Object-Oriented Programming in Java

Lecture 4 - Class Libraries

Emily Lucia Antosch

HAW Hamburg

13.08.2025

Contents

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

2.1 Strings

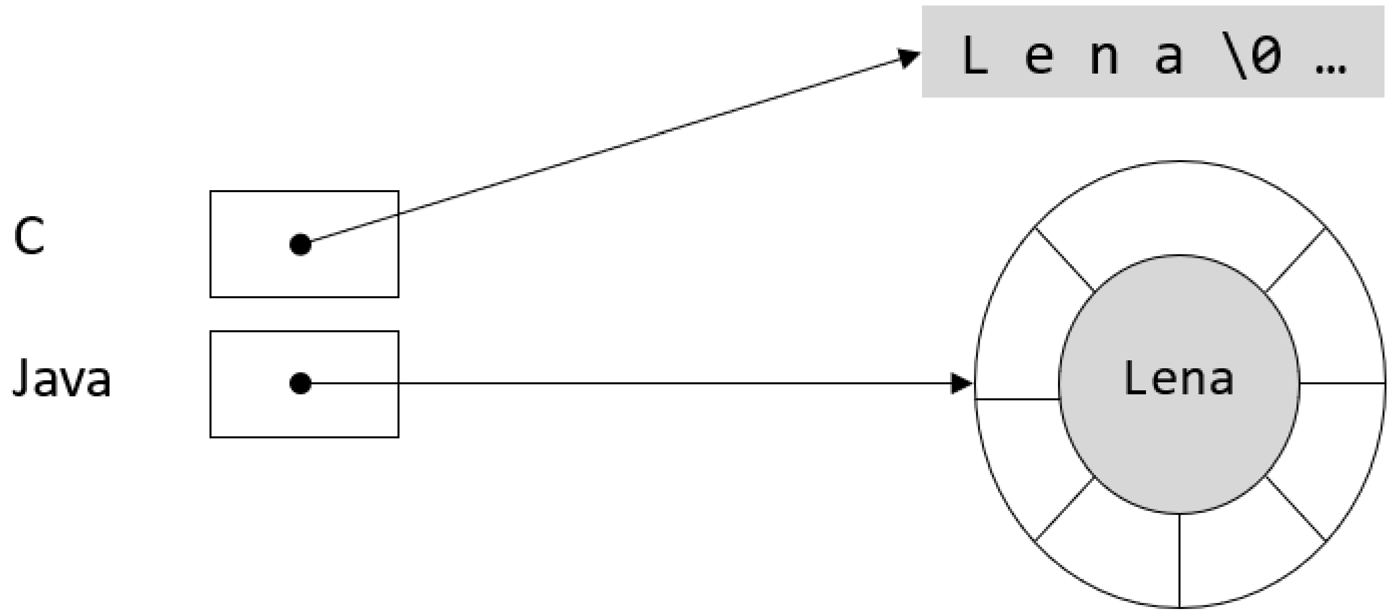


Figure 1: Strings: Differences between C and Java

2.2 Creating Strings

- Creation also using the `new` operator:

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

- Alternatively by assigning a literal:

```java
String name = "Lena";
```

- Assignment of a literal also possible after creation:

```java
String name = new String("Lena");
name = "Birgit";
```

2.3 Strings as Immutable

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

2.3 Strings as Immutable

Question

What output does the following program produce?

Java

```java
public static void main(String[] args) {
        String lena1 = new String("Lena");
        String lena2 = lena1;

        System.out.println("lena1: " + lena1 + "\nlena2: " + lena2);
        System.out.println("References equal: " + (lena1 == lena2));

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

2.3 Strings as Immutable

- For illustration

