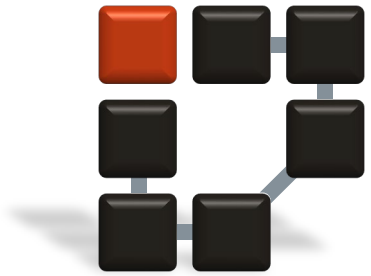
Informatik 1 für Nebenfachstudierende Grundmodul

Java – Methoden

Kai-Steffen Hielscher
Folienversion: 7. Januar 2020



Informatik 7
Rechnernetze und
Kommunikationssysteme

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Inhaltsübersicht

- Kapitel 3 - Java
  - Einführung
  - Lexikalische Struktur
  - Datentypen und Variablen
  - Operatoren und Ausdrücke
  - Anweisungen und Ablaufsteuerung
  - **Methoden**

Methoden

- Methode fasst Codeabschnitt zusammen
  - Code für eine bestimmte Aufgabe
  - Unterprogramm
  - einer Klasse zugeordnet (OOP)
  - werden von anderen Programmteilen aufgerufen
  - Werte können als Parameter übergeben werden
  - Rückkehr zum aufrufenden Programmcode nach Beendigung
  - können dabei Ergebnisse zurückliefern
  - in nicht objektorientierten Programmierparadigmen: Funktionen und Prozeduren

Deklaration von Methoden

- stets innerhalb einer Klasse
- Syntax:

```
public static Rückgabetyp Methodenname (Parameterliste)
{
    Methodenrumpf
}
```

- *Rückgabetyp*: Typ des Ergebnisses, z.B. int, double, …
  - wird kein Wert zurückgegeben: `void`
- *Methodenname*: gültiger Bezeichner in Java
- *Parameterliste*: Variablendeklarationen, mit Kommata getrennt, kann auch leer sein
- *Methodenrumpf*: Programmcode, der in der Methode ausgeführt werden soll

Deklaration von Methoden

- Beispiel:

```
public static int produkt(int a, int b) {
    int ergebnis;
    ergebnis = a * b;
    // Rückgabe des Ergebnisses fehlt noch
    …
}
```

Parameterübergabe

- bei Deklaration der Methode werden neue Variablen als Parameter deklariert, die nur in der Methode gültig sind
- Typ der Parameter wird bei Deklaration festgelegt

Ergebnisrückgabe

- Typ des Ergebnisses wird bei Deklaration der Methode festgelegt
- Anweisung `return` beendet Ausführung der Methode und erlaubt Rückgabe eines Wertes
  - Typ des Ergebnisses muss Typ in Methodendeklaration entsprechen
  - auch Literale können mit `return` zurückgeliefert werden
  - bei Methoden mit Rückgabetyp `void`:
    `return;`
    ohne Wert dahinter, kann als letzte Anweisung auch entfallen
  - kann öfters in der Methode vorkommen, z.B. Fallunterscheidung

Ergebnisrückgabe

- Beispiel:

```
public static int produkt(int a, int b) {
    int ergebnis;
    ergebnis = a * b;
    return ergebnis;
}
```

Ergebnisrückgabe

- Beispiel:

```
public static double betrag(double a) {
    if(a < 0.0) {
        return -a;
    } else {
        return a;
    }
}
```

Aufruf von Methoden

- Ausdruck:
  `Methodenname (Parameterliste)`
- Parameterliste: Liste von Ausdrücken, durch Kommata getrennt, werden in der Reihenfolge an die Variablen in der Methodendeklaration übergeben
  - **Parameter** der Methode
  - bei leerer Parameterliste in der Deklaration Aufruf mit leerer Liste
  - Typ beim Aufruf muss Typ bei Deklaration entsprechen
  - Kopie des Inhalts wird erstellt und an Methode übergeben, **call by value**
  - Methodenaufruf kann also den Wert von Variablen, die gegebenenfalls als Parameter übergeben werden, nicht ändern
- Aufruf einer Methode liefert einen Wert zurück, kann mit Zuweisungsoperator einer Variable zugewiesen werden, sofern Rückgabetyp nicht `void`

Aufruf von Methoden

- Beispiel:

```
public static int produkt(int a, int b) {
    int ergebnis;
    ergebnis = a * b;
    return ergebnis;
}
…
int x = 2;
int y = 3;
int z = produkt(x, y);
```

