

Modul 226a

Objektorientiert implementieren - Java

Dokumentation

Inhaltsverzeichnis

[1 Theorie 2](#_Toc526279437)

[1.1 Begriffe 2](#_Toc526279438)

[1.2 Anleitungen 5](#_Toc526279439)

[1.3 Rund um Datentypen 5](#_Toc526279440)

[1.4 Operatoren 6](#_Toc526279441)

[1.5 Dokumentation 7](#_Toc526279442)

[1.6 JUnit 7](#_Toc526279443)

[2 Codebeispiele 8](#_Toc526279444)

[2.1 Schleifen und Verzweigungen 8](#_Toc526279445)

[2.2 Examples 9](#_Toc526279446)

# 

# Theorie

## Begriffe

**Beziehungen**: Zwei Objekte müssen sich kennen, damit sie mit eineinander kommunizieren können. Man unterscheidet zwischen 3 Beziehungsarten:

**Assoziation**: Klasse A kennt Klasse B. Man

**Aggregation**: Klasse B enthält 1 oder mehrere Objekte der Klasse A

**Komposition**: Klasse B muss mindestens 1 Objekt der Klasse A haben.

**Dateihandeling**: **//** mehr dazu bei Codebeispielen

Erzeugen einer Datei:

File einFile = new File ("/home/neemo", "test.txt")

// Falls die Datei im gleichen Verzeichnis erstellt weren soll, den ersten Parameter weglassen.

Überprüfungen:

if (einFile.exists()) // File vorhanden?

if (einFile.isFile()) // Ist es eine Datei?

if (einFile.isDirectory()) // Ist es ein Verzeichnis?

Bearbeitungsmöglichkeiten:

renameTo()// umbenennen

delete()// löschen

mkdir()// erstellt ein Verzeichnis

list()// listet Verzeichnisinhalt auf

**Ein- und Ausgabe**:

System.in. **//** Standardeingabe

System.out. **//** Standardausgabe

System.err. **//** Standardausgabe für Fehlermeldungen

**Exception:** Mit einer Exception werden Probleme behandelt. Ist der Code in Ordnung, läuft der ganz normal ab. Ist jedoch ein Fehler aufgetreten, wird dieser mit einer Exception behandelt. Mehr dazu bei den Codebeispielen

**Instanz**: Ein "Alias" oder Variablenname für eine Klasse. Klassen können mehrere

Instanzen haben.

// Regisseur **Maersu** = new Regisseur ( );

// Regisseur **Neemo** = new Regisseur ( );

// Datentyp **Variable** = new Methode ( );

**Instanzvariablen** sollten immer "private" gesetzt werden, damit man sie von Ausserhalb nicht direkt Ansprechen kann. Die *Übergabe* eines Objektwerts soll immer "set" davor haben, eine zum *Abfragen* des Wertes sollte "get" davor haben.

**IV:** setAussage() **Ü:** setAussage(String neuerText) **A:** getAussage()

**Java Dokumentation**: Im vorinstallieren Programm "Zeal" ist alles dokumentiert. Im linken Fensterbereich kann man einfach suchen, was man gerade braucht.

**Klasse**: In einer Klasse sind Methoden und Eigenschaften eines Objektes definiert. Die Klasse dient als Vorlage, aus der dann beliebig viele Objekte erzeugt werden können.

**Klassen und Datentypen**: Klassen sind Datentypen. Eine Klasse beschreibt, wie sich Objekte verhalten und welche Eigentschaften sie haben.

**Konstanten** **//** Konstanten sind Variablen, deren Wert nicht mehr geändert werden können. Folgendermassen definiert man eine konstante Variable:

final String unsereNamen = "Marcel und Neemo";

**Konstruktor**: Ein Konstruktor enthält immer den gleichen Namen wie die Klasse. Die Methode Aussage wird Konstruktor genannt.

Konstruktor = gleicher Name wie Klasse & kein Rückgabewert wie void, etc.

public class Aussage {

public Aussage() {

aussage = "public Aussage ist ein Konstruktor."

}

}

**Main (-) - Methode**: Main (-) {…} dient als Einstiegspunkt für das Java Programm. Da es nur

**Methode**: Eine Methode ist im Grunde genommen eine Funktion. Sie sind Programmteile,

welche bestimmte Teilaufgaben lösen. Es wird gesagt, was jetzt genau passieren sollte.

**Objekt**: Objekte sind konkrete Instanzen von Klassen. Sie haben bestimmte Eigenschaften, definierte Fähigkeiteung und ein bestimmtes Verhalten.

