Technische Hochschule Rosenheim
Technical University of Applied Sciences

# Programming Basics – WiSe21/22

## Arrays

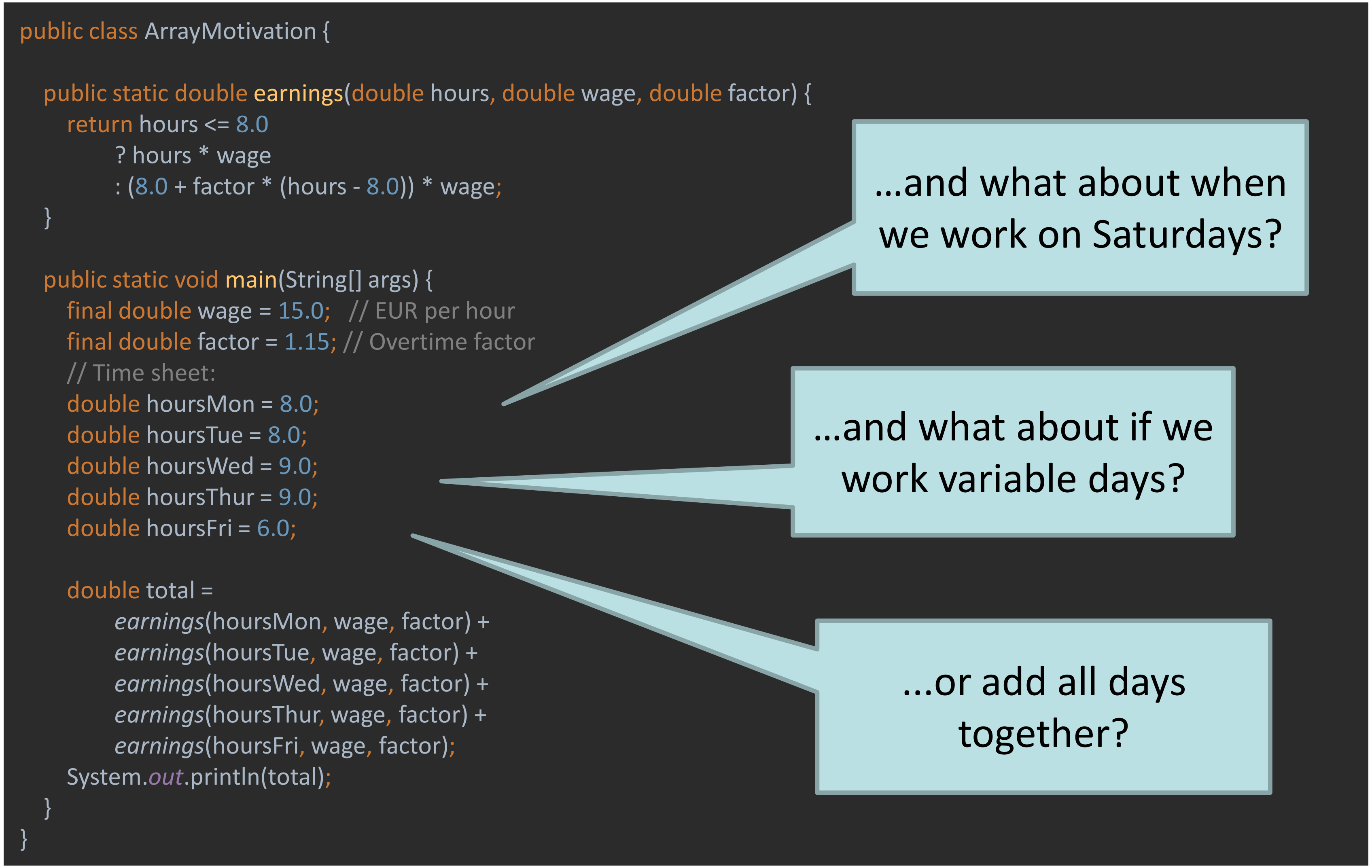
Prof. Dr. Silke Lechner-Greite

# Table of contents - overall

Technische
Hochschule
**Rosenheim**

1. Introduction
2. Fundamental language concepts
3. Control structures
4. Methods
5. Arrays
6. Object orientation
7. Classes
8. Packages
9. Characters and Strings
10. Unit Testing
11. Exceptions
12. I/O

