

# Java

## Interfaces

---

Moritz Pflügner, Yannick Spörl

28. November 2018

Java-Kurs

- UML
- Abstract-Classes

# Abstract

```
1 public abstract class AbstractExample {
2
3     public void printHello() {
4         System.out.println("Hello");
5     }
6
7     public abstract String getName();
8 }
9
10 public class Example extends AbstractExample {
11
12     @Override
13     public String getName() {
14         return "Example";
15     }
16 }
```

## **Additional Control Structure**

---

```
1 public static void main (String[] args) {  
2  
3     int address = 2;  
4  
5     if (address == 1) {  
6         System.out.println("Dear Sir,");  
7     } else if (address == 2) {  
8         System.out.println("Dear Madam,");  
9     } else if (address == 4) {  
10        System.out.println("Dear Friend,");  
11    } else {  
12        System.out.println("Dear Sir/Madam,");  
13    }  
14 }  
15
```

# Differentiate with Switch

```
1 public static void main (String[] args) {  
2  
3     int address = 2;  
4  
5     switch(address) {  
6         case 1:  
7             System.out.println("Dear Sir,");  
8             break;  
9         case 2:  
10            System.out.println("Dear Madam,");  
11            break;  
12        case 4:  
13            System.out.println("Dear Friend,");  
14            break;  
15        default:  
16            System.out.println("Dear Sir/Madam,");  
17            break;  
18    }  
19 }  
20
```

# Differentiate with Switch

Depending on a variable you can switch the execution paths using the keyword **switch**. This works with `int`, `char` and `String`.

The variable is compared with the value following the keyword `case`. If they are equal the program will enter the corresponding case block. If nothing fits the program will enter the default block.

```
1      public static void main (String[] args) {  
2          switch(intVariable) {  
3              case 1:  
4                  doSomething();  
5                  break;  
6              default:  
7                  doOtherThings();  
8                  break;  
9          }  
10     }  
11
```

# Break

After the last command of the case block you can tell the program to leave using **break**.

Without **break** the program will continue regardless of whether a new case started, like in the example below.

```
1      public static void main (String[] args) {  
2  
3          switch( 1 ) {  
4              case 1:  
5                  System.out.println("enter case 1");  
6              case 2:  
7                  System.out.println("enter case 2");  
8                  break;  
9              default:  
10                 System.out.println("enter default case");  
11                 break;  
12          }  
13      }  
14
```



The keyword **break** also stops the execution of loops.

```
1  public static void main (String[] args) {  
2  
3      for (int i = 1; i < 10; i++) {  
4          System.out.println("i = " + i);  
5          if (i == 3) {  
6              break;  
7          }  
8      }  
9  }
```

The keyword **continue** jumps to the next loop step.

```
1  public static void main (String[] args) {  
2  
3      for (int i = 1; i < 10; i++) {  
4          if (i == 3) {  
5              continue;  
6          }  
7          System.out.println("i = " + i);  
8      }  
9  }  
10
```

Return statement gives back data

```
1  class Numbers {  
2      private int a = 4;  
3      private int b = 5;  
4  
5      public Number() {}  
6  
7      public int addNumbers() {  
8          return a + b;  
9      }  
10 }  
11  
12 ...  
13  
14 Numbers numbers = new Numbers();  
15 int return = numbers.addNumbers();  
16
```

Return works with every primitiv and complex data type.

```
1 public String getName() {  
2     return "Klaus";  
3 }  
4  
5 private Calculator calc;  
6 public Calculator getCalcualtor() {  
7     return calc;  
8 }  
9
```

Functions of type void do not have a return value. They are used for e.g. Setters

```
1 public void setNumber(int number) {  
2     this.number = number;  
3 }  
4
```

# Static

---

# Static Keyword

An object is an instance of a class with its attributes and methods. The object is the actor and the class just a blueprint.

Static class members are not linked to a certain instance of the class. Therefore the class can also be an actor.

