Variablen, Datentypen, Operatoren

Einführung in die Programmierung 1

Sommersemester 21

TU Wien

Überblick

- Variablen und Datentypen
- Operatoren

Variablen und Datentypen

Variablen

- Variable
  - Ein Programm verarbeitet **Daten**, die in **Variablen** abgelegt werden
  - Ein Programm legt die **Ergebnisse** wieder in solchen **Variablen** ab
  - Eine Variable ist eine **benannte Speicherstelle**
- Einige Kennzeichen für eine Variable
  - Name
  - Datentyp
  - Wert

Variable

- Wird einmal vereinbart (deklariert)
  - Name (Bezeichner)
  - Datentyp
- Beispiel
  - Variable für eine ganze Zahl mit dem Namen `number` deklarieren

```
int number;
```

  - Bedeutung
    - **`int`** = Datentyp (steht für ganze Zahlen)
    - `number` = Name (für Zugriff)

Datentyp

- Daten haben einen bestimmten Typ (Datentyp)
- Der Datentyp definiert
  - Menge von **Werten**, die zu diesem Typ gehören (Wertebereich)
  - Menge von **Operationen**, die mit den Werten des Typs ausgeführt werden können
- Compiler kann vor der Ausführung des Programms Überprüfungen durchführen

Einfache Datentypen in Java

Ganze Zahlen	byte, short, int, long
Gleitkommazahlen	float, double
Logische Werte	boolean
Zeichen	char, *String*

- Hinweis zu String (Sequenz von Zeichen)
  - Ist kein einfacher Datentyp
  - Wird in bestimmten Situationen wie ein einfacher Datentyp behandelt

Name (Bezeichner)

- Bezeichner
  - Ist eine Folge von Zeichen
  - Einschränkungen
    - Java-Grammatik gibt erlaubten Aufbau vor
    - https://docs.oracle.com/javase/specs/jls/se15/html/jls-3.html#jls-3.8
    - Beispiele

Korrekte Bezeichner	Nicht korrekte Bezeichner
counter	1counter
carWheel	car-Wheel oder car/Wheel
MAX_COUNT	while
test2	$%

Sinnvolle Bezeichner

- Bezeichner sollten bestimmten Regeln folgen
  - Lesbar und verständlich
  - Compiler wird alle korrekten Bezeichner akzeptieren
  - Ein korrekter Bezeichner muss noch lange nicht sinnvoll sein

Einige Beispiele	**Negatives Beispiel**	**Positives Beispiel**
Vollständige Wörter	c ★	counter
Kleinschreibung	COUNTER ★★	counter
Neue Teile mit Großbuchstaben („lowerCamelCase“)	firstletterintext	firstLetterInText
Sinnvolle Bezeichner	$0x0l	number
Abkürzungen vermeiden	bc	byteCount
Englische Bezeichner bevorzugen	erschtaBuchstobn	firstLetter

★ Nur bei Schleifenvariablen und kleinen Test- oder Beispielprogrammen üblich

★★ Nur bei Konstanten üblich

Wert zuweisen

- Form
  - `Variable = Wert;`
- Ablauf (Aktion)
  - Rechte Seite wird zuerst ausgewertet
  - Danach wird der Wert der linken Seite zugewiesen
- Aufbau
  - Links steht eine Variable
  - Rechts kann z. B. ein fixer Wert (Literal), eine Variable, eine Berechnung etc. stehen

Deklaration und Initialisierung (1)

- Deklaration mit Initialisierung

```
int number = 10;
```

- Einfache Deklaration, danach Zuweisung (ohne Datentyp)

```
int number;
…
number = 10;
```

Deklaration und Initialisierung (2)

- Deklaration mit Initialisierung, Deklaration mit Zuweisung aus anderer Variable bzw. einem Ausdruck

```
int number = 10;
int number2 = number;
int number3 = number * 42 + 23;
```

- Hinweis
  - Vor der ersten Verwendung auf der rechten Seite **muss** eine Variable einen Wert erhalten (entweder explizit oder automatisch initialisiert)

