Leibniz Universität IT Services

Leibniz Universität Hannover

Java - Eingabe und Ausgabe

Streams (Datenströme)

- Transport von Daten.
- Input-Streams (Eingabe) lesen Daten von einer Quelle.
- Output-Streams schreiben Daten in eine Senke.

Konsolen

- Kommandozeile. Terminal.
- Pro „Zeile“ werden textbasierte Befehle zur Steuerung von Softwareprogrammen oder zur Navigation in Ordnern eingegeben.
- Z. B. Das Betriebssystem Windows hat die Microsoft Eingabeaufforderung PowerShell als Konsole.
- Die IDE „NetBeans“ nutzt das Fenster output als „Konsole“.

Standard-Streams in Java

- `System.out`. Ausgabe auf die Konsole.
- `System.in`. Lesen von der Konsole. Mit Hilfe der Tastatur werden Zeichen auf der Konsole eingegeben.
- `System.err`. Ausgabe von Fehlermeldungen auf die Konsole. NetBeans kennzeichnet diese mit einer roten Schriftfarbe.

Ausgabe in die Konsole

```
String satz01 = "Mr Mousebender: Tell me, do you have
                 any cheese at all?";
String satz02 = "Henry Wenslydale: Yes.";

System.out.println(satz01);
System.out.println(satz02);

System.out.print(satz01 + '\n');
System.out.print(satz02 + '\n');
```

Erläuterung

- Beiden Methoden wird in den runden Klammern ein String als Argument übergeben.
- Die Methode `.print()` gibt nur das Argument aus. Falls mehrere Ausgaben nach einander folgen, werden diese nicht durch einen Zeilenumbruch getrennt.
- Die Methode `.println()` gibt diesen String aus und setzt automatisch einen Zeilenumbruch.

Formatierung der Ausgabe

```
System.out.printf("Temperatur im %-20s %5.2f %n",
                  "Januar", 7.6);
```

- Die Methode `.printf()` formatiert die übergebenen Variablen entsprechend der Angaben.
- Als erster Parameter wird der Methode ein Formatstring übergeben. Dieser String besteht aus nicht veränderbaren Zeichen, Platzhalter für die dynamischen Bestandteile und Formatierungszeichen zu den Platzhaltern.
- Die anderen Parameter enthalten die Werte für die dynamischen Bestandteile des Formatierungsstrings.

Formatierungszeichen

```
System.out.printf("Temperatur im %-20s %5.2f %n",
                  "Januar", 7.6);
```

- Beginn mit dem Prozentzeichen.
- Der Buchstabe kennzeichnet den Datentyp. `%s` ist ein Platzhalter für ein String. `%f` stellt eine Gleitkommazahl dar.
- In Abhängigkeit des Typs stehen weitere Formatierungsmöglichkeiten zur Verfügung.

Beispiele für Platzhalter

Platzhalter	Datentyp
%s	String
%d	Ganzzahl
%f	Gleitkommazahl
%n	New Line

- Siehe https://docs.oracle.com/javase/tutorial/java/data/numberformat.html

Formatierungszeichen

%	schalter	breite	.	genauigkeit	typ

- Schalter. Wird der Wert rechts- oder linksbündig ausgerichtet?
- Breite. Mindestanzahl der auszugebenden Zeichen.
- Genauigkeit. Bei Gleitkommazahlen wird zum Beispiel die Anzahl der Nachkommastellen angegeben.
- Typ. Ein Buchstabe, der eine bestimmte Gruppe von Werten symbolisiert.

