Einführung in die Programmierung für NF

Rückgabewerte, Fallunterscheidung, Schleifen

FUNKTIONEN UND PROZEDUREN

Funktion und Prozedur

- Methoden bestehen aus Funktionen (Ausdrücken) und Prozeduren (Anweisungen)

Funktion

- Funktionen beschreiben mathematische Algorithmen
- Eine Funktion kann auf viele Werte Angewendet werden

Funktion - Beispiel

Kreiszylinder

- Mathematische Funktion
  volumen(r,h) = $r^2 * \pi * h$

- Methode

```
public double zylinVolume (double radius, double high){
    return radius * radius * Math.PI * high;
}
```

- Methodenaufruf

```
zylinVolume (1.5, 1.0);      ->  4,71
```

Funktion

- Null, einer oder mehrere Parameter
- Der Funktion ist der **Ergebnistyp** vorangestellt, d.h. der Typ der Werte die als Resultat geliefert werden
- Der Rumpf einer Funktion ist ein Ausdruck in dem die Parameter vorkommen
- Das Ergebnis wird durch **return** gekennzeichnet
- Beim Aufruf müssen so viele Argumente, wie die Funktion Parameter hat, vorhanden sein

Prozedur

- Prozeduren sind Methoden, die keine Ergebnisse berechnen
- Prozeduren lösen Aktionen aus

Prozedur - Beispiel

Die Methode soll einen Punkt drucken

- Methode

```
public static void paintPoint () {
    System.out.print (".");
}
```

- Methodenaufruf

```
paintPoint();
```

Prozedur

- Prozeduren sind wie Funktionen, sie geben aber kein Ergebnis
- Da die Prozedur kein Ergebniss hat wird ihr der Pseudo-Typ **void** vorangestellt
- Im Rupf steht **kein** return

FALLUNTERSCHEIDUNG

Fallunterscheidung

- Die **Fallunterscheidung** (*Conditional*) in Java hat die Form
  - **if** (*BoolExpression*) *Statement*

  bzw.
  - **if** (*BoolExpression*) *Statement* **else** *Statement*

- **Beispiel:** Kontofluss 1

```
if (kontoStand >= betrag)
    kontoStand = kontoStand - betrag;
```

- **Beispiel:** Kontofluss 2

```
if (kontoStand >= betrag)
    kontoStand = kontoStand - betrag;
else
    kontoStand = kontoStand - betrag - UEBERZIEH_GEBUEHR;
```

Was ist hier falsch?

```
if (kontoStand >= betrag)
    gebuehren = gebuehren + TRANSAKTIONS_GEBUEHR;
    kontoStand = kontoStand - betrag;
```

Was ist hier falsch?

```
if (kontoStand >= betrag)
   gebuehren = gebuehren + TRANSAKTIONS_GEBUEHR;
   kontoStand = kontoStand - betrag;
```

Das if bezieht sich nur auf die erste Anweisung, der Betrag wird immer vom Kontostand abgezogen.

```
if (kontoStand >= betrag)
   gebuehren = gebuehren + TRANSAKTIONS_GEBUEHR;

kontoStand = kontoStand - betrag;
```

Mehrfache Anweisungen

Blockklammern sind nötig, wenn die Fallunterscheidung mehrere Anweisungen umschließen soll.

Beispiel:

```
if (kontoStand >= betrag) {
 kontoStand = kontoStand - betrag;
 gebuehren = gebuehren + ABHEBE_GEBUEHR;
}
else {
 kontoStand = kontoStand – betrag - UEBERZIEH_GEBUEHR;
 gebuehren = gebuehren + UEBERZIEH_GEBUEHR;
}
```

Dangling else

```
if (!kontoGesperrt)
  if (kontoStand >= betrag) {
        kontoStand = kontoStand - betrag;
        System.out.println("Abhebung erfolgreich.");
  }
else System.out.println("Abhebung nicht erlaubt.");
```

Vorsicht! Das else bezieht sich auf das zweite if.

→ In Java schreibt man bei if (fast) immer Blockklammern, auch wenn der Block nur eine einzige Anweisung enthält.

SCHLEIFEN

Iteration

Drei Konstrukte zur Iteration

- while
- for
- do (behandeln wir nicht)

While-Schleifen

- Die **while**-Schleife (Nichtterminalsymbol *Iteration*) hat die Form

  while (*BoolExpression*) *Statement*

Beispiel:

```
int n = 1;
int end = 10;
while (n <= end) {
   System.out.println(n);
   n++;
 }
```

Beispiel:

```
int qs = 0;
int x = 352;
while (x > 0) {
    qs = qs + x % 10;
    x = x / 10;
}
```

for- Schleifen

- Die häufigste Form der while-Schleife ist:

```
int i = start;          // Initialisierung
while (i < end) {       // Bedingung
...
i++;                    // Zählerkorrektur durch konstante Änderung
}
```

- Abkürzende Schreibweise:

```
for (int i = start; i < end; i++) {
...
}
```

for-Schleifen

- Beispiel:

```
int end = 10;
for (int i = start; i < end; i++) {
 System.out.println(i);
}
```

- Allgemein hat eine **for**-Schleife die Gestalt

  for (Initialisierung; Bedingung; Zählerkorrektur) *Statement*

  Dabei wird zunächst die Initialisierung ausgeführt.

  Dann wird, solange die Bedingung wahr ist, *Statement* ausgeführt und der Zähler geändert (gemäß des Zählerkorrektur-*Statement*).

- Eine in der Initialisierung deklarierte Variable ist nur im Schleifenrumpf gültig.

for-Schleifen

- Guter Stil ist es, **for**-Schleifen nur folgendermaßen zu schreiben:

  for (Setze Zähler auf Startwert;

  Teste, ob Zähler den Endwert erreicht hat;

  ändere Zähler) {

  ...

  // Zähler, Startwert, Endwert und

  // Inkrement werden in diesem Block

  // nicht verändert.

  }

- Die Zählervariable sollte bei der Schleifeninitialisierung deklariert werden.

Vielen Dank für Ihre Aufmerksamkeit