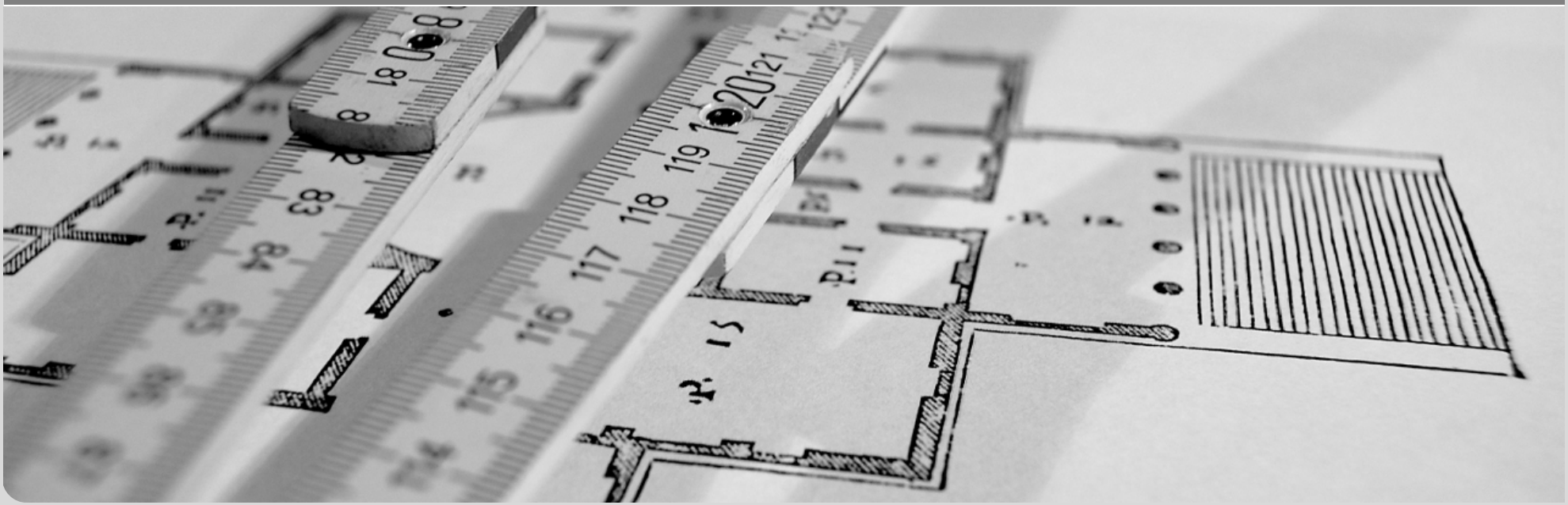
Karlsruher Institut für Technologie

Software Design and Quality

Programmieren
02. Tutorium

Organisatorisches, Enum, Variablen und Objekte, Konstruktoren, Methoden, Bedingungen, Schleifen

Robin Rüde | 10. November 2014



Gliederung

1

Organisatorisches

Organisatorisches

Materialien

`http://tutorium.studium.sexy`

Meine Mailadresse, diese Folien, Links, Code

Haben alle:

- den Disclaimer abgegeben?

Organisatorisches

Materialien

`http://tutorium.studium.sexy`

Meine Mailadresse, diese Folien, Links, Code

Haben alle:

- den Disclaimer abgegeben?
- sich einmal in den Praktomaten eingeloggt?

Organisatorisches

Materialien

`http://tutorium.studium.sexy`

Meine Mailadresse, diese Folien, Links, Code

Haben alle:

- den Disclaimer abgegeben?
- sich einmal in den Praktomaten eingeloggt?
- das Übungsblatt abgegeben?

