ArithmeticException f) {
        System.err.println(" ArithmeticException : " + f);
    }
  }  // main
}
```

exception2

Java: Exception (externe Abfrage)

```
import java.io.*;

public class exception3 {

  public static Double loadDouble(String sFileName)
     {
    return new Double(1);  // verkürzt
  }

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

Java: Exception (externe Abfrage)

```
import java.io.*;

public class exception4 {

  public static Double loadDouble(String sFileName) throws IOException {
    return new Double(1);
  }

  public static void main(String argv[]) {
    Double d;
    d = loadDouble2("c:\\1.dat");
  }  // main
}
```

Compiler:

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

Java: Exception

```
import java.io.*;

public class exception4 {

  public static Double loadDouble2(String sFileName) throws IOException {
    return new Double(1);
  }

  public static void main(String argv[]) {
 try {
      d = loadDouble2("c:\\1.dat");
    }
    catch (IOException f) {
        System.err.println("IOException: " + f);
    }
  }  // main
} // class
```

Java: Throw Exception

```
public static Double loadDouble(String sFileName) throws IOException {
  int i=1;
  if (i==1) throw new EOFException("Modul loadDouble (Header)");
  return new Double(1);
}  // 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\inf2>java exception5
main: java.io.EOFException: Modul loadDouble (Header)

D:\HS-Harz\inf2>
```

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 {

        public HeaderFalseVersion() {
        }

        public HeaderFalseVersion(String error) {
                super(error);
        }

}  // HeaderFalseVersion
```

Java: finally Klausel

Beim Auslösen einer Exception wird sofort das jeweilige Modul verlassen.

Problem:

Lokale Resourcen werden nicht sauber entfernt.

Abhilfe:

finally Klausel

Java: finally Klausel

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

}
```

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.

