Einführung in die Programmierung

Jean-Claude Graf

[2019-09-17 Di]

Contents

Header **3**
Ziel der Vorlesung . . . 3
Media . . . 4
Slides . . . 4
Video . . . 4
Übungen . . . 4

0.0 Einleitung **5**
Programm, Programmiereun und co. . . . 5
Programm . . . 5
Programmieren . . . 5
Probleme . . . 6
Programmiersprache . . . 6
Java . . . 6

1.0 EBNF Notation **7**
EBNF Beschreibung . . . 7
EBNF Regel . . . 8
Linke-Seite (LHS) . . . 8
Rechte-Seite (RHS) . . . 8
Verbindung . . . 8
Kontrollelemente (*Control Forms*) . . . 9
Überprüfung für Legalität . . . 12
Informeller Bestimmung: . . . 12
Tabellen . . . 12
Ableitungsbäume . . . 13
Sonderzeichen . . . 14
Äquivalenz von Beschreibungen . . . 14
Syntax und Semantik . . . 15
Menge von Zahlen . . . 16
Äquivalenz von Mengen . . . 16
Graphische Darstellung von EBNF Regeln . . . 16

2.0 Einfach Java Programme **19**
EBNF . . . 19
Einleitung . . . 19
Classe . . . 20
Methode . . . 20
String . . . 21
Kommentare . . . 22

2.1 Methoden . . . 22
2.2 Typen und Variable . . . 23
Typen . . . 23
Variabeln . . . 27
2.3 Schleifen . . . 28
for Schleife . . . 28
2.4 Methoden mit Parameter . . . 29
2.5 if Anweisungen . . . 30
Boolean . . . 30
TODO 2.X Einschub Input . . . 31
TODO Strings . . . 32
2.6 Nochmals Schleifen . . . 32
Kurzform zur Aktualisierung des Loop Counters . . . 32
Schleifen Probleme . . . 33
Terminierung von Loops . . . 33
2.7 Rückgabe für Methoden . . . 34
2.8 Sichtbarkeit von Vaiablennamen . . . 34

3.0 Arrays **34**

4.0 Klassen und Objekte **35**
4.1 Klassen . . . 35
Referenzvariabel . . . 36
Value Semantics . . . 37
Reference Semantics . . . 37
4.4 Attribute . . . 37
4.5 Methoden . . . 37
4.6 Konstruktor . . . 38
4.7 Sichtbarkeit für Attribute . . . 38
4.8 Static Methoden und Variabeln . . . 39

Einschub Random und Math **39**
Random . . . 39
Math . . . 40

4.3 Klassen (selber entwickeln) **40**
4.3.1 Einleitung . . . 40
4.3.2 OOP in der Praxis . . . 40

5.1 Graphische Benutzeroberfläche **41**
Window . . . 41

5.2 Input/Output mit Dateien **42**

5.3 Scanner im Einsatz **43**

6.0) Arbeiten mit Objekten und Klassen **44**
6.1) Einleigung-Datenstruktur mit Verknüpfungen . . . 44
6.2) Entwurf von abgekapselten Klassen . . . 44
6.3) Hinweise und Regel für verständliche Programme . . . 45
Code Struktur . . . 45
Namen . . . 45
Weiteres . . . 46

Speicher und Addressen 46
6.5) Mehr Optionen für Kontrolle der Sichbarkeit 47

Header

- Einführung in die Programmierung
- Prof. Dr. Thomas Gross
- Department Informatik
- Laboratory for Software Technology
- Vorlesungsverzeichnis
- Vorlesungswebseite

Ziel der Vorlesung

- Kompetenz
  - Korrekte Programme systematisch erstellen
- Wichtig sind:
  - Aufmerksamkeit
  - Imagination, Phantasie
  - Übung
- Java
  1. Programm lesen
  2. Programm verstehen
  3. Programm erstellen

Media

Slides

- unbeschrieben
  - ev. vor der Vorlesung
- beschrieben (nur mit wichtigen Notizen)
  - 24h-48h nach Vorlesung

Video

- übertragenes Video
- Hauptprojektor

Übungen

- Aufgabenblätter
  - auf Website publiziert
  - in Übungsgruppe vor-besprochen
- Praxisübung
  - Lösung im Internet
- Bonusübung
  - ab 4. od 5. Aufgabenblatt
  - Max 0.25 Note Bonus für Basisprüfung
  - selber lösen
    - Mit anderen reden, aber keine Notizen davon mitnehmen und innert mind 1h danach nichts aufschreiben

---

0.0 Einleitung

Programm, Programmiereun und co.

Programm

- *Programmieren* - *Programm*: griechisch *programma* = schriftliche Banntmachung, Aufruf
- Folge von Anweisungen
- Von Computer ausgeführt
  - Wir müssen wissen was er alles versteht. Mögliche Anweisungen -> Programmiersprache
- Realisiert einen Algorithmus

Programmieren

- Erstellen eines Programms/Software Entwicklung
- Beinhaltet alle Aspekte von Entwurf bis Installation
- Bedenken was nicht offensichtlich ist macht es fordernd
- *Programming as universal activity* by Vinton Cerf, CACM March 2016, vol 59(3) p7
  - analyzing problems
  - breaking them into manageable parts
  - finding solutions
  - integrating the results
- Lösungen finden
  - Für Mensch (Lösung beschreiben)
  - Für Maschiene (Anweisung)
- Wichtiges Informatik Konzept

Probleme

- Möglichkeiten und Grenzen der maschinellen Informationsverarbeitung
  - es exists unmögliche Probleme
  - Kosten der Berechnung sind sehr wichtig

Programmiersprache

- Sprache für Computer (führt aus)
- Folge von mögliche Anweisungen
- Sprache für Menschen (schreibt, liest)
- Geben vor wie Löaungen für ein Problem beschrieben werden können

Java

- *Industrial strength* Sprache
  - sehr verbreitet
- Viele Konzepte
  - nicht alle in dieser Vorlesung behandelt
- JSHell
  - nur für simple Snippets
- Beispiel:
  - Grosse Zahlen können zu unerwarteten Lösungen führen
  - Daher, was sind legale Eingabewerte für Zahlen?

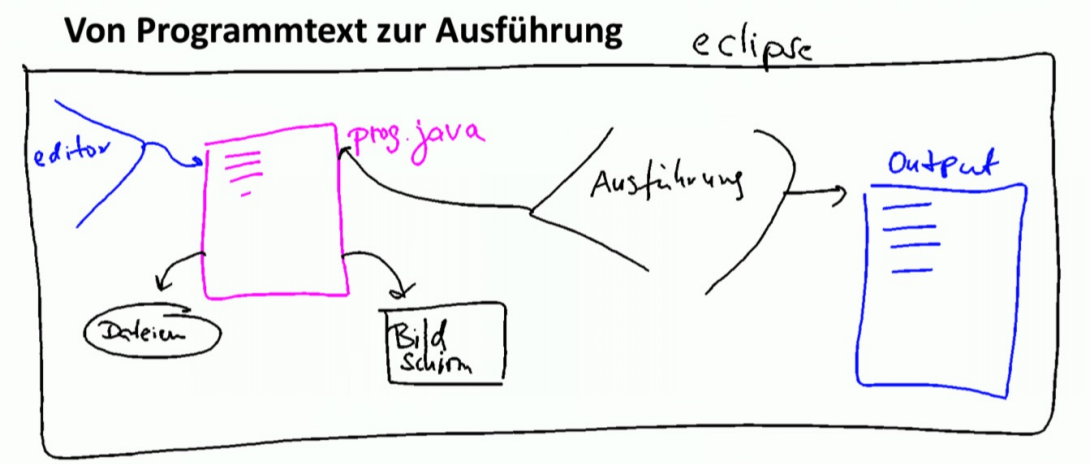


Figure 1: Funktionsweise von Eclipse (jedem IDE)

1.0 EBNF Notation

E - Extended
B - Backus
N - Naur oder Normal
F - Form

- Beschreibt die Syntax einer Sprache
- EBNF ist eine Erweiterung von BNF
- Vier elementaren Ausdrucksmöglichkeiten ("control forms") die in Java vorkommen
- Symbol kann Name, Keyword, Anweisung, Zahl etc. sein

EBNF Beschreibung

- haben einen Namen (für kompliziertere Beschreibungen zu erstellen)
  - gewählter Name ist irrelevant
- erstellen ist ähnlich wie Programmieren in Java
- sind sehr *formal*
  - sehr präzise und verständlich
- Sind eine Menge EBNF Regeln

  - Menge: -> Reihenfolge unwichtig

- geben an welche Symbole erlaubt sind

  - Alphabet muss jedoch nicht explizit definiert werden. Es besteht einfach aus den Verwendeten Symbolen

- Reihenfolge der Regeln ist irrelevant

  - Konvention:

    * von einfachen nach komplexen Regeln

    * Name der letzten Regel ist der Name der relevanten Beschreibung

EBNF Regel

- Haben 3 Bestandteile

- Gibt an welche Symbole erlaubt sind

Linke-Seite (LHS)

- Wort, welches der Name der Regel ist (kursiv- (oder in '<>' als Ersatzdarstellung) und kleingeschrieben)

Rechte-Seite (RHS)

- Beschreibt den Namen

- Kann enthalten

  - Namen anderer Beschreibungen

  - Buchstaben (Definiertes Set möglicher Werten)

