Kontrollstrukturen, Pseudocode und Modulo-Rechnung

CoMa-Übung III

TU Berlin

29.10.2012

Themen der Übung

1. Bedingte Anweisungen
2. Vergleiche und logische Operatoren
3. Fallunterscheidungen
4. Zeichen und Zeichenketten
5. Schleifen
6. Pseudocode
7. Modulo-Rechnung

Bedingte Anweisungen

Die `if`-Anweisung

- `if (condition) doA; else doB;` führt `doA;` aus, wenn `condition` erfüllt ist, und `doB;` wenn nicht
- `condition` ist vom Typ `boolean` (d.h. ein Wahrheitswert) und kann entweder `true` oder `false` sein

```
1  public class Temperature {
2    public static void main(String[] args) {
3      boolean condition = true;
4      if (condition)
5        System.out.println("Die Bedingung ist wahr!");
6      else
7        System.out.println("Die Bedingung ist falsch!");
8
9    }
10 }
```


Vergleiche und logische Operatoren

Vergleiche

- `a == b` liefert für primitive Datentypen genau dann `true`, wenn *a* **gleich** *b* ist, und sonst `false`
- `a != b` liefert für primitive Datentypen genau dann `true`, wenn *a* **nicht gleich** *b* ist, und sonst `false`
- `a < b`, `a <= b`, `a >= b`, `a > b` liefert für Zahlen genau dann `true`, wenn $a < b$, $a \leq b$, $a \geq b$, $a > b$ ist, und sonst `false`

Logische Operatoren

Es gibt in Java vier logische Operatoren, die mit einem oder zwei `boolean`-Werten arbeiten und einen neuen `boolean`-Wert zurückgeben.

- `!a`: **nicht** a
- `a && b`: a **und** b
- `a || b`: a **oder** b
- `a ^ b`: **entweder** a **oder** b (exklusives oder, xor)

Logische Operatoren

Übersicht

boolean a	boolean b	!a	a && b	a || b	a ^ b
true	true	false	true	true	false
true	false	false	false	true	true
false	true	true	false	true	true
false	false	true	false	false	false

Logischer Kurzschluss

- Java wertet Boolschen Ausdruck nur aus, bis der Wert feststeht
- `true || a`, `false && b` → die Werte von `a` und `b` sind egal
- Das wird als *logischer Kurzschluss* bezeichnet
- Soll auf jeden Fall der ganze Term ausgewertet werden, gibt es alternative Und/Oder-Operatoren `&` und `|`
- Gleicher Effekt, aber ohne logischen Kurzschluss

Operatorenvorrang (Höchste zuerst)

Operatoren	Beschreibung
+,-,!	Unäres Plus, unäres Minus, logisches Nicht
*,/,%	Multiplikation, Division, Rest
+,-	Addition, Subtraktion, Konkatenation von Strings
<,>,<=,>=	Numerische Vergleiche
==,!=	Gleichheit
&	Logisches Und
^	Logisches Xor
|	Logisches Oder
&&	Logisches konditionales Und
||	Logisches konditionales Oder
?:	Bedingungsoperator

- Bei gleicher Priorität wird von links nach rechts ausgewertet
- 1 Argument > 2 Argumente > 3 Argumente, Punkt- vor Strich
- Numerische Vergleiche > Gleichheit > Und > Oder

Zurück zu `if`

Mehrere Anweisungen

- Sollen bei einer `if`-Abfrage mehrere Anweisungen ausgeführt werden, müssen Blöcke benutzt werden
- `if (condition) { doA; doB; } else { doC; doD; }`
- Für komplexere Fallunterscheidungen gibt es das Konstrukt
- `if (c1) { doA; } else if (c2) { doB } else { doC; }`
- Beliebig viele `else if`'s
- Der erste passende Fall wird ausgeführt, der Rest nicht

```
1  if (number == 0) {
2    sign = 0;
3  } else if (number > 0) {
4    sign = 1;
5  } else {
6    sign = -1;
7  }
```


Beispiele

```
1  boolean a = true && false || true;
2  boolean b = !(true || false);
3  boolean c = !a ^ !b;
4  int number = 1;
5  if (c) {
6    number = number * 2;
7  } else if (c && number > 0) {
8    number = number * 3;
9  } else {
10   number = 0;
11 }
```

Was kommt raus?

