

# Objektorientierte Programmierung in Java

## Vorlesung 4 - Klassenbibliotheken

Emily Lucia Antosch

HAW Hamburg

22.10.2024

# Inhaltsverzeichnis

1. Einleitung .....	2
2. Zeichenketten .....	6
3. Arrays .....	32
4. Mehrdimensionale Arrays .....	53
5. Listen .....	62
6. foreach-Schleife .....	67
7. Wrapperklassen & Math-Klasse .....	74
8. License Notice .....	82

# 1. Einleitung

---

# 1.1 Wo sind wir gerade?

- Zuletzt haben wir uns mit Klassen und Objekten beschäftigt.
- Sie können nun
  - einfache Klassen in Java schreiben,
  - aus den Klassen Objekte erzeugen, Attribute verwenden und Methoden aufrufen und
  - Klassenvariablen und Klassenmethoden verwenden.
- Heute geht es weiter mit **Klassenbibliotheken**.

# 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 wenden Zeichenketten beispielsweise für eine formatierte Ausgabe von Daten an.
- Sie organisieren gleichartige Daten in Feldern, Matrizen sowie Listen.
- Sie wandeln Zeichenketten in Zahlenwerte und wenden mathematische Funktionen auf Zahlenwerte an.

## 2. Zeichenketten

---

## 2.1 Zeichenketten

- Zeichenketten in C
  - Variablen: Zeiger auf Array des primitiven Datentyps char
  - Speichergröße vom Programmierer verwaltet
  - Datentyp hat keine Methoden
- Zeichenketten in Java:
  - Zeichenketten sind Objekte der Klasse String.
  - Variablen referenzieren Objekte
  - Speichergröße vom Objekt verwaltet
  - Datentyp stellt Methoden zur Verfügung



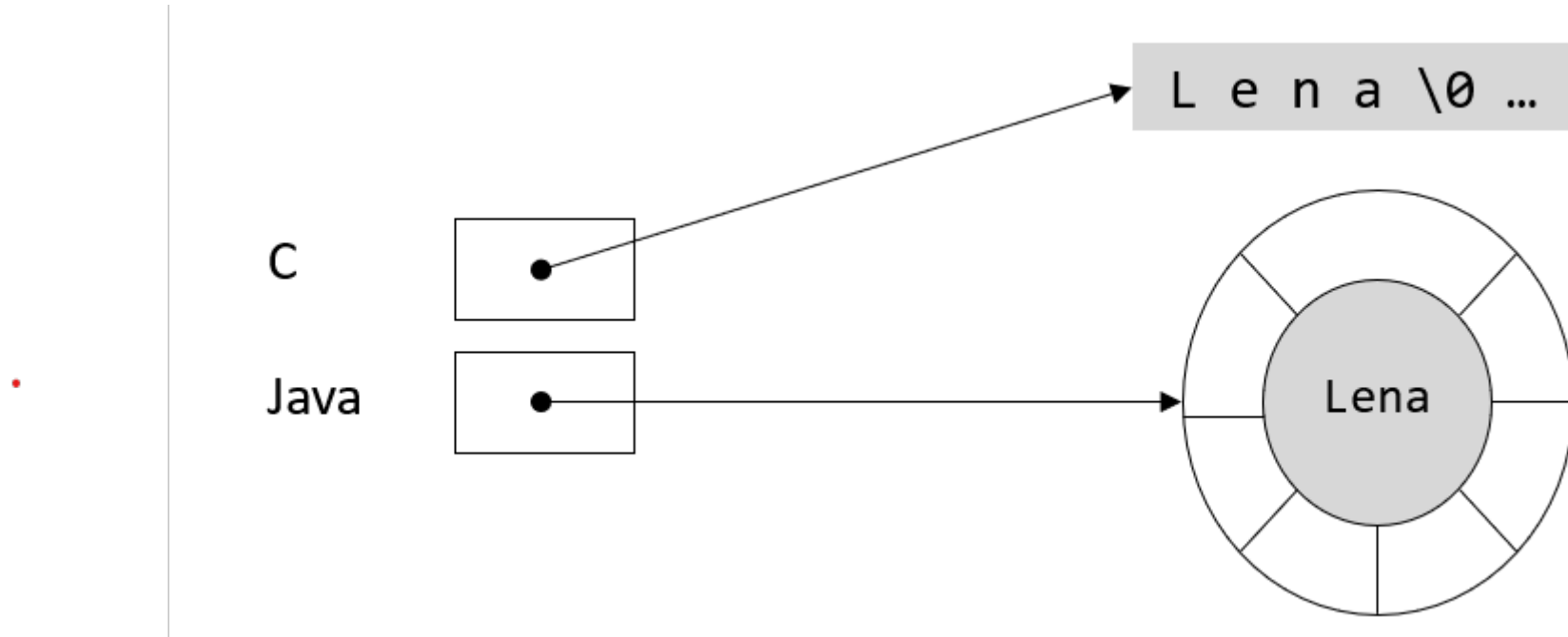


Abbildung 1: Zeichenketten: Unterschiede C und Java

## 2.2 Zeichenketten erzeugen

- Erzeugung erfolgt auch über den new-Operator:

```
1 String name = new String("Lena");
```



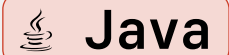
- Alternativ auch durch Zuweisung eines Literales:

```
1 String name = "Lena";
```



- Zuweisung eines Literales auch nach Erzeugung möglich:

```
1 String name = new String("Lena");  
2 name = "Birgit";
```



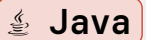
### ! Merke

- Wie in vielen objektorientierten Sprachen:
  - Objekte der Klasse String sind unveränderbar (immutable).
  - Ihr Wert kann nach Erzeugung nicht modifiziert werden.
  - Zum mehrstufigen Aufbau existiert die Klasse StringBuilder

### ? Frage

Welche Ausgabe erzeugt folgendes Programm?

```
1  public static void main(String[] args) {
2      String lena1 = new String("Lena");
3      String lena2 = lena1;
4
5      System.out.println("lena1: " + lena1 + "\nlena2: " + lena2);
6      System.out.println("Referenzen gleich: " + (lena1 == lena2));
7
8      lena2 += " B.";
9      System.out.println("\nlena1: " + lena1 + "\nlena2: " + lena2);
10     System.out.println("Referenzen gleich: " + (lena1 == lena2));
11 }
```