  - Kombinationen der vier Kontrollelementen

Verbindung

- "ist definiert als" ($\Leftarrow$) Spezielles zeichen

- trennt LHS und RHS
  LHS $\Leftarrow$ RHS

Kontrollelemente (*Control Forms*)

1. Aufreihung (*sequence*)
   - von links nach rechts gelesen
   - Reihenfolge ist wichtig
     - 'abc' != 'cba'
   - Zwischenraum zwischen Aufreihung ist nicht relevant
     - Leerzeichen muss explizit gezeigt werden wenn eines vorhanden sein sollte
   - Andere EBNF Regeln können als Baustein verwendet werden

   digit-d $\Leftarrow d$
   $digit - 2 \Leftarrow 2$
   $digit - 8 \Leftarrow 8$
   $raum \Leftarrow digit - ddigit - 2digit - 8$

2. Entscheidung (*decision*)
   (a) Auswahl
   - Menge von Alternativen
   - Reihenfolge ist nicht wichtig
   - Durch | (*stroke*) getrennt
   - Um Umklarheiten zu vermeiden können wir Klammern () verwenden um die Reihenfolge zu verdeutlichen
     - wie in der Mathe

   <digit> $\Leftarrow$ 1|2|3|4|5|6|7|8|9|0

(b) Option
   - wie die Alternative jedoch können wir auch "nichts" wählen
   - Element(e) in [] (*Square Brackets*) und durch | getrennt
   - ϵ (Epsilon) steht für "nichts"

<initials> $\Leftarrow T[R]G//TRGoderTG$
$---$
$< Vorzeichen > \Leftarrow [+|-]$
$---$
$< pm > \Leftarrow +|-$
$< vorzeichen > \Leftarrow [pm]$
$---$
$< choice > \Leftarrow [a|]//Whleaoder"nichts"$
be

3. Wiederholung (*repetition*)
    - Zu Wiederholende Ausdruck steht in (*Curly Brackets*)
    - kann 0, 1, oder beliebig mal wiederholt werden
        - 0 Wiederholungen heisst Element fehlt (Epsilon)
        - können nicht sagen das etwas genau n mal wiederholt werden soll

<folge> $\Leftarrow a//kanneineAnzahlvona, oderauchsein$
$---$
$< digit > \Leftarrow 1|2|3|4|5|6|7|8|9|0$
$< integer > \Leftarrow [+|-]digitdigit//IntegralDarstellung$

4. Rekursion (*recursion*)
    - Manchaml nötig für komplexe Beschreibungen
    - Direkt Rekursiv
        - Der Name der Linken Seite wird **direkt** (LHS in RHS) auf der Rechten Seite der Regel verwendet
          <r> $\Leftarrow A|A < r >$
            * Ist AAB legal?

Table 1: Tabellen überprüfung

	Regel
$< r >$	
$B\|Ar$	#1
Ar	#2
$A(b\|Ar)$	#1
AAr	#2
$AA(B\|Ar)$	#1
AAB	#2

    - Indirekt Rekusiv

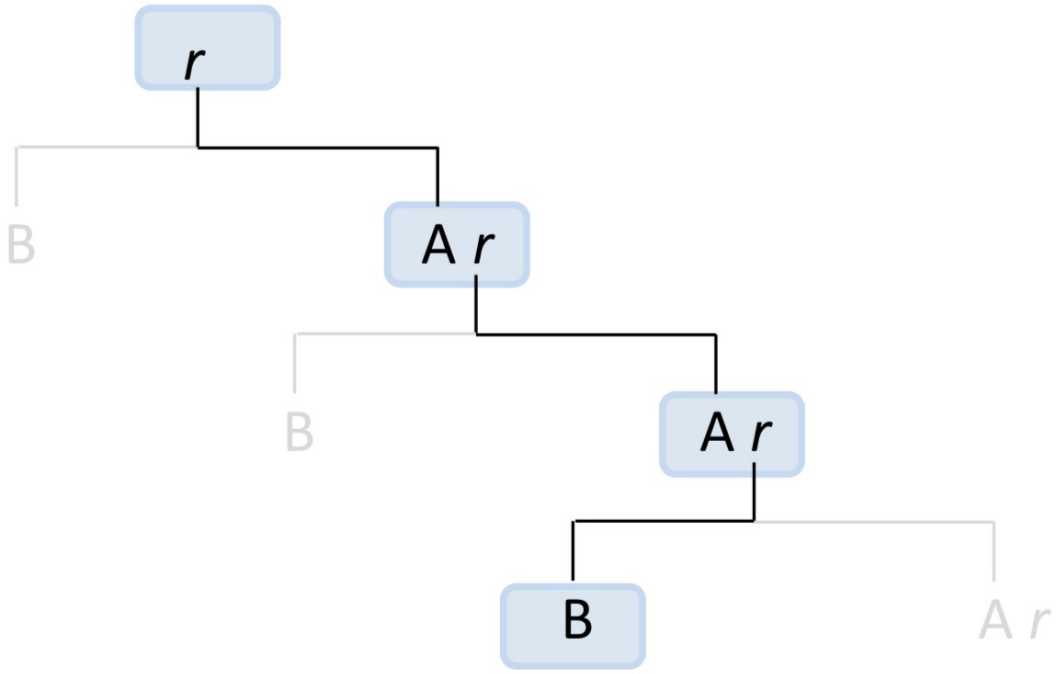
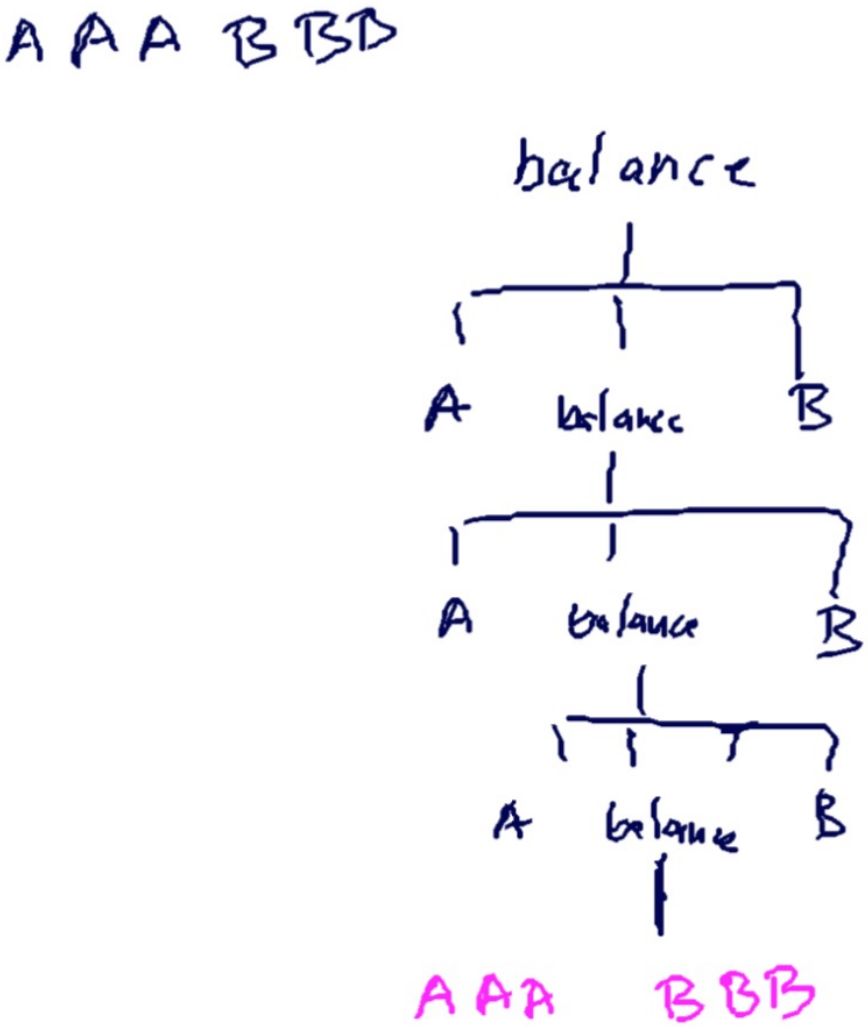


Figure 2: Ableitungbaum überprüfung

– Der Name einer Regel kommt in der RHS einer anderen Regel der selben Beschreibung vor
name1 $\Leftarrow A name2$
$name2 \Leftarrow B name1|C$

- Nicht alle Rekursionen können durch Wiederholungen ausgedrückt werden

  – keine Beschreibung ohne Rekursion für $A^n B^n \forall n \geq 0$
  balance $\Leftarrow |A balance B$



- BNF erhielt anfänglich nur Rekursion und Auswahl
- Nicklaus Wirth fügte das E hinzu

Überprüfung für Legalität

Eine gebegenes Symbol (Folge von Buchstaben) kann für eine EBNF beschreibung legal sein, wenn **alle** Buchstaben des Symbols mit den Elementen der EBNF Regel übereinstimmen, oder ansonnsten illegal, sein. Ein legales symbol *entspricht* einer EBNF Beschreibung.

