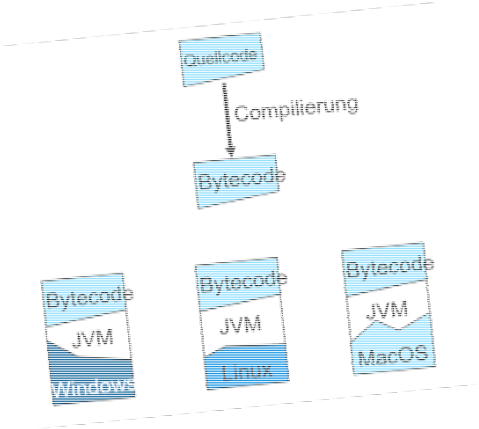
Vorlesung Programmieren
Thema 3: Variablen und Ausdrücke

Olaf Herden
Fakultät Technik
Studiengang Informatik




```
int i = (int) (x*y) % 2;
```

```
System.out.println("Hallo Welt!");
```

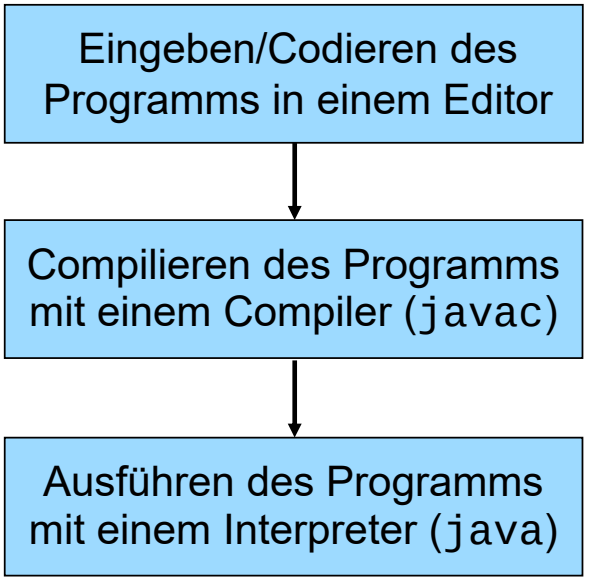
Stand: 11/2023

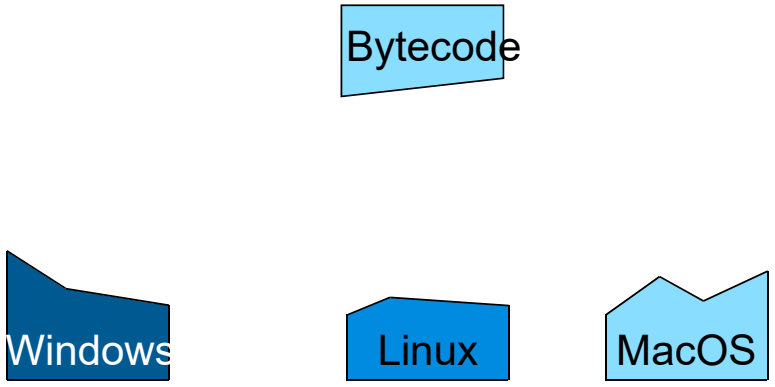
Gliederung

- Grundlagen
- Variablen
- Ausdrücken

Erstellen und Ausführen Java-Programm (I)

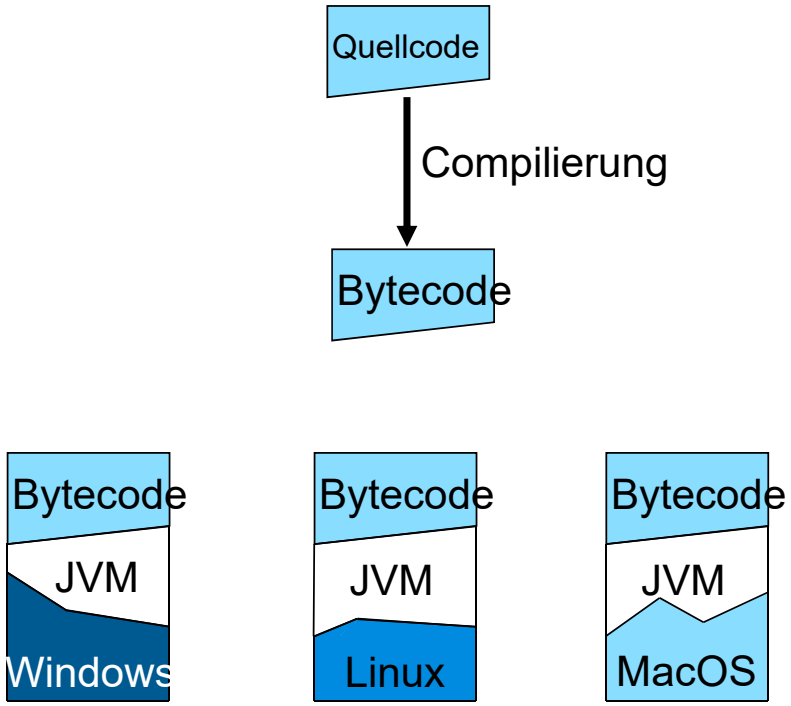
- Compiler erzeugt sog. Bytecode
- Dieser ist Plattform-unabhängig
- Bytecode wird zur Laufzeit interpretiert
- Der Interpreter heißt JVM (Java Virtual Machine)
- Dieser ist Plattform-abhängig





