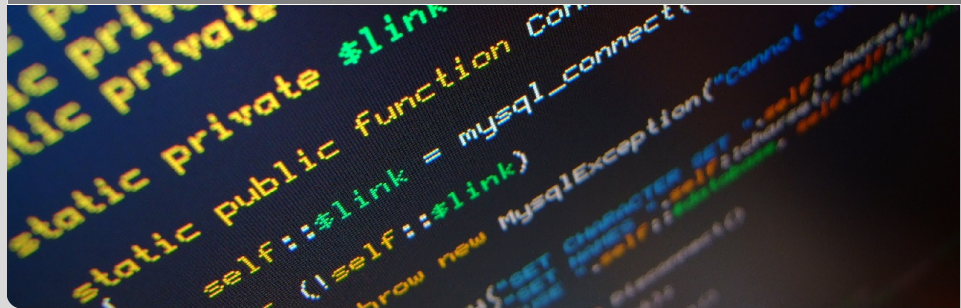


# Tutorium Programmieren

Tut Nr.2: Objekte und Methoden

Michael Friedrich | 5. / 7.11.2013

INSTITUT FÜR THEORETISCHE INFORMATIK



- 1 Variablen
  - Variablenoperationen
- 2 Klassen
  - Erzeugung von Objekten
- 3 "Verhalten"modellieren
  - Methoden
- 4 Code Conventions
- 5 Zum Übungsblatt 1
  - Hinweise
- 6 Hinweise zum Praktomat

## Was ist eine Variable?

Eine Variable ist ein Bezeichner ( $\equiv$  Name) für einen Speicherplatz im Rechner.

## Variablen in Java - Beispiel:

```
int meineZahl; //Deklariert eine int – Variable
meineZahl = 16; //setzt sie auf die Zahl 16
boolean istWahr = true; //Deklariert eine boolean – Variable
//und weist ihr sofort einen Wert zu
meineZahl = 42; //aendert den Wert von 'meineZahl'
```

## Was ist eine Variable?

Eine Variable ist ein Bezeichner ( $\equiv$  Name) für einen Speicherplatz im Rechner.

## Variablen in Java - Beispiel:

```
int meineZahl; //Deklariert eine int – Variable
meineZahl = 16; //setzt sie auf die Zahl 16
boolean istWahr = true; //Deklariert eine boolean – Variable
//und weist ihr sofort einen Wert zu
meineZahl = 42; //aendert den Wert von 'meineZahl'
```

## Variablen in Java - Allgemein:

- **Deklaration** - Namen & Typ festlegen:  
*Typ Variablenname;*
- **Initialisierung** - erstmalige Zuweisung eines Werts:  
*Variablenname = Wert;*
  - gleichzeitige Deklaration & Initialisierung:  
*Typ Variablenname = Wert;*
- **Zuweisung** - Setzen eines Wertes:  
*Variablenname = Wert*

**Attribute** sind Variablen auf Klassenebene.

## Variablen in Java - Allgemein:

- **Deklaration** - Namen & Typ festlegen:  
*Typ Variablenname;*
- **Initialisierung** - erstmalige Zuweisung eines Werts:  
*Variablenname = Wert;*
  - gleichzeitige Deklaration & Initialisierung:  
*Typ Variablenname = Wert;*
- **Zuweisung** - Setzen eines Wertes:  
*Variablenname = Wert*

**Attribute** sind Variablen auf Klassenebene.

Typ	Erklärung	Beispiel-Werte
<b>boolean</b>	<i>Wahrheitswerte</i>	<b>true, false</b>
<b>char</b>	<i>16-Bit-Unicode-Zeichen</i>	'A', '\n', '\u05D0'
<b>byte</b>	<i>8-Bit-Ganzzahl</i>	12
<b>short</b>	<i>16-Bit-Ganzzahl</i>	12
<b>int</b>	<i>32-Bit-Ganzzahl</i>	12
<b>long</b>	<i>64-Bit-Ganzzahl</i>	12L, 141
<b>float</b>	<i>32-Bit-Gleitpunktzahlen</i>	9.81F, 0.379E-8F, 2f
<b>double</b>	<i>64-Bit-Gleitpunktzahlen</i>	9.81, 0.379E-8, 3e1