# Arrays



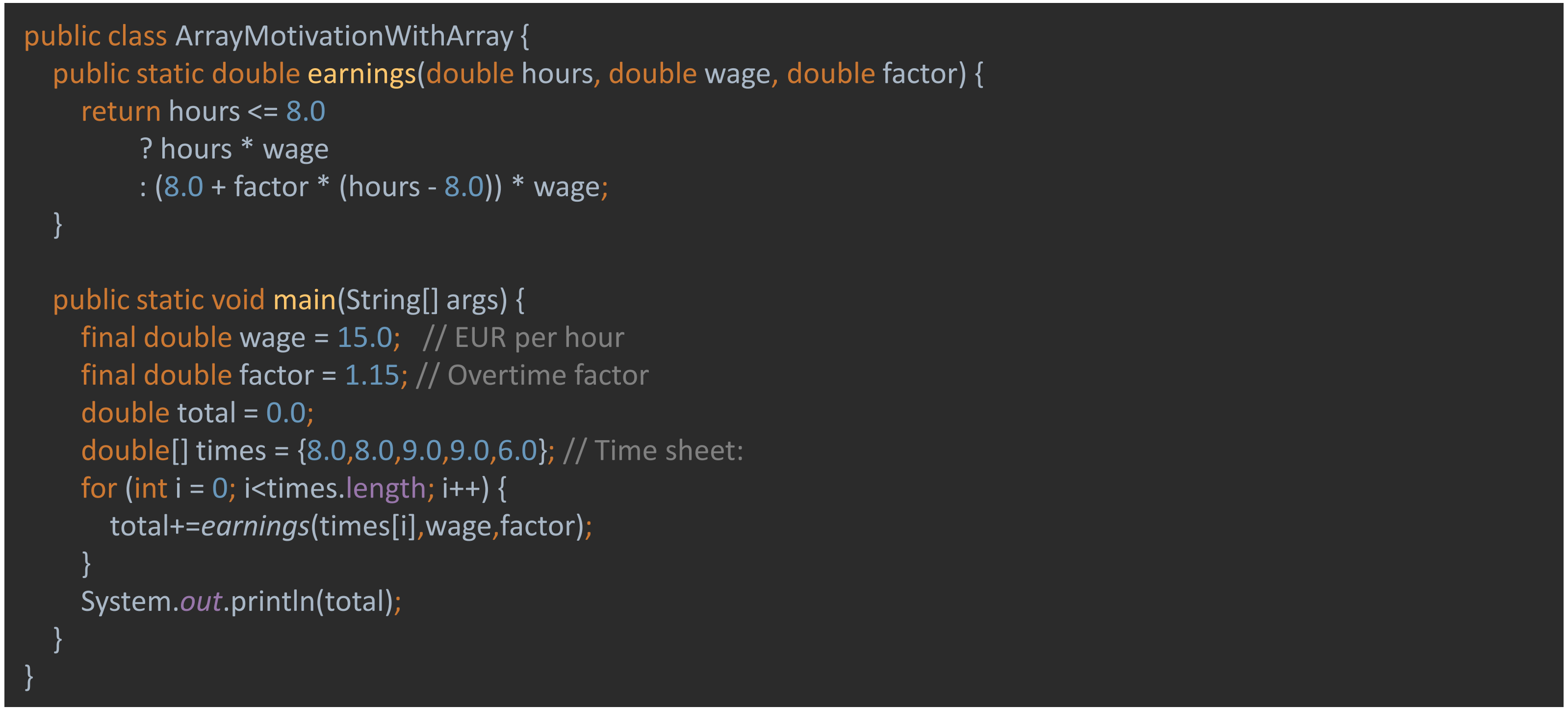
```
public class ArrayMotivation {

  public static double earnings(double hours, double wage, double factor) {
    return hours <= 8.0
        ? hours * wage
        : (8.0 + factor * (hours - 8.0)) * wage;
  }

  public static void main(String[] args) {
    final double wage = 15.0;   // EUR per hour
    final double factor = 1.15; // Overtime factor
    // Time sheet:
    double hoursMon = 8.0;
    double hoursTue = 8.0;
    double hoursWed = 9.0;
    double hoursThur = 9.0;
    double hoursFri = 6.0;

    double total =
        earnings(hoursMon, wage, factor) +
        earnings(hoursTue, wage, factor) +
        earnings(hoursWed, wage, factor) +
        earnings(hoursThur, wage, factor) +
        earnings(hoursFri, wage, factor);
    System.out.println(total);
  }
}
```

...and what about when we work on Saturdays?

...and what about if we work variable days?

...or add all days together?

# Arrays



```java
public class ArrayMotivationWithArray {
   public static double earnings(double hours, double wage, double factor) {
      return hours <= 8.0
            ? hours * wage
            : (8.0 + factor * (hours - 8.0)) * wage;
   }

   public static void main(String[] args) {
      final double wage = 15.0;   // EUR per hour
      final double factor = 1.15; // Overtime factor
      double total = 0.0;
      double[] times = {8.0,8.0,9.0,9.0,6.0}; // Time sheet:
      for (int i = 0; i<times.length; i++) {
         total+=earnings(times[i],wage,factor);
      }
      System.out.println(total);
   }
}
```

## Chapter 5: Arrays

5.1 One-dimensional arrays

5.2 n-dimensional arrays

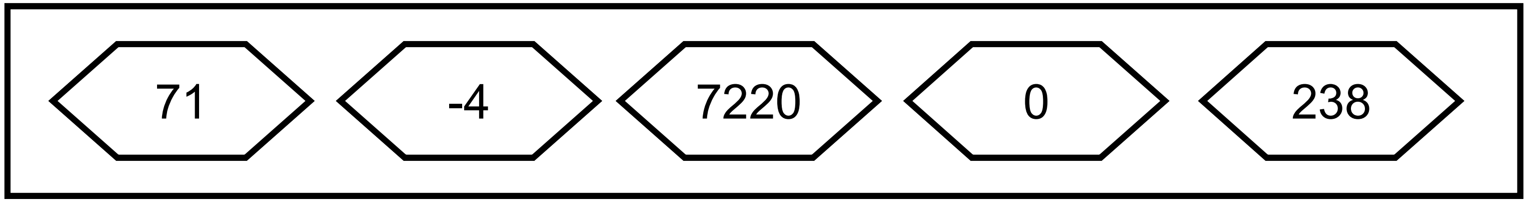
5.3 Useful helper methods (search and sorting methods)

5.4 Extended `for` loop

# Motivation

- Motivation
  - Sometimes you need multiple similar elements in a structure
  - These should then usually also be processed in the same way

- Solution: container types
  - Encapsulates elements of the same type in a specific structure
  - Provides basic functions for this structure to simplify data handling



71 | -4 | 7220 | 0 | 238

# Array

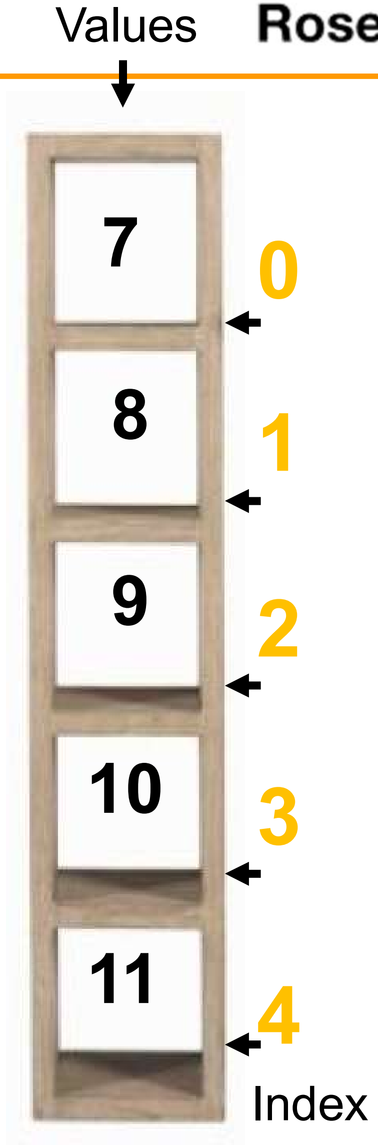
- Special version of a container type
- Synonym: field
- Sequential succession of individual elements of the same type (primitive data types or non-primitive (reference) types)
- Individual elements interchangeable (stored values of the elements can be changed at any time)
- Direct access to individual elements via index
- Total length fixed (number of elements cannot be changed after declaration)
- Concept available in nearly every programming language