Erstellen und Ausführen Java-Programm (II)

- .



Erstellen und Ausführen Java-Programm (II)

- Beispielprogramm, das einfachen Text ausgibt:

```
(1) public class Beispiel{
(2)   public static void main(String[] args){
(3)     System.out.println("Beispiel eines Java-Programms!");
(4)   }
(5) }
```

- Datei muss den Namen der Klasse (hier: Beispiel) und die Endung `.java` tragen
- Übersetzung wird gestartet mit: `javac Beispiel.java`
- Erzeugte Bytecode-Datei hat den Namen `Beispiel.class`
- Ausführung wird gestartet mit `java Beispiel`

Ausgabe mit `printf` (I)

- Neben `System.out.println` steht Funktion `System.out.printf` zur Verfügung (f = formatted)
- Funktion bekommt mehrere Argumente:
  - Im ersten Argument das %-Zeichen als Platzhalter auftreten, das durch die weiteren Argumente ersetzt wird
  - Hinter dem %-Zeichen muss der Datentyp (`s`,`f`,`d`, ...) angegeben werden
  - Optional kann die Breite der Ausgabe angegeben werden
  - In der Zeichenkette können Steuerzeichen angegeben werden:
    - `\n` für Zeilenumbruch
    - `\t` für Tabulator
- Anmerkung:
  - Datentypen werden später eingeführt
  - Ausführliche Darstellung aller möglichen Optionen: siehe Doku

Ausgabe mit `printf` (II)

- Beispiel:

```
( 1) public class Beispiel2{
( 2)   public static void main(String[] args){
( 3)
( 4)     System.out.printf
( 5)             ("Das ist ein %s Programm!\n","tolles");
( 6)     System.out.printf
( 7)             ("Das ist die Zahl %f!\n",3.3815268368);
( 8)     System.out.printf
( 9)             ("Das ist die Zahl %4.3f formatiert auf 3
(10)                 Nachkommastellen!\n",3.3815268368);
(11)   }
(12) }
```

führt zur Ausgabe

```
Das ist ein tolles Programm!
Das ist die Zahl 3,381527!
Das ist die Zahl 3,382 formatiert auf 3 Nachkommastellen!
```

Unicode-Zeichensatz

- Jede Programmiersprache unterstützt bestimmten Zeichensatz
- Herkömmlich: ASCII, 7 bzw. 8 Bit, 256 Zeichen darstellbar
- Java unterstützt UNICODE:
  - Basiert auf 16 Bit ⇒ 65536 Zeichen darstellbar
  - Alle relevanten Alphabete (lateinisch, kyrillisch, hebräisch, ...) darstellbar
  - Kompatibilität zu ASCII: Erste 256 UNICODE-Zeichen sind ASCII-Zeichen
  - Darstellung eines Zeichens mittels `\uxxxx` (Unicode-Escape-Sequenz), wobei `xxxx` der hexadezimale Code des Zeichens ist
  - Beispiel: „A“ hat den UNICODE 65, 65 ist hexadezimal 41 ⇒ UNICODE-Darstellung von „A“ ist `\u0041`
  - In Zeichenketten dürfen Unicode-Escape-Sequenzen mit „normalen“ Zeichen kombiniert werden, z.B. Anfang entspricht \u0041nfang

Kommentare (I)

