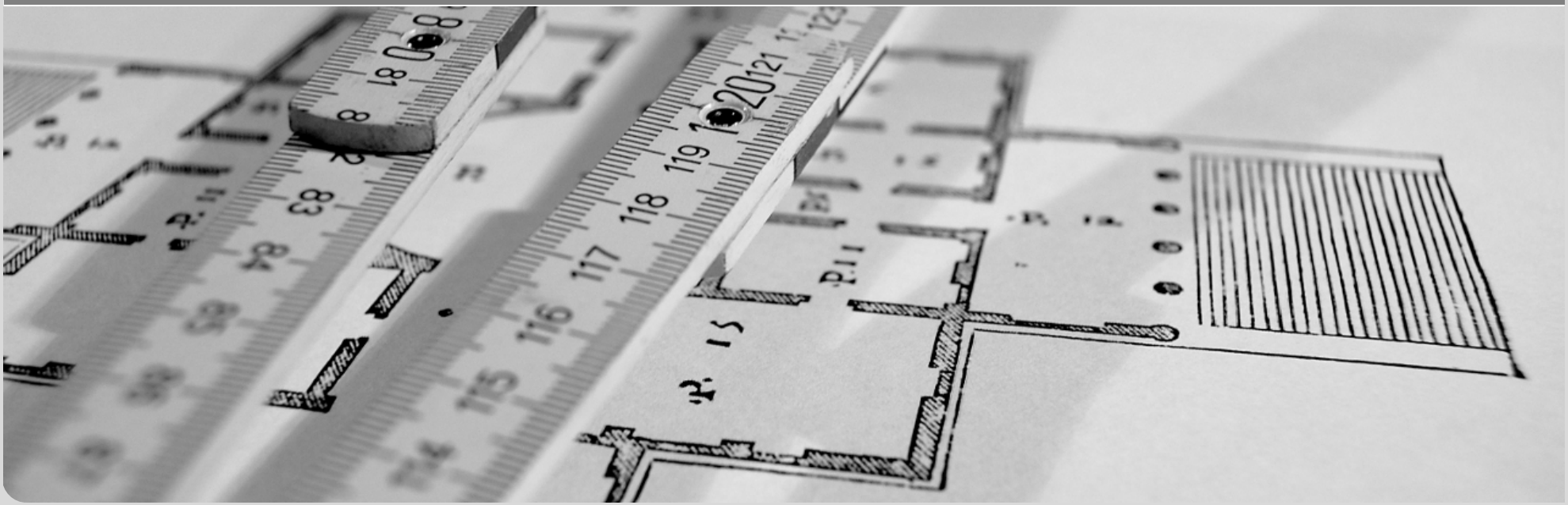
Karlsruher Institut für Technologie

Software Design and Quality

Programmieren
03. Tutorium

1. Übungsblatt, Diverses, Wiederholung, Methoden, Kontrollstrukturen, Übungen

Robin Rüde | 17. November 2014



Gliederung

1. Organisatorisches
   - Übungsblatt 1
   - Übungsblatt 2
2. Stoff
   - Diverses
   - Wiederholung
   - Methoden-Überladen
   - Main-Methode
   - Schleifen
     - break
     - continue
3. Aufgaben
   - Aufgabe 1: Konto
   - Aufgabe 2: Einmaleins
   - Aufgabe 3: Sinus

1

Organisatorisches

Organisatorisches

- Die Folien und Links sind *immernoch* auf `http://tutorium.studium.sexy`
- Anleitung fürs VPN (Praktomat von zuhause) im Ilias
- FAQ auf der Vorlesungswebseite

Übungsblatt 1

- Korrekturkriterien zum Übungsblatt großteils vorgegeben
  $\rightarrow$ Punktevergabe nicht unbedingt von mir bestimmt oder beliebig anpassbar
- Häufigster Fehler: Aufgabe nicht gelesen
- keine IDEs
- leere Main-Methoden sind unnötig
- Klassenattribute nicht in die main-Methode!
- Möglichst kein Denglisch
- Codestil! (Einrückungen, Variablennamen, ...), demnächst Abzug dafür