**Pakete**: Der Java Sourcecode wird in Paketen zusammengefasst. Sie sollten kleingeschrieben werden. Pakete können wie ein Verzeichnis im Dateisystem aufgebaut werden, mit einem Punkt werden diese getrennt: Bsp: ch.modul226.krimi = krimi innerhalb des Paketes modul226 innerhalb des Paketes ch. Im Programm sieht das so aus: package ch.modul226.krimi;

Um ConsoleReader zu aufrufen zu können muss man diesen importieren:

import ch.modul226.utils.\*; Für alle Java Pakete, einfach in die Java Doku reinschauen.

**Parameter**: Ein Parameter ist ein Übergabewert. Bei der Deklaration einer Methode kann man so einen Wert "verlangen". Beim Ansprechen der Methode muss daher ein Parameter gesetzt werden:

String aussage = "Beispiel Text";

public void setAussage(String neuerText) {

aussage = neuerText;

System.out.println(aussage);

}

Regisseur.setAussage("Mit dem wird die Aussage ersetzt.")

OUTPUT: Mit dem wird die Aussage ersetzt.

**Parameterübergabe**:

**Pass by value (Wertübergabe)** ist der Prozess, bei dem man den Wert einer Variablen kopiert und diesen dann an eine Methode übergibt. In dieser Methode kann man den Wert der Variable beliebig ändern, ohne den ursprünglichen Wert zu verändern. Ausserhalb dieser Methode hat die Variable also immernoch denselben Wert.

Alle primitiven Datentypen werden in Java mit "Pass by value" übergeben.

**Pass by reference (Referenzaufruf)** ist der Prozess, bei dem anstatt einer Kopie des Wertes einer Variablen, die Verlinkung zum Originalwert übertragen wird. Mit dieser Methode wird auch der Originalwert verändert.

Pass by reference wird für alle Objekte in Java benutzt (Für primitive Datentypen kann man dies nicht verwenden)

Mit "this." macht man eine Referenz auf das Objekt, in dem wir uns gerade befinden. Dies hilft eigentlich nur für das Aufheben von Namensproblemen.

public void setGeschwindigkeit(int geschwindigkeit) {

this.geschwindigkeit = geschwindigkeit;

}

**Signatur einer Methode**: Wenn mehrere Methoden mit demselben Namen existieren, fügt der Compiler dem Methodennamen die Datentypen der Eingabeparameter automatisch hinzu.

Die Kombination aus Bezeichnung und Parameter wird auch Signatur einer Methode genannt.

Wenn der Compiler anhand der Signatur erkennen muss welche Methode gemeint ist spricht man von einer überladenen Methode.

int rechne(int zahl1, int zahl2); // Signatur == rechne\_int\_int

int rechne(int zahl1, double zahl2); // Signatur == rechne\_int\_double

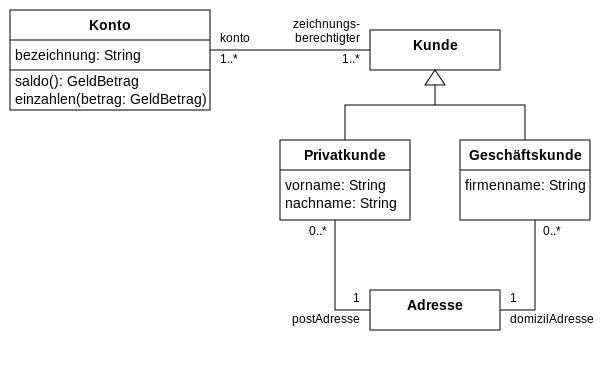
**UML (Unified Modeling Language)**: In objektorientierten Designs (OOD) werden UML's verwendet um eine Übersicht zu erhalten. UML kennt zahlreiche normierte Diagramme zur Darstellung statischer und dynamischer Beziehungen zwischen Klassen und Objekten. In einem ***Klassendiagramm*** werden alle Klassen aufgelistet, und welche Beziehung diese zu einander haben. Dasselbe gibt es auch für Objekte: das **Objektdiagramm**.

Abbildung 1 // Klassendiagramm

Arten der Beziehung:

1 // 1 // genau eine  
0,1 // c // konditionell, keine oder eine   
\* // mc // multiple, keine Einschränkung  
0... // mc // multiple  
1... // m // mindestens eine

**Zugriffsmodifikatoren**: Diese regeln, wer welche Zugriffsrechte auf eine Variable oder Methode hat. Entweder aus der gleichen Klasse, aus einer anderen Klasse im selben Paket oder aus einer abgeleiteten Klasse. Es gibt *private, public* und *protected.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Schlüsselwort** | **Klasse** | **Paket** | **Unterklasse** | **Alle anderen** |
| public | ja | ja | ja | ja |
| private ja | nein | nein | nein |  |
| protected | ja | ja | ja | nein |

## Anleitungen

**Neues Projekt mit Paket erstellen**:

Neues Java-Projekt **//** Projektname eingeben **//** Ansonsten keine Angaben

**Neue Klasse hinzufügen**:

Im Projekt: New > Class **//** Package eingeben **//** Name eingeben **//** Pro Projekt nur 1 Main!

