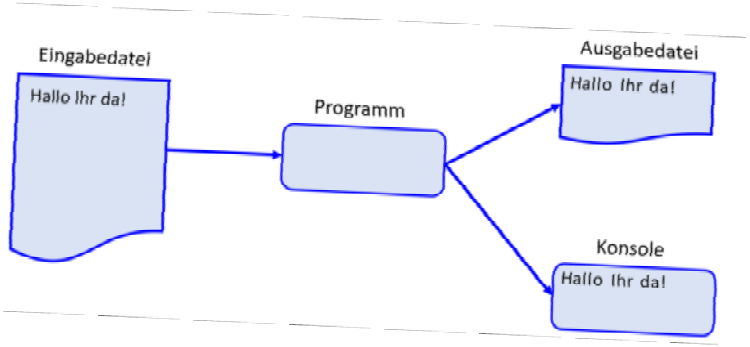
DHBW Duale Hochschule Baden-Württemberg Stuttgart Campus Horb

Vorlesung Programmieren
Thema 12: Ein- und Ausgabe

Olaf Herden
Fakultät Technik
Studiengang Informatik




```
4711#Meier#Anton#...
4712#Müller#Berta#...
4713#Kaiser#Cäsar#...
```

Stand: 04/2024

Gliederung

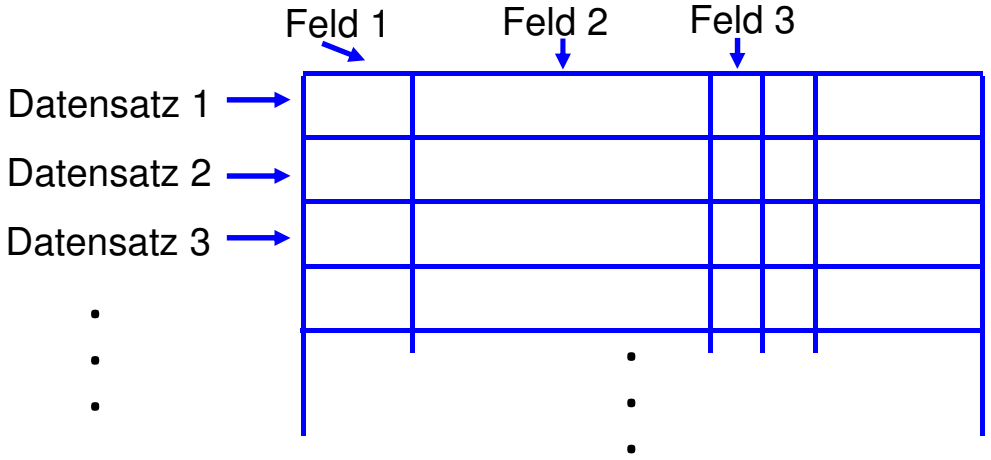
- Dateiorganisation
- Zugriff auf das Dateisystem
- Lesen und Schreiben in Dateien
- Eingabe über die Tastatur
- Klasse `PrintWriter`
- Weitere I/O-Klassen

Motivation

- Bisher: Programme rechnen, keine Interaktion mit Umgebung
- Wünschenswert: Ein- und Ausgaben
  - Kommunikation zwischen Mensch und Maschine
  - Kommunikation von Maschinen untereinander (z.B. Rechner-Drucker)
  - Persistenz von Objekten
- Persistenz:
  - Dauerhaftes Speichern von Daten über das Ende einer Programmausführung hinaus
  - Daten können vom gleichen oder anderen Programmen später weiter verwendet werden
  - Wird erreicht, indem die Daten in einer Datei oder Datenbank abgelegt werden
  - Gegenbegriff: Transienz
- Zu klären: Wie wird gespeichert (Format einer Datei), damit Leser und Schreiber in Datei gespeicherte Daten verstehen?
- In Java: Reihe vordefinierter Formate in Form von Klassen/Interfaces im Paket `java.io.*`

Dateien (I)

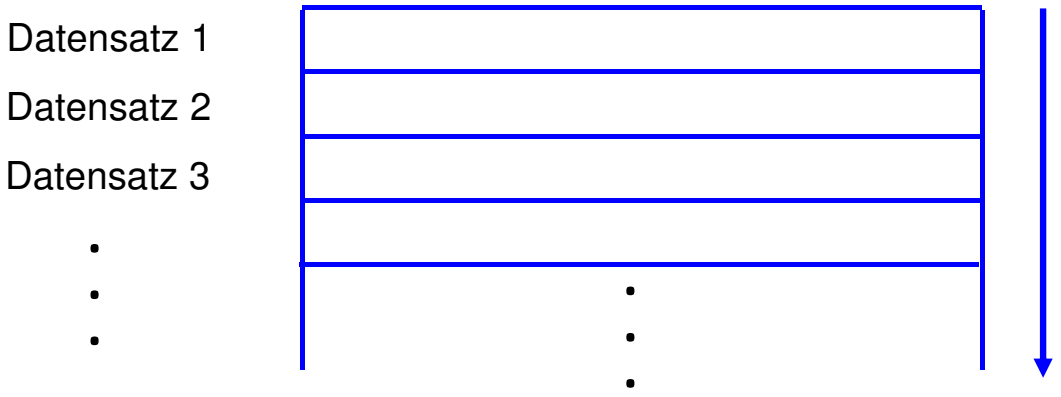
- Besteht aus Folge von gleichartig aufgebauten Datensätzen
- Jeder Datensatz besteht aus unterschiedlichen Elementen, die als Felder bezeichnet werden:



- Je nach Zugriffsart unterscheidet man folgende Formen:
  - Sequentielle Dateien
  - Direkte Dateien
  - Indexsequentielle Dateien

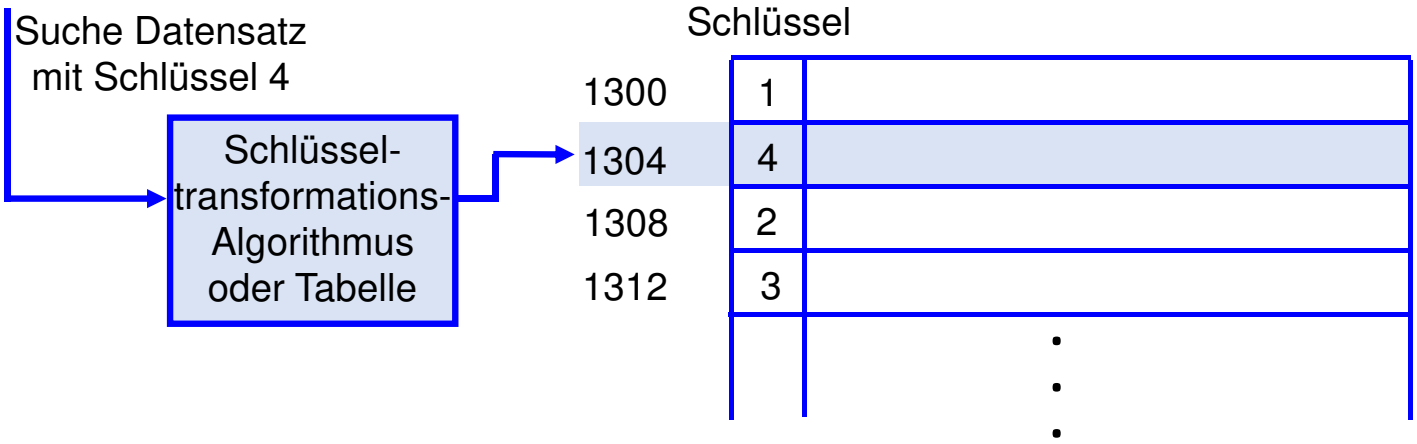
Sequentielle Dateien

- Speichern Datensätze fortlaufend in Reihenfolge der Eingabe
- Zugriff auf Datensätze nur in dieser Reihenfolge



Direkte Dateien

- Zugriff über Schlüssel
- Mittels Tabelle oder Algorithmus wird Adresse des Datensatzes ermittelt
- Schlüssel: Datensatzteil, der diesen eindeutig identifiziert

