Java

Control Statements and OOP

Julius Felchow (Mail - julius.felchow@mailbox.tu-dresden.de),
Benjamin Weller (Mail - benjamin.weller@tu-dresden.de)

13. November 2019

Java-Kurs

Overview

1. Recall last session
2. Functions
3. Control Statements
   - if then else
   - for
   - while
4. OOP in Java
   - General information
   - Methods
   - Return Value
   - Constructor
5. Conclusion
   - An Example

Recall last session

Conclusion

Data Types

- int, long
- float, double
- String

Hello World example.

Small introduction to functions.

Functions

```
1  public class Hello {
2
3      public static void main(String[] args) {
4          printingTest();
5      }
6
7      public static void printingTest() {
8          System.out.println("This is a printing Test!");
9      }
10 }
```

Prints "*This is a printing Test!*" to the console.

Functions

Parameter

```
1  public class Hello {
2
3      public static void main(String[] args) {
4          printParameter("Print this sentence!");
5          printParameter("Now print this!");
6      }
7
8      public static void printParameter(String print) {
9          System.out.println(print);
10     }
11 }
```

The main method gives the parameter "*Print this sentence!*" to the function "printParameter".

Function then prints "*Print this sentence!*" to the console and afterwards printParameter("Now print this!");

Return values

```
1  public class Hello {
2
3      public static void main(String[] args) {
4          System.out.println(getSentence());
5      }
6
7      public static String getSentence() {
8          return "Print this sentence!";
9      }
10 }
```

Get sentence returns "*Print this sentence!*" to the main function.

Prints "*Print this sentence!*" to the console.

Control Statements

Control Statements

- if, else, else if
- for
- while

If

```
if(condition) {
    // do something if condition is true
}
```

If Else

```
if(condition) {
    // do something if condition is true
} else {
    // do this instead, if condition is false
}
```

Conditions?

How to compare things:

- == Equal
- != Not Equal
- > Greater Than
- >= Greater or Equal than

Note: You can concatenate multiple conditions with && (AND) or || (OR)

If Then Else example

```
1  public class IteExample {
2      public static void main(String[] args) {
3          int input = /*here could be any int value*/;
4
5          if(input < 10) {
6              System.out.println("Input is smaller than 10");
7          } else if(input == 10) {
8              System.out.println("Input is 10");
9          } else {
10             System.out.println("Input is bigger than 10");
11         }
12     }
13 }
```

The else can also get a if, to specify a condition for the block.

for

```
for(initial value, condition, change) {
    // do code while condition is true
}
```

for example

```java
public class ForExample {

    public static void main(String[] args) {
        for(int i = 0; i < 16; i++) {
            System.out.print("na ");
        }
        System.out.println("BATMAN!");
    }
}
```

while

```
while(condition) {
    // do code while condition is true
}
```

while example

```
1  public class WhileExample {
2
3      public static void main(String[] args) {
4          boolean loopAgain = true;
5          String loop = "Lo";
6          while(loopAgain) {
7              System.out.println(loop);
8              loop = loop + "ooo";
9
10             if(loop.length() > 10) {
11                 loopAgain = false;
12                 loop = loop + "oop!";
13             }
14         }
15         System.out.println(loop);
16     }
17 }
```

What is the expected output?

OOP in Java

Object Oriented Programming

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
```

The creation of an object works similar.

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

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  }
```

The class *Student* has two methods: *void setName(String newName)* and *String getName()*.

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  }
```

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**:

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

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

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

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  }
```

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  }
```

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  }
```

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

A constructor can have arguments as well!

Conclusion

An Example

You want to program an enrollment system, for a programming course.

Your classes are:

student	who wants to attend the course
lesson	which is a part of the course
tutor	the guy with the bandshirt
room	where your lessons take place
...	

Class Student

```
1 public static void main(String[] args) {
2     Student peter = new Student();
3     peter.changeName("Peter");
4 }
```