

# Object-Oriented Programming in Java

## Lecture 4 - Class Libraries

Emily Lucia Antosch

HAW Hamburg

13.08.2025

# Contents

1. Introduction .....	2
2. Strings .....	6
3. Arrays .....	30
4. Multidimensional Arrays .....	51
5. Lists .....	60
6. foreach Loop .....	65
7. Wrapper Classes & Math Class .....	72
8. License Notice .....	80

# 1. Introduction

---

# 1.1 Where Are We Currently?

- Last time we dealt with classes and objects.
- You can now
  - ▶ write simple classes in Java,
  - ▶ create objects from classes, use attributes and call methods and
  - ▶ use class variables and class methods.
- Today we continue with **Class Libraries**.

# 1.1 Where Are We Currently?

1. Imperative Concepts
2. Classes and Objects
3. **Class Library**
4. Inheritance
5. Interfaces
6. Graphical User Interfaces
7. Exception Handling
8. Input and Output
9. Multithreading (Parallel Computing)

## 1.2 The Goal of This Chapter

- You apply strings, for example, for formatted output of data.
- You organize similar data in fields, matrices and lists.
- You convert strings to numerical values and apply mathematical functions to numerical values.

## 2. Strings

---

## 2.1 Strings

- Strings in C
  - ▶ Variables: Pointer to array of primitive data type `char`
  - ▶ Memory size managed by programmer
  - ▶ Data type has no methods
- Strings in Java:
  - ▶ Strings are objects of class `String`.
  - ▶ Variables reference objects
  - ▶ Memory size managed by object
  - ▶ Data type provides methods



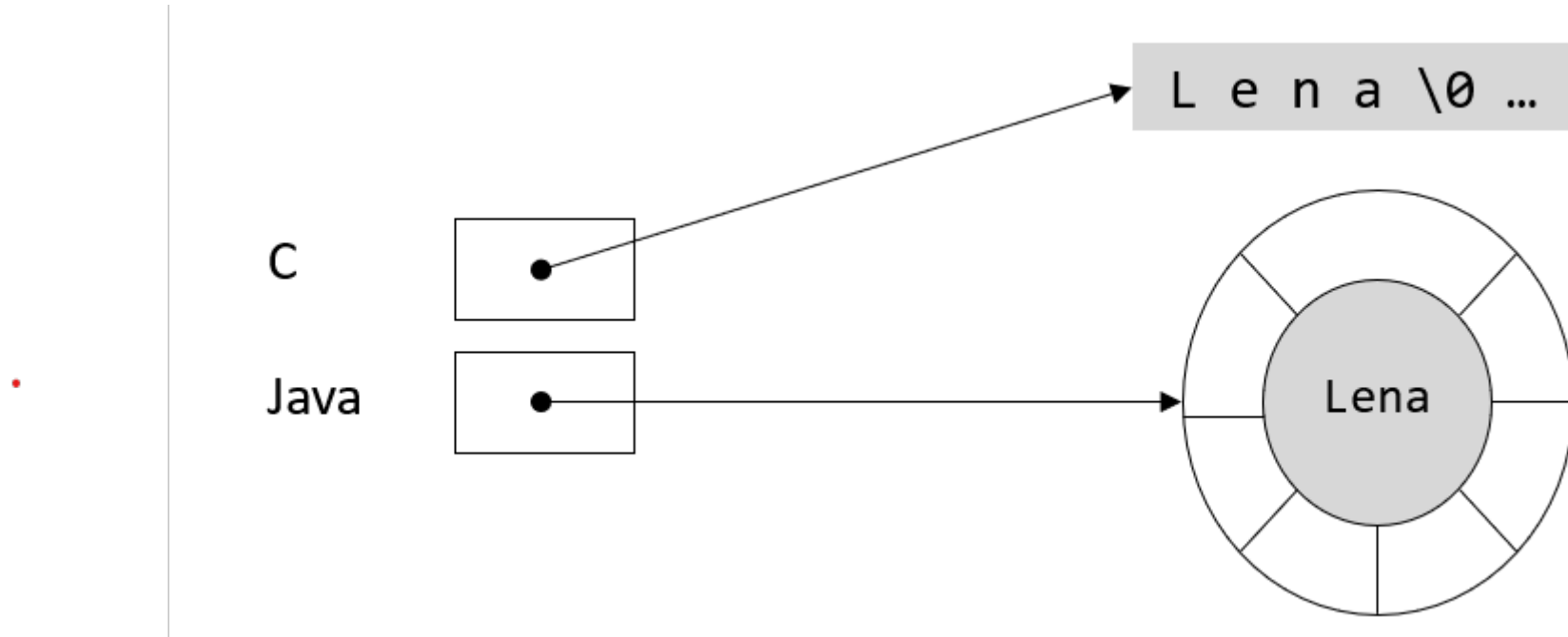
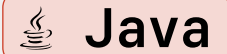