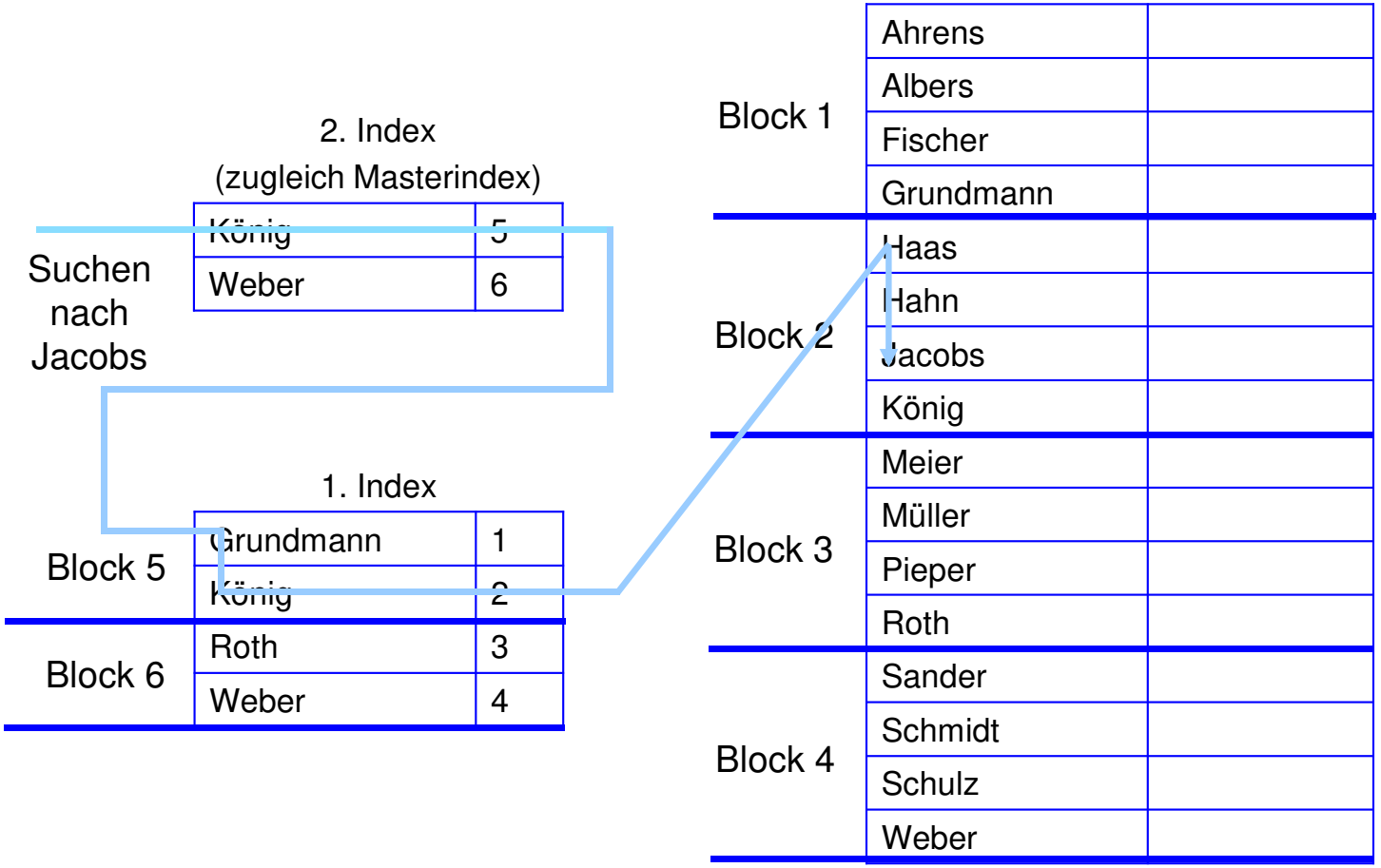


- Bei Zugriff auf Datensatz kein Lesen aller vor ihm liegenden Datensätze notwendig
- Damit: Zugriffszeit auf Datensatz unabhängig von Position in Datei

Indexsequentielle Dateien (I)

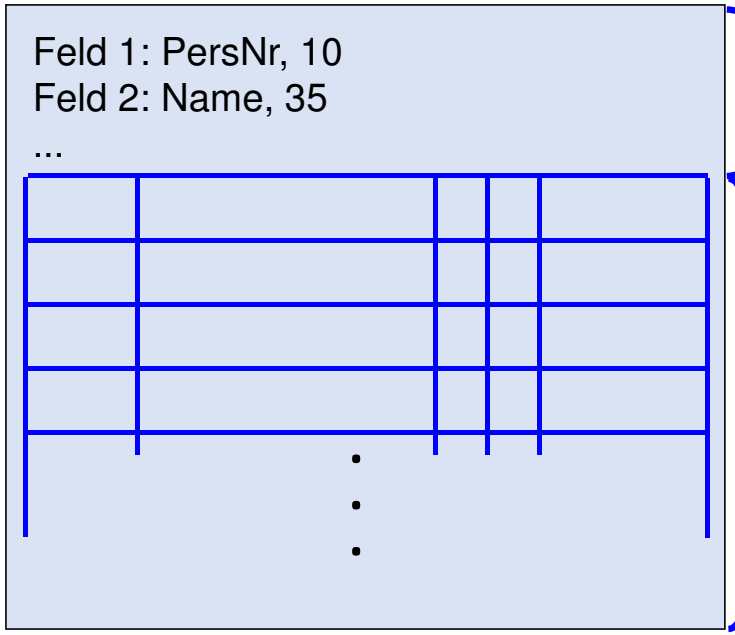
- Mischform aus sequentieller und direkter Dateiorganisation:
  - Folge von Datensätzen, die sequentiell nach bestimmten Merkmal geordnet sind
  - Unterteilung in gleich große Einheiten (Blöcke)
  - Blöcke werden unter Index zusammengefasst
  - Indexeintrag jeweils größtes in Block vorkommende Merkmal
  - Sequentielles Durchlaufen der Datensätze innerhalb eines Blocks
  - Blockgröße: Größe eines physikalischen Blocks der Speichereinheit
  - Mehrere Indexe (z.B. so viele wie ein Block aufnehmen kann) werden wieder unter einem Index zusammengefasst
  - Mehrstufige Fortsetzung dieser Vorgehensweise
  - Masterindex: Index der letzten Stufe

Indexsequentielle Dateien (II)



Weitere Begriffe (I)

- Header: am Dateianfang, Informationen über weitere Daten
- Body: Rest der Datei, „eigentliche“ Daten
- Metadaten: „Daten über Daten“



Datei-Header: Enthält Metadaten, die den Aufbau der Datei beschreiben

Datei-Body: Enthält die eigentlichen Daten, die wie in den Metadaten beschrieben aufgebaut sind

Weitere Begriffe (II)

- Trennung Felder in Datensätzen:
  - Mit Leerzeichen bis Maximallänge aufgefüllt
  - CSV (Comma Separated Value): durch Trennzeichen getrennt
  - Zeichen für Satzende definieren (z.B. Zeilenende)
- Beispiel:
  - Aufgefüllt:

```
4711 Meier          Anton      ...
4712 Müller         Berta      ...
4713 Kaiser         Cäsar      ...
...
```

  - CSV (mit # als Trennzeichen):

```
4711#Meier#Anton#...
4712#Müller#Berta#...
4713#Kaiser#Cäsar#...
...
```

Übersicht

- Dateiorganisation
- Zugriff auf das Dateisystem
- Lesen und Schreiben in Dateien
- Eingabe über die Tastatur
- Klasse `PrintWriter`
- Weitere I/O-Klassen

Dateisystem

- Jedes Betriebssystem stellt ein Dateisystem zur Verfügung
- Besteht aus Dateien und Verzeichnisstruktur
- Dateien: Enthalten die eigentlichen Daten
- Verzeichnisstruktur: Organisiert die Dateien
- Dateien:
  - Attribute: Name, Typ, Größe, Datum, ...
  - Operationen: Erzeugen, Löschen, Lesen, Schreiben, ...
- Verzeichnisstruktur:
  - Organisiert die Dateien in baumförmiger Struktur
  - Operationen: Erzeugen, Löschen, Inhalt anzeigen, ...

