



# Objektorientierte Programmierung in Java

## Vorlesung 9 - Eingabe und Ausgabe

Emily Lucia Antosch

HAW Hamburg

06.11.2024

# Inhaltsverzeichnis

1. Einleitung .....	3
2. Stream-Konzept & Bildschirmausgabe .....	7
3. Tastatureingabe .....	12
4. Byteströme & Zeichenströme .....	25
5. Dateien .....	32
6. License Notice .....	38

# 1. Einleitung

---

# 1.1 Wo sind wir gerade?

- In der letzten Vorlesung haben wir uns mit dem Erstellen von graphischen Oberflächen beschäftigt
- Sie können nun
  - ▶ Ausnahmen werfen und fangen,
  - ▶ mit `try` und `catch` Ausnahmen behandeln
  - ▶ und eigene Ausnahmetypen definieren.
- Heute geht es weiter mit den **Ausnahmebehandlungen**.

# 1.1 Wo sind wir gerade?

1. Imperative Konzepte
2. Klassen und Objekte
3. Klassenbibliothek
4. Vererbung
5. Schnittstellen
6. Graphische Oberflächen
7. Ausnahmebehandlung
8. **Eingaben und Ausgaben**
9. Multithreading (Parallel Computing)

## 1.2 Das Ziel dieses Kapitels

- Sie lesen Zeichen, Zeichenketten sowie Zahlenwerte von der Tastatur ein.
- Sie verketteten und verwenden im Java SDK enthaltene Eingabe- und Ausgabeströme zur Eingabe und Ausgabe von Bytes, Zeichen und Textzeilen.
- Sie lesen und schreiben Zeichenketten aus bzw. in Textdateien.

## **2. Stream-Konzept & BildschirmAusgabe**

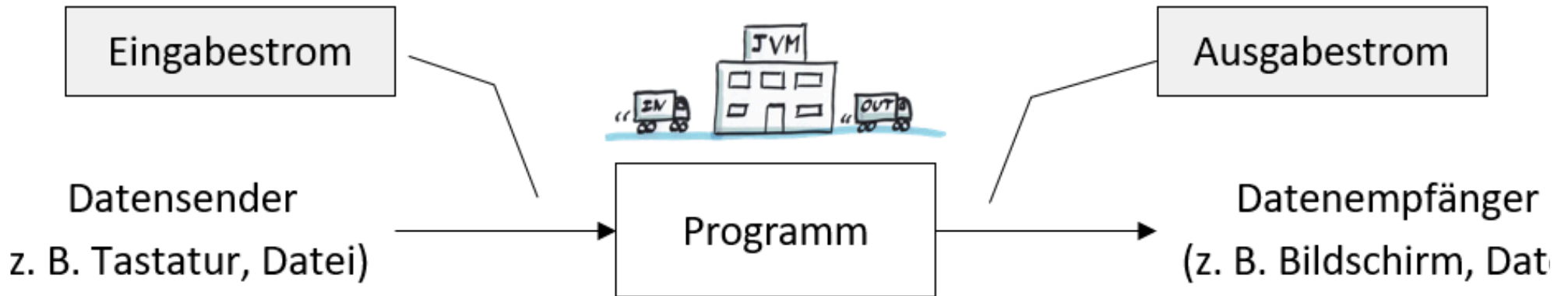
---



# 2.1 Stream-Konzept

## 2. Stream-Konzept & Bildschirmausgabe

- Strom (Stream): Transportiert Daten von Sender („Quelle“) zu Empfänger („Senke“)
- Eingabe (Input): Einlesen von Daten in ein Programm
- Ausgabe (Output): Daten verlassen ein Programm
- Klassenbibliothek enthält etwa 50 Klassen für alle wichtigen Ein- und Ausgabevarianten



