II.2.2 Methoden, Unterprogramme und Parameter

Freitag, 3. November 2017 08:30

Methoden: Unterprogramme

Unterprogramm:
- parametrisierter Anweisungsblock mit Namen
- Aufruf über Namen
- Parameter dienen dazu, Information v. aufrufender Stelle an U-Programm zu geben
- Nach Beendigung d. U-Prog wird Ausführung an der aufrufenden Stelle fortgesetzt.

Bei Aufruf einer Methode:
- Zuerst wird der aktuelle Parameter ausgewertet
  (Im Bsp: 1000 + 570.22 wird berechnet)
- Die Werte der aktuellen Parameter werden den formalen Parametern zugewiesen. (Parameterübergabe)
  (Im Bsp: Kapital wird auf 1570.22 gesetzt.)
- Rumpf der aufgerufenen Methode wird ausgeführt.

Bei nicht-void Methoden: "return" beendet die Methode und liefert Wert seines Ausdrucks zurück.
(Im Bsp wird 1617.3266 zurück geliefert.)
- Danach wird das Prog. an der aufrufenden Stelle fortgesetzt.

Methoden ohne Ergebnis (void): Prozeduren
Methoden mit Ergebnis (nicht-void Rückgabetyp): Funktionen

Sinn von Prozeduren

- Ausgabe

- Ausgabe
- Seiteneffekte

Bsp für Ausgabe-Prozedur: drucke
Bsp für Seiteneffekt-Prozedur: sortiere

Hierfür: betrachte Parameterübergabe genauer

In der Informatik existieren 3 wesentliche Mechanismen zur Parameterübergabe:

- call by value } in imperativen Sprachen verbreitet
- call by reference }
- call by name ← vor allem bei fkt. Progsprachen (z.B. Haskell)

Call by value

- Aktueller Parameter (im Methodenaufruf) wird erst ausgewertet.
- Wert des akt. Parameters wird dann in formalen Parameter der Methode kopiert.
- Änderungen des formalen Parameters in der aufgerufenen Methode bewirken keine Änderung des aktuellen Parameters.

In Java findet bei primitiven Datentypen immer cbv-Parameterübergabe statt.

Speicherverwaltung bei Methodenaufrufen

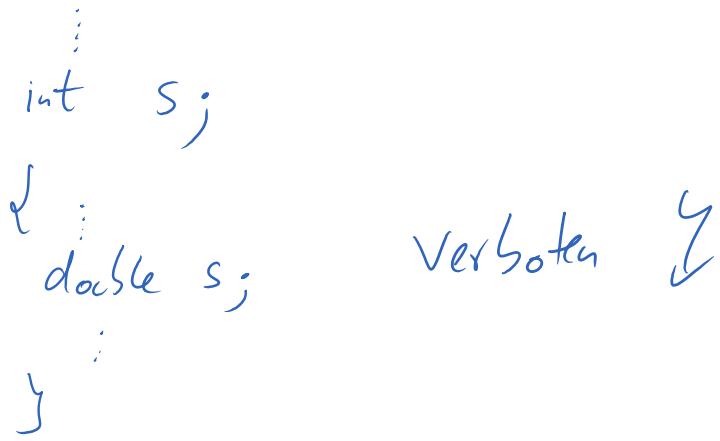
- Stack (bzw. Runtime Stack / Laufzeitkeller) speichert die Werte der Variablen.
- Immer wenn ein neuer Block betreten wird, wird "oben" auf dem Stack ein neuer Speicherbereich angelegt (Frame), in dem die Variablen gespeichert werden, die in diesem Block deklariert werden.
- Beim Verlassen des Blocks wird sein Speicherbereich wieder gelöscht. Dh: zuletzt angelegter Speicherbereich wird als erstes wieder gelöscht ("last in, first out").
- Jede Variable ist nur in dem Block zugreifbar, in dem sie deklariert wurde und in dessen Unterblöcken.
- Rumpf einer Methode ist auch ein Block ⇒ Laufzeitkeller-Prinzip wird auch hier genutzt.
  (In der aufgerufenen Methode kann man aber nicht auf Variablen der aufrufenden Methode zugreifen.
  Dafür sind in der aufgerufenen Methode auch Variablen erlaubt, die den gleichen Namen haben wie in der aufrufenden Methode.)