Organisatorisches

Materialien

`http://tutorium.studium.sexy`

Meine Mailadresse, diese Folien, Links, Code

Haben alle:

- den Disclaimer abgegeben?
- sich einmal in den Praktomaten eingeloggt?
- das Übungsblatt abgegeben?
- 1. Hinweis: im Praktomaten Teil A unter Teil A, Teil B unter Teil B und Teil C unter Teil C hochladen
- 2. Hinweis: bei jedem Teil immer **ALLE** Dateien hochladen

Bearbeitung der Übungsblätter

Es darf mehr gemacht werden, beachtet aber:

- keine Bonuspunkte
- für falsches gibt es trotzdem Punktabzug

2

Enum

Enum/Aufzählungen: Verwendung und Syntax

- Aufzählungen
- Klasse: java.lang.Enum
- Zugriff auf die Elemente: `<Enum-Name>.<Element>`
- enums in eigene Java-Datei mit dem Namen Name.java (wie auch Klassen)

Enum/Aufzählungen: Verwendung und Syntax

- Aufzählungen
- Klasse: java.lang.Enum
- Zugriff auf die Elemente: `<Enum-Name>.<Element>`
- enums in eigene Java-Datei mit dem Namen Name.java (wie auch Klassen)

Syntax:

```
enum Name { // Signalwort enum gefolgt vom Namen des enums
        ELEM1, ELEM2, ..., ELEMN;
}
```

Beispiel

Beispiel: Aufzählung von Farben

`Color.java`

```
enum Color {
        RED, GREEN, BROWN, BLACK;
}
```

- `Color c = Color.RED`
- um `Color.RED` in c zu speichern

3

Variablen und Objekte

Spezielle Werte

Über- und Unterläufe

Wird bei einer Berechnung der Wertebereich verlassen, so wird am oberen bzw. unteren Ende weitergezählt.

```
2.147.483.647 + 1 =  - 2.147.483.648
```

(dies sind die Intervallgrenzen von int-Werten)

Spezielle Werte

Über- und Unterläufe

Wird bei einer Berechnung der Wertebereich verlassen, so wird am oberen bzw. unteren Ende weitergezählt.

`2.147.483.647 + 1 = - 2.147.483.648`

(dies sind die Intervallgrenzen von int-Werten)

Fließkommazahlen:

- NaN - Not a number (Zum Beispiel negative Wurzel)
- `POSITIVE_INFINITY`: Entsteht beim Teilen durch 0 (5 / 0)
- `NEGATIVE_INFINITY`: z.B. bei log(0)
- Es gibt eine positive und negative Null (z. Bsp bei Unterläufen kann sie auftreten)
- Gleichheit von Fließkommazahlen:
  `Math.abs(x-y) < 0.000000001`

Speicher (Tafel)

Achtung

Vereinfachtes Modell! Jeder Datentyp braucht unterschiedlich viele Speicherzellen.

- Primitive Datentypen
  - Wert steht jeweils direkt in einer eigenen Speicherzelle
- Objekte
  - Variable beinhaltet indirekte Referenz auf erstes Attribut im Speicher
  - Attribute liegen ab dem ersten nacheinander im Speicher

```
class Point { int x; int y; ...}
... void main() {
        int a = 5;
        Point p = new Point(0,10);

        int b = a;
        Point q = p;
}
```

Speicher (Tafel)

```
class Point { int x; int y; ...}
... void main() {
        int a = 5;
        Point p = new Point(0,10);
        int b = a;
        Point q = p;


        a = a + 1;
        p.x = p.x + 1;

        System.out.println(b);
        System.out.println(q.x);
}
```

Konstanten

- analog zur Variablendeklaration.
- Schlüsselwort `final`
- Wert kann nach Initialisierung nicht mehr geändert werden
- Beispiel: `final double pi = 3.14159;`

null

- Objektvariablen können den Wert `null` annehmen
- Variable zeigt auf kein Objekt
- -> kein Zugriff auf Methoden und Attribute!
- `Vehicle v1 = null;`
- `v1 = new Vehicle();` um dann v1 ein neues Objekt zuzuweisen
- oder eben `v1 = v2` wenn v2 ein Objekt vom Typ Vehicle ist.

null 2

```
public class Ausgabe {

        int get42() {
                return 42;
        }

        public static void main(String[] args) {
                Ausgabe a = null;
                System.out.println(a.get42());
        }
}
```

Was passiert hier?

null 2

```
public class Ausgabe {

        int get42() {
                return 42;
        }

        public static void main(String[] args) {
                Ausgabe a = null;
                System.out.println(a.get42());
        }
}
```

Was passiert hier?

- Compiler-Fehler?

null 2

```
public class Ausgabe {

        int get42() {
                return 42;
        }

        public static void main(String[] args) {
                Ausgabe a = null;
                System.out.println(a.get42());
        }
}
```

Was passiert hier?