Java

## 2.3 Zeichenketten als unveränderbar

- Zur Veranschaulichung

```
1 String lena1 = new String("Lena");  
2 String lena2 = lena1;
```

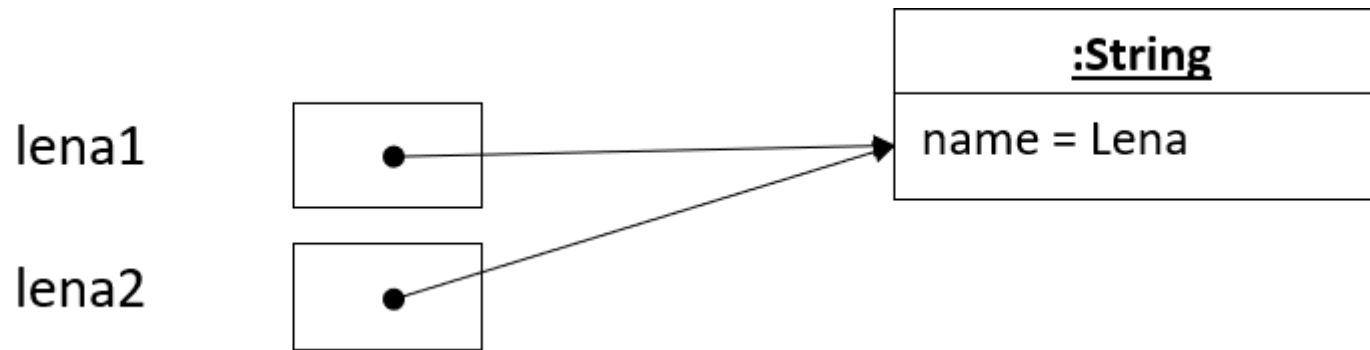


Abbildung 2: Referenz auf Zeichenkette

## 2.3 Zeichenketten als unveränderbar

```
1 lena2 += " B.";
```

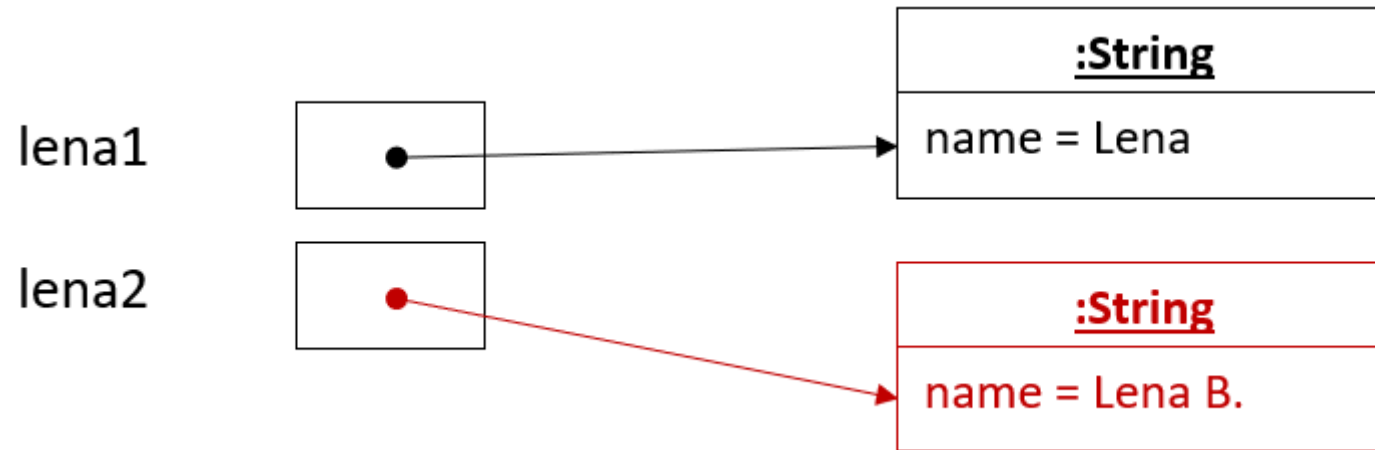
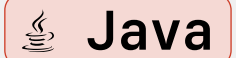
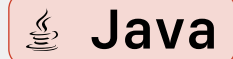


Abbildung 3: Änderung der Zeichenkette führt zu neuem Objekt

## 2.4 Verkettung von Zeichenketten

- Zeichenketten sind über den Plus-Operator verknüpfbar:

```
1 String name = "Lena " + "oder " + "dann ";  
2     name = name + "doch " + "wieder ";  
3     name += "Birgit?";
```



- Implizite Umwandlung anderer Datentypen in ein String-Objekt:
  - Auswertung der Plus-Operatoren von links nach rechts
  - Umwandlung in String, sofern der andere Operand nicht vom Typ String ist

### ? Frage

Was wird ausgegeben?

```
1 int a = 20;  
2 int b = 22;  
3 System.out.println("Jahr: " + a + b);  
4 System.out.println(a + b + " (Jahr)");
```





## 2.5 toString()-Methode

```
1 public String toString() {  
2     // Methodenrumpf  
3     // Rückgabe eines Objektes vom Typ String  
4 }
```



- Methodenkopf vorgeschrieben
- Methodenrumpf frei programmierbar
- Gibt für Objekte einen String zurück, der Objekt beschreiben sollte
- Wird bei impliziter Umwandlung eines Objektes in einen String aufgerufen

### Aufgabe 1

Lassen Sie uns das einmal ausprobieren!

- Erzeugen Sie eine Klasse Person und implementieren Sie die toString()-Methode
- Überprüfen Sie den impliziten Aufruf mittels der Konsolenausgabe.