# Wertebereiche der elementaren Datentypen

Typ	kleinster Wert	größter Wert
<b>char</b>	\u0000 (0)	\uFFFF (65.535)
<b>byte</b>	-128	127
<b>short</b>	-32.768	32.767
<b>int</b>	-2.147.483.648	2.147.483.647
<b>long</b>	-9.223.372.036.854.775.808	9.223.372.036.854.775.807
<b>float</b>	$-3.4028235 \cdot 10^{38}$	$3.4028235 \cdot 10^{38}$
<b>double</b>	$-1.7976931348623157 \cdot 10^{308}$	$1.7976931348623157 \cdot 10^{308}$



## (die wichtigsten) Operationen mit Zahlen

- $x + y$ ,  $x - y$ ,  $x * y$ ,  $x / y$ : (fast) wie aus der Mathematik bekannt
- $x \% y$ : Modulo (Rest der Division von  $x$  und  $y$ )

→ liefern wieder Zahlen zurück

- $x == y$ ,  $x != y$ : Gleichheit, Ungleichheit
- $x > y$ ,  $x < y$ ,  $x >= y$ ,  $x <= y$ : größer/kleiner (gleich)

→ liefern boolesche Werte (true oder false) zurück

## Konkatenation von Zeichenketten

- ***string1 + string2*** hängt beide Strings aneinander

→ liefert wieder einen String zurück

## (die wichtigsten) Operationen mit Zahlen

- $x + y$ ,  $x - y$ ,  $x * y$ ,  $x / y$ : (fast) wie aus der Mathematik bekannt
- $x \% y$ : Modulo (Rest der Division von  $x$  und  $y$ )

→ liefern wieder Zahlen zurück

- $x == y$ ,  $x != y$ : Gleichheit, Ungleichheit
- $x > y$ ,  $x < y$ ,  $x >= y$ ,  $x <= y$ : größer/kleiner (gleich)

→ liefern boolesche Werte (true oder false) zurück

## Konkatenation von Zeichenketten

- *string1* + *string2* hängt beide Strings aneinander

→ liefert wieder einen String zurück

## (die wichtigsten) Operationen mit Zahlen

- $x + y$ ,  $x - y$ ,  $x * y$ ,  $x / y$ : (fast) wie aus der Mathematik bekannt
- $x \% y$ : Modulo (Rest der Division von  $x$  und  $y$ )

→ liefern wieder Zahlen zurück

- $x == y$ ,  $x != y$ : Gleichheit, Ungleichheit
- $x > y$ ,  $x < y$ ,  $x >= y$ ,  $x <= y$ : größer/kleiner (gleich)

→ liefern boolesche Werte (true oder false) zurück

## Konkatenation von Zeichenketten

- ***string1 + string2*** hängt beide Strings aneinander

→ liefert wieder einen String zurück

## Operationen mit Wahrheitswerten

- **!a** : NOT
- **a && b**: logisches AND
- **a || b**: logisches OR
- **a ^ b**: XOR (exklusives OR)  
→ liefern wieder boolesche Werte zurück

## Vorsicht bei der Division!

Die Division zweier Ganzzahlen ergibt wieder eine Ganzzahl.

## Beispiel:

```
int w = 5 / 2; //in w steht nun 2
double x = 5 / 2 //in x auch! (genauer gesagt 2.0)
double y = 5 / 2.0 //hier kommt nun wirklich 2.5 raus
double z = 5 / 2d //hier auch
```

⇒ bei Fließkommadivisionen muss mindestens ein Operand explizit als Fließkommazahl ausgewiesen werden.

## Vorsicht bei der Division!

Die Division zweier Ganzzahlen ergibt wieder eine Ganzzahl.

## Beispiel:

```
int w = 5 / 2; //in w steht nun 2
double x = 5 / 2 //in x auch! (genauer gesagt 2.0)
double y = 5 / 2.0 //hier kommt nun wirklich 2.5 raus
double z = 5 / 2d //hier auch
```

⇒ bei Fließkommadivisionen muss mindestens ein Operand explizit als Fließkommazahl ausgewiesen werden.

