Kapitel 08

Methoden und deren Aufrufe

Fachgebiet Eingebettete Systeme und ihre Anwendungen
Prof. Dr. Andreas Koch

TECHNISCHE UNIVERSITÄT DARMSTADT

Inhalt von Kapitel 08
Methoden und deren Aufrufe

8.1 Pakete und Klassenpfade

Was sind Pakete und wie kann man auf diese zugreifen?

8.2 Objekt- und Klassenmethoden

Was sind grundlegende Unterschiede und wie wird mit ihnen gearbeitet?

8.3 Überladung

Was bedeutet Überladung und wo kann sie überall stattfinden?

- Methodenüberladung
- Konstruktorüberladung

8.1 Pakete und Klassenpfade

Was sind Pakete und was nutzen sie bei der Programmierung?

Java stellt mehrere tausend Klassenbibliotheken mit Standard- und Grundfunktionen zur Verfügung.

- Zur Strukturierung dieser Bibliotheken, wurden sie zu zusammengehörigen Gruppen, den Paketen, gebündelt.
- Ein Paket kann geschachtelt werden und wieder aus Paketen bestehen.
- Aus der Übung ist z. B. das Paket `java.util` bekannt, das mit der Klasse `java.util.Random` einen Zufallsgenerator zur Verfügung stellt.

Pakete und Klassenpfade

Klassenpfade

Die Menge aller Klassen ist grundsätzlich hierarchisch organisiert.

- Beispiel: Die Klasse `java.awt.Window`
  - `java`: Die Klasse gehört zum Standardumfang von Java.
  - `awt`: Abstract Windowing Toolkit, d. h. die Klasse gehört zum Baukasten für Fenster einer grafischen Oberfläche.
  - `Window`: Objekte der Klasse `Window` können z. B. Fenster zur Anzeige oder Flächen für Zeichnungen sein.
- Die Kurzbeschreibung vieler weiterer Pakete findet man in Kapitel 8.1.1 der Java-Referenz „Java ist auch eine Insel“

Pakete und Klassenpfade

Zugriff auf Klassenbibliotheken

Die Klassenbibliotheken stehen nicht automatisch zur Verfügung.

- Zugriff erfolgt entweder über den vollständigen Klassenpfad, z. B.
  ```
  java.util.Random rd = new java.util.Random();
  ```
- oder direkt, wenn die Klassenbibliothek am Anfang der zugreifenden Klasse importiert wurde, z. B.
  ```
  import java.util.Random;
  ...
  Random rd = new Random();
  ```
- Ausnahme: Die Klassen im Paket `java.lang` werden so häufig verwendet, dass sie automatisch von jeder Klasse importiert werden. Z. B. `java.lang.String;`

8.2 Objekt- und Klassenmethoden

Was sind Objekt- und was sind Klassenmethoden?

Analogie zu Objekt- und Klassenvariablen

- Klassenmethoden erkennt man am Schlüsselwort `static`.
- Fast keine der selbst erstellten Methoden hatte bisher ein `static`.
  - Bisher wurden fast ausschließlich Objektmethoden erstellt.
- Klassenmethoden können aufgerufen werden, ohne zuerst ein Objekt der Klasse zu erzeugen.
- Beispiel von KarelJ

```
1 // lege 3 Beeper auf die Kreuzung
2 // von 5. Straße und 5. Avenue
3 World.placeBeepers(5, 5, 3);
```

Objekt- und Klassenmethoden

Was sind Objekt- und was sind Klassenmethoden?

```
01 public class Mensch {
02   String objektName;
03   public Mensch(String objektName) {
04     this.objektName = objektName;
05   }
06
07   public void objektMethode() {
08     System.out.println("Ich heiße "+objektName);
09   }
10   public static void klassenMethode() {
11     System.out.println("Ich habe ein Herz.");
12   }
13 }
```

Objekt- und Klassenmethoden

Klassenmethoden vs. Methoden einer Klasse

Klassenmethode und Methode einer Klasse bedeutet nicht dasselbe.