Übungsblatt 2

- Teil A: Begründung nicht vergessen! 9 mal ein Wort/eine Zahl und ein Satz.
- Teil B: Getter-Methoden sollen nichts tun außer einen Wert zurückgeben.
  Vier Dateien abgeben.
- Teil D: Annäherungen für π, nicht $\frac{\pi}{4}$ ausgeben

2

Stoff

Inkrement/Dekrement-operatoren

- postIncrement: `x++`
  - Erhöht x um eins
  - Gibt vorherigen x-Wert zurück
- preIncrement: `++x`
  - Erhöht x um eins
  - Gibt neuen x-Wert zurück

```
int x = 0;
System.out.println(x++); // 0
System.out.println(x);   // 1
int x = 0;
System.out.println(++x); // 1
System.out.println(x);   // 1
```

Escape-Sequenz

- Spezielle Zeichen sind in Java geschützt, z.B. " und ’
- Verwendung als Zeichen in Strings durch escapen mit \
- Zeilenumbruch: mit \n (zum Beispiel innerhalb von Strings)

Beispiel:

```
String s = "Fritz sagt:\n \"Hallo.\" ";
System.out.print(s);
```

Ausgabe:

```
Fritz sagt:
"Hallo."
```

Codestyle: Benennungsregeln

- Variablen - „sprechende“ Namen
  - camelCase: int year**O**f**M**anfuacture
  - Erster Buchstabe: klein
- Konstanten
  - CAPS und Unterstriche: `final int SECTOR_COUNT;`
- Klassen
  - camelCase: **H**ard**D**isk**D**rive
  - Ersters Buchstabe: groß
- enums: wie Klassen

Einrückungen und Klammern

```
class ClassName {
    void methodName(String s, String k) {
        // Code
        System.out.println("Bla");
        if (x) {
            System.out.println("Bla");
        } else {
            System.out.println("Bla");
        }
    }
}
```

Einrückungen und Klammern

```
class␣ClassName␣{
    ⇥void␣methodName(String␣s,␣String␣k)␣{
    ⇥   ⇥//␣Code
    ⇥   ⇥System.out.println("Bla");
    ⇥   ⇥if␣(x)␣{
    ⇥   ⇥   ⇥System.out.println("Bla");
    ⇥   ⇥}␣else␣{
    ⇥   ⇥   ⇥System.out.println("Bla");
    ⇥   ⇥}
    ⇥}
}
```

final, null und enums

final

final deklariert ein Attribut oder eine Variable als Konstante. Nach der Initialisierung kann sie dann **nicht** mehr geändert werden.

null

- Zuweisung an Objektvariable
- Variable zeigt dann auf **kein** Objekt
- kein Zugriff auf Methoden und Attribute (NullPointerException)

Enums

Definition:

```
enum <EnumName> { ELEM1, ELEM2, ... }
```

Verwendung: Aufzählungen

Zugriff: `<EnumName>.<ELEMENT>`

Wiederholung: Methoden

Aufruf von Objektmethoden

Methoden werden in einem konkreten Objektbezug aufgerufen, d.h.:

```
class Vehicle { void brake(int i) { /* do nothing */ }}
...
Vehicle v1 = new Vehicle();
v1.brake(5); // Aufruf von break mit 5 als Parameter.
```

Wiederholung: Getter und Setter

Verwendung

zum Setzen / Holen von Attributen,
zur Zugriffskontrolle von Attributen (z.B. nur getter für readonly)

Aufbau

getter:

```
<Type> get<Attribute>() { return this.<attribute>; }
```

setter:

```
void set<Attribute>(<Type> <Attribute>) {
    this.<Attribute> = <Attribute>;
}
```

Methoden überladen

Man kann Methoden überladen, d.h.