- Compiler-Fehler?
- Laufzeit-Fehler?

null 2

```
public class Ausgabe {

        int get42() {
                return 42;
        }

        public static void main(String[] args) {
                Ausgabe a = null;
                System.out.println(a.get42());
        }
}
```

Was passiert hier?

- Compiler-Fehler?
- Laufzeit-Fehler?
- Kein Fehler?

4

Konstruktoren und Methoden

Methoden: Syntax

```
<ReturnType> methodName(<Typ> param1, <Typ> param2, ...) {
        // method code
        return returnValue;
}
```

- `ReturnType` ist ein Datentyp oder `void`
- Name von Methoden starten üblicherweise mit Kleinbuchstaben und dann camelCase (genau wie Variablen)
- Parameter wie bei Konstruktoren
- Rückgabe im Rumpf mit Schlüsselwort `return` gefolgt vom zurückzugebenden Wert/Variable
- oder komplett ohne `return` wenn der Rückgabetyp `void` ist.

Void

- `void` signalisiert keine Rückgabe (“leer“)

Wieso ist so etwas sinnvoll?

Void

- `void` signalisiert keine Rückgabe (“leer“)

Wieso ist so etwas sinnvoll?

Zustandsverändernde Methoden ohne Rückgabe (e. g. Setter) oder reine Ausgabemethoden.

Beispiel:

```
void printText(String text) {
        System.out.println(text);
}
```

Aufruf von Methoden

- Objektmethoden haben Bezug zu einem bestimmten Objekt
- Aufruf: `ObjektName.methodenName(Parameter);`

```
class Call {

        public void main(String[] args) {
                Call c = new Call();
                double doubleVal = c.callMe(5.5);
                c.printMe(doubleVal);
        }

        double callMe(double val) {
                return 2 * val;
        }

        void printMe(double val) {
                System.out.println(val);
        }
}
```

Konstruktoren

sind spezielle Methoden, die

- Instanzen von Klassen erzeugen
- bei Erzeugung eines Objektes mit `new` aufgerufen werden
- den gleichen Namen wie die Klasse haben
- keinen Rückgabetyp haben
- aber Parameter haben können

Syntax

```
<ClassName>(<Typ> param1, <Typ> param2, ...) {
        // do some stuff
        // (e. g. set attributes or call other methods)
}
```

Syntax

```
<ClassName>(<Typ> param1, <Typ> param2, ...) {
        // do some stuff
        // (e. g. set attributes or call other methods)
}
```

Was passiert in der main-Methode?

```
class Example1 {
        String text;

        Example1(String text1) {
                text = text1;
        }

        public static void main(String[] args) {
                Example1 ex = new Example1("Hallo Welt!");
                System.out.println(ex.text);
        }
}
```

Mehrere Konstruktoren?

Können mehrere Konstruktoren definiert werden?

Mehrere Konstruktoren?

Können mehrere Konstruktoren definiert werden?

Ja!

Aber: Müssen dann unterschiedliche Signaturen haben.

Signatur

Besteht aus:

- Name der Methode
- Anzahl der Parameter
- Reihenfolge der Parameter
- Typen der Parameter
- Rückgabetyp

Welche der genannten Punkte können bei einem Konstruktor variiert werden?

Beispiel

Was wird jeweils aufgerufen?

```
class Example2 {
        String text;

        Example2(String text1) {
                text = text1;
        }

        Exmaple2() {
                this("Leerer Konstruktor");
        }
        public static void main(String[] args) {
                Example2 ex = new Example2("Hallo Welt!");
                System.out.println(ex.text);
                ex = new Example2();
                System.out.println(ex.text);
        }
}
```

Defaultkonstruktor

- Jede Klasse hat mindestens einen Konstruktor
- wenn kein Konstruktor angegeben wird, wird implizit `NamederKlasse() { }` verwendet
- sobald ein Konstruktor definiert wurde, gibt es den Defaultkonstruktor nicht mehr!