- Methoden können nur in Klassen definiert werden.
- Also gehört jede Methode zu irgendeiner Klasse.
  - Klassenmethoden sind also Methoden einer Klasse.
  - Aber auch Objektmethoden sind Methoden einer Klasse.
- Methoden einer Klasse bezeichnet also immer sowohl die Klassenmethoden, als auch die Objektmethoden einer Klasse.

Objekt- und Klassenmethoden

Was sind Klassenmethoden?

In anderen Programmiersprachen gibt es den Begriff Unterprogramm.

- Diese können von überall aufgerufen und ausgeführt werden.
- Stellen i. d. R. Standard- und Grundfunktionen zur Verfügung.

In Java gibt es keine Unterprogramme sondern Klassenmethoden.

- Beispiele für solche Standard- und Grundfunktionen:

```
1     // mathematische Grundfunktionen zur Berechnung der Wurzel
2     double x = java.lang.Math.sqrt(16);
3     // es genügt auch Math.asin zur Berechnung des Arkussinus
4     x = Math.asin(0.73);
5     // die Standardausgabe auf der Konsole
6     System.out.println("Geht immer.");
```

Objekt- und Klassenmethoden

Aufruf von Klassenmethoden mit Variablen

In den Zeilen 4 bis 6 entsteht immer das gleiche Resultat.

- Zeile 4: Zugriff auf die Methode über den Klassenpfad.
- Zeile 5: Zugriff auf die Methode über die Klasse. Pfad ist bekannt.
- Zeile 6: Zugriff auf die Methode über die Variable `c` vom Typ `Character`. Auch wenn eine Klassenmethode vorliegt, ist diese wie eine Objektmethode verwendbar. Die Umkehrung funktioniert aber nicht.

```
1     char c1 = 'a';
2     Character c = new Character('b');
3     // Umwandlung von 'a' nach 'A'
4     char c2 = java.lang.Character.toUpperCase(c1);
5     char c3 = Character.toUpperCase(c1);
6     char c4 = c.toUpperCase(c1);
```

Objekt- und Klassenmethoden

Aufruf von Klassenmethoden mit Variablen

Warum Klassenmethoden mit Variablen aufrufen, wenn der Aufruf auch ohne Variable vom Typ der Klasse funktioniert?

- Wenn eine Methode mit einer Variablen aufgerufen wird, spielt es keine Rolle, ob eine Objekt- oder eine Klassenvariable vorliegt.
- Wenn im späteren Verlauf Bedarf besteht, eine Objektmethode überall verfügbar zu machen, kann diese, soweit sie weder Objektvariablen noch Objektmethoden verwendet, durch Hinzufügen von `static` geändert werden.
  - Siehe dazu die Einschränkungen auf der folgenden Folie
- Die bisherigen Methodenaufrufe werden davon nicht beeinflusst und können ohne Änderung weiter verwendet werden.

Objekt- und Klassenmethoden

Einschränkungen für Klassenmethoden

Weil Klassenmethoden mit einer Klasse statt mit einem Objekt verknüpft sind, gelten zwei wichtige Einschränkungen.

- Klassenmethoden dürfen nicht auf Objektvariablen zugreifen, die in der Klasse definiert sind.
  - Kann nicht funktionieren, weil Objektvariablen erst mit einem Objekt initialisiert werden. Ein Zugriff ist ohne Objekt nicht möglich.
  - Zugriff auf Klassenvariablen ist aber möglich.
- Klassenmethoden dürfen keine Objektmethoden aufrufen, nur andere Klassenmethoden.

Objekt- und Klassenmethoden

Objektmethoden

Im Gegensatz zu Klassenmethoden können Objektmethoden nur mit dem Namen einer **Variable** vom Typ der Klasse aufgerufen werden.

