Java

Controll Statements and OOP

Alexander Hesse, Leonard Follner, Max Langer

24. April 2017

Java-Kurs

Overview

1. Recalling last session
2. Controll Statements
   - Ite
   - 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

Controll Statements

Controll Statements

- if, else, else if
- for
- while

If Then Else

```
if(condition) {
    // do something if condition is true
} else if(another condition){
    // do if "else if" condition is true
} else {
    // otherwise do this
}
```

If Then Else example

```java
public class IteExample {

    public static void main(String[] args) {
        int myNumber = 5;

        if(myNumber == 3) {
            System.out.println("Strange number");
        } else if(myNumber == 2) {
            System.out.println("Unreachable code");
        } else {
            System.out.println("Will be printed");
        }
    }

}
```

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

```
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 <= 10; i++) {
            System.out.print("na ");
        }
        System.out.println("BATMAN!");
    }

}
```

while

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

while example

```java
public class WhileExample {

    public static void main(String[] args) {
        int a = 0;
        while(a <= 10) {
            System.out.println(a);
            a++; // Otherwise you would get an endless loop
        }
    }

}
```

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.

```
        int a; // declare a
        a = 273; // assign 273 to a

```

The creation of an object works similar.

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

```

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

```
    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
public class Calc {

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

    public static void main(String[] args) {
        Calc calculator = new Calc();
        int sum = calculator.add(3, 8);
        System.out.print("3 + 8 = " + sum);
        // prints: 3 + 8 = 11
    }

}
```

Constructors

```java
public class Calc {

    private int summand1;
    private int summand2;

    public Calc() {
        summand1 = 0;
        summand2 = 0;
    }

}

```

A constructor gets called upon creation of the object

Constructors with Arguments

```java
public class Calc {

    private int summand1;
    private int summand2;

    public Calc(int x, int y) {
        summand1 = x;
        summand2 = y;
    }

}

```

```java
[...]
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

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