Dh:

```
public static void main(...) {
    int s;
    ...
    f(s);
    ...
}
```

erlaubt ✓

```
... f(...) {
    ...
    double s;
    ...
}
```



Call-by-reference

- Aktueller Parameter ist eine Variable.
- Formaler Parameter wird Verweis auf den aktuellen Parameter.
- Jede Änderung des formalen Parameters der Methode bewirkt Änderung des aktuellen Parameters.

In Java findet bei nicht-primitiven Datentypen "eine Art" Call-by-reference Parameterübergabe statt:

- Bei solchen Datentypen wird der Wert des aktuellen Parameters (and) in den formalen Parameter beim Methodenaufruf kopiert.
- Da der aktuelle Parameter die Adresse des Objekts enthält, zeigen danach aktueller und formaler Parameter auf das gleiche Objekt.
- Wenn das Objekt über den formalen Parameter in der aufgerufenen Methode verändert wird, dann hat das auch Auswirkungen auf die aufrufende Methode (Seiteneffekt)

Wirkungen auf die aufrufende Methode (Seiteneffekt)

Bei Rechteck s;

("null" = Zeiger ins Leere,
kann im Java-Prog. benutzt werden.
z.B. if (s == null) {...})

Hiermit kann man cbr simulieren.
(Es ist aber eigentlich cbv, wobei die Variablen bei nicht-primitiven Datentypen die Adresse des Objekts speichern.)

Unterschied zu echtem cbr

- Wird dem formalen Parameter ein neues Objekt zugewiesen, so ändert dies nichts am aktuellen Parameter in Java.
- Bei echtem cbr würden aktueller u. formaler Parameter komplett identifiziert. Gibt es in Java nicht.

Zusammenfassung

In Java hängt es v. Datentyp ab, ob Variable Werte oder Referenzen speichert.

Bsp für Prozedur mit Seiteneffekten: sortiere
Bsp illustriert die "cbr-ähnliche" Parameterübergabe
(gleiches Sortierverfahren wie früher).

Vararg-Parameter

- erlauben es, Methoden mit variabler Anzahl von formalen Parametern zu deklarieren.
- Aktuelle Parameter dürfen Liste von Array-Elementen sein
- Vararg-Parameter dürfen nur als letzte formale Parameter kommen.

statische / nicht-statische Methoden u. Attribute

static: Methode bzw. Attribut hängt nur von der Klasse ab

nicht-static: —"— hängt v. jeweiligen Objekt ab.

Insbesondere können die Attribute bei jedem Objekt unterschiedliche Werte haben.

z.B. r.laenge
r.flaeche()
(greift auf laenge
u. breite von r zu)

Bisher:

- Klassen, in denen alles

- Klassen, in denen alles statisch ist (z.B. Sort).
  ⇒ reine Sammlung v. Unterprogrammen
- Klassen, in denen nichts statisch ist (z.B. Rechteck)
  ⇒ reine Datenstrukturen

Aber: Klasse kann sowohl statische als auch nicht-statische Komponenten haben.

z.B. statisches Attribut flaechenberechnung, das protokolliert, wie oft eine Fläche berechnet wird.
Bsp gibt aus: 0 1 2

Zugriff auf nicht-statische Eigenschaften über Objekte:

r.flaeche
s.laenge
↑
Objekt

In nicht-statischer Methode kann man direkt auf nicht-statische Eigenschaften der aktuellen Objekte zu-

nicht-statische Eigenschaften des aktuellen Objekts zugreifen (z.B. "laenge" in der Methode "flaeche").

Zugriff auf statische Attribute/Methoden über Klassen:

Rechteck.flaechenberechnung

Sort.sortiere

↑

Klasse

Wenn man ....• weglässt, ist die Methode/Attrib. aus der aktuellen Klasse gemeint.

Es ist auch

r.flaechenberechnung

erlaubt. Dies ist gleichbedeutend mit

Rechteck.flaechenberechnung

↑

Klasse des Objekts r

Generell:

Alle Methoden, die eine Eigenschaft des Objekts ausdrücken (d.h., die auf die Objekt-Attribute lesend

die Objekt-Attribute lesend oder schreibend zugreifen), sollten nicht statisch sein.

