Einfache Rechenstrukturen und Kontrollfluss I

Martin Wirsing

in Zusammenarbeit mit
Moritz Hammer und Axel Rauschmayer

http://www.pst.informatik.uni-muenchen.de/lehre/SS06/infoII/

SS 06

Ziele

- Lernen Kommentare zu Java-Dokumenten zu schreiben
- Verstehen der Grunddatentypen von Java
- Verstehen von Typkonversion in Java
- Lernen lokale Variablen und Konstanten zu initialisieren
- Verstehen der Speicherorganisation von lokalen Variablen
- Lernen imperative Programme in Java mit

  Zuweisung, sequ. Komposition, Block

  zu schreiben

Kommentare in Java

„The view that documentation is something that is added to a program after it has been commissioned seems to be

wrong in principle, and counterproductive in practice.

Instead, documentation must be regarded as an integral part of the process of design and coding. „

C. A. R. Hoare (Turing-Preisträger):

Hints on Programming Language Design, 1973

C.A.R Hoare , *1934
Erfinder von Quicksort, Hoare Logik,
Strukt. Programmierung, CSP, Occam
Turing-Preis 1980

Darstellungen für Kommentare in Java

- Durch

```
// bla, bla}
```

  wird eine Zeile oder ein Rest einer Zeile zum Kommentar.

- Zur Erzeugung von Kommentaren zu Klassen und Methoden werden die Klammern

  `/**` und `*/`

  verwendet.

  Solche Kommentare werden in den mit dem Befehl **`javadoc`** erzeugten Report mit aufgenommen.

Die Klasse Hallo dokumentiert

```
 /**
Diese Klasse dient nur zum Anzeigen des Strings "Hallo, Welt!„
auf den Bildschirm
 */
public class HalloDoc
{       /**    Die Methode main druckt "Hallo, Welt!„
         */
        public static void main (String[] args)
        {       System.out.println("Hallo, Welt!");
        }
}
```


Erzeugung der Dokumentation

- Mit dem Befehl

  javadoc `HalloDoc.java`

wird automatisch eine Beschreibung der Klasse `HalloDoc` erzeugt und in die Datei

`HalloDoc.html`

geschrieben.

Spezielle Variablen bei javadoc

- `@see` für Verweise
- `@author` für Namen des Autors
- `@version` für die Version
- `@param` für die Methodenparameter

Die Klasse Square (ausführlich dokumentiert)

```
 /** Diese Klasse dient zur Berechnung des Quadrats.
@author Martin Wirsing
@version 1.1
 */
public class Square
{       /**  Diese Methode dient nur zur Illustration der
        Parameterbehandlung durch javadoc.
        @param value ist ein formaler Parameter vom Typ int
        @return das Quadrat von value
         */
        public static int square (int value)
        { return value*value; }
}
```

Eine Testklasse für Square

```
 /**    Diese Klasse dient nur zum Test von Square
 */
public class Programm
{       /** Die Methode main druckt einen Testfall von square
         */
        public static void main (String[] args)
        {       int wert = 17;
                System.out.println("Das Quadrat von "
                        + wert + " ist " + Square.square(wert));
        }
}
```


Erzeugung der Dokumentation

- Mit dem Befehlen

  `javadoc Square.java`

  `javadoc Programm.java`

  werden automatisch Beschreibungen der Klassen `Programm und Square` erzeugt und in die Dateien

  `Square.html und Programm.html`

  geschrieben.

- Um Autoren- und Versionsinformation zu erhalten, müssen beim javadoc-Befehl die Optionen für Autor und Version angegeben werden:

  `javadoc –author –version Square.java`

Grunddatentypen in Java

- Ganze Zahlen
- Gleitpunktzahlen
- Zeichen
- Boole'sche Werte

- Keine Grunddatentypen sind
  - Felder
  - Strings

Ganze Zahlen

`byte`	1 Byte	-128	bis	127
		-2^7	bis	2^7-1
`short`	2 Byte	-32768	bis	32767
`int`	4 Byte	-2,147,483,648	bis	2,147,483,647
		-2^{31}	bis	$2^{31}-1$
`long`	8 Byte			

-9,223,372,036,854,775,808 bis 9,223,372,036,854,775,807

Gleitpunktzahlen

- `float` 4 Byte bis ca. 10^{38}
- `double` 8 Byte bis ca. 10^{308}

nach IEEE-754--Standard (1985)

Beispiele:

- `double:` `6.22, 622E-2 , 62.2e-1`
- `float:` `6.22F, 622E-2F, 62.2e-1f`

Arithmet. Operationen und Vergleichsoperationen

`*` Multiplikation, `/` Division, `%` Modulo (Rest)

`+` Addition, `-` Subtraktion

`>` größer, `>=` größer oder gleich

`<` kleiner, `<=` kleiner oder gleich

`==` gleich, `!=` nicht gleich

(`=` Zuweisung wird als Gleichheit geschrieben)

Typkonversion

„Kleiner-Beziehung“ zwischen Datentypen:

`byte` < `short` < `int` < `long` < `float` < `double`

Java konvertiert Ausdrücke automatisch in den allgemeineren Typ.
Beispiele:

`1 + 1.7`	ist vom Typ **`double`**
`1.0f + 1`	ist vom Typ **`float`**
`1.0f + 1.0`	ist vom Typ **`double`**

Typkonversion

Type Casting:

Erzwingen der Typkonversion (zum spezielleren Typ `type`) durch Voranstellen von „`(`**`type`**`)`“.