Figure 1: Strings: Differences between C and Java

## 2.2 Creating Strings

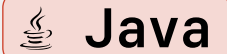
- Creation also using the `new` operator:

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



- Alternatively by assigning a literal:

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



- Assignment of a literal also possible after creation:

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



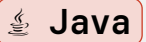
### ! Memorize

- As in many object-oriented languages:
  - ▶ Objects of class `String` are immutable.
  - ▶ Their value cannot be modified after creation.
  - ▶ For multi-stage construction, the `StringBuilder` class exists

### ? Question

What output does the following program produce?

```
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("References equal: " + (lena1 == lena2));
7
8      lena2 += " B.";
9      System.out.println("\nlena1: " + lena1 + "\nlena2: " + lena2);
10     System.out.println("References equal: " + (lena1 == lena2));
11 }
```



## 2.3 Strings as Immutable

- For illustration

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

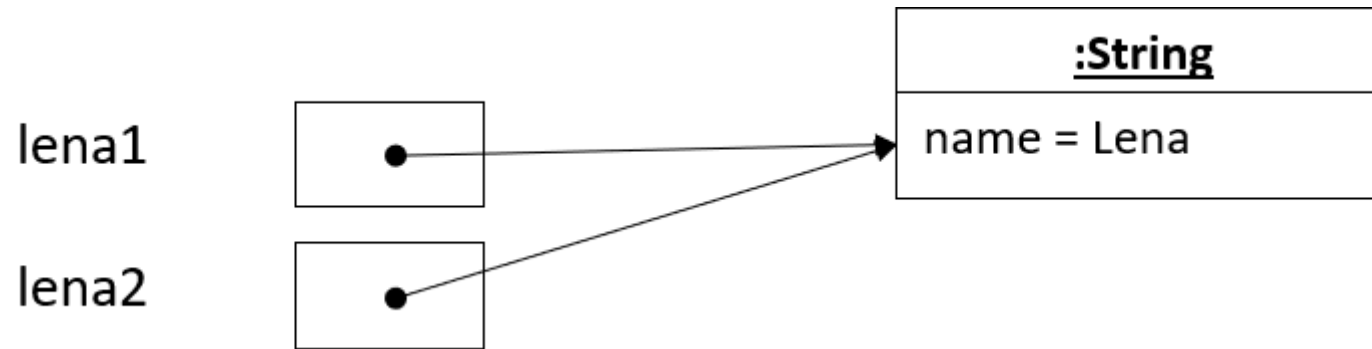
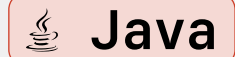


Figure 2: Reference to string

## 2.3 Strings as Immutable

## 2. Strings

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

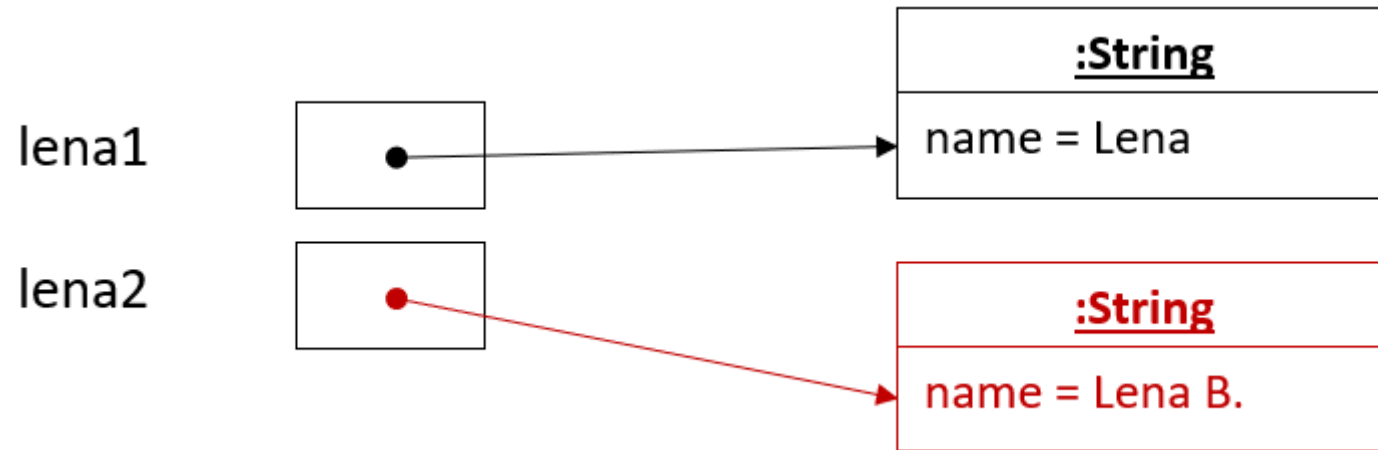
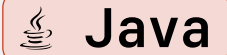


Figure 3: Changing the string leads to new object

## 2.4 String Concatenation

- Strings can be concatenated using the plus operator:

```
1 String name = "Lena " + "or " + "then ";  
2     name = name + "after all " + "again ";  
3     name += "Birgit?";
```

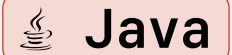


- Implicit conversion of other data types to a String object:
  - ▶ Evaluation of plus operators from left to right
  - ▶ Conversion to String if the other operand is not of type String

### ? Question

What will be output?