Bsp: flaeche, toString

Konvertiert Objekt in String. Wird von System.out.print aufgerufen.

Aufzählungstypen (Enumerations)

- illustriert Verwendung v. stat. Attributen
- enum ist Kurzschreibweise für eine Klasse mit wenigen statischen Attributen
- Aber: Man kann von enums keine weiteren Objekte mehr erzeugen.
- Außerdem existieren für alle Aufzählungstypen bestimmte vordef. Methoden. (z.B. ordinal(), gibt den Index des jeweiligen Objekts im values()-Array zurück)

gibt im values() Array zurück).

- toString ist auch vordef. für Aufzählungstypen
- switch ist nicht nur für int-, char-, String möglich, sondern auch für Aufzählungstypen.

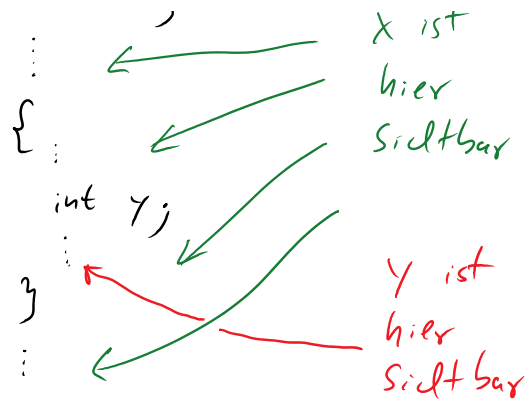
Gültigkeit v. Bezeichnern

- Welche Bezeichner sind wo sichtbar, wenn sie gleich heißen?
- Grund für Wiederverwendung v. Bezeichnern
  - Lässt sich kaum vermeiden bei modularer Prog-Entw.
  - Kann Lesbarkeit erhöhen

Regeln

- Alle Bezeichner müssen vereinbart werden.
  Jede Variable gehört zum innersten Block, in dem sie vereinbart wurde.

```
{ ⋮
  int x;
  ⋮   <-- x ist hier
  ^
```



}

- Variablen-Bezeichner kann erst nach seiner Deklaration benutzt. Aber Methoden u. Klassen sind im ganzen Bereich ihrer Deklaration benutzbar.

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

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

- Namensgleiche in Methode überdecken Bezeichner außerhalb der Methode.

```
    :
    int x;
  void f() {
```

```
int x;
void f(..) {
    double x;   ← ab hier bezeichnet x die double-Var. der Methode
    :
}
```

- Bezeichner müssen verschiedene Namen innerhalb eines Bereichs haben. Ausnahme: Bezeichner für verschiedenartige Prog.-Elemente dürfen gleich heißen.

```
int x;
double x;
```
verboten

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

Es darf gleichnamige Variablen, Methoden, Klassen geben.

- Methoden dürfen den gleichen Namen haben, wenn ihre formalen Parameter "unterschiedlich genug" sind.

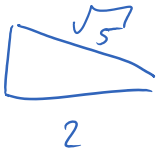
"unterschiedlich genug" sind.
(Überladung v. Methoden → nächste Vorlesung).

Bsp-Prog:

- stat. Var. x gilt in ganzer Klasse Gueltigkeit
- Formaler Par. x der Methode überdeckt die stat. Var. x
- Jede java-Datei darf mehrere Klassen enthalten, aber höchstens eine public-Klasse.
  Dateiname = Name der public-Klasse.
- Objektvariablen x, y, z gelten in Klasse Dreieck.
- In Methode setze werden x, y, z von den formalen Par. überdeckt.
  d.x ist das x-Attribut des Objekts d
- In Methode flaeche sind zunächst x, y, z die Objektattribute.
  Aber ab der Deklaration
  double y = ...
  ↖

Klasse y - ...

überdeckt dieses y die Objektvariable y.

- Dreieck 

hat Fläche 1.

Man sollte Bezeichner nur dann wiederverwenden, wenn es Verständlichkeit erhöht.