## 2.5 toString()-Methode

```
1  public class Person {  
2      String firstName, surname;  
3  
4      public Person(String firstName, String surname) {  
5          this.firstName = firstName;  
6          this.surname = surname;  
7      }  
8  
9      public String toString() {  
10         return firstName + " " + surname;  
11     }  
}
```

## 2.5 toString()-Methode

## 2. Zeichenketten

```
12    }
```

## 2.5 toString()-Methode

```
1 Person lena = new Person("Lena", "Jensen");  
2 String name = "Name: " + lena;  
3 System.out.println(lena);  
4 System.out.println(name);
```

## 2.6 String-Methoden

- Weitere Methoden beinhalten z.B.:
  - Länge der Zeichenkette
  - Zeichen an bestimmten Position (Erstes Zeichen hat Index 0!)
  - Bestimmtes Zeichen ersetzen
  - Bestimmtes Zeichen oder Teilzeichenkette suchen
  - Zeichenkette teilen
  - Umwandlung in Kleinbuchstaben oder Großbuchstaben
  - Vergleich zweier Zeichenketten
  - Und noch einige Weitere!

### ☰ Aufgabe 2

- Ersetzen Sie „Humbug“ durch „Hamburg“.

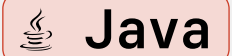
```
1 String hamburg = "Willkommen in Humbug!";  
2 hamburg = hamburg.replace("Humbug", "Hamburg");  
3 System.out.println(hamburg);
```

**Java**

### ? Frage

Was wird ausgegeben?

```
1 String upper = "Willkommen in Hamburg!";  
2 String lower = "willkommen in hamburg!";  
3  
4 System.out.println(lower.equals(upper));  
5 System.out.println(lower.equals(upper.toLowerCase()));  
6 System.out.println(lower.equalsIgnoreCase(upper));
```





## 2.7 String-Formatierung

- Gerne gefragt:
  - Kann man auch das Format des Strings bei der Ausgabe anpassen?  
**Ja, natürlich!**
- Klassenmethode `format()`:
  - Erzeugt eine formatierte Zeichenkette
  - Es erfolgt keine Ausgabe auf Konsole.
  - Syntax (fast) identisch mit `printf()` aus C/C++

### ? Frage

Was wird ausgegeben?

```
1  double wind = 21.4532;  
2  String weather = String.format("%s %d: %.1f km/h",  
    "Station", 7, wind);  
3  System.out.println(weather);
```



### ? Frage

Was wird ausgegeben?

```
1  double wind = 21.4532;  
2  String weather = String.format("%s %d: %.1f km/h",  
    "Station", 7, wind);  
3  System.out.println(weather);
```



- **Ausgabe:** Station 7: 21,5 km/h

## 2.7 String-Formatierung

- Formatangaben:

**%**[ArgumentNr.] [Flags] [MindestanzahlZeichen] [Genauigkeit]

Format

Format	Bedeutung
f, e, g	Fließkommazahl ( <u>float</u> )
d	Ganzzahl ( <u>decimal</u> )
o	Oktale Ganzzahl ( <u>octal</u> )
x, X	Hexadezimale Ganzzahl
s	Zeichenkette ( <u>string</u> )
t	Datum und Zeit ( <u>time</u> )
b	Wahrheitswert ( <u>boolean</u> )

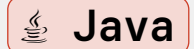
Flag	Bedeutung
-	Linksbündig
+	Vorzeichen immer ausgeben
0	Zahlen links mit 0 auffüllen
,	Zahlen mit Tausenderpunkten
(	Negative Zahlen in Klammern

Abbildung 4: Formate und Flags

### ? Frage

Was wird ausgegeben?

```
1 double wind = 21.4532;  
2 System.out.println(String.format("%2.2f km/h", wind));  
3 System.out.println(String.format("%8.2f km/h", wind));  
4 System.out.println(String.format("%08.2f km/h", wind));
```

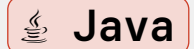




### ? Frage

Was wird ausgegeben?

```
1 double wind = 21.4532;  
2 System.out.println(String.format("%2.2f km/h", wind));  
3 System.out.println(String.format("%8.2f km/h", wind));  
4 System.out.println(String.format("%08.2f km/h", wind));
```



- Ausgabe:
  - 21,45 km/h
  - 21,45 km/h

## 2.7 String-Formatierung

- 00021,45 km/h



### ! Merke

- Mindestanzahl der Zeichen:
  - Schließt Nachkommastellen, sowie das Komma ein
  - Schneidet keine Vorkommastellen ab

### ? Frage

- Was fällt auf?

```
1 double wind = 21.4532;
```

**Java**

```
2 System.out.println(String.format("%2.2f km/h", wind));
```

- Ausgabe: 21,45 km/h
- 
-

### ? Frage

- Was fällt auf?

```
1 double wind = 21.4532;
```




```
2 System.out.println(String.format("%2.2f km/h", wind));
```

- Ausgabe: 21,45 km/h
- Oben bei Ausgabe „deutsches Nachkomma“ statt „englischer Punkt“
- Durch Lokalisierung vorgegeben



### Beispiel

```
1 double wind = 21.4532;
```

 **Java**

```
2 System.out.println(String.format(Locale.US, "%2.2f  
km/h", wind));
```

```
3 System.out.println(String.format(Locale.GERMAN,  
"%2.2f km/h", wind));
```

- Ausgabe: 21.45 km/h 21,45 km/h

## 3. Arrays

---

## 3.1 Arrays (Felder)

- Arrays in C:
  - Variablen: Zeiger auf erstes Element des Arrays im Speicher
  - Speichergröße vom Programmierer verwaltet
  - Datentyp hat keine Methoden
- Arrays in Java:
  - Arrays sind Objekte.
  - Variablen referenzieren Objekte
  - Speichergröße vom Objekt verwaltet
  - Datentyp stellt Methoden zur Verfügung

