Programmiermethodik 1
Programmiertechnik

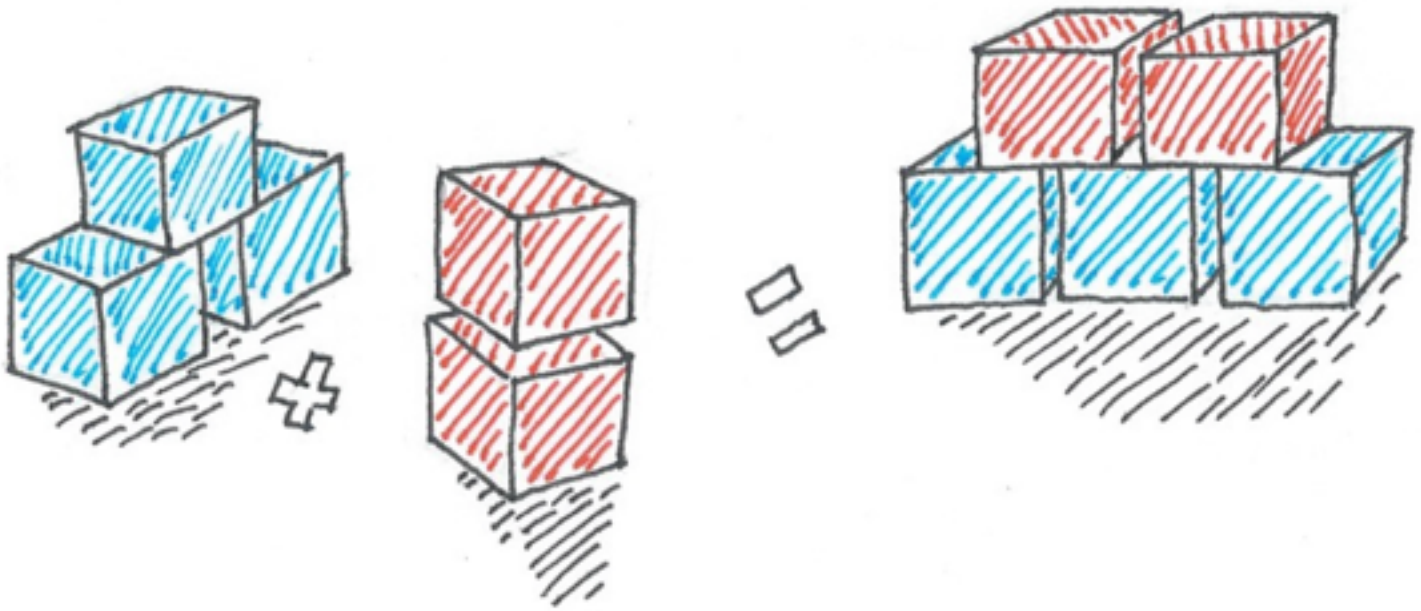
Arithmetische Ausdrücke und Zahlen

Wiederholung

- Bezeichner
- Variablen
- Ein- und Ausgabe
- Ganzzahlen und Literale

Ausblick

Worum gehts?



Agenda

- Arithmetische Ausdrücke
- Fließkommazahlen
- Kompatibilität

Arithmetische Ausdrücke

Grundrechenarten

- Schreibweise arithmetischer Ausdrücke ähnlich zur Mathematik
    - „Prinzip der geringsten Verwunderung“
- einfachste Darstellung:
    - zwei Literale (als Operanden) und Operator
    - Beispiel: 1+2
- Syntax: <Operand> <Operator> <Operand>

Grundrechenarten

- Operatoren für die Grundrechenarten:
  - Addition : +
  - Subtraktion: -
  - Multiplikation: *
  - Division: /

- Multiplikationsoperator * muss immer angegeben werden
  - anders als z.B. in der Mathematik

Arithmetische Ausdrücke

- Berechnung und Ausgabe eines arithmetischen Ausdrucks:
  - System.out.println(1 + 2); // Ausgabe: 3
- Text (Literal für Zeichenketten) wird immer unverändert ausgegeben
  - in Anführungszeichen
  - System.out.println("1 + 2"); // Ausgabe: 1 + 2
- Darstellung von ganzen Zahlen ist eindeutig:
  - System.out.println(2); // Ausgabe: 2
  - System.out.println(+2); // Ausgabe: 2
  - System.out.println(0x2); // Ausgabe: 2 (Hexadezimal)
  - System.out.println(0b10); // Ausgabe: 2 (Binär)

Arithmetische Ausdrücke

- Ganzzahlige Division
- Arithmetik (bisher) ausschließlich ganzzahlig
- ganzzahlige Division schneidet Nachkommaanteil des Ergebnisses ab
    - kein Runden!