- Terminologie: Die Methode `append` wurde auf `str` angewandt.
  - Hinweis: `str` ist nicht das eigentliche Objekt. Es ist ein Verweis darauf.
- Was ist der Fehler?

```
1     StringBuffer str = new StringBuffer("Hello");
2     // Erlaubt
3     str.append(" World");
4     // Verboten
5     StringBuffer.append(" World");
6     java.lang.StringBuffer.append(" World");
```

Objekt- und Klassenmethoden

Objektmethoden

Wieso ist die Ausführung mit Klassenname nicht erlaubt?

- to append (englisch: anhängen)
- In Zeile 3 wird der String `World` and den String `Hello` angehängt. Resultat ist der neue String `Hello World`.
- Woran soll der String `World` angehängt werden, wenn, wie in Zeile 5 und 6, kein Verweis auf ein Objekt vorhanden ist?
- An diesem Problem ändert auch der genaue Klassenpfad nichts.

Objekt- und Klassenmethoden

Objekte als Methodenparameter

Im Grunde ist `str` nichts anderes als ein zweiter Parameter der Methode `append`, der

- nicht in der Parameterliste auftaucht,
- sondern durch einen Punkt getrennt vor den Methodennamen geschrieben wird.

Die syntaktische Konvention `str.append(" World")` soll das Programmverständnis fördern. `append( str, " World"),` wie in anderen Sprachen, könnte die gleiche Funktionalität abdecken.

- Die Methode `append` ist Bestandteil der Klasse von `str`.
- Hebt hervor, dass `str` auf ein Objekt verweist.

8.3 Überladung

Wiederholung Methodensignatur

Die Signatur einer Methode setzt sich zusammen aus

- dem vollständigen Namen,
- der Anzahl der formalen Parameter (potentiell auch ohne Parameter),
- den Typen der formalen Parameter in ihrer Reihenfolge in der Parameterliste,
- dem Rückgabe- / Ergebnistyp der Methode (bzw. `void`) und
- möglichen Schlüsselwörtern wie z. B. `public` oder `static`.

```
01   public static int count(char c, String text, double di)
```

Überladung

Was bedeutet Überladung und wo kann man sie überall finden?

Zwei oder mehr Methoden einer Klasse dürfen dieselbe Signatur haben, wenn

- sie sich entweder in der Anzahl der Parameter unterscheiden oder
- bei gleicher Anzahl sich die Liste der Typen der Parameter unterscheidet.
- Eine derart duplizierte Methode heißt überladen.

Überladung

Überladung. Eine Klasse kann mehr als einen Konstruktor oder mehr als eine Methode mit dem gleichen Namen enthalten, solange jede von ihnen einen unterscheidbaren Satz von Parametertypen definiert.

- Unterschiedliche Anzahl von Parametern und/oder
- unterschiedliche Parametertypen.

Überladung

```
01 public class Printer{
02     public static void print(){
03         System.out.println("Keine Eingabe!");
04     }
05     public static void print(int i){
06         System.out.println("Integer mit Wert: "+i);
07     }
08     public static void print(double i){
09         System.out.println("Double mit Wert: "+i);
10     }
11     public static void print(String i){
12         System.out.println("String mit Inhalt: "+i);
13     }
14     public static void print(char i){
15         System.out.println("Char mit Wert: "+i);
16     }
17 }
```

Überladung

Methodenaufruf innerhalb der Klasse Printer:

```
1       // Methodenaufruf innerhalb der Klasse Printer
2       public static void main(String[] args){
3           print();
4           print(1);
5           print(1.2);
6           print("test");
7           print('D');
8       }
```

Ausgabe auf der Konsole nach obigem Aufruf:

```
Keine Eingabe!
Integer mit Wert: 1
Double mit Wert: 1.2
String mit Inhalt: test
Char mit Wert: D
```

Überladung

Keine zulässigen Methodendefinitionen:

```
01 public class Printer{
02     public void print(int i){
03         System.out.println("Integer mit Wert: "+i);
04     }
05     private static void print(int i){
06         System.out.println("Double mit Wert: "+i);
07     }
08     public static int print(int i){
09         System.out.println("String mit Inhalt: "+i);
10         return i;
11     }
12     public static void print(int i){
13         System.out.println("Char mit Wert: "+i);
14     }
15 }
```

Überladung

Gerade die Standard- und Grundfunktionen existieren in verschiedenen Varianten.