## Beispiel

```
class Student {  
  
    String name;  
    int semester;  
    int matriculationNumber;  
  
    //Ein Konstruktor fuer den Studenten  
    Student(String name, int semester, int matriculationNumber) {  
        this.name = name;  
        this.semester = semester;  
        this.matriculationNumber = matriculationNumber;  
    }  
}
```

```
class Main {  
    public static void main(String[] args) {  
        //einen Studenten mit unserem Konstruktor erzeugen  
        Student maxMustermann = new Student("Max Mustermann", 3, 1234567);  
        // ...  
    }  
}
```

## Allgemein:

- Konstrukturen dienen der **Initialisierung** eines Objektes  
→ Anfangswerte werden gesetzt
- Syntax: *Klassenname(Parameter) {...}*
- **jede** Erzeugung eines Objektes beginnt mit einem Konstruktoraufruf



## Der Standardkonstruktor

- Wird kein eigener Konstruktor geschrieben, erzeugt der Java-Compiler automatisch einen **Standardkonstruktor** ohne Parameter
- In der Regel ist ein eigener Konstruktor aber sinnvoller, da mit ihm (u.a.) sofort die Attribute des Objekts initialisiert werden können

## Beispiel:

```
Notebook myNotebook;  
// deklariert eine Notebook – Variable  
myNotebook = new Notebook();  
// erzeugt ein neues Notebook – Objekt und weist es myNotebook zu  
Notebook yourNotebook = new Notebook();  
// alles auf einmal
```

Erinnerung: Strings müssen **nicht** mit **new** erzeugt werden!

```
String hello = "Hello World";
```

## Beispiel:

```
Notebook myNotebook;  
// deklariert eine Notebook – Variable  
myNotebook = new Notebook();  
// erzeugt ein neues Notebook – Objekt und weist es myNotebook zu  
Notebook yourNotebook = new Notebook();  
// alles auf einmal
```

Erinnerung: Strings müssen **nicht** mit **new** erzeugt werden!

```
String hello = "Hello World";
```

## Noch ein Beispiel:

```
Notebook notebook1 = new Notebook();  
Notebook notebook2 = new Notebook();  
Notebook notebook3 = notebook1;  
notebook1.eingeschaltet = true;  
notebook2.eingeschaltet = true;  
notebook3.eingeschaltet = false;
```

Wie viele Notebooks sind eingeschaltet?

## Noch ein Beispiel:

```
Notebook notebook1 = new Notebook();  
Notebook notebook2 = new Notebook();  
Notebook notebook3 = notebook1;  
notebook1.eingeschaltet = true;  
notebook2.eingeschaltet = true;  
notebook3.eingeschaltet = false;
```

Wie viele Notebooks sind eingeschaltet?

Nur eins, nämlich notebook2!

Bei notebook1 und notebook3 handelt es sich um **dieselben** Objekte!

# Was ist hier passiert?

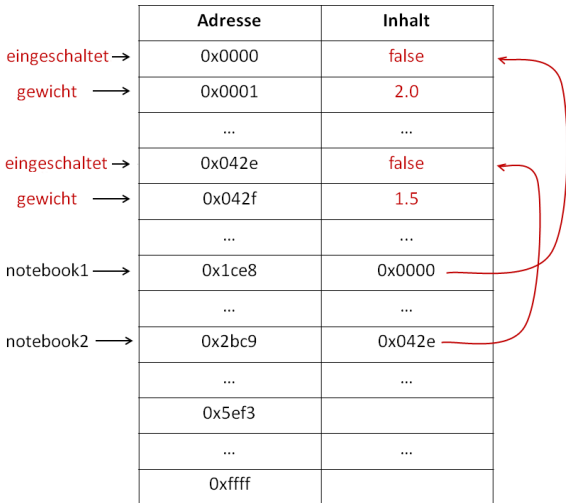
1. **notebook1** & **notebook2** wurden deklariert:

	Adresse	Inhalt
	0x0000	
	0x0001	
	...	...
	0x042e	
	0x042f	
	...	...
notebook1 →	0x1ce8	
	...	...
notebook2 →	0x2bc9	
	...	...
	0x5ef3	
	...	...
	0xffff	

# Was ist hier passiert?

2. Objekte wurden mit **new** erzeugt:

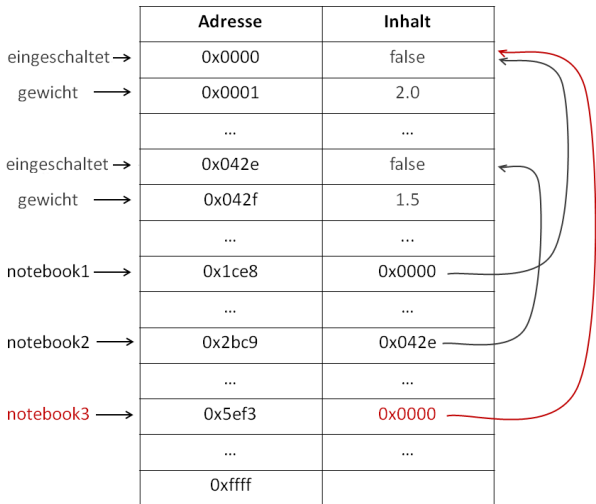
	Adresse	Inhalt
eingeschaltet →	0x0000	false
gewicht →	0x0001	2.0
	...	...
eingeschaltet →	0x042e	false
gewicht →	0x042f	1.5
	...	...
notebook1 →	0x1ce8	0x0000
	...	...
notebook2 →	0x2bc9	0x042e
	...	...
	0x5ef3	
	...	...
	0xffff	



# Was ist hier passiert?

3. **notebook3** wurde die Referenz von **notebook1** zugewiesen:

	Adresse	Inhalt
eingeschaltet →	0x0000	false
gewicht →	0x0001	2.0
	...	...
eingeschaltet →	0x042e	false
gewicht →	0x042f	1.5
	...	...
notebook1 →	0x1ce8	0x0000
	...	...
notebook2 →	0x2bc9	0x042e
	...	...
notebook3 →	0x5ef3	0x0000
	...	...
	0xffff	

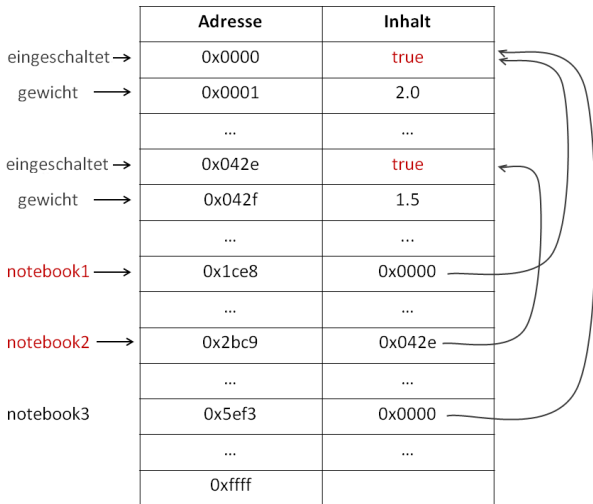




# Was ist hier passiert?

## 4. **notebook1** & **notebook2** wurden eingeschaltet:

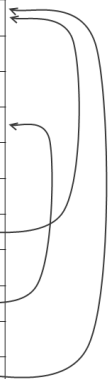
	Adresse	Inhalt
eingeschaltet →	0x0000	true
gewicht →	0x0001	2.0
	...	...
eingeschaltet →	0x042e	true
gewicht →	0x042f	1.5
	...	...
notebook1 →	0x1ce8	0x0000
	...	...
notebook2 →	0x2bc9	0x042e
	...	...
notebook3	0x5ef3	0x0000
	...	...
	0xffff	



# Was ist hier passiert?

5. **notebook3** (= **notebook1**) wurde ausgeschaltet:

	Adresse	Inhalt
eingeschaltet →	0x0000	false
gewicht →	0x0001	2.0
	...	...
eingeschaltet →	0x042e	true
gewicht →	0x042f	1.5
	...	...
notebook1 →	0x1ce8	0x0000
	...	...
notebook2 →	0x2bc9	0x042e
	...	...
<b>notebook3</b>	0x5ef3	0x0000
	...	...
	0xffff	



## Beachte:

Anders als bei elementaren Typen wird bei Klassentypen durch den "=" - Operator nur die **Referenz** auf das Objekt und nicht das Objekt selbst kopiert!