Klassennamen haben folgende **Regeln**:

Beginn mit Grossbuchstaben, jede Klasse ist eine eigene Datei mit demselben Namen, Name darf keine Umlaute und Leerzeichen beinhalten (Zahen und $ erlaubt, aber nicht empfohlen).

**Programm ausführen**:

Debug As > Java Application oder oben in der Leiste auf den Käfer klicken.

**Hilfe aufrufen**:

Mittels Ctrl + Leerschlag ruft man die Codierhilfe auf.

## Rund um Datentypen

**Datentypen**:

byte **//** Ganzzahl (8bit)

short **//** Ganzzahl (16bit)

int **//** Ganzzahl (32bit) int Zahl = 17;

long **//** Ganzzahl (64bit) long Nummer = 2348739271;

float **//** Fliesskommazahl (32bit)

double **//** Fliesskommazahl (64bit) double pi = 3.14159;

boolean **//** true/false

char **//** einzelnes Zeichenchar zeichenP = "P";

String **//** Zeichenkette String wort = "Hallo"; Additionen mit Strings möglich

**Array** **//** Sammlung von Objekten

int [ ] vieleZahlen = { 1, 2, 4, 6, 35, 935, 1337 };

vieleZahlen = new int [ 420 ];

**Collections** **//** Collections sind dynamische Arrays (ArrayList)

ArrayList<Integer> vieleZahlen = new ArrayList<Integer>();

vieleZahlen.add(newInteger(1337)); // fügt 1337 in den Array vieleZahlen hinzu

Integer wert = vieleZahlen.get(3) // verlangt den vierten Wert im Array

System.out.println(zahlen.count()); // zählt alle Elemente im Array

Folgendes überprüft die Länge eines Arrays:

**for** (sohrt index = 0; index < vieleZahlen.length; index++) **{**

vieleZahlen[index] = index;

**}**

**Wrapperklassen**

Jede Wrapperklasse hat einen Konstruktor, welcher einen primitiven Datentyp als Parameter verlangt.

|  |  |
| --- | --- |
| Primitiver Datentyp | Wrapper-Klasse |
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| boolean | Boolean |
| char | Character |
| float | Float |
| double | Double |
| void | Void |

## Operatoren

**Operationszeichen & Operatoren**:

== **//** gleich

!= **//** ungleich

< **//** kleiner als

<= **//** kleiner oder gleich

> **//** grösser als

>= **//** grösser oder gleich

! **//** nicht

&& **//** UND

| | **//** ODER

## Dokumentation

**//** Einzeiliger Kommentar

**/\*** Mehrzeiliger Kommentar **\*/**

**/\*\***

**\*** JavaDoc-Kommentar

**\*** Bei einer Erstellung von Programmdokumentationen werden diese

**\*** Kommentare automatisch berücksichtigt.

**\***

**\*** *@author Name des Autors*

**\*** *@see Verweis "siehe auch…"*

**\*** *@version Version*

**\*** *@since Seit wann dies enthalten ist*

**\*** *@param Eingabeparameter einer Methode*

**\*** *@return Beschreibung des Rückgabewertes*

**\*** *@throws Beschreibung allfälliger Exceptions*

**\*/**

Um aus den Kommentaren eine Dokumentation zu erstellen, muss man folgendes machen.

In der Leiste auf **Project** und danach **Generate Javadoc**.

## JUnit

JUnit dient zum Testen anderer Methoden. Es wird eine eigene JUnit Methode erstellt, welche danach zur anderen Methode "weitergeleitet" werden muss. Pro JUnit Methode ist nur eine einzige Überprüfung möglich.

**Erstellen**: Rechtsklick auf die Klasse **//** Other **//** Java > JUnit **//** JUnit Test Case **//**

Next und die gewünschte Methode auswählen. Fertig.