Static class members are:

- static attributes, often called class variables
- static methods, often called class methods

# Class Variables

In the setter `count` is addressed via `Example.count`. Using `this.count` is misleading, because `count` is a class variable.

```
1  public class Example {  
2  
3      public static count;  
4  
5      public setCount(int count) {  
6          Example.count = count;  
7      }  
8  }  
9
```



# Class Variables - Test

The test prints the class variable `Example.count` which is altered by the different instances of the class *Example*.

```
1  public class ExampleTest {  
2  
3      public static void main (String[] args) {  
4          Example e1 = new Example();  
5          Example e2 = new Example();  
6  
7          e1.setCount(4);  
8          System.out.println(Example.count); // prints: 4  
9          e2.setCount(8);  
10         System.out.println(Example.count); // prints: 8  
11     }  
12 }  
13
```

# Class Methods

Static methods can be called without an object. They can modify class variables but not attributes (object variables).

```
1 public class Example {  
2  
3     public static count;  
4  
5     public static void setCount(int count) {  
6         Example.count = count;  
7     }  
8 }  
9
```

```
1 public static void main (String[] args) {  
2  
3     Example.setCount(4);  
4 }  
5
```

# Static is an One-Way

Methods from objects can:

- access attributes (object variables)
- access class variables
- call methods
- call static methods

Class methods can:

- access class variables
- call static methods

# Interfaces

---

An **interface** is a well defined set of constants and methods a class have to **implement**.

You can access objects through their interfaces. So you can work with different kinds of objects easily.

For Example: A post office offers to ship letters, postcards and packages. With an interface *Trackable* you can collect the positions unified. It is not important how a letter calculates its position. It is important that the letter communicate its position through the methods from the interface.

# Interface Trackable

An interface contains method signatures. A signature is the definition of a method without the implementation.

```
1  public interface Trackable {  
2  
3      public int getStatus(int identifier);  
4  
5      public Position getPosition(int identifier);  
6  }  
7
```

Note: The name of an interface often ends with the suffix *-able*.

# Letter implements Trackable

```
1  public class Letter implements Trackable {
2
3      public Position position;
4      private int identifier;
5
6      public int getStatus(int identifier) {
7          return this.identifier;
8      }
9
10     public Position getPosition(int identifier) {
11         return this.position;
12     }
13 }
14
```

The classes *Postcard* and *Package* also implement the interface *Trackable*.

# Access through an Interface

```
1  public static void main(String[] args) {  
2  
3      Trackable letter_1 = new Letter();  
4      Trackable letter_2 = new Letter();  
5      Trackable postcard_1 = new Postcard();  
6      Trackable package_1 = new Package();  
7  
8      letter_1.getPosition(2345);  
9      postcard_1.getStatus(1234);  
10 }  
11
```



# Two Interfaces

A class can implement multiple interfaces.

```
1 public interface Buyable {  
2  
3     // constant  
4     public float tax = 1.19f;  
5  
6     public float getPrice();  
7 }  
8
```

```
1 public interface Trackable {  
2  
3     public int getStatus(int identifier);  
4  
5     public Position getPosition(int identifier);  
6 }  
7
```

# Postcard implements Buyable and Trackable

```
1  public class Postcard implements Buyable, Trackable {
2
3      public Position position;
4      private int identifier;
5      private float priceWithoutVAT;
6
7      public float getPrice() {
8          return priceWithoutVAT * tax;
9      }
10
11     public int getStatus(int identifier) {
12         return this.identifier;
13     }
14
15     public Position getPosition(int identifier) {
16         return this.position;
17     }
18 }
19
```

# Access multiple Interfaces

```
1      public static void main(String[] args) {  
2  
3          Trackable postcard_T = new Postcard();  
4          Postcard postcard_P = new Postcard();  
5          Buyable postcard_B = new Postcard();  
6  
7          postcard_T.getStatus(1234);  
8          postcard_B.getPrice();  
9          postcard_P.getStatus(1234);  
10         postcard_P.getPrice();  
11     }  
12
```

postcard\_P can access both interfaces.

postcard\_T can access Trackable.

postcard\_B can access Buyable.