- Zur Dokumentation müssen in den Quellcode Kommentare eingebaut werden
- In Java gibt es drei Arten von Kommentaren:
  - Standard-, Block- oder Bereichskommentar:
    - Beginnt mit `/*` und endet mit `*/`
    - Schachtelungen sind nicht möglich
  - Zeilenkommentar: Beginnt mit `//` und endet mit dem Zeilenende
  - Dokumentationskommentar:
    - Beginnt mit `/**` und endet mit `*/`
    - Generierung von HTML-Dokumentationen mit Werkzeug `javadoc` möglich (detaillierte Vorstellung später)

Kommentare (II)

- Beispiel Bereichskommentar:

```
(1) public class Beispiel_Bereichskommentar{
(2)
(3)   public static void main(String[] args){
(4)      /* Mit dem folgenden
(5)         Befehl wird der
(6)         angegebene Text ausgegeben */
(7)      System.out.println("Beispiel Bereichskommentar!");
(8)   }
(9) }
```

Kommentare (III)

- Beispiel Dokumentationskommentar:

```
( 1) /** Beispiel für ein Java-Programm
( 2)     Gibt den angegebenen Text aus
( 3)     Dies ist ein Dokumentationskommentar, d.h. mit dem
( 4)     Werkzeug javadoc erzeugte HTML-Dokumentation
( 5)     enthält diesen Kommentar.
( 6) */
( 7) public class Beispiel_Dokukommentar{
( 8)
( 9)   public static void main(String[] args){
(10)      System.out.println("Beispiel Dokumentationskommentar!");
(11)   }
(12) }
```

Token (I)

- Token: Elementare Objekte einer Programmiersprache
- In Java gibt es fünf Arten von Token:
  - Schlüsselwörter: Essentielle Bestandteile der Sprachdefinition, z.B. `public`, `if`, `while`, …
  - Bezeichner: Namen für Klassen, Methoden, Variablen etc.
  - Literale: Werte, die einer Variablen zugewiesen werden können, z.B. `true`, `15`, `„Hallo Welt!“`, …
  - Trennzeichen: Dienen dem Trennen bzw. Zusammenfassen von Codeteilen, z.B. `( ) { } ; ,` …
  - Operatoren: Legen eine Operation mit Variablen bzw. Konstanten fest, z.B. `+`, `-`, `<`, `>=`, ...
- Bemerkungen:
  - Schlüsselwörter werden auch als reservierte Wörter bezeichnet
  - Operatoren und Trennzeichen werden manchmal auch als Symbole bezeichnet

Token (II)

- Token werden wie folgt voneinander getrennt:
  - Leerzeichen (Blank)
  - Tabulator
  - Zeilenende
  - Zeilenvorschub
  - Seitenvorschub

- Beim Auswerten durch den Compiler wird die Strategie des „gefräßigen“ Tokenizers angewendet, d.h. es werden möglichst viele Zeichen zusammengefasst
- Beispiel:
  - `main` wird als Schlüsselwort erkannt und nicht als einzelne Variable `m`, `a`, `i`, `n`

Token (III): Java-Schlüsselworte

- Dürfen nicht als Bezeichner verwendet werden

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

Token (IV): Bezeichner

- Benennung von festgelegten Einheiten:
  - Variablennamen
  - Methodennamen
  - Klassennamen
  - …
- Konventionen für Bezeichner:
  - Erstes Zeichen: Buchstabe, Unterstrich oder Dollarzeichen
  - Ab dem zweiten Zeichen: Zusätzlich Ziffern und beliebige Unicodezeichen möglich
- Beispiele:
  - `berechne`, `_berechne`, `berechne23` sind korrekte Bezeichner
  - `23berechne`, `?berechne` sind hingegen inkorrekt

Übersicht

- Grundlagen
- Variablen
- Ausdrücken

Variablen und Ausdrücke

- Problemstellung:
  - Gegeben: Startwert und Endwert
  - Gesucht: Summe ganzer Zahlen zwischen diesen Werten
  - Beispiel:
    - Gegeben: 14 und 18
    - Summe „dazwischen“: 15 + 16 + 17 = 48
- Dazu notwendig:
  - „Gedächtnis“: Speicher, in dem der Wert der Summe abgelegt wird
  - Operation, in der die Addition durchgeführt wird
- „Gedächtnis“ wird durch Variablen gebildet
- Operationen bilden zusammen mit Variablen und Konstanten Ausdrücke

Variablen (I)

- Variable besitzt folgende Eigenschaften:
  - Reservierung von Speicherplatz (Variablendefinition)
  - Festlegung eines Namens (Variablendeklaration)
  - Festlegung eines Datentyps
  - Initialisierung
- Syntax:

```
<Variablendefinition> ::= <Datentyp>
                          <Bezeichner> ["=" <Ausdruck>]
                           {"," <Bezeichner> ["=" <Ausdruck>]}
                          ";"
```

- Beispiele:

```
(1) int i;
(2) int j=0;
(3) boolean a, b, c;
(4) boolean a=true, b, c=true;
(5) int x=y=5;
```

Nicht verboten, aber unschön!