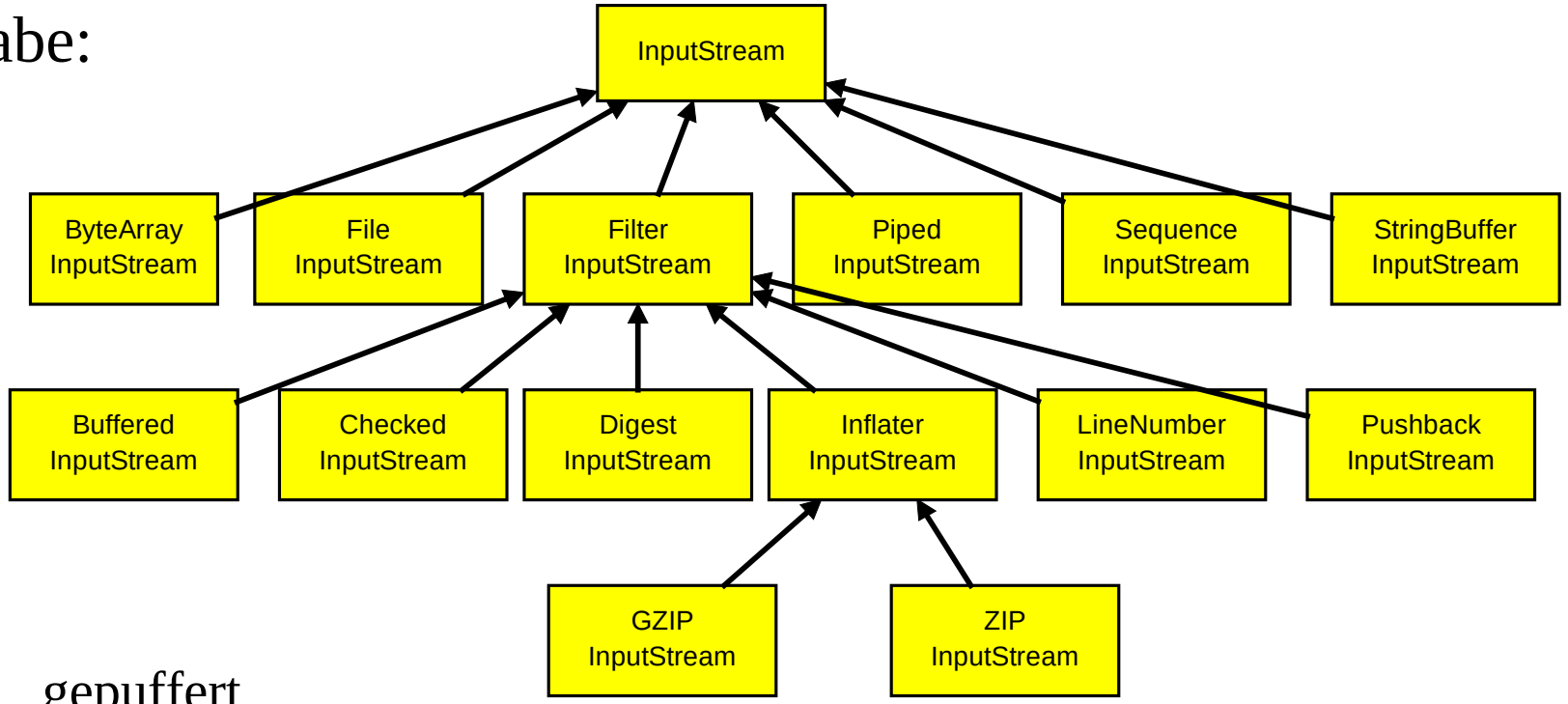
4) Weiterreichen von Exceptions

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

Datenverarbeitung in Java

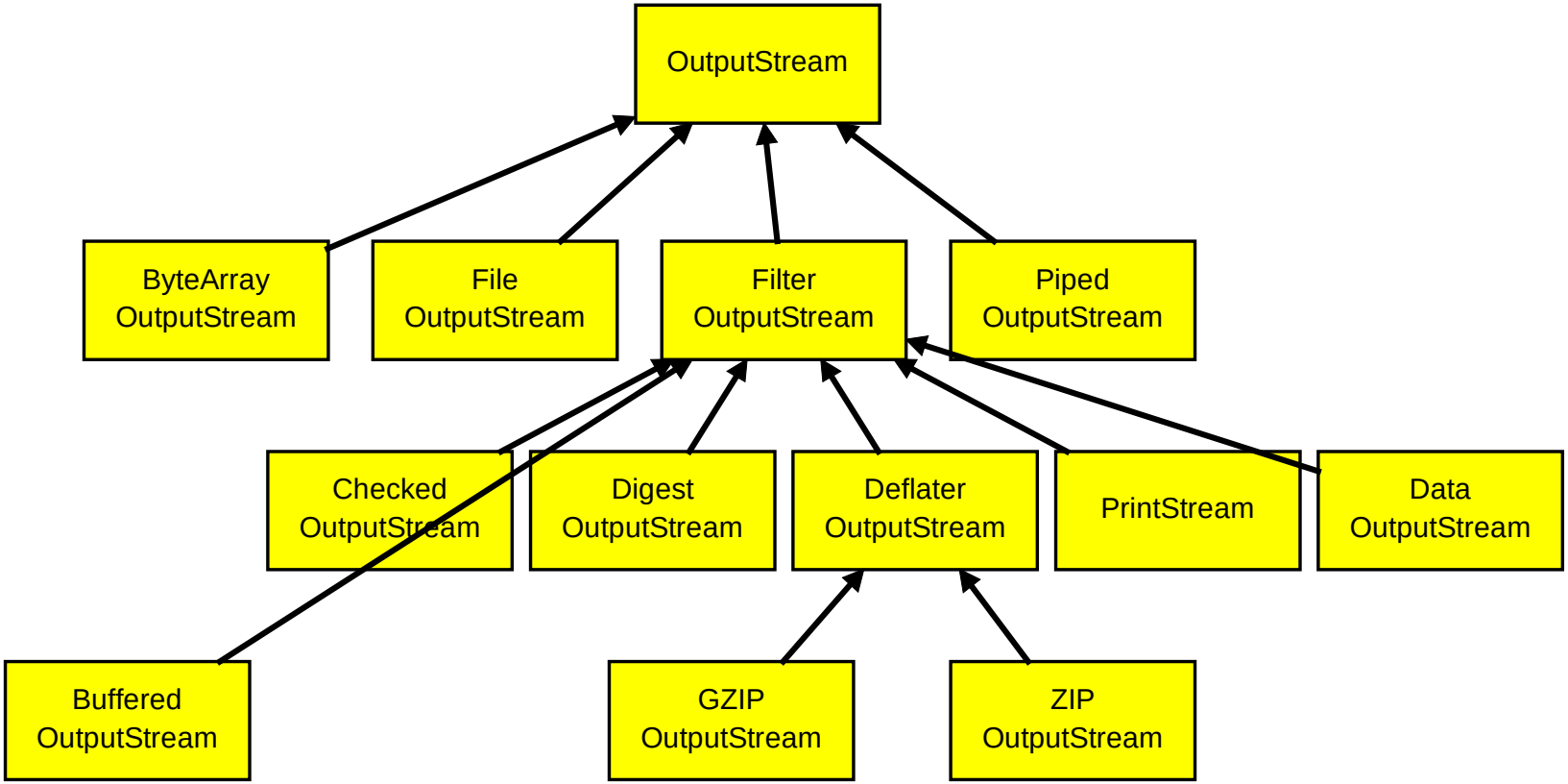
Es steht eine Vielzahl von Klassen/Modulen zur Verfügung (58):