# Basic idea: one-dimensional array

- Visual representation
  - An array is like a storage rack or register
  - The number of shelves/ compartments is arbitrary
  - When the storage rack is built, the number of shelves is fixed
  - There can be a maximum of one item stored per shelf
  - A shelf can also be empty
  - Only items of the same type are located within a storage rack

# Arrays of primitive type

- Contents of primitive type
  - Contents stored directly on shelf
  - That is, the array element directly contains the data value



# Creation and use of arrays

- 3 steps are required:
  1. Declaration of an array variable == reference variable pointing to the array in memory
  2. Specification of the array size
  3. Write and read access to the array elements

# Declaration of an array variable (1)

- Meaning
  - There is a corresponding array type for each Java type/element type
  - Array types are thus a type family
  - Array type specifies the element type – a specific array *object* has a fixed, unchangeable size
- Syntax
  - Element type, followed by empty square brackets
  - *type* [ ]
- Examples:
  - `int[],boolean[],char[], String[]`

Speech: "int array", "boolean array", "char array", "String array"

# Declaration of an array variable (2)

- Specification of array type and variable name

- Examples:

```
int[] countList;
double[] measuredValues;
String[] words;
```

# Specification of the array size (1)

- Arrays are reference types, whose objects must be explicitly created

- Method:
  - Creating an array with the element type *type:*
    **`new`** *`type`*`[`*`expression`*`]`
    - *`type`*: data type of the individual elements
    - *`expression`*
      - Number of elements
      - Any expression that returns an **`int`** type result
  - The number of elements is specified during runtime when calling **`new`**, and cannot be changed afterwards!

