Grundelemente der Programmierung: Basisdatentypen

Software Entwicklung 1

Annette Bieniusa, Mathias Weber, Peter Zeller

In diesem Kapitel werden wir erste Programme in der Programmiersprache Java schreiben. Wir werden die zwei grundlegenden Varianten zur Ausführung von Programmen vorstellen (Übersetzung und direkte Interpretation) und Variante diskutieren, die für Java gewählt wurde.
Im Mittelpunkt stehen dannach die Begriffe Wert, Ausdruck und Typ. Unsere ersten Java-Programme verwenden Datentypen, die in der Sprache integriert sind: primitive Datentypen (`int`, `double`, `boolean`, `char`) und Strings. Mit Hilfe von mathematischen und logischen Operatoren formulieren wir Ausdrücke und besprechen deren Auswertungssemantik. Als weiteres grundlegendes Programmierkonstrukt führen wir außerdem Variablen ein.
Das Kapitel schließt mit einem Beispielprogramm, die typische Verwendungen diese Konstrukte illustrieren.

1 Erste Schritte in Java

Als erstes Java Programm werden wir hier das “Hello World!” - Programm betrachten. Das Programm soll den Text “Hello World!” auf der Konsole ausgeben. Es wird klassischerweise als Beispiel verwendet, um eine Programmiersprache kurz vorzustellen.

```
/* Ausgabe von "Hello World!" auf der Kommandozeile */
public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

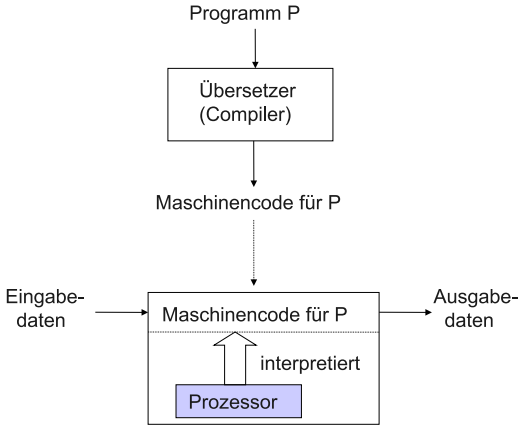
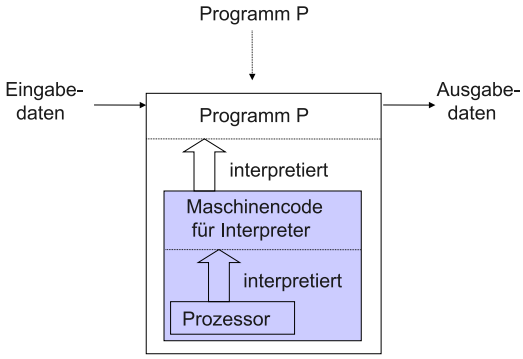
Dieses einfache Java-Programm besteht aus den folgenden Elementen:

- Die erste Zeile `/* ... */` ist ein Kommentar, der das Programm informell beschreibt.
- Das Programm besteht aus einer Klasse, die wir `Hello` genannt haben. Es muss daher in einer Textdatei namens `Hello.java` gespeichert werden.

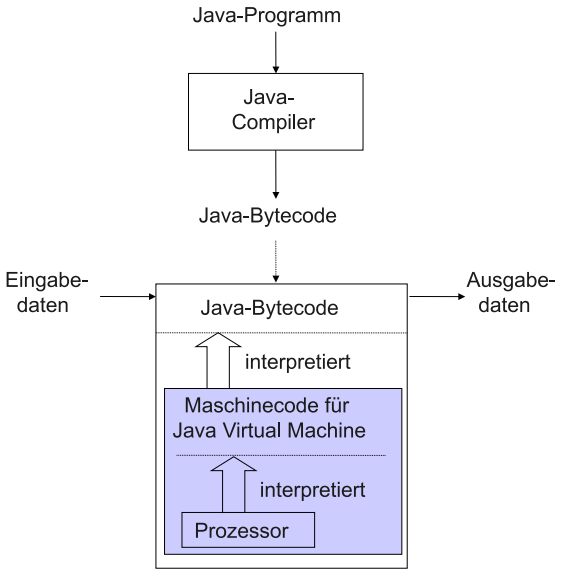
- In dieser Klasse gibt es eine `main()` - Methode; hier startet die Programmausführung.
- Das Programm selbst besteht aus einer Sequenz von Anweisungen, die durch ; voneinander getrennt werden. Diese Anweisungen bilden den ***Rumpf*** der Methode.
- Die Anweisung `System.out.println()` ruft eine Bibliotheksfunktion auf, die einen Text auf der Konsole ausgibt. Der auszugebende Text wird dabei zwischen die Klammern geschrieben.
- Befehlszeilenargumente können mittels `String[] args` übergeben werden (→ siehe Ende dieses Abschnitts)

1.1 Prinzipielle Ausführungsvarianten

Programme können auf verschiedene Arten ausgeführt werden:

1. Mittels Übersetzung: Dabei wird der Programmtext in Maschinensprache übersetzt und dann direkt vom Prozessor ausgeführt.	
2. Mittels direkter Interpretation: Hier wird der Programmtext von einem Interpreter ausgeführt, der wiederum – zusammen mit dem interpretierten Code – vom Prozessor ausgeführt wird.	

Java verwendet eine Mischform aus Übersetzung und Interpretation:



Das Programmieren in Java umfasst grob die folgenden Schritte:

- Programmtext schreiben
- Programm übersetzen / compilieren
- Programm ausführen

Editor ⟶ Hello.java ⟶ Compiler ⟶ Hello.class ⟶ JVM ⟶ "Hello, World!"

Zwei Schritte haben wir hier zunächst vernachlässigt:

- Integration/Komposition mehrerer Programmteile, die unabhängig von einander entwickelt und evtl. übersetzt wurden
- Installation der Programme auf der Host-Maschine

In der Praxis gewinnen diese Schritte durch die Vielzahl an bereits existierenden Teil-Programmen und Plattformen von Bedeutung.

2 Basisdatentypen

Literaturhinweis: Kapitel 1.2 aus R. Sedgewick, K. Wayne: Einführung in die Programmierung mit Java. 2011, Pearson Studium.

Programme verarbeiten Informationen in Form von Daten. Die Art der Daten wird durch den entsprechenden ***Datentyp*** festgelegt. Ein Datentyp hat einen Wertebereich und eine dazugehörige Menge an Operationen
Beispiel:

- Texte und Zeichenfolgen werden als String behandelt; können z.B. zusammengefügt oder in Großbuchstaben umgewandelt werden.
- Messwerte werden oft als Zahlen repräsentiert; sie können z.B. addiert, multipliziert, verglichen werden.

In Java gibt es zwei Arten von ***Werten*** (engl. *values*):

- elementare Datenwerte (Zahlen, Wahrheitswerte, Zeichen, ...)
