IT I

Ronald Ortner

(Folien basierend auf Peter Auers Folien aus dem WS 2009/10)

Warum Programmieren ?

- Programmierfertigkeit ist eine Basistechnik.
- Programmiersprache ist eindeutiges und formales Ausdrucksmittel für Konzepte.
  - Programm ist ausführbar und (teilweise) überprüfbar.
  - OOP ist als Ausdrucksmittel gut geeignet.
- Programmieren fördert die Fähigkeit, die Lösungsschritte für ein Problem klar darzulegen.
- Besseres Verständnis für die Arbeitsweise eines Computers

Inhalt

- Objektorientiertes Programmieren (OOP) in Java
- Fähigkeit zur Abstraktion und Modellbildung
- Simulation

Programmieren

- **Gute Nachricht:**

  Jeder kann programmieren erlernen.

- **Schlechte Nachricht:**

  Wie das Erlernen anderer komplexer Fähigkeiten ist das zeitaufwändig!

Programmieren ist Übungssache!

Gegeben seien die Klassen *Produkt* und *Anlage*.

Die Klasse *Produkt* verfügt über eine Methode

public boolean hatPrioritaet (),

die *true* zurückgibt, wenn das Produkt Priorität hat, ansonsten *false*.

Die Klasse Anlage verfügt über eine Methode

public int gibBearbeitungszeit (Produkt p),

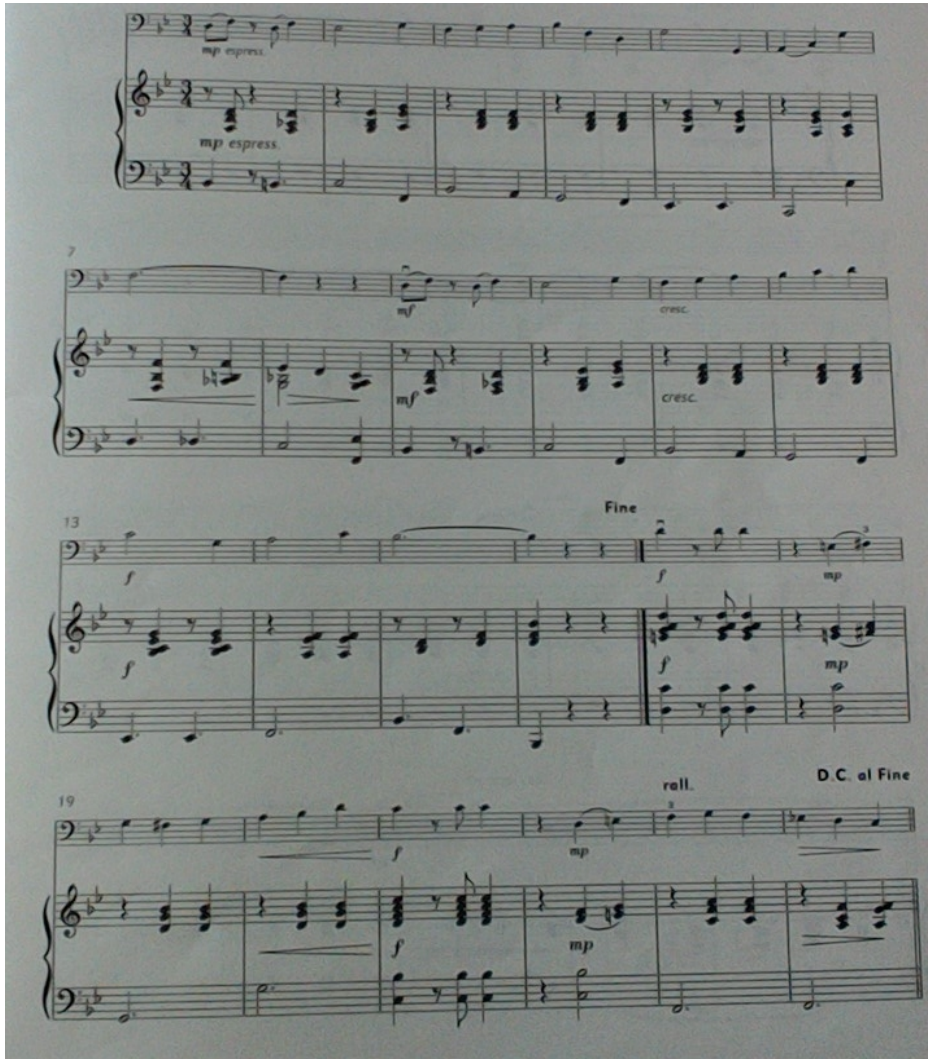
die die Bearbeitungszeit des Produktes *p* auf der Anlage zurückgibt.

