II.2.2 Methoden, Unterprogramme und Parameter

Freitag, 26. Oktober 2018 13:30

Unterprog = parametrisierter Anweisungsblock mit Namen

→ kann über seinen Namen aufgerufen werden. Nach der Ausführung des U-Programms wird Programm an der Aufrufstelle fortgesetzt.

Ablauf des Methodenaufrufs

- Ausdrücke im akt. Parametern werden ausgewertet.
- Parameterübergabe: formale Parameter bekommen die Werte der akt. Parameter

- Rumpf der aufgerufenen Methode wird ausgeführt.
- Ausführung der Methode kann Wert zurückliefern ("return").
- Danach wird Prog. hinter der Aufrufstelle fortgesetzt.

2 Arten von Methoden

- Funktionen (Rückgabetyp ≠ void). Sie liefern (mit return) ein Ergebnis zurück, wobei "return" die Methode beendet.
- Prozeduren (Rückgabetyp void).

Sinn von Prozeduren:

- Ausgabe (z.B. drucke)

- Seiteneffekte (z.B. sortiere)

↑

Um dies zu verstehen, betrachten wir die Parameterübergabe genauer.

In der Informatik gibt es verschiedene Arten der Parameterübergabe:

- call by value ← typisch
- call by reference ← für imperative Prog.

call by name ← eher in funktionalen Prog.-Spr.

Call by value

- aktuelles Parameter (im Methodenaufruf) wird ausgewertet (s hat den Wert 2.1)
- Wert des akt. Parameters wird in formalen Parameter

wird in formalen Parameter der Methode kopiert.
(r wird auf 2.1 gesetzt)

- Änderungen des formalen Parameters in der Methode bewirken keine Änderung des akt. Parameters.
(Zum Schluss hat s immer noch den Wert 2.1.)

Speicherverwaltung bei Blöcken / Methoden

- Jedem Block entspricht ein Speicherbereich (frame) auf dem Laufzeitkeller (run-time stack), auf dem die Werte der lokal deklarierten Variablen abgelegt sind.

gelegt sind.

- Bei Eintritt in Block wird dazugehöriger Speicherbereich oben auf dem Laufzeitkeller angelegt. ("oben" in Richtung des Kellerwachstums). Wenn Kontrollfluss den Block verlässt, wird Speicherbereich wieder freigegeben.
- Zuletzt angelegter Speicherbereich wird als erstes wieder freigegeben (Last In - First Out).
- Jede Variable ist nur in dem Block zugreifbar, in dem sie deklariert wurde und in dessen Unterblöcken.
- Java verbietet die Deklaration gleicher Variablen-

ration gleicher Variablen-
namen in Unter-Blöcken:

```
int x;
{ double x;   ← nicht erlaubt
  :
}
```

Laufzeitkeller bei Methoden

- Methodenrümpfe sind auch Blöcke
  ⇒ beim Methodenaufruf wird ein neuer Speicherbereich auf Laufzeitkeller angelegt, in dem die Variablen (und formalen Parameter) der Methode gespeichert werden.

Anders als bei geschachtelten Blöcken:

- Variablennamen in verschiedenen Methoden können gleich sein.
- Aufgerufene Methode kann nicht auf Variablen der aufrufenden Methode zugreifen.

In Java

- bei primitiven Datentypen findet cbv-Parameterübergabe statt.
- bei anderen Datentypen findet "eine Art call-by-reference-Parameterübergabe" statt.

Call-by-reference

- aktueller Parameter ist Variable (z.B. s)

Variable (z.B. s)

- formaler Parameter (r) wird Verweis auf den aktuellen Parameter.
- Jede Änderung des formalen Parameters bewirkt Änderung des akt. Parameters.

(Zum Schluss wäre s = 4.6.)

In Java

- Wenn formaler Parameter nicht von prim. Datentyp ist, dann enthalten akt. und formaler Parameter beide Verweis (Referenz) auf das Objekt.
- Methodenaufruf: Wert des akt. Param. wird in formalen Param. kopiert.

formalen Param. kopiert.
⇒ Beide zeigen auf das gleiche Objekt.

- Wird das Objekt in der Methode geändert, dann hat das Auswirkungen auf die aufrufende Methode.
(⇒ Seiteneffekt)

Java macht nicht wirklich cbr, sondern nur cbv (aber man kann damit cbr zum Teil simulieren, weil Objekte über Referenzen gespeichert werden).

Falls dem formalen Parameter ein anderer Wert zugewiesen wird, beeinflusst

dies den akt. Parameter nicht.