Eingabe:



- gepuffert
- Filter für Dateinamen
- für Zeichen, Zeichenketten, Objekte, Token
- mit Pipe-Verfahren

Ausgabe in eine Datei



Java: Ausgabe in eine Datei

Basis aller Ausgaben ist die Klasse OutputStream

Weitere Spezifikation durch z.B:

- FileOutputStream

Weitere Spezifikation durch:

- BufferedOutputStream
- PrintStream
- DataOutStream
- ZIPOutStream
- CheckedOutStream

Java: Ausgabe in eine Datei (PrintStream)

```
public class write1 {

  public static void main(String argv[]) {
            // Stream Definition

          FileOutputStream Fout = new FileOutputStream("1.dat");
            // Druckausgabe
        PrintStream p = new PrintStream(Fout);
        p.println("Hallo");
        p.println("ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789");
        p.println("123.34");
        p.print("abc");
        p.print("123");
        p.close();

  }  // main
}
```

Java: Ausgabe in eine Datei (PrintStream)

Ergebnis:

In der Datei „1.dat“ werden folgende Zeilen geschrieben:

Hallo
ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789
123.34
abc123

Ausgabe in eine Datei (DataOutputStream)

Die Klasse „DataOutputStream“erlaubt die binäre Speicherung von Variablen aus Java in eine Datei.

Unterstützte Datentypen:	**Methode:**
Boolean	writeBoolean
Byte	writeByte
Char	writeChar
Double	writeDouble
Float	writeFloat
Int	writeInt
Long	writeLong
Short	writeShort

Java: Ausgabe in eine Datei (DataOutputStream)

```
public static void main(String argv[]) {
   double d;
   try {
      FileOutputStream Fout = new FileOutputStream("1.dbl");
                        // Defintion der Ausgabe
      DataOutputStream DataOut = new DataOutputStream(Fout);
      d = 1234.0;
        DataOut.writeDouble(d);
      d = 1234.11111;
        DataOut.writeDouble(d);
        DataOut.writeDouble(-1234.0);
      DataOut.close();
     }
     catch (IOException ee) {
        System.err.println("IOException: " + ee);
     }
 }  // main
```