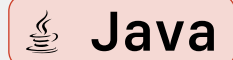
```
1 int a = 20;  
2 int b = 22;  
3 System.out.println("Year: " + a + b);  
4 System.out.println(a + b + " (Year)");
```





## 2.5 toString() Method

```
1 public String toString() {  
2     // Method body  
3     // Return of an object of type String  
4 }
```



- Method header prescribed
- Method body freely programmable
- Returns a String for objects that should describe the object
- Is called during implicit conversion of an object to a String

### Task 1

Let's try this out!

- Create a Person class and implement the toString() method
- 
- Check the implicit call using console output.

## 2.5 toString() Method

```
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     }  
12 }
```

## 2.5 toString() Method

```
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 Methods

- Additional methods include, for example:
  - ▶ Length of the string
  - ▶ Character at specific position (First character has index 0!)
  - ▶ Replace specific character
  - ▶ Search for specific character or substring
  - ▶ Split string
  - ▶ Conversion to lowercase or uppercase
  - ▶ Comparison of two strings
  - ▶ And several more!

### Task 2

- Replace “Humbug” with “Hamburg”.

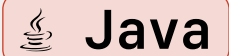
```
1 String hamburg = "Welcome to Humbug!";  
2 hamburg = hamburg.replace("Humbug", "Hamburg");  
3 System.out.println(hamburg);
```

**Java**

### ? Question

What will be output?

```
1 String upper = "Welcome to Hamburg!";
2 String lower = "welcome to 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 Formatting

- Often asked:
  - ▶ Can you also adjust the format of the string during output?  
**Yes, of course!**
- Class method `format()`:
  - ▶ Creates a formatted string
  - ▶ No output to console occurs.
  - ▶ Syntax (almost) identical to `printf()` from C/C++



### ? Question

What will be output?

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



Java

### ? Question

What will be output?

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



Java

- **Output:** Station 7: 21.5 km/h

## 2.7 String Formatting

- Format specifications:

**%**[ArgumentNo.] [Flags] [MinimumNumberCharacters] [.Precision]  
Format

Format	Meaning
f, e, g	float
d	Integer
o	Octal
x, X	Hexadecimal
s	String
t	time
b	boolean

Flag	Meaning
-	Left Alignment
+	Print Prefix
0	Fill empty space with 0
,	Add commas for thousands
(	Add parentheses for negative numbers

Table 1: Formats and Flags

### ? Question

What will be the output?

```
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));
```

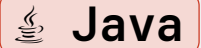


Java

### ? Question

What will be the output?

```
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));
```



- Output:
  - ▶ 21.45 km/h
  - ▶ 21.45 km/h
  - ▶ 00021.45 km/h

### ! Memorize

- Minimum number of characters:
  - ▶ Includes decimal places as well as the comma
  - ▶ Does not cut off any digits before the decimal point

### ? Question

- What do you notice?

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

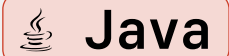
**Java**

- Output: 21.45 km/h
- 
-

### ? Question

- What do you notice?

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




- Output: 21.45 km/h
- Above in output “German decimal comma” instead of “English point”
- Specified by localization





### Example

```
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));
```

- Output: 21.45 km/h 21.45 km/h

## 3. Arrays

---

- Arrays in C:
  - ▶ Variables: Pointer to first element of the array in memory
  - ▶ Memory size managed by programmer
  - ▶ Data type has no methods
- Arrays in Java:
  - ▶ Arrays are objects.
  - ▶ Variables reference objects
  - ▶ Memory size managed by object
  - ▶ Data type provides methods