Beispiele:

`(`**`byte`**`) 3`	ist vom Typ **`byte`**
`(`**`int`**`)(2.0 + 5.0)`	ist vom Typ **`int`**
`(`**`float`**`)1.3e-7`	ist vom Typ **`float`**

Bei der Typkonversion in einen spezielleren Typ kann Information verloren gehen.

Beispiele:

```
(int) 5.2 ==  5
(int)-5.2 == -5
```

Zeichen

- Typ **`char`** (für character)
- bezeichnet Menge der Zeichen aus dem Unicode-Zeichensatz
- **`char`** umfasst ASCII-Zeichensatz mit kleinen und großen Buchstaben, Zahlen und verschiedenen Sonderzeichen
- Darstellung von Zeichen durch Umrahmung mit Apostroph

  Beispiel: `'a' , 'A' , '1' , '9'`
- Zeichenketten: werden mit Doppelapostroph umrahmt und sind vom Typ `String` (eine Klasse): „Wirsing“, „Info“

Boole‘sche Werte

Der Typ **`boolean`** hat genau zwei Werte, **`true`** und **`false`**.

op strikt bedeutet, dass der Wert von `x op y` undefiniert ist, falls der Wert von x **oder** der Wert von y undefiniert ist, z.B. `(false & undef) == undef`.

George Boole 1815-1864, Entwickler des Booleschen Rings Boolesche Algebra

Boole‘sche Operatoren

!	strikte Negation	
&	strikte Konjunktion („und“, auch bitweise Addition)	
^	strikte Disjunktion („entweder-oder“)	
|	strikte Adjunktion („oder“)	

op sequentiell bedeutet, dass `x op y` von links nach rechts ausgewertet wird und die Undefiniertheit von `y` keine Rolle spielt, wenn der Wert von `x op y` schon “klar“ ist. Bsp. `(false && undef) == false`.

&&	sequentielle Konjunktion	(`andalso` in SML)
||	sequentielle Adjunktion	(`orelse` in SML)

Boole‘sche Werte

„entweder-oder“ **und** **„oder“**

^	true	false
true	false	true
false	true	false

|,||	true	false
true	true	true
false	true	false

true ^ false == true
true ^ true == false

true | false == true
true | true == true

Boole‘sche Werte

Beispiel für die strikte/sequentielle Konjunktion

```
int teiler = 0;
(teiler != 0)  &&  (100/teiler > 1) == false  // Ok
(teiler != 0)  &  (100/teiler > 1)  == false
                                    // Laufzeitfehler
```

Beispiel für die strikte/sequentielle Adjunktion

```
true || (1/0 == 1) == true;          // Ok
true |  (1/0 == 1)                   // Laufzeitfehler
```

Korrespondenz SML - Java

	Java	SML
Gleitpunktzahlen	`float, double`	`real`
unäres Minus	`-`	`~`
Division	`/`	`/`
Modulo (Rest)	`%`	nicht vorhanden
Konversion nach ganze Zahl	`(int)`	`truncate`
Ganze Zahlen	`int`	`int`
Ganzzahldiv.	`/`	`div`
Modulo (Rest)	`%`	`mod`
Boole'sche Werte	`boolean`	`bool`
strikte Konj.	`&`	nicht vorhanden
sequ. Konj.	`&&`	`andalso`
strikte Adj.	`|`	nicht vorhanden
sequ. Adj.	`||`	`orelse`
strikte Disj.	`^`	nicht vorhanden
Wörter	`String`	`string`
Konkatenation	`+`	`^`

Deklaration lokaler Variablen

Eine einfache **Deklaration lokaler Variablen** hat die Form

```
<Type>  <VarName> = <Expression>;
```

//Deklaration mit Initialisierung

Beispiel:

```
int total = -5;          //total hat den Initialwert -5
int quadrat = total * total;
boolean aussage = false;
```

Bemerkung:

Auf die Initialisierung kann verzichtet werden, wenn zur Übersetzungszeit nachgewiesen werden kann, dass die Variable initialisiert wird, bevor sie benutzt wird.

Zustand

- Ein Zustand ist eine **Belegung der Variablen mit Werten**.
- Der Zustand der lokalen Variablen wird beschrieben als **Liste von Variablennamen und zugehörigen Werten**.
- Lokale Variablen werden im **„Keller“ (engl. „Stack“)** gespeichert.