```java
String lena1 = new String("Lena");
String lena2 = lena1;
```

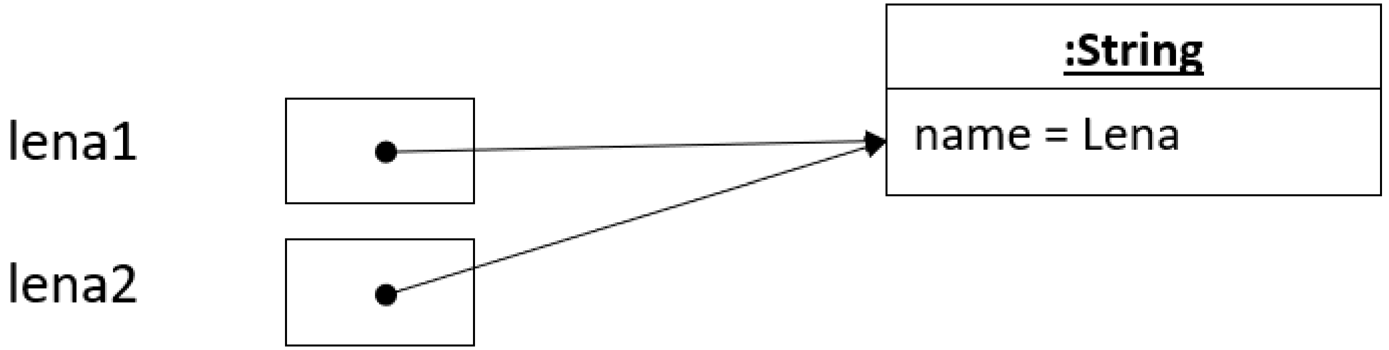


Figure 2: Reference to string

2.3 Strings as Immutable

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

Java

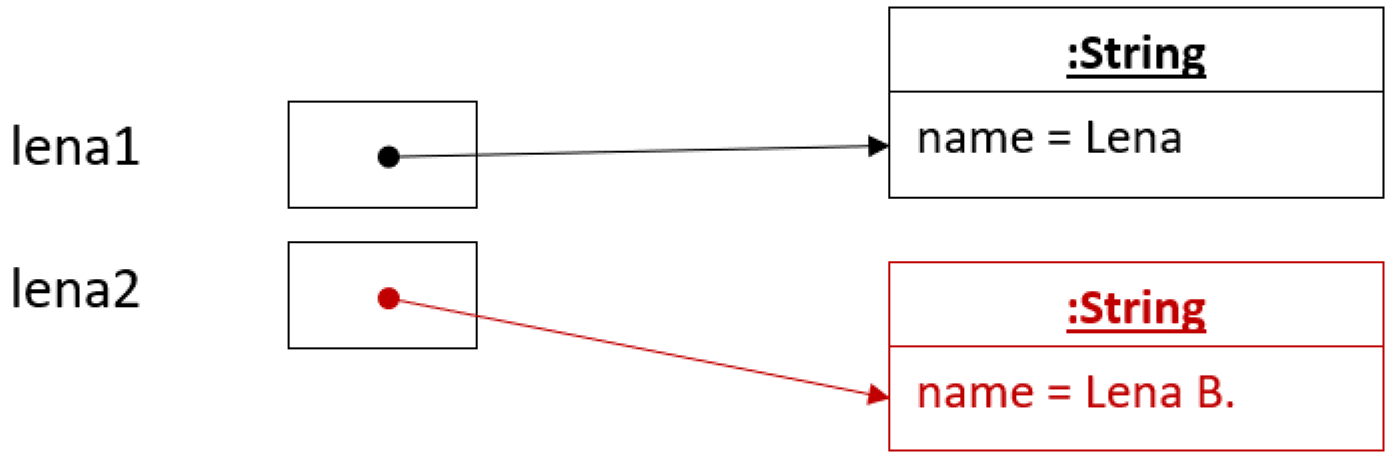


Figure 3: Changing the string leads to new object

2.4 String Concatenation

- Strings can be concatenated using the plus operator:

```java
String name = "Lena " + "or " + "then ";
   name = name + "after all " + "again ";
   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

2.4 String Concatenation

Question

What will be output?

Java

```java
int a = 20;
int b = 22;
System.out.println("Year: " + a + b);
System.out.println(a + b + " (Year)");
```

2.5 `toString()` Method

```java
public String toString() {
        // Method body
        // Return of an object of type String
    }
```

- 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

2.5 `toString()` Method

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

```java
public class Person {
        String firstName, surname;

        public Person(String firstName, String surname) {
            this.firstName = firstName;
            this.surname = surname;
        }

        public String toString() {
            return firstName + " " + surname;
        }
```

2.5 `toString()` Method

```
    }
```

2.5 `toString()` Method

```java
Person lena = new Person("Lena", "Jensen");
String name = "Name: " + lena;
System.out.println(lena);
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!

2.6 String Methods

Task 2

- Replace “Humbug” with “Hamburg”.

Java
```java
String hamburg = "Welcome to Humbug!";
hamburg = hamburg.replace("Humbug", "Hamburg");
System.out.println(hamburg);
```

2.6 String Methods

Question

What will be output?

Java
```java
String upper = "Welcome to Hamburg!";
String lower = "welcome to hamburg!";

System.out.println(lower.equals(upper));
System.out.println(lower.equals(upper.toLowerCase()));
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++

2.7 String Formatting

Question

What will be output?

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

2.7 String Formatting

Question

What will be output?