Die Methode `java.lang.StringBuffer.append` existiert mehr als zehn mal. Beispiele:

- Mit einem `String`-Parameter wird die Zeichenkette im Parameter hinten an das Objekt angefügt, auf das `append` angewendet wurde.
- Mit einem `char`-Parameter wird ein Zeichen hinten angefügt.
- Mit einem `double`-Parameter wird der numerische Wert dieses `double`-Parameters als Zeichenkette hinten angefügt.

Überladung

Warum müssen sich die Methoden in den Parametern unterscheiden?

Es reicht nicht aus, wenn sie sich in anderen Bestandteilen der Signatur unterscheiden.

- Der Compiler kann sonst nicht mehr für jeden Aufruf zweifelsfrei entscheiden, welche Methode nun eigentlich gemeint ist.
- Einen Unterschied in der Modifier-Liste (`public`, `private`, etc.) könnte man einem Aufruf einer Methode nicht ansehen.
- Speziell den Rückgabetyp kann der Compiler nicht erkennen, wenn der Rückgabewert einer Methode beim Aufruf unter den Tisch fällt.

Überladung

Beispiel

Welche der beiden Varianten der Methode `MeineKlasse.f` ist denn nun mit `meinObjekt.f(1)` gemeint?

- Durch das Verbot, Methoden allein durch Variation des Rückgabetyps zu überladen, ergibt die Deklaration der Methode `f` eine Fehlermeldung vom Compiler.

```
1 public class MeineKlasse {
2   public int f (int n) { ... }
3   // verboten!
4   public char f (int n) { ... }
5 }
6 ...
7 MeineKlasse meinObjekt = new MeineKlasse();
8 meinObjekt.f(1);
```

Überladung

Anmerkung zur Klarstellung

Überladene Methoden einer Klasse dürfen sich **auch** in Rückgabetyp und Modifier unterscheiden.

- Aber dies darf nicht der **einzige** Unterschied sein.
- Es muss ein Unterscheid in der Parameterliste existieren.

Selbstverständlich dürfen Methoden aus verschiedenen Klassen identische Namen und zugleich identische Parameterlisten haben.

- Methoden aus verschiedenen Klassen sind eindeutig ansprechbar.

Überladung

Signatur und Interpreter

Der Java-Interpreter `java`

- Bekommt den Namen einer Java-Klasse als Argument beim Aufruf mit und
- erwatet als Einstiegspunkt immer eine Methode in dieser Klasse mit ganz bestimmter Signatur.

Konkret: Soll ein Java-Programm über eine Shell aufgerufen werden statt über BlueJ, erwartet der Interpreter die Methode `main` in genau der Form, wie unten.

```
1  public static void main(String[] args)
```

Überladung

Signatur und Interpreter

Die Methode `main` wird vom Interpreter als Programmstart aufgerufen.

- Wenn der Interpreter am Ende der Methode angelangt ist, ist das Programm beendet.
- Wenn die erwartete Methode nicht mit genau der erwarteten Signatur vorhanden ist, bricht der Interpreter mit einer Fehlermeldung ab.

Der Parameter `String[] args` ermöglicht z. B. die Übergabe von Dateinamen, die verwendet werden sollen, an das Programm.

- Der Quelltext zum Programm ist auf der nächsten Folie:

```
java Echo Hallo du da
```

Überladung

Signatur und Interpreter

```
01 public class Echo {
02   /**
03    * args ist ein Array von Strings. Main gibt alle Strings
04    * @param args
05    */
06   public static void main(String[] args) {
07     System.out.println();
08     // Jeder übergebene String wird in einer Zeile ausgegeben
09     for (int i = 0; i < args.length; i++) {
10       System.out.println(args[i]);
11     }
12     System.out.println();
13   }
14 }
```

Überladung von Konstruktoren

Was sind und machen Konstruktoren?