## 2.1 Stream-Konzept

## 2. Stream-Konzept & Bildschirmausgabe

- Mit dem, was wir schon gelernt haben:
  - ▶ Was sind eigentlich die Bestandteile von `System.out.println()`?
- Nur das macht Sinn:
  - ▶ `System`: Klasse (da keine Variable `System` deklariert)
  - ▶ `out`: Klassenvariable von `System`, referenziert ein Objekt
  - ▶ `println()`: Methode des über `out` referenzierten Objektes
- Ausgabestrom:
  - ▶ `System.out` referenziert Objekt der Klasse `PrintStream`
  - ▶ Objekt ist mit Bildschirm verbunden

# 2.1 Stream-Konzept

## 2. Stream-Konzept & Bildschirmausgabe


- Ausgewählte Methoden der Klasse `PrintStream`:

Methode	Bedeutung
<code>println(String message)</code>	Ausgabe mit Zeilenumbruch („ <i>print line</i> “)
<code>print(String message)</code>	Ausgabe ohne Zeilenumbruch
<code>printf(String format, Object... arg)</code>	Formatierte Ausgabe (vergleiche <i>String.format()</i> )
<code>format(String format, Object... arg)</code>	Formatierte Ausgabe (vergleiche <i>String.format()</i> )

### ? Frage

- Was wird ausgegeben?

```
1 public static void main(String[] args) {
2     double tempHawaiiCelsius = 15.97;
3     double tempHamburgCelsius = 22.71;
4     String.format("Hawaii: %.1f °C", tempHawaiiCelsius); System.out.printf("Hamburg: %.1f °C", tempHamburgCelsius);
5 }
```

 Java

# 2.1 Stream-Konzept

In System referenzierte Ströme:

Referenz	Datentyp	Bedeutung
<u>System.out</u>	<u>PrintStream</u>	Ausgabe auf Bildschirm
<u>System.err</u>	<u>PrintStream</u>	Fehlerausgabe auf Bildschirm
<u>System.in</u>	<u>InputStream</u>	Eingabe von Tastatur

## 2. Stream-Konzept & Bildschirmausgabe

# **3. Tastatureingabe**

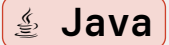
---

- Bietet Methoden zum Einlesen von Texten und einfachen Datentypen (z.B. int)
- Texteingabe wird analysiert und interpretiert („Parsen“, z.B. in Ganzzahl wandeln)
- Erzeugung und Beendigung:
  - ▶ Scanner-Objekt wird im Konstruktor mit Eingabestrom verbunden
  - ▶ Die Verbindung sollte über die Scanner-Methode `close()` beendet werden.



### Beispiel

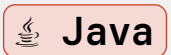
```
1 public class ScannerLine {
2     public static void main(String[] args) {
3         Scanner scanner = new Scanner(System.in);
4
5         System.out.print("Bitte einen Satz eingeben: ");
6         System.out.println(scanner.nextLine());
7         scanner.close();
8     }
9 }
```



### ? Frage

- Hoppla, was passiert hier?

```
1 public class ScannerToken {
2     public static void main(String[] args) {
3         Scanner scanner = new Scanner(System.in);
4
5         System.out.print("Bitte einen Satz eingeben: ");
6         System.out.println(scanner.next());
7         scanner.close();
8     }
9 }
```



- Methode `next()`: Nur erstes Wort anstatt ganzer Satz eingelesen und ausgegeben
- Es werden Wörter und Zeilen unterschieden.



- Trennzeichen mehrerer Eingaben:
  - ▶ Token: Einzelne Wörter oder Werte (z.B. Ganzzahl)
  - ▶ Token in Eingabe durch Trennzeichen getrennt
  - ▶ Standardtrennzeichen ist ein Whitespace (d.h. Leerzeichen, Tabulator, Zeilenumbruch)
- Methoden:
  - ▶ Trennzeichen über Methode `useDelimiter()` änderbar
  - ▶ Über Methode `hasNext()` Abfrage, ob noch Token vorhanden sind

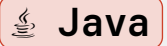
### Aufgabe 1

- Schreiben Sie ein Programm, das einen Satz über `next()` einliest.