Literal

- Direkte Darstellung von Werten
- Beispiele
  - int: 7482, 123_456, -234, 012, 0x3F, 0b11011
  - double: 24.75, -0.3423, 2., .3, 1.34e2
  - boolean: true, false
  - char: 'a', '1', '%', '\n'
  - String: "Hello", "Input:"
- Unterschiedliche Darstellungsmöglichkeiten
  - Z. B. dezimal, hexadezimal, oktal, binär für ganze Zahlen
- Aufbau
  - https://docs.oracle.com/javase/specs/jls/se15/html/jls-3.html#jls-3.10

Ausgabe von Literalen und Variablen (1)

- Ausgabe
  - Muss nicht selbst programmiert werden
  - Wiederverwendung von Bibliotheksmethoden aus der Java-API (Application Programming Interface)
- Aufbau

```
System.out.println(…);
```

Klasse

Standard Ausgabestrom

Methode

Ausgabe von Literalen und Variablen (2)

- Beispiele für Varianten

```
System.out.print(4);      // Prints 4 to console
System.out.print("Hello"); // Prints Hello
System.out.print(a);      // Prints value of variable a
System.out.println(a);    // Prints value of variable and goes to next line
System.out.println();     // Prints empty line
System.out.print(4 + 2); // Prints 6 to console
System.out.print("Hello" + 4); // Prints Hello4 (concatenates characters)
```

Ausdruck (Verknüpfung von Variablen und/oder Literalen durch Operatoren)

Kommentare in Java

- Dienen zum Beschreiben des Codes, werden nicht übersetzt
- Formen
  - Einzeilig

```
// Prints numbers ranging …
```

  - Mehrzeilig

```
/*
Prints
numbers
ranging …
*/
```

  - Dokumentation

```
/**
This method …
*/
```

Beispiel (Deklarationen)

```java
public class Declarations {

    public static void main(String[] args) {
        int number = 145;
        long largeNumber = 100000000000000L;
        float weight = 75.5f;
        double heavyWeight = 2534.25;
        char input = 'a';
        boolean check = true;
        System.out.println(number + " " +
                largeNumber + " " +
                weight + " " +
                heavyWeight + " " +
                input + " " +
                check);
    }
}
```

```
145 100000000000000 75.5 2534.25 a true
```

Konstanten

- Finale Variablen
  - Deklaration (danach keine Zuweisung möglich!)

```
final int x = 10;
final double d = 10.0;
```

  - Aufgeschobene Initialisierung

```
final int x;
…
…
x = 10;
```

- Konstanten

```
final static int VAL = 10;
```

Einfache Datentypen in Java – Übersicht

Typ	Größe in Bits	Werte	Standard
boolean		true oder false [Repräsentation von der JVM abhängig]	
char	16	'\u0000' bis '\uFFFF' (0 bis 65535)	(ISO Unicode character set)
byte	8	–128 bis +127 (-2^7 bis $2^7 - 1$)	
short	16	–32 768 bis +32 767 (-2^{15} bis $2^{15} - 1$)	
int	32	–2 147 483 648 bis +2 147 483 647 (-2^{31} bis $2^{31} - 1$)	
long	64	–9 223 372 036 854 775 808 bis +9 223 372 036 854 775 807 (-2^{63} bis $2^{63} - 1$)	
float	32	Negativer Bereich: –3,4028234663852886e+38 bis –1,40129846432481707e–45 Positiver Bereich: 1,40129846432481707e–45 bis 3,4028234663852886e+38	(IEEE 754 floating point)
double	64	Negativer Bereich: –1,7976931348623157e+308 bis –4,94065645841246544e–324 Positiver Bereich: 4,94065645841246544e–324 bis 1,7976931348623157e+308	(IEEE 754 floating point)

Operatoren

Operatoren

- Über Operatoren erhalten wir die Möglichkeit, Variablen mit Inhalten zu belegen oder die in ihnen gespeicherten Werte zu verändern

Operatoren – Operandenanzahl

- Unäre Operatoren (besitzen einen Operanden)
  - `-x`
- Binäre Operatoren (besitzen zwei Operanden)
  - `a + b`
- Ternäre Operatoren (besitzen drei Operanden)
  - `evaluation ? firstResult : secondResult`

