Einstieg in die Informatik mit Java

Methoden

Gerd Bohlender

Institut für Angewandte und Numerische Mathematik

Gliederung

Gliederung

Methoden

Methoden bilden das Analogon zu den aus Pascal, C und C++ bekannten *Funktionen* und dienen dazu, mehrfach vorkommenden Quelltext zusammenzufassen und Programme übersichtlicher zu gestalten.
In diesem Abschnitt werden zunächst nur statische Methoden, d. h. nicht objektorientierte Methoden, behandelt, die aus diesem Grunde mit dem Modifizierer `static` markiert werden.

Syntax

Modifizierer Typspezifikation Methodenname
`(`*formale Parameterliste*`) {`
Deklarationen und Anweisungen
`}`

Gliederung

Methodendefinition

Die formale Parameterliste setzt sich jeweils aus der Typspezifikation gefolgt vom zugehörigen Bezeichner zusammen und wird durch Kommata getrennt:
Typ 1 Name 1, . . . , *Typ n Name n*

- Die formale Parameterliste darf leer sein, die runden Klammern hingegen müssen stehen.
- Für jeden Parameter muss eine eigene Typspezifikation existieren.
- Feldtypen, Zeichenketten und beliebige andere Objekte sind als Parameter und Ergebnisse zugelassen. Bei Feldern muss die korrekte Anzahl von eckigen Klammern, jedoch ohne die Dimension, zusammen mit dem Basistyp angegeben werden.

Methodendefinition

- Die Rückgabe des Ergebnisses erfolgt über

  `return` *Ausdruck*`;`

  Der Ausdruck muss dabei zuweisungskompatibel zum Typ der Methode sein.
- Mit der `return`–Anweisung wird die Methode abgebrochen und das angegebene Ergebnis zurückgeliefert. Insgesamt muss ein `return` durchlaufen werden, sonst tritt ein Fehler auf.

Achtung

Dies ist eine beliebte Fehlerquelle bei der Verwendung von bedingten und verzweigten Anweisungen!

Methodendefinition

- Eine Schachtelung von Methoden ist im Gegensatz zu Pascal und C++ nicht erlaubt. Methoden müssen stets auf äußerster Ebene in einer Klasse definiert werden.
- Wird kein Ergebnis benötigt, so ist der Ergebnistyp `void` und die `return`–Anweisung lautet schlicht:

  ```
  return;
  ```

  Diese Anweisung muss nicht explizit angegeben werden, sondern wird am Ende der Methode automatisch ausgeführt.

Methodendefinition

Beispiel

Methode zur Berechnung von x^n mit n ganzzahlig

```
static double power (double x, int n) {
    int m;
    double y = 1.0;
    if (n>=0) m = n;
// Absolut-Betrag von n
    else m = -n;
    // Berechnung von x^m
    for (int i=0; i<m; ++i) y*=x;
    if (n>=0) return y;
    else return 1.0/y;
}
```

Der Aufruf erfolgt dann im Hauptprogramm über:

```
double y = power (a,5);
```

Die Methode `Math.pow()` bestimmt ebenfalls x^y, wobei y dann auch eine Gleitkommazahl sein darf.

Gliederung

Parameterübergabe, Methodenaufruf

Syntax

Methodenname (*aktuelle Parameterliste*)

- Die aktuelle Parameterliste muss genauso viele Parameter besitzen wie die formale Parameter–Liste.
- Im Gegensatz zu Pascal und C++ werden alle Parameter einer Methode als *Werteparameter* übergeben.
- Die aktuellen Parameter werden von links nach rechts ausgewertet.
- Beim Aufruf der Methode wird das aktuelle Argument ausgewertet und das Ergebnis in einer lokalen Variablen mit dem Namen des formalen Parameters abgelegt, d. h. es wird stets mit einer Kopie der übergebenen Daten gearbeitet.

Parameterübergabe, Methodenaufruf

- Weiterhin muss der Methodenaufruf in einem Ausdruck des entsprechenden Typs erfolgen.
- Ist die formale (und damit auch die aktuelle) Parameterliste leer, so müssen die runden Klammern trotzdem stehen.
- Der aktuelle Parameter lässt sich bei einer Übergabe als Werteparameter nicht verändern. Für dieses Vorhaben gibt es in Java Referenztypen.

Beispiel

```
static int f (int i, int j) { //Definition von f
   return (i+j);
}
...                                  //im Hauptprogramm:
int i = 1;
System.out.println(f(i++,i)); //Aufruf: f(1,2) ergibt 3
```

Gliederung

Referenztypen bei Methoden

Objekte, also auch Felder und Zeichenketten, sind Referenztypen. Auch sie werden per Wertaufruf übergeben, d. h. die Referenz wird in eine lokale Variable kopiert. Folglich kann die aktuelle Größe im aufrufenden Programm nicht verändert werden.

Achtung

Die Werte, auf die die Referenz verweisen, werden hingegen nicht kopiert und können deshalb verändert werden!