# 3.1 Klasse Scanner

## 3. Tastatureingabe

```
1  public class ScannerNext {
2      public static void main(String[] args) {
3          Scanner scanner = new Scanner(System.in);
4
5          System.out.print("Bitte einen Satz eingeben: ");
6          while (scanner.hasNext()) {
7              System.out.println(scanner.next());
8          }
9          scanner.close();
10     }
11 }
```



### ? Frage

- Was geschieht, wenn Sie `scanner.hasNext()` durch `true` ersetzen?
- Wie verhält sich `next()` sobald alle Wörter eingelesen sind?

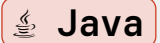
# 3.1 Klasse Scanner

- Spezielle Methoden für einfache Datentypen:
  - ▶ Einlesen: `nextBoolean()`, `nextInt()`, `nextDouble()`, ...
  - ▶ Abfrage: `hasNextBoolean()`, `hasNextInt()`, `hasNextDouble()`, ...

## ? Frage

- Welche Ausgaben werden für die Eingaben „127“, „128“ und „Hamburg“ erzeugt?

```
1 public class ScannerByte1 {
2     public static void main(String[] args) {
3         Scanner scanner = new Scanner(System.in);
4
5         System.out.print("Bitte einen byte-Wert eingeben: ");
6         System.out.println("Eingegeben: " + scanner.nextByte());
7         scanner.close();
8     }
9 }
```



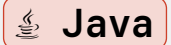
- Fehler beim Parsen:
  - ▶ Eingaben „128“ und „Hamburg“: Ausnahme vom Typ `InputMismatchException`
  - ▶ Hat Basisklasse `RuntimeException` (Ausnahmebehandlung nicht zwingend erforderlich)

### Aufgabe 2

- Das Programm soll nicht durch eine Ausnahme beendet werden:
  - Finden Sie zwei unterschiedliche Möglichkeiten, dies zu vermeiden.
  - Implementieren Sie diese Möglichkeiten.
- Ansätze:
  - ▶ Fangen der Ausnahme
  - ▶ Abfrage über `hasNextByte()`

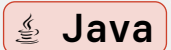
- Ausnahme fangen:

```
1  public class ScannerByte2 {
2      public static void main(String[] args) {
3          Scanner scanner = new Scanner(System.in);
4
5          System.out.print("Bitte einen byte-Wert eingeben: ");
6          try {
7              System.out.println("Eingegeben: " + scanner.nextByte());
8          } catch (InputMismatchException e) {
9              System.out.println("Eingabe ist kein byte-Wert.");
10         } finally {
11             scanner.close();
12         }
13     }
14 }
```



- Datentyp abfragen:

```
1  public class ScannerByte3 {
2      public static void main(String[] args) {
3          Scanner scanner = new Scanner(System.in);
4
5          System.out.print("Bitte einen byte-Wert eingeben: ");
6          if (scanner.hasNextByte()) {
7              System.out.println("Eingegeben: " + scanner.nextByte());
8          } else {
9              System.out.println("Kein byte-Wert: " + scanner.next());
10         }
11         scanner.close();
12     }
13 }
```



### Aufgabe 3

- Einlesen der Komponenten eines Vektors (Datentyp `int`)
- Komponenten einlesen bis anderes Token (z. B. ein Buchstabe) eingegeben wurde
- Ausgabe des Vektors sowie des Betrages