Informeller Bestimmung:

Wir gehen das Symbol von links nach rechts durch und überprüfen ob es allen Regeln der Beschreibung entspricht

Tabellen

Sind **formaler** und **kompakter**

- Aufbau:
  - 1. Zeile: Namen der zu testenden EBNF Beschreibung
  - folgende Zeilen verden aus der vorgehenden abgeleitet
  - letzte Zeile: Symbol (gegebenes Symbol wenn es der Beschreibung entspricht)
- Durchführung:
  1. Ersetzen eines Namen (LHS) durch die entsprechende Definition (RHS)
  2. Wahle einer Alternative
  3. Entscheidung ob ein optionales Element dabei ist oder nicht
  4. Bestimmung der Zahl der Wiederholungen
  5. 1. & 2. Regeln dürfen als einzige zusammengefasst werden
- Bsp. Überprüfung von +142 für $< digit >$

Status	Reason (rule #)
$< interger >$	Given
$[+\|-] < digit > < digit >$	Replace $< integer >$ by RHS (#1)
$[+] < digit > < digit >$	Choose + alternative (#2#
$+ < digit > < digit >$	Include option (3)
$+1 < digit >$	Replace the first $< digit >$ by 1 alternative (#1 & #2)
$+1 < digit >< digit >$	Use two repetitions (#4)
$+14 < digit >$	Replace the first <digit> by 4 alternative (#1 & #2)
+142	Replace the first <digit> by 2 alternative (#1 & #2)

Ableitungsbäume

Grafische Darstellung eines Beweises

- Aufbau:
  - Oben: Name der zu testenden EBNF Beschreibung
  - Unten: Symbol (gegebenes Symbol wenn es der Beschreibung entspricht)
- Durchführung:

– Kanten Zeigen welche Regel verwendet wird um von einer Zeile zur nächsten zu kommen

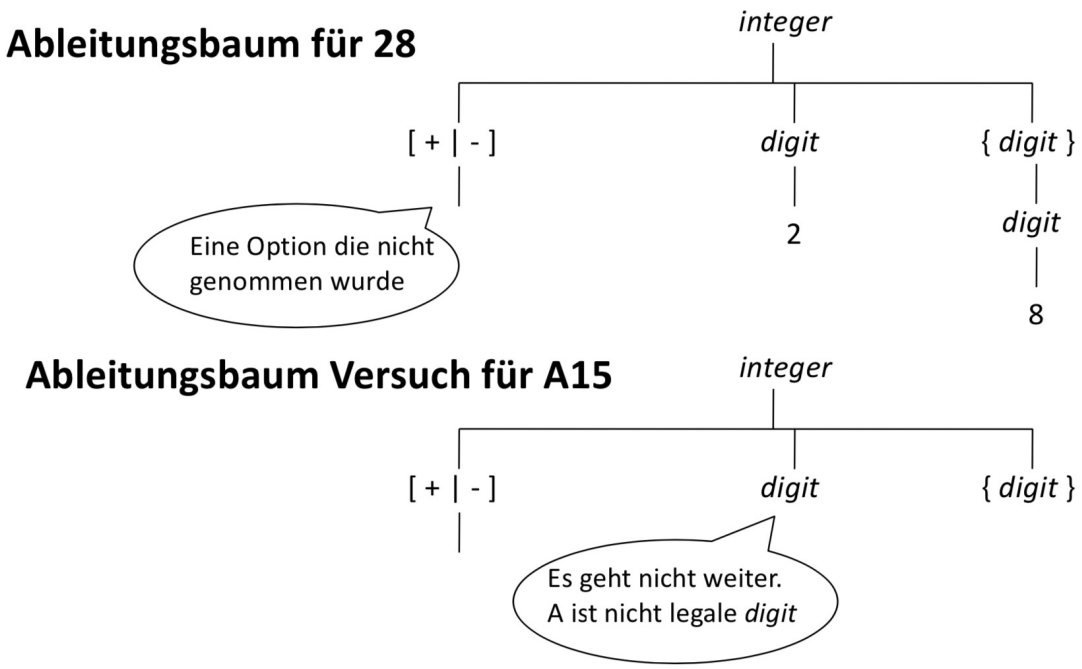


Figure 3: Ableitungbaum

Sonderzeichen

- Viele Zeichen habe eine spezielle Bedeutung in EBNF
  - {, }, [,], |, (,), ⇒,
  - weiter ev. auch <, > und "
- Um ein solches Zeichen als Symbol zu verwenden wir es in einem Quadrat (oder als Ersatzdarstellung in Anzührungszueichen) geschrieben
  - $\boxed{\{}$

Äquivalenz von Beschreibungen

- EBNF Beschreibungen sind äquivalent wenn sie die selben Symbolen legal resp. illegal sind
- die Sprache der Beschreibungen sind identisch
- Zwei Beschreibungen B_1 und B_2 definieren die selbe Sprache:

- Symbol legal fur B_1: dann auch legal fur B_2
- Symbol illegal fur B_1: dann auch illegal fur B_2
- Symbol legal fur B_2: dann auch legal fur B_1
- Symbol illegal fur B_2: dann auch illegal fur B_1

-> äquivalenz zwischen folgenden Beschreibungen liegen vor:
<singn> $\Leftarrow +|-$
$< digit > \Leftarrow 0|1|2|3|4|5|6|7|8|9$
$< integer > \Leftarrow [< sign >]< digit >$
$---$
$< integer > \Leftarrow [+|-]0|1|2|3|4|5|6|7|8|9$

Syntax und Semantik

- *Syntax*: Form
- *Semantik*: Bedeutung
- Syntak ist viel einfach festzulegen als die Semantik
- Eine Beschreibung kann zwar syntaxisch richtig sein, jedoch keinen Sinn machen
  - Bsp. Alle lesenden Schiffe riechen gelb
- zwei wichtige Semantik Fragen:
  1. Können unterschiedliche Symbole die selbe Bedeutung haben?
  2. Kann ein Symbol verschiedene Bedeutungen haben?
- Der Kontext spielt eine wichtige Rolle
  - z.B. haben 0012 und 12 die selbe Bedeutung? Kommt auf den Kontext drauf an
    - Mathematik: ja
    - Pin: nein

Menge von Zahlen

- Klammern mit eine Box oder Anführungzeichen escapen

<intergerlist> $\Leftarrow <integer>, <integer>$
$<integerset> \Leftarrow \boxed{\{}[<integerlist>]\boxed{\}}$

- Entsprechen einer Beschreibung (sind legal) wenn sie legal für alle Elemente sind
  - müssen den Beweis nicht bis zum Ende führen, sondern können beim Erreichen eines Lemmas stoppen

Äquivalenz von Mengen

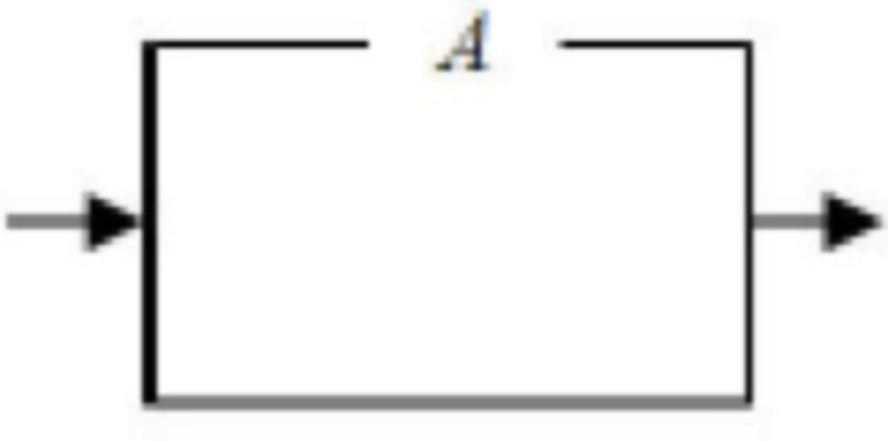
- Mehrfachnennungen und Reihenfolge sind irrelevant
- Kanonische Darstellung (geordnet von klein (links) zu gross (rechts))
  - Kann nicht durch EBNF Regeln erzwungen werden
- Langsam kommen EBNF Regeln an ihre Grenzen

<nonzero> $\Leftarrow 1|2|3|4|5|6|7|8|9$
$<sign> \Leftarrow [+|-]$
$<integer> \Leftarrow <sign><nonzero><nonzero>|0$

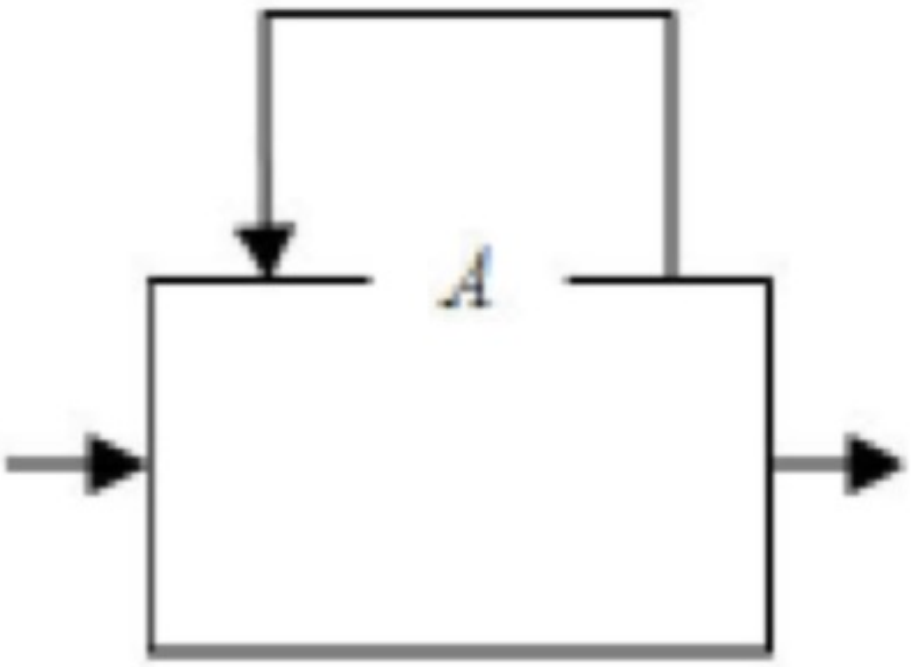
Graphische Darstellung von EBNF Regeln

- Syntax Graph: graphische Darstellung
- Machen es leichter zu erkennen, welche Zeichen in einem Symbol (in welcher Reihenfolge) auftreten können
- Aufreigung: $ABCD$
  - Durch jedes element der Reihe