Operatoren – Position

- Präfixform
  - Operator steht vor seinem bzw. seinen Operanden
  - `++counter`
- Postfixform
  - Operator steht hinter seinem bzw. seinen Operanden
  - `counter++`
- Infixform
  - Operator steht zwischen seinen Operanden (binäre Operatoren)
  - `a + b`

Operatoren – Auswertungsreihenfolge

- Linksassoziativ
  - Auswertung von links nach rechts
  - z. B. `3 + 4 + 5` entspricht `(3 + 4) + 5`
- Rechtsassoziativ
  - Auswertung von rechts nach links
  - Beispiel siehe nächste Folie

Zuweisungsoperator =

- Binärer Operator
  - Beispiele

```
int a;
a = 1;
int b = 3;
```

- Rechtsassoziativ und sein Wert muss nicht konstant sein
  - Beispiel

```
int a = 1;
int b = a;
```

  - Komplexeres Beispiel

```
int a, b;
a = b = 1;
```

Arithmetische Operatoren

- Linksassoziativ wenn binär, rechtassoziativ wenn unär
- Die wichtigsten arithmetischen Operatoren

Name	**Verwendung**	**Operator liefert**
Minus (unär)	- Op1	Vorzeichenwechsel
Minus (binär)	Op1 - Op2	Differenz
Plus	Op1 + Op2	Summe
Multiplikation	Op1 * Op2	Produkt
Division	Op1 / Op2	Ganzzahlige Division bei ganzen Zahlen (Nachkommaanteil wird abgeschnitten), Quotient
Modulo	Op1 % Op2	Rest bei Division

Beispiel (ganzzahlige Berechnungen)

```
public class Operators {

    public static void main(String[] args) {
        int x = 10;
        int y = 20;
        int z = 30;
        int a = x * y;
        int b = a + -z - 15;
        System.out.println(a + " " + b);
        a = b / x;
        b = b % x;
        System.out.println(a + " " + b);
    }
}
```

```
200 155
15 5
```

Verkürzte Schreibweise

- Allgemeine Form

Operation	**Bezeichnung**	**entspricht**
Op1 += Op2	Additionszuweisung	Op1 = Op1 + Op2
Op1 -= Op2	Subtraktionszuweisung	Op1 = Op1 - Op2
Op1 *= Op2	Multiplikationszuweisung	Op1 = Op1 * Op2
Op1 /= Op2	Divisionszuweisung	Op1 = Op1 / Op2
Op1 %= Op2	Modulozuweisung	Op1 = Op1 % Op2

Verkürzte Schreibweise – Beispiele

- Gegebene Deklarationen (ganze Zahlen)
  - `c = 3`
  - `d = 5`
  - `e = 4`
  - `f = 6`
  - `g = 12;`

Operator	Ausdruck	Erklärung	Ergebnis
+=	c += 7	c = c + 7	10
-=	d -= 4	d = d - 4	1
*=	e *= 5	e = e * 5	20
/=	f /= 3	f = f / 3	2
%=	g %= 9	g = g % 9	3

Inkrement und Dekrement

- Inkrementoperator (++) bzw. Dekrementoperator (--)
  - Wert einer Variable um 1 erhöhen bzw. verringern
- Beispiel ++
  - `a++;` entspricht `a += 1;` entspricht `a = a + 1;`

Operator	Benennung	Beispiel	Erklärung
++	Präinkrement	++a	a wird **vor** seiner weiteren Verwendung um 1 erhöht
++	Postinkrement	a++	a wird **nach** seiner weiteren Verwendung um 1 erhöht
--	Prädekrement	--b	b wird **vor** seiner weiteren Verwendung um 1 verringert
--	Postdekrement	b--	b wird **nach** seiner weiteren Verwendung um 1 verringert