Beispiele für Strings

Das	Erläuterung
%s	Unformatierter String
%20s	Rechtsbündige Ausgabe. Die Angabe 20 legt die Mindestanzahl der auszugebenden Zeichen fest. Falls der String kleiner ist, wird mit Leerzeichen aufgefüllt.
%-20s	Das Minuszeichen symbolisiert eine linksbündige Ausgabe.
%%	Prozentzeichen
%n	Neue Zeile

Beispiele für Ganzzahlen

	Erläuterung
%b %B	Boolean
%d	Ganzzahl.
%4d	Mindestens 4 Stellen werden genutzt. Falls die Zahl weniger Stellen hat, wird diese mit Leerzeichen aufgefüllt.
%04d	Mindestens 4 Stellen werden genutzt. Falls die Zahl weniger Stellen hat, wird diese mit Nullen aufgefüllt.

Beispiele für Gleitkommazahlen

	Erläuterung
%f	Gleitkommazahl
%.2f	Die Gleitkommazahl wird in diesem Beispiel auf 2 Stellen nach dem Komma gerundet.
%5.2f	Rechts vom Punkt wird die Gesamtzahl der auszugebenden Zeichen angegeben. Links vom Punkt wird die Anzahl der Nachkommastellen angegeben. In diesem Beispiel würde eine Zahl mit zwei Stellen vor und nach dem Komma angezeigt. Falls die Zahl mehr Stellen benötigt, werden diese angezeigt.

Nutzung der Klasse OutputStreamWriter

```
import java.io.OutputStreamWriter;

public class output {

    public static void main(String[] args) {
        OutputStreamWriter output = null;

        try{
            output =
                new OutputStreamWriter(System.out);

            output.write("Zeichen: \n");
            output.flush();
```

Import der Klasse

```
import java.io.OutputStreamWriter;
```

- Die Klasse `OutputStreamWriter` ist in dem Paket `java.io` definiert.
- Der Punktoperator verbindet die Ordner (Pakete) mit den darin enthaltenen Dateien (Klassen). Ordner können wiederum Unterordner enthalten.
- In diesem Beispiel wird die Datei `OutputStreamWriter` in dem Ordner `io` genutzt. Der Ordner `io` liegt in dem Wurzelordner `java`.

Variable der Klasse Scanner

```
OutputStreamWriter output = null;
```

- Objektvariable vom Datentyp `OutputStreamWriter`.
- Variablen, die ein Verweis auf eine Instanz von der Klasse `OutputStreamWriter` speichern können.
- Momentan verweist die Variable auf kein Objekt. Die Objektvariable ist mit dem Wert `null` initialisiert.

Erzeugung einer Instanz der Klasse ...

```
output = new OutputStreamWriter(System.out);
```

- Das Schlüsselwort `new OutputStreamWriter` erzeugt ein neues Objekt von der Klasse `OutputStreamWriter`.
- Die Instanz wird mit Hilfe des, in den runden Klammern, übergebenen Parameter konstruiert. Die Parameterliste wird durch die runden Klammern begrenzt.
- In diesem Beispiel wird die Standardausgabe (`System.out`) als Vorlage genutzt.

Schreiben von Zeichen

```
output.write("Zeichen: \n");
```

- Die Methode .write() schreibt einen String in den Ausgabestrom.
- Als Parameter wird der Methode der zu schreibende Text übergeben.
- Die Methode und die Instanzvariable werden mit dem Punkt verbunden.

Schreiben von gepufferten Zeichen

```
output.flush();
```

- Die Methode `.flush()` schreibt die gepufferte Daten in den Stream.
- Die Daten werden in dem damit verbundenen Stream sichtbar.

Welcher Zeichensatz wird verwendet?

```
output.getEncoding();
```

- Die Methode `.getEncoding()` gibt den verwendeten Zeichensatz des Ausgabestreams zurück.
- Die Standardausgabe der IDE NetBeans nutzt standardmäßig UTF-8.

