

Java

Controll Statements & OOP

Max Langer, Leonard Follner, Alexander Hesse

24. Oktober 2016

Java-Kurs

Overview

1. Recalling last session

2. Control Statements

 if

 for

 while

3. OOP in Java

 General information

 Methods

 Return Value

 Constructor

4. Conclusion

 An Example

Recalling last session

Conclusion

Datatypes

- int, long
- float, double
- String

Hello World example

Control Statements

Control Statements

- if, else, else if
- for
- while

If Then Else

```
1  if(condition) {  
2      // do something if condition is true  
3  } else if(another condition){  
4      // do if "else if" condition is true  
5  } else {  
6      // otherwise do this  
7  }
```

If Then Else example

```
1 public class ItExample {  
2  
3     public static void main(String[] args) {  
4         int myNumber = 5;  
5  
6         if(myNumber == 3) {  
7             System.out.println("Strange number");  
8         } else if(myNumber == 2) {  
9             System.out.println("Unreachable code");  
10        } else {  
11            System.out.println("Will be printed");  
12        }  
13    }  
14  
15 }
```


Conditions?

How to compare things:

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

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

for

```
1  for(initial value, condition, change) {  
2      // do code while condition is true  
3  }
```

for example

```
1 public class ForExample {  
2  
3     public static void main(String[] args) {  
4         for(int i = 0; i <= 10; i++) {  
5             System.out.print("na ");  
6         }  
7         System.out.println("BATMAN!");  
8     }  
9  
10 }
```

while

```
1 while(condition) {  
2     // do code while condition is true  
3 }
```

while example

```
1 public class WhileExample {  
2  
3     public static void main(String[] args) {  
4         int a = 0;  
5         while(a <= 10) {  
6             System.out.println(a);  
7             a++; // Otherwise you would get an endless loop ∞  
8         }  
9     }  
10  
11 }
```

OOP in Java

Object Oriented Programming

Class Student

```
1  public class Student {  
2  
3      // Attributes  
4      private String name;  
5      private int matriculationNumber;  
6  
7  
8      // Methods  
9      public void setName(String name) {  
10         this.name = name;  
11     }  
12  
13     public int getMatriculationNumber() {  
14         return matriculationNumber;  
15     }  
16  
17 }
```


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

```
1 public class Main {  
2  
3     public static void main(String[] args) {  
4         Student example = new Student(); // creation  
5         example.setName("Jane"); // method call  
6         String name = example.getName();  
7         System.out.println(name); // Prints "Jane"  
8     }  
9  
10 }  
11
```

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 = x;  
9     }  
10  
11 }  
12
```

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

A constructor can have Arguments aswell!

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 }
```