# Java

Object Oriented Programming

Hannes Ueck, Jakob Krude
25. November 2020

Java-Kurs

## Overview

# Arrays

An array is a data-type that can hold a **fixed number** of elements. An Element can be any simple data-type or object.

```java
public static void main(String[] args) {

    int[] intArray = new int[10];
    intArray[8] = 7; // assign 7 to the 9th element
    intArray[9] = 8; // assign 8 to the last element

    System.out.println(intArray[8]); // prints: 7
}

```

You can access every element via an index. A n-element array has indexes from 0 to (n-1).

# Array Initialization

You can initialize an array with a set of elements.

```java
public static void main(String[] args) {

    int[] intArray = {3, 2, 7};

    System.out.println(intArray[0]); // prints: 3
    System.out.println(intArray[1]); // prints: 2
    System.out.println(intArray[2]); // prints: 7
}
```

## Alternative Declaration

There two possible positions for the square brackets.

```
1   public static void main(String[] args) {
2
3       // version 1
4       int[] intArray1 = new int[10];
5
6       // version 2
7       int intArray2[] = new int[10];
8   }
9
```

## 2-Dimensional Array

Arrays work with more than one dimension. An m-dimensional array has m indexes for one element.

```java
public static void main(String[] args) {

    // an array with 100 elements
    int[][] intArray = new int[10][10];

    intArray[0][0] = 0;
    intArray[0][9] = 9;
    intArray[9][9] = 99;
}
```

## Assignment with Loops

Loops are often used to assign elements in arrays.

```java
public static void main(String[] args) {

    int[][] intArray = new int[10][10];

    for(int i = 0; i < 10; i++) {
        for(int j = 0; j < 10; j++) {
            intArray[i][j] = i*10 + j;
        }
    }
}
```

Loops are often used to assign elements in arrays.

```java
public static void main(String[] args) {

    Student[][] studentArray = new Student[10][10];

    for(int i = 0; i < 10; i++) {
        for(int j = 0; j < 10; j++) {
            intArray[i][j] = new Student();
        }
    }
}
```

# OOP in Java

## Class Student

```java
public class Student {

    // Attributes
    private String name;
    private int matriculationNumber;


    // Methods
    public void setName(String name) {
        this.name = name;
    }

    public int getMatriculationNumber() {
        return matriculationNumber;
    }

}
```

## Creation

We learned how to declare and assign a primitive datatype.

```
1        int a; // declare a
2        a = 273; // assign 273 to a
3
```

The creation of an object works similar.

```
1        Student example = new Student();
2        // create an instance of Student
3
```

The **object** derived from a **class** is also called **instance**. The variable is called the **reference**.

## Calling a Method

```java
public class Student {

    private String name;

    public String getName() {
        return name;
    }

    public void setName(String newName) {
        name = newName;
    }

}
```

The class *Student* has two methods: *void printTimetable()* and *void printName()*.

## Calling a Method

```java
public class Main {

    public static void main(String[] args) {
        Student example = new Student(); // creation
        example.setName("Jane"); // method call
        String name = example.getName();
        System.out.println(name); // Prints "Jane"
    }

}
```

You can call a method of an object after its creation with
**reference.methodName();**.

## Calling a Method

```java
public class Student {

    private String name;

    public void setName(String newName) {
        name = newName;
        printName();      // Call own method
        this.printName(); // Or this way
    }

    public void printName() {
        System.out.println(name);
    }

}
```

You can call a method of the own object by simply writing
**methodName();** or **this.methodName();**

# Methods with Arguments

```java
public class Calc {

    public void add(int summand1, int summand2) {
        System.out.println(summand1 + summand2);
    }

    public static void main(String[] args) {
        int summandA = 1;
        int summandB = 2;
        Calc calculator = new Calc();
        System.out.print("1 + 2 = ");
        calculator.add(summandA, summandB);
        // prints: 3
    }

}
```

# Methods with Return Value

A method without a return value is indicated by **void**:

```java
public void add(int summand1, int summand2) {
    System.out.println(summand1 + summand2);
}
```

A method with an **int** as return value:

```java
public int add(int summand1, int summand2) {
    return summand1 + summand2;
}
```

## Calling Methods with a return value

```java
1    public class Calc {
2
3        public int add(int summand1, int summand2) {
4            return summand1 + summand2;
5        }
6
7        public static void main(String[] args) {
8            Calc calculator = new Calc();
9            int sum = calculator.add(3, 8);
10           System.out.print("3 + 8 = " + sum);
11           // prints: 3 + 8 = 11
12       }
13
14   }
15
```

# Constructors

```
1    public class Calc {
2
3        private int summand1;
4        private int summand2;
5
6        public Calc() {
7            summand1 = 0;
8            summand2 = 0;
9        }
10
11   }
12
```

A constructor gets called upon creation of the object

## Constructors with Arguments

```
1    public class Calc {
2
3        private int summand1;
4        private int summand2;
5
6        public Calc(int x, int y) {
7            summand1 = x;
8            summand2 = y;
9        }
10
11    }
12
```

```
1    [...]
2    Calc myCalc = new Calc(7, 9);
3
```

A constructor can have arguments as well!