Kapitel 2: Einfache Programme

- Grundsymbole
- Schlüsselwörter, Kommentare, Literale
- Bezeichner, Variablen, Konstanten
- Typen und Typprüfung
- Zuweisungen
- Operatoren und arithmetische Ausdrücke
- Ein-/Ausgabe

Grundsymbole

- Lexikalische Einheiten
  - Zeichenorientierte Sicht: Einzelne Zeichen als Einheiten (‘w‘ ‘h‘ ‘i‘ ‘l‘ ‘e‘).
  - Symbolorientierte Sicht: Lexikalische Einheiten als Symbole (‘while‘).
- Arten von Grundsymbolen
  - Struktursymbole { } ( ) ;
  - Schlüsselwörter **while if class** …
  - Kommentare /* … */ // …
  - Werte (Konstanten) 1 3.14 ‘a‘ “Die Summe ist: “
  - Bezeichner (Namen) *sum i n int*
  - Operatoren , . ( ) [ ] + – * / = < > …

Schlüsselwörter und Struktursymbole

- Syntaktische Bedeutung
  - dienen zur Strukturierung von Programmen.
  - Struktursymbole: { } ( ) ;
  - Schlüsselwörter: **while**, **if**, **class**, …
  - Java: alle Schlüsselwörter in Kleinbuchstaben
- Schlüsselwörter sind *reservierte Wörter*
  - dürfen nicht als Bezeichner für Variablen etc. verwendet werden.
  - Fehlerbeispiele

    int *while* = 27;

    String *class* = “1A“;

Liste aller Java-Schlüsselwörter

abstract	double	int	static
boolean	else	interface	super
break	extends	long	switch
byte	final	native	synchronized
case	finally	new	this
catch	float	null	throw
char	for	package	throws
class	goto	private	transient
const	if	protected	try
continue	implements	public	void
default	import	return	volatile
do	instanceof	short	while

Kommentare

- Kommentare (Anmerkungen) in Programmen
  - Werden bei der Übersetzung ausgeblendet
  - Dienen der Erklärung und Dokumentation für den (menschlichen) Leser
- Kommentare bis zum Zeilenende
  - Stehen zwischen ‘//‘ und Zeilenende: // example of a comment
- Klammerkommentare
  - Stehen zwischen ‘/*‘ und ‘*/‘: /* example of a comment */
  - Können sich über mehrere Zeilen erstrecken
  - Dürfen nicht geschachtelt sein
  - Eignen sich zum „Auskommentieren“ von Programmteilen

Kommentare

- Beispiel für einen ungültigen Kommentar

```
/* A wrong comment:
   /* nested comments are not allowed */
   // but this is ok
... */
```

Verwendung von Kommentaren

- Regeln zur Verwendung
  - Nur das kommentieren, was nicht unmittelbar im Programm steht
    Gegenbeispiel `int n; // number`
- Javadoc
  - Klammerkommentare zwischen '/**' und '*/'
  - Werden in automatisch generierte Dokumentation übernommen
- Sprache: Deutsch oder Englisch?
  - *Englisch*: Passt besser zu Schlüsselwörtern (if, while, …) · meist kürzer · internationale Kooperation einfacher
  - *Deutsch*: Hierzulande einfacher verständlich
  - Nicht Sprachen mischen!

Übersetzung und Dokumentation

Programm MyClass.java

```
/** my demo class */
class MyClass {
   /**
    … ⟨Javadoc-Kommentar⟩ …
   */
   … ⟨Deklarationen⟩ …
   … ⟨Anweisungen⟩ …
}
```

Übersetzer → **Bytecode MyClass.class**

– enthält ausführbaren Bytecode
– Kommentare wurden verworfen

Javadoc → **Dokumentation MyClass.html**

– enthält Deklarationen und Javadoc-Kommentare
– Anweisungen wurden verworfen

Wertekonstanten (Literale)

- Literale stellen Werte dar (unbenannte oder anonyme Konstanten)
  Jeder Wert gehört zu einem Typ, der aus der Notation des Wertes hervorgeht.

- Beispiele
  - Ganze Zahlen (*integer*): 1 32768 0 -3 (dezimal)
    0x2A (hexadezimal, 2·16+10=42)
  - Gleitpunktzahlen (*float/double*): 1.0 3.1415926 3.0e-5 1.0f
  - Zeichen (*char*): ‘a‘ ‘A‘ ‘0‘
  - Zeichenketten (*String*): “Die Summe ist: “
  - Wahrheitswerte (*boolean*): true false

- Beispiele für Sonderzeichen (in *char* und *String*)

\n	Zeilenschaltung („newline“)
\t	Tabulatorschritt
\\	Zeichen ‘\‘ (Backslash)
\‘ \“	einfache bzw. doppelte Anführungsstriche

Bezeichner

- Bezeichner (Identifikatoren) sind Namen für Objekte
  - Bez. werden explizit deklariert und mit einem Typ versehen.
  - Bez. bezeichnen Variablen, Typen, Klassen, Operationen, …
  - Bez. stellen Querverbindungen in Programmen her.