# Specification of the array size (2)

- Examples of variables declared in slide 12:
  - `countList = new int[4];`
  - `measuredValues = new double[1+17*4];`

  -
    ```
    int length = 17;
    words = new String[length];
    ```

# Initialising with default values

- Meaning
  - Elements of an array are automatically initialised with default values when they are created
  - The default value depends on the data type

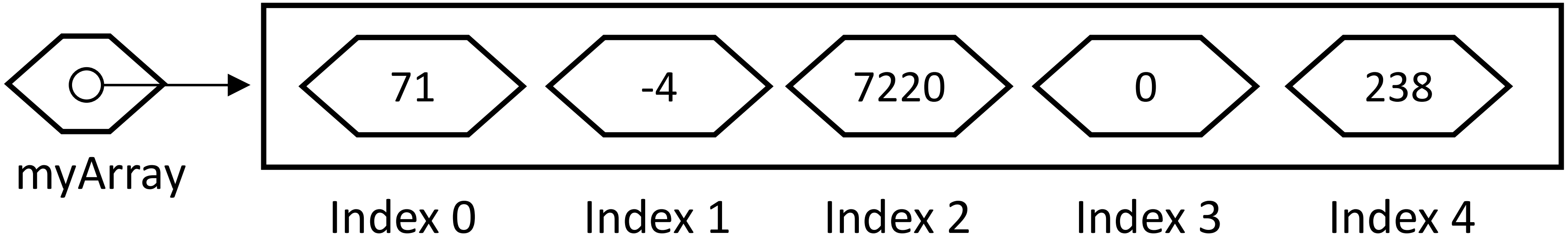
- Example:
  - `int[] numbers = new int [5];`
  - Array of 5 `int` elements, which are initialised with `0`
  - Assigned to the array variable `numbers`
  - Only creates the array!
  - In case of an array of reference variables, elements get initialised with `null`
    - `Book[] books = new Book[5];`

# Initialising via array literals

- General
  - literal is a constant explicit value of a type
  - i.e. `true` for type `boolean`, `17` for type `int`
- Array literals
  - An array literal is a constant of an array type
  - A new array is allocated to match the list of given values (see example below)
  - Length of the list of given values determines the length of the array
  - Array initialises with the values of the list in the same order
  - List elements are any expressions, compatible with the array type
- Example:
  - `int[] arr= new int[]{71, -4, 7220, 0, 238}`
  - `String[] visitor= new String[] {"otto", "rudi"};`

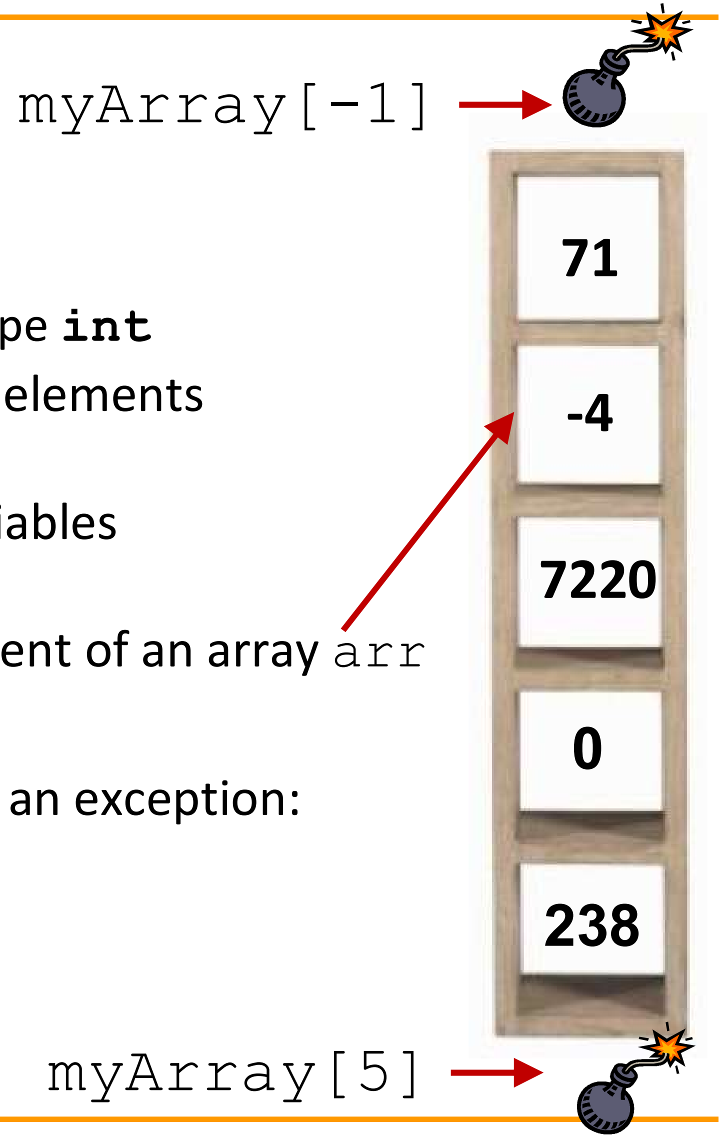
# Access to array elements (1)

- Meaning
  - Elements of an array follow a consecutive sequence
  - Each element has a unique position in the array
  - The position is identified by an integer-based index
  - Indexing starts at 0, then continues sequentially
  - Index of the last element is (array length – 1) (because indexing starts @ 0)



# Access to array elements (2)

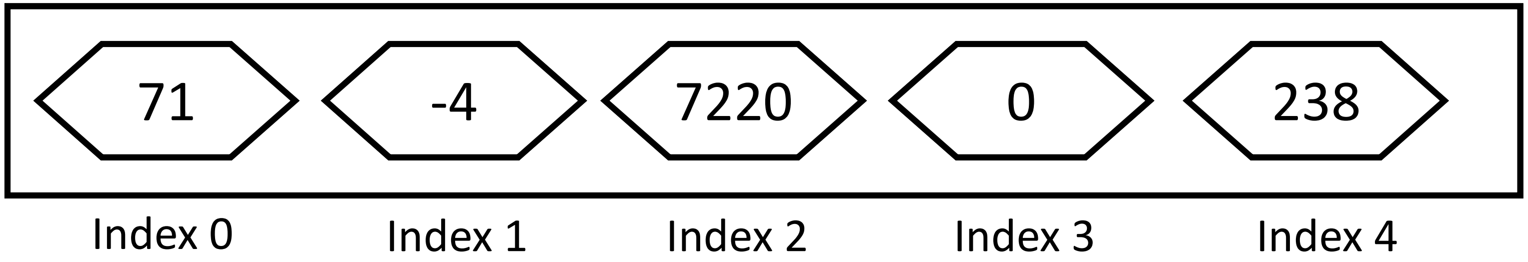
- Access to individual array element via index
  - Syntax: `array[expression]`
    - *`array`*: reference to the array
    - *`expression`*: expression with result of type **`int`**
  - Access to an individual element leaves other elements of the array unchanged
  - Array elements can be used like ordinary variables of the element type
  - Example: `arr[1]` accesses the second element of an array `arr`
- Index error
  - If the index value is not allowed, JVM throws an exception: `ArrayIndexOutOfBoundsException`
  - Negative index is never allowed
  - JVM checks all accesses to array elements



# Writing into array

- Examples:

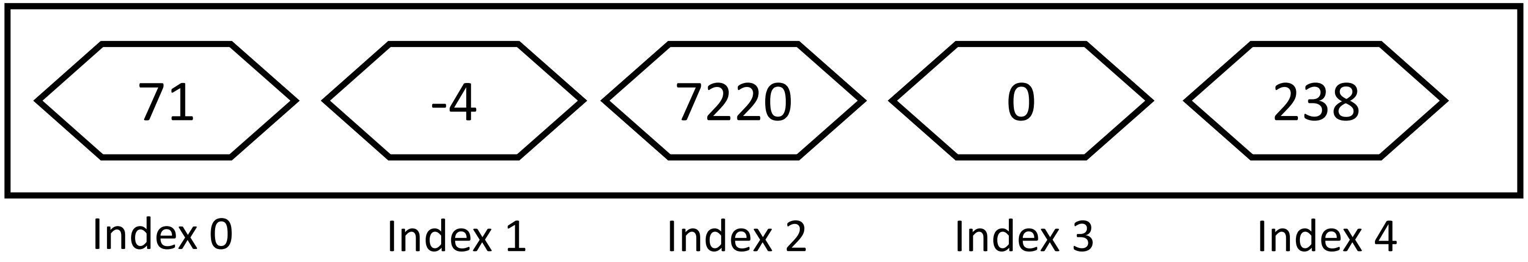
```
int[] myArray = new int[5];
myArray[0] = 71;
myArray[1] = -4;
myArray[2] = 7220;
myArray[3] = 0;
myArray[4] = 238;
```



| 71 | -4 | 7220 | 0 | 238 |
|---|---|---|---|---|
| Index 0 | Index 1 | Index 2 | Index 3 | Index 4 |

# Reading from array

- Examples:
  - `int myValue = myArray[3];`
  - What value do we get?



| Index 0 | Index 1 | Index 2 | Index 3 | Index 4 |
|---|---|---|---|---|
| 71 | -4 | 7220 | 0 | 238 |

# Determining the length of an array

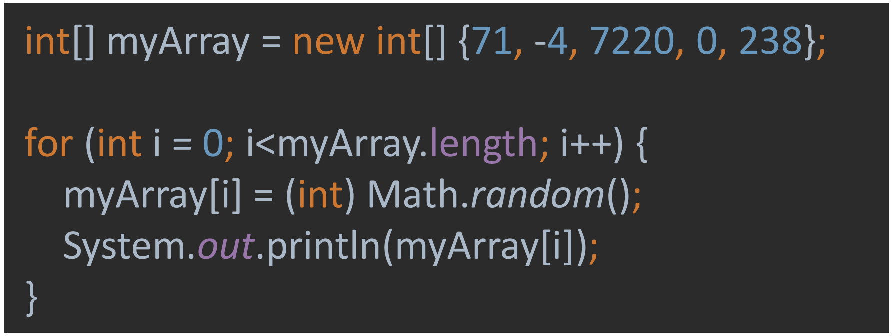
- Meaning
  - To edit an array, you have to know how many elements it contains
  - This property is readable from the array
    - Publicly readable **`final`** attribute `length` for the number of elements
    - Access analogous to attributes in objects
    - *array*.`length`

- Example:

```
int[] myArray= new int[] {71, -4, 7220, 0, 238};
System.out.println(myArray.length);
```

What value do we get?

# Writing into an array: for-Loop



```
int[] myArray = new int[] {71, -4, 7220, 0, 238};

for (int i = 0; i<myArray.length; i++) {
   myArray[i] = (int) Math.random();
   System.out.println(myArray[i]);
}
```

- For-loop: read/write access @ index position

# Exercise – Access to array elements

- Live exercise
  - Complete Task 1a to 1e on the live exercises sheet "Arrays"

## Chapter 5: Arrays

5.1 One-dimensional arrays

5.2 n-dimensional arrays

5.3 Useful helper methods (search and sorting methods)

5.4 Extended `for` loop

# N-dimensional arrays (1)

- In practice, one-dimensional arrays are often not sufficient
- Example: working with tables
- Solution: n-dimensional arrays
- Syntax in Java:
  - A pair of square brackets is used for each dimension
  - The corresponding number of elements is given for each dimension
- Example: two-dimensional table

```
double[][] table = new double[3][4];

alternatively:
double[][] table = {{1.3, 3.5, 7.8, 4.0},
                    {7.5, 8.3, 6.9, 3.2},
                    {4.5, 6.7, 3.4, 7.2}};
```

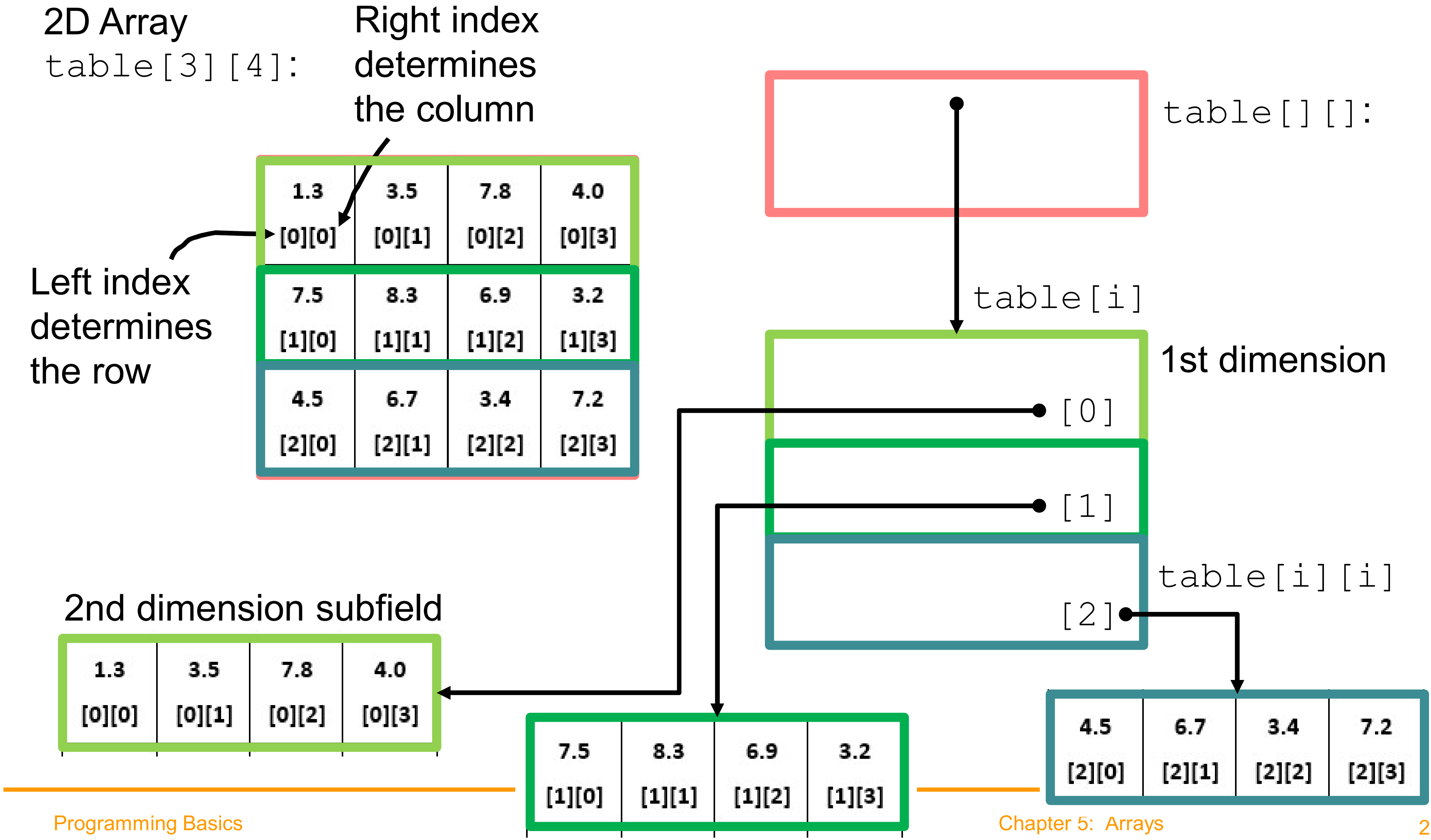
# N-dimensional arrays (2)

- Strictly speaking, there are no n-dimensional arrays in Java.