Beispiel:

Textuell: `[(total,-5), (quadrat,25), (aussage, false)]`

Im Speicher:

Lok. Variable	Adresse	Wert
total	1000	-5
quadrat	1001	25
aussage	1002	false

Iterierte Deklaration lokaler Variablen

Beispiel:

```
int total = 17, max = 100, i, j;
```

ist eine Abkürzung für

```
int total = 17;
int max = 100;
int i;
int j;
```

Deklaration lokaler Konstanten

- Eine Konstante wird durch Angabe des „Modifiers“ **`final`** deklariert.
  Beispiel: `final int TOTAL = 100;`
- Konstanten werden i.a. mit Großbuchstaben geschrieben.
- In Java 5 gibt es Enumerationstypen zur Definition von Konstanten.
- Konstanten sollten (wie auch Variablen) „sprechende“ Namen besitzen.
- Nie „Magic Numbers“ verwenden

  Beispiele:
- Anstelle von 365 im Programm für „Anzahl der Tage im Jahr“ verwende man besser `final int TAGE_PRO_JAHR = 365;`
- Für die mathematischen Größen π und *e* verwende man anstelle von 3.14159 und 2.7182 besser `Math.Pi` bzw. `Math.E`

Enum-Konstanten und der Typ `void`

- **`void`** bezeichnet den trivialen Typ, der genau ein Element enthält, d.h. **`void`** entspricht dem Typ `unit` von SML.
- Konstanten können ab Java 5 auch mit Hilfe von Enumerationstypen eingeführt werden. Dazu werden ein Typname und die Liste ALLER Elemente des Typs angegeben.
  Beispiele:

```
enum Farbe {ROT, TUERKIS, GRUEN, PURPUR, BLAU, GELB};
enum Note {AUSGEZEICHNET,SEHR_GUT, GUT, BEFRIEDIGEND,
           AUSREICHEND, UNGENUEGEND};
```

Typname

Elemente

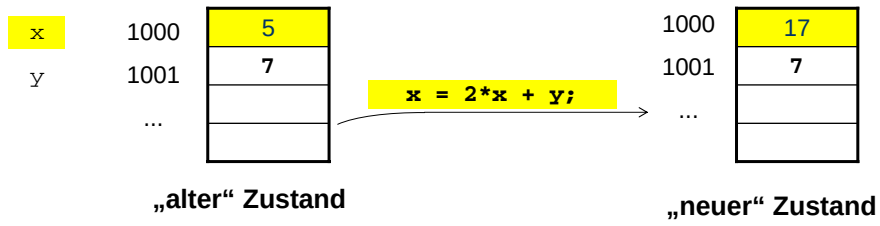
Zuweisung

Bei der **Zuweisung**

```
<VarName> = <Expression>;
```

wird der Wert `w` der `<Expression>` im „alten“ Zustand berechnet und im Nachfolgezustand der Variablen `<VarName>` als neuer Wert zugewiesen.

Beispiel:



Zuweisung: Textuelle Darstellung

Beispiel textuell:

„alter“ Zustand `s1 = [(x, 5), (y,7), (b, true)]`

Zuweisung `x = 2*x+y;`

„neuer“ Zustand `s2 = [(x, 17), (y,7), (b, true)]`

Zuweisung: Abkürzende Schreibweisen

Abkürzungen

`x++;`	steht für `x = x + 1;`
`x--;`	steht für `x = x - 1;`
`x op= <Ausdruck>;`	steht für `x = x op <Ausdruck>`

Beispiele

`x += y;`	steht für `x = x + y;`
`b &&= c;`	steht für `b = b && c;`
`x += 3*y;`	steht für `x = x + 3*y;`

Zusammenfassung

- Ein **Java-Programm** besteht aus einer oder mehreren **Klassen** (und **Schnittstellen**).
  - Klassen enthalten **Attribute** und die Deklarationen von **Methoden** und **Konstruktoren**.
  - Eine Methode besteht aus einer **Sequenz von Anweisungen**, die den Berechnungsablauf festlegen.
- Jede **selbstablaufende** Java-Anwendung enthält eine Methode „**main**“.
- Ein Java-Programm wird mit einem **Übersetzer in Byte-Code** übersetzt, der dann mit einem **Interpreter**, der **JVM** , ausgeführt wird.
- Java-Programme sollten gut dokumentiert werden. Mit **`javadoc`** kann automatisch eine übersichtliche Dokumentation erzeugt werden.

Zusammenfassung 2

- Java besitzt
  - 4 Grunddatentypen für ganze Zahlen (**byte, short, int, long**) und
  - 2 Grunddatentypen für Gleitpunktzahlen (**float, double**).
- Dazu kommen noch **boolean**, **char** und **void**.
- String ist **kein** Grunddatentyp.
- Java hat eine **automatische Konversion in den allgemeineren** Grunddatentyp.
- Konversion in einen **spezielleren Datentyp** geschieht explizit durch **Typcasting**. Dabei kann Information verloren gehen.