Aufgabe: Schreiben Sie eine Methode

ArrayList<Produkt> waehleProdukte(ArrayList< Produkt > dieProdukte, int zeit , Anlage a),

die Produkte aus *dieProdukte* zur Bearbeitung auf Anlage *a* auswählt und in einer ArrayList zurückgibt, die innerhalb von *zeit* Zeiteinheiten bearbeitet werden können. Dabei sollen zunächst möglichst viele Produkte mit Priorität gewählt werden, anschließend sollen für die gegebenenfalls noch verbleibende Zeit möglichst viele Produkte ohne Priorität gewählt werden.

Programmieren ist Übungssache!



Verbreitete Irrtümer

- Programmieren lernt man durch Zusehen.
- Wenn man ein Programm versteht, hätte man es auch selbst schreiben können.
- Wenn's drauf ankommt (Prüfung), genügen die erlaubten schriftlichen Unterlagen, um die richtige Lösung zu finden.
- Für einen positiven Übungsabschluss genügen

  - neben VO und UE zusätzlicher Zeitaufwand von wöchentlich 10-15 Minuten,

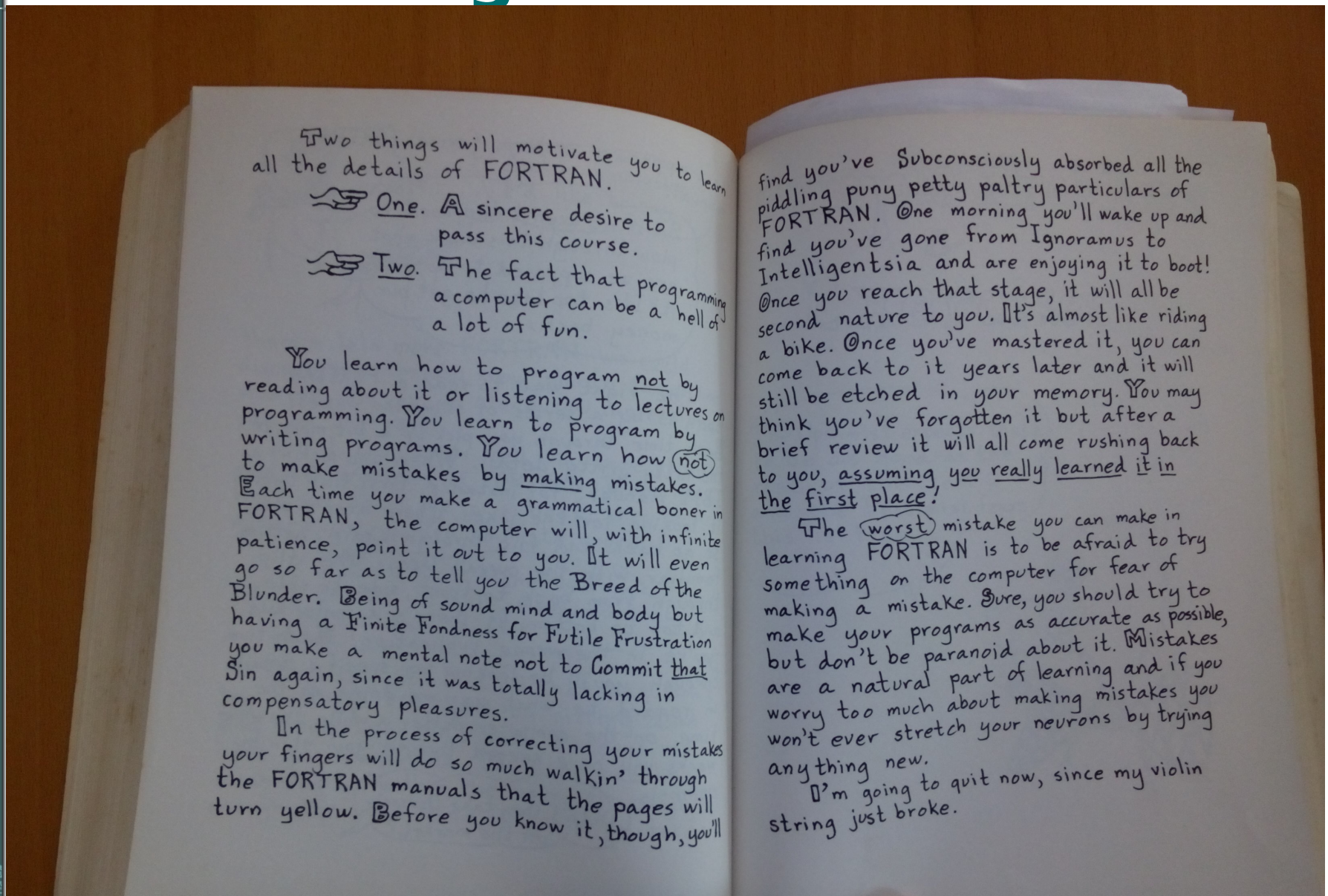
  - je ein Tutoriumsbesuch in der Woche vor den Prüfungen.