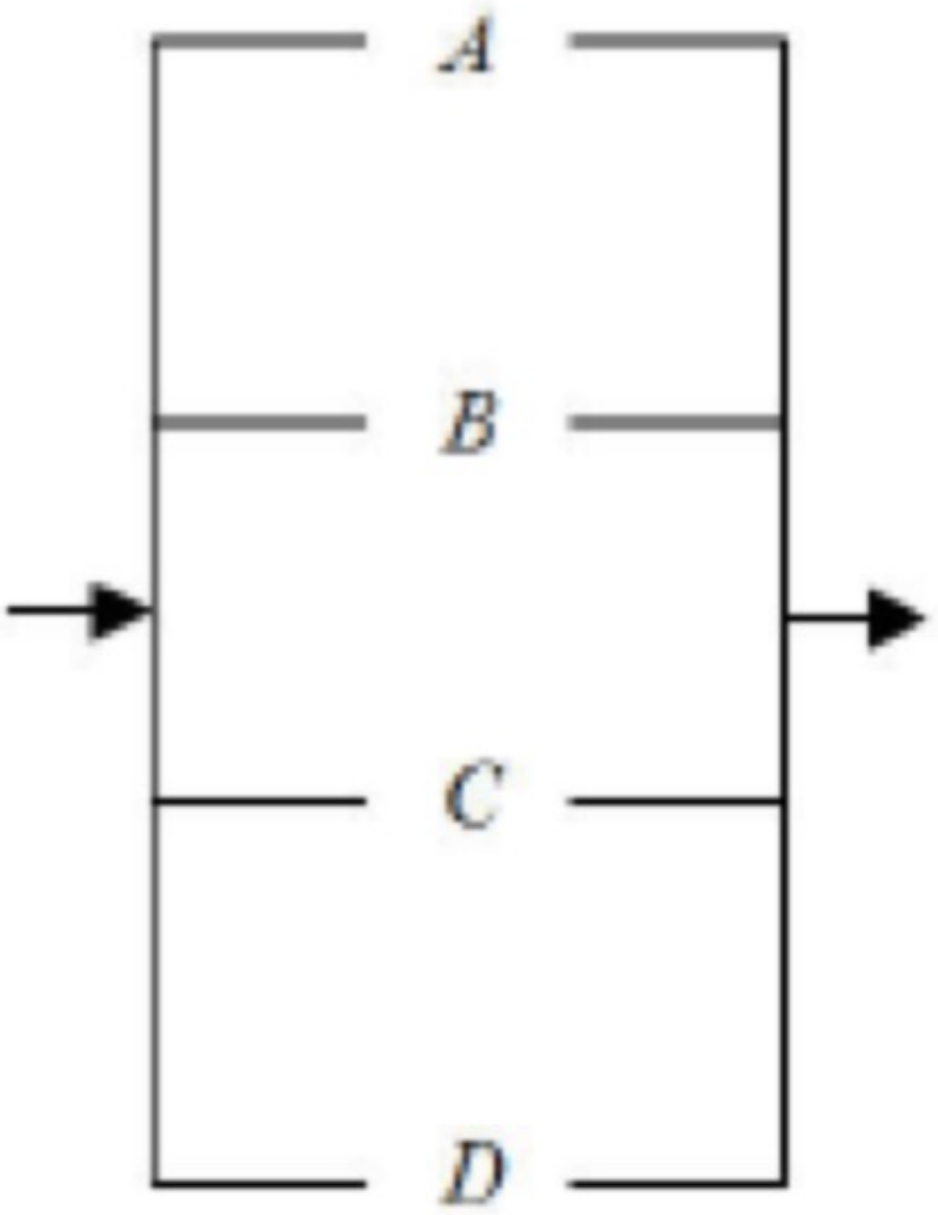
- Option: $[A]$
  - Ein element der Leiter



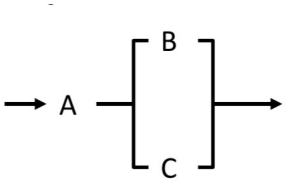
- Wiederholung: A
  - entweder Kante ohne oder Kante mit Element

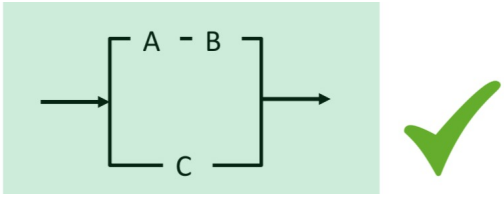


- Auswahl: $A|B|C|D$
  - Pfeil von rechts zurück nach links

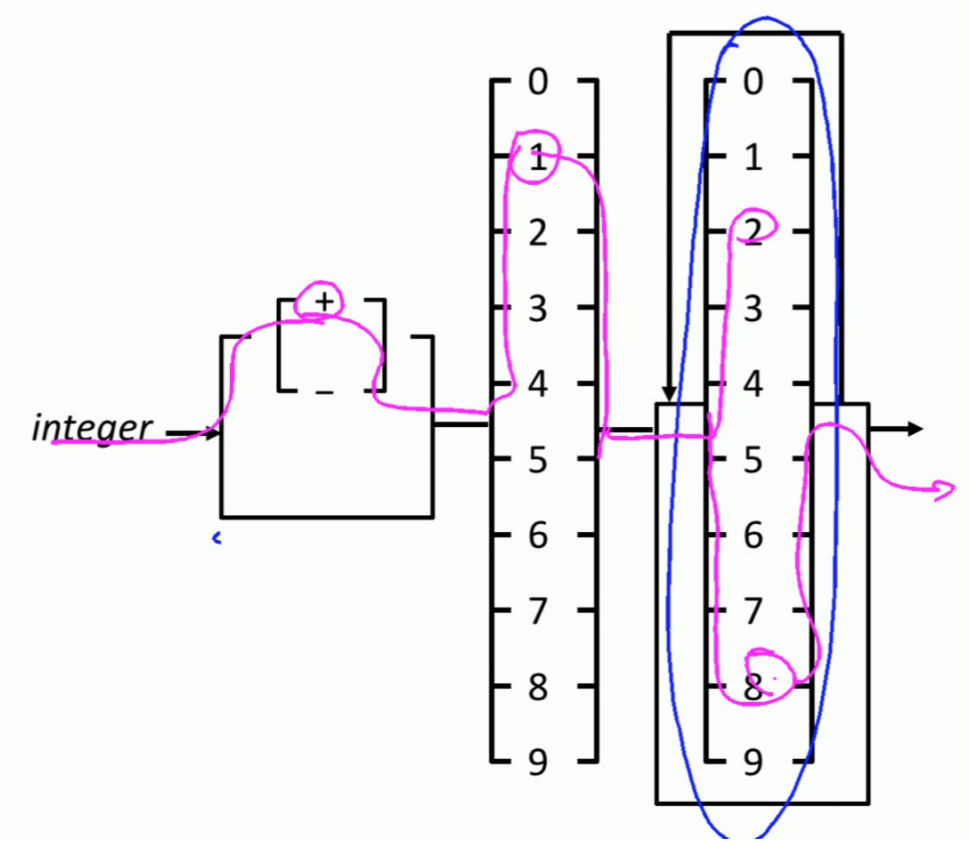


- Klammersetzung: $AB|C$
  - Klammern sind sehr wichtig





- Können einen Graphen in einen anderen einsetzen/substitutionieren
  - interne Namen von Beschreibungen verschwinden



- Können von einem Graphen auf legale Symbole zurückführen oder sogar die Beschreibung mit den Regeln rekonstruieren

2.0 Einfach Java Programme

EBNF

- Hält die Synatx der Regeln von Java fest
- Namen / identifier:
  - mind. 1 Zeichen lang
  - mit Buchstaben anfangen [a-zA-Z]
  - kann Buchstaben und Ziffern enthalten

Einleitung

- Programm besteht aus mind einer Klasse, mind einer Methode *main* und einer Reihe von Answeisungen

lowercaseletter ⟸ a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

uppercaseletter ⟸ A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

letter ⟸ *lowercaseletter* | *uppercaseletter*

digit ⟸ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7| 8 | 9

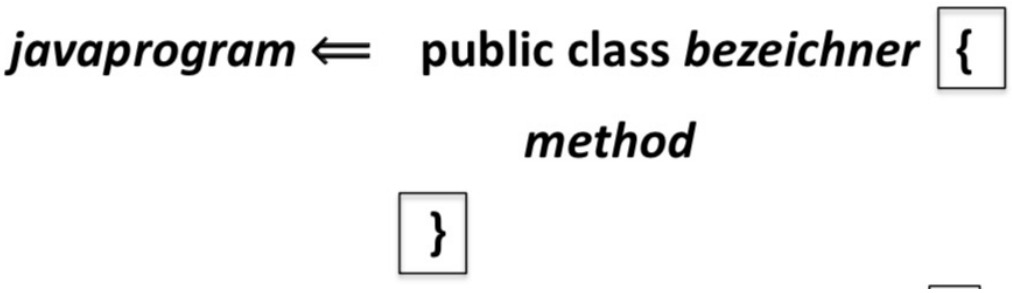
bezeichner ⟸ *letter* { *letter* | *digit* }