Einlesen von der Konsole

```
import java.io.IOException;
public class Java13_IOEingabe {
    public static void main(String[] args) {
        String teilnehmerA = "";
        byte eingabe[] = new byte[256];
        int anzahlBytesGelesen = -1;

        try{
            System.out.print("Eingabe: ");
            anzahlBytesGelesen = System.in.read(eingabe);
        }
        catch (IOException ex){
            ex.printStackTrace();
```

Einlesen von der Konsole

```
byte eingabe[] = new byte[256];
int anzahlBytesGelesen = -1;

anzahlBytesGelesen = System.in.read(eingabe);
```

- Mit Hilfe der Methode `.read()` kann von der Konsole byteweise eingelesen werden.
- Jedes ASCII-Zeichen kann in einem Byte (8 Bits) dargestellt werden. Das Zeichen A wird mit der Ganzzahl 65 kodiert. Das Byte 1000 0001 speichert das Zeichen A.

Argument der Methode

```
byte eingabe[] = new byte[256];
int anzahlBytesGelesen = -1;

anzahlBytesGelesen = System.in.read(eingabe);
```

- In diesem Beispiel wird der Methode ein Array vom Datentyp byte übergeben. In diesem Array werden die Zeichen eingelesen. Die maximale Anzahl der einzulesenden Zeichen wird durch die Länge des Arrays vorgegeben.
- Falls die Klammern leer sind, wird ein Zeichen eingelesen.

Rückgabewert der Methode

```
byte eingabe[] = new byte[256];
int anzahlBytesGelesen = -1;

anzahlBytesGelesen = System.in.read(eingabe);
```

- Die Eingabe muss mit der Eingabetaste abgeschlossen werden.
- Die Methode gibt die Anzahl der tatsächlich gelesenen Bytes zurück.

Bytes → String

```
teilnehmerA =
      new String(eingabe,0,anzahlBytesGelesen);
```

- Mit Hilfe der Anweisung new String() wird eine Instanz von einem String erzeugt.
- Als erstes Argument wird der um zu wandelnde Byte-Array angegeben.
- Als zweites Argument wird ein Offset übergeben? In diesem Beispiel soll das Array ab dem ersten Element umgewandelt weden.
- Das dritte Argument gibt die maximale Anzahl der umzuwandelnden Elemente an.

Abfangen eines IO-Fehlers

```
catch (java.io.IOException ex){

```

- Der Fehler `java.IO.IOException` muss beim Einlesen von der Konsole abgefangen werden. Andernfalls wird ein Syntaxfehler angezeigt.
- Die Klasse `IOException` ist in dem Paket `java.IO` definiert.
- Durch die Anweisung `import java.io.IOException wird` die Angabe des Paketes in der catch-Anweisung nicht benötigt.

Import von ...

```
import java.io.IOException;
import java.io.*;
```

- Mit Hilfe des Schlüsselwortes `import` wird ein Paket oder eine Klasse und deren Attribute / Methoden in einem Java-Programm entsprechend ihrer Zugriffsspezifikation sichtbar gemacht.
- Die Angabe des Paketes zu einer Klasse wird an den Anfang eines Java-Programms ausgelagert.
- Falls viele Klassen aus einem Paket eingebunden werden sollen, kann ein Sternchen als Platzhalter für „alle“ gesetzt werden.

Hinweise

- Die `import`-Anweisung muss außerhalb einer Klasse, nach der `package`-Anweisung gesetzt werden.
- Das Paket `java.lang` wird automatisch in allen Java-Programmen eingebunden.

Zeichenorientierte Datenströme

- Verarbeitung von Zeichen, die von der Konsole gelesen werden.
- Ein- und Ausgabe von Daten, die in einem Texteditor angesehen und bearbeitet werden können.
- Arbeiten mit Text in einer bestimmten Zeichencodierung. In Java werden Daten im Unicode-Format verarbeitet.

Einlesen von zeichenorientierten Streams

- … mit der Klasse `java.io.InputStreamReader`. Eine bestimmte Anzahl von Zeichen wird eingelesen.
- … mit der Klasse `java.io.BufferedReader`. Es werden so viele Zeichen eingelesen, bis der Nutzer Return drückt.
- … mit der Klasse `java.util.Scanner` seit JDK 5.0.