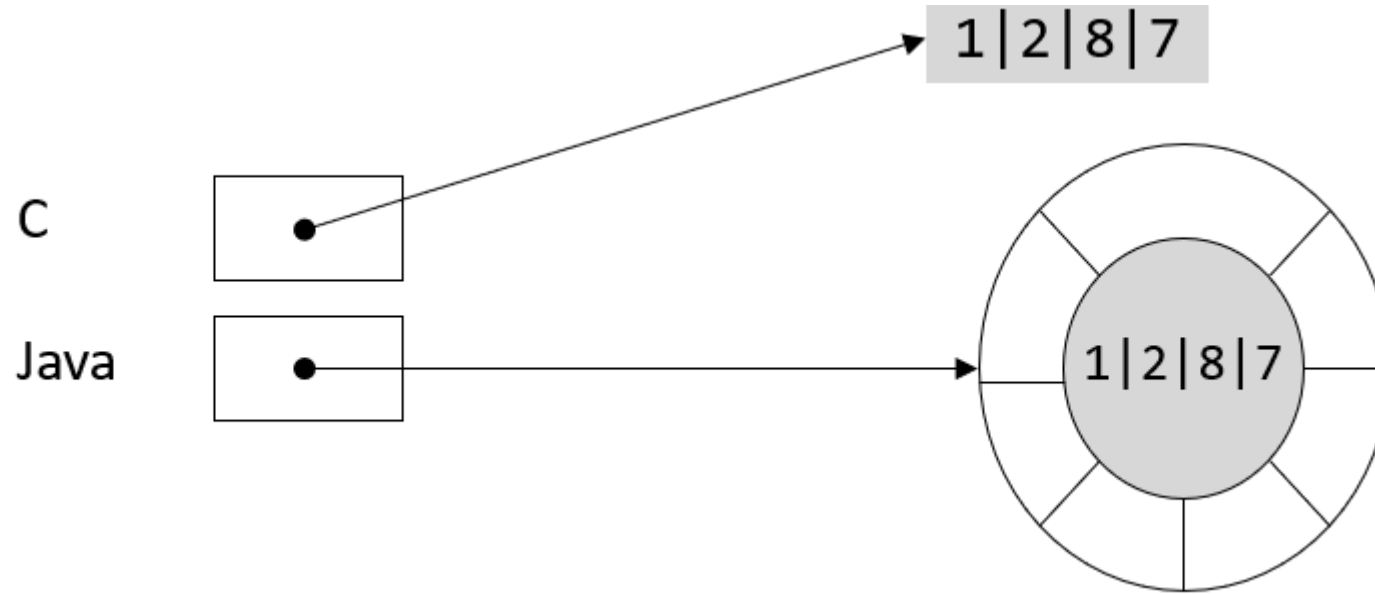


Figure 1: Arrays in Java and C

## 3.2 Creating Arrays

- Collection of elements with the same data type
- Data type becomes array through square brackets (e.g. `int[]`, `String[]`)
- Array classes are separate (additional) data types
- Declaration:
  - ▶ Does not require specification of length
  - ▶ Variable can reference arrays of any length
  - ▶ Declaration does not create object, but reference variable

## 3.2 Creating Arrays

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



Java

### ! Memorize

- Brackets after variable names allowed, but not recommended (Why?)

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



Java

## 3.3 Creation: Dynamic Declaration

- Create array object using new operator
- Number of fields in square brackets
- Note: No round “constructor brackets” after data type
- Values in array are initialized with 0, 0.0, false or null

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

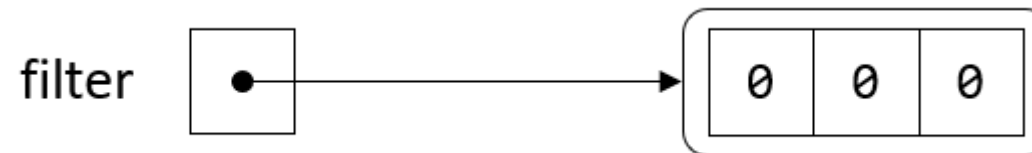
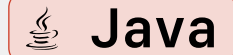


Figure 2: Creation of an array

## 3.4 Creation: Assigning Elements

- Access to array element via index in square brackets
- First element has index 0

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

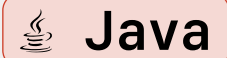


Figure 3: Assigning values through index access



## 3.5 Creation: Static Declaration

- You can assign values to an array already when creating the object.
- Values in curly braces and separated by commas
- Allowed with and without use of the new operator

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

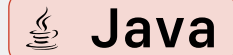


Figure 4: Filling during declaration

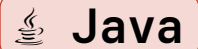
## 3.6 Properties: Array Classes

- Arrays are objects of the corresponding class:
  - ▶ Arrays have methods.
  - ▶ Number of elements via instance variable length

### ? Question

Which array is created by the code?

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



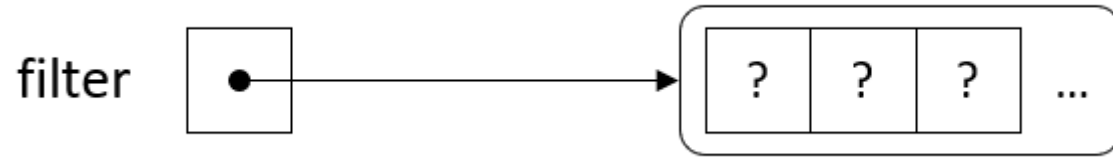


Figure 5: Filling through for loop

- Indices:
  - ▶ When accessing element, checks whether index is in allowed range
  - ▶ More in chapter on exceptions and error handling



### Example

Examples of allowed and disallowed indices:

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

 **Java**

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

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

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

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

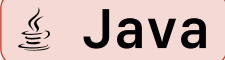


Figure 6: Indices of array `filter`

## 3.6 Properties: Array Classes

- Arrays can be declared for any data types (including custom classes)
- Objects must be of the same type (or subtype, more on this with inheritance)
- Not the objects stored, but references to the objects

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

**Java**

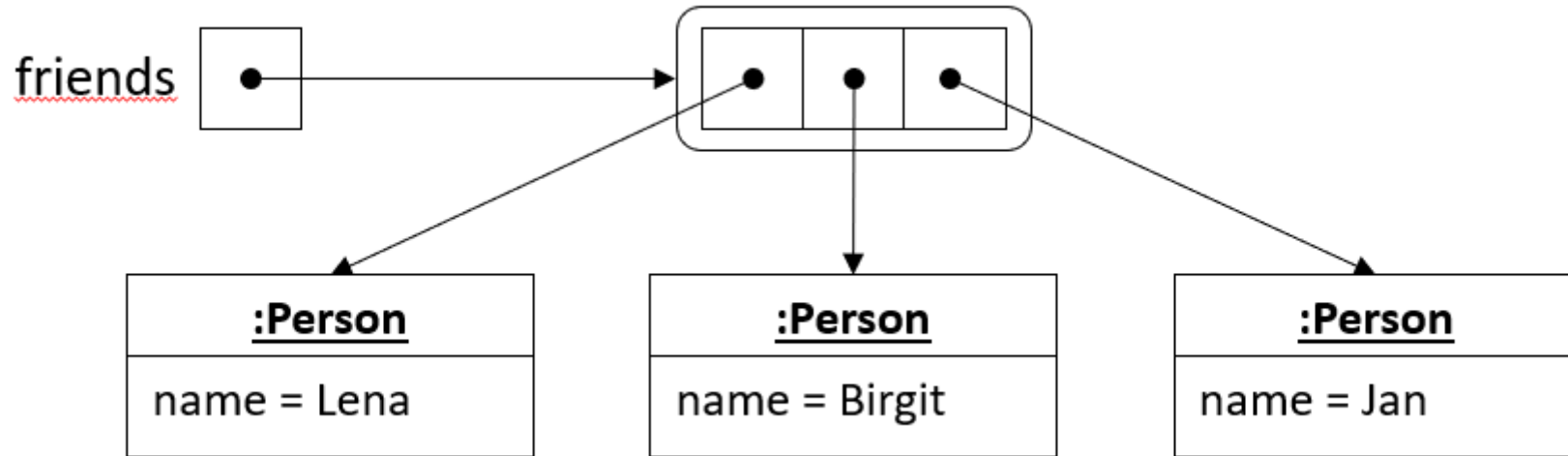


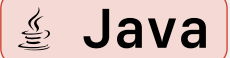
Figure 7: Arrays of objects



### ? Question

What will be output?

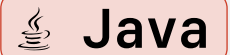
```
1 int[] a = {1, 2, 3, 4, 5};  
2 int[] b = a;  
3 b[3] = 0;  
4 System.out.println(b[2]);  
5 System.out.println(a[3]);
```



### ? Question

What will be output?


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



### ? Question

What will be output?

```
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  }
```

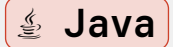
 Java

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

### Task 3

- Write a method that sorts the elements of an `int[]` array in ascending order.
- Test the method using the array {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 }
```



Java

## 4. Multidimensional Arrays

---

## 4.1 Multidimensional Arrays

## 4. Multidimensional Arrays

- Multidimensional arrays are “arrays of arrays”.
- Example: `int[][]` is array whose elements are of data type `int[]`.

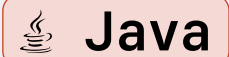
Dynamic declaration:

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



Static declaration:

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





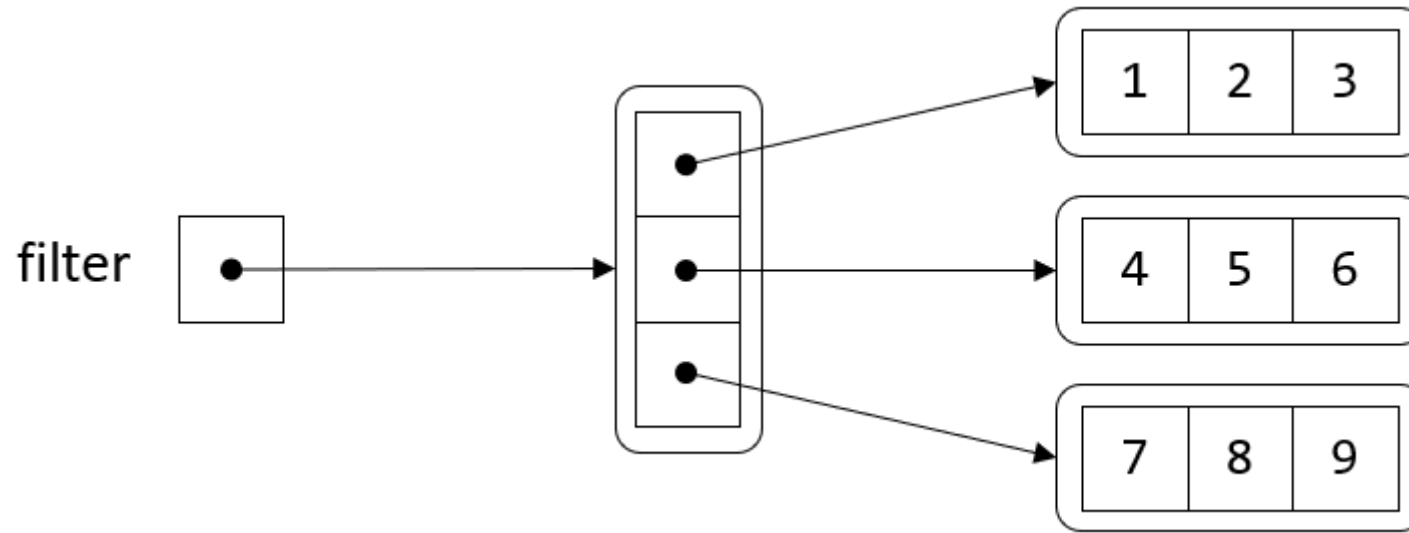


Figure 8: Multidimensional array

### ? Question

What will be output?

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

 **Java**