Figure 4: Java Regeln

- Wir müssen uns strikt an die Anweisungen halten, sonst gibt es Probleme beim kompolieren → Error Message
  - nicht alle Fehlermeldungen sind intuitiv
- jshell erlaubt das ausführen von Code wie in einer Hauptclass und Main Methode
- Bestimmte Namen sind durch Java reserviert und dürfen nicht vorkommen

Classe

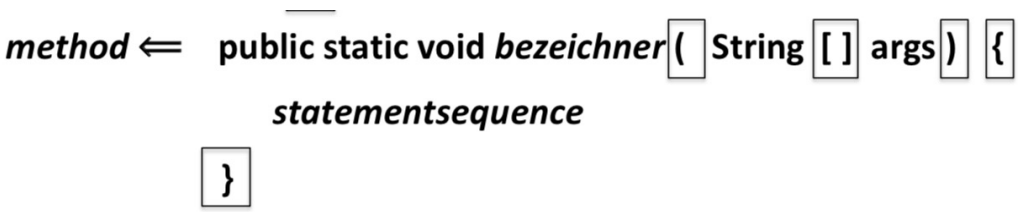
- nur eine *class* pro Datei (fürs Erste)
- Name/Bezeichner der *class* gleich Name der Datei
- Konvention: UpperCamelCase



Methode

- *publicstaticvoidmain* muss zwingen in einer Klasse vorkommen
- *main* muss denn auszufürenden Code beinhalten

- Konvention lowerCamelCase



- *println* Java Medhode/Funktion
  - gibt Strings aus mit neuer Zeile
- *print*
  - gibt String aus ohne neue Zeile

String

- Text zwischen Anführungszeichen "bla"
- Folge von Buchstaben
- Darf nur eine Zeile lang sein
- Darf keine Anführungzeichen beinhalten
- Sonderzeichen
  - Ersatzdarstellungen (escape sequences)
    Startet mit Backslash
    * new line character
    * æTab character
    * ΞQuotation mark characeter
    *
      Backslash character

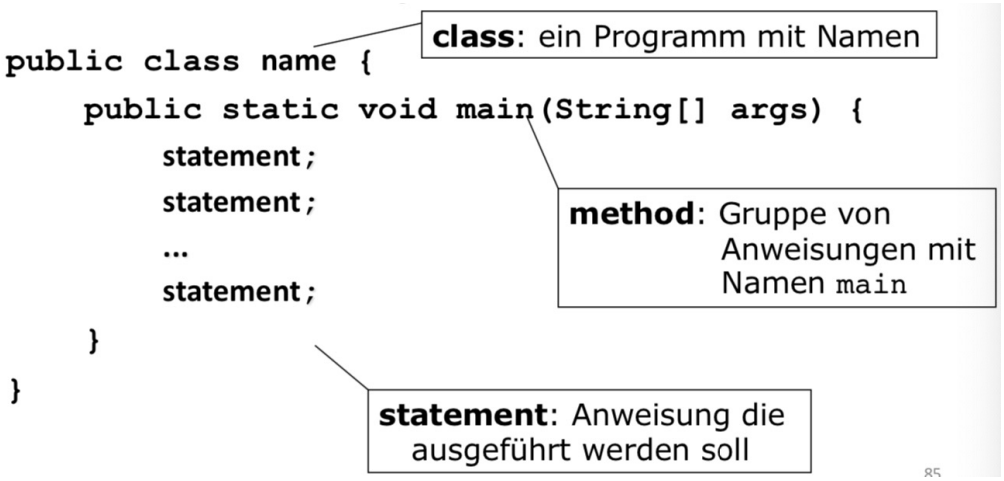
Kommentare

- Notizen im Programmcode

```
// Text bis zum Ende der Zeile
/* Text bis zum
naechsten
*/
```

  - Verwendung:
    - Anfang des Programms
      - Zweck
      - Algorithmus
      - Author
    - Anfang jeder Methode
    - Wenn Code nicht direkt verständliche

```
/*
 * Author: Ein Student; Herbst 2016, Uebung 1
 * Entwurf uebernommen von einer Frau XXXX (Assistentin)
 */
```



2.1 Methoden

- Folge von strukturierten Anweisungen mit Namen
- Wiederholungen können vermieden werden
- *static*

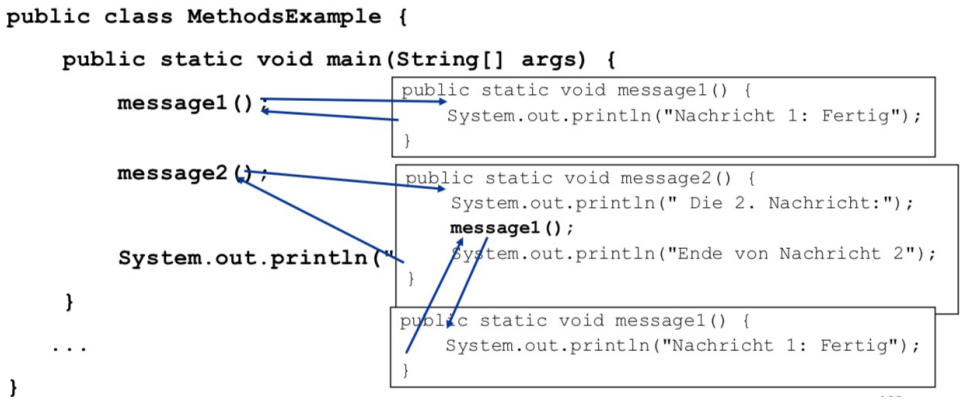
  - gibt der Methode weitere Eigenschaften
  - *main* ist *static*
  - *main* wird automatisch aufgerufen

- Methoden finden, die Teilprobleme lösen
  - Teilprobleme einzeln lösen

- Müssen *deklariert* werden, befor sie aufgerufen wrden können

- Mehrfachaufruf möglich

- Folge der Ausführungen wird *control flow* genannt
  - Zuerst wird die Methode komplet ausgeführt, erst dann wird zurück zur Aufrufermethode gegangen
    ⇒ geradliniger Kontrollflow
  - Anweisungsreihenfolge ist explizit
    * Control flow kann aktiv geändert werden



- Anordnung der Methoden in der Class ist irrelevant

2.2 Typen und Variable

Typen

- Beschreiben Eigenschaften von Daten

- Haben Auswirkung auf:

  - Darstellung der Werte
  - mögliche Operationen
- Sagen wie Daten im Computer gespeichert sind
  - Intern wird alles in Binär gespeichert
  - Definiert durch eine Tabelle (Ähnlich wie ASCII)
- ist ein Binärstring 01100001 nun eine Zahl 97 oder ein Zeichen a?
  - Datentyp gibt nötige Zusatzinformation
- Tatentypen sollten immer explizit angegeben werden
  - Compiler kann auch selbstständig den Typ herausfinden, ist jedoch nicht so empfolen
- Vrhinden Fehler
- Erlauben Optimierungen
- Von wo kommen Typen
  - in der Sprache Definierte Type (*primitive types*)

Table 2: Primative Types

Name	Beschreibung
int	ganze Zahlen
long	ganze lange Zahlen
double	reelle Zahlen
char	einzelne Buchstaben
characterboolean	logische Werte

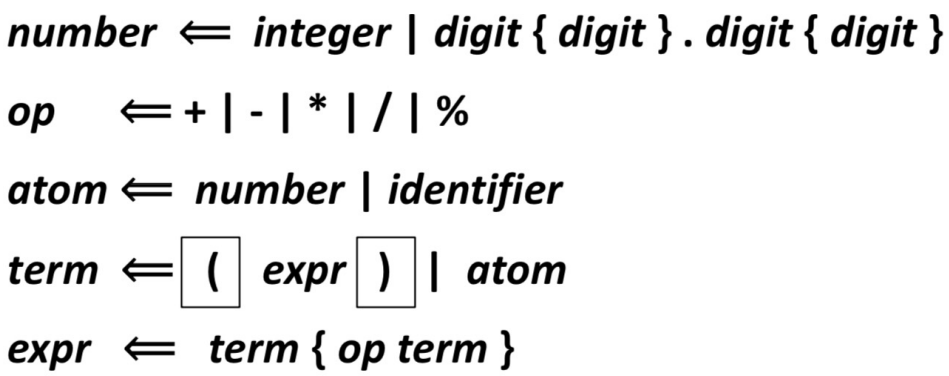
  - Typen aus Bibliotheken
  - Benutzer-definierte Typen
- Werte haben einen (festen) Typen
  - $TypeATypeA \rightarrow TypeA$

    * z.b. $14/3 = 4$

1. Ausdrücke (Expressions)

    - Ein Wert (*literal value*), Operanden, Operatoren oder eine kombination von mehreren sind Ausdrücke für einen Typen