Beispiele

- 11/4 → 2 (nicht 2.75)
- -11/4 → -2 (nicht -2.75)
- 4/2
- 3/2
- 1/2
- 2/1
- 100/50
- 100/49
- 100/51

Modulo

- fünfte „Grundrechenart“: Divisionsrest = Modulus
- Operatorzeichen: %
- Beispiele:
  - 11%4 → 3
  - 8%4 → 0
  - 7%3 → 1
  - wird in Java mit ganzzahliger Division berechnet
    - a%b = a–(a/b)*b
- Vorsicht bei negativen Zahlen
  - die mathematische Definition des Modulus a mod b ergibt immer Ergebnisse zwischen 0 und b-1
  - nicht so in Java

Übung: Modulo-Operator

- Geben Sie die Ergebnisse der Auswertung folgender Modulo-Ausdrücke an:
  - 4 % 2
  - 3 % 3
  - 17 % 5
  - 5 % 17
  - -5 % -2
  - 5 % -2
  - -5 % 2

Erinnerung
a%b = a–(a/b)*b

Zusammengesetzte Ausdrücke

- Ausdrücke können kombiniert werden
- Definition: Ein Ausdruck ist ein …
  - elementarer Ausdruck: z.B. Literal
  - zusammengesetzter Ausdruck: mehrere Ausdrücke, die durch einen Operator verknüpft sind
- Beispiele für zusammengesetzte Ausdrücke:
  - 3 + 2 * 4
  - 2 * 3 + 4 * 5
  - 100 - 10 - 20
  - + 4 * +5
  - -2 - -1
  - 11 - 11/4 * 4

Semantik

- Unterscheidung
  - Syntax von Ausdrücken liegt fest (Grammatik)
  - aber die Semantik (= Berechnung der Werte) ist zu klären
- Ergebnis ist abhängig von der Reihenfolge der Anwendung der Operatoren
- Beispiel: 2 + 3 * 4
  - Reihenfolge: Addition vor Multiplikation
    - 2+3*4 → 5*4 → 20
  - Multiplikation vor Addition
    - 2+3*4 → 2+12 → 14
  - nur ein Ergebnis kann richtig sein!

Priorität

- Auswertungsreihenfolge folgt Priorität der Operatoren
    - auch Bindungsstärke oder Operatorenvorrang genannt
- Priorität der Punkt-Operatoren (*, /, %) ist höher als die der „Strich-Operatoren“ (+, -)
- deckt sich mit bekanntem Vorgehen: „Prinzip der geringsten Verwunderung“
- Zurück zum Beispiel:
    - 2 + 3 * 4 → 2 + 12 → 14

Klammern

- runde Klammern (…) erzwingen eine bestimmte Auswertungsreihenfolge
- eingeklammerte Teilausdrücke werden immer zuerst ausgerechnet
  - (2 + 3) * 4 → 5 * 4 → 20
- Klammern sind um jeden Ausdruck erlaubt:
  - dabei spielt es keine Rolle, ob Klammern die Auswertungsreihenfolge beeinflussen oder nicht
- Beispiele
  - 2 + (3 * 4) → 2 + 12 → 14
  - (2 + 3) → 5
  - (((2))) → 2

Unäre Vorzeichenoperatoren

- unäre Vorzeichenoperatoren + und - stehen vor jeweils einem einzigen Operanden
  - auch einstellige Operatoren genannt
  - "-" tauscht das Vorzeichen
  - "+" existiert nur aus Symmetriegründen, kann weggelassen werden
- Priorität der unären Operatoren ist höher als die der binären Operatoren
  - auch zweistellige Operatoren genannt
- Beispiele
  - -(1 + 2) → -(3) → -3
  - 3*-4 → 3*(-4) → -12
  - -3+-4 → (-3) + (-4) → -7
  - -(2 + -3) → -(-1) → 1

Assoziativität („Bindungsrichtung“)