  (Im Zweifelsfall "Programmieren" durch "Klavierspielen" ersetzen!)

Wie schafft man IT I?

- Selbst programmieren ist wichtig!
- Alle Programme vom Übungsblatt programmieren!
- Ein Programm nicht nur einmal sondern mehrmals schreiben!
- Alternative Lösungen programmieren!
- Programme mit Kollegen vergleichen, verstehen, und nachprogrammieren!
- Eigene Problemstellungen programmieren!
- Dinge ausprobieren!
- Programme testen!
- Notfalls im Tutorium Probleme besprechen!

Wie lernt man Programmieren?



Two things will motivate you to learn all the details of FORTRAN.

One. A sincere desire to pass this course.

Two. The fact that programming a computer can be a hell of a lot of fun.

You learn how to program not by reading about it or listening to lectures on programming. You learn to program by writing programs. You learn how (not) to make mistakes by making mistakes. Each time you make a grammatical boner in FORTRAN, the computer will, with infinite patience, point it out to you. It will even go so far as to tell you the Breed of the Blunder. Being of sound mind and body but having a Finite Fondness for Futile Frustration you make a mental note not to Commit that Sin again, since it was totally lacking in compensatory pleasures.

In the process of correcting your mistakes your fingers will do so much walkin' through the FORTRAN manuals that the pages will turn yellow. Before you know it, though, you'll find you've Subconsciously absorbed all the piddling puny petty paltry particulars of FORTRAN. One morning you'll wake up and find you've gone from Ignoramus to Intelligentsia and are enjoying it to boot! Once you reach that stage, it will all be second nature to you. It's almost like riding a bike. Once you've mastered it, you can come back to it years later and it will still be etched in your memory. You may think you've forgotten it but after a brief review it will all come rushing back to you, assuming you really learned it in the first place!

The (worst) mistake you can make in learning FORTRAN is to be afraid to try something on the computer for fear of making a mistake. Sure, you should try to make your programs as accurate as possible, but don't be paranoid about it. Mistakes are a natural part of learning and if you worry too much about making mistakes you won't ever stretch your neurons by trying anything new.

I'm going to quit now, since my violin string just broke.

Wieviel Zeit haben Sie?

- Wieviele Stunden planen Sie pro Woche im aktuellen Semester für IT I aufzuwenden?
  (exkl. Besuch der VO und UE)

- Wieviele Stunden Zeit haben Sie pro Woche für CuP aufgewandt?

Lva-Evaluierung

Wie hoch war Ihr durchschnittlicher zeitlicher Aufwand pro Woche für die Übung (WS 15/16)?

- Weniger als 1 Stunden: 9%
- 1 bis 3 Stunden: 26%
- 3 bis 6 Stunden: 30%
- 6 bis 9 Stunden: 35%

Lva-Evaluierung

Wie hoch war Ihr durchschnittlicher zeitlicher Aufwand pro Woche für die Übung (WS 17/18)?

- Weniger als 1 Stunden: 0%
- 1 bis 3 Stunden: 29%
- 3 bis 6 Stunden: 47%
- 6 bis 9 Stunden: 24%

Brauch ich CuP für IT I?

Formal:

- Seit WS 17/18 ist die VO:CuP formale Voraussetzung für die UE: IT I.
- D.h. ohne positiven VO:CuP Abschluss keine Anmeldung zur UE.
- Die UE: CuP ist keine *formale* Voraussetzung.