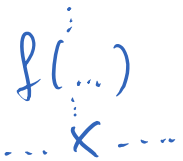
Programmierstil:

Wenn man Information in Methode übergeben will, dann sollte dies über Parameter erfolgen und nicht über globale Variablen:

```
    ⋮
public static int x;
    ⋮

public static int f(...){
        ⋮
        x = x + 1;
}
    ⋮

public static void main(..){
        ⋮
    ..x...
        ⋮
    f(...)
```



}

Bsp. für Methode mit (gewünschtem) Seiteneffekt:

sortiere

Vararg-Parameter

- erlauben es, Methoden mit variablem Anzahl an formalen Parametern zu definieren
- aktuelle Parameter dürfen Liste v. Array-Elementen sein

static / nicht-static

- nicht-statisches Attribut/Methode: Eigenschaft des Objekts, Attributwerte

des Objekts, Attributwerte können für versch. Objekte unterschiedlich sein.

```
r.laenge = 2.5;
s.laenge = 2.1;
r.flaeche()
```

↑ vor . steht das Objekt, zu dem das Attribut bzw. die Methode gehört.

Bei Aufruf ohne . ist die Eigenschaft des aktuellen Objekts gemeint (in nicht-statischen Methoden).

- statisches Attribut/Methode: Eigenschaft der Klasse

Bisher:

* Klassen, in denen alles nicht-statisch war (Datenstrukturen wie z.B. Rechteck)
* Klassen, in denen alles statisch war (Sammlung von Unterprogrammen).

Jetzt: Klassen, die sowohl statische als auch nicht-statische Komponenten haben.

Rechteck.flaechenberechnung

↑

vor. steht eine Klasse, kein Objekt.

Hier: flaechenberechnung zählt mit, wie oft die

zählt mit, wie oft die Methode "flaeche()" aufgerufen wird.
Bsp hat die Ausgabe
0
1
2

Beim Aufruf nicht-statischer Attribute/Methoden ohne . : aktuelle Klasse ist gemeint.

Aber:
bei statischen Eigenschaften ist auch ein Objekt vor . erlaubt. Gemeint ist dann die Klasse des Objekts:

Rechteck.flaechenberechnung
= r.flaechenberechnung
= s.flaechenberechnung

r.laenge

≠ s.laenge

Rechteck.laenge verboten

Zugriff auf nicht-statische Eigenschaften ist nur über Objekt möglich

Generell

- Alle Methoden, die auf Objekt-Attribute zugreifen, sollten nicht-statisch sein (drückt eine Eigenschaft des Objekts aus)

Bsp: flaeche(), toString()

wird von
System.out.print
aufgerufen.
⇒ Wenn man für eigenen
Datentyp eine solche
toString-Methode
implementiert, dann kann
man System.out.print für
Objekte des eigenen Daten-
typs aufrufen.

Aufzählungsdatentypen

illustriert Verwendung von
statischen Attributen
(enumerations)

Attribute des enum-Typs
sind Konstanten. Sie sind
~~statisch~~ (d.h. sie hängen nur
v. Klasse ab).

Man kann keine weiteren Objekte eines enum-Typs erzeugen:

```
Tag t = new Tag();
```

verboten

Für jeden enum-Typ werden bestimmte Methoden automatisch erzeugt.

enum-Klassen dürfen bel. statische + nicht-statische Methoden + Attribute enthalten.

switch-Anweisung ist nicht nur für int-, char- und String-Werte verwendbar, sondern auch für enum-Typen.

Gültigkeit von Bezeichnern

Was passiert, wenn Bezeichner mehrfach verwendet werden?

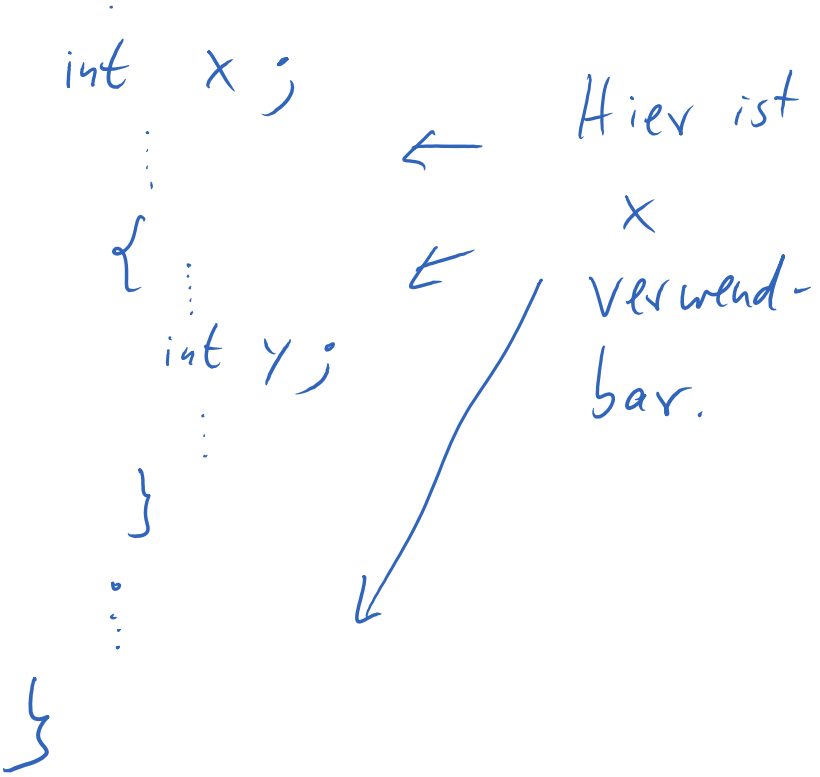
Gründe:

- Modularität: Verschiedene Programmteile werden v. versch. Programmierern entwickelt.
  ⇒ Verwendung gleicher Bezeichner kaum vermeidbar.
- Sinnvolle Wiederverwendung gleicher Bezeichner kann die Lesbarkeit erhöhen.