Java

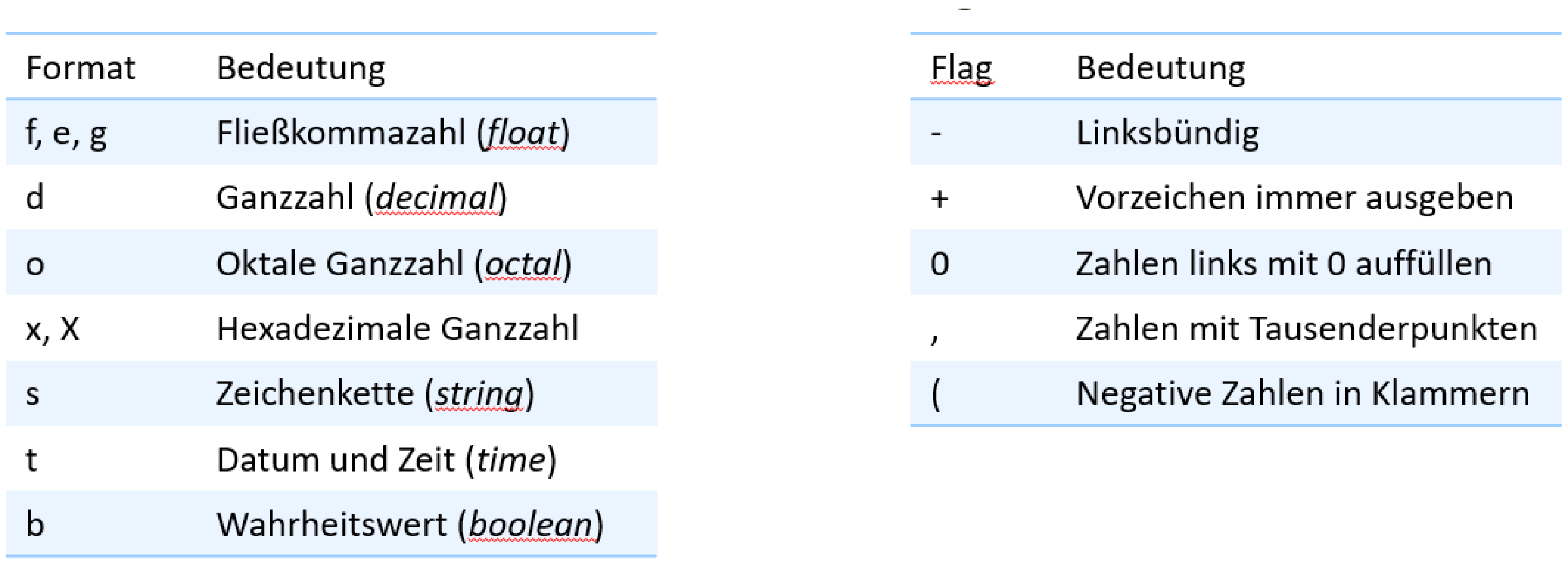
```java
double wind = 21.4532;
String weather = String.format("%s %d: %.1f km/h", "Station", 7, wind);
System.out.println(weather);
```

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

2.7 String Formatting

- Format specifications:

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



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

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

Figure 4: Formats and Flags

2.7 String Formatting

Question

What will be output?

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

2.7 String Formatting

2.7 String Formatting

Question

What will be output?

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

- Output:
  - 21.45 km/h
  - 21.45 km/h

2.7 String Formatting

- 00021.45 km/h

2.7 String Formatting

! **Memorize**

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

2.7 String Formatting

Question

- What do you notice?

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

- Output: 21.45 km/h
-
-

2.7 String Formatting

Question

- What do you notice?

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

Java

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

2.7 String Formatting

Example

```java
double wind = 21.4532;
System.out.println(String.format(Locale.US, "%2.2f km/h", wind));
System.out.println(String.format(Locale.GERMAN, "%2.2f km/h", wind));
```