### Beispiel

Integer-Komponenten (mit anderem Zeichen beenden): 7 4 0 15 Ende

$$a = [7, 4, 0, 15]^T$$

$$\|a\| = 17,03$$



```
1  public class ScannerVektor {
2      public static void main(String[] args) {
3          Scanner scanner = new Scanner(System.in);
4          ArrayList<Integer> vector = new ArrayList<Integer>();
5          System.out.print("Integer-Komponenten (mit anderem Zeichen beenden): ");
6          while (scanner.hasNextInt())
7              vector.add(scanner.nextInt());
8          scanner.close();
9          if (vector.size() > 0) {
10             System.out.print("a = [" + vector.get(0));
11             long sumOfSquares = vector.get(0) * vector.get(0);
12
13             for (int i = 1; i < vector.size(); i++) {
14                 System.out.print(", " + vector.get(i));
15                 sumOfSquares += vector.get(i) * vector.get(i);
16             }
17             System.out.println("]^T");
18             System.out.printf("||a|| = %.2f\n", Math.sqrt(sumOfSquares));
19         }
20     }
21 }
```



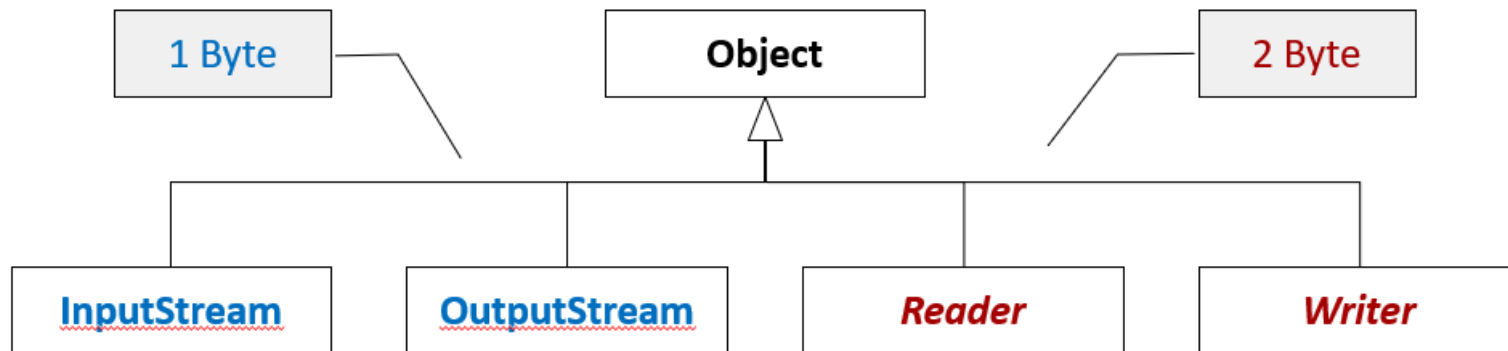
## **4. Byteströme & Zeichenströme**

---

# 4.1 Byteströme & Zeichenströme

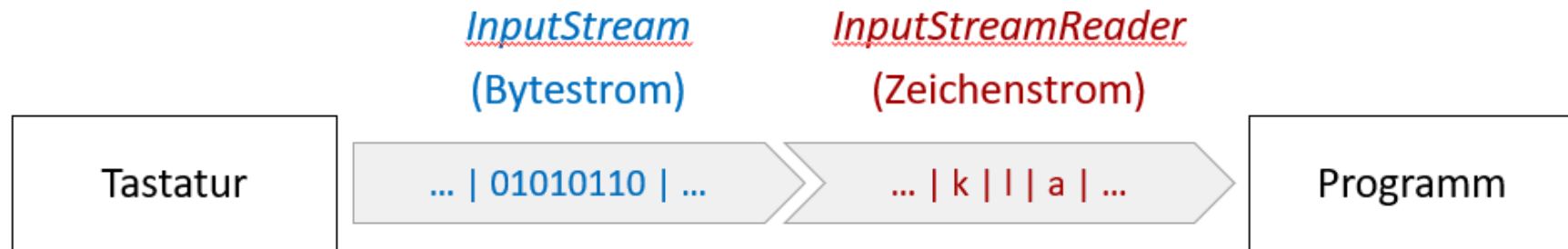
## 4. Byteströme & Zeichenströme

- Was war nochmal die Besonderheit von Zeichen in Java?
  - ▶ Alle Zeichen als 2 Byte (Unicode) codiert
  - ▶ Unterscheide: Ströme, die Elemente aus 1 Byte oder 2 Byte („Zeichen“) transportieren
- Byteströme (byteorientierte Ströme):
  - ▶ Transportierten einzelne Bytes
  - ▶ Klassen `InputStream` und `OutputStream` sowie hiervon abgeleitete Klassen
- Zeichenströme (characterorientierte Ströme):
  - ▶ Transportierten Zeichen aus jeweils 2 Byte