Zugriff Dateisystem in Java

- Klasse `File` im Paket `java.io`:
  - Plattformunabhängige Datei- und Verzeichnisnamen
  - Verzeichnisse erzeugen, umbenennen und löschen
  - (Temporäre) Dateien erzeugen
  - Abfrage Eigenschaften wie Existenz, Lesbarkeit, Typ, Größe etc.
- Konstruktor `public File (String <Pfadname>)` erzeugt Objekt mit dem angegebenen Pfadnamen
- Beispiel: Temporäre Datei erzeugen

```
File datei = File.createTempFile("Test","TXT");
```

schreibt eine Datei mit angegebenem Prä- und Suffix in das TEMP-Verzeichnis

Wichtige Methoden in `java.io.File` (I)

- Methoden zum Abprüfen von Eigenschaften:
  - **`public boolean`** `exists()`: Prüft, ob Objekt existiert
  - **`public boolean`** `canRead()`: Prüft, ob Objekt gelesen werden kann
  - **`public boolean`** `canWrite()`: Prüft, ob Objekt geschrieben werden kann
  - **`public boolean`** `isFile()`: Prüft, ob Objekt Datei ist
  - **`public boolean`** `isDirectory()`: Prüft, ob Objekt Verzeichnis ist
  - **`public boolean`** `isHidden()`: Prüft, ob Objekt Eigenschaft versteckt besitzt
  - **`public long`** `length()`: Gibt Länge des Objekts zurück

Wichtige Methoden in `java.io.File` (II)

- Methoden für Verzeichnisse:
  - `public boolean delete()`: Löscht Datei oder Verzeichnis
  - `public void deleteOnExit()`: Löscht Objekt beim Beenden der VM
  - `public boolean mkdir()`: Legt Verzeichnis an
  - `public boolean mkdirs()`: Legt Verzeichnis und eventuelle benötigte Elternverzeichnisse an
  - `public String[] list()`: Liefert Feld mit Auflistung aller Einträge im entsprechenden Verzeichnis

Übersicht

- Dateiorganisation
- Zugriff auf das Dateisystem
- Lesen und Schreiben in Dateien
- Eingabe über die Tastatur
- Klasse `PrintWriter`
- Weitere I/O-Klassen

Datenströme (Streams)

- Abstrakte Konstrukte mit der Fähigkeit
  - Daten auf ein imaginäres Ausgabegerät zu schreiben
  - Daten von einem imaginären Eingabegerät zu lesen

- Sequentiell organisierte Datenmengen
- Möglichkeit Zugriff auf reale Ein- bzw. Ausgabegeräte in konkreten Unterklassen
  - Konkrete Datenquellen: Datei, Platte, Tastatur, URL, Socket (Netzwerk), ...
  - Konkrete Datensenken: Datei, Platte, Bildschirm, Drucker, Socket (Netzwerk), ...
- Realisierung aller Ein- und Ausgaben in Java über Datenströme (Ausnahme GUI)
- Unterstützung unterschiedlicher Datenströme durch Paket `java.io`

Prinzipielle Verarbeitungsweise

- Prinzipiell verlaufen Lesen und Schreiben gleich
- Lesen:
  - Öffnen eines Datenstroms
  - Solange Informationen vorhanden ⇒ Lese diese Information
  - Schließe den Datenstrom
- Schreiben:
  - Öffnen eines Datenstroms
  - Solange Informationen vorhanden ⇒ Schreibe diese Information
  - Schließe den Datenstrom
- In Java werden zwei Arten von Datenströmen unterschieden:
  - Byteströme: Folgen von 8-bit-Zeichen (Bytes)
    - Abstrakte Klassen in `java.io`: `InputStream` und `OutputStream`
  - Zeichenströme: Folgen von 16-bit-Zeichen (Unicode-Zeichen)
    - Abstrakte Klassen in `java.io`: `Reader` und `Writer`

Eingabeklasse `InputStream`

- Abstrakte Klasse `java.io.InputStream`
  - Superklasse aller Byte-Eingabeströme
  - Definiert grundlegende Methoden aller Byte Eingabestrom-Klassen
- Wichtigste Methoden:
  - `public abstract int read() throws IOException`
    - Liest ein Byte aus Datenstrom
    - Muss von Unterklasse überschrieben werden
  - `public int read(byte[] b) throws IOException`
    - Liest Feld von Bytes aus einem Datenstrom
    - Falls von Unterklasse nicht überschrieben, wird read() verwendet
  - `public int available() throws IOException`
    - Ermittelt Anzahl noch lesbarer Bytes
  - `public void close() throws IOException`
    - Schließt Datenstrom und gibt Systemressourcen frei