**Ausführen**: Rechtsklick auf JUnit Methode **//** Run As **//** JUnit Test

Beispiel für einen Test:

public void testConvert() {

TemperaturRechner rechner = new TemperaturRechner();

double ergebnis = rechner.convert(37.5);

assertEquals(99.5, ergebnis, 0.0001);

}

# Codebeispiele

## Schleifen und Verzweigungen

**Bedinung | Auszuführender Code**

*int zahl = 3;*

**while** (zahl < 20) **{**

System.out.println("Aktuelle Zahl ist: " + zahl);

zahl++;

**}**

*int zahl = 4;*

**do {**

System.out.println("Aktuelle Zahl ist: " + zahl);

**}** **while** (zahl < 20);

(Initialisierung ; Bedingung ; Schritt)

**for** (int i = 0; i < 5; i++) **{**

System.out.println("Dies ist eine Schleife die 5 Mal ausgeführt wird");

**}**

*int[] vieleZahlen = { 1, 2, 4, 6, 35, 935, 1337 };*

**for** (int element : vieleZahlen) **{**

System.out.println("Alle Zahlen: " + element); **}**

*int zahl = 6;*

**if** (zahl < 5) **{**

System.out.println("Zahl ist kleiner als 5.");  
**}** **else if** (zahl > 5) **{**

System.out.println("Zahl ist grösser als 5.");  
**}** **else {**

System.out.println("Zahl ist 5.");  
**}**

System.out.println("Ihr Name: "+(name == null ? "Unbekannt" : name));

*int zahl = 7;*

**Switch** (zahl) **{**

case 6:

System.out.println("Die Zahl ist 6"); break;

case 7:

System.out.println("Die Zahl ist 7"); break;

default:

System.out.println("Falls nichts zutrifft kommt der Default");

**}**

## Examples

**Hello World**

System.out.println("Hello World!");

**Klasse Regisseur mit zwei Methoden**

public class Regisseur {

public void go() {

System.out.println("Klappe, die erste, Action!");

}

public void stop() {

System.out.println("Alles im Kasten, Szene beendet!");

}

} // Klasse Regisseur mit Methode *go* und *stop*

**Neuer Regisseur (Variablenname) + Ausführen der Methoden (Welche in der Klasse sind)**:

Regisseur Planissimo = new Regisseur(); // Dies kann man mit jeder Klasse machen.

Planissimo.go()

Planissimo.stop() // Ausführen Methode go und stop

**Methode mit Parameterübergabe**:

Int addieren(int zahl1, int zahl2) {

int ergebnis = zahl1 + zahl2;

return ergebnis;  
}

System.out.println("Das Ergebnis ist " + addieren(17, 4) );

**Neue Objekte erzeugen und Konstruktor erstellen**:

Double kommaZahl = new Double(23.785);

String name = new String("Perfektissimo");

Person Maersu = new Person()

**Exception-Handling**:

int zahl1 = ConsoleReader.reatInteger("Eingabe Zahl 1");

int zahl2 = ConsoleReader.reatInteger("Eingabe Zahl 2");

try {

int ergebnis = zahl2 / zahl1;

System.out.println("Das Ergebnis lautet: " + ergebnis);

}

catch (ArithmeticExeption ex) {

System.out.println("Es ist ein Fehler bei der Berechnung aufgetreten");

}

In der Dokumentation unter "java.lang" findest du alle Arten der Exceptions.

**Datei auslesen**:

File file = new File("C:\Users\neemo\Desktop\test.txt");

BufferedReader br = new BufferedReader(new FileReader(file));

String st;

while ((st = br.readLine()) != null)

System.out.println(st);

}

## Exception

package apraafh;

public class Test {

static class HungryException extends Exception{

public HungryException(String name) {

super(name);

}

}

public static void main(String[] args) {

int a = 0;

int b = 1;

int result = 0;

char[] letters = {'a','b','c'};

try {

problematicMethod();

System.out.println(letters[1]);

System.out.println(letters[4]);

result = b / a;

} catch (ArithmeticException e) {

System.out.println("Go away");

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Invalid");

} catch (HungryException e) {

System.out.println("vely wong");

}

System.out.println(result);

}

static void problematicMethod() throws HungryException{

throw new HungryException("I'm hungry");

}

## JUnit

package apraafh;

public class Calculator {

public int add(int a, int b) {

return a+b;

}

public int multiply(int a, int b) {

return a\*b;

}

public static void main(String[] args) {

Calculator calcy = new Calculator();

System.out.println(calcy.add(3, 5));

System.out.println(calcy.multiply(2, 7));

}

}

package apraafh;

import static org.junit.Assert.assertEquals;

import org.junit.runner.RunWith;

import org.junit.runners.Suite;

import org.junit.runners.Suite.SuiteClasses;

import org.junit.Test;

public class AllTests {

@Test

public void TestShouldAddNumbers() {

Calculator clacy = new Calculator();

assertEquals("negative+positive=neutral", 0, clacy.add(7, -7));

}

@Test

public void TestShouldMultiplyNumbers() {

Calculator clacy = new Calculator();

assertEquals("very wrong", 0, clacy.multiply(7, -7));

}

}