Klasse „Scanner“

```
import java.util.Scanner;

public class Java13_SystemIn {

    public static void main(String[] args) {
        String teilnehmerA = "";
        Scanner input;

        input = new Scanner(System.in);
        System.out.println("Eingabe: ");
        teilnehmerA = input.nextLine();
        input.close();
```

Import der Klasse

```
import java.util.Scanner;
```

- Die Klasse `Scanner` ist in dem Paket `java.util` definiert.
- Der Punktoperator verbindet die Ordner (Pakete) mit den darin enthaltenen Dateien (Klassen). Ordner können wiederum Unterordner enthalten.

Variable der Klasse Scanner

```
Scanner input;
```

- Objektvariable vom Datentyp `Scanner`.
- Variablen, die ein Verweis auf eine Instanz von der Klasse `Scanner` speichern können.

Erzeugung einer Instanz der Klasse Scanner

```
Scanner input = new Scanner(System.in);
```

- Das Schlüsselwort `new Scanner` erzeugt ein neues Objekt von der Klasse `Scanner`.
- Die Instanz wird mit Hilfe des, in den runden Klammern, übergebenen Parameter konstruiert. Die Parameterliste wird durch die runden Klammern begrenzt.
- Die einzelnen Parameter in der Liste werden durch ein Komma getrennt.

Zwingend erforderlicher Parameter

```
Scanner input = new Scanner(System.in);
```

- Von welchen Datenstrom wird eingelesen?
- In diesem Beispiel wird die Standardeingabe (`System.in`) als Vorlage übergeben.
- Aber: Ein beliebiger Eingabestrom kann genutzt werden.

Schließen des Datenstroms

```
Scanner input = new Scanner(System.in);
input.close();
```

- Die Methode `.close()` schließt einen Datenstrom (in diesem Fall `System.in`).
- Die Objektvariable `input` verweist auf einen Datenstrom, der geschlossen wird.
- Falls die Objektvariable auf kein passenden Stream verweist, wird der Fehler `NoSuchElementException` ausgegeben.

Einlesen von Strings von der Konsole

```
strTemperatur[countMonat] = input.nextLine();
strTemperatur[countMonat] = input.next();
```

- Die Methode `.next()` liest Zeichen ein. Es werden alle Zeichen bis zum ersten Leerzeichen eingelesen.
- Die Methode `.nextLine()` liest einen String bis zum Zeile-Endezeichen „\n“ ein. Das Zeilen-Endezeichen wird nicht gelesen.

Einlesen von Zahlen

```
dblTemperatur[countMonat] = input.nextDouble();
intTemperatur[countMonat] = input.nextInt();
```

- Die Methode `.nextDouble()` liest eine Gleitkommazahl vom Typ `double` ein.
- Die Methode `.nextInt()` liest eine Ganzzahl vom Typ `int` ein.

… in Abhängigkeit des Eingabegebietsschema

```
Scanner input = new Scanner(System.in);
input.useLocale(Locale.GERMANY);
```

- Standardmäßig wird das, am Rechner eingestellte Eingabegebietsschema genutzt.
- Mit Hilfe der Methode `.useLocale()` kann das gewünschte Gebietsschema erstellt werden.
- Das Paket `java.util.Locale` hat viele Konstanten für die Auswahl eines Eingabegebietsschema. In diesem Beispiel wird das Gebietsschema „DE“ genutzt. Siehe https://www.tutorialspoint.com/java/util/java_util_locale.htm

… in Abhängigkeit des Zeichensatzes

```
String strUmlaute = "";
Scanner input = new Scanner(System.in, "iso-8859-1");

System.out.println("Umlaute: ");
strUmlaute = input.nextLine();

input.close();
```

- In diesem Kurs wird die Konsole output der IDE NetBeans genutzt.
- Die Konsole output stellt eingelesene Umlaute wie zum Beispiel ä, ü, ö und so weiter als Rechtecke dar.