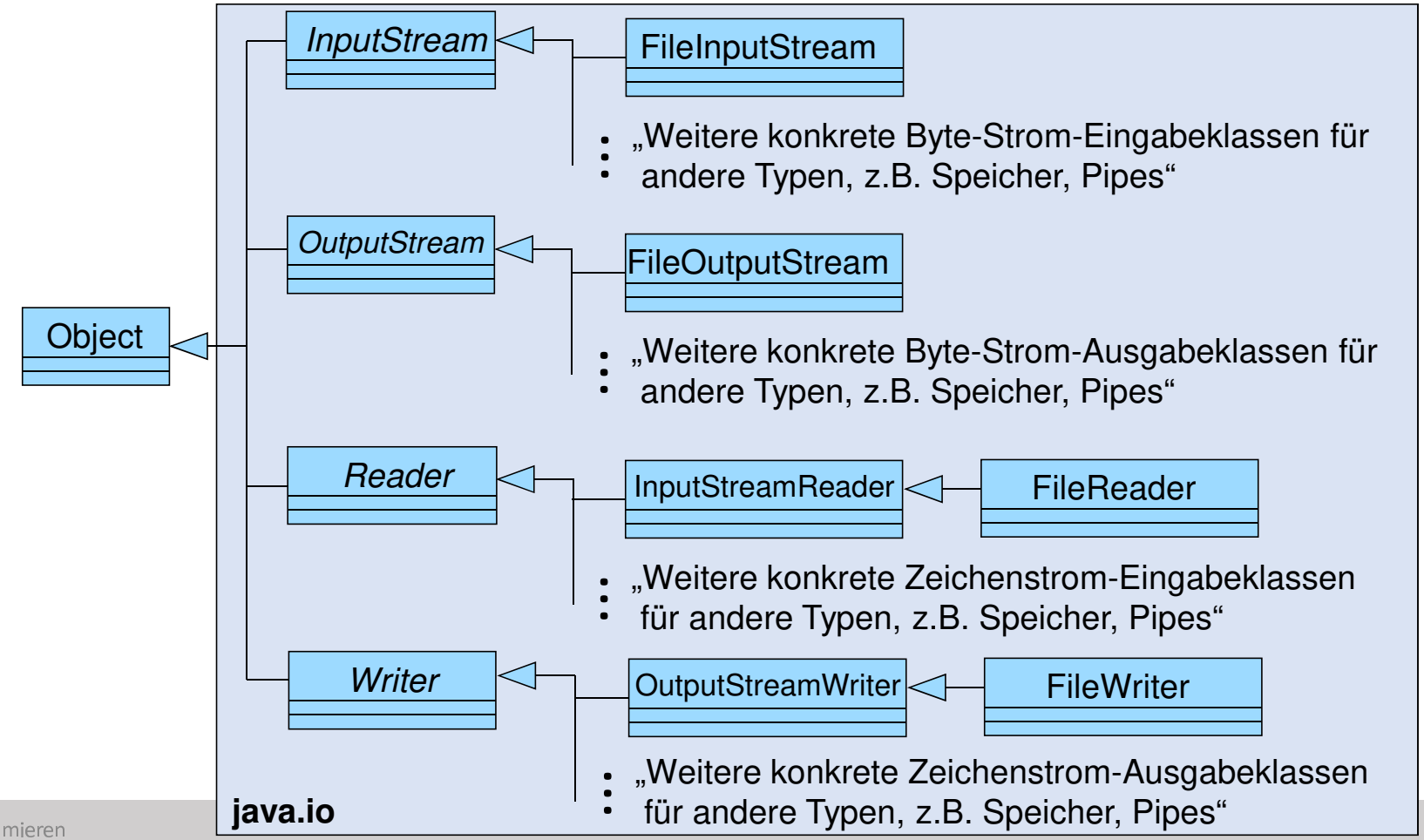
Ausgabeklasse `OutputStream`

- Abstrakte Klasse `java.io.OutputStream`
  - Superklasse aller Byte-Ausgabeströme
  - Definiert die grundlegenden Methoden aller Byte Ausgabestrom-Klassen
- Wichtigste Methoden:
  - `public abstract void write(int b) throws IOException`
    - Schreibt ein Byte in einen Datenstrom
    - Muss von Unterklasse überschrieben werden
  - `public void write(byte[] b) throws IOException`
    - Schreibt Feld von Bytes in Datenstrom
    - Falls von Unterklasse nicht überschrieben, wird write(int b) verwendet
  - `public void flush() throws IOException`
    - Entleerung einer Puffers
- Abstrakte Klassen `Reader` und `Writer`: analoger Aufbau
- Unterschied: Verarbeitung von Zeichen anstelle von Bytes

Konkrete Klassen für Datenströme

- Konkretisierung abstrakter Stromklassen für Typ der Quelle bzw. Senke
- Als Quellen- bzw. Senkentyp stehen zur Verfügung:
  - Speicher
  - Datei
  - Pipe(lines)
- Konkrete Klassen für Bearbeitung von Dateien:
  - Zur Bytestrom-basierten Verarbeitung: `FileInputStream`, `FileOutputStream`
  - Zur Zeichenstrom-basierten Verarbeitung: `FileReader`, `FileWriter`