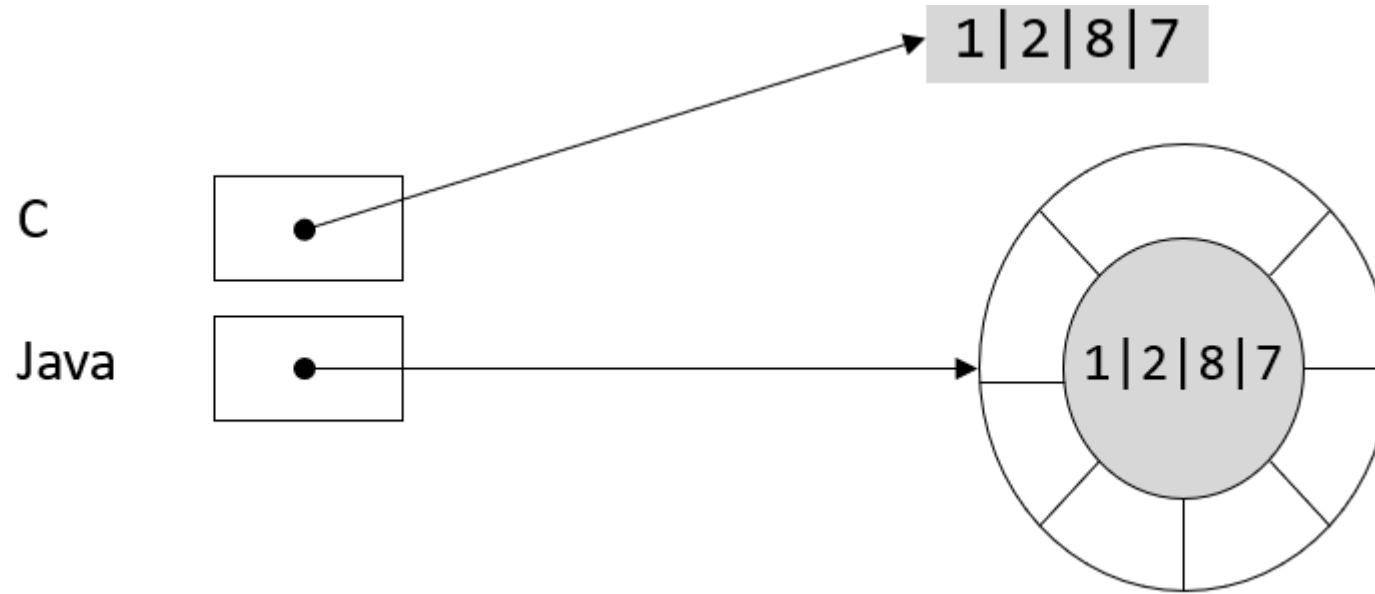


Abbildung 1: Arrays in Java und C

## 3.2 Erzeugung von Arrays

- Sammlung von Elementen mit gleichem Datentyp
- Datentyp wird durch eckige Klammern zum Array (z.B. `int[]`, `String[]`)
- Array-Klassen sind eigene (weitere) Datentypen
- Deklaration:
  - Erfordert keine Angabe der Länge
  - Variable kann Arrays beliebiger Länge referenzieren
  - Deklaration erzeugt kein Objekt, sondern Referenzvariable

```
1 int[] filter;
```

**Java**



### ! Merke

- Klammern hinter Variablennamen zulässig, aber nicht empfohlen (Warum?)

```
1 int filter[];
```



## 3.3 Erzeugung: Dynamische Deklaration

- Array-Objekt mittels new-Operator erzeugen
- Anzahl der Felder in eckigen Klammern
- Beachte: Keine runden „Konstruktor-Klammern“ hinter Datentyp
- Werte im Array werden mit 0, 0.0, false bzw. null initialisiert

```
1 int[] filter = new int[];
```

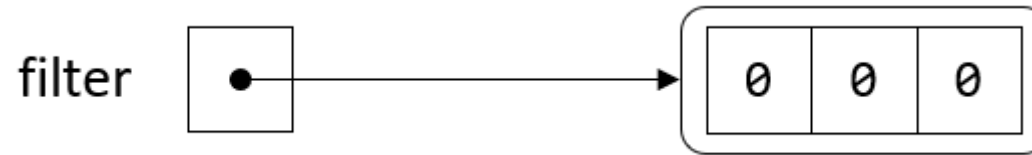
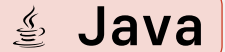


Abbildung 2: Erstellung eines Arrays

## 3.4 Erzeugung: Zuweisen von Elementen

- Zugriff auf Array-Element über Index in eckigen Klammern
- Erstes Element besitzt Index 0

```
1 int[] filter = new int[3];  
2 filter[0] = 1;  
3 filter[1] = 2;  
4 filter[2] = 1;
```



Abbildung 3: Zuweisen von Werten durch Indexzugriff

## 3.5 Erzeugung: Statische Deklaration

- Man kann einem Array bereits bei Erzeugung des Objektes die Werte zuweisen.
- Werte in geschweiften Klammern und durch Kommas getrennt
- Mit und ohne Verwendung des new-Operators zulässig

```
1 int[] filter = {1, 2, 1};
```



```
2 int[] filter = new int[] {1, 2, 1};
```

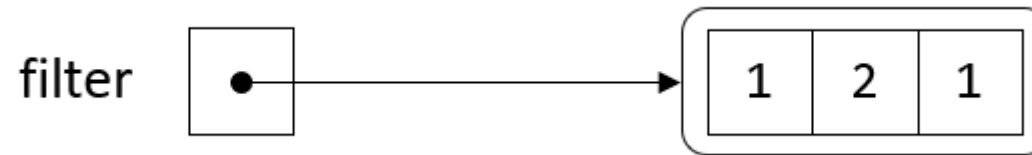


Abbildung 4: Füllen bei Deklaration

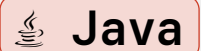
## 3.6 Eigenschaften: Array-Klassen

- Arrays sind Objekte der entsprechenden Klasse:
  - Arrays besitzen Methoden.
  - Anzahl der Elemente über Instanzvariable `length`

### ? Frage

Welches Array wird durch den Code erzeugt?

```
1 int[] filter = new int[3];
2 for (int i = 0; i < filter.length; i++) {
3     filter[i] = i * i;
4 }
```



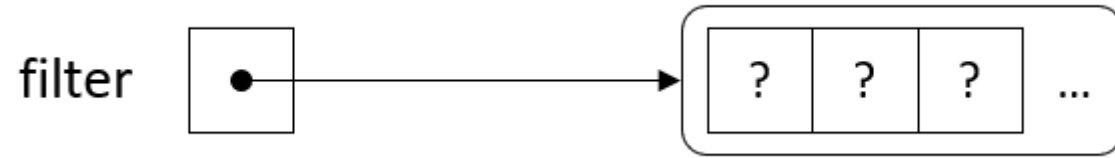


Abbildung 5: Befüllen durch for-Schleife

## 3.6 Eigenschaften: Array-Klassen

- Indizes:
  - Bei Zugriff auf Element überprüft, ob Index im erlaubten Bereich liegt
  - Mehr im Kapitel über Ausnahmen und Fehlerbehandlung



