Java

Object Oriented Programming

Hannes Ueck, Jakob Krude

25. November 2020

Java-Kurs

Overview

1. Arrays
    - Multi-Dimensional Array
2. OOP in Java
    - General information
    - Methods
    - Return Value
    - Constructor

Arrays

Array

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

```
1     public static void main(String[] args) {
2
3         int[] intArray = new int[10];
4         intArray[8] = 7; // assign 7 to the 9th element
5         intArray[9] = 8; // assign 8 to the last element
6
7         System.out.println(intArray[8]); // prints: 7
8     }
9
```

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.

```
1    public static void main(String[] args) {
2
3        int[] intArray = {3, 2, 7};
4
5        System.out.println(intArray[0]); // prints: 3
6        System.out.println(intArray[1]); // prints: 2
7        System.out.println(intArray[2]); // prints: 7
8    }
9
```

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.

```
1     public static void main(String[] args) {
2
3         // an array with 100 elements
4         int[][] intArray = new int[10][10];
5
6         intArray[0][0] = 0;
7         intArray[0][9] = 9;
8         intArray[9][9] = 99;
9     }
10
```

Assignment with Loops

Loops are often used to assign elements in arrays.

```
1     public static void main(String[] args) {
2
3         int[][] intArray = new int[10][10];
4
5         for(int i = 0; i < 10; i++) {
6             for(int j = 0; j < 10; j++) {
7                 intArray[i][j] = i*10 + j;
8             }
9         }
10    }
11
```

Arrays with objects

Loops are often used to assign elements in arrays.

```
1     public static void main(String[] args) {
2
3         Student[][] studentArray = new Student[10][10];
4
5         for(int i = 0; i < 10; i++) {
6             for(int j = 0; j < 10; j++) {
7                 intArray[i][j] = new Student();
8             }
9         }
10    }
11
```

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

```
1   public class Student {
2
3       private String name;
4
5       public String getName() {
6           return name;
7       }
8
9       public void setName(String newName) {
10          name = newName;
11      }
12
13  }
14
```

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

```
1   public class Student {
2
3       private String name;
4
5       public void setName(String newName) {
6           name = newName;
7           printName();   // Call own method
8           this.printName(); // Or this way
9       }
10
11      public void printName() {
12          System.out.println(name);
13      }
14
15  }
16
```

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

Methods with Arguments

```
1  public class Calc {
2
3      public void add(int summand1, int summand2) {
4          System.out.println(summand1 + summand2);
5      }
6
7      public static void main(String[] args) {
8          int summandA = 1;
9          int summandB = 2;
10         Calc calculator = new Calc();
11         System.out.print("1 + 2 = ");
12         calculator.add(summandA, summandB);
13         // prints: 3
14     }
15
16 }
17
```

Methods with Return Value

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

```
1     public void add(int summand1, int summand2) {
2         System.out.println(summand1 + summand2);
3     }
4
```

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

```
1     public int add(int summand1, int summand2) {
2         return summand1 + summand2;
3     }
4
```

Calling Methods with a return value

```
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!

Real-World

Model