- Output: 21.45 km/h 21.45 km/h

3. Arrays

3.1 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

3.1 Arrays

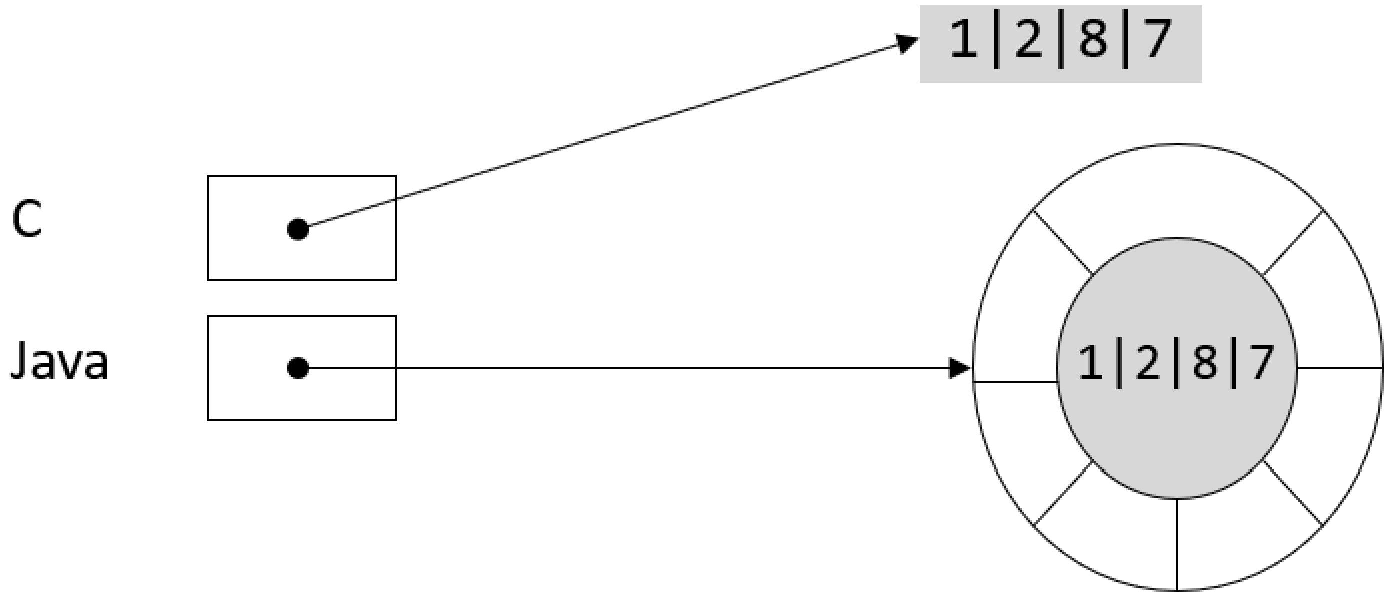


Figure 1: Arrays in Java and C

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

```java
int[] filter;
```

Java

3.2 Creating Arrays

Memorize

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

```java
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

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

Java

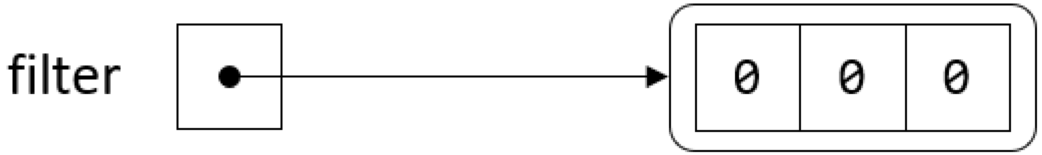


Figure 2: Creation of an array

3.4 Creation: Assigning Elements

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

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

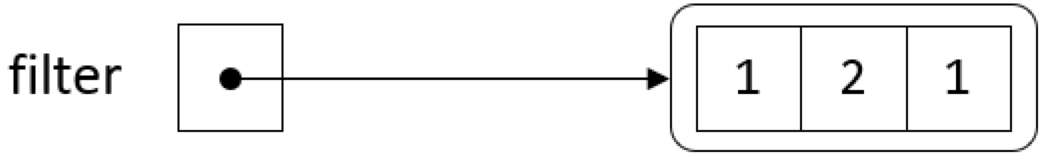


Figure 3: Assigning values through index access

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

Java

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

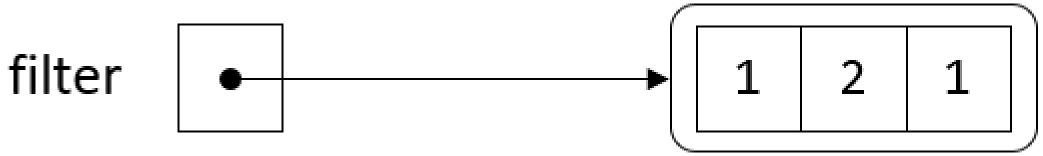


Figure 4: Filling during declaration

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?

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

Java

3.6 Properties: Array Classes

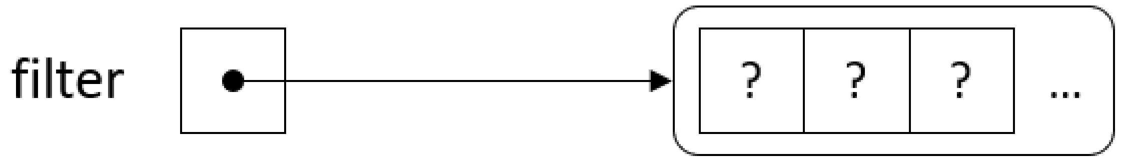


Figure 5: Filling through `for` loop

3.6 Properties: Array Classes

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

3.6 Properties: Array Classes

Example

Examples of allowed and disallowed indices:

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

3.6 Properties: Array Classes

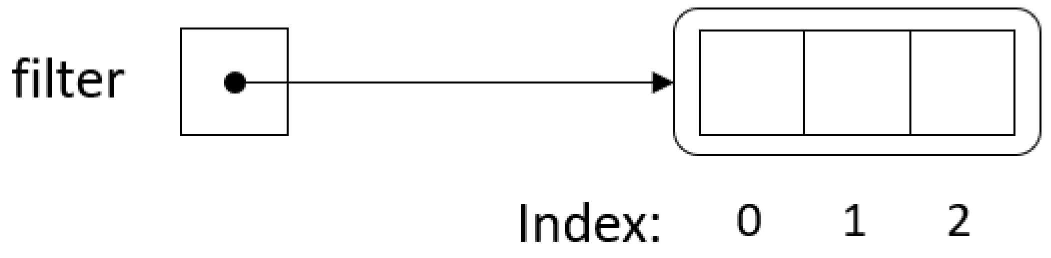


Figure 6: Indices of array `filter`

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

Java

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

3.6 Properties: Array Classes

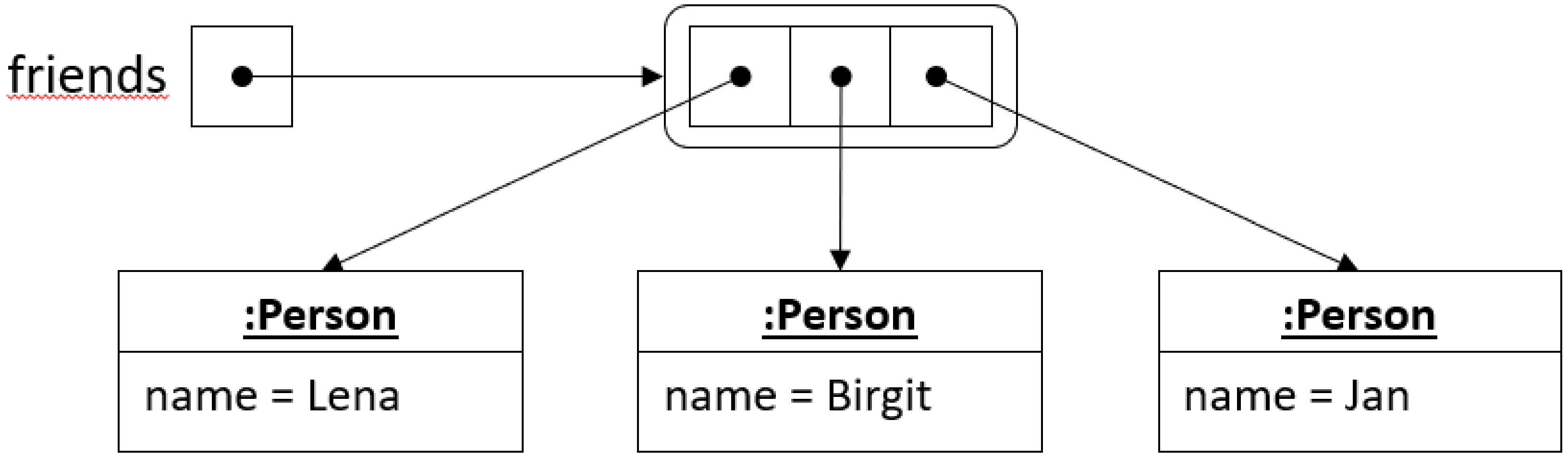


Figure 7: Arrays of objects

3.7 Questions

Question

What will be output?

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

3.7 Questions

Question

What will be output?

Java

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

3.7 Questions

Question

What will be output?

Java