Java: Ausgabe in eine Datei (DataOutputStream)

```
public static void main(String argv[]) {
    int i;
    try {
        FileOutputStream Fout = new FileOutputStream("1.int");
                        // Druckausgabe
        DataOutputStream DataOut = new DataOutputStream(Fout);
        i = 1234;
          DataOut.writeInt(i);
        i = 4321;
          DataOut.writeInt(i);
        DataOut.writeInt(-1234);
        DataOut.close();
      }
      catch (IOException ee) {
          System.err.println("IOException: " + ee);
      }
} // main
```

Einlesen aus einer Datei (DataInputStream)

Die Klasse „DataInputStream“erlaubt das binäre Lesen von Variablen auseiner Datei.

Unterstützte Datentypen:	**Methode:**
Boolean	readBoolean
Byte	readByte
Char	readChar
Double	readDouble
Float	readFloat
Int	readInt
Long	readLong
Short	readShort

Lesen aus einer ASCII-Datei (zeilenweise)

```
import java.io.*;

public class read2 {
  public static void main(String argv[])  throws IOException {
    FileInputStream fin;
    DataInputStream din;
    String s;

    fin = new FileInputStream("1.dat");
    din = new DataInputStream(fin);
    while (  (s=din.readLine()) != null) {
        System.out.println(s);
    }
  }  // main
}
```

Einlesen aus einer Datei (Byte-Weise)

```
public static void main(String argv[])  throws IOException {
   FileInputStream fin;                    DataInputStream din;
   char ch;                                byte b;

   try {
     fin = new FileInputStream("1.dat");
     din = new DataInputStream(fin);
     while  (true) {
        b =din.readByte();
        ch = (char) (b);
        System.out.println(ch);
     }
   }
     catch (IOException ee) {
          System.err.println("IOException: " + ee);
   }
}
```

Einlesen aus einer Datei (binär)

```
private static void read(String sFilename){
  int i;
  try {
      FileInputStream Fin = new FileInputStream(sFilename);
                          // Druckausgabe
      DataInputStream DataIn = new DataInputStream(Fin);
      i = DataIn.readInt();
       System.out.println(i);
      i = DataIn.readInt();
       System.out.println(i);
      i = DataIn.readInt();
       System.out.println(i);
       DataIn.close();
    }
    catch (IOException ee) {
       System.err.println("IOException: " + ee);
    }
}
```

Einlesen aus einer Datei (binär)

Aufgerufen wird die Methode read mit einem Parameter.

```
        // Einlesen von int Zahlen
        // Einlesen der Zahlen
public static void main(String argv[]) {
  read("1.int");
}  // main
```

Beispieldateien:

read1.java	Zeichenweise Einlesen aus einer ASCII-Datei
read2.java	Zeilenweise Einlesen aus einer ASCII-Datei
read3.java	Zeilenweise Einlesen aus einer ASCII-Datei, mit BufferedReader
read4.java	Einlesen aus einer binären Datei, in der int-Werte gespeichert wurden
read5.java	Einlesen aus einer binären Datei, in der double-Werte gespeichert wurden
write1.java	Ausgabe in eine Datei, Printstream, ohne Exception
write2.java	Ausgabe in eine Datei, Printstream, mit Exception
write3.java	Ausgabe in eine Datei, Printstream, mit Exception, mit BufferedOutputStream
write4.java	Ausgabe in eine Datei, Printstream, mit Exception, mit BufferedOutputStream, Ausgabe mittels einer int Variable
write5.java	Ausgabe double Werten, binäre Ausgabe
write6.java	Ausgabe int Werten, binäre Ausgabe
write7.java	Ausgabe int+double Werten, binäre Ausgabe
write8.java	Ausgabe Char Werte, binäre Ausgabe
readwrite1.java	Ein- und Ausgabe von int-Werten, binär
readwrite2.java	Ein- und Ausgabe von double+int-Werten, binär