```
class Car {


}
```

Real-World

Color

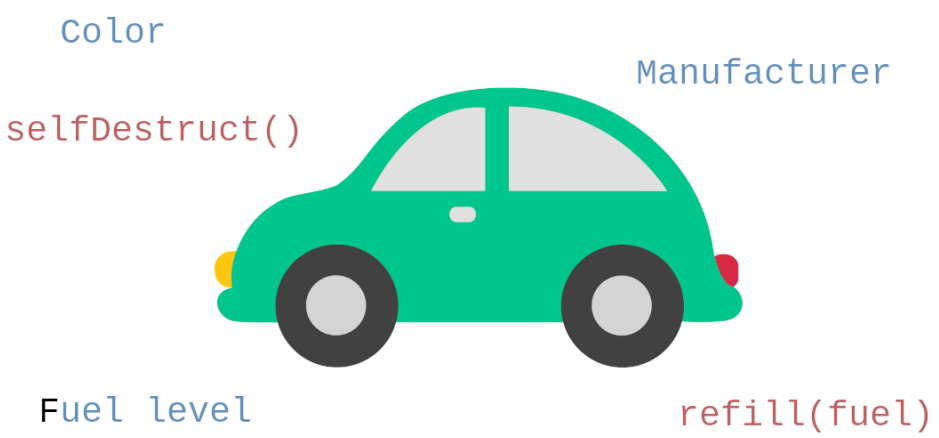
Manufacturer

Fuel level

Model

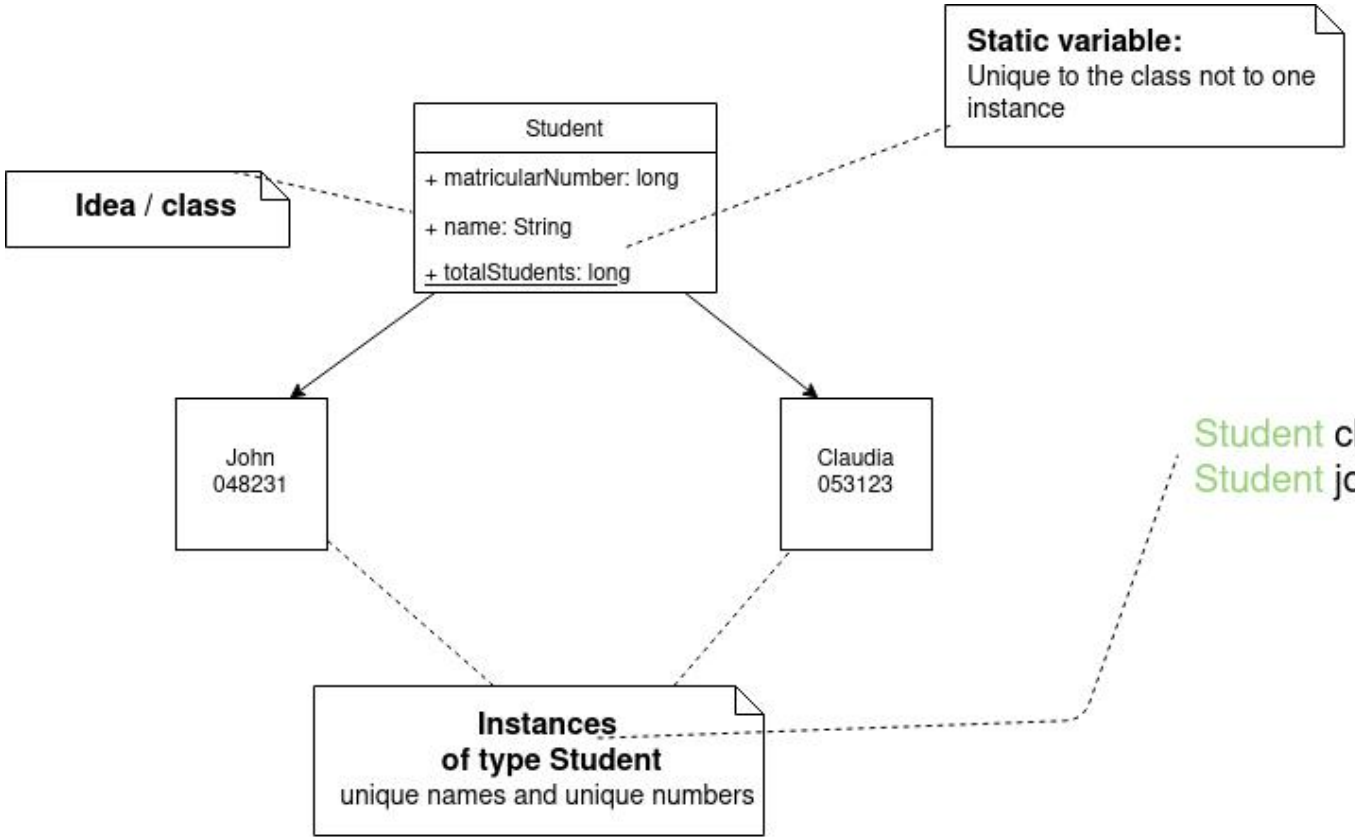
```
class Car {
    String Manufacturer;
    String color;
    double fuelLevel;
}
```

Real-World



Model

```
class Car {
    String Manufacturer;
    String color;
    double fuelLevel;
    void refill(double fuel){...};
    void selfDestruct(){...};
}
```




```
Student claudia = new Student("Claudia", 053123);
Student john = new Student("John", 048231);
```