Festlegung des Zeichensatzes

```
Scanner input = new Scanner(System.in, "iso-8859-1");
```

- Mit Hilfe von `new` wird ein neues Objekt vom Typ `Scanner` erzeugt.
- Für die Konstruktion eines neuen Scanners werden die angegebenen Werte in der Parameterliste benötigt. Die Parameterliste wird durch die runden Klammern begrenzt.
- Die einzelnen Parameter in der Liste werden durch ein Komma getrennt.

Parameter bei der Konstruktion

```
Scanner input = new Scanner(System.in, "iso-8859-1");
```

- Als erster Parameter wird eine Vorlage für die Konstruktion an den Konstrukteur übergeben. In diesem Beispiel wird die Standardeingabe (`System.in`) als Vorlage übergeben.
- Als zweiter Parameter wird der Zeichensatz angegeben, der von der Konsole oder Datei genutzt wird. In diesem Beispiel wird für das Schreiben in die Konsole der Zeichensatz ISO-8859-1 genutzt.

Ist ein Wert zum Einlesen vorhanden?

```
while (input.hasNext()) {
    if (input.hasNextDouble()) {
        messwerte[index] = input.nextDouble();
        index++;
    } else {
        input.next();
    }

}
```

Überprüfung

```
while (input.hasNext()) {

```

- Ist ein Element zum Einlesen im Datenstrom vorhanden?
- Die Methode `hasNext()` liefert `True`, wenn ein Element im Datenstrom zum Einlesen bereit steht.

… in Abhängigkeit des Datentyps

```
if (input.hasNextDouble()) {
if (input.hasNextInt()) {
```

- Kann das nächste einzulesende Element als Gleitkommazahl / Ganzzahl entsprechend des Eingabegebietsschemas interpretiert werden?
- Wenn das Element nicht entsprechend des Datentyps interpretiert werden kann, wird die Ausnahme `InputMismatchException` ausgegeben.

Klasse „BufferedReader“

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class Java13_IOBufferedReader {
    public static void main(String[] args) {
        String zeile = null;
        BufferedReader input;

        input = new BufferedReader(
                    new InputStreamReader(System.in));
        zeile = input.readLine();
        input.close();
```

Import der Klasse

```
import java.io.BufferedReader;
```

- Die Klasse `BufferedReader` ist in dem Paket `java.io` definiert.
- Der Punktoperator verbindet die Ordner (Pakete) mit den darin enthaltenen Dateien (Klassen). Ordner können wiederum Unterordner enthalten.

Variable der Klasse ...

```
BufferedReader input;
```

- Objektvariable vom Datentyp `BufferedReader`.
- Variablen, die ein Verweis auf eine Instanz von der Klasse `BufferedReader` speichern können.
- Objektvariable, die auf einen Zeichenbuffer verweist. Mit Hilfe des Buffers werden Zeichen zwischengespeichert.

Erzeugung einer Instanz der Klasse ...

```
input = new BufferedReader(datei);
```

- Das Schlüsselwort `new BufferedReader` erzeugt ein neues Objekt von der Klasse `BufferedReader`.
- Die Instanz wird mit Hilfe des, in den runden Klammern, übergebenen Parameter konstruiert. Als Argument wird dem Bauplan übergeben, wo eingelesen werden soll. Es können zum Beispiel Daten aus einer Datei gelesen werden.

Lesen von der Standardeingabe

```
inputReader = new InputStreamReader(System.in);
input = new BufferedReader(inputReader);
```

- `InputStreamReader` ist eine Brücke zwischen einem Byte-Strom und einem zeichenbasierten Strom.

Erzeugung eines zeichenbasierten Stroms

