Java

Interfaces

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

9. Dezember 2019

Java-Kurs

Overview

Additional Control Structure

Differentiate

```
    public static void main (String[] args) {

        int address = 2;

        if (address == 1) {
            System.out.println("Dear Sir,");
        } else if (address == 2) {
            System.out.println("Dear Madam,");
        } else if (address == 4) {
            System.out.println("Dear Friend,");
        } else {
            System.out.println("Dear Sir/Madam,");
        }
    }

```

Differentiate with Switch

```
    public static void main (String[] args) {

        int address = 2;

        switch(address) {
            case 1:
                System.out.println("Dear Sir,");
                break;
            case 2:
                System.out.println("Dear Madam,");
                break;
            case 4:
                System.out.println("Dear Friend,");
                break;
            default:
                System.out.println("Dear Sir/Madam,");
                break;
        }
    }

```

Differentiate with Switch

Depending on a variable you can switch the execution paths using the keyword **switch**.

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.

```
    public static void main (String[] args) {
        switch(intVariable) {
            case 1:
                doSomething();
                break;
            default:
                doOtherThings();
                break;
        }
    }

```

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.

```java
    public static void main (String[] args) {

        switch( 1 ) {
            case 1:
                System.out.println("enter case 1");
            case 2:
                System.out.println("enter case 2");
                break;
            default:
                System.out.println("enter default case");
                break;
        }
    }

```

Break

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

```
    public static void main (String[] args) {

        for (int i = 1; 1 < 10; i++) {
            System.out.println("i = " + i);
            if (i == 3) {
                break;
            }
        }
    }

```

Continue

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

```
    public static void main (String[] args) {

        for (int i = 1; 1 < 10; i++) {
            if (i == 3) {
                continue;
            }
            System.out.println("i = " + i);
        }
    }

```

return

Return statement gives back data

```java
    class Numbers {
        private int a = 4;
        private int b = 5;

        public Number() {}

        public int addNumbers() {
            return a + b;
        }
    }

    ...

    Numbers numbers = new Numbers();
    int return = numbers.addNumbers();

```

Return works with every primitiv and complex data type.

return

```java
    public String getName() {
        return "Klaus";
    }

    private Calculator calc;
    public Calculator getCalcualtor() {
        return calc;
    }

```

return

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

```java
    public void setNumber(int number) {
        this.number = number;
    }

```

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.

```
    public class Example {

        public static count;

        public setCount(int count) {
            Example.count = count;
        }
    }

```

Class Variables - Test

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

```
    public class ExampleTest {

        public static void main (String[] args) {
            Example e1 = new Example();
            Example e2 = new Example();

            e1.setCount(4);
            System.out.println(Example.count); // prints: 4
            e2.setCount(8);
            System.out.println(Example.count); // prints: 8
        }
    }

```

Class Methods

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

```
    public class Example {

        public static count;

        public static void setCount(int count) {
            Example.count = count;
        }
    }

```

```
    public static void main (String[] args) {

        Example.setCount(4);
    }

```

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.

```java
public interface Trackable {

    public int getStatus(int identifier);

    public Position getPosition(int identifier);
}

```

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

Letter implements Trackable

```
public class Letter implements Trackable {

    public Position position;
    private int identifier;

    public int getStatus(int identifier) {
        return this.identifier;
    }

    public Position getPosition(int identifier) {
        return this.position;
    }
}

```

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

Access through an Interface

```
    public static void main(String[] args) {

        Trackable letter_1 = new Letter();
        Trackable letter_2 = new Letter();
        Trackable postcard_1 = new Postcard();
        Trackable package_1 = new Package();

        letter_1.getPosition(2345);
        postcard_1.getStatus(1234);
    }

```

Two Interfaces

A class can implement multiple interfaces.

```java
    public interface Buyable {

        // constant
        public float tax = 1.19f;

        public float getPrice();
    }

```

```java
    public interface Trackable {

        public int getStatus(int identifier);

        public Position getPosition(int identifier);
    }

```

Postcard implements Buyable and Trackable

```java
public class Postcard implements Buyable, Trackable {

    public Position position;
    private int identifier;
    private float priceWithoutVAT;

    public float getPrice() {
        return priceWithoutVAT * tax;
    }

    public int getStatus(int identifier) {
        return this.identifier;
    }

    public Position getPosition(int identifier) {
        return this.position;
    }
}

```

Access multiple Interfaces

```
    public static void main(String[] args) {

        Trackable postcard_T = new Postcard();
        Postcard postcard_P = new Postcard();
        Buyable postcard_B = new Postcard();

        postcard_T.getStatus(1234);
        postcard_B.getPrice();
        postcard_P.getStatus(1234);
        postcard_P.getPrice();
    }

```

postcard_P can access both interfaces.

postcard_T can access Trackable.

postcard_B can access Buyable.