Regeln

- Jeder Bezeichner gehört zu dem Block, in dem er deklariert wurde.

```
{ ⋮
  int x;
```

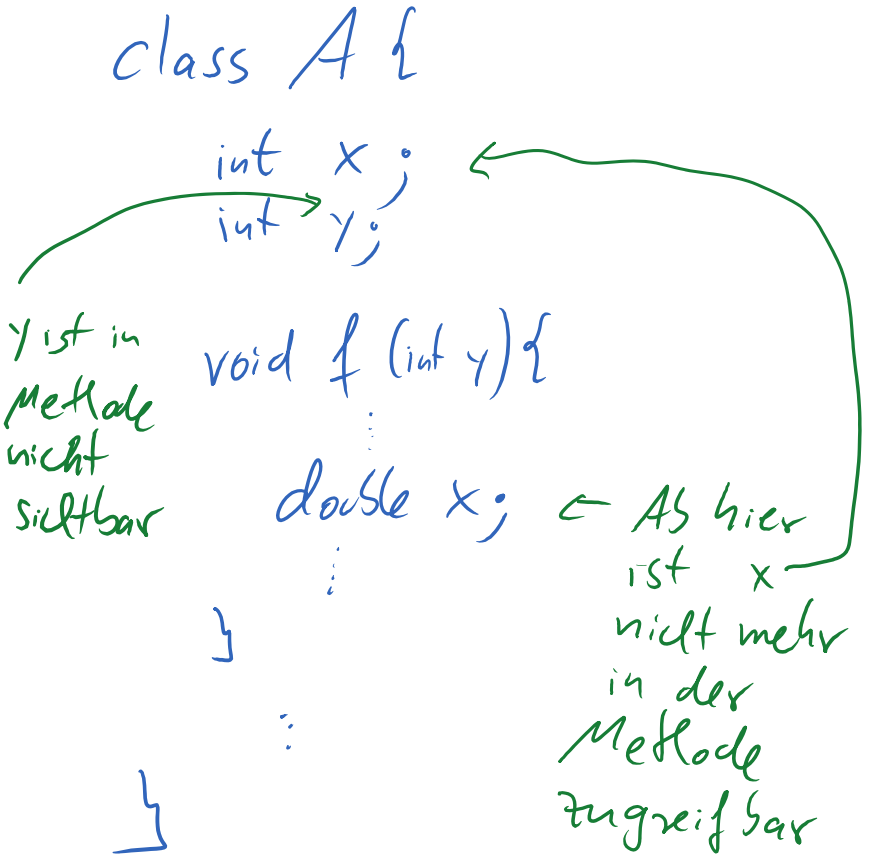


- Variablenbezeichner sind erst nach ihrer Deklaration verwendbar. Bei Methoden + Klassen spielt die Reihenfolge keine Rolle.

```
void f(...) {
    ...
    g(...)
    ...
}
```

```
void g(..) {
    :
    f(..)
    :
}
```

- Namensgleiche Bezeichner in einer Methode überdecken Bezeichner außerhalb der Methode.



- In einem Block müssen Bezeichner verschieden sein,

es sei denn, sie bezeichnen verschiedenartige Programmelemente.

```
{   int x;
    double x;   verboten
}

{   int x;
    :
    void x(...) {... }
    :
}
```

Klasse, Methode, Variable könnten gleich heißen (nicht unbedingt guter Stil).

- Methoden mit "genügend verschiedenen" formalen Parametern dürfen gleich heißen (Überladen von

Methoden → später)

Jede Java-Datei darf höchstens eine public-Klasse enthalten (und muss so heißen, wie die public-Klasse).

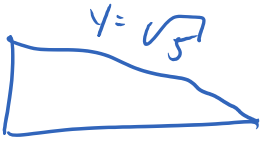
Bsp-Programm:

- stat. Variable x gilt in der ganzen Klasse Gültigkeit.
- formaler Parameter x der Methode main überdeckt diese statische Variable.
- In Methode setze bezeichnen x, y, z die formalen Parameter der Methode. Aber d.x bedeutet die Eigenschaft* (nicht-stat. Attribut)

genschaft* (nicht-stat. Attribut) des Objekts d.

- In der 1. Zeile von flaeche bezeichnen x, y, z die Attribute des jeweiligen Objekts.
  Ab der 2. Zeile überdeckt die lokale Variable y das gleichnamige Objekt-Attribut.

Bsp: z = 1



hat die Fläche 1