Aufruf von Methoden

- Beispiel:

```
public static int malzwei(int a) {
    a *= 2;
    return a;
}
…
int a = 2;
int b = malzwei(a);

// nun hat b den Wert 4, a ist weiterhin 2
// call by value
```

Überladen von Methoden

- mehrere Methoden dürfen gleichen Namen haben, sofern sie sich in Parameterliste unterscheiden
  - Zahl der Parameter
  - Typ der Parameter an der jeweiligen Position
- **keine** Methoden mit gleicher Parameterliste, jedoch unterschiedlichem Rückgabewert
- Namen der Parameter irrelevant, Typ ist wichtig
- Unterscheidung beim Aufruf anhand der Typen der übergebenen Parameter

Überladen von Methoden

- Beispiel:

```
public static int malzwei(int a) {
    return a * 2;
}

public static double malzwei(double a) {
    return a * 2.0;
}
```

Referenzdatentypen als Parameter

- wird ein Referenzdatentyp als Parameter übergeben, so wird beim Methodenaufruf lediglich eine Kopie der Referenz erstellt
  - es gibt also nach Aufruf eine weitere Referenz auf die gleichen Daten
  - Methode kann so also Werte in aufrufenden Programmteilen verändern, obwohl das eigentlich mit **call by value** nicht möglich sein sollte
  - **Seiteneffekt**
- Deklaration einer Methode mit Referenzdatentyp: **call by reference**

Referenzdatentypen als Parameter

- Beispiel:

```
public static int malzwei(int[] n) {
    n[0] *= 2;
    return n[0];
}
…
int a[] = {1};
System.out.println(malzwei(a));
// gibt 2 aus (Rückgabewert),
// a[0] ist aber hier nun auch 2
System.out.println(malzwei(a));
// gibt nun 4 als Rückgabewert aus,
// a[0] ist hier nun 4
```

Sichtbarkeit und Verdecken von Variablen

- Java-Programm ist Klasse, die `main`-Methode enthält
- kann Klassenvariablen besitzen
- Methoden der Klasse können auf Klassenvariablen zugreifen und deren Werte auch verändern
- Innerhalb von Methoden verdecken lokale Variablen und Parameter die Klassenvariablen mit gleichem Namen, diese Klassenvariablen sind nicht sichtbar, Zugriff über die Bezeichner nicht möglich

Sichtbarkeit und Verdecken von Variablen

- Beispiel:

```
public static class Sicht {
    static int a = 1, b = 2, c = 3;
    public static int malzwei(int a) {
        a *= 2;
        return a;
    }
    public static int zweib() {
        b *= 2;
        return b;
    }
    public static int amalb() {
        int c = a * b;
        return c;
    }
    public static void main(String[] args) {
        System.out.println(malzwei(a)); // 2
        System.out.println(malzwei(a)); // 2
        System.out.println(zweib());    // 4
        System.out.println(zweib());    // 8
        a = 10;
        System.out.println(amalb());    // 80
        System.out.println(c);          // 3

    }
}
```

Rekursiver Aufruf

- innerhalb eines Methodenrumpfes können weitere Methoden aufgerufen werden
- es ist sogar möglich, dass sich Methoden selbst aufrufen
- **rekursive** Methode
- wenn dies immer geschieht: Endlosschleife
- daher: meist an **Bedingungen** geknüpft
- Methode sollte **terminieren**

Rekursiver Aufruf

- Beispiel:

```
public static int fakultaet(int n) {
    if(n == 0) {
        return 1;
    } else {
        return n * fakultaet(n-1);
    }
}
```

Rekursiver Aufruf

- rekursive Methodenaufrufe
  - können leicht zu Endlosschleifen führen (Fehler)
  - sind oft langsamer in der Ausführung als normale Schleifen
  - erlauben aber manchmal eine kompakte und elegante Lösung eines Problems

Methoden

- Methoden erlauben die Wiederverwendbarkeit von Code
- wesentliche Strukturierungsmittel für wiederkehrende Aufgaben
- Modularisierung
- Bausteine zur Realisierung komplexerer Programme
- Code wird übersichtlicher
- kein Copy&Paste nötig
- der Code kann an einer Stelle verändert bzw. korrigiert werden (kein Search&Replace nötig)
- objektorientierte Programmierung: Aufruf von Methoden anderer Objekte (Klassen)