- `true && false || true` → `false || true` → `true`
- `!(true || false)` → `!(true)` → `false`
- `!a ^ !b` → `!true ^ !false` → `false ^ !false` → `false ^ true` → `true`
- `number = 2`

Der Bedingungsoperator und `switch`

Der Bedingungsoperator ?:

- Zuweisung abhängig von einer Bedingung
- `maximum = (a > b)? a : b;`
- Kurzform für: `if (a > b) maximum = a; else maximum = b;`

Der `switch`-Befehl

- Für Fallunterscheidungen von Ganzzahlen, Zeichen, Zeichenketten oder Aufzählungstypen mit vielen Fällen
- `switch (month) {`
- `case 1: System.out.println("Januar"); break;`
- `case 2: System.out.println("Februar"); break;`
- `default: System.out.println("Kein anderer Fall passt.");`
- `}`
- Ohne das `break;` würden ab dem ersten passenden Fall alle Fälle ausgeführt
- Nur Konstanten als Fälle möglich

Beispiel

```java
import java.util.Scanner;
public class Switch {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    double x = scanner.nextDouble();
    char operator = scanner.next().charAt(0);
    double y = scanner.nextDouble();
    switch (operator) {
      case '+':
        System.out.println(x + y);
        break;
      case '-':
        System.out.println(x - y);
        break;
      case '*':
        System.out.println(x * y);
        break;
      case '/':
        System.out.println(x / y);
        break;
    }
  }
}
```


Zeichen

Zeichen in Java

- Zeichen werden in Java als 16-bit Unicode-Charaktere dargestellt
- Primitiver Datentyp `char` – Zahl zwischen 0 und 65535
- Klasse `Character` – Kann ein Zeichen darstellen und bietet nützliche Methoden für den Umgang mit Zeichen
- Automatische Umwandlung zwischen `char` und `Character`
- Zeichen-Konstanten werden durch einfache Anführungszeichen ausgewiesen: z.B. `'a'`

Nützliche Methoden der Klasse `Character`

- `Character.isDigit('a'); Character.isLetter('a');`
- `Character.isWhitespace('a');`
- `Character.isLowerCase('a'); Character.isUpperCase('a');`
- `Character.toLowerCase('a'); Character.toUpperCase('a');`

Zeichenketten

Zeichenketten in Java

- Die Klasse `String` stellt nicht-veränderbare Zeichenketten dar
- Zeichen-Konstanten werden durch doppelte Anführungszeichen ausgewiesen: z.B. `"Hello"`
- Der Konkatenations-Operator + hängt zwei Strings aneinander `"Hello" + "World"` und erzeugt einen neuen String `"HelloWorld"`
- Bekommt der Konkatenations-Operator einen String und etwas anderes, wird das andere in eine String-Darstellung umgewandelt

Nützliche Methoden der Klasse `String`

- `s.length()`: gibt die Anzahl Zeichen in `s` zurück
- `s.charAt(int i)`: gibt das Zeichen an Position `i` in `s` zurück. Das erste Zeichen ist bei 0, das letzte bei `s.length()-1`.
- `s.equals(String t)`: sind `s` und `t` gleich? (`==`, `!=` funktionieren hier nicht, da wir es nicht mit primitiven Datentypen zu tun haben).

Schleifen

while-Schleifen

- Wiederholen eine oder mehrere Anweisungen, solange eine Bedingung erfüllt ist. Die Bedingung ist ein `boolean`
- Ist die Bedingung beim ersten Erreichen der Schleife nicht erfüllt, werden die Anweisungen nie ausgeführt
- Bleibt die Bedingung erfüllt, werden die Anweisungen potentiell unendlich oft ausgeführt → Vorsicht!

```
1 int i = 1;
2 while (i <= 10) {
3   System.out.println(i*i);
4 }
```

```
1 int i = 1;
2 while (i <= 10) {
3   System.out.println(i*i);
4   i = i + 1;
5 }
```


Schleifen (2)

do-while-Schleifen

- Wiederholen eine oder mehrere Anweisungen, solange eine Bedingung erfüllt ist. Die Bedingung ist ein `boolean`
- Die Anweisungen in der Schleife werden mindestens einmal ausgeführt
- Bleibt die Bedingung erfüllt, werden die Anweisungen potentiell unendlich oft ausgeführt → Vorsicht!

```
1 int i = 1;
2 do {
3   System.out.println(i*i);
4 } while (i <= 10);
```

```
1 int i = 1;
2 do {
3   System.out.println(i*i);
4   i = i + 1;
5 } while (i <= 10);
```


Schleifen (3)

for-Schleifen

- Spezielle Variante von `while`-Schleifen, in den gezählt werden soll
- Bestehen aus einer Initialisierung, einer Bedingung und einer Aktualisierung des Zählers

```
1 for (Initialisierung; Bedingung; Aktualisierung) {
2 ...
3 }
```

```
1 for (int i=1; i<=n; i=i+1) {
2 ...
3 }
```

```
1 int i=1;
2 while (i<=n) {
3   ...
4   i=i+1;
5 }
```


Beispiel

```
1  import java.util.Scanner;
2  public class Calculator {
3      public static void main(String[] args) {
4          Scanner scanner = new Scanner(System.in);
5          double x = scanner.nextDouble();
6          boolean stop = false;
7          while (!stop) {
8              char operator = scanner.next().charAt(0);
9              double y = scanner.nextDouble();
10             switch (operator) {
11                 case '+': x = x + y; break;
12                 case '-': x = x - y; break;
13                 case '*': x = x * y; break;
14                 case '/': x = x / y; break;
15                 default: stop = true;
16             }
17             System.out.print(" = " + x);
18         }
19     }
20 }
```