### Beispiel

Beispiele erlaubter und nicht erlaubter Indizes:

```
1 int[] filter = new int[3];
```



Java

```
2 filter[0] = -1;
```

```
3 filter[2] = 4;
```

```
4 filter[-1] = 1;
```

```
5 filter[3] = 2;
```





Abbildung 6: Indizes des Arrays filter

## 3.6 Eigenschaften: Array-Klassen

- Arrays für beliebige Datentypen (auch eigene Klassen) deklarierbar
- Objekte müssen vom gleichen Typ sein (oder Subtyp, hierzu mehr bei Vererbung)
- Nicht die Objekte gespeichert, sondern Referenzen zu den Objekten

```
1 Person[] friends = new Person[3];  
2 friends[0] = new Person("Lena");  
3 friends[1] = new Person("Birgit");  
4 friends[2] = new Person("Jan");
```



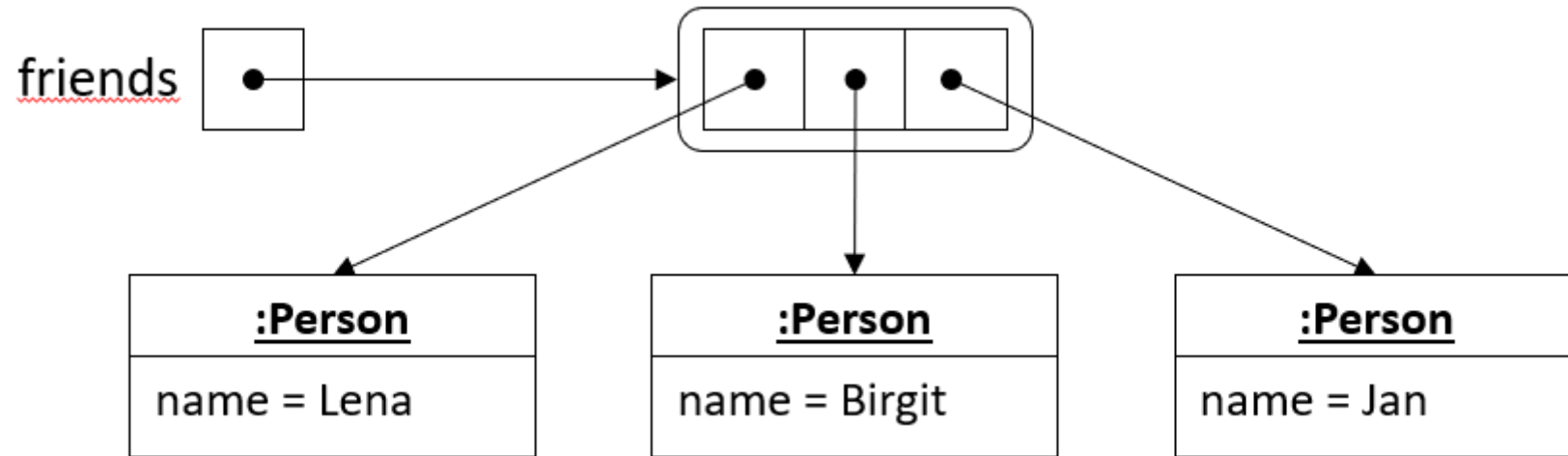


Abbildung 7: Arrays aus Objekten

### ? Frage

Was wird ausgegeben?

```
1 int[] a = {1, 2, 3, 4, 5};  
2 int[] b = a;  
3 b[3] = 0;  
4 System.out.println(b[2]);  
5 System.out.println(a[3]);
```



### ? Frage

Was wird ausgegeben?

```
1 int[] a = {1, 2, 3, 4, 5};  
2 int[] b = {1, 2, 3, 4, 5};  
3 System.out.println(a == b);
```

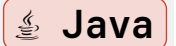


Java

### ? Frage

Was wird ausgegeben?

```
1  public class ArrayDemo {
2      static int[] createSortedArray(int a, int b) {
3          if (a < b) {
4              return new int[] {a, b};
5          } else {
6              return new int[] {b, a};
7          }
8      }
9  }
```



```
10      public static void main(String[] args) {  
11          System.out.println( createSortedArray(7, 4)[1] );  
12      }  
13  }
```

### ☰ Aufgabe 3

- Schreiben Sie eine Methode, die die Elemente eines `int[]`-Arrays aufsteigend sortiert.
- Testen Sie die Methode anhand des Arrays `{10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 7}`.



```
1  public static void sort(int[] a) {
2      int i = 0;
3      while (i <= a.length - 2) {
4          if (a[i] > a[i+1]) {
5              // Swap elements and shift index to left element
6              int temp = a[i];
7              a[i] = a[i+1];
8              a[i+1] = temp;
9              if (i > 0)
10                 i--;
11          } else {
12              i++;
13          }
14      }
15 }
```



## 4. Mehrdimensionale Arrays

---

- Mehrdimensionale Arrays sind „Felder von Feldern“.
- Beispiel: `int[][]` ist Array, dessen Elemente vom Datentyp `int[]` sind.