  - ▶ Abstrakte Klassen `Reader` und `Writer` sowie hiervon abgeleitete Klassen



# 4.1 Byteströme & Zeichenströme

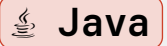
- Tastatur liefert Strom aus einzelnen Bytes (z. B. `System.in` vom Datentyp `InputStream`)
  - ▶ Java-Zeichen bestehen aus 2 Bytes
  - ▶ Bytestrom mit Zeichenstrom verbinden
- Anmerkungen:
  - ▶ Ziel im Folgenden: Veranschaulichung der Verkettung von Strömen
  - ▶ Ja, Tastatureingaben (`Code ≤ 255`) müssten Sie nicht mit einem Zeichenstrom verketteten.
  - ▶ Ja, verwenden Sie für Tastatureingaben ruhig `Scanner`.



# 4.1 Byteströme & Zeichenströme

## 4. Byteströme & Zeichenströme

```
1  public class KeyboardReader1 {
2      public static void main(String[] args) throws IOException {
3          InputStreamReader reader = new InputStreamReader(System.in);
4
5          System.out.print("Bitte ein Zeichen eingeben: ");
6          System.out.println(reader.read());
7          System.out.println(reader.read());
8          System.out.println(reader.read());
9          reader.close();
10     }
11 }
```



Java

# 4.1 Byteströme & Zeichenströme

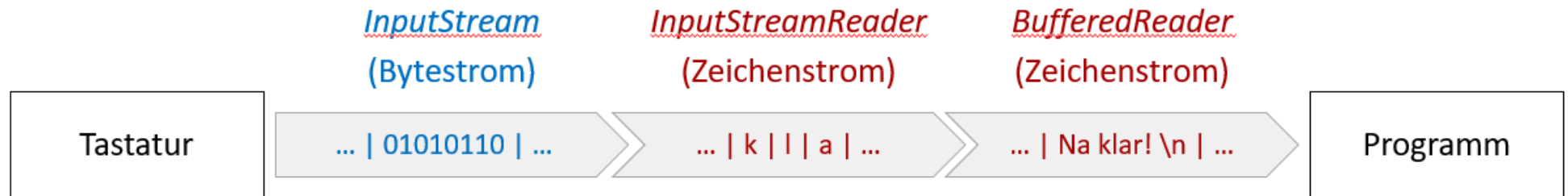
## 4. Byteströme & Zeichenströme

### ? Frage

- Warum wird `read()` dreimal aufgerufen?
- Warum sind die zweite und dritte Ausgabe immer 13 und 10?

# 4.1 Byteströme & Zeichenströme

- `BufferedReader` liest einen Zeichenstrom und puffert die Zeichen
- Bietet z.B. Methode `readLine()` zum Auslesen einer Zeile
- Analog gibt Klasse `BufferedWriter` ganze Zeile über `newLine()` aus



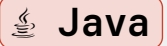
## ☰ Aufgabe 4

- ▶ Ändern Sie das vorherige Beispiel folgendermaßen ab:
  - Einlesen zweier Zeilen über
  - `BufferedReader` Anschließend beide Zeilen ausgeben