- mehrere Methoden mit identischem Namen
- aber unterschiedlicher Signatur
- Variation von
  - Anzahl der Parameter
  - Reihenfolge der Parameter
  - Typ(en) der Parameter
  - Rückgabetyp *nur bei eindeutigen Parametern*

Beispiel

```
double calculateArea(Circle c) {
    // Code
}
double calculateArea(Square s) {
    // Code
}
int calculateArea(int a, int b) {
    // calculates area of an integer rectangle
    // Code
}
```

Arrays (kurz)

- Arrays sind Sammlungen von Variablen des gleichen Typs.
- Erstellen mit

  ```
  typ[] array = new typ[];
  ```

- Zugreifen auf Elemente mit `array[index];`
- Sehr oft in Kombination mit Schleifen

Beispiel:

```
int[] squares = new int[10];
for(int i = 0; i < squares.length; i ++) {
    squares[i] = i * i;
}
```

Main-Methode

- Einstiegpunkt in Programm
- `String[] args` beinhaltet Liste der übergebene Argumente
- `java Programm parameter0 parameter1 ...`
- Zugriff in der main-Methode: `args[0]`

Beispiel:

```
public static void main(String[] args) {
    // Anzahl der Elemente in args muss genau 1 sein
    if (args.length != 1) {
        return;
    }
    System.out.println(args[0]); // Ausgabe des ersten Elemen
}
```

toString()-Methode

- Nutzung zur Ausgabe von Objekten
- (Object + String) $\Leftrightarrow$ (Object.toString() + String)

```
class Human {
    String prename, lastname;
    Human(String prename, String lastname) {
        this.prename = prename; this.lastname = lastname;
    }
    String toString() {
        return prename + " " + lastname;
    }

    public static void main(String[] args) {
        Human max = new Human("Max", "Mustermann");
        Human erika = new Human("Erika", "Mustermann");
        System.out.println(max + ", " + erika);
        // Max Mustermann, Erika Mustermann
    }
}
```

Schlüsselwort: static

- definiert Variable oder Methode als Klassenvariable/methode
- Zugriff über Klassenobjekt
- sollte **sparsam** verwendet werden.
- Aufruf: `<ClassName>.<MethodName>(<Parameters>)`
- Aufruf: `<ClassName>.<AttributName>`

Beispiel:

```
class Human {
    static int humanCount;

    int id = 0;

    Human() {
        this.id = Human.humanCount++;
    }
}
```

Ausdrücke vs. Anweisungen

- Ausdrücke
  - Haben einen Typen
  - sind beliebig schachtelbar (`func1(func2(x), func3(y) + z)`)
- Anweisungen
  - haben keinen Typen
  - nicht schachtelbar

```
int x = <ausdruck vom typ int>;
return <ausdruck vom returntypen der funktion>;
```

if-Bedingung

```
if(<bedingung>) {
    <anweisungen>;
} else if(<bedingung>) {
    <anweisungen>;
} [else if ...]
  else {
    <anweisungen>;
}
```

switch-Anweisung

```
switch (<ausdruck>) {
    case <wert1>:
        <anweisungen>;
        break;
    case <wert2>:
        <anweisungen>;
        break;
    [case ...]
    default:
        <anweisungen>;
}
```

break nicht vergessen!!

switch-Anweisung (Beispiel)

```
enum Color { RED, BLUE, GREEN }
...
String colorToRGB(Color c) {
    String rgb;
    switch (c) {
        case RED:
            rgb = "#ff0000";
            break;
        case GREEN:
            rgb = "#00ff00";
            break;
        case BLUE:
            rgb = "#0000ff";
            break;
        default:
            rgb = "unknown color";
    }
    return rgb;
}
```

Schleifen

```
while(<bedingung>) {
    <anweisungen>;
}

do {
    <anweisungen>
} while(<bedingung>);
// <anweisung> wird mind. einmal ausgefhrt.

for(<init>; <bedingung>; <ende>) {
    <anweisungen>;
}
```

break;

break; bricht eine Schleife ab

```
int find(int[] array, int n) {
    int i;
    for(i = 0; i < array.length; i++) {
        if(array[i] == n) {
            break;
        }
    }
    return i;
}
```

Was macht dieser Code?

continue;

continue; bricht die aktuelle Iteration einer Schleife ab

```
int filterSum(int[] array, int n) {
    int sum = 0;
    for(int i = 0; i < array.length; i++) {
        if(array[i] > n) {
            continue;
        }
        sum += array[i];
    }
    return sum;
}
```

Was macht dieser Code?

continue und break

Verwendung

Sowohl `continue` als auch `break` sollten sparsam bis gar nicht verwendet werden. Beide greifen in den Kontrollfluss ein und machen den Code deutlich schwerer lesbarer, da die Abbruchbedingungen der Schleifen verschleiert werden. Einzige Ausnahme: switch.