Beispiel (Inkrementierung)

```java
public class IncrementTest {

    public static void main(String[] args) {
        int a = 0, b = 0, c = 0;
        System.out.println("a = " + a + ", b = " + b + ", c = " + c);
        a = 3;
        System.out.println("a = " + a + ", b = " + b + ", c = " + c);
        b = ++a;
        System.out.println("a = " + a + ", b = " + b + ", c = " + c);
        c = a++;
        System.out.println("a = " + a + ", b = " + b + ", c = " + c);
        while (a < 10) {
            System.out.println(a);
            a++;
        }
    }

}
```

```
a = 0, b = 0, c = 0
a = 3, b = 0, c = 0
a = 4, b = 4, c = 0
a = 5, b = 4, c = 4
5
6
7
8
9
```

Ganze Zahlen – Endlichkeit

- Keine Anzeige eines Überlaufs
- Beispiel

```
int x, y;
x = Integer.MAX_VALUE; // x = 2147483647
y = x + 1; // y = -2147483648 = Integer.MIN_VALUE
y = x * 2; // -2 !!!
```

Konstante MAX_VALUE in der Klasse Integer

- Eigenarten
  - Bei additiven Operationen kann über die Grenzen hinaus und wieder zurück gerechnet werden
  - Bei multiplikativen Operationen ist dabei Vorsicht geboten

Ganze Zahlen – Abbruch

- Division durch 0 nicht erlaubt
- Sofortiger Programmabbruch
  - Es wird eine Ausnahme (Exception) geworfen
  - Z. B. Ausgabe in IntelliJ
    - Exception in thread "main" java.lang.ArithmeticException: / by zero …
  - Nicht fortfahren und „Ergebnis“ ignorieren

Gleitkommazahlen – Probleme (1)

- Die Größe und die Genauigkeit sind endlich
  - Nicht alle Zahlen können genau dargestellt werden (z. B. 0.1)
    - Berechnungen führen manchmal zu „falschen“ Ergebnissen
  - Beispiele

```
System.out.println(0.1 + 0.2); // 0.30000000000000004
System.out.println(0.1 + 0.3); // 0.4
```

- Grundsätzlich sollte daher bei Gleitkommazahlen (und Berechnungen damit) **nicht** auf Gleichheit getestet werden

Gleitkommazahlen – Probleme (2)

- Probleme bei arithmetischen Operationen mit unterschiedlich großen Zahlen

```
double epsilon = 1E-14;
double x = 10.0 + epsilon;
double y = epsilon / (x - 10.0);
```

  - y sollte 1 sein
  - Obiger Code ergibt aber 0.9382499223688533

Ausdruck (expression)

- Auswertung ergibt einen einzigen wohl definierten Wert
- Kann Teil eines größeren Ausdrucks sein
- Einfache Form
  - Konstante, Zahl etc.
- Komplexere Form
  - Kombination von Operatoren und Operanden
  - Beispiel: 3 + 4 erzeugt den Wert 7

Anweisung (statement)

- Einzelne Vorschrift, die im Rahmen der Abarbeitung des Programms auszuführen ist
- Bisherige Beispiele
  - Deklarationen
  - Durch Semikolon abgeschlossene Ausdrücke, z. B. `x = x + y;`
  - if-Anweisung
  - while-Schleife
- Anweisungen haben keinen Wert
  - Sie können nicht Teil eines größeren Ausdrucks sein
  - Sie können aber **Seiteneffekte** haben
    - `i++;` möglich und sinnvoll (i wird um 1 erhöht)

Vergleichsoperatoren

- Vergleiche liefern einen booleschen Wert (true, false)
- Operatoren für Vergleiche

Java - Notation	Mathematische Notation	Beispiel für true	Beispiel für false
a < b	a < b	2 < 13	2 < 2
a > b	a > b	13 > 2	2 > 13
a <= b	a ≤ b	2 <= 2	3 <= 2
a >= b	a ≥ b	3 >= 2	2 >= 3
a == b	a = b	2 == 2	2 == 3
a != b	a ≠ b	2 != 3	2 != 2

Logische Ausdrücke

- Wertebereich umfasst 2 Werte
  - `true` und `false`
- Beispiele für logische Operatoren
  - Negation: !
  - Oder: || (|)
  - Und: && (&)
  - XOR: ^

a	!a
true	false
false	true

a	b	a && b	a || b	a ^ b
false	false	false	false	false
false	true	false	true	true
true	false	false	true	true
true	true	true	true	false