Variablen (II): Datentypen (I)

- Warum Datentypen?
  - Daten haben unterschiedliche Eigenschaften (Zahlen, Zeichenketten, ...)
  - Reservierung von passendem Speicherplatz
  - Überprüfung sinnvoller Operationen
- Datentypen in Java:

```
boolean     true/false
char        16-Bit-Unicode
byte        8-Bit-Integer, vorzeichenbehaftetes 2er-Komplement
short       16-Bit-Integer, vorzeichenbehaftetes 2er-Komplement
int         32-Bit-Integer, vorzeichenbehaftetes 2er-Komplement
long        64-Bit-Integer, vorzeichenbehaftetes 2er-Komplement
float       32-Bit-Gleitkommazahl (IEEE 754-1985)
double      64-Bit-Gleitkommazahl (IEEE 754-1985)
(String     Zeichenketten (Folgen von 16-Bit-Unicode-Zeichen))
```

- Datentypen haben Standardwert, einen kleinst- und größtmöglichen Wert

Variablen (III): Datentypen (II)

Typ	Standardwert	Kleinster Wert	Größter Wert
boolean	false	---	---
char	`\u0000`	`\u0000`	`\uFFFF`
byte	0	-128	127
short	0	-32768	32767
int	0	-2147483648	2147483647
long	0	-9223372036854775808	9223372036854775807
float	0	+-3.40282347E+38	+-1.40239846E-45
double	0	+-1.79769313486231570E+308	+-4.94065645841246544E-324

Initialisierung (I)

- Implizit durch den Standardwert (Default)
- Explizit durch Angabe von Literalen oder Ausdrücken
- Datentyp `boolean`:
  - Angabe von `true` bzw. `false`
  - Bsp.: `boolean b = true;`
- Datentyp `char`:
  - Angabe des Zeichens in einfachen Hochkommata
  - Bsp.: `char c = 'A';`
  - Sonderzeichen werden als sog. Escape-Sequenz dargestellt, z.B. '\n'
  - Bsp.: `char c = '\n';`

Initialisierung (II)

- Datentyp `int`:
  - Angabe der Zahl in dezimaler, oktaler oder hexadezimaler Notation
  - Beispiele:

```
int i=55; //Dezimal, keine weitere Angabe
int i=067; //Oktal, Angabe von führender Null
int i=0x37; //Hexadezimal, Angabe von 0x
```

- Anm.: Für Binärzahlen gibt es in Java keine Notation!
- Datentyp `long`:
  - Angabe der Zahl mit nachgestelltem l oder L
  - Bsp.: `long l=55L;`

Initialisierung (III)

- Datentyp `double`:
  - Angabe der Dezimalzahl mit Punkt, Exponent als e oder E
  - Beispiele:

```
double i=55.;       //Dezimalzahl mit Punkt
double i=5.5e1;     //Dezimalzahl in Exponentialschreibweise
double i=5500.E-2; //Dezimalzahl in Exponentialschreibweise
```

- Datentyp `float`:
  - Wie bei double mit nachgestelltem f oder F
  - Beispiele:

```
float i=55.f;       // Floatzahl mit Punkt
float i=5.5e1f;     // Floatzahl in Exponentialschreibweise
float i=5500.E-2f; // Floatzahl in Exponentialschreibweise
```

Initialisierung (IV)

- Datentyp `String`:
  - Angabe der Zeichenkette in Anführungszeichen
  - Beispiele:

```
String s = “Hallo Horb“;
String s = “Hier bin ich!“;
```

- Anmerkung:
  - Zeichenketten sind hier der Vollständigkeit halber erwähnt
  - Ansonsten: String ist technisch etwas anderes als double, int, …
  - Später mehr

Konstanten

- Konstante:
  - Einmalige Wertzuweisung
  - Keine Veränderung mehr möglich
- Festlegung wie Variablen (Datentyp, Name, Wert)
- Zusätzliche Schlüsselwort `final` voranstellen
- Konvention: Name in Großstbuchstaben
- Beispiele:

```
final int MAXIMUM = 100;
final float PI = 3.1415;
```