Inhaltlich:

- CuP-Stoff wird wiederholt.
- Dies geschieht jedoch im Schnelldurchlauf.

Organisatorisches

- Vorlesung
  - Donnerstag 9:15 bis 10:45
    (außer vor den Tests: Di 10.12. & 28.1., 10 bis 12)
- Übung
  - Wissensüberprüfungen: Do 11-12 (HR)
  - Besprechung von Beispielen: Do 12-13 (HS FöT)
  - Verschiebung um eine Stunde wäre möglich
- Tutorium
  - Aufgrund zu weniger HörerInnen findet dieses Semester kein Tutorium statt.
- **Anmeldung** für UE im MUOnline endet am 10.10.

Vorlesung

- Demonstration von Konzepten und Techniken an kleineren und größeren Beispielen.
- Das ist (hoffentlich) instruktiv.
- Jedoch:

  Programmieren lernt man nicht durch Zusehen!

Übungsaufgaben

- Übungsaufgaben
  - erhalten Sie jeweils in der VO
  - Einige Beispiele abzugeben (Abgabesystem). Andere sind bis zur UE vorzubereiten. Auch diese Beispiele werden gewertet (Kreuzerlliste).
  - Abgabe bis Donnerstag der darauf folgenden Woche um 8:00.
  - Abgabe wird automatisch geprüft.
  - Sie erhalten eine Bestätigung bzw. eventuelle Fehlermeldungen.

Eigenständiges Lösen der Übungsbeispiele

- Für einen guten Lernerfolg ist das eigenständige Lösen der Übungsbeispiele Voraussetzung.

- Habe ich den Eindruck, dass Beispiele nicht eigenständig gelöst wurden (z.B. identische Beispiele bei Besprechung in den Übungen), behalte ich mir vor, Beispiele nicht zu werten.

Testen der programmierten Übungsbeispiele

Ein Beispiel ist erst dann gemacht, wenn es auch getestet wurde!

Auf den Kreuzerllisten ist auch anzugeben, ob ein Beispiel getestet wurde!

Ablauf der Übung

Übungsstunden:

1. Gemeinsame Wissensüberprüfung im Hilbertraum
   (meist am PC, manchmal auch am Papier!)
2. Besprechen der zuletzt abgegebenen Beispiele durch Studierende
3. Falls Zeit bleibt:
   Bearbeiten der nächsten Beispiele
   (am eigenen Rechner!)

Beurteilung UE

- Die **Übungsnote** ergibt sich aus:
  - Anzahl der abgegebenen Beispiele zu 15%
  - Wissensüberprüfungen zu 25%
  - Zwischentest (am 12.12. ab 10h) zu 25%
  - Abschlusstest (am 30.1. ab 9:30h) zu 35%

- Für eine positive Beurteilung sind mehr als 50% des erreichbaren Ergebnisses notwendig.

Beurteilung VO

- Für die **Vorlesung** gibt es eine eigene schriftliche Abschlussprüfung, ähnlich der CuP-Abschlussprüfung (aber nicht am PC).
- Prüfungstermin am Mi 29.1., 10-12
- Anmeldung über MU Online.

Vorlesungsunterlagen

- Vorlesungsskriptum über MU Online für angemeldete Studierende zu Download

- Folien, Beispiele, Übungsblätter, Softwarelinks auf der Lehrveranstaltungsseite

Literatur

- Arbeitsbuch:

  David J. Barnes & Michael Kölling:
  Java lernen mit BlueJ
  5. Auflage, Pearson Studium, 2013.

- Für Ambitionierte:

  Joshua Bloch: ***Effective Java***

  2. Auflage, Addison Wesley, 2008.

Java Software

- Wir verwenden in den Übungen
  - BlueJ 3.1.7
  - Java JDK 1.8 (Java 8)

Konzepte der OOP (1)

- ***Objekte*** und ***Klassen***
  - Eine Klasse beschreibt eine bestimmte Art von Objekten, z.B. *Kreiden* oder *Autos*.
  - Ein Objekt ist dann z.B. ein bestimmtes Stück Kreide oder ein bestimmtes Auto.
  - Durch die Klassenbeschreibung werden die gemeinsamen Eigenschaften der Objekte dieser Klasse festgelegt.
  - Objekte in einem Programm können, aber müssen keinem realen Objekt entsprechen.