Aufbau Paket `java.io`

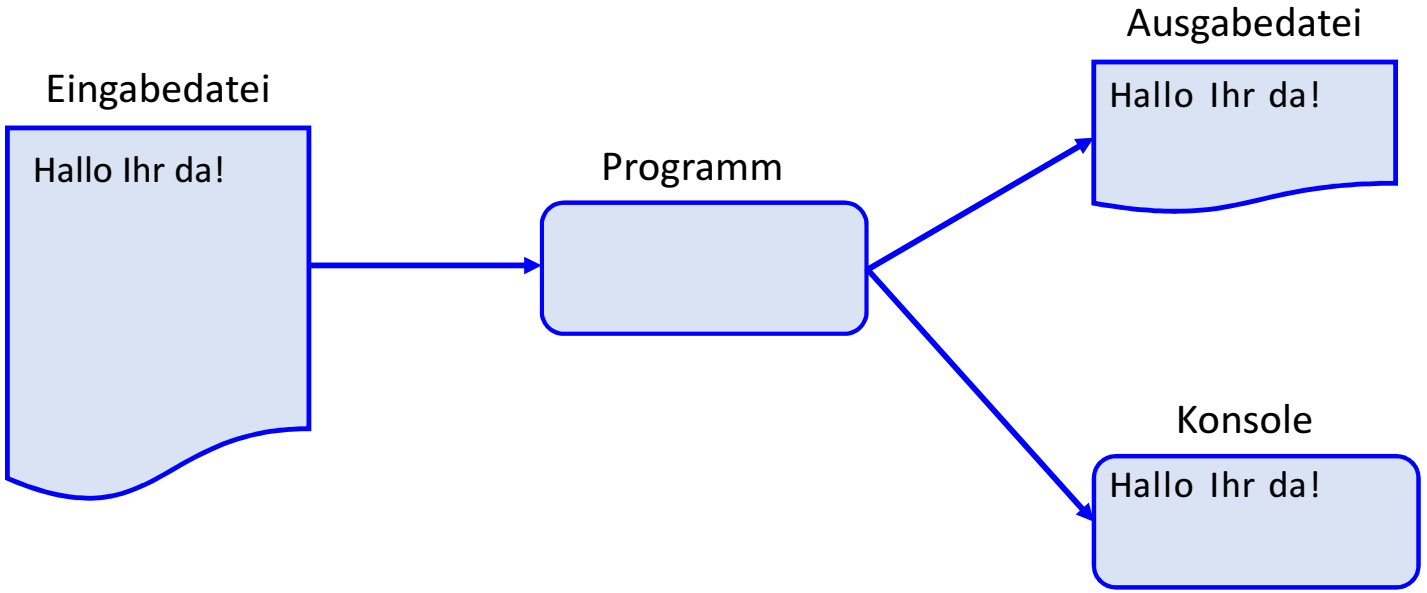


Datenstromklassen

	Zeichenströme	Byteströme
Hauptspeicher	`CharArrayReader` `CharArrayWriter`	`ByteArrayInputStream` `ByteArrayOutputStream`
	`StringReader` `StringWriter`	`StringBufferInputStream`
Pipe	`PipeReader` `PipeWriter`	`PipedInputStream` `PipedOutputStream`
Datei	`FileReader` `FileWriter`	`FileInputStream` `FileOutputStream`

`FileReader` und `FileWriter` (II)

- Ermöglichen Lesen aus bzw. Schreiben in Dateien
- Beispiel: Zeichenweise aus Eingabedatei zeichenweise lesen und in Ausgabedatei und auf Konsole schreiben



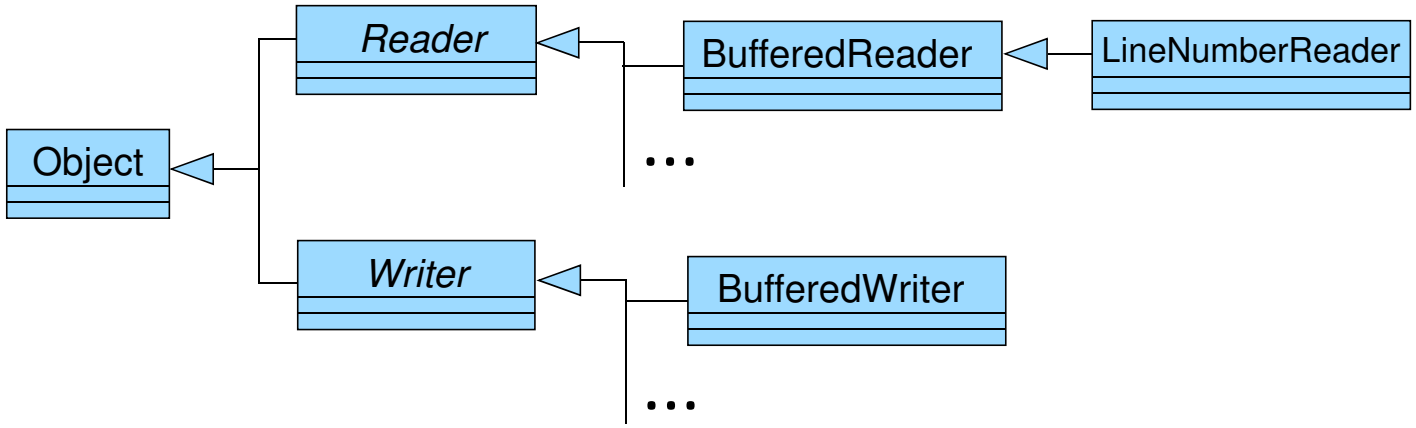
FileReader und FileWriter (I)

```
( 1) import java.io.*;
( 2) public class Bsp2{
( 3)   public static void main(String[] args){
( 4)     try{
( 5)       FileReader eingabe = new FileReader("Ein.TXT");
( 6)       FileWriter ausgabe = new FileWriter("Aus.TXT");
( 7)       int zeichen;
( 8)       while ((zeichen=eingabe.read()) != -1){
( 9)         ausgabe.write(zeichen);
(10)         System.out.print((char) zeichen);
(11)       }
(12)       eingabe.close();
(13)       ausgabe.close();
(14)     }
(15)   catch (Throwable e) {;}
(16) }}
```

- Anmerkung: `try-catch`-Block ist hier zwingend notwendig!