    - Werden während der Ausführung ausgewertet



2. Operatoren

    - Verknüpfen Werte oder Ausdrücke

        - $+$ Addition

        - - Subtraktion

        - * Multiplikation

        - / Division

        - % Modulo

            * gibt den Rest der int division zurück

3. Assoziativität

    - Bestimmt welche Elemente ein Operand verbindet

    - $a \bigodot b \bigodot c$

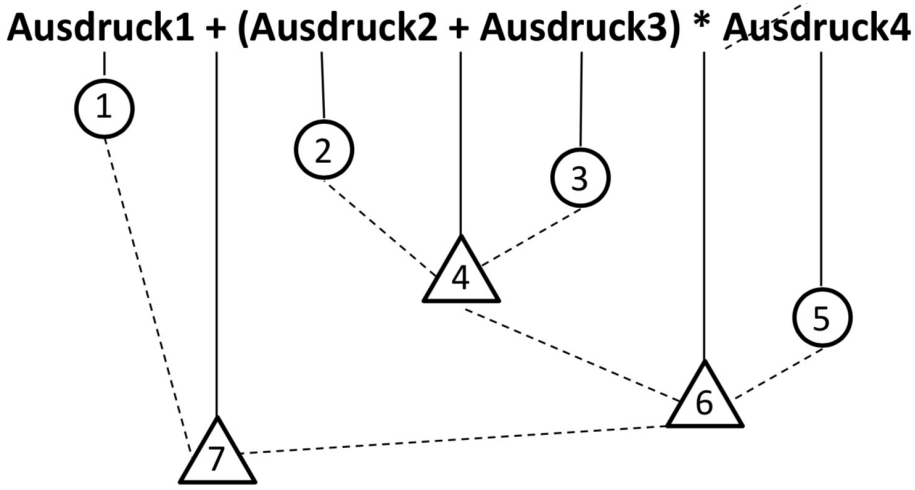
        - Rechtsassoziativ: $a \bigodot (b \bigodot c)$

        - Linksassoziativ: $(a \bigodot b) \bigodot c$

    - die meisten bekannten Operatoren sind Linksassoziativ

4. Rangordnung (*Precedence*)
    - Bestimmt welche Operatoren in welcher Reihenfolge ausgeführt werden
    - Rangordnung von $\odot$ und $\otimes$ entscheidet wenn ein Ausdruck mehrere Operanden X, Y, Z hat

5. Operanden und Operatoren
    - Ausführ Reihenfolge bei vorkommen von mehreren Operatoren
        - Rangordnung
        - Assoziativität
        - von links nach rechts
    - mit Klammern kann eine Abweichung von diesen Regeln erzwungen werden.



6. *Long*
    - löst das Problem von *int* (Grosse Zahle verhalten sich "komisch")
        - kann aber auch überfordert werden
    - Durch anfügen von L ans Ende der Zahl wird die Zahl als *Long* definiert

7. *Double*
    - Reelle Zahlen
    - durch hinzuführen von $.$ oder $.0$ zu einer ganzen Zahl, macht sie zu *Double*

8. Kombinieren von Ausdrücken und Typen

- *Int* oder *Long* und *Double* werden kombiniert zu *Double*
  - $double \bigodot intOrLong) double$
- Wird in einem Audruck für jede Operation einzeln gemacht
  - dabei wird die Rangordnung und Assoziativität beachtet

9. Umwandlung von Typen
   - implizit: Durch kompination von Typen
   - explizip: durch type cast
     - cast bezieht sich nur auf den Ausdruck direkt dahinter (Rechtsassoziativ)
       - $(type) expression$
     - höhere Rangordnung als Arithmetische Operatoren
10. String Operatoren
    - + Operator verkettet (*concatenation*) *Strings*
    - Zahlen werden automatich zu *String* konvertiert wenn man den Operator für *String* und Zahl verwendet

Variabeln

- Name der es erlauft, auf einen gespeicherten Wert zuzugreifen
- verhindert Wiederholungen und doppeltes Berechnen
- Ablauf
  - Deklaration: gibt Name und Typ an
    - Reserviert Speicher für den Wert
      $intmyNumber$;

      typeidentifier ⟸ *bezeichner*
      variableidentifier ⟸ *bezeichner*
      variabledeclaration ⟸ *typeidentifier variableidentifier* { , *variableidentifier* } ;
  - Zuweisung(*Assignment*): speichert einen Wert in der Variabel

* mittels = Zeichen
  $myNumber = 5;$

$$variableidentifier \Longleftarrow bezeichner$$
$$assignment \Longleftarrow variableidentifier = expression\ ;$$

* Wiederholte Zuweisungen sind erlaubt

– Gebrauch: in einem Ausdruck durch aufrufen des Variabelnamen

- Deklaration und Zuwesung separat oder auch kombiniert in einer Zeile erfolgen
  $intx = 5;$

$$typeidentifier \Longleftarrow bezeichner$$
$$variableidentifier \Longleftarrow bezeichner$$
$$variablelist \Longleftarrow variableidentifier\ \{\ ,\ variableidentifier\ \}$$
$$variableinitialization \Longleftarrow variableidentifier\ =\ expr$$
$$variablespecification \Longleftarrow variableinitialization\ |\ variablelist$$
$$variabledeclaration \Longleftarrow typeidentifier\ variablespecification\ ;$$

- Java ist passed by Value

- Können nur Werte ihres Types speichern

2.3 Schleifen

- Wiederholtes Ausführen von Anweisungen

for **Schleife**

- wird einmal Inizialisiert

  – legt die Variabel (Zählervariabel) der Schleife fest (*loop counter*)

- solange *Test* true wird folgendes Wiederholt

  – *Statements*

  – *Update*

```
for (initialization; test; update) {
      statement;
      statement;
      ...
      statement;
}
```

header (Kopf)

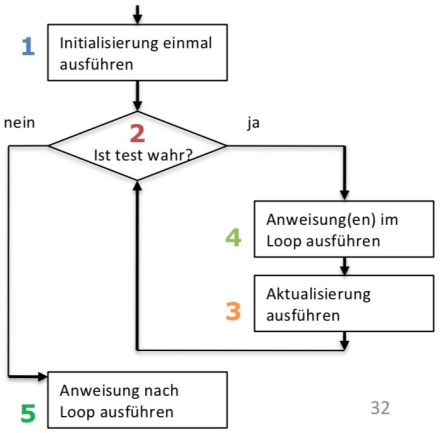
body (Rumpf)

- Loop Counter muss sich ändern sonst terminiert der Loop nie

```
for (int i = 1; i <= 4; i = i+1) {
  System.out.println(i + " hoch 2 = " + (i * i));
}
System.out.println("Whoa!");
```

Output:

```
1 hoch 2 = 1
2 hoch 2 = 4
3 hoch 2 = 9
4 hoch 2 = 16
Whoa!
```



- Können verschachtelt sein

2.4 Methoden mit Parameter

- Parameter: Ein Wert den eine aufgerufene Methode von der aufrufenden Methode erhält
- Parametrifizierung: mit Parametern versehen
- Parameter werden mit Datentyp bei der Methoden Deklaration angegeben
  - **Alle** Parameter müssen im richtigen Datentyp gegeben werden
- Parameter werden beim Aufrufen der Methode übergeben
- Parameter in der Deklaration heisst Formaler Parameter (*Formal Parameter*)
- Der tatsächlich übergebene Wert heisst tatäschlicher Parameter (*Actual Parameter*) oder Argument

- Argument wird in der Methode in einer Parameter Variable gespeichert
- Die Methode erhält lediglich den Wert und hat z.B. keine Referenz zur Variabel von wo der Wert kommt
  - Die Variabel die übergeben wurde kann sich nicht ändern, da die Methode keine Referenz zu ihr hat
  - Parameter der Basistypen werden bei der Übergabe kopiert (*Value Semantics*)

2.5 if Anweisungen

- erlauben Verzweigungen
- if Block nur ausgeführt, wenn ein bestimmter Test wahr ist
  - else Block wird ansonnsten ausgeführt
- Vergleichsketten sind in Java nicht erlaubt ~~$2 <= x <= 10$~~
- Operatoren &&, || und ! um Aussagen zu verbinden
- Pfade:
  - Genau ein Pfad: if, else if, else
  - o oder 1 Pfad: if, else if, else if
  - 0, 1, oder viele: if, if, if

Boolean

- können nur wahr oder falsch sein
- ein Vergleich ist ein Boolischer Ausdruck
- werden mit boolischen Operatoren kombiniert
- werden in Tests nicht mit true oder false verglichen
  - ~~isTrue == true~~

- werden von links nach rechts folgend ihrer Preszendenz ausgewertet
- Bedingte Auswertung
  - Java beendet die evaluation sobald ein teil des Tests falsch ist
  - (false && egalWasHierSteht) -> egalWasHierSteht wird nicht ausgewertet

TODO 2.X Einschub Input

- Liest Input von der Konsole
- Standard Input: Vordefiniertes Inputfenster
  - *System.in*
- Standard Output: Vordefiniertes Outputfenster
  - *System.out*
- Normalerweise int Input- und Outputfenster das selber
- Output ist einfacher als Input
  - Da man nicht weis was der Benutzer alles eingibt?
- *Scanner* erlaubt Services um von der Konsole zu lesen
  - Erlaubt Input von unterschiedlichen Quellen (Datei, Konsole, Webseite...)
  - Kommt von Bibliotheke *java.util*
  - Import mit /import java.util.*;
  - Scanner constuction: *Scanner name = new Scanner(System.in);*
  - Wartet bis der User "ENTER" oder "RETURN" clickt