Konzepte der OOP (2)

- ***Attribute***

  beschreiben die (relevanten) Merkmale der Objekte einer Klasse.
  - Z.B.: Ein Attribut eines Autos könnte sein Kennzeichen sein.
- Der ***Datentyp***

  eines Attributs bestimmt, welche Werte das Attribut annehmen kann.
  - Z.B.: Die Kennzeichen eines Autos ist ein String (eine Zeichenkette).
- Der Zustand eines Objektes ist durch die Werte seiner Attribute bestimmt.
- Die Klassenbeschreibung gibt an, welche Attribute die Objekte der Klasse haben.

Datentypen in Java

- Primitive Datentypen
  - Genau 8 Typen:
    int, boolean, double, long, char, float, byte, short
- Vordefinierte Typen:
  - Z.B. String
- Selbstdefinierte Typen/Klassen
  - Z.B. Auto

Konzepte der OOP (3)

- ***Konstruktoren*** erzeugen Objekte einer Klasse.
- *Java*: Konstruktoren heißen wie die zugehörige Klasse. Ein Konstruktor wird mit dem Schlüsselwort ***new*** aufgerufen.
- Der Zustand eines neuen Objektes muss durch Setzen der Attributwerte definiert werden (*Initialisierung* des Objektes).

Syntax einer Java-Klasse

- Klasse:

class *Klassenname*
{
    Attributdefinition
    …

    Konstruktordefinition
    …

    Methodendefinition
    …
}

- Attributdefinition

    private *Datentyp Attributname*;

Konzepte der OOP (4)

- ***Methoden***

  sind „Aufforderungen“ an Objekte.
  - Methoden haben einen *Namen*, können mit *Parametern* versehen sein, und können einen *Rückgabewert* liefern.
  - Für jede Methode muss angegeben werden, welche Anweisungen sie ausführt.
  - Die Rückgabe eines Wertes erfolgt mittels ***return***.

Parameter

- Durch Angabe von Parametern können Konstruktoren Objekte mit verschiedenen Attributwerten erzeugen.
- Methoden verwenden i.a. ebenfalls Parameter zur Ausführung.
- Parameter dürfen nicht mit Attributen verwechselt werden:
  - Parameter stehen nur während der Ausführung des Konstruktors/der Methode zur Verfügung.
  - Attribute bleiben während der gesamten Lebenszeit des Objektes erhalten.
  - Auf Attribute kann eindeutig mit dem Schlüsselwort ***this*** zugegriffen werden.

Methoden in Java

- Syntax:

```
Datentyp Methodenname(Parameterliste) {
    Methodenrumpf
}
```

- *Datentyp* bezeichnet den Datentyp des Rückgabewertes. Wenn die Methode keinen Rückgabewert liefert, steht hier ***void***.
- *Parameterliste* ist eine Liste der Parameter (durch Beistriche getrennt) mit ihren Datentypen. Die Parameterliste kann auch leer sein.
- Der *Methodenrumpf* gibt an, welche Anweisungen von der Methode ausgeführt werden.

Vergleichsoperatoren in Java

==, !=, <, >, <=, >=

- Diese Operatoren vergleichen zwei Werte und liefern als Ergebnis einen Wert vom Datentyp *boolean* (also *true* oder *false*).
- Die Operatoren <, >, <=, >= sind auf primitive arithmetische Datentypen (z.B. *int*, *double*, *long*) anwendbar.
- Der Gleichheitsoperator == ist auf alle Datentypen anwendbar (ebenso !=).
  - Für *double*-Werte ist das wegen des Rundungsfehlers meist nicht sinnvoll.
  - Der Vergleich von Objektreferenzen liefert nur *true*, wenn die Referenzen auf dasselbe Objekt zeigen.

Die bedingte Anweisung (1)

```
if( Ausdruck ) {
    Anweisungsblock1
} else {
    Anweisungsblock2
}
```

- *Ausdruck* muss einen Wert vom Datentyp *boolean* liefert.
- *Anweisungsblock1* wird nur ausgeführt, wenn Ausdruck *true* liefert, *Anweisungsblock2* wird nur ausgeführt, wenn *Ausdruck false* liefert.