```java
public class ArrayDemo {
        static int[] createSortedArray(int a, int b) {
                if (a < b) {
                        return new int[] {a, b};
                } else {
                        return new int[] {b, a};
                }
        }

```

3.7 Questions

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

3.7 Questions

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}`.

3.7 Questions

Java

```java
public static void sort(int[] a) {
    int i = 0;
    while (i <= a.length - 2) {
        if (a[i] > a[i+1]) {
            // Swap elements and shift index to left element
            int temp = a[i];
            a[i] = a[i+1];
            a[i+1] = temp;
            if (i > 0)
                i--;
        } else {
            i++;
        }
    }
}
```

4. Multidimensional Arrays

4.1 Multidimensional Arrays

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

Dynamic declaration:

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

Java

Static declaration:

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

Java

4.1 Multidimensional Arrays

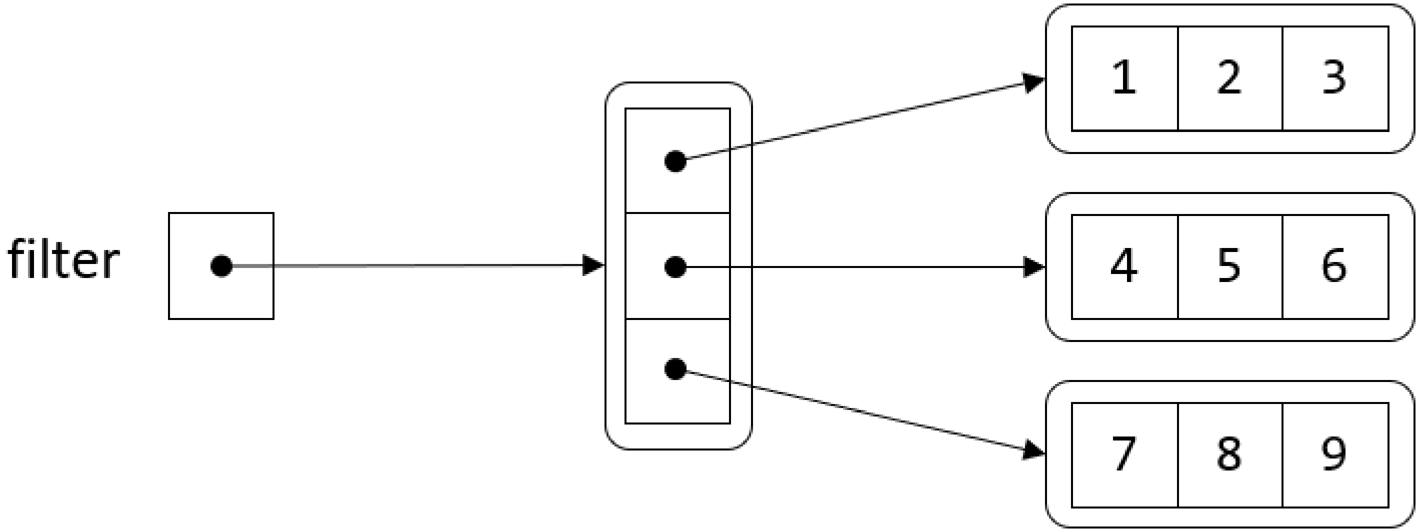


Figure 8: Multidimensional array

4.1 Multidimensional Arrays

Question

What will be output?

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

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

System.out.println(a[1][1]);
System.out.println(a[2][0]);
```