Konstruktoren wurden bereits einführend behandelt.

- Sie geben an, wie ein Objekt einer Klasse initialisiert werden soll.
- Sie werden syntaktisch wie Objektmethoden definiert. Aber
  - sie haben **keinen** Rückgabetyp (z. B. `void`) wie Methoden.
  - sie müssen genau so heißen, wie die Klasse.
- Eine Klasse kann beliebig viele Konstruktoren besitzen.
  - Ist kein Konstruktor definiert, wird der leere Konstruktor verwendet.
  - Sind mehrere Konstruktoren definiert, kommt es zu **Überladung**.

Überladung von Konstruktoren

Was sind und machen Konstruktoren?

Für zusammengesetzte Objekte werden Konstruktoren verwendet, um das Objekt zu initialisieren.

- D. h. es werden Werte berechnet, mit denen alle Objektvariablen initialisiert werden.
- Im einfachsten Fall wird jeder Objektvariable ein Parameter des Konstruktors zugewiesen.
- Konstruktoren können auch wieder Objekte erzeugen.

Überladung

Beispiele für Konstruktoren

```
01   // String und StringBuffer mit String als Parameter
02   String str1 = new String("Hallo");
03   StringBuffer str2 = new StringBuffer("Hallo");
04   // String und StringBuffer mit leerer Parameterliste
05   String str1 = new String();
06   StringBuffer str2 = new StringBuffer();
07   // Farben
08   Color red = new Color(255, 0, 0); // RBG Zahlen von 0...255
09   Color red = new Color(1.0, 0.0, 0.0); // RBG Zahlen von 0.0.
10   Color red = new Color(16711680); // RBG #FF0000 als .dec
11   Color red = new Color(Integer.parseInt("FF0000",16));
```

Überladung von Konstruktoren

Erzwungener Aufruf von Konstruktoren

Eine Klasse kann einen oder mehrere Konstruktoren haben.

- Ist mindestens ein Konstruktor vorhanden, muss einer davon bei der Objekterzeugung mit `new` aufgerufen werden.
  - Ansonsten gibt es eine Fehlermeldung.
- Es kann also erzwungen werden, ein Objekt bewusst zu initialisieren.

```
1 // führt zu einem Fehler
2 Robot karel = new Robot();
3
4 // erzeugt einen gültigen Roboter
5 Robot karel = new Robot( 1, 1, 10, KarelJConstants.North);
```

Überladung von Konstruktoren

Verschachtelte Konstruktoren

Ein Konstruktor einer Klasse kann auch in einem anderen Konstruktor aufgerufen werden.

- Syntax: Schlüsselwort `this` steht vor der Parameterliste
- Ein Aufruf dieser Art muss im zweiten Konstruktor an erster Stelle stehen und bezieht sich auf den ersten Konstruktor der Klasse.

Ein Konstruktor kann nur auf drei Arten aufgerufen werden.

- Objektinitialisierung mit `new`
- Aufruf aus einem zweiten Konstruktor der Klasse mit `this`
- Bei abgeleiteten Klassen wird auf den Konstruktor der Ursprungsklasse mit `super` verwiesen

Überladung von Konstruktoren

```
01 public class Tester {
02   private int i;
03   private double d;
04   private char c;
05   public Tester(int i, double d, char c) {
06     this.i = i; // initialisiert alle Objektvariablen
07     this.d = d;
08     this.c = c;
09   }
10   public Tester(int i) {
11     this (i, 3.14, 'a'); // d und c erhalten Standardwerte
12   }
13   public Tester(int i, char c) {
14     this (i, 3.14, c); // d erhält einen Standardwert
15   }
16 }
```