Dynamische Deklaration:

```
1  int[][] filter = new int[3][4];
```



Statische Deklaration:

```
1  int[][] filter = {{1,2,3}, {4,5,6}, {7,8,9}};
```



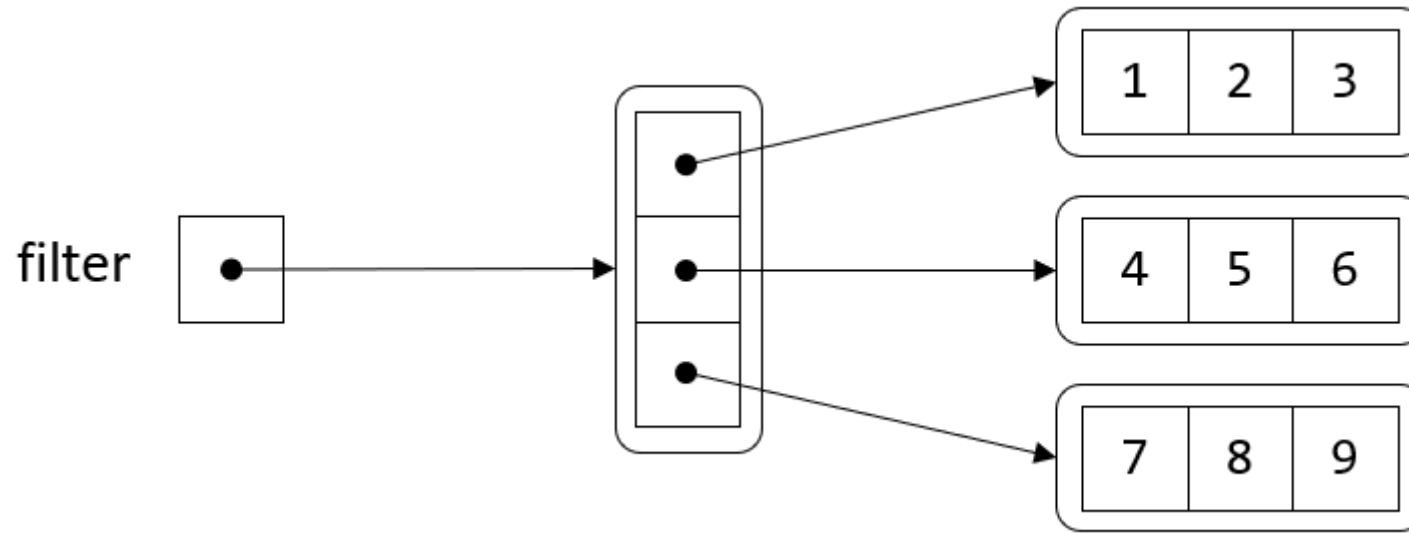


Abbildung 8: Mehrdimensionales Array

### ? Frage

Was wird ausgegeben?

```
1  int[][] a = {{1,2}, {3,4}, {5,6}};
```



```
2
```

```
3  System.out.println(a.length);
```

```
4  System.out.println(a[2].length);
```

```
5
```

```
6  System.out.println(a[1][1]);
```

```
7  System.out.println(a[2][0]);
```

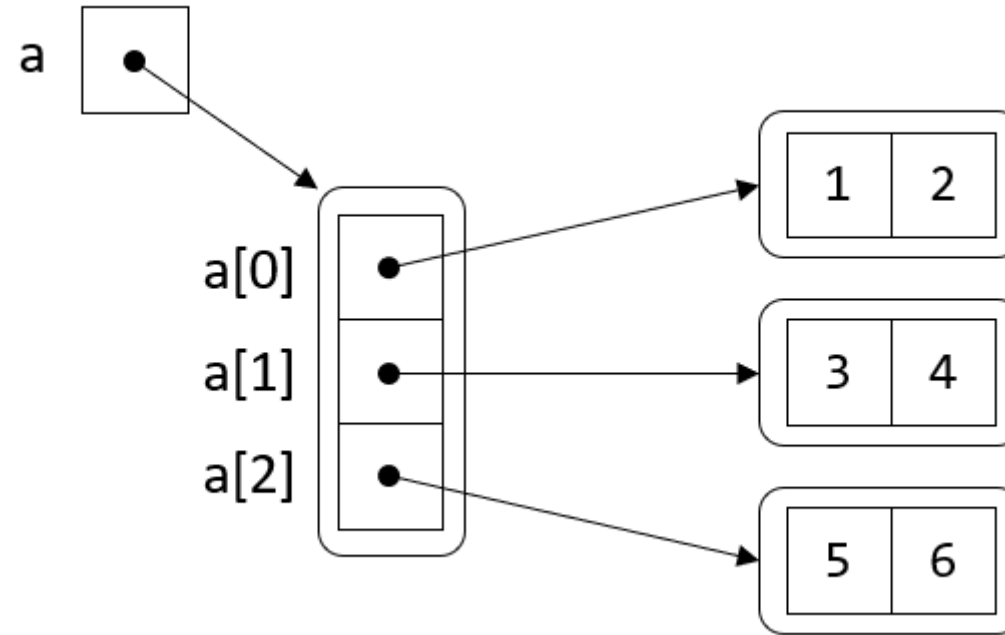


Abbildung 9: Mehrdimensionale Arrays mit Werten

### ? Frage

Was wird ausgegeben?

```
1 int[][] a = {{1,2}, {3,4}, {5,6}};  
2 int[] b = a[0];  
3 int c = b[1];  
4  
5 b[1] = 7;  
6 System.out.println(a[0][1]);  
7 System.out.println(c);
```



