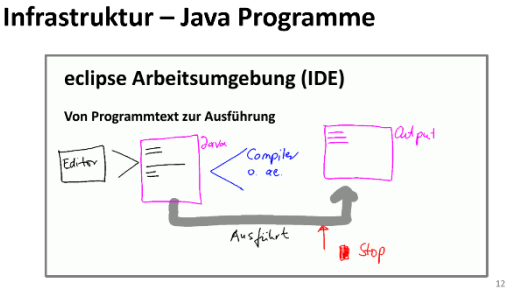
Eprog Einfache Java Programme

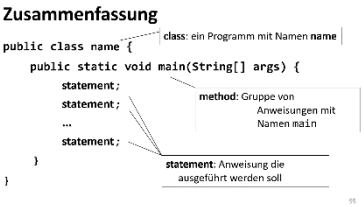
Java Programme

- Ganzes Programm
  - Braucht Compiler
- Für jede Datei…

Erstellen / übersetzen (compilieren)

  -
  - Ausführen
  - Modifizieren
- Einzelne Anweisunen
  - Braucht Shell
- Für jede Anweisung…
  - Read
  - Evaluate
  - Print
  - Loop
  - REPL
- Name des Programms gleich Name der Datei
- Viele (Textbearbeitungs)Methoden lassen das Objekt mit dem die Methode arbeitet unverändert
- Bezeichner: Muss mit Buchstaben (gross/klein) anfangen (oder _ $)

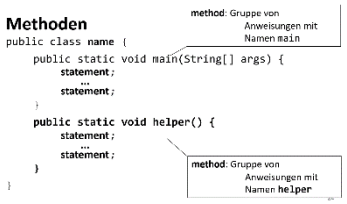




Sonderzeichen

\t	tab character
\n	neue Zeile (new line character)
\”	double quote character
\\	backslash character

Methoden



- Strukturieren die Anweisungen
- Wiederholungen zu vermeiden
- Main wird automatisch ausgeführt
- Aufrufen:
  - Object.methodName();
  - Ohne Objekt → mit static
- Abfolge der Ausführung von Anweisungen: Kontrollfuss (control flow)
- Java: Anweisungsreihenfolge ist explizit

Typen/Variablen

- Typ beschreibt Eigenschaften von Daten
  - Wertebereich

    - Operationen
    - Darstellung (welche Folge von 0 und 1 für einen Wert gewählt wird)
- Typen verhindern Fehler, erlauben Optimierung
- Beschreibt Menge (Kategorie) von Daten Werten
- Variable: benötigt Name und auf was für Werte sich die Variable beziehen kann

Modulo

- Finde letzte Ziffer einer ganzen Zahl
- Finde letzte 4 Ziffern
- Entscheide ob Zahl gerade ist
- Rangordnung (Precedence) ist wichtig: * / stärker als + -

Assoziativität («Associativity») -- Bindung

- **Die Assoziativität eines Operators ⊙ hält fest wie ein Operand zu verwenden ist:**

  $X \odot Y \odot Z$

- **Y ist mit dem *linken* Operator verknüpft: links-assoziativ («left-associative»)**

  $X \odot Y \odot Z = (X \odot Y) \odot Z$

- Y ist mit dem *rechten* Operator verknüpft: rechts-assoziativ («right-associative») $X \odot Y \odot Z = X \odot (Y \odot Z)$

Typ Umwandlungen

- Explizite Umwandlungen heissen cast, type cast
- (type) expression
    - (int) ((doube) 19/5)
- type ist Operator → rechts-assoziativ

Assoziativität

- **Links-assoziativ: Y ist mit dem *linken* Operator verknüpft («left-associative», «left-to-right associative")**

  $X \odot Y \odot Z = (X \odot Y) \odot Z$

  Viele der uns bekannten Operatoren: +, *,

- rechts-assoziativ: Y ist mit dem *rechten* Operator verknüpft ("right-associative», "right-to-left associative»)

  Später werden wir Beispiele sehen (es gibt einige!)