- Priorität regelt Vorrang bei unterschiedlichen Operatoren
- aber: auch bei mehreren gleichrangigen Operatoren gibt es alternative Auswertungsmöglichkeiten
- Beispiel: 8 - 3 - 2
  - linkes Minus zuerst: 8 - 3 - 2 → 5 - 2 → 3
  - rechtes Minus zuerst: 8 - 3 - 2 → 8 - 1 → 7
- Operatoren haben eine charakteristische Assoziativität (Bindungsrichtung)
  - rechts- oder links-assoziativ
- alle binären arithmetischen Operatoren sind links-assoziativ
  - „Der am weitesten links stehende Operator wird zuerst ausgewertet“
- Demnach:
  - 8 - 3 - 2 → 5 - 2 → 3

Operatorentabelle

- Auszug aus der Tabelle mit der Operator-Auswertungsreihenfolge
  - vollständige Tabelle in EMIL

2	++	pre- or postfix increment	right
	--	pre- or postfix decrement	
	+ -	unary plus, minus	
	~	bitwise NOT	
	!	boolean (logical) NOT	
	(type)	type cast	
	new	object creation	
3	* / %	multiplication, division, remainder	left
4	+ -	addition, substraction	left
	+	string concatenation	

Spezialfall des Operators +

- Operator "+" zur Verknüpfung von Text-Zeichenketten miteinander
    - auch Konkatenation genannt

- Beispiel:
    - System.out.println("Hallo," + " Welt!");
    - Ausgabe: Hallo, Welt!

Zuweisungsoperatoren

- = x = y
- *= x *= y (x = x * y)
- /= x /= y (x = x / y)
- %= x %= y (x = x % y)
- += x += y (x = x + y)
- -= x -= y (x = x - y)

- &= x &= y (x = x & y) // Binärzahlen
- |= x |= y (x = x | y)
- ^= x ^= y (x = x ^ y)
- <<= x <<= y (x = x << y)
- >>= x >>= y (x = x >> y)
- >>>= x >>>= y (x = x >>> y)

op= als Kurzform für

x op= y ⇔ x = x op y

Unveränderliche Variablen

- Variablenwerte werden manchmal einmal zugewiesen und sollen sich dann nicht mehr ändern
    - ähnlich Konstante
- optionaler Zusatz bei der Deklaration (Modifizierer oder Modifier): final
- final erlaubt nur eine einzige Wertzuweisung für eine Variable
    - Syntax: final <Typ> <Variablenname> [= <Ausdruck>];
- final-Deklarationen helfen bei der Entwicklung von Programmen
    - Compiler kann mehr Fehler

Unveränderliche Variablen

- Beispiele:
  - final int maxAnzahlStudenten = 40;
  - final int lichtgeschwindigkeit;
  - lichtgeschwindigkeit = 299793218;
  - maxAnzahlStudenten = 0; // Error: 2. Zuweisung

Vorgriff: Modifier

- bisher gesehen:
  - public (siehe public static void main ...)
  - final
- weitere:
  - protected
  - private
  - abstract
  - static
  - native
  - transient
  - volatile
  - synchronized
  - strictfp

Übung: SummenRechner

- Erstellen Sie ein Programm SummenRechner, das 2 übergebene Integer-Werte addiert und das Ergebnis ausgibt!

- Anforderungsanalyse
- Eingabe
    - der Benutzer gibt 2 ganzzahlige Werte ein
- Ausgabe
    - die Summe der beiden Werte wird berechnet und ausgegeben

Fließkommazahlen

Datentypen: Fließkommazahlen

- auch genannt: Gleitkommazahlen
- oft benötigt:
  - rationale Zahlen (z.B. 3/4)
  - irrationale Zahlen (z.B. $\pi = 3.141592\ldots$)
  - sehr große oder sehr kleine Werte (zum Beispiel 1023, 10–34)
  - also: mit ganzen Zahlen umständlich oder überhaupt nicht auszudrücken
- Lösung: zweiter numerischer Datentyp: Fließkommazahlen
- Typbezeichnung in Java mit reserviertem Wort double
  - Verwendung in Variablendeklarationen wie int
  - Fließkommaliterale sind standardmäßig vom Typ double
  - Typ float identisch, nur ungenauer (weniger Stellen)