Zeilenweises Lesen (I)

- Häufige Anforderung: Nicht zeichen-, sondern zeilenweise Lesen und Schreiben
- Klassen `LineNumberReader` (erbt über `BufferedReader` von `Reader`) und `BufferedWriter` (erbt von `Writer`) stellen dies zur Verfügung
- Klassendiagramm:



- Buffer (Puffer): Datenstruktur zur Speicherung Sequenz von Werten eines elementaren Datentyps

Zeilenweises Lesen (II)

```
( 1) import java.io.*;
( 2) public class ZeilenweisesLesen{
( 3)   public static void main(String[] args){
( 4)     try{
( 5)       String zeile;
( 6)       FileReader ein = new FileReader("Eingabe.TXT");
( 7)       FileWriter aus = new FileWriter("Ausgabe.TXT");
( 8)       LineNumberReader einZ = new LineNumberReader(ein);
( 9)       BufferedWriter ausZ = new BufferedWriter(aus);
(10)       while ((zeile=einZ.readLine()) != null){
(11)         System.out.println(zeile);
(12)         ausZ.write(zeile);
(13)         ausZ.newLine();
(14)       }
(15)       einZ.close();
(16)       ausZ.close();
(17)     }
(18)     catch (Throwable e) {;}
(19)   }
(20) }
```

Übersicht

- Dateiorganisation
- Zugriff auf das Dateisystem
- Lesen und Schreiben in Dateien
- Eingabe über die Tastatur
- Klasse `PrintWriter`
- Weitere I/O-Klassen

Eingabe über Tastatur

- Klassen `BufferedReader` und `InputStreamReader` ermöglichen Einlesen von Werten über Tastatur
- Analog zu `System.out` für Ausgabe `System.in` für Eingabe
- Objekt vom Typ String, ggf. Typecast durchführen

Beispiel

```
( 1) import java.io.*;
( 2) public class Tastatur{
( 3)     public static void main(String[] args){
( 4)       try{
( 5)         BufferedReader eingabe = new BufferedReader
( 6)               (new InputStreamReader(System.in));
( 7)         String eingabezeile = "";
( 8)         System.out.println("Bitte Eingabe machen: ");
( 9)         eingabezeile = eingabe.readLine();
(10)         System.out.println("Die Eingabe war : " +
(11)                                          eingabezeile);
(12)       }
(13)       catch (Throwable e) {;}
(14)     }
(15) }
```

Übersicht

- Dateiorganisation
- Zugriff auf das Dateisystem
- Lesen und Schreiben in Dateien
- Eingabe über die Tastatur
- Klasse `PrintWriter`
- Weitere I/O-Klassen

Konstruktoren

- Klasse `PrintWriter`
- Schreibt Zeichen in Ausgabestrom
- Konstruktoren:
  - `public PrintWriter(Writer)`
  - `public PrintWriter(Writer, boolean)`
  - `public PrintWriter(OutputStream)`
  - `public PrintWriter(OutputStream, boolean)`
  - Boolescher Parameter gibt an, ob automatischer Flush (Leeren des Zwischenpuffers) durchgeführt werden soll

Methoden

- Methoden:
  - `public void write(<Typ>)`
    - Schreibt in Ausgabe
    - <Typ> kann Integer oder String oder `char`-Feld sein
  - `public void flush()`
    - Führt auf Ausgabestrom Flush durch
  - `public void close()`
    - Schließt Ausgabestrom
  - `public void print(<Typ oder Objekt>)`
    - Schreibt in Ausgabe

Beispiel

```
( 1) // Defintion von Variablen und Objekten
( 2) Person p = new Person("Franz","Meier",75);
( 3) int i = 6;
( 4) String s = "Hallo";
( 5) // Anlegen eines PrintWriter-Objektes ...
( 6) PrintWriter pw = new PrintWriter(System.out,true);
( 7) // ... und ausgeben der Variablen und Objekte
( 8) pw.println(i);
( 9) pw.println(s);
(10) pw.println(p);
(11) // Schließen des Ausgabestroms
(12) pw.close();
```

führt zur Ausgabe

```
6
Hallo
Franz Meier ist 75 Jahre alt.
```

Übersicht

- Dateiorganisation
- Zugriff auf das Dateisystem
- Lesen und Schreiben in Dateien
- Eingabe über die Tastatur
- Klasse `PrintWriter`
- Weitere I/O-Klassen

Weitere Klassen in `java.io`

- `ObjectInputStream` und `ObjectOutputStream`
  - Führen Ein-/Ausgabe von Objekten durch
  - Objekte müssen serialisierbar sein
- `CheckedInputStream` und `CheckedOutputStream`
  - Führen Ein-/Ausgabe inkl. Berechnung von Prüfsummen durch
- `InflaterInputStream` und `DeflaterOutputStream`
  - Führen Ein-/Ausgabe mit Dekomprimierung bzw. Komprimierung durch
- `(G)ZIPInputStream` und `(G)ZIPOutputStream`
  - Führen Ein-/Ausgabe im (G)ZIP-Format durch
- `RandomAccessFile`
  - Ermöglicht wahlfreien Zugriff (lesend und schreibend) auf Dateien