Felder (I)

- Motivation: Verwaltung einer Menge von Werten gleichen Typs
- Datenstruktur Feld (Array)
- Anlegen Felder in Java:

```
<Datentyp> [] = new <Datentyp> [<N>];
```

- Beispiel:

```
int[] feld = new int[10];
```

- Folgende Struktur wird angelegt:

0	1	2	3	4	5	6	7	8	9	← Index
0	0	0	0	0	0	0	0	0	0	← Werte

Felder (II)

0	1	2	3	4	5	6	7	8	9	← Index
0	0	0	0	0	0	0	0	0	0	← Werte

- Zugriff auf Feldwerte über Index, z.B.

```
feld[3] = 7;
feld[9] = 6;
```

führt zu

0	1	2	3	4	5	6	7	8	9
0	0	0	7	0	0	0	0	0	6

- Anmerkungen:
  - Operator new wird später ausführlich behandelt
  - Verwendung ungültiger Index => Fehler (Array out of Bounds Exception)
  - Verwaltung von Elementen gleichen Typs auch mit anderen Datenstrukturen (z.B. Liste) möglich

Felder (III)

- Alternativ (zu new) implizites Anlegen, z.B.

```
int[] feld_2 = {2,4,6,8,11};
```

führt zu

0	1	2	3	4
2	4	6	8	11

- Ermittlung der Feldlänge: `<Feldname>.length`, z.B.

```
System.out.println(feld_2.length);
```

führt zu Ausgabe

```
5
```

Mehrdimensionale Felder (I)

- Felder können auch mehrdimensional sein
- Syntax: Mehrfache Verwendung von []
- Beispiel: `int[][] feld = new int[6][3];`
- Angelegt wird folgende Struktur:

Index 1. Dimension (Zeilen) →

	0	1	2
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0
5	0	0	0

← Index 2. Dimension (Spalten)

← Werte

Mehrdimensionale Felder (II)

- Zugriff auf Werte durch Angabe mehrerer Indexpositionen
- Beispiel:

```
feld[3][0] = 7; feld [2][2] = 6;
```

führt zu

	0	1	2
0	0	0	0
1	0	0	0
2	0	0	6
3	7	0	0
4	0	0	0
5	0	0	0

Mehrdimensionale Felder (III)

- Auch hier: Implizites Definieren möglich
- Beispiel:

```
int[][] feld = {{,,3,7,,},{1,3,,3,2,},{,0,0,1,0,1}};
```

führt zu

	0	1	2
0	0	1	0
1	0	3	0
2	3	0	0
3	7	3	1
4	0	2	0
5	0	0	1

Mehrdimensionale Felder (IV)

- Feldgröße wird ermittelt durch Voranstellen von [] vor `length`, d.h.
  - `feld.length` für die Größe der ersten Dimension
  - `feld[].length` für die Größe der zweiten Dimension
  - `feld[][].length` für die Größe der dritten Dimension
  - usw.
- Beispiel:

```
int[] my_Array_1 = new int[5][4][2];
int[] my_Array_2 = new int[25][36][49];
System.out.println(my_Array_1.length);
System.out.println(my_Array_2[][].length);
```

führt zu Ausgabe

```
5
49
```

Aufzählungen

- Aufzählungen (Enumerationen) sind (meistens kleine,) endliche Wertebereiche
- Beispiele:
  - Menge von Farben
  - Menge der Wochentage
- Definition mit Schlüsselwort `enum`
- Beispiele:

```
public enum Ampelfarbe{rot,gelb,grün};
public enum Wochentag{Mo,Di,Mi,Do,Fr,Sa,So};
```

- Nun: Anlegen von Variablen dieser neuen Typen
- Beispiele:

```
Ampelfarbe a = Ampelfarbe.rot;
Wochentag w = Wochentag.Fr;
```

- Anm.: Aufzählungen und `public` werden später (im Abschnitt Klassen) vertieft

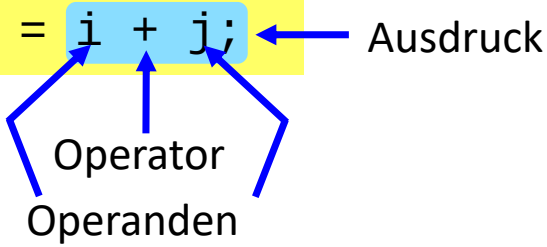