- All arrays in Java are one-dimensional. However, the elements of an array can be arrays again.
- This creates nested arrays ➔ several levels of nesting == dimension of the array.

- Nested arrays are initialised by nested enumerations. The length of an array can be queried by the constant `length`.
- With field.length you get the length of the first dimension.
- With field[i].lenght you get the length of the second dimension, where I corresponds to an index of the first dimension.

# N-dimensional arrays (3)

*Based on : Balzert, H.: Java: Der Einstieg in die Programmierung*



2D Array
`table[3][4]:`

Right index determines the column

Left index determines the row

| | | | |
|---|---|---|---|
| 1.3<br>[0][0] | 3.5<br>[0][1] | 7.8<br>[0][2] | 4.0<br>[0][3] |
| 7.5<br>[1][0] | 8.3<br>[1][1] | 6.9<br>[1][2] | 3.2<br>[1][3] |
| 4.5<br>[2][0] | 6.7<br>[2][1] | 3.4<br>[2][2] | 7.2<br>[2][3] |

`table[][]:`

`table[i]`

1st dimension

`[0]`

`[1]`

`[2]`

`table[i][i]`

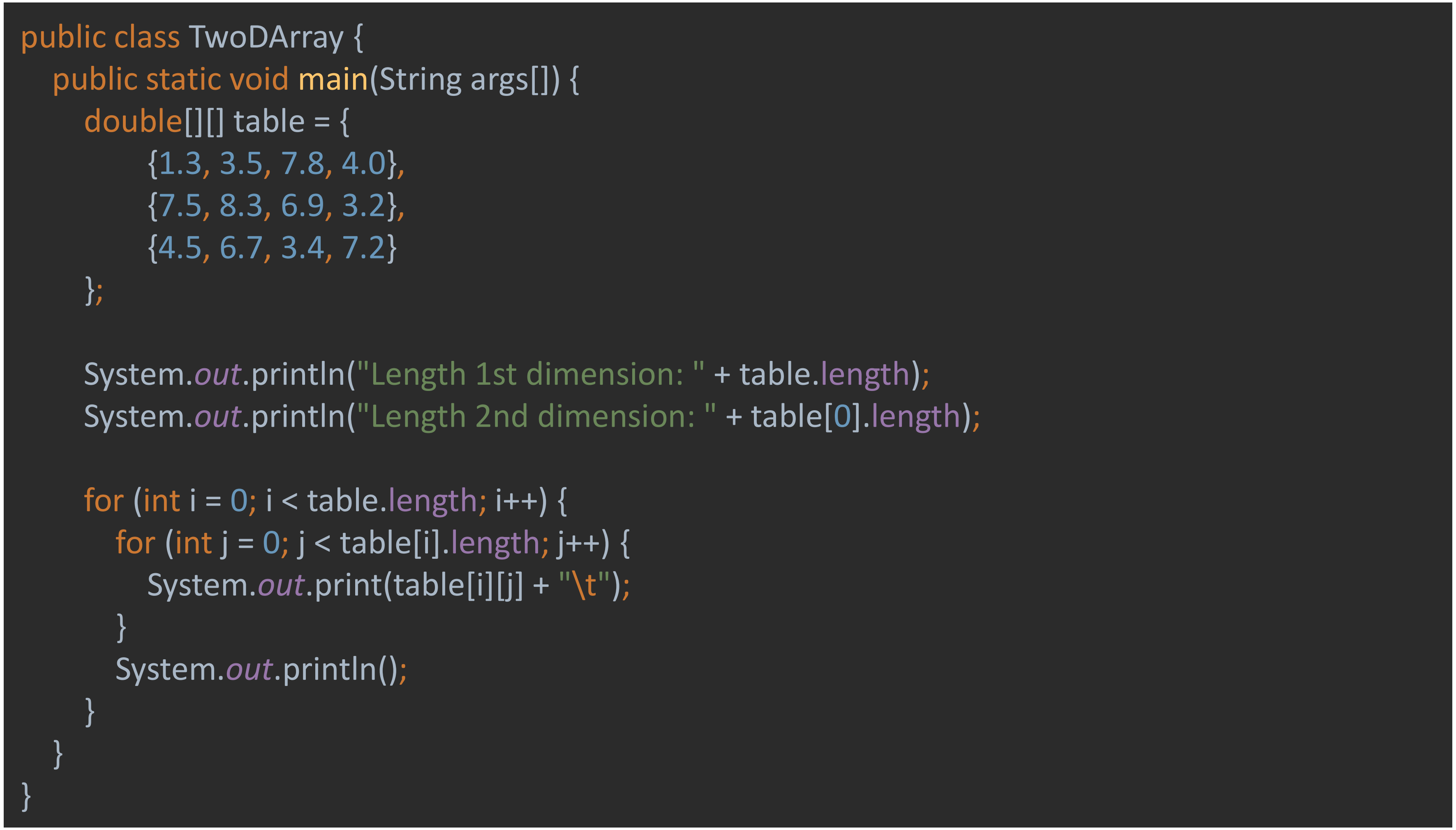
2nd dimension subfield

| | | | |
|---|---|---|---|
| 1.3<br>[0][0] | 3.5<br>[0][1] | 7.8<br>[0][2] | 4.0<br>[0][3] |

| | | | |
|---|---|---|---|
| 7.5<br>[1][0] | 8.3<br>[1][1] | 6.9<br>[1][2] | 3.2<br>[1][3] |

| | | | |
|---|---|---|---|
| 4.5<br>[2][0] | 6.7<br>[2][1] | 3.4<br>[2][2] | 7.2<br>[2][3] |

# N-dimensional arrays (3)