4.1 Multidimensional Arrays

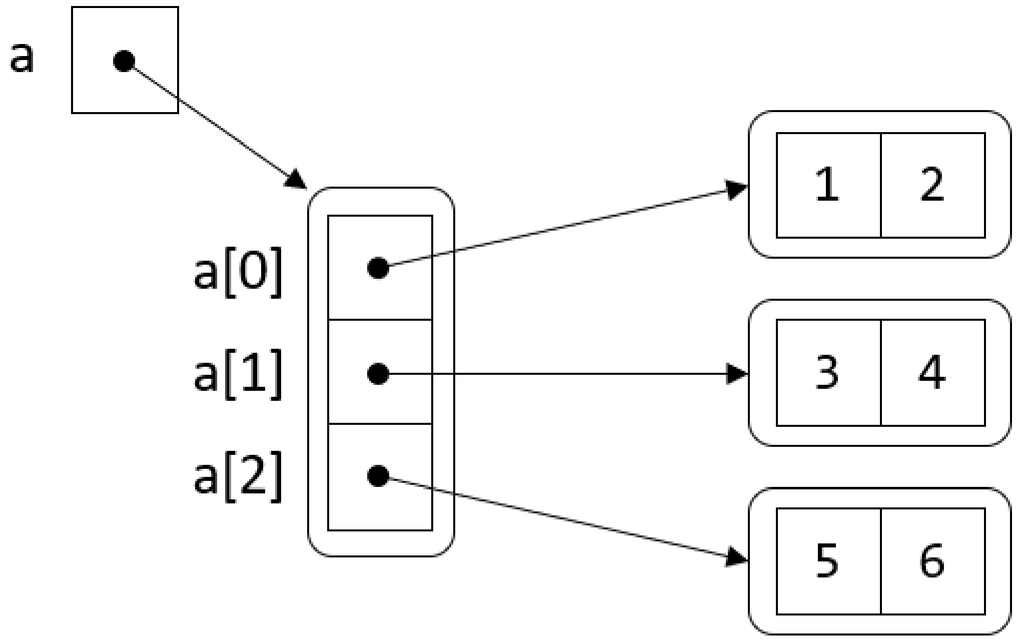


Figure 9: Multidimensional arrays with values

4.1 Multidimensional Arrays

Question

What will be output?

Java

```java
int[][] a = {{1,2}, {3,4}, {5,6}};
int[] b = a[0];
int c = b[1];

b[1] = 7;
System.out.println(a[0][1]);
System.out.println(c);
```