Literale

- Fließkommaliterale müssen mindestens eines der folgenden Merkmale aufweisen:
  - Nachkommaanteil, mit einem Dezimalpunkt abgetrennt
    - Beispiele: 3.14, 0.001, -123.04, 21200.0
  - Zehnerexponent, mit E oder e markiert
    - E kann gelesen werden als „mal-zehn-hoch“
    - Beispiele: 1E23 (1·1023), 1e-34 (1·10–34), 6.670E-11 (6.670·10–11), -4.17e-4 (–4.14·10–4)
    - Zehnerexponenten sind immer ganzzahlig, mit optionalem Vorzeichen
  - Fließkomma-Suffix (nachgestelltes Zeichen) D oder d (für double)
    - Beispiele: 1D, -234d, 0.001D, 1e-34d

Datentypen: Fließkommazahlen

- mehrere Schreibweisen des gleichen Wertes möglich
    - 20.5 = 0.0205E3 = 205000E-4
- Beispiele für Typen von numerischen Literalen:
    - 20 (int)
    - 20.0 (double)
    - 20E0 (double)
    - 20.E0 (double)
    - 20d (double)
    - 20D (double)
- rechnerisch gleiche double- und int-Literale sind im Quellcode nicht beliebig austauschbar!

double-Variablen

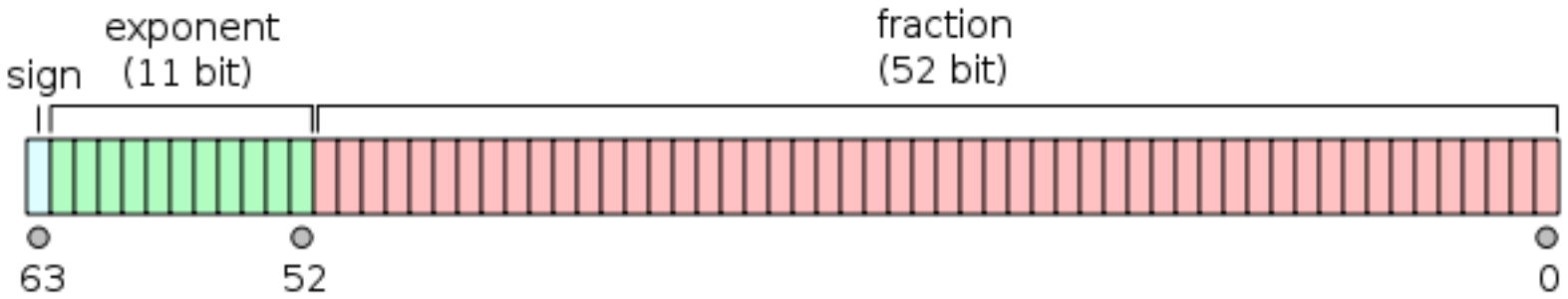
- Beispiel:
  - double entfernung = 234.45;
- Hinweis
  - Typ kann sich nach Deklaration nicht ändern, der Wert sehr wohl

Ausflug: Polymorphie

- numerische Operatoren arbeiten mit int- und double-Operanden
- Typ des Ergebnisses ist abhängig vom Typ der Operanden
- Beispiele
    - 20 / 8 // 2
    - 20.0 / 8.0 // 2.5
- gleicher Operator (hier: Divisionsoperator /) löst intern unterschiedliche Mechanismen aus
    - Polymorphie
- allgemeines Phänomen, taucht an vielen Stellen in vielen Programmiersprachen auf!
    - siehe auch Text-Konkatenation mit +

Datentypen: Fließkommazahlen

- double
- Genauigkeit: ca. 16 Dezimalstellen!



Grenze	Wert	Vordefinierte Variable
größter positiver Wert	1.79769.. · 10308	Double.MAX_VALUE
kleinster positiver Wert	4.94065.. · 10–324	Double.MIN_VALUE
kleinster negativer Wert	–4.94065.. · 10–324	-Double.MIN_VALUE
größter negativer Wert	–1.79769.. · 10308	-Double.MAX_VALUE

Datentypen: Ganzzahlen und Fließkommazahlen

- Fließkomma-Arithmetik rechnerisch viel genauer, wozu noch ganzzahlige Arithmetik?
- int-Arithmetik hat Vorteile:
    - double-Arithmetik ist langsamer als int-Arithmetik
    - double-Werte brauchen doppelt so viel Platz wie int-Werte
    - double-Arithmetik macht (im Gegensatz zu int) manchmal Rundungsfehler
        - (1.0/x)*x
        - liefert für manche x (z.B. 49.0) nicht 1.0 (sondern 0.999999999999999)
- int wenn möglich, double wenn nötig!