- **Es gibt Operatoren die sind *assoziativ* (in der Mathematik)**
    - Rechts—assoziativ *und* links—assoziativ: $(X \odot Y) \odot Z = X \odot (Y \odot Z)$

Variable

Name, der es erlaubt, auf einen gespeicherten Wert zuzugreifen

- Deklaration
- Initialisierung
- Gebrauch
- Zuweisung ist keine algebraische Gleichung!

Operanden und Operatoren

- **Operand wird vom Operator mit höherer Rang Ordnung («precedence», Präzedenz) verwendet**
- **Wenn zwei Operatoren die selbe Rang Ordnung haben, dann entscheidet die Assoziativität**
- **Wenn zwei Operatoren die selbe Rang Ordnung und Assoziativität haben, dann werden die (Teil)Ausdrücke von links nach rechts ausgewertet.**
- **Wenn etwas anderes gewünscht wird: Klammern verwenden!**

Deklaration

Variable müssen deklariert sein bevor sie verwendet werden können.

type name;

EBNF Description *variabledeclaration*

typeidentifier ⟸ bezeichner
variableidentifier ⟸ bezeichner
variabledeclaration ⟸ typeidentifier variableidentifier { , variableidentifier } ;

Ableiten mit Aussagen

- Positionen im Code haben Namen (Annahme)
    - Point A
    - Point B
- Alle Anweisungen die davor erscheinen, sind ausgeführt, wenn wir diesen Punkt erreichen
- Keine Anweisung danach wurde ausgeführt
- Hoare Logik:
    - Vorwärts und rückwärts schliessen
    - Von einer Anweisung zu mehreren Anweisungen und Blöchken
- Wichtig für die Definition von Schnittstellen (zwischen Modulen) wenn wir entscheiden müssen welche Bedingungen erfüllt sein müssen (um eine Methode aufzurufen).
- Vorwärts schliessen:
    - Simuliert die Ausführung des Programms (für viele «Inputs» «gleichzeitig»)
    - Bestimmt was sich aus den ursprünglichen Annahmen herleiten lässt
    - Sehr praktisch wenn eine Invariante gelten soll (Invariant = etwas, das sich nicht ändert)

Beispiel

- **Vorwärts schliessen**
    - Vom Zustand *vor* der Ausführung eines Programm(segments)
    - Nehmen wir an wir wissen (oder vermuten) $w > 0$

```
// w > 0
x = 17;
// w > 0  ∧  x == 17
y = 42;
// w > 0  ∧  x == 17 ∧  y == 42
z = w + x + y;
// w > 0 ∧ x == 17 ∧ y == 42 ∧ z > 59
```

    - Jetzt wissen wir einiges mehr über das Programm, u.a. $z > 59$

Beispiel

- **Rückwärts schliessen:**
    - Nehmen wir an wir wollen dass z nach Ausführung negativ ist

```
// w + 17 + 42 < 0
x = 17;
// w + x + 42 < 0
y = 42;
// w + x + y < 0
z = w + x + y;
// z < 0
```

- Rückwärts schliessen
    - Bestimmt hinreichende Bedingungen

Pre- und Postconditions

- Precondition: notwendige Vorbedingung, die erfüllt sein müssen (vor Ausführung einer Anweisung)
- Postcondition: Ergebnis der Ausführung (wenn Precondition erfüllt)
- Precondition, Anweisung und Postcondition hängen zusammen
- Aussagen (Pre, Post) sind logische (bool'sche) Ausdrücke die sich auf den Zustand eines Programms beziehen
- Zwischen { und } steht eine logische Aussage

Hoare Tripel

- Zwei Aussagen und ein Programmsegment
    - {P} S {Q}
        - P Precondition
        - S Statement
        - Q Postcondition
    - Gültig:
        - Zustand P gültig, Ausführung von S gibt Zustand Q
        - Wenn P wahr ist vor der Ausführung von S, dann muss Q nachher wahr sein
-
-