4.1 Multidimensional Arrays

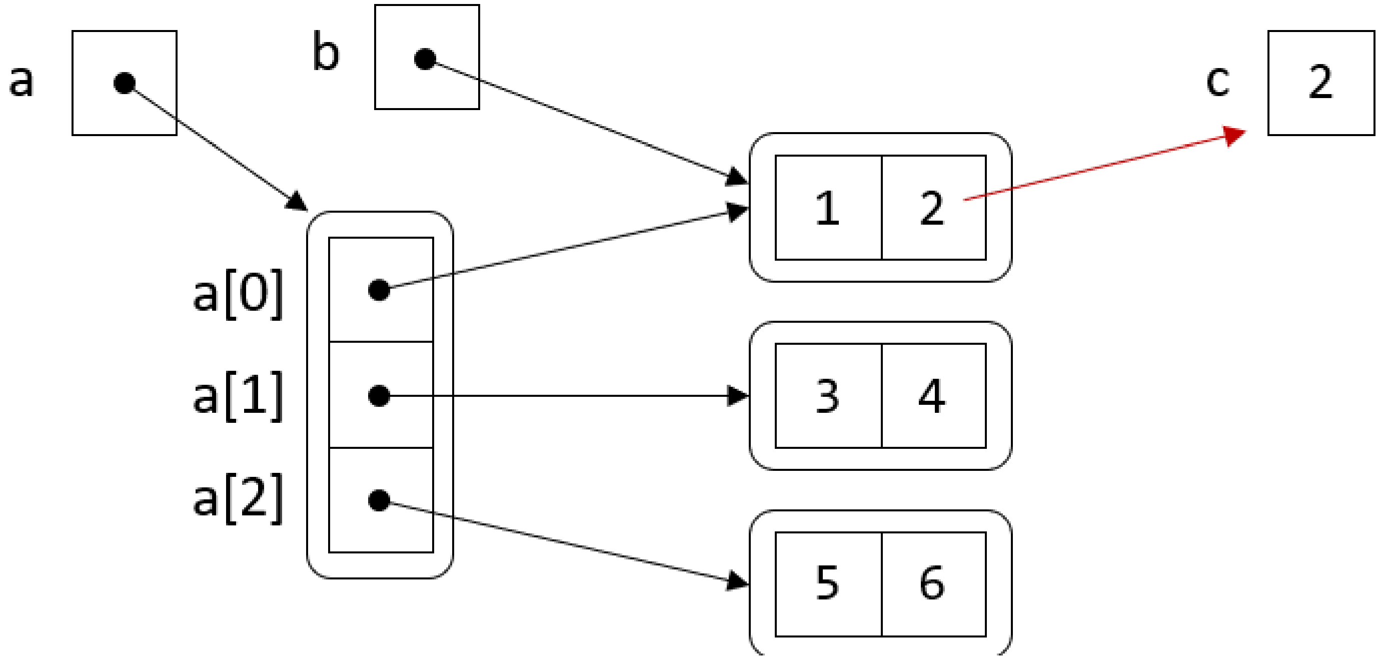


Figure 10: Complex multidimensional arrays

4.1 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

- 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!

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

Java

4.1 Multidimensional Arrays

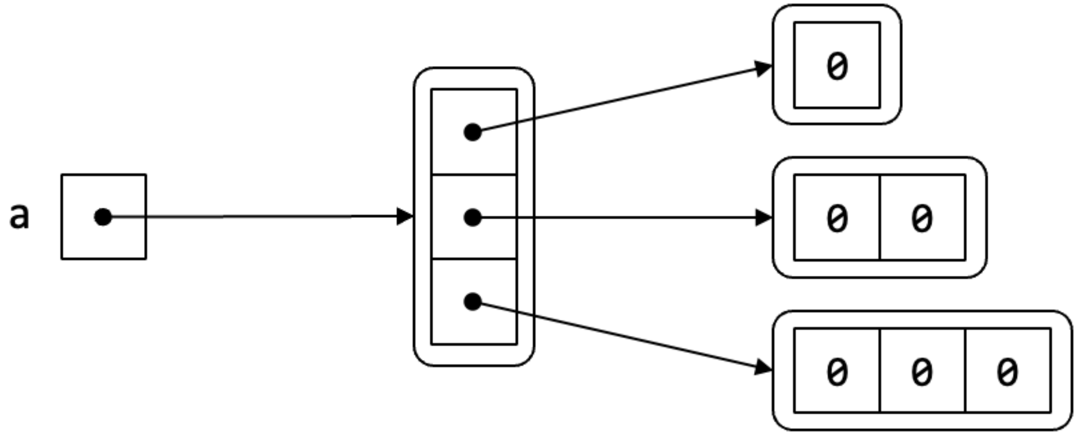


Figure 11: Multidimensional array in the shape of a triangle

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

Java

```java
public class ArrayListDemo {
        public static void main(String[] args) {
            ArrayList<String> names = new ArrayList<String>();
            names.add("Lena");
            names.add("Birgit");
            names.add("Jan");
            names.add(new String("Jan"));
        }
    }
```

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

Java

```java
ArrayList<String> names = new ArrayList<String>();
    String birgit = "Birgit";
    names.add("Lena");
    names.add(birgit);

    for (int i = 0; i < names.size(); i++) {
        System.out.println(names.get(i));
    }

    if (names.contains(birgit)) {
        names.remove(birgit);
    }
```

6. foreach Loop

6.1 foreach Loop

```java
for (DataType Variable : IterationObject) {
      Statements
}
```

- 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

6.1 foreach Loop

Question

What will be output?