Übung: Fließkommazahlen

- Geben Sie Ausdrücke an, in denen die beschriebenen Berechnungen durchgeführt werden und das Ergebnis jeweils in einer Variable ergebnis abgelegt wird.
    - drei geteilt durch vier
    - Umfang eines Kreises: zwei mal Pi mal Radius
    - ein Fünftel plus zwei

Kompatibilität

Typkonvertierung

- int → double
- zwei Operanden gleichen Typs
  - Ergebnistyp = Operandentyp
  - 1 + 2 → 3 (int)
- gemischte Operandentypen int/double:
  - Ergebnistyp ist immer double
  - 1.0 + 2 → 3.0 (double)
  - 1 + 2.0 → 3.0 (double)
  - 1.0 + 2.0 → 3.0 (double)
  - erst Umwandlung des int-Operanden in double, dann weiter mit zwei Operanden gleichen Typs
    - 1.0 + 2 → 1.0 + 2.0 → 3.0

Implizite Typkonversion

- automatische (stillschweigende) Umwandlung eines Typs in einen anderen
- Konversion int → double findet immer dann statt, wenn int verfügbar ist, aber double gebraucht wird

Implizite Typkonversion

- int → double
- zu jedem int-Wert gibt es einen äquivalenten double-Wert
    - Beispiel: 2 → 2.0
- aber: zu vielen double-Werten gibt es keinen äquivalenten int-Wert
    - zum Beispiel 1E100
- deshalb: keine implizite Typkonversion von double → int
- Beispiele:
    - zulässig wegen impliziter Typkonversion int → double:
        - double d = 2; // implizite Typkonversion 2 → 2.0
    - Fehler mangels impliziter Typkonversion:
        - int i = 2.0; // Fehler!

Datentypen: Kompatibilität

- allgemein: ein Typ T ist kompatibel zu einem anderen Typ U, wenn ein Wert vom Typ T einer Variablen vom Typ U zugewiesen werden kann
- Beispielcode schematisch:
    - T varTypeT = …;
    - U varTypeU;
    - varTypeU = varTypeT ; // ok falls T kompatibel zu U
- int kompatibel zu double
    - implizite Typkonversion
- aber: double nicht kompatibel zu int
    - Kompatibilitätsbeziehung nicht symmetrisch

Datentypen: Explizite Typkonversionen

- explizite Typkonversion: Erzwingen einer Typkonversion
  - z.B. double → int
  - engl. type cast
- Syntax: (<Zieltyp>) <Ausdruck>
- Ergebnistyp des Ausdrucks wird in den Zieltyp umgewandelt
  - Typkonversion ist synaktisch ein unärer (einstelliger) rechts-assoziativer Operator
  - Typkonversion hat hohe Priorität, wie andere unäre Operatoren

Beispiele

- (int) 2.5 * 3 → 2 * 3 → 6
- (int) -2.5 → -2
- -(int) 2.5 → -2

- ggf. Klammern setzen für andere Auswertungsreihenfolge
    - (int)(2.5 * 3) → (int)(7.5) → 7

Übersicht: Zahlentypen

Typ	Werte	Länge (Bit)	größter positiver Wert	größter negativer Wert
byte	ganze Zahlen	8	27-1 (127)	-27 (-128)
short	ganze Zahlen	16	215-1 (32767)	-215 (-32768)
int	ganze Zahlen	32	231-1 (ca. 2 · 109)	-231 (ca. -2 · 109)
long	ganze Zahlen	64	263-1 (ca. 9 · 1018)	-263 (ca. -9 · 1018)
float	Fließkomma-zahlen	32	3.40282347 · 1038	-3.40282347 · 1038
double	Fließkomma-zahlen	64	1.79769.. · 10308	-1.79769.. · 10308

Kompatibilität und Konvertierung

- es wird in folgenden Fällen eine implizite Typkonvertierung durchgeführt:
  - bei der Auswertung eines Ausdrucks, wenn Operanden unterschiedlichen Typ besitzen
  - bei einer Wertzuweisung, wenn der Typ der Variablen und des zugewiesenen Wertes nicht identisch sind

Zusammenfassung

- Arithmetische Ausdrücke
- Fließkommazahlen
- Kompatibilität