- Referenzen auf Objekte (⇒ näheres dazu im Abschnitt zu "Objekte")

Wie für abstrakte Objekte oder Begriffe typisch, besitzen Werte

- keinen Ort,
- keine Lebensdauer,
- keinen veränderbaren Zustand,
- kein Verhalten.

Ein ***Typ*** (engl. *type*) charakterisiert Werte, auf denen die gleichen Operationen zulässig sind. Typisierte Sprachen besitzen ein Typsystem, das für jeden Wert festlegt, von welchem Typ er ist.
Ausdrücke sind das Sprachmittel zur Beschreibung von Werten.
Ein ***Ausdruck*** (engl. ***expression***) in Java ist (u.a.)

- ein Literal,
- ein Bezeichner,
- die Anwendung einer Operation auf einen oder mehrere Ausdrücke,
- oder aufgebaut aus Sprachmitteln, die erst später behandelt werden.

Jeder Ausdruck hat einen Typ:

- Der Typ eines Literals ergibt sich aus dem Wertebereich.

- Der Typ eines Bezeichners ergibt sich aus dem Wert, den er bezeichnet.
- Der Typ einer Operation ist der Ergebnistyp der Operation.

Wir betrachten zunächst die Basisdatentypen und elementare Operatoren für Elemente dieser Datentypen, wie man sie in jeder Programmier-, Spezifikations- und Modellierungssprache findet. Die Basisdatentypen (engl. *basic / primitive data types*) bilden die Grundlage zur Definition weiterer Typen und Datenstrukturen.

2.1 Ganze Zahlen / Integer

Dem Typbezeichner `int` ist in Java eine Wertemenge zugeordnet, die die ganzen Zahlen von -2147483648 bis 2147483647 enthält.

Typbezeichner	`int`
Werte	ganzen Zahlen von -2^{31} bis $2^{31}-1$
Typische Literale	`1, 0, -99`
Operationen	`+` (Addition) `-` (Subtraktion) `*` (Multiplikation) `/` (Division) `%` (Modulo)

Beispiele:

Ausdruck	Wert	Hinweis
5 - 3	2	
5 / 3	1	Ganzzahlige Division!
5 % 3	2	
1 / 0		Laufzeitfehler
3 - 5 - 4	-6	Linksassoziativ
3 - (5 - 4)	2	
3 * 5 - 4	11	Priorität

Innerhalb der Wertemenge $[-2^{31}, 2^{31}-1]$ sind die Operatoren auf beschränkten ganzen Zahlen gleich den üblichen mathematischen Operatoren. Außerhalb der Wertemenge ist ihr Verhalten nicht definiert. So wertet beispielsweise 2147483647 + 1 zu -2147483648 aus (***Überlauf***). Weitere Datentypen für ganze Zahlen: `long` (64-bit Integer), `short` (16-bit Integer), `byte` (8-bit Integer).

2.2 Gleitkommazahlen

Dem Typbezeichner `double` ist in Java die (endliche!) Wertemenge der 64-bit Gleitkommazahlen zugeordnet.

Typbezeichner	`double`
Werte	reelle Zahlen (nach IEEE-Standard 754)
Typische Literale	`3.1415, 1.98e20, -1.0`
Operationen	`+` (Addition) `-` (Subtraktion) `*` (Multiplikation) `/` (Division)

Beispiele:

Ausdruck	Wert	Hinweis
3.14 + .03	3.17	
5.0 / 3.0	1.6666666666666667	Rundungsfehler möglich
1.0 / 0.0	`Infinity`	spezieller Wert

Mit Gleitkommazahlen sind im Allg. keine beliebig präzisen Berechnungen möglich. Sie bieten nur eine Näherung an die reelle Zahlen $\mathcal{R}$, haben aber nur endliche Darstellung mit eingeschränkter Präzision. So sind die Integer sind in der Programmierung in vielen Programmiersprachen keine Teilmenge der Gleitkommazahlen! Nicht alle Integer können z.B. präzise als Gleitkommazahl dargestellt werden. Durch die eingeschränkte Präzision kann es leicht zu Rundungsfehlern kommen, die sich akkumulieren können und Ergebnisse unter Umständen sogar unbrauchbar machen. Das Fachgebiet der Numerik befasst sich mit dieser Problematik.
Die Standardisierung der Gleitkommazahlen nach IEEE-Standard 754 definiert neben den Zahlenwerten selbst zwei spezielle Werte:

- `Infinity` (wenn Darstellung zu groß)
- `NaN` (not a number; wenn Berechnung undefiniert)

Weiterer Datentyp für Gleitkommazahlen: `float` (32-bit Gleitkommazahlen)

2.3 Boolesche Werte

Dem Typbezeichner `boolean` ist die Wertemenge der Wahrheitswerte $\{true, false\}$ zugeordnet.

Typbezeichner	`boolean`
Werte	wahr, falsch
Literale	`true`, `false`
Operationen	`&&` (Und) `||` (Oder) `!` (Nicht)

Semantik der Operationen:

- `a && b` ist `true`, wenn beide Operanden `true` sind; sonst `false`

- `a || b` ist `false`, wenn beide Operanden `false` sind; sonst `true`
- `!a` ist `true`, wenn `a` `false` ist; und `false`, wenn `true` ist

Frage 1: Zu welchen Werten ergeben sich folgende Ausdrücke?