```
inputReader = new InputStreamReader(System.in);
```

- Zuerst wird mit Hilfe der Anweisung `new InputStreamReader()` aus der byte-orientierten Standardeingabe ein zeichenbasierter Datenstrom erzeugt.
- Die Klasse ist in der Bibliothek `java.io.InputStreamReader` definiert.
- Falls kein Zeichensatz angegeben ist, wird der Standard-Datensatz für die Konvertierung genutzt.

Gepufferten Stream erzeugen

```
inputReader = new InputStreamReader(System.in);
input = new BufferedReader(inputReader);
```

- Der zeichenbasierte Datenstrom der Objektvariablen `inputReader` wird dem Konstruktor der Klasse `BufferedReader` als Vorlage übergeben.
- Mit Hilfe der Objektvariablen `input` kann von einem zeichenbasierten gepufferten Datenstrom gelesen werden.

Klasse „BufferedReader“

```
Charset inputCharset = Charset.forName("ISO-8859-1");

InputStreamReader inputReader = null;
BufferedReader stdin = null;

inputReader = new InputStreamReader(System.in,
                                    inputCharset);

stdin = new BufferedReader(inputReader);
output = new OutputStreamWriter(System.out);
```

Objekt „Zeichensatz“

```
Charset inputCharset = Charset.forName("ISO-8859-1");
```

- Mit Hilfe der Klassenmethode `.forName()` der Klasse `Charset`, wird ein Objekt von einem bestimmten Zeichensatz erzeugt.
- Der gewünschte Zeichensatz wird der Methode als Parameter übergeben.
- Für die Klasse `Charset` muss die Bibliothek `java.nio.charset.Charset` importiert werden.

Angabe eines Zeichensatzes

```
inputReader = new InputStreamReader(System.in,
                                    inputCharset);
```

- Zuerst wird mit Hilfe der Anweisung `new InputStreamReader()` aus der byte-orientierten Standardeingabe ein zeichenbasierter Datenstrom erzeugt.
- Als erster Parameter wird die Vorlage für den zu lesenden Stream angegeben.
- Als zweiter Parameter kann ein Objekt `Charset` übergeben werden. Mit Hilfe dieses Objekts wird der Zeichensatz der einzulesenden Zeichen festgelegt.

Abfangen eines Fehlers „Zeichenkodierung“

```
catch (UnsupportedEncodingException e) {
            System.out.println("Falscher Zeichensatz");
}
catch(IOException e){
            e.printStackTrace();
}
```

- Der Laufzeitfehler `UnsupportedEncodingException` ist in der Bibliothek `java.io.UnsupportedEncodingException` definiert.
- Laufzeitfehler bezüglich der Zeichenkodierung müssen vor Ein- und Ausgabefehlern abgefangen werden.

Schließen des Datenstroms

```
input.close();
```

- Die Objektvariable `input` verweist auf einen Datenstrom, der geschlossen wird.
- Falls die Objektvariable auf kein passenden Stream verweist, wird der Fehler `NoSuchElementException` ausgegeben.

Zeilenweise einlesen

```
zeile = input.readLine();
```

- Die Methode `.readLine()` liest eine Zeile aus dem Datenstrom.
- Die Methode liest bis zum Zeichen „\n“ oder „\r“ oder einer Kombination aus beiden.
- Falls das Ende des Datenstroms erreicht ist, wird `null` zurückgegeben.

Einlesen von Zeichen

```
wert = input.read();
zeichen = (char)wert;
```

- Die Methode `.read()` liest die Kodierung eines Zeichens aus einem Stream aus.
- Diese Kodierung kann wiederum in ein Character konvertiert werden.
- Hinweis: Zeichen, die nicht dargestellt werden können, werden als Fragezeichen oder Rechteck angezeigt.

Trennung von Zeilen

```
import java.util.StringTokenizer;

StringTokenizer token;
token = new StringTokenizer(zeile, " ");
```

- Die Klasse `java.util.StringTokenizer` bietet die Möglichkeit eine String an einem Trennzeichen zu trennen. Standardmäßig wird ein String mit Hilfe der Leerzeichen getrennt.
- Der getrennte String wird in einem Objekt `StringTokenizer` gespeichert.

Trennung des Strings