## Ausnahme:

Strings lassen sich durch "=" kopieren, obwohl String ein Klassentyp ist.

## Beachte:

Anders als bei elementaren Typen wird bei Klassentypen durch den "=" - Operator nur die **Referenz** auf das Objekt und nicht das Objekt selbst kopiert!

## Ausnahme:

Strings lassen sich durch "=" kopieren, obwohl String ein Klassentyp ist.

## Was ist eine Methode?

Eine Methode ist ein Konstrukt, mit dem das **dynamische Verhalten** von Objekten realisiert wird.

Mit anderen Worten: Mit Methoden können Objekte endlich etwas "machen"!

## Was ist eine Methode?

Eine Methode ist ein Konstrukt, mit dem das **dynamische Verhalten** von Objekten realisiert wird.

Mit anderen Worten: Mit Methoden können Objekte endlich etwas "machen"!

## Beispiel:

```
class Calculator {  
  
    int memValue;           //der Speicher des Taschenrechners  
  
    int add(int x, int y) {  
        return x + y;      //gibt die Summe von x und y zurueck  
    }  
  
    void memoryWrite(int x) {  
        this.memValue = x; //schreibt den Wert von x in den Speicher  
    }  
  
    int memoryRead() {  
        return this.memValue; //gibt den gespeicherten Wert zurueck  
    }  
  
    void memoryReset() {  
        this.memValue = 0; //setzt den Speicher auf 0 zurueck  
    }  
}
```

## Allgemein:

- *Rückgabetyt Methodenname (Parameter) { Methodenrumpf }*
- Ergebnisrückgabe mit **return** *Wert/Variable*;
- Methoden ohne Rückgabewert haben den "Rückgabetyt" **void**



# Wie benutzt man nun den Taschenrechner?

## Beispiel:

```
class CalculatorTest {  
  
    public static void main(String[] args) {  
        Calculator myCalc = new Calculator();    //einen Taschenrechner  
                                                //erzeugen  
        int sum = myCalc.add(17, 12);           //etwas addieren  
        System.out.println("17 + 12 = " + sum); //und ausgeben  
        myCalc.memoryWrite(42);                 //den Speicher beschreiben  
        System.out.println(myCalc.memoryRead()); //auslesen  
        myCalc.memoryReset();                   //und loeschen  
        System.out.println(myCalc.memoryRead()); //testen, ob wirklich  
                                                //geloescht  
    }  
}
```

⇒ alle Klassen kompilieren & Programm mit "java CalculatorTest"  
ausführen!

## Allgemein

- In einer (separaten) Klasse eine **main-Methode** schreiben:  
`public static void main(String[] args) {...}`

## Warum?

- Code wesentlich leichter zu lesen, übersichtlicher, schöner
- leichteres Einarbeiten in Projekte (Teamarbeit)
- Jeder versteht den code (nicht nur du selbst!)
- Man versteht den code selbst besser!

Code Conventions für Java von Oracle:

<http://www.oracle.com/technetwork/java/codeconventions-150003.pdf>

## Zeilenbreite

Zeilen sollten nicht länger als 120 Zeichen sein

## Kommentare

- sehr nützlich um Code verständlich zu machen
- wichtiges Werkzeug für Dokumentation (javadoc)

```
// gibt die Anzahl Studenten an
int studentCount;

/* es gibt in Java
auch mehrzeilige
Kommentare */
```

## Eine Deklaration pro Zeile, gleicher Datentyp

```
/* guter Stil: */  
int studentCount; // gibt die Anzahl der Studenten an  
int tutorCount; // gibt die Anzahl der Tutoren an  
  
/* schlechter Stil: */  
int studentCount, tutorCount;  
  
/* noch schlimmer: */  
int x, y, z[];
```