Pseudocode

Probleme

- Fließtext-Beschreibungen von Algorithmen: meist lang & nicht präzise
- Java-Code: schwerer zu verstehen, viele technische Details
- → Pseudocode als Mittelweg

Idee von Pseudocode

- Code für Menschen statt für Computer
- Relativ präzise und exakt, aber keine technischen Details
- Keine sprach-spezifische Syntax
- Keine expliziten Typdeklarationen
- Keine Effizienz-Tricks
- → wird daher oft in der Literatur verwendet

Pseudocode – Grundform & Beispiel

Grundform

- `Algorithmenname(Parameterliste)`
- `Input:`
- `Output:`
- `Liste der Programmschritte`

Beispiel

```
maximum(a,b)
Input:  a, b ∈ ℝ
Output:  max{a, b}
IF (a > b) THEN
     RETURN a
ELSE
     RETURN b
ENDIF
```


Pseudocode – Befehle

Bedeutung	Pseudocode	Java-Code
Zuweisung	:=	=
Vergleich	=, ≠, ≤, ≥, <, >	==, !=, <=, >=, <, >
Logisches Und	AND, ∧	&&
Logisches Oder	OR, ∨	||
Logisches Nicht	NOT, ¬	!
Kommentar	//	//, /* */
Rückgabe	RETURN	return

Bedingte Anweisungen

```
IF (condition) THEN
     ...
ELSE IF (condition2) THEN
     ...
ELSE
     ...
ENDIF
```

```
1 if (condition) {
2
3 } else if (condition2) {
4
5 } else {
6
7 }
```


Pseudocode – Befehle (2)

while-Schleifen

```
WHILE (condition) DO
     ...
ENDWHILE
```

```
1 while (condition) {
2
3 }
```

```
DO
     ...
WHILE (condition)
```

```
1 do {
2
3 } while (condition);
```

for-Schleifen

```
FOR i := 1 TO n DO
     ...
ENDFOR
```

```
1 for (int i=1; i<=n; i=i+1) {
2
3 }
```

```
FORALL a ∈ A DO
     ...
ENDFOR
```

```
1 for (Datentyp a : A) {
2
3 }
```

Pseudocode – Zusammenfassung

Einfache Algorithmen

- Einfache Operationen / Algorithmen sind in Pseudocode erlaubt
- `A := {b,a,c}`
- `sortiere A`
- b := `wähle ein zufälliges` $a \in A$

Zusammenfassung

- HA fragt nach Java: kein Pseudocode
- HA fragt nach Algo: Pseudocode
- Keine Algorithmen als Fließtext!
- Kopf des Pseudocode (Name, Parameter, Input, Output) wichtig
- Kein `goto` erlaubt
- `.`-Operator erlaubt

Modulo-Rechnung

Division mit Rest

Sei $a \in \mathbb{Z}$ und $b \in \mathbb{N} \setminus \{0\}$. Dann gibt es eindeutig bestimmte Zahlen $k, r \in \mathbb{Z}$, so dass gilt

$$a = k \cdot b + r, \quad r \in \{0, 1, \ldots, b-1\}$$

r wird als der **Rest** bezeichnet, der bei der Division von a durch b entsteht.

Schreibweisen

Wir bezeichnen r auch als

$$a \mod b \qquad \text{(lies: } a \textbf{ modulo } b\text{)}$$

Außerdem schreiben wir

$$a \equiv b \mod n \qquad \text{(lies: } a \textbf{ kongruent } b \textbf{ modulo } n\text{)}$$

wenn $a \mod n = b \mod n$, d.h. wenn a und b den gleichen Rest bei Division durch n haben.

Modulo-Rechnung in Java

Division mit Rest in Java

Den Rest einer Division kann man in Java mit dem Operator % erhalten. Er ist für Zahlen a, b wie folgt definiert:

```
a % b = a - (int)(a/b) · b;
```

- Der % Operator funktioniert auch für Gleitkommazahlen
- Das `(int)` sorgt dafür, dass aus `a/b` in jedem Fall ein `int` wird (notfalls wird abgerundet)
- Bei dieser Definition kann ein negatives Ergebnis herauskommen!
- Anders als bei mathematischer Definition, Vorsicht!

Beispiele

- $42 \mod 5 = (8 \cdot 5 + 2) \mod 5 = 2 \mod 5$, d.h. $42 \equiv 2 \mod 5$
- $-47 \mod 11 = (-5 \cdot 11 + 8) \mod 11 = 8 \mod 11$, d.h. $-47 \equiv 8 \mod 11$
- $-47 \,\%\, 11 = -47 - (int)(-47/11) \cdot 11 = -47 - (-4) \cdot 11 = -47 - (-44) = -3$