```
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]);
```

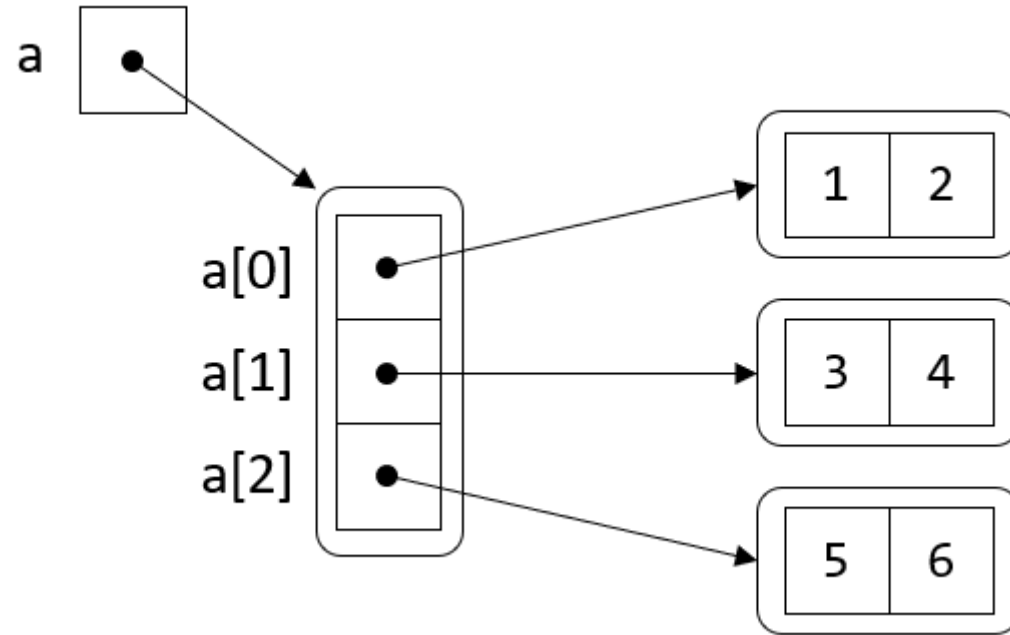


Figure 9: Multidimensional arrays with values

### ? Question

What will be output?