3

Aufgaben

Aufgabe: Konto (1/6)

Modelliere ein Bankkonto.
Ein Bankkonto hat einen ganzzahligen Kontostand.
Man kann Geld in das Bankkonto einzahlen und abheben.
Modelliere die Klasse so, dass die folgende main-Funktion funktioniert:

```
public static void main(String[] args) {
    BankAccount giro = new BankAccount();
    giro.deposit(1000);
    giro.withdraw(200);
    System.out.println(giro.moneyAmount);
    // Ausgabe: 800
}
```

Aufgabe: Konto (2/6)

Erweitere die Klasse Bankkonto um einen Konstruktor, der den initialen Kontostand angibt.

```
BankAccount giro = new BankAccount(100);
System.out.println(giro.moneyAmount);
// Ausgabe: 100
```

Aufgabe: Konto (3/6)

Außerdem soll man von einem Konto auf ein anderes wie folgt Geld überweisen können:

```
BankAccount giro = new BankAccount(1000);
BankAccount tagesgeld = new BankAccount(0);

giro.transfer(tagesgeld, 500);
System.out.println(tagesgeld.moneyAmount);
// Ausgabe: 500
```

Aufgabe: Konto (4/6)

Erweitere die Klasse Bankkonto um eine Kontonummer.
Die Kontonummer wird beim Konstruieren automatisch ausgewählt.
Das erste Konto hat die Kontonummer 0, das zweite die Kontonummer 1 usw.

```
BankAccount giro = new BankAccount();
BankAccount depot = new BankAccount();
BankAccount tagesgeld = new BankAccount();

System.out.println(tagesgeld.id);
// Ausgabe: 2
```

Aufgabe: Konto (5/6)

Erweitere die Klasse Bankkonto um einen Namen.
Schreibe Für das Attribut Name außerdem einen getter und einen setter.

```
BankAccount giro = new BankAccount(100);
giro.setName("Girokonto");

System.out.println(giro.getName());
// Ausgabe: Girokonto
```

Aufgabe: Konto (6/6)

Erweitere die Klasse Bankkonto um eine `toString()`-Methode.
Der Code

```
BankAccount giro = new BankAccount(100);
giro.setName("Girokonto");
BankAccount tagesgeld = new BankAccount(500);
tagesgeld.setName("Tagesgeld");

System.out.println(giro);
System.out.println(tagesgeld);
```

Soll die folgende Ausgabe erzeugen:

```
Girokonto: KontoNr: 0, Betrag: 100
Tagesgeld: KontoNr: 1, Betrag: 500
```

Falls ihr denkt ihr braucht vor dem 2. Blatt noch Übung / Feedback, könnt ihr die Aufgaben daheim nochmal bearbeiten / fertigstellen und mir per Mail schicken.

Aufgabe: Kleines Einmaleins

- Level 1:
  Schreibe ein Programm, das die Zahlen von 1 bis 10 ausgibt.
- Level 2:
  Schreibe ein Programm, das das Einmaleins in der Form `"x * y = z"` ausgibt, wobei x und y hintereinander alle Zahlen zwischen 1 und 10 sind.
- Level 3:
  Schreibe ein Programm, das das Einmaleins in Tabellenform ausgibt
- Level 4:
  Schreibe ein Programm, das das Einmaleins als korrekt ausgerichtete Tabelle mit | als Rändern ausgibt

Aufgabe: Sinus

- Level 1:
  Schreibe eine Schleife, die mithilfe der Funktion `Math.sin` die Sinuswerte aller Zahlen von 0 bis 2*PI in `0.1`-Schritten ausgibt.
- Level 2:
  Passe den Wertebereich der Sinuskurve so an, dass die Ausgabe nur Zahlen zwischen 0 und 30 enthält.
- Level 3:
  Schreibe ein Programm, das eine Sinuskurve in die Konsole malt.

Ende

Fragen?

Ende

Fragen?

Vielen Dank für eure Aufmerksamkeit!