Beispiele für logische Operatoren

```
int counter = 1;
while (counter <= 15) {
    if ((counter % 2 == 0) && (counter % 3 == 0)) {
        System.out.println(counter);
    }
    counter = counter + 1;
}
```

```
6
12
```

```
int counter = 1;
while (counter <= 15) {
    if (!(counter % 2 == 0) && !(counter % 3 == 0)) {
        System.out.println(counter);
    }
    counter = counter + 1;
}
```

```
1
5
7
11
13
```

```
int counter = 1;
while (counter <= 15) {
    if ((counter % 2 == 0) || (counter % 3 == 0)) {
        System.out.println(counter);
    }
    counter = counter + 1;
}
```

```
2
3
4
6
8
9
10
12
14
15
```

```
int counter = 1;
while (counter <= 15) {
    if (!(counter % 2 == 0) || !(counter % 3 == 0)) {
        System.out.println(counter);
    }
    counter = counter + 1;
}
```

```
1
2
3
4
5
7
8
9
10
11
13
14
15
```

```
int counter = 1;
while (counter <= 15) {
    if ((counter % 2 == 0) ^ (counter % 3 == 0)) {
        System.out.println(counter);
    }
    counter = counter + 1;
}
```

```
2
3
4
8
9
10
14
15
```

```
int counter = 1;
while (counter <= 15) {
    if (!(counter % 2 == 0) ^ !(counter % 3 == 0)) {
        System.out.println(counter);
    }
    counter = counter + 1;
}
```

```
2
3
4
8
9
10
14
15
```

Logische Operatoren – Auswertung

- Teilweise Auswertung („*Short circuit evaluation*“)
- `x && y` ist immer `false`, falls `x == false`
  - Sobald x auf false auswertet, wird die gesamte Auswertung beendet
- `x || y` ergibt immer `true`, falls `x == true`
  - Sobald x auf true auswertet, wird die gesamte Auswertung beendet

Teilweise auswerten

- Erhöhte Robustheit
- y darf Berechnungen enthalten, die ungültig sind im Fall
  - (`x && y`) bei `x == false`
  - (`x || y`) bei `x == true`
- Beispiel
  - `(x != 0 && y/x > 2)` – keine Division durch 0 möglich
- Vollständige Auswertung mit & und |

Operatoren zur Manipulation von Bits

- Direkte Manipulation der einzelnen Bits von Ganzzahlen

Java - Notation	Bedeutung
&	UND-Operator für Bits
|	ODER-Operator für Bits
^	XOR-Operator für Bits
~	Negation der Bits
<<	Logischer Shift nach links
>>	Arithmetischer Shift nach rechts
>>>	Logischer Shift nach rechts

Beispiel (Bitoperationen)

Achtung: Künstliches Beispiel mit Zahlen, praktische Beispiele folgen noch in späteren Vorlesungen!

```
public class BitTest {
    public static void main(String[] args) {
        int a = 15;                    // 00000000 00000000 00000000 00001111
        int b = 28;                    // 00000000 00000000 00000000 00011100
        System.out.println(a & b);     // 00000000 00000000 00000000 00001100 -> 12
        System.out.println(a | b);     // 00000000 00000000 00000000 00011111 -> 31
        System.out.println(a ^ b);     // 00000000 00000000 00000000 00010011 -> 19
        int c = -118;                  // 11111111 11111111 11111111 10001010
        System.out.println(~c);        // 00000000 00000000 00000000 01110101 -> 117
        System.out.println(a << 1);    // 00000000 00000000 00000000 00011110 -> 30
        System.out.println(a << 2);    // 00000000 00000000 00000000 00111100 -> 60
        System.out.println(b >> 1);    // 00000000 00000000 00000000 00001110 -> 14
        System.out.println(b >> 3);    // 00000000 00000000 00000000 00000011 -> 3
        System.out.println(c >> 2);    // 11111111 11111111 11111111 11100010 -> -30
        System.out.println(c >>> 2);   // 00111111 11111111 11111111 11100010 ->
                                       // 1073741794
    }
}
```