- EBNF

Ident	= *Letter* { *Letter* | *Digit* }.
Letter	= “a“ | “b“ | … | “z“ | “ä“ | “ö“ | “ü“ | “A“ | … | “Z“ | … | “_“ | “$“.
Digit	= “0“ | “1“ | “2“ | “3“ | “4“ | “5“ | “6“ | “7“ | “8“ | “9“.

- Aufbau von Bezeichnern
  - Bez. bestehen aus Buchstaben, Ziffern und Unterstrichen (‘_‘).
  - Bez. müssen mit einem Buchstaben beginnen.
  - Bez. können beliebige Länge haben.
  - Groß- und Kleinschreibung ist signifikant! Also: min ≠ Min

Bezeichner: Beispiele

- Beispiele und Gegenbeispiele

x3y	ok
sum	ok
zähler	ok
3B	keine Ziffer zu Beginn!
a name	kein Leerzeichen!
10%ofSum	keine Sonderzeichen!

- Wahl von Bezeichnern
  - Lesbar, aber nicht zu lang: sum, value
  - Hilfsvariablen eher kurz: i, j, x, y
  - deutsch oder englisch? – vgl. Kommentare
  - Zusammenfügung von Wörtern durch
    - Großbuchstaben: inputFactor
    - Unterstriche: input_factor

Deklaration von Variablen

- Variablendeklaration

String *title*;	*title* bezeichnet einen (noch undefinierten) String
int x, y;	x, y sind Variablen für ganzzahlige Werte

- Variablendeklaration mit Initialisierung

  String *title* = “Die Summe ist: “;

  int x = 0, y = 1;

- Vorgegebene Standardbezeichner in Java
  - Basistypen: int, float, double, char, String, boolean
  - Variablen: System.in, System.out

Benannte Konstanten

- Charakterisierung
  Benannte Konstanten sind unveränderbare „Variablen“:
  - bekommen bei der Deklaration einen Wert zugewiesen.
  - belegen keinen Speicherplatz.
- Konstantendeklarationen
  - Stehen auf oberster (d.h. Klassen-)Ebene.
  - Werden eingeleitet durch **static** und **final**:
    static final float MwSt = 0.16; // Mehrwertsteuersatz 16%
- Nutzen benannter Konstanten
  - Bessere Lesbarkeit von Programmen (‘MwSt‘ ist klarer als ‘0.16‘).
  - Bessere Änderbarkeit (Wert muss nur an einer Stelle geändert werden).

Datentypen

- Ein Datentyp definiert …
  - die Menge von Werten, die zu diesem Typ gehören.
  - die Operationen, die auf Objekten des Typs angewendet werden können.
- Beispiel *int*
  - Wertemenge: ganze Zahlen.
  - Operationen: arithmetische Operationen, Vergleiche.
- Rolle von Datentypen
  - Der Datentyp legt den benötigten Speicherplatz für Variablen fest.
    Variable ist benannter Speicherplatz bestimmter Größe.
  - Die Zulässigkeit von Operationen auf Variablen kann geprüft werden.
  - Man kann in Java neue, eigene Datentypen definieren.

Basistypen in Java

- Wahrheitswerte

boolean		true false

- Zeichen

char	16 Bit	gemäß Unicode 1.1.5

- Ganze Zahlen

byte	8 Bit	-128 … 127
short	16 Bit	-32768 … 32767
int	32 Bit	-2 147 483 648 … 2 147 483 647
long	64 Bit	$-2^{63} \dots 2^{63}-1$

- Gleitpunktzahlen

float	32 Bit	gemäß IEEE 754-1985 Standard
double	64 Bit	–“–

Statische Typprüfung

- Sprachen mit statischer Typprüfung
  - Bezeichner bekommen bei der Deklaration einen Typ zugeordnet.
  - Typfehler werden bei der Übersetzung des Programms erkannt.
  - Fehlerbeispiele: char c = “hallo“; int pi = 3.14159;
  - Sprachen: Pascal, Modula2, C++, Java

- Sprachen ohne statische Typprüfung
  - Spielarten
    - Keine Deklaration bzw. auch typfreie Deklaration (Bsp. C) von Variablen.
    - Dynamische Typprüfung: Typfehler werden zur Laufzeit erkannt.
    - Keine Typprüfung: Typfehler werden gar nicht erkannt.
  - Höhere Fehleranfälligkeit bei der Programmierung
  - Sprachen: Lisp, Smalltalk, Prolog, awk

Zuweisungen

- Aufbau einer Zuweisung
  - Basisform in EBNF:

    Assignment = *Ident* "=" *Expr*.
  - Linke Seite enthält Ziel der Zuweisung (in der Regel Variable).
  - Rechte Seite enthält Wert der Zuweisung (Ausdruck).
  - Zuweisungssymbol in Java ist "=".

  - Beispiel:

    sum = sum + zähler
  - Lies nicht „sum *ist* …“ sondern „sum *wird zu* …“ oder „sum *ergibt sich aus* …“.