Weiteres zur Ein-Ausgabe

- Klassen in Paket `java.io` ohne Pufferung ⇒ Paket `newio`
- Zerlegen einer Eingabe in Einzelteile (Token) ⇒ Tokenizer (siehe Übung)
- Zugriff auf Datenbank ⇒ jdbc-Treiber und Klassen, Frameworks z.B. Hibernate

Zusammenfassung (I)

- Dateiorganisation:
  - Sequentielle Dateien
  - Direkte Dateien
  - Indexsequentielle Dateien
- Zugriff auf das Dateisystem:
  - Dateien verwalten
  - Verzeichnisse verwalten
- Lesen und Schreiben in Dateien:
  - Datenströme
  - Aufbau Paket `java.io`
  - Konkrete Klassen
  - Zeichenweises Lesen/Schreiben
  - Zeilenweises Lesen/Schreiben

Zusammenfassung (II)

- Eingabe über die Tastatur
- Klasse `PrintWriter`:
  - Konstruktoren
  - Methoden
- Weitere I/O-Klassen:
  - Weitere Klassen im Paket `java.io`
  - Weitere Konzepte

Aufgaben (I)

- **Aufgabe 1**

  Während der Entwicklung von größeren, Datenbank-basierten Systemen ist das Erzeugen von Testdaten eine notwendige Tätigkeit.

  In dieser Aufgabe sollen Dateien mit Testdatensätzen für Kundendaten erzeugt werden.

  Ein einzelner Datensatz soll dabei aus folgenden Feldern bestehen:

  - Sechsstellige, eindeutige Kundennummer
  - Nachname (max. 20 Zeichen)
  - Vorname (max. 20 Zeichen)
  - Straße (max. 20 Zeichen)
  - Hausnummer (max. 5 Zeichen)
  - PLZ (fünf Stellen)
  - Ort (max. 20 Zeichen)
  - Umsatz (Zahlwert zwischen 500 und 2000)

Aufgaben (II)

Für die interne Organisation dieser Datei gibt es zwei Möglichkeiten:

- Felder fester Breite, ist ein tatsächlicher Eintrag nicht so breit wie das Maximalformat, so wird mit Leerzeichen aufgefüllt

Beispiel:

```
123456Meier                Max             Rosenweg           67    ...
123457Müller               Fritz           Nelkenstraße       123   ...
...
```

- Trennzeichen-Dateien (CSV-Dateien), d.h. es wird ein festes Trennzeichen vereinbart (z.B. #), jedes Feld wird in seiner tatsächlichen Länge in die Datei geschrieben und die einzelnen Felder durch das Trennzeichen voneinander getrennt

Beispiel:

```
123456#Meier#Max#Rosenweg#67#...
123457#Müller#Fritz#Nelkenstraße#123...
...
```

Aufgaben (III)

Schreiben Sie ein Java-Programm, das für das oben genannte Beispiel jeweils eine Datei mit Feldern fester Breite und eine mit Trennzeichen erstellt.
Folgende Anforderungen sollen dabei erfüllt werden:

- Die Anzahl der Datensätze soll über die Kommandozeile erfolgen
- Die Nach- und Vornamen sowie Straßen, PLZ und Orte sollen aus vorgegebenen Dateien gelesen werden und jeweils in einem Feld von Strings abgespeichert werden
- Realisieren Sie hierzu eine Klasse StringFeld
- Die Kundennummern sollen ab einem festen Startwert fortlaufend vergeben werden
- Hausnummer und Umsatz sollen per Zufall generiert werden

Aufgaben (IV)

Folgende Hinweise:

- Die Länge eines Strings kann mit der Methode `length()` ermittelt werden
- Eine (Pseudo-)Zufallszahl zwischen 0 und 1 wird mit der Methode `Math.random()` erzeugt, Rückgabetyp ist `double`
- Verwenden Sie als Trennzeichen #
- Im Moodle-Kurs befinden sich Eingabedateien

Aufgaben (V)

- **Aufgabe 2**

  Im Paket `java.util` gibt es eine Klasse `StringTokenizer`, im Paket `java.io` eine Klasse `StreamTokenizer`. Beide haben die Aufgabe, eine Zeichenkette bzw. einen Eingabestrom in Token zu zerlegen. Realisieren Sie mit einer dieser beiden Klassen ein Programm, das aus einer Eingabedatei liest und die Tokentypen Zahl und Zeichenkette unterscheiden kann. Es soll jedes einzelne Token mit Angabe seines Typs ausgegeben werden. Am Ende soll die Anzahl der gelesenen Zahlen und Zeichenketten sowie die Gesamtzahl gelesener Token ausgegeben werden.

  Beispielsweise soll zur Eingabedatei

```
Dieser Text hat 6
Worte und 2 Zahlen.
```

  die Ausgabe

Aufgaben (VI)

```
ZEICHENKETTE : Dieser
ZEICHENKETTE : Text
ZEICHENKETTE : hat
ZAHL         : 6
ZEICHENKETTE : Worte
ZEICHENKETTE : und
ZAHL         : 2
ZEICHENKETTE : Zahlen.
Gelesene Zahlen           : 2
Gelesene Zeichenketten    : 6
Insgesamt gelesene Token  : 8
```

geliefert werden.