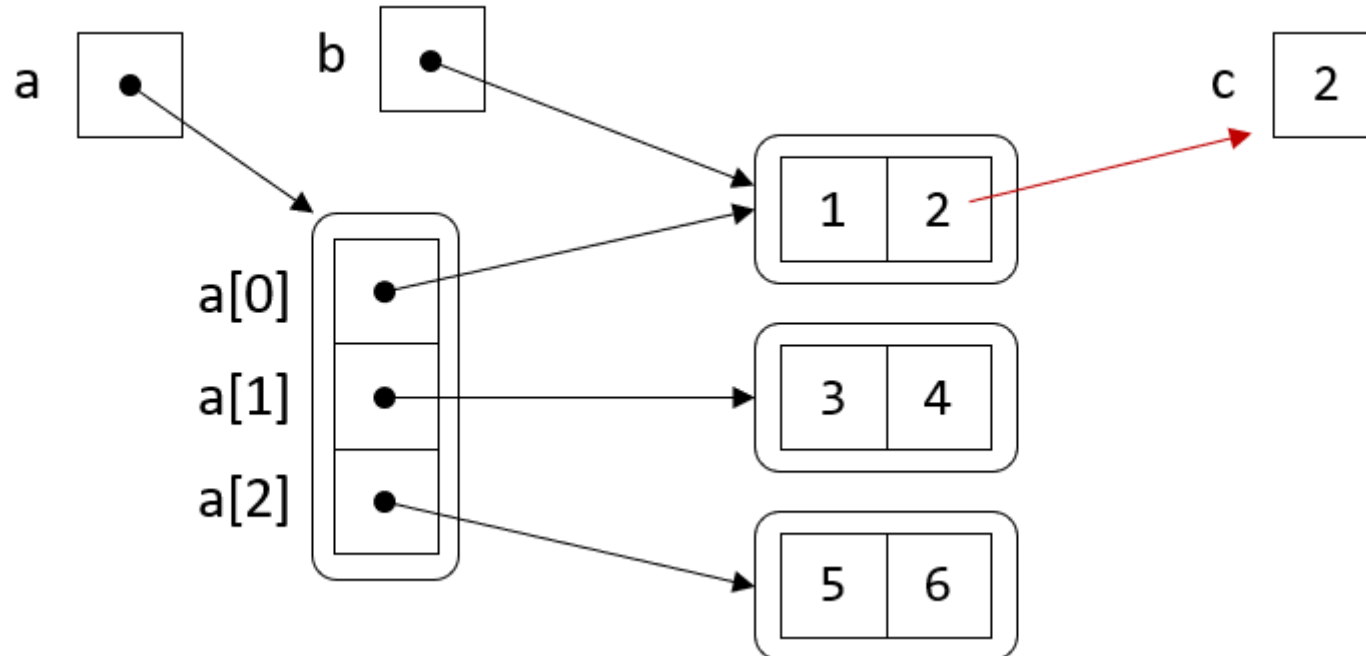


Abbildung 10: Komplizierte mehrdimensionale Arrays



- Mehrdimensionale Arrays müssen nicht rechteckig sein
- Beispiel: Jeder Zeile eines zweidimensionalen Arrays eigenes Array zuweisen

### ☰ Aufgabe 4

Erstellen Sie eine Dreiecksmatrix mittels einer for-Schleife!

## 4.1 Mehrdimensionale Arrays

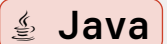
## 4. Mehrdimensionale Arrays

- Mehrdimensionale Arrays müssen nicht rechteckig sein
- Beispiel: Jeder Zeile eines zweidimensionalen Arrays eigenes Array zuweisen

### ☰ Aufgabe 5

Erstellen Sie eine Dreiecksmatrix mittels einer for-Schleife!

```
1 int[][] a = new int[3][];  
2 for (int i = 0; i < a.length; i++) {  
3     a[i] = new int[i + 1];  
4 }
```



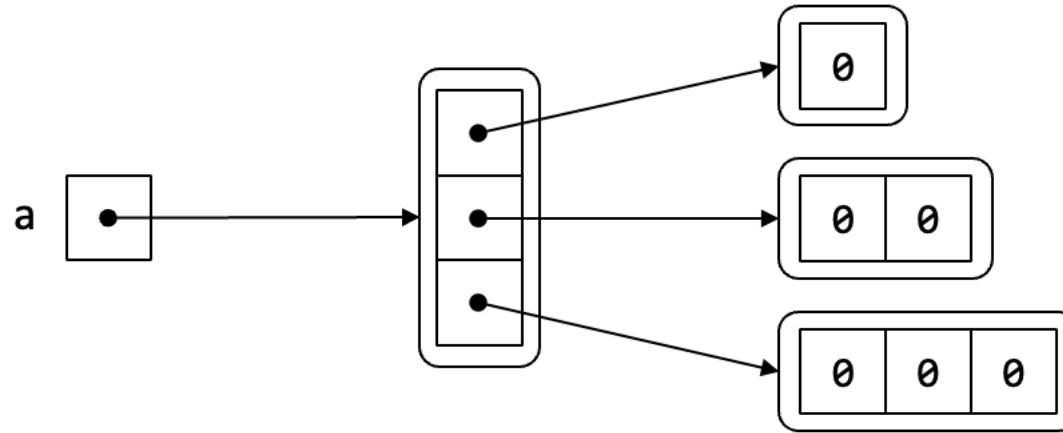


Abbildung 11: Mehrdimensionales Array in der Form eines Dreiecks

## 5. Listen

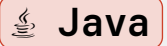
---

## 5.1 ArrayList

- Arrays: Größe nach Erzeugung nicht mehr änderbar („semidynamisch“)
- Listen: Elemente können hinzugefügt oder entfernt werden („dynamisch“)
  - Datentyp zu speichernder Elemente in spitzen Klammern (siehe unten: String)

# 5.1 ArrayList

```
1 public class ArrayListDemo {
2     public static void main(String[] args) {
3         ArrayList<String> names = new ArrayList<String>();
4         names.add("Lena");
5         names.add("Birgit");
6         names.add("Jan");
7         names.add(new String("Jan"));
8     }
9 }
```

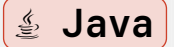


## 5.1 ArrayList

- Beispiele:
  - Anzahl der Elemente (`size()`)
  - Zugriff auf Elemente (`get()`)
  - Abfrage, ob bestimmtes Element in Liste ist (`contains()`)
  - Element aus Liste entfernen (`remove()`)

# 5.1 ArrayList

```
1  ArrayList<String> names = new ArrayList<String>();
2  String birgit = "Birgit";
3  names.add("Lena");
4  names.add(birgit);
5
6  for (int i = 0; i < names.size(); i++) {
7      System.out.println(names.get(i));
8  }
9
10 if (names.contains(birgit)) {
11     names.remove(birgit);
12 }
```

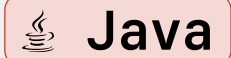




## 6. foreach-Schleife

---

```
1  for (Datentyp Variable : Iterationsobjekt) {  
2      Anweisungen  
3  }
```



- Motivation:
  - Mitunter jedes Element z.B. eines Arrays oder einer Liste benötigt
  - Aber: Position innerhalb des Arrays oder der Liste wird nicht benötigt
  - Daher auch kein Schleifenzähler als Index benötigt

- Schleife iteriert vom ersten bis zum letzten Element durch Array (oder Liste):
- Beim ersten Durchlauf hat Variable den Wert des 1. Elements
- Beim zweiten Durchlauf hat Variable den Wert des 2. Elements und so weiter
- Beim letzten Durchlauf hat Variable den Wert des letzten Elements

### ? Frage

Was wird ausgegeben?

```
1 int[] a = {7, 1, 3, 8};
```



Java

```
2
```

```
3 for (int element : a) {
```

```
4     System.out.println("Element: " + element);
```

```
5 }
```