Operatorvorrang

- Vorrang bei unterschiedlichen Operatoren
  - Zum Beispiel „Punkt- vor Strichrechnung“
- Regelt die implizite Klammerung bei Operatoren auf **verschiedenen** Stufen
- Zu unterscheiden von Assoziativität
  - Regelt die implizite Klammerung bei Operatoren auf **gleicher** Stufe

Operatorvorrang – Beispiel

- `x = 2 + 4 * 5;`
- Operatoren: *, +, = (geordnet nach Vorrang)
- 4 * 5 auswerten
- 2 + 20 berechnen
- 22 der Variable x zuweisen

Operatorrangfolge in Java (nicht vollständig)

Stärke	Vorranggruppen	Assoziativität
14	( ), [ ], ., ++ (postfix), -- (postfix)	links
13	++ (präfix), -- (präfix), +(unär), -(unär), ~, !, (type), new	rechts
12	*, /, %	links
11	+, -, +(String-Konkatenation)	links
10	<<, >>, >>>	links
9	<, <=, >, >=, instanceof	links
8	==, !=, == (Referenz), != (Referenz)	links
7	& (bitweises UND), & (logisches UND)	links
6	^ (bitweises XOR), ^ (logisches XOR)	links
5	| (bitweises ODER), | (logisches ODER)	links
4	&& (logisches UND mit Short-Circuit-Evaluation)	links
3	|| (logisches ODER mit Short-Circuit-Evaluation)	links
2	?: (bedingte Auswertung)	rechts
1	=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, >>>=	rechts

Ausdrücke und unterschiedliche Datentypen

- Variablen unterschiedlichen Typs in einem Ausdruck
  - Welchen Typ hat der Ausdruck?
- Für alle arithmetischen Operationen gilt
  - Der „kleinere“ Operand wird vor der Ausführung der Operation in den „größeren“ konvertiert
  - Der Ausdruck bekommt dann den gleichen Typ wie seine Operanden
  - Der kleinste Typ ist aber zumindest `int`
- Beispiel (a + b ist vom Typ long)

```
int a = 1234567;
long b = 12345678L;
long c = a + b;
```

Typkompatibilität

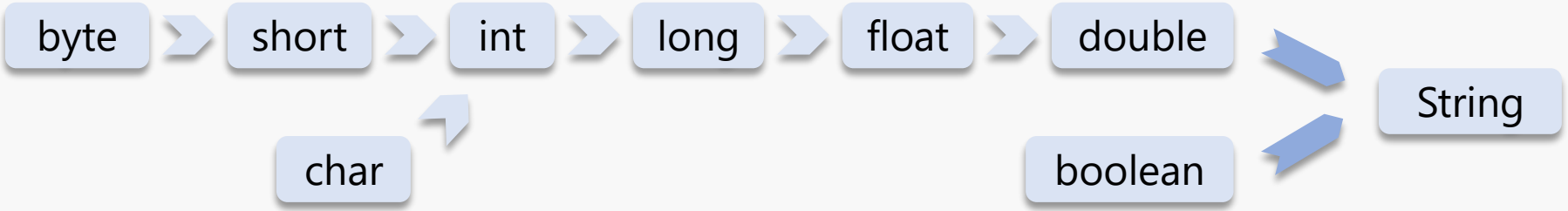
- Operanden eines Ausdrucks müssen kompatibel sein
- Beispiel für ein Problem

```
int x;
x = 2.5;
```

  - Gibt einen Fehler (!), da die Datentypen nicht kompatibel sind
- Zwei Typen sind kompatibel, wenn
  - sie gleich sind,
  - wenn sie durch implizite Typanpassung zum gleichen Typ führen

Implizite Typanpassung

- Automatische Typanpassung
- Nur in eine Richtung (**vereinfacht**)



Explizite Typanpassung (Cast)

- Form

```
(Zieltyp) Ausdruck
```

- Beispiel von vorher

```
int x;
x = (int) 2.5;
```

- Kann zu Datenverlusten führen!