- Regel: Typen müssen zuweisungskompatibel sein
  - Linker Typ und rechter Typ sind gleich (z.B. *int*) oder
  - Linker Typ schließt rechten Typ ein (Typhierarchie), z.B. *double* $\supset$ *float.*

Arithmetische Ausdrücke

- Binäre Operatoren

+	Addition	3 + 5	(= 8)
–	Subtraktion	5 – 3	(= 2)
*	Multiplikation	3 * 5	(= 15)
/	(ganzzahlige) Division	5 / 3	(= 1)
%	Rest bei Division (modulo)	5 % 3	(= 2)

  Es gilt „Punkt vor Strich“, d.h. *, / und % haben Vorrang vor + und –

- Unäre Operatoren

+	Identität	+ x	(= x)
–	Vorzeichenumkehr	– x	(= –x)

Logische Operatoren

- Vergleichsoperatoren

  Vergleichen zwei Ausdrücke vom selben Typ und liefern *boolean*.

==, !=	gleich, ungleich	5 != 3	(= true)
<, <=	kleiner, kleiner oder gleich	5 < 3	(= false)
>, >=	größer, größer oder gleich	5 >= 3	(= true)

- Boolesche Operatoren (*George Boole*, 1815-1864)

  Verknüpfen Werte vom Typ *boolean*.

&&	und (sequentiell)	x > 0 && x < 0	(= false)
||	oder (sequentiell)	x > 0 || x <= 0	(= true)
!	nicht (unär)	! false	(= true)

- Sequentielle Auswertung

  Bsp. if (y != 0 && x / y > 10) … Division durch Null, wenn keine sequentielle Auswertung stattfindet.

Weitere Operatoren

- Bedingter Ausdruck (ternärer Operator)
  - Unterschiedliche Werte in Abhängigkeit von einer Bedingung

    EBNF: *condexpr* = *condition* "?" *expr* ":" *expr*
  - Beispiel: Maximum zweier Zahlen

    float max = (a>b) ? a : b;
  - Beispiel: Berechnung des Vorzeichens von x

    int sign = (x>0) ? 1 : (x<0) ? –1 : 0;

- Konkatenation von Strings

  \+ Hintereinanderfügung „Hali“ + „hallo“ (= „Halihallo“)

- Zugriff auf Elemente eines Array

  arg [*i*] *i*-tes Element im Array *arg*, $0 \leq i <$ *arg.length*

Typkonversion (cast)

- Implizite Typumwandlung

  byte → short → int → long → float → double - - → String

  char ↗ int; boolean - - - → String

  Beispiele:

      float x = 3; // *int* wird in *float* umgewandelt
      System.out.println ("Sum = " + (m + k)); // Was passiert ohne Klammern?

- Explizite Typumwandlung

  (*type*) *expr* Typumwandlung, z.B. durch Abschneiden

  Beispiel: float x = 4.7f, y = 2.3f; (int) (x/y) ergibt 2

  (int) x/y ergibt 1

- Umwandlung aus *String* ist umständlich

  int n = (int) "437" // geht nicht

  int m = Integer.parseInt ("437"); // so geht es

Spezielle Operatoren für Zahlen

- Zuweisungsoperatoren
  – Das Muster '*x = x op y*' tritt häufig auf.
  – Kurzschreibweisen dafür: x += y, x–= y, x *= y, x /= y, x %= y
  – Eher vermeiden, da meist schwer lesbar.

- Inkrementierung und Dekrementierung
  – Erhöhung oder Erniedrigung um Eins:

Ausdruck	Wert des Ausdrucks	anschließender Wert von x	
x++	x	x + 1	(erst lesen, dann ändern)
++x	x + 1	x + 1	(erst ändern, dann lesen)
x--	x	x – 1	
--x	x – 1	x – 1	

  – Darf nur auf Variablen angewendet werden, nicht auf allgem. Ausdrücke.

Ein- und Ausgabe

- Ausgabe („Standardausgabe“, z.B. auf die Konsole)
  - Variable *System.out* ist vom Typ *PrintStream.*
  - Ausgabefunktionen für die Basistypen.
    System.out.print (*char*), System.out.print (*int*), …
  - Ausgabefunktionen mit abschließender Zeilenschaltung.
    System.out.println(), System.out.println(*char*), …
  - Automatische Konvertierung nach *String*
    System.out.println („Die Summe ist: “ + sum);
  - Details in der Dokumentation nachlesen.

- Eingabe („Standardeingabe“, z.B. von der Tastatur)
  - Variable *System.in* ist vom Typ *InputStream.*
  - Eingabe ist in Java wesentlich komplexer als Ausgabe.

Ein- und Ausgabe

•Ausgabe

Das folgende Beispiel fordert den Benutzer auf, zwei Zahlen einzutippen, liest die Zahlen, addiert sie und gibt ihre Summe aus.

```
int i, j, sum;
Out.print("Type 2 numbers: "); // the user types e.g. 3  17  followed by return key
i = In.readInt(); //reads 3 and stores it in i
j = In.readInt(); // reads 17 and stores it in j
sum = i + j;
Out.println("Sum = " + sum);
```