```
(7 >= 45) == true
sein || !sein          für geeignete Variable sein
```

Vergleichsoperatoren

Die folgenden Operatoren vergleichen primitive numerische Werte *gleichen Typs* miteinander und liefern als Ergebnis einen boolschen Wert.

`==`	Gleichheit
`!=`	Ungleichheit
`>, >=`	größer bzw. größer gleich
`<, <=`	kleiner bzw. kleiner gleich

Wenn unterschiedliche Funktionen oder andere Programmelemente den gleichen Bezeichner haben, spricht man vom ***Überladen*** des Bezeichners (engl. *overloading*).
Wie in den obigen Datentypen gezeigt, können Operatorbezeichner in Java **überladen** werden, d.h. in Abhängigkeit vom Typ ihrer Argumente bezeichnen sie unterschiedliche Funktionen.
Beispiele:

- `+` arbeitet z.B. auf `int` und `double`
- `==` arbeitet z.B. auf `int` und `boolean`

2.4 Character

Der Datentyp `char` repräsentiert einzelne Zeichen aus dem Unicode-Zeichensatz, z.B. Ziffern, Groß- und Kleinbuchstaben, Satzzeichen, White space (Leerzeichen, Tabularzeichen, Zeilenumbruchzeichen), etc.

Typbezeichner	`char`
Werte	Zeichen
Typische Literale	`'a', '7', 'B'`

`char`-Werte stellen die Unicode-Zeichen eigentlich als Zahlen zwischen 0 und 65535 (16-bit) dar.
Beispiele: `'a' == 97` , `'b' == 98` , `'A' == 65` , `'B' == 66` , `'0' == 48` und `'1' == 49` .
Man kann sie daher vergleichen und auch mit ihnen rechnen:

```
boolean istBuchstabe = (c >= 'a' && c <= 'z')
                     || (c >= 'A' && c <= 'Z');

// Umwandlung von Zeichen '3' zu Zahl 3:
char c = '3';
int i = c - '0';
```

Spezielle Zeichen:

'\"'	doppeltes Anführungszeichen
'\''	einfaches Anführungszeichen
'\n'	Zeilenumbruch
'\t'	Tabulatorzeichen
'\b'	Backspace
'\\'	Backslash-Zeichen

2.5 Strings

Dem Typbezeichner `String` ist in Java die Wertemenge der Zeichenketten zugeordnet.

Typbezeichner	`String`
Werte	Zeichenketten, Text
Literale	`"Hallo", "Die 7 Zwerge",`
Operation	`+` (Konkatenation, Verkettung)

Beispiele:

`"Hello, " + "World!"`	`"Hello, World!"`
`"12 " + "34"`	`"12 34"`
`"12" + "34"`	`"1234"`
`"Die " + 7 + " Zwerge"`	`"Die 7 Zwerge"`

Strings sind in Java keine primitiven Datentypen, jedoch in die Sprachdefinition fest integriert. Wenn primitive Datentypen mit einem String konkateniert werden, werden sie automatisch in Strings umgewandelt. Strings können auf der Kommandozeile direkt ausgegeben oder auch von dort eingelesen werden, wie wir im nächsten Abschnitt sehen werden.

Frage 2: Zu welchen Werten ergeben sich folgende Ausdrücke?

`"seven" + "three"`	ergibt sich zu
`"4" + "5"`	ergibt sich zu
`"" + 9 + 7`	ergibt sich zu
`9 + 7 + ""`	ergibt sich zu
`"7"  + "+" + "5"`	ergibt sich zu

2.6 Präzedenzregeln bei Operatoren

Wenn Ausdrücke nicht vollständig geklammert sind, ist im Allgemeinen nicht klar, wie ihr Syntaxbaum aussieht und wie die Auswertung zu erfolgen hat.

Beispiele :