Methode	Description
nextInt()	reads an Int from the user and returnss it
nextDouble()	reads a Double from the user
next()	reads a one-word String from the user
nextLine()	reads a one-line String from the user

- Fordern (*prompt*) the User auf eine Eingabe zu machen
- *nextInt()* liest eine Folge von Ziffern (Token) und wandelt diese in einen Int um
  - Tokens werden durch *whitespaces* getrennt (*space*, *tab*, *new line*)

```
  System.out.print("Wie alt sind Sie? "); // prompt
int alter = console.nextInt();
System.out.println("Ihre Eingabe " + alter);
```

TODO Strings

- Zugriff auf die Buchstaben mittels Index
  - Startet mit Index 0
  - Buchstaben sind vom Basistyp *char*
- Ein String ist kein Array
- Sind sehr wichtig und werden vom Compiler daher anders behandelt

Methode	Description
charAt(index)	Character at index
indexOf(str)	index where the start of the given string appears. If not found -1
length()	nunber of characters in this string
substring(index1, index2)	the characters in this string from index1 (inclusive) to index2 (exclusive)
substring(index1)	grabs till end of string
toLowerCase()	a new String with all lowercase lettes
toUpperCase()	a new String with all uppercase letters

2.6 Nochmals Schleifen

Kurzform zur Aktualisierung des Loop Counters

Increment oder decrement um 1 ist sehr häufig

- $i = i + 1 \Rightarrow i++$
  - $i++$ wird zuerst verwendet, danach erhöht.
    $y = x++; \Rightarrow temp = x; x = x + 1; y = temp;$
  - Diese Operatioen sind nicht so effizient wie man denken könnte und führen zu verwirrung und Fehler

- $i = i + 1 \Rightarrow i+ = 1$

- Increments in einer || und expression sind gefährlich, denn diese werden nicht zwingend ausgeführt

  - $(ex1||ex2) \neq (ex2||ex1)$

Schleifen Probleme

Wir wollen alle Zahlen, kommagetrennt bis n ausgeben.

- Verhindern das ein Komma am Schluss kommt

- Lösung:
  Wir Printen den erste Fahl ausserhalb des Loops und dann immer eine Komma, gefolgt von der Zahl. Oder können auch den letzten Fall am Ende nach dem Loop Printen

- *off-by-one* Error ist sehr läufig
  Loop wurde einmal zuviel ode zu wenig ausgeführt

Terminierung von Loops

1. while Schleife

   - unkestimmte Schleife (*idefinite loop*): Anzahl der Ausführungen ist im voraus nicht bekannt

```
while (test) {
    statement(s);
}
```

   - Sentinel: Ein Wert der das Ende einer Reihe anzeigt

     - While loop wird terminiert sobald der sentinel eingegeben wird

   - do-wile löst das Problem das etwas bei eingabe eines sientinels nochmals eingegeben wird

2.7 Rückgabe für Methoden

- erlaubt kommunikation zwischen Aufrufer und Methode
- *Return value* wird an Aufrufer gegeben
- *return* gibt die Expression zurück und beendet die Methode
  - kann auch keinen Wert zurückgeben
- Typ muss stimmen
- Bei verschachtelten If/Else macht der Kompiler manchmal blöd wenn ein If nur true sein kann.

2.8 Sichtbarkeit von Vaiablennamen

- Sichtbarkeit (*Scope*)
- Variabeln müssen deklariert sein bevor sie sichtbar sind
- Sichtbar von Deklaration bis zum Ende des Blocks für den die Variable deklariert ist
  - Block ist durch { und } begrenzt

3.0 Arrays

- Sind Objekte
- Mehrere Werte des selben Typs speichern
- Element: Wert eines Array
- Index: Zahl un ein Element des Arrays auzuwählen
- Base: Erste Element hat Index 0

```
type[] name = new type[length];
```

- Länge kann eine beliebiger int sein

- Länge kann auch implizit gegeben sein durch die Anzahl elemente bei der Zuweisung
- Anhängig von Typ wird der Array mit unterschiedlichen defaultwerten initiaisiert
- Zugriff via Index:

```
name[index]; // Access
name[index] = value; // assign
```

- Legaler index zwischen 0 und lenght -1
- *name.length* liefert die Anzahl Elemente des Arrays
- werden häufig zusammen mit Schleifen verwendet
- Methoden

binarySearch(array, value)	returns the index of the given value in a sorted array (or < 0 if not fou
copyOf(array, length)	returns a new copy of an array
equals(array1, array2)	returns true if the two arrays contain same elements in the same order
fill(array, value)	sets every element to the given value
sort(array)	arranges the elements into sorted order
toString(array)	returns a string representing the array, such as "[10, 30, - 25, 17]"

- Bei der Übergabe von Arrays al Paameter oder als Return wert müssen wir [] Klammern verwenden

4.0 Klassen und Objekte

4.1 Klassen

- wird mit Keyword *class* definiert
- Bieten einen *Service* an
- Typen von Klassen
  - Namenloser Dienst
    - wird geladen und ausgeführt
  - Mit Namen ausgewählter Dienste

  - Eigene Klassen

- Beschreibt einen Typ

- Referenzvariabel verweist auf ein Objekt

- Objekt ist der Sammelbegriff aller Datenwerte die durch eine Klasse beschrieben werden

  - wird erschaffen

    * wird durch *new* operator gemacht, oder durch eine Initialisierung

```
String s; int counter;
s = "hello " + counter;
```

- Vergleichsoperatoren funktionieren nicht für Objekte

  - $"Hello" == "Hello" \Rightarrow false$

  - *equals* wird stattdessen verwend

equals(str)	whether two strings contain the same characters
equalsIgnoreCase(str)	whether two strings contain the same characters, ignoring upper vs. lov
startsWith(str)	whether one contains other's characters at start
endsWith(str)	whether one contains other's characters at end
contains(str)	whether the given string is found within this one

Referenzvariabel

- Referenzvariabel (*reference type variable*)

- verweisen immer auf das selbe Objekt

- bei der Zuweisung muss jedoch der Typ stimmen

- wenn einer Referenzvariabel eine neue Referenzvariabel assigned wird, geht der initiale Wert verlohren(er existiert jedoch weiter im Speicher)

- Diese "verlohreren" Werte werden automatisch gelöscht

- Änderung der Referenzvariabel ändert die originale Variabel

Value Semantics

- Passed by Value
- trifft für die Primitive Types zu (int, boolean etc.)

Reference Semantics

- Passed by Reference (gibt es eigentlich nicht in Java, aber irgendwie trotzdem)
- Erlauben einer Methode den Parameter direkt zu bearbeiten ohne diesen zu kopieren
  - Updates in place
- Sparen Zeite und Platz
- Funktioniert mit Objekten

4.4 Attribute

- Variabel (*field*) innerhalb eines Objects
- Zugriff via Referenzvariabel
- "null" kann verwendet werden um Referenzvariabeln zurückzusetzen
- Dereferenzieren (*dereference*) wia Dotnotation durch Referenzvariabel auf Wert zugreifen

4.5 Methoden

- Instanzmethode existiert innerhalb jedes Objekts einer Klasse und beschreibt das Verhalten eines Objects
- Impliziter Parameter: Das Objekt für das die Methode aufgerufen wird
  - wird immer übergeben
  - System weiss nicht wie ein Objekt gedruckt werden soll. Eine custom printmethode löst das Problem
    - "toString()" wird immer verwendet wenn ein nicht String resp nicht primitive Type gedruckt wird

- Accessor Methode wird verwendet um auf den Zustand eines Objekts zuzugreifen
  - Read-only method
- Mutator Method wird verwendet um den Zusatand eines Objekts zu verändern

4.6 Konstruktor

- Wird durch "new" verwendet um ein neues Objekt zu erstellen
- Wir geben die Werte für die Attribute direkt mit bei der Initialisierung
  - Der Kustruktor verarbeitet diese Parameter
- Der Konstruktor hat keinen Typen, resp der Typ ist der Name der Klasse
- Der Konstruktor ist keine Methode
- Attribute die nicht durch Konstruktor gesetzt werden bekommen den Wert null
- Wir können mehrere Konstruktoren haben, jedoch dürfen sie nicht die slebe Anzahl Parameter haben
- in line if/Else: booleanExpression ? expression1 : expression2

4.7 Sichtbarkeit für Attribute

- Encapsulation
  - Objekte haben einen Zustand, nicht nur Klasse
  - Können angeben von wo man auf welches Attribut zugreifen kann
    - private: nicht von ausserhalb der Klasse zugreiffbar
      - Zugriff von Accessor und Mutator Methoden geht
- Bei Namenskonflikten "gewinnt" die innerte deklaration.
- *this* verweis innerhalb einer Methode auf das Objekt selbst

  - kann auch innerhalb eines Konstruktors auf einen anderen Konstruktor zugreifen *this(parameter1, parameter2)*

- Shaddowing:
  - zwei Veriabeln den selben Namen
  - Illegar in Java, ausser für Attribute
- public macht etwas von überall her sichtbar

4.8 Static Methoden und Variabeln

- Modul
  - Kein eigenständiges Programm, sondern wird von einem anderen Programm verwendet.
  - Kein Mail Methode
  - Wird via die Klassenreferenz verwendet
- static ist eine Ausnahme, normal ist mit Attributen
- Leicht die übersicht zu verlieren
  - wenig Gründe für static
    - z.b. bei final (Wert kann nach einmal setzten nicht mehr verändert werden)
- static sollte private oder final sein für attribute
- statische Methoden dürfen nicht this verwenden!
  - Objektattribute können nicht gelesen/geschrieben werden