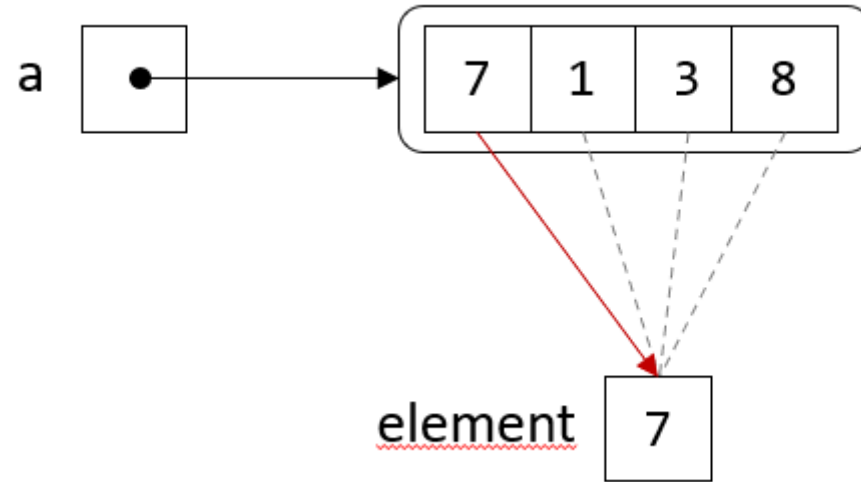
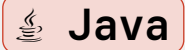


Abbildung 12: Ergebnis der foreach-Schleife

### ☰ Aufgabe 6

- Erstellen Sie Folgendes unter Verwendung einer foreach-Schleife:
- Methode, die den Mittelwert der in einem Array enthaltenen Zahlen zurückgibt
- Programm, das die Methode verwendet

```
1  static double average(double[] numbers) {
2      double sum = 0.0;
3
4      for(double number : numbers) {
5          sum += number;
6      }
7      return sum / numbers.length;
8  }
9
10 public static void main(String[] args) {
11     double[] a = {1.43, 2, .2, 6.32, 7.1, 8.1};
12     System.out.println("Average = " + average(a));
13 }
```



## 7. Wrapperklassen & Math-Klasse

---



- Primitive Datentypen:
  - Speichern Wert (z.B. Ganzzahl) direkt
  - Besitzen keine Methoden
- Wrapperklassen:
  - „Packen“ (to wrap) primitive Datentypen in Klassen ein
  - Stellen Methoden (z.B. für Ganzzahlen) zur Verfügung

# 7.1 Wrapperklassen

## 7. Wrapperklassen & Math-Klasse

Primitiver Datentyp	Zugehörige Wrapperklasse
<u>boolean</u>	Boolean
byte	Byte
<u>short</u>	Short
<u>int</u>	<i>Integer</i>
<u>long</u>	Long
<u>char</u>	<u><i>Character</i></u>
<u>float</u>	<u>Float</u>
double	Double

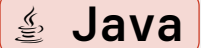
Abbildung 13: Wrapperklassen für primitive Datentypen

# 7.1 Wrapperklassen

## 7. Wrapperklassen & Math-Klasse

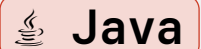
- Primitiven Datentypen in String umwandeln

```
1  int a = 7;  
2  Integer b = new Integer(a);  
3  String c = b.toString();
```



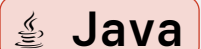
- Kürzere Alternative über Klassenmethode:

```
1  String a = Integer.toString(7);
```



- String in primitiven Datentypen umwandeln:

```
1  String a = "7";  
2  int b = Integer.parseInt(a);
```

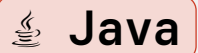


# 7.1 Wrapperklassen

## 7. Wrapperklassen & Math-Klasse

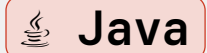
- Umwandlungen:
  - Boxing: Umwandlung primitiver Datentyp in Objekt einer Wrapperklasse
  - Unboxing: Umwandlung Objekt einer Wrapperklasse in primitiven Datentyp

```
1 Integer object = new Integer(24); //Boxing of int value
2 int noObject = object.intValue(); //Unboxing of object
```



- Autoboxing: Automatische Umwandlungen (beide Richtungen)

```
1 Integer object = 24; //Automatic boxing of int value
2 int noObject = object; //Automatic unboxing of object
```



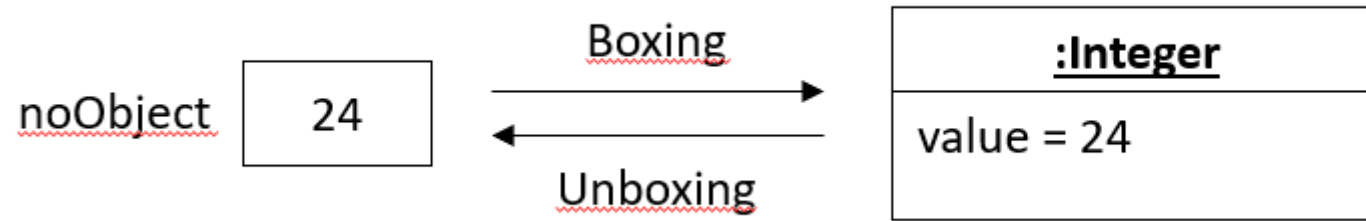


Abbildung 14: Typumwandlung mit Wrapperklassen

- Mathematische Konstanten: Eulerzahl  $e$ , Kreiszahl  $\pi$
- Mathematische Funktionen (als Klassenmethoden), z.B.:
  - Trigonometrische Funktionen
  - Rundung
  - Betrag
  - Exponentialfunktion und Logarithmus
  - Maximum und Minimum
  - Wurzeln
  - Zufallszahlen



### Beispiel

```
1 double angleDeg = 127.5;  
2 double angleRad = Math.toRadians(angleDeg);  
3 System.out.printf("cos(%.2f) = %.2f\n", angleRad,  
  Math.cos(angleRad));
```

**Java**

## 8. License Notice

---



## 8.1 Attribution

- This work is shared under the CC BY-NC-SA 4.0 License and the respective Public License
- <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.