```
public class TwoDArray {
   public static void main(String args[]) {
      double[][] table = {
            {1.3, 3.5, 7.8, 4.0},
            {7.5, 8.3, 6.9, 3.2},
            {4.5, 6.7, 3.4, 7.2}
      };

      System.out.println("Length 1st dimension: " + table.length);
      System.out.println("Length 2nd dimension: " + table[0].length);

      for (int i = 0; i < table.length; i++) {
         for (int j = 0; j < table[i].length; j++) {
            System.out.print(table[i][j] + "\t");
         }
         System.out.println();
      }
   }
}
```

## Chapter 5: Arrays

5.1 One-dimensional arrays

5.2 n-dimensional arrays

5.3 Useful helper methods (search and sorting methods)

5.4 Extended `for` loop

# Useful helper methods (1)

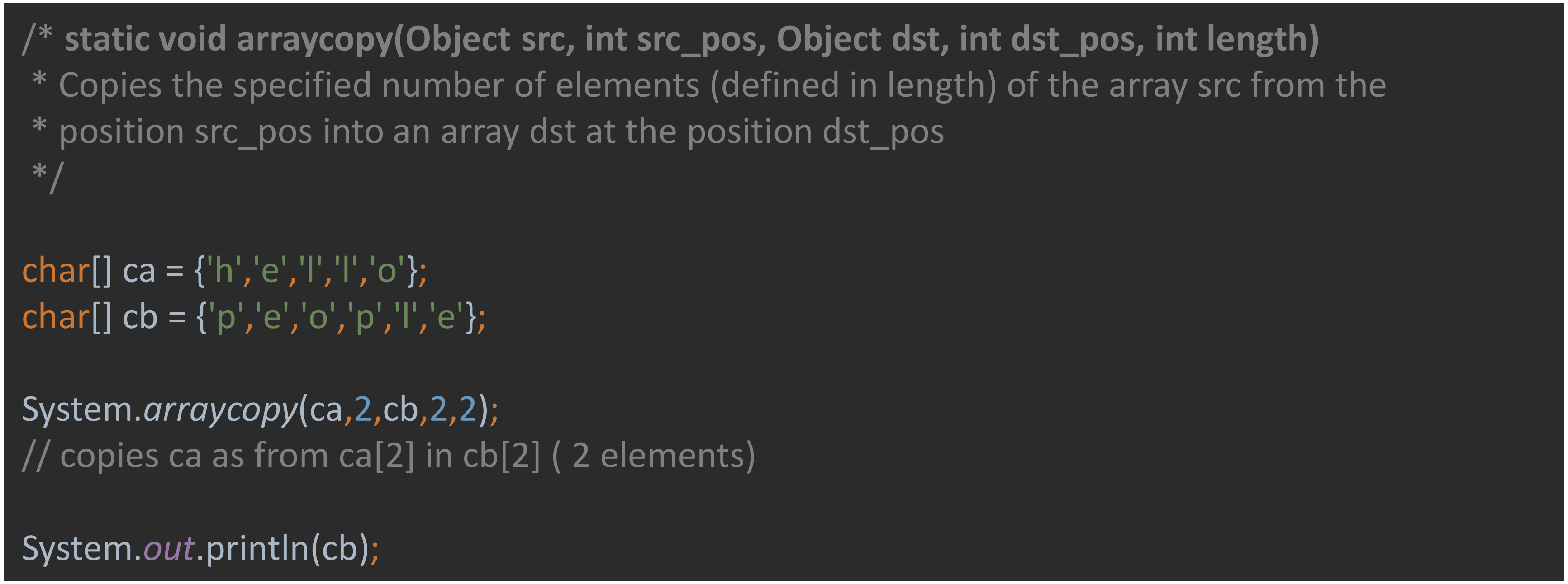
- Class `java.util.Arrays` provides very powerful static methods for handling arrays, for example:
  - sorting arrays
  - searching arrays

# Exercise – Sorting method

- Live exercise
  - Complete Task 2 on the live exercises sheet “Arrays”
  - You have 10 minutes.

# Useful helper methods (2)

- Class `java.lang.System` includes a static method `arraycopy` for copying arrays or parts of them



```
/* static void arraycopy(Object src, int src_pos, Object dst, int dst_pos, int length)
 * Copies the specified number of elements (defined in length) of the array src from the
 * position src_pos into an array dst at the position dst_pos
 */

char[] ca = {'h','e','l','l','o'};
char[] cb = {'p','e','o','p','l','e'};

System.arraycopy(ca,2,cb,2,2);
// copies ca as from ca[2] in cb[2] ( 2 elements)

System.out.println(cb);
```

What output do we get?

# Chapter 5: Arrays

5.1 One-dimensional arrays

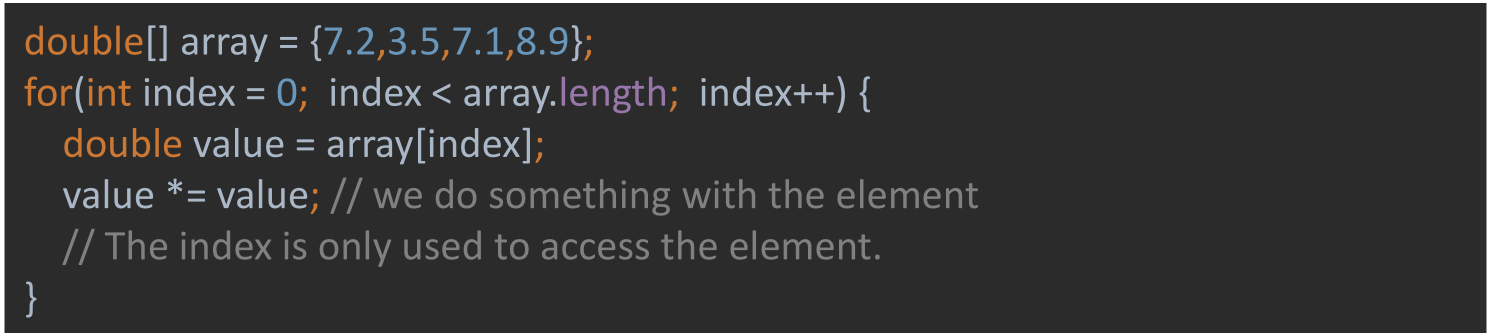
5.2 n-dimensional arrays

5.3 Useful helper methods (search and sorting methods)

5.4 Extended `for` loop

# Running sequentially through an array

- Sequential run through:
  - Often you want to process the elements in an array one after the other
  - The order of processing is often from the first to the last element
- Example:



```
double[] array = {7.2,3.5,7.1,8.9};
for(int index = 0;  index < array.length;  index++) {
   double value = array[index];
   value *= value; // we do something with the element
   // The index is only used to access the element.
}
```

# `forEach` loops (1)

- Simplification:
  - **`forEach`** loop
  - Short form of a **`for`** loop for a specific purpose
  - Applicable to other data structures
  - Schema: **`for`** (*type variable* : *array*)
    {*statement(s)*;}

- Example:

```
int[] array = {1,3,5,7,9};
for (int element: array)
  System.out.println(element);
```

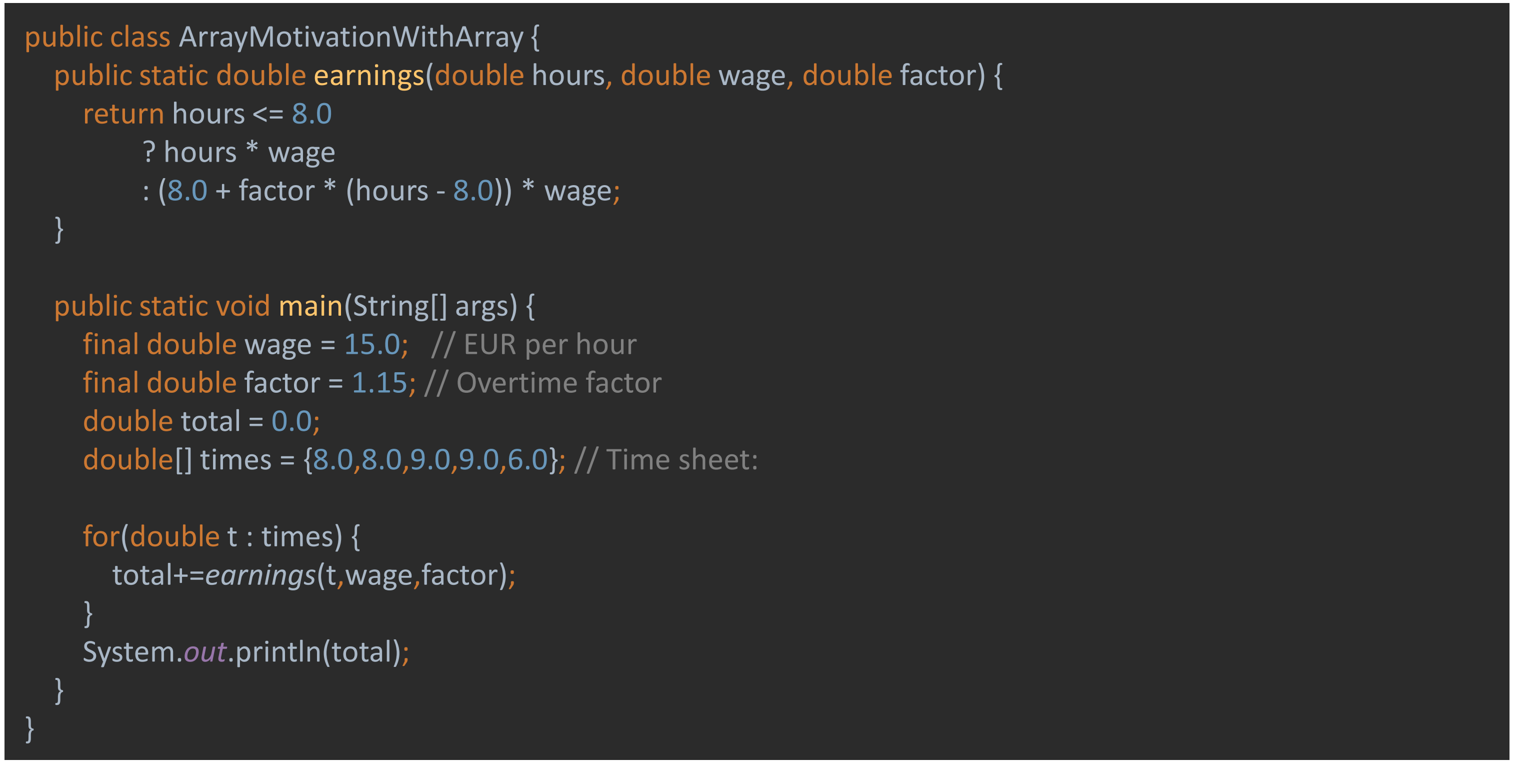
# `forEach` loops (2)

- Limitations:
  - Can only read - but not write to - the array
  - Always start with the first element
  - Sequential run, no jumps, no omissions
  - Only *one* array, not several in parallel
  - Premature termination only with `break`

# Exercise – `forEach` loop

- Live exercise
  - Complete Task 3 on the live exercises sheet “Arrays”
  - You have 5 minutes.

# Initial example with `forEach`



```java
public class ArrayMotivationWithArray {
   public static double earnings(double hours, double wage, double factor) {
      return hours <= 8.0
            ? hours * wage
            : (8.0 + factor * (hours - 8.0)) * wage;
   }

   public static void main(String[] args) {
      final double wage = 15.0;   // EUR per hour
      final double factor = 1.15; // Overtime factor
      double total = 0.0;
      double[] times = {8.0,8.0,9.0,9.0,6.0}; // Time sheet:

      for(double t : times) {
         total+=earnings(t,wage,factor);
      }
      System.out.println(total);
   }
}
```