```
StringTokenizer token;
token = new StringTokenizer(zeile, " ");
```

- Die Anweisung `new StringTokenizer erzeugt` ein Objekt vom Typ `StringTokenizer`)
- Dem Konstruktor der Instanz wird als erstes Argument der zu trennende String übergeben. Optional kann dem Konstruktor das Trennzeichen übergeben werden. In diesem Beispiel wird das Standard-Trennzeichen „Leerzeichen“ genutzt.

„Wörter“ lesen

```
while(token.hasMoreElements() == true){
      element = token.nextElement().toString();
}
```

- Die Methode `.hasMoreElements()` gibt true zurück, wenn weitere Elemente vorhanden sind.
- Die Methode `.nextElement()` holt das nächste Objekt. Das nächste Objekt entspricht einem „Wort“ in der Zeichenkette.
- Dieses Objekt wird in einem String mit Hilfe der Methode `.toString()` umgewandelt.

Schreiben und Lesen von zeichenorient. Streams

- Im Paket `java.io` sind verschiedene Klassen zum Lesen und Schreiben von Zeichen aus einer Datei etc. definiert.
- `java.io.Reader` und `java.io.Writer`.
- `java.io.FileReader` und `java.io.FileWriter`.

Klasse „FileReader“

```
import java.io.FileReader;

public class Java13_FileRead {
   public static void main(String[] args) {
        String zeile;
        String pfad = "C:\\NetBeansProjects\\";
        String datei = "dax.csv";

        FileReader dateiRead =
              new FileReader(pfad + datei);
```

Import der Klasse

```
import java.io.FileReader;
```

- Die Klasse `FileReader` ist in dem Paket `java.io` definiert.
- Der Punktoperator verbindet die Ordner (Pakete) mit den darin enthaltenen Dateien (Klassen). Ordner können wiederum Unterordner enthalten.

Variable der Klasse ...

```
FileReader dateiRead
```

- Objektvariable vom Datentyp `FileReader`.
- Variablen, die ein Verweis auf eine Instanz von der Klasse `FileReader` speichern können.
- Objektvariable, die auf eine textbasierte Datei verweist.

Erzeugung einer Instanz der Klasse ...

```
FileReader dateiRead = new FileReader(pfad + datei)
```

- Das Schlüsselwort `new FileReader` erzeugt ein neues Objekt von der Klasse `FileReader`.
- Dem Konstruktor wird die Datei übergeben, die gelesen werden soll.
- Die Ausnahme `FileNotFoundException` wird ausgegeben, falls die Datei an dem angegebenen Speicherort nicht vorhanden ist.

Abfangen von Fehlern

```
try (
 FileReader datei = new FileReader(pfad + datei);
 BufferedReader inp = new BufferedReader(dateiRead))
{
```

- Versuche, die in dem `try`-Block zusammengefassten Anweisungen auszuführen …
- In den runden Klammern nach dem Schlüsselwort `try` können Ressourcen angegeben werden, die mit der Methode .close() geschlossen werden müssen.

try plus Ressourcen

```
try (
 FileReader datei = new FileReader(pfad + datei);
 BufferedReader inp = new BufferedReader(dateiRead))
{

```

- Einführung mit Java 7.
- Die Definition der Ressourcen werden in den runden Klammern wie jede andere Java-Anweisung mit dem Semikolon beendet.
- Wenn die Klasse das Interface `java.lang.AutoCloseable` implementiert hat, wird die Ressourcen automatisch am Ende des `try`-blocks geschlossen.

Zeilenweises einlesen