Werte der Attribute wenn bei Aufruf des Konstruktors nicht gesetzt:

boolean	false
byte, short, int	0
long	0L
float	0.0f
double	0.0
char	’u0000’
Objekt-Referenz	null

Schlüsselwort: this

- Verwendung innerhalb von Methoden
- Referenziert die aktuelle Instanz einer Klasse (= aktuelles Objekt)

```
class ExampleThis {
        int x;
        String abc;

        ExampleThis(int x, String abc) {
                this.x = x;
                this.abc = abc;
        }

        setX(int x) {
                this.x = x;
        }
}
```

Getter und Setter-Methoden

- dienen der Zugriffskontrolle
- normale Methoden mit der Aufgabe ein Attribut zurückzugeben oder zu ändern
- Getter: geben Wert eines Attributs zurück
  - meist: <AttributTyp> getAttributName()
  - Ausnahme bei boolean: <AttributTyp> isAttributName();
- Setter (setzen einen Wert): setAttributName(<AttributTyp> name);

```
class Vehicle {
        int wheelCount;
        // ... more code
        int getWheelCount() {
                return wheelCount;
        }

        void setWheelCount(int count) {
                this.wheelCount = count;
        }
}
```

Aufgabe 1

Schreibt eine Klasse `Square`, die 2 Koordinaten speichert. Die Koordinaten sind Objekte der Klasse `Point`. Die Klasse `Square` soll

- 2 Methoden besitzen, die
  1. die Seitenlänge berechnen
  2. die Fläche des Quadrats berechnen
- 2 Konstruktoren besitzen
  1. mit zwei `Points` als Parameter (links oben und rechts unten).
  2. mit einem Punkt (links oben) und der Seitenlänge als Parameter

Die Klasse `Point` besitzt zwei Integer Attribute mit den Namen x und y. Sie besitzt weiterhin einen Konstruktor, der als erstes die x-Koordinate und als zweites die y-Koordinate erhält.
Schreibt zusätzlich in der Klasse `Square` eine main-Methode, die beide Konstruktoren benutzt und die Seitenlänge, Fläche und alle Koordinaten ausgibt.

5

Bedingungen und Schleifen

if - else

```
if (condition) {
        // called when condition == true
} else {
        // called when condition is false
}
// Beispiel
int i = 5;
int z = 11;
if (i > 5 && z < 10) {
        i--;
} else {
        z--;
}
```

- Bedingte Verzweigungen
- condition: boolescher Ausdruck

while-Schleife

- einfache Schleife
- Anweisung(en) werden ausgeführt, bis `condition` das erste Mal zu `false` auswertet.
- Prüfung der Bedingung nur VOR dem Durchlaufs
- Syntax:

```
while(condition) {
        // instructions
}
```

for-Schleife

- init: Ausführung vor dem ersten Durchlauf
- Condition: Prüfung vor jedem Schleifendurchlauf
- Count: Ausführung nach jedem Durchlauf item meist für Zählschleifen

```
for(init; Condition; After) {
        // instructions
}
```

While vs. For

```
while (liste.hasElements() {
        liste.removeLast();
}
doAnotherThing();

for(int z = 0; z < 10; z++) {
        doSomething();
        // hier ist z deklariert
}
// hier nicht
doAnotherThing();
```

Aufgabe 2

Gerade Zahlen

Schreibe ein Programm, dass alle geraden Zahlen von 1 bis 20 zwei Mal ausgibt.
Nutze dazu bei der ersten Ausgabe eine while-Schleife und bei der zweiten Ausgabe eine for-Schleife.

Lösung zu Aufgabe 2

```
class PrintEvenNumbers {
        public static void main(String[] args) {
                int i = 1;
                while (i <= 20) {
                        if (i % 2 == 0) { //  modulo 2
                                System.out.println(i);
                        }
                        i++;
                }
                System.out.println("----------------");
                for(i = 2; i <=20; i = i + 2) {
                        System.out.println(i);
                }
        }
}
```

6

Fragen und Kritik

Ende

Fragen...

- zum Stoff?
- zum Übungsblatt?
- sonstiges?

Ende

Fragen...

- zum Stoff?
- zum Übungsblatt?
- sonstiges?

Kritik? (gerne auch per Mail oder persönlich :))

Ende

Fragen...

- zum Stoff?
- zum Übungsblatt?
- sonstiges?

Kritik? (gerne auch per Mail oder persönlich :))

Danke für eure Aufmerksamkeit! :)