Die bedingte Anweisung (2)

- Der else-Zweig kann auch fehlen:

```
if( Ausdruck ) {
    Anweisungsblock1
}
```

- Die bedingte Anweisung erlaubt es, den Programmablauf von Bedingungen abhängig zu machen.
- In fast allen Problemstellungen müssen bedingte Anweisungen verwendet werden.

Die bedingte Anweisung (3)

- Verschachtelte **if**-Anweisungen:

```
if( Ausdruck1 ) {
    Anweisungsblock1
} else if (Ausdruck2 ) {
    Anweisungsblock2
} else {
    Anweisungsblock3
}
```

- Äquivalent zu:

```
if( Ausdruck1 ) {
    Anweisungsblock1
} else {
        if (Ausdruck2 ) {
            Anweisungsblock2
        } else {
            Anweisungsblock3
        }
}
```

Logische Operatoren

! (Negation), & (Und), | (Oder), &&, ||

- Mit den logischen Operatoren können Ausdrücke vom Datentyp boolean verknüpft werden.
- Bei komplizierteren Ausdrücken ist auf korrekte und ausreichende Klammernsetzung zu achten.

a1	a2	!a1	a1&a2	a1|a2
true	true	false	true	true
true	false	false	false	true
false	true	true	false	true
false	false	true	false	false

Interaktion zwischen Objekten

- Eine Methode eines Objektes kann eine Methode eines anderen Objektes (oder auch eine eigene Methode) aufrufen.
- Dazu muss das fremde Objekt zugreifbar sein.
  - Z.B. ist ein Auto über das Attribut *versAuto* in KfzPolizze zugreifbar (siehe Beispiel im Skriptum).

Aufruf von Objektmethoden

- Eine Methode eines Objektes kann eine Methode eines anderen Objektes (oder auch eine eigene Methode) aufrufen.

Methodenaufruf:

objektname.methodenname(parameterwerte)

Lokale Variablen

- In Methoden können lokale Variablen definiert werden.
- Diese dienen meist zum Speichern von Zwischenergebnissen oder Hilfswerten.
- Einer lokalen Variable ist immer ein Datentyp zugeordnet.

Zuweisung (in Java)

- Durch eine Zuweisung erhält ein Attribut oder eine lokale Variable einen Wert.
- Frühere Werte werden durch eine Zuweisung überschrieben.

- *Syntax:*

  Variable = *Ausdruck*;

- *Variable* ist ein Attribut oder eine lokale Variable.
- Durch den *Ausdruck* wird der Wert berechnet, der der Variablen zugewiesen wird.

return (in Java)

- Mittels **return** kann eine Methode einen Wert an die aufrufende Stelle zurückliefern.
- Eine Methode wird durch Ausführen eines **return** immer beendet.
- *Syntax:*

  return *Ausdruck*;

- Durch den *Ausdruck* wird der Wert berechnet, der zurückgeliefert wird.

Die Objekt-Referenz *null*

- Der Wert *null* einer Objekt-Referenz bedeutet, dass dem Attribut oder der Variablen kein Objekt zugewiesen ist.

Einige Java-Besonderheiten

- Strings können mit dem Operator + aneinander gehängt werden.
- Ausdrücke können über mehrere Zeilen reichen.
- Eine Zahl wird automatisch in einen String umgewandelt, wenn an dieser Stelle ein String erwartet wird:

```
"0" + value
"" + value
```

Java Konventionen

- Konventionen dienen der Übersichtlichkeit und Lesbarkeit der Programme.
- In den Übungen können die Konventionen automatisch geprüft werden.

Java-Konventionen für die Übungen

- Klassennamen beginnen mit einem Großbuchstaben.
- Attribut-, Methoden-, Parameternamen und lokale Variablen beginnen mit einem Kleinbuchstaben.
- Einrückungsstufen (bei Klassendefinitionen, Konstruktoren, Methoden) sind genau vier Leerzeichen.
- Eine Einrückungsstufe enden mit einer schließenden geschwungenen Klammer „}“, die nicht mehr eingerückt wird.
- Alle Attribute sind **private**.

Zu ÜBeispielen/Tests: Angabe verstehen

- **Angabe:** *„Schreiben Sie Klasse **Auto** mit Konstruktor*

  ***Auto(int ps, String kennzeichen)**,*

  *der ein Auto mit **ps** PS und Kennzeichen kennzeichen erstellt."*

- Zu schreibende Klasse muss natürlich entsprechende Attribute haben!