# 4.1 Byteströme & Zeichenströme

## 4. Byteströme & Zeichenströme

```
1  public class KeyboardReader2 {
2      public static void main(String[] args) throws IOException {
3          InputStreamReader reader = new InputStreamReader(System.in);
4          BufferedReader bufferedReader = new BufferedReader(reader);
5
6          System.out.print("Bitte erste Zeile eingeben: ");
7          String line1 = bufferedReader.readLine();
8          System.out.print("Bitte zweite Zeile eingeben: ");
9          String line2 = bufferedReader.readLine();
10
11         System.out.println(line1);
12         System.out.println(line2);
13         reader.close();
14     }
15 }
```



Java



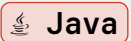
# 5. Dateien

---

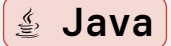
# 5.1 Dateien und Verzeichnissen

- Klasse `File` repräsentiert Datei oder Verzeichnis
  - Objekte beinhalten Informationen über Datei, nicht deren Inhalt
  - IntelliJ verwendet das Projekt-Verzeichnis als Stammverzeichnis zum Lesen/Schreiben.

```
1  public class CreateFile {
2      public static void main(String[] args) throws IOException {
3          File file = new File("Testdatei.txt");
4          boolean isExists = file.exists();
5
6          if (!isExists) {
7              System.out.println("Datei erzeugen");
8              isExists = file.createNewFile();
9          }
10
11         if (isExists && file.isFile()) {
12             System.out.println("Lesen: " + file.canRead());
13             System.out.println("Schreiben: " + file.canWrite());
14             file.delete();
15         }
16     }
17 }
```



```
1  public class ListDirectory {
2      public static void main(String[] args) {
3          File directory = new File(".");
4
5          if (directory.isDirectory()) {
6              String[] children = directory.list();
7              for (String child : children) {
8                  System.out.println(child);
9              }
10         }
11     }
12 }
```



Java

# 5.1 Dateien und Verzeichnissen

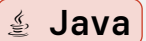
- Byteströme:
  - ▶ Dateien über Klassen `FileInputStream` lesen und über `FileOutputStream` schreiben
- Zeichenströme (z.B. Textdateien):
  - ▶ Dateien über `FileReader` lesen und über `FileWriter` schreiben
  - ▶ Gepufferte Zeichenströme über `BufferedReader` und `BufferedWriter`



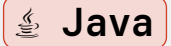
# 5.1 Dateien und Verzeichnissen

- Lassen Sie uns das anwenden:
  - Erstellen Sie ein Programm, das eine Textdatei schreibt.
  - Erstellen Sie ein weiteres Programm, das den Inhalt der Textdatei einliest und ausgibt.

```
1  public class WriteFile {
2      public static void main(String[] args) throws IOException {
3          File file = new File("Testdatei.txt");
4          FileWriter writer = new FileWriter(file);
5          BufferedWriter bufferedWriter = new BufferedWriter(writer);
6
7          bufferedWriter.write("Dies ist die erste Zeile.");
8          bufferedWriter.newLine();
9          bufferedWriter.write("Und hier kommt die zweite Zeile.");
10         bufferedWriter.newLine();
11         bufferedWriter.close();
12     }
13 }
```



```
1  public class ReadFile {
2      public static void main(String[] args) throws IOException {
3          File file = new File("Testdatei.txt");
4          FileReader reader = new FileReader(file);
5          BufferedReader bufferedReader = new BufferedReader(reader);
6
7          while (bufferedReader.ready()) {
8              System.out.println(bufferedReader.readLine());
9          }
10         bufferedReader.close();
11     }
12 }
```



## 6. License Notice

---

## 6.1 Attribution

- This work is shared under the CC BY-NC-SA 4.0 License and the respective Public License
- `link(„https://creativecommons.org/licenses/by-nc-sa/4.0/“)`
- This work is based off of the work Prof. Dr. Marc Hensel.
- Some of the images and texts, as well as the layout were changed.
- The base material was supplied in private, therefore the link to the source cannot be shared with the audience.