Beispiel (Typanpassung)

```
public class TypeTest {
    public static void main(String[] args) {
        short s = 10;
        float f = 100.5f;
        double d = 200.1, sum;
        boolean flag = true;
        char character = 'x';
        String header = "Results:";
        System.out.println(header);
        System.out.println("s = " + s + " f = " + f + " d = " + d);
        sum = s + f + d;
        System.out.println(flag);
        System.out.println(character);
        System.out.println("sum = " + sum);
        System.out.println(2 + "test" + 3);
        System.out.println(2 + 3 + "test");
        System.out.println("test" + 2 + 3);
        System.out.println("test" + 2 * 3);
    }
}
```

```
Results:
s = 10 f = 100.5 d = 200.1
true
x
sum = 310.6
2test3
5test
test23
test6
```

Local Variable Type Inference (ab Java 10)

- Keine explizite Typangabe auf der linken Seite einer Variablendeklaration
  - Nur bei lokalen Variablen (z. B. innerhalb von main)
  - Compiler muss anhand der rechten Seite den konkreten Typ ermitteln können
- Beispiel (sehr einfach)

```
public class VarTest {
    public static void main(String[] args) {
        var a = 10;
        var b = 12.5;
        var c = "Hello";
        var d = Integer.MAX_VALUE;
        var e = Long.parseLong("123");
        System.out.println(a + " " + b + " " + c + " " + d + " " + e);
    }
}
```

Methode, die einen Wert vom Typ `long` zurückliefert

```
10 12.5 Hello 2147483647 123
```

Daten einlesen

Ein erstes kurzes Beispiel

Motivation

- Variablen immer mit fixen Werten belegen
  - Nicht praxistauglich
  - Oft möchten wir Daten selbst eingeben und damit die Daten in unserem Programm (teilweise) bestimmen
- Eingabe von Daten
  - Unterschiedliche Möglichkeiten
  - Nachfolgend wird die Eingabe über Kommandozeile mittels Scanner beschrieben

Einlesen von Daten (Scanner)

- Scanner
  - Einlesen aus dem Eingabefenster über `System.in`
- Aufgabe des Scanners
  - Unterteilt Eingabetext in sogenannte Tokens
- Tokens
  - Primitive Datentypen oder Strings
  - Standardmäßig durch Leerzeichen getrennt
- Anlegen einer Scanner-Variable
  - Komplexer als bei einfachen Variablen
  - Klasse Scanner muss dafür auch importiert werden
    - Ist nicht automatisch bekannt

Beispiel (Einlesen einer Zahl, einfach)

```
import java.util.Scanner;

public class ReadInteger {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int i = input.nextInt();
        System.out.println(i);
    }

}
```

Weist darauf hin, dass (die Klasse) Scanner benutzt wird

Scanner-Variable auf `System.in` anlegen

Methode (Funktion) `nextInt` aufrufen. Funktioniert nur korrekt, wenn auch die Eingabe eine ganze Zahl ist. Details folgen noch!

Beispiel (Zeitumrechnung)

```
import java.util.Scanner;

public class TimeCalc {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Please insert number of seconds: ");
        int seconds = input.nextInt();
        System.out.println(seconds + " seconds are "
                + (seconds / 3600) + " hour(s), "
                + (seconds % 3600 / 60) + " minutes and "
                + (seconds % 60) + " seconds");
    }
}
```

```
Please insert number of seconds: 1234
1234 seconds are 0 hour(s), 20 minutes and 34 seconds
```

Beispiel (Durchschnitt von drei Zahlen)

```
First number: 10
Second number: 30
Third number: 40
Average = 26.666666666666668
```

```java
import java.util.Scanner;

public class AverageOfThreeNumbers {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int a, b, c;
        System.out.print("First number: ");
        a = input.nextInt();
        System.out.print("Second number: ");
        b = input.nextInt();
        System.out.print("Third number: ");
        c = input.nextInt();
        System.out.println("Average = " + (a + b + c) / 3.0);
    }

}
```

Zusammenfassung

- Variablen
  - Deklarationen
  - Datentypen
  - Zuweisungen
- Operatoren
  - Arithmetische Operatoren
  - Logische Operatoren
  - Shift-Operatoren
  - Operatorrangfolge
  - Typanpassung
- Einlesen von einfachen Daten