## Variablen vor Blocks deklarieren

- Variablen sollten so früh wie möglich deklariert werden
- Nicht erst dann, wenn sie benötigt werden (Ausnahme: for-Schleife)

```
// Beispiel:  
public void myMethod() {  
    int x;  
  
    if(condition) {  
        int y;  
        ...  
    }  
}
```

## Klassen, Interfaces und Funktionen sauber formatieren

- Kein Leerzeichen zwischen Methodenname und "("
- { in der gleichen Zeile wie Klassen/Methodenname
- } in einer eigenen Zeile am Ende
- Methodennamen mit freier Zeile trennen

```
class Sample extends Object {  
    int ivar1;  
    int ivar2;  
  
    Sample(int i, int j) {  
        ivar1 = i;  
        ivar2 = j;  
    }  
  
    int emptyMethod() { }  
  
    ...  
}
```

## Leerzeichen bei Operatoren setzen

- sieht einfach besser aus und ist übersichtlicher!

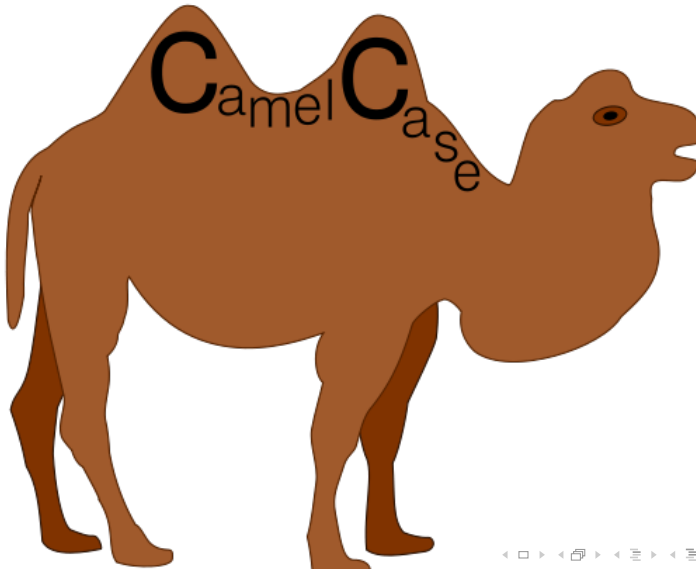
```
// unerwünscht:  
int x=42;  
int y=(x-32)*10+9;
```

```
// besser:  
int x = 42;  
int y = (x - 32) * 10 + 9;
```



## Naming Conventions

- Klassen sollten Substantive sein, beginnend mit Großbuchstaben  
`class ImageSprite { ... }`
- Methoden sollten Verben sein, beginnend mit Kleinbuchstaben  
`private int getCount();`
- Variablen beginnen mit Kleinbuchstaben, kurz aber aussagekräftig  
`int imageWidth;`
- Konstanten: jeder Buchstabe groß, Wörter getrennt durch Underscore  
`public static final int MAX_WIDTH = 800`



# Tutoriumsaufgabe

- Hat sich jeder im Praktomat angemeldet?
- Abgabe **NUR** im Praktomat.
- Abgabe immer Montags bis 13 Uhr.
- Der Praktomat spinnt manchmal etwas, also immer so früh wie möglich probieren abzugeben!
- Für das erste Blatt überprüft der Praktomat nur die Zeilenbreite: Maximal 120 Zeichen!
- In Zukunft wird auch Übersetzbarkeit mit javac geprüft.

## Warnung!

- *Nicht abschreiben!*
- Schon bei **einmaligem** Nachweis verwirkt man die Chance auf den Übungsschein!
- Ohne Schein darf man die Abschlussaufgabe nicht schreiben!
- Nur mit beidem besteht man das Modul Programmieren!
- Programmieren ist Teil der Orientierungsprüfung!
- Ohne bestandene Orientierungsprüfung bis zum 3. Semester fällt man aus dem Studium und darf bundesweit das Studienfach nicht mehr belegen.

# Vielen Dank für die Aufmerksamkeit!

Habt ihr noch Fragen?