```
try (
 FileReader datei = new FileReader(pfad + datei);
 BufferedReader inp = new BufferedReader(dateiRead))
{
            zeile = input.readLine();
```

- Die Daten der Datei werden in einem `BufferedReader` zwischengespeichert.
- Aus diesem Buffer werden die Daten mit Hilfe der Methode `.readLine()` gelesen.

Klasse „FileWriter“

```
import java.io.FileWriter;
import java.io.IOException;

public class Java13_IOFileWrite {
    public static void main(String[] args) {
        try (
              FileWriter dateiWrite = new FileWriter(file)
            )
        {
          dateiWrite .write(zeile);
          dateiWrite .append("\n");
```

Import der Klasse

```
import java.io.FileWriter;
```

- Die Klasse `FileWriter` ist in dem Paket `java.io` definiert.
- Der Punktoperator verbindet die Ordner (Pakete) mit den darin enthaltenen Dateien (Klassen). Ordner können wiederum Unterordner enthalten.

Variable der Klasse ...

```
FileWriter dateiWrite
```

- Objektvariable vom Datentyp File`Writer`.
- Variablen, die ein Verweis auf eine Instanz von der Klasse File`Writer` speichern können.
- Objektvariable, die auf eine vorhandene zeichenbasierte Datei verweist.

Erzeugung einer Instanz der Klasse ...

```
FileWriter dateiWrite = new FileWriter(file)
```

- Das Schlüsselwort `new FileWriter` erzeugt ein neues Objekt von der Klasse `FileWriter`.
- Dem Konstruktor wird ein Objekt `File` übergeben.

Klasse „File“

```
import java.io.File;

File file = null;

try{
    file = new File (pfad + datei);

    if (!file.exists()){
        file.createNewFile();
   }

}
```

Import der Klasse

```
import java.io.File;
```

- Die Klasse File ist in dem Paket `java.io` definiert.
- Der Punktoperator verbindet die Ordner (Pakete) mit den darin enthaltenen Dateien (Klassen). Ordner können wiederum Unterordner enthalten.

Variable der Klasse ...

```
File file = null;
```

- Objektvariable vom Datentyp `File`.
- Die Variable kann auf eine bestehende Datei verweisen.
- Das Objekt enthält Informationen zu der Datei, deren Referenz in der Objektvariablen gespeichert ist.
- In diesem Beispiel ist die Objektvariable undefiniert. Die Zuweisung `null` setzt eine Variable auf ein leeres Objekt.

Erzeugung einer Instanz der Klasse ...

```
file = new File (pfad + datei);
```

- Das Schlüsselwort `new File` erzeugt ein neues Objekt von der Klasse File.
- Dem Konstruktor wird der absolute Pfad zu einer Datei und der Dateiname als String übergeben.

Ist die Datei vorhanden?

```
if (!file.exists()  &&  !file.isFile()){
      file.createNewFile();
}
```

- Die Methode `.exists()` liefert True zurück, wenn die Datei an dem angegebenen Speicherort vorhanden ist.
- Die Methode `.isFile()` überprüft, ob die Informationen in dem Objekt als Datei interpretiert werden können.
- Falls die Datei nicht vorhanden ist, wird eine leere Datei mit Hilfe der Methode `.createNewFile()` entsprechend der Informationen im Objekt erzeugt werden.

Attribute der Datei

```
if (!file.canWrite()){
     System.exit(0);
}
```

- Die Methode `.canWrite()` liefert True zurück, wenn die Datei in die Datei geschrieben werden kann.
- Die Methode `.canRead()` überprüft, ob die Daten aus der Datei gelesen werden können.
- In diesem Beispiel wird das Programm durch die Anweisung `System.exit(0)` beendet, wenn in die Datei keine Daten geschrieben werden können.

Schreiben in eine Datei

```
FileWriter dateiWrite = new FileWriter(file)
dateiWrite.write(zeile);
dateiWrite.append("\n");
```

- Die Methode `.write()` schreibt einen String in die Datei.
- Die Methode `.append()` hängt einen String oder Char an den bestehenden Text an.