Referenztypen bei Methoden

Beispiel

```
static void demo1 (int[] x) {
    x[0]++;
}
...                              // im Hauptprogramm:
int[] y = new int[] {1, 2};
demo1 (y);                       // Aufruf der Methode demo1
                                 // x und y zeigen beide auf
                                 // gleiche Speicheradressen!
                                 // x=y={2,2}
```

Achtung

Dies entspricht aus den folgenden beiden Gründen nur teilweise einem Referenzaufruf von Pascal (`var`) oder C++ (`&`).

Referenztypen bei Methoden

(1) Es besteht keine Wahlmöglichkeit. Grundtypen sind immer Werte, Objekte immer Referenzen.

(2) Auch bei Referenztypen kann das Objekt selber (d. h. die Referenz) nicht verändert werden, nur seine Komponenten.

Beispiel

```
static void demo2 (int[] x) {
    x = new int[] {4, 5, 6};   //x komplett neues Objekt
    System.out.println (x[0]);
}
...                            //im Hauptprogramm:
int[] y = new int[] {1,2};
demo2 (y);                     //Aufruf der Meth. demo2
System.out.println (y[0]);     //ergibt y[0]=1, x[0]=4
```

Gliederung

Überladen von Methoden

In Java können wie in C++ mehrere Methoden mit gleichem Namen in einer Klasse existieren.

Beispiel

```
double max (double x, double y) { ... }
int    max (int    x, int    y) { ... }
```

- Die Methoden müssen unterschiedliche Parameterlisten besitzen.
- Der Ergebnistyp ist frei wählbar. Allerdings ist es nicht erlaubt, dass sich die Methoden nur in ihrem Ergebnistyp unterscheiden.
  $\rightarrow$ *Signatur*: Name und Parameterliste sind entscheidend.

Bemerkungen zu Methoden

- Im Gegensatz zu C++ gibt es keine separate Deklaration der Methoden.
- Lokale Variablen dürfen nicht den gleichen Namen wie ein formaler Parameter besitzen.
- Die aktuelle und formale Parameterliste müssen nicht die gleichen Bezeichner enthalten.

Gliederung

Hauptprogrammparameter

Das Hauptprogramm bei Java (Applets sind hiervon ausgenommen) ist eine Methode der Form

```
public static void main (String[] args) {
        //args ist ein Feld von Zeichenketten.
```

- Argumente werden an der Kommandozeile mit Leerzeichen getrennt.
- Die Umleitung der Standardausgabe vom Bildschirm in eine Datei erfolgt durch

  > *Dateiname*

  entsprechend erfolgt die Eingabe über

  < *Dateiname*

Achtung

Diese Angaben zählen nicht zur Argumentliste!

Hauptprogrammparameter

- Im Gegensatz zu C++ kommt der Programmname ebenfalls nicht in der Argumentliste vor.

Beispiel

```
java MeinPrg Hallo Welt  // args[0]=Hallo
                         // args[1]=Welt
java MeinPrg < Daten.dat > Ausgabe.txt Hallo Welt
                         // args[0]=Hallo
                         // args[1]=Welt
```

Gliederung

Rekursion

Es werden bei Methoden zwei Rekursionsarten unterschieden:

(1) Bei der *direkten Rekursion* ruft sich die Methode selber auf,

(2) bei der *indirekten Rekursion* rufen sich zwei oder mehrere Methoden gegenseitig auf, d. h. Methode `a()` ruft Methode `b()` und umgekehrt.

- Lokale Variablen werden bei jedem rekursiven Aufruf erneut angelegt.
- Es handelt sich bei der Rekursion um ein sehr leistungsfähiges Werkzeug, allerdings muss darauf geachtet werden, dass die Rekursion nach endlich vielen Schritten endet.
- In vielen Fällen ist die Iteration mit einer Schleife effizienter.

Rekursion - Demo-Beispiele

Beispiel

Berechnung von x^n, $n \in \mathbb{N}$

$$x^n = \begin{cases} x^{n-1} \cdot x, & \text{falls } n > 0, \\ 1, & \text{sonst.} \end{cases}$$

Beispiel

Berechnung der *Fibonacci–Zahlen*

$$\begin{aligned} f_0 &= 1, \\ f_1 &= 1, \\ f_i &= f_{i-1} + f_{i-2} \quad \text{für } i \geq 2. \end{aligned}$$

Rekursion - Beispiel Fibonaccizahlen

```
public class Fibonacci {
    static int fibonacci (int i) {
        if (i<=1) return 1;              // nicht-rek. Teil
        else  return (fibonacci(i-1)+fibonacci(i-2));
                                         // rek. Teil
    }
    public static void main (String[] args) {
        for (int i=0; i<=100; ++i)
            System.out.println (fibonacci (i));
    }
}
```

Achtung

Ein großer Vorteil liegt hier in der direkten Umsetzung der mathematischen Formel! Allerdings ist dies in diesem konkreten Fall sehr ineffizient, da viele Werte mehrfach berechnet werden!