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

  for (int element : a) {
      System.out.println("Element: " + element);
  }
```

Java

6.1 foreach Loop

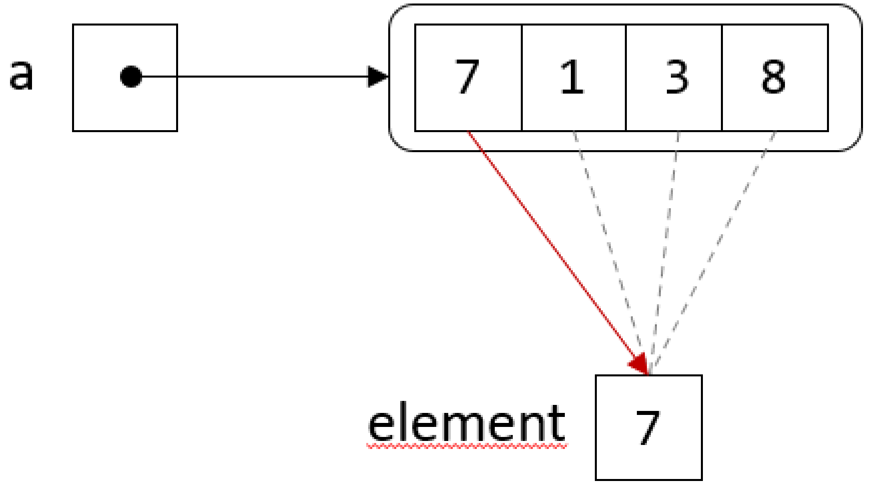


Figure 12: Result of foreach loop

6.1 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

Java

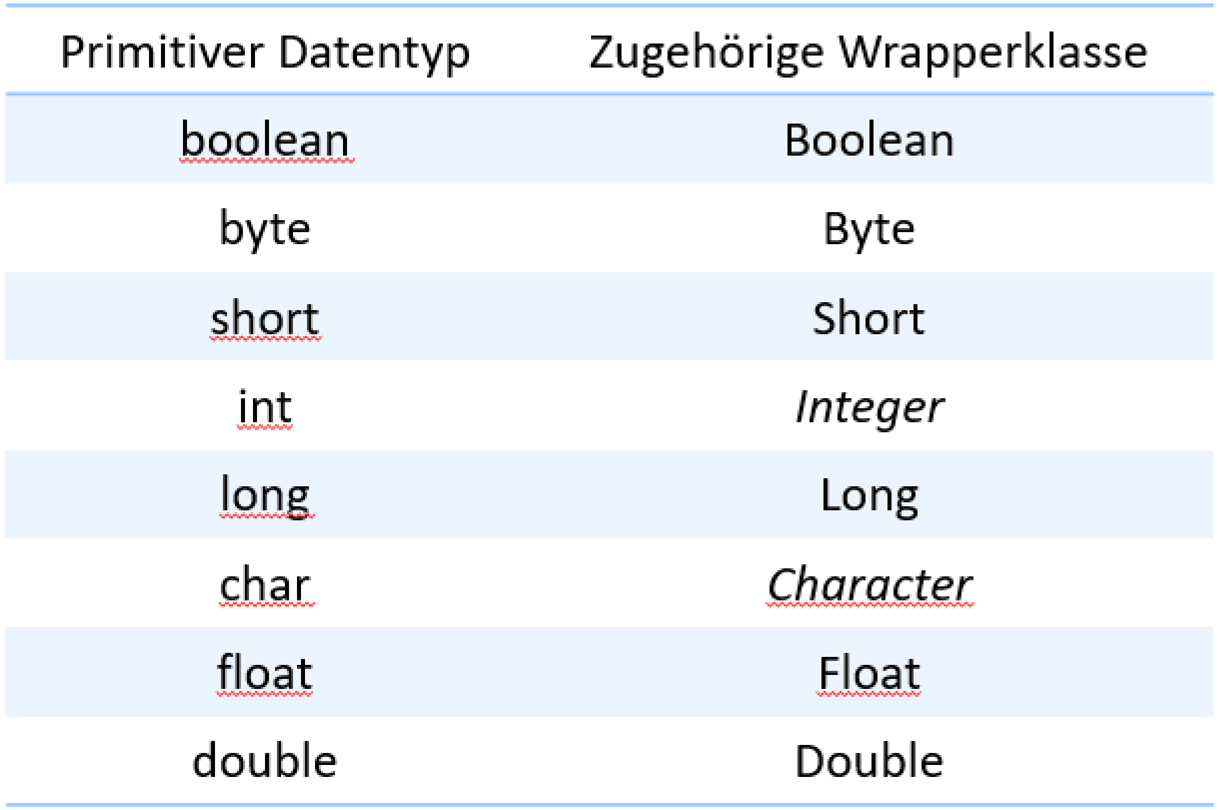
```java
static double average(double[] numbers) {
        double sum = 0.0;

        for(double number : numbers) {
            sum += number;
        }
        return sum / numbers.length;
    }

    public static void main(String[] args) {
        double[] a = {1.43, 2, .2, 6.32, 7.1, 8.1};
        System.out.println("Average = " + average(a));
    }
```

7. Wrapper Classes & `Math` Class

7.1 Wrapper Classes

- 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



Primitiver Datentyp	Zugehörige Wrapperklasse
boolean	Boolean
byte	Byte
short	Short
int	*Integer*
long	Long
char	*Character*
float	Float
double	Double

Figure 13: Wrapper classes for primitive data types

7.1 Wrapper Classes

- Convert primitive data types to String

```java
int a = 7;
Integer b = new Integer(a);
String c = b.toString();
```

- Shorter alternative via class method:

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

- Convert String to primitive data types:

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

7.1 Wrapper Classes

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

Java

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

- Autoboxing: Automatic conversions (both directions)

Java

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

7.1 Wrapper Classes

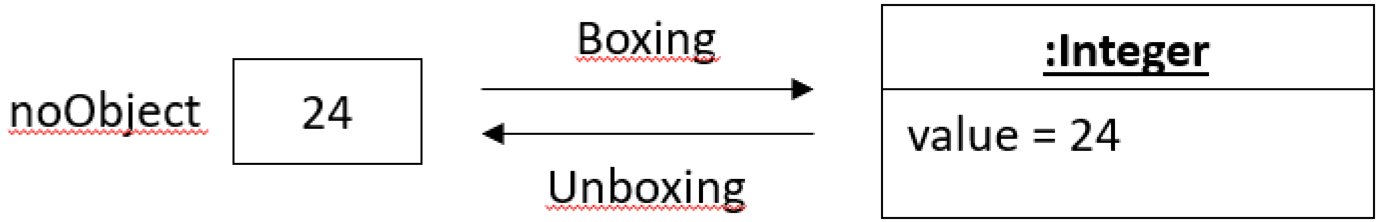


Figure 14: Type conversion with wrapper classes

7.2 `Math` Class

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

7.2 Math Class

Example

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

8. License Notice

8.1 Attribution