Einschub Random und Math

Random

- liefert Pseudozufallszahlen

- teil von java.util

Method	Description
nextInt()	returns a random integer
nextInt(max)	returns a random integer in the range [0, max)
nextDouble()	returns a random real number in range [0.0, 1.0)

Math

- Sammlung von Mathe Operanden
- Teil von java.util
- auch wenn ein Returnvalue ein theoretischer Int ist, wird ein Double returned

4.3 Klassen (selber entwickeln)

- Objekte fassen zusammengehörende Daten zusammen
- Programm wird übersichtlicher und robuster

4.3.1 Einleitung

- Eine Art neue Typen zu beschreiben
- Objekt (object): Gebilde das Zustand (state) und Verhalten (behaviour) verbindet
  - stellt services zur Verfügung (Methoden)
- OOP: Program als Menge von aufeinanderwirkenden Objekte
- Klassen beschreiben Objekte
- Objekte sind Exemplate (Instances) einer Klasse
- OOP geht auch ohne Klassen, aber dies ist nicht verbreitet

4.3.2 OOP in der Praxis

- grosse Softwaresysteme

- Modellierungen realter Simulationen
- Abstraktion: lassen irrelevante Sachen weg in der Klasse
- *Client* verwendet eine bestimmte Klasse
- Klassen schreibt man gross
- Attribute werden wie bei Methoden übergeben
- Zugriff auf Atribute via dot dotation

5.1 Graphische Benutzeroberfläche

- GUI: Graphical User Interface
- Thema: Input / Output I/O
- Vorteile GUI
  - 2 oder 3 Dimensional
  - Eingabe durch Maus/Geste
  - Oft intuitiver und ansprechender
- Nachteil GUI
  - Höhere Komplexität
    - Kontrollfluss: Programm muss immer auf den Benutzer reagieren
    - Verschiedene Steuerelemente
    - An Fenstergrösse anpassen

Window

- new Window("name", b, h)
- Window Methods

open()	öffne Fenster
close()	schliesse Fenster
waitUntilClosed()	warte bis manuell geschlossen
isOpen()	überprüfe ob geöffnet
fillRect(x, y, h, b)	Zeichne Rechteck
setColor(r, g, b)	ändere Farbe für fenster
fillRect()	
fillCircle()	
fillOval()	
fillRect(x, y, 1, 1)	Zeichnen eines pixels -> frei Zeichnen
refresh()	änderungen werden angezeigt
refresh(int waitTime)	wartet eine best Anzahl milisekunden
refreshAndClear()	macht alles weiss und bemalt es dann wieder
isKeyPressed()	
isLeftMouseButtonPressed()	
isRightMouseButtonPressed()	
wasKeyTyped()	
wasMouseButtonClicked()	
getMouseX()	
getMouseY()	
drawRect()	
drawCircle()	
drawLine()	
drawString()	
drawImage()	
drawImageCentered()	
setColor()	
setStrokeWidth()	
setFontSize()	

- Input:
  - Tasten drücken
  - Mausclick

5.2 Input/Output mit Dateien

- import java.io.*;
- new File(fileName);
  - Kann auch Ordner handeln
- wird mit Scanner kombiniert um Files zu lesen. Dafür übergeben wir dem Scanner das File Object

  - jedoch throwt der Scanner eine Exception welcher gehandelt werden muss
    - falls man aus einer Dateil liest, welche nicht existiet
    - mit throws kann man sagen das eine Exception nicht gefangen werden soll, resp. weitergeleitet wird -> crash
      - public static void foo() throws type

exists()
canRead()
getName()
length()
rename()

- print(), println() kann auch zu File schreiben
  - übergeben File Object als Parameter
  - alte Datei wird überschrieben
- Alle System.out können über einen Alternativen Stream ausgegeben werden

5.3 Scanner im Einsatz

- jenachdem was wir für einen Typen werwarten benutzen wir eine andere lese Methode
- input cursor scannt denn input -> identifiziert ein Token (wo fängt z.B. der nächste Int an)
- Der Scanenr *konsumiert* das Token (liest es ein)
- mit $.hasNextDouble()$ kann mit einer Schleife eine unbekannte Anzahl Daten eingelesen werden

hasNext()	if there is a next token
hasNextInt()	if next token int exists
hasNextDouble()	if next token double exists

- $.next()$ geht zum nächsten Token ohne es zu konsumieren
- Zeilenbasierter Scanner kann verwendet werden wenn auf einer Zeile verschiedene Anzahle von verschienenen Typen existieren

nextLine()	Returns the entire line till
hasNextLine()	Checks if there is another line

  - wird konsumiert, jedoch nicht an die Methode übergeben

- Der Scanner auch auf einen String angewant werden

  - Dies ist hilfreich wenn wird z.b. mit einem Scanner die Zeilen aus einem File einlesen, und ein 2. Scanner verarbeitet die Zeilen (Strings) dann

- Wenn ein File bereits existsiert sollten wir e.v. ein neue File erstellen, da sonnst die File überschriben wird wenn der Scanner für den Output zuständig ist

6.0) Arbeiten mit Objekten und Klassen

6.1) Einleigung-Datenstruktur mit Verknüpfungen

- Arrays haben eine feste Strukture, jedoch wollen wir manchmal mehr "dynamisch" arbeiten können

- Listen lösen diese Problem

- Listen mit Construktoren sind mühsam

6.2) Entwurf von abgekapselten Klassen

- System.exit(-1) wird verwendet um ein Programm zu beenden

  - In einer return methode mussen wird ein "dummy" return einfügen damit der Compile nicht reklamiert

- Methoden können den selben Namen haben wenn sie sich in der Parametern unterscheiden

- Array Vorteile:

  - Konstante Zugriffszeit

- Vorteile List

  - Länge variabel

6.3) Hinweise und Regel für verständliche Programme

Code Struktur

- gut lesbar und kompakt
- Nicht mehr als 100 Zeichen Pro Zeile
  - neue Zeile für jede Anweisung
  - Keine unnötigen neuen Zeilen
- Aufeinandervolgende Anweisungen untereinander
- Blöcke einrücken
- Geschweifte Klammern verwenden
- Klammern setzen
- Lange if Statements mit oder oder und auf mehrere Zeilen
- main Methode entweder zuoberst oder zuunterst

Namen

- Nur Buchstaben und Ziffern ohne Spezialzeichen
- Klassennamen
  - gross
- Methodennamen
  - klein
  - Verb
- Variabelnnamen
  - beschreibend

- Kurze Namen für Loopcounter
- Keine Typ/Metainformationen im Namen

Weiteres

- Masseinheiten soll im Namen vorhanden sein
- Deklaration zusammen mit Initialisierung
- Bei Arrays Eckigen Klammern hiter Typ und nicht hinter Namen
- Durch Faktorisierung gemeinsame Codeelemente herausarbeiten -> Kompakterer Code
- Keine unnötigen mehrfachverechnungen in if Statements da es den Code länger macht
- Fragezeicheoperator: (test) ? value1 : value2
  - kicht so effizient aber kompakter

Speicher und Addressen

- In 4 Digit Hex angegeben
- Addressen sind in Java "unsichtbar"
- 4 Bytes werden zu einem Wort zusammengefasst und daher sind die Addressen in 4er Schritten
- Speicher ist in verschiedene Bereich aufgeteilt:
  - Variabeln die immer existierene: in "static data"
  - durch new Operator erschaffene daten: "heap"
  - aufgerufene Methoden: "Stack Frame"
    - In einem Stack organisiert
- Müssen stets genügend Platz frei haben damit sich der Stack und Heap nicht überlappen

- Reference Variabel enthält Addressen des Objekt Exemplars
- Stack zeigt auf Exemplar im Heap mittels eines Pointers
  - Gibt es Bereiche im Heap die keine Referenzen haben zählt es als garbage und wird von der garbage collection empfernt.
- Valuesemantics und Referencesemantics wird im Kern gleich behandelt

6.5) Mehr Optionen für Kontrolle der Sichbarkeit

- package ist eine Ansammlung von zusammengehörenden Klassen
  - Datei kann nur in einem Package sein
  - Klassen legen fest in welcher Package sie ist
  - Schaffen einen namespace um namenskonflikte zu vermeinden
- Package -> Verzeichnis
- Klasse -> Datei
- Root des Package wird durc den Class Path, oder Ausruferverzeichnis von java gegeben
- Class Path:
  - Path an welchem Java class Files sucht
  - Konfiguriert in Eclipse
- Package
  - Deklaration
    - Anfang der Datei
    - package packageName;
  - Import

        * import PackageName.*;
        * Namenskonflikte werden unterschiedlich behandelt
    - Können auch ohne import auf Packages zugreifen
        * Durch den vollen Namen
        * packagename.className
        * Praktisch um namenskonflikte zu vermeiden
    - Gliedern das Projekt
    - Default Package
        * alle Daten die nicht explizit in einem Package sind sind automatisch da
        * Können nicht importiert werden
        * Können nicht von anderen Klassen verwendet werden

- Access Modifiers:
    - Public: in allen Klassen nach Import
    - Private: nur in der Klasse selbst
    - default (package). In der Klasse und allen anderen Klasse des packages