```
3 == 5 == true
false == true || true
false && true || true
```

Präzedenzregeln legen fest, wie Ausdrücke zu strukturieren sind:

- Am stärksten binden unäre Operatoren[1] in Präfixform[2].
- Binäre Infix-Operatoren in Java sind (in der Regel) linksassoziativ.
- Präzedenzregeln für Infix-Operatoren:
  - `*, / ,%` binden stärker als `+` und `-`
  - `!` bindet stärker als `&&`, und `&&` stärker als `||`
- Boolsche Operatoren `&&` und `||` sind *nicht-strikt*; d.h. die Auswertung beginnt mit dem ersten (linken) Operanden, und nur wenn dieser Wert den Wert des Gesamtausdrucks nicht bestimmt, wird der rechte Operand ausgewertet.
  Beispiel: Die Auswertung des Ausdrucks `(x != 0) && (y / x == 5)` testet zunächst, ob `x != 0`; nur wenn dieser Teilausdruck zu `true` auswertet, wird der zweite Teilausdruck `(y / x == 5)` ausgewertet. Falls `x != 0` zu `false` auswertet, ergibt sich der Gesamtausdruck zu `false`.
  Wichtig: Die arithmetischen Operatoren sind strikt; d.h. hier werden immer alle Operanden ausgewertet.

3 Variablen

Eine ***Speichervariable*** (oder einfach nur: Variable) ist ein Speicher/Behälter für Werte. Dieser Wert kann sich – im Unterschied zur mathematischen Verwendung – im Laufe eines Programms verändern. Charakteristische Operationen auf einer Variablen v sind:

- ***Zuweisen*** eines Werts w an v;
- ***Lesen*** des Wertes, den v enthält/speichert/hat.

Der Zustand einer Variablen v ist undefiniert, wenn ihr noch kein Wert zugewiesen wurde; andernfalls ist der Zustand von v durch den gespeicherten Wert charakterisiert. Variablen stellen wir graphisch meist durch Rechtecke dar:

v:	true	v *enthält/speichert* den Wert `true`
x:	7	x *enthält/speichert* den Wert 7

[1] *Unäre* Operatoren haben einen Operanden, *binäre* Operatoren haben zwei Operanden, *ternäre* Operatoren drei, etc.
[2] Präfix-Operatoren stehen *vor* dem Operator, Infix-Operatoren stehen *zwischen* den Operanden, Postfix-Operatoren *hinter* den Operanden

Variablen werden über ***Bezeichner*** (engl. *identifier*) referenziert. Für Bezeichner muss in Java gelten:

- Bezeichner bestehen aus beliebig vielen Unicode-Zeichen (insbesondere Buchstaben, Ziffern und Unterstrich _).
- Sie starten nicht mit einer Ziffer.
- Bezeichner dürfen nicht mit Schlüsselwörtern oder Konstanten übereinstimmen.
- Bezeichner sind *case-sensitive*, d.h. es wird zwischen Groß- und Kleinschreibung unterschieden.

Eine Variablendeklaration hat in Java die Form:

```
Variablendeklaration →
    Typbezeichner Variablenbezeichner;
```

Sie definiert eine neue Variable vom angegebenen Typ mit dem angegebenem Bezeichner. Variablen müssen immer mit einem Typ deklariert werden und ihnen darf nur ein Wert mit "kompatiblem" Typ zugewiesen werden. Welche Typen miteinander "kompatibel" sind, werden wir im Laufe der Vorlesung sehen. Der Typ einer Variablen wird bei der Deklaration festgelegt und kann sich im weiteren Programmverlauf nicht ändern.
Beispiele:

```
int a;
boolean b;
double meineGleitkommavariable;
```

3.1 Zuweisungen

Syntax in Java:

```
Zuweisung →
    Variable = Ausdruck;
```

Semantik: Werte den Ausdruck aus und weise das Ergebnis der Variablen zu.

In Java ist eine Zuweisung syntaktisch ein Ausdruck, liefert also einen Wert und zwar das Ergebnis der Auswertung von der rechten Seite der Zuweisung.

Beispiele:

```
a = 27 % 23;
b = true;
meineGleitkommavariable = 3.14;
```

Sprechweise:
"Variable `b` ergibt sich zu `true`" oder "Variable `b` wird `true` zugewiesen".

Variablen in Java müssen vor Verwendung deklariert und initialisiert werden. Andernfalls meldet der Compiler einen Übersetzungfehler.

4 Programmbeispiel: Schaltjahre

Betrachten wir zunächst ein Programm, dass die bisher vorgestellten Konzept an einem praktischen Beispiel erläutert: Wir wollen Programm schreiben, das für eine eingegebene Jahreszahl bestimmt, ob es sich um ein Schaltjahr handelt.
Aus Wikipedia-Artikel *Schaltjahr*:

- Die durch 4 ganzzahlig teilbaren Jahre sind Schaltjahre. [...]
- Die durch 100 ganzzahlig teilbaren Jahre (z.B. 1700, 1800, 1900, 2100 und 2200) sind keine Schaltjahre. [...]
- Schließlich sind die ganzzahlig durch 400 teilbaren Jahre doch Schaltjahre. Damit sind 1600, 2000, 2400, ... jeweils wieder Schaltjahre. [...]

Eine mögliche Implementierung in Java ist diese hier (`Schaltjahr.java`):

```
public class Schaltjahre {
  public static void main(String[] args) {
    String eingabe;
    int jahr;
    boolean istSchaltjahr;

    eingabe = args[0];
    jahr = Integer.parseInt(eingabe);

    istSchaltjahr = (jahr % 4 == 0) && (jahr % 100 != 0);
    istSchaltjahr = istSchaltjahr || (jahr % 400 == 0);

    System.out.println(istSchaltjahr);
  }
}
```

4.1 Eingabe/Ausgabe von Daten

In unseren Beispielen verwenden wir zunächst die Eingabe auf Kommandozeilenebene bei Programmstart. Das erste Argument ist in der `main`-Methode unter `args[0]` verfügbar; das zweite Argument unter `args[1]` verfügbar, etc. Da die Argumente immer als `String`-Werte übergeben werden, müssen sie evtl. in einen anderen Datentyp umgewandelt werden (⇒ siehe Abschnitt 4.2).

Beispiel Compilieren und Ausführen von `Schaltjahr` mit Parameter:

```
> javac Schaltjahr.java
> java Schaltjahr 2004
true
> java Schaltjahr 1900
false
```

Fehlt der Parameter beim Aufruf, wird ein Fehler ausgegeben!

4.2 Typumwandlungen

Bisweilen muss man Werte eines Types in Werte eines anderen Typs umwandeln (sogenannter ***Cast***). So wurde im Programmbeispiel zur Ermittlung von Schaltjahren in Abschnitt 4 das Kommandozeilenargument `args[0]`, das vom Typ `String` ist, in einen Integer umgewandelt.
Zur Umwandlung von Strings in numerische Werte gibt es spezielle Bibliotheksmethoden:

`int Integer.parseInt(String s)`	Wandelt `s` in `int`-Wert um
`double Double.parseDouble(String s)`	Wandelt `s` in `double`-Wert um
`boolean Boolean.parseBoolean(String s)`	Wandelt `s` in `boolean`-Wert um

Expliziter Typ-Cast Java hat auch Typumwandlungen für primitive Datentypen vorgesehen. Dieser muss explizit im Programm angegeben werden, in dem der Typname in Klammern einem Ausdruck vorangestellt wird. Dabei muss man verstärkt auf die Klammerung achten, da Casts stärker binden als die arithmetischen und logischen Operatoren.
Achtung: Dabei dieser Art von Typumwandlung kann es zu Informationsverlust kommen. *Beispiel:* `(int)2.71828` ergibt den `int`-Wert `2`. Der Nachkomma-Teil wird dabei "abgeschnitten".

Alternativ kann man Runden mittels der Bibliotheksfunktion `long Math.round(double d)` und explizitem Cast von `long` auf `int`.
Beispiel: `(int) Math.round(2.71828)` liefert `3`.

Automatische Promotion von Zahlen Daten von primitiven numerischen Typen können in einem Kontext verwendet werden, wenn in diesem ein Wert eines Datentyps verwendet wird, der einen größeren Wertebereich abdeckt. Dabei ensteht **kein** Informationsverlust!

- *Beispiel:* Bei der Auswertung von `4.0 * 3` wird `3` zunächst automatisch in einen `double`-Wert umgewandelt; dannach wird die Multiplikation auf `double` durchgeführt.
- *Beispiel:* Bei der Auswertung von `"Ocean's " + 11` wird `11` zunächst automatisch in einen `String`-Wert umgewandelt, dannach wird die Konkatenation auf `String` durchgeführt