```
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);
```



Java

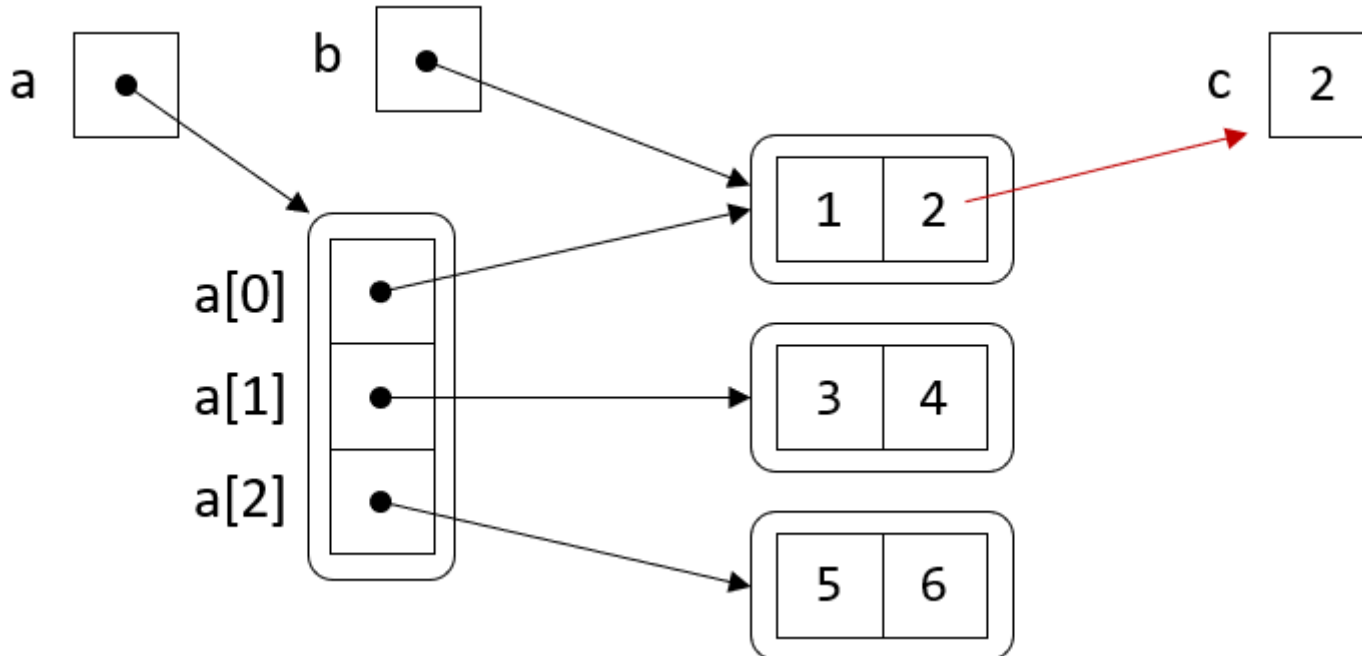


Figure 10: Complex multidimensional arrays

## 4.1 Multidimensional Arrays

## 4. Multidimensional Arrays

- Multidimensional arrays do not have to be rectangular
- Example: Assign own array to each row of a two-dimensional array

### Task 4

Create a triangle matrix using a for loop!

## 4.1 Multidimensional Arrays

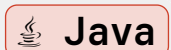
## 4. Multidimensional Arrays

- Multidimensional arrays do not have to be rectangular
- Example: Assign own array to each row of a two-dimensional array

### Task 5

Create a triangle matrix using a for loop!

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



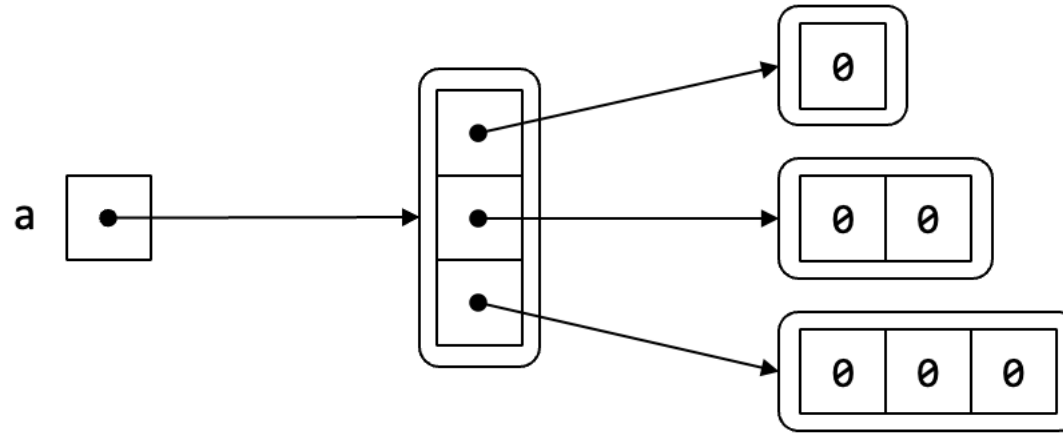


Figure 11: Multidimensional array in the shape of a triangle



## 5. Lists

---

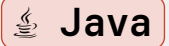
## 5.1 ArrayList

- Arrays: Size cannot be changed after creation (“semi-dynamic”)
- Lists: Elements can be added or removed (“dynamic”)
  - ▶ Data type of elements to be stored in angle brackets (see below: String)

# 5.1 ArrayList

## 5. Lists

```
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 }
```



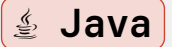
## 5.1 ArrayList

- Examples:
  - ▶ Number of elements (`size()`)
  - ▶ Access to elements (`get()`)
  - ▶ Query whether specific element is in list (`contains()`)
  - ▶ Remove element from list (`remove()`)

# 5.1 ArrayList

## 5. Lists

```
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 }
```



Java

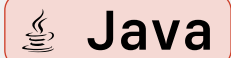
## 6. foreach Loop

---

# 6.1 foreach Loop

## 6. foreach Loop

```
1  for (DataType Variable : IterationObject) {  
2      Statements  
3  }
```



- Motivation:
  - ▶ Sometimes every element e.g. of an array or a list is needed
  - ▶ But: Position within the array or list is not needed
  - ▶ Therefore no loop counter as index needed

## 6.1 foreach Loop


- Loop iterates through array (or list) from first to last element:
- On first pass, variable has the value of the 1st element
- On second pass, variable has the value of the 2nd element and so on
- On last pass, variable has the value of the last element



### ? Question

What will be output?

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

 **Java**

```
2
```

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

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

```
5 }
```

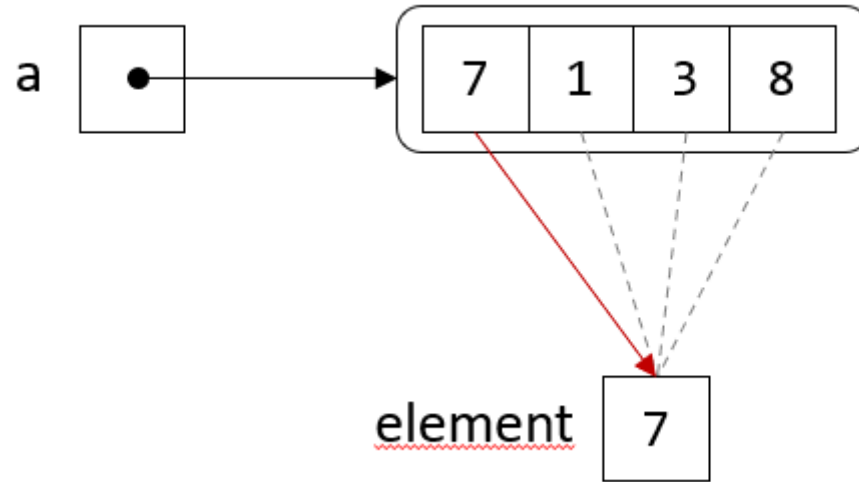


Figure 12: Result of foreach loop

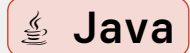
### Task 6

- Create the following using a foreach loop:
- Method that returns the average of the numbers contained in an array
- Program that uses the method

# 6.1 foreach Loop

## 6. foreach Loop

```
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. Wrapper Classes & Math Class**

---

# 7.1 Wrapper Classes

## 7. Wrapper Classes & Math Class

- Primitive data types:
  - ▶ Store value (e.g. integer) directly
  - ▶ Have no methods
- Wrapper classes:
  - ▶ “Wrap” primitive data types into classes
  - ▶ Provide methods (e.g. for integers)

# 7.1 Wrapper Classes

## 7. Wrapper Classes & Math Class

Primitive	Wrapper Class
boolean	Boolean
byte	Byte
short	Short
int	Integer
long	Long
char	Character
float	Float
double	Double

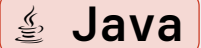
Table 1: Formats and Flags

# 7.1 Wrapper Classes

## 7. Wrapper Classes & Math Class

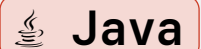
- Convert primitive data types to String

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



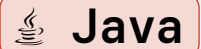
- Shorter alternative via class method:

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



- Convert String to primitive data types:

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



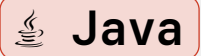


# 7.1 Wrapper Classes

## 7. Wrapper Classes & Math Class

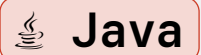
- Conversions:
  - ▶ Boxing: Conversion of primitive data type to object of wrapper class
  - ▶ Unboxing: Conversion of object of wrapper class to primitive data type

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



- Autoboxing: Automatic conversions (both directions)

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



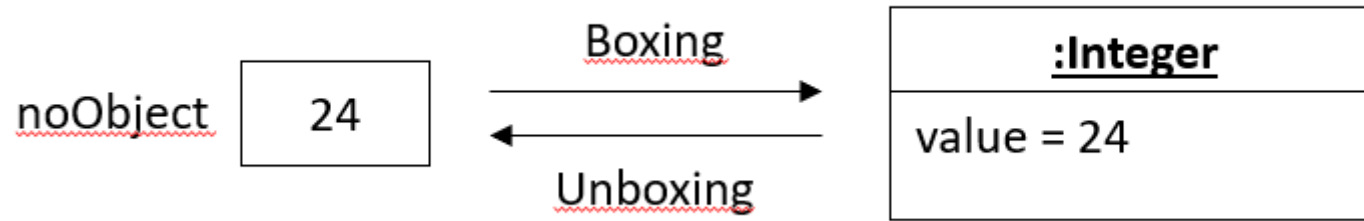


Figure 13: Type conversion with wrapper classes

- Mathematical constants: Euler's number  $e$ , pi  $\pi$
- Mathematical functions (as class methods), e.g.:
  - ▶ Trigonometric functions
  - ▶ Rounding
  - ▶ Absolute value
  - ▶ Exponential function and logarithm
  - ▶ Maximum and minimum
  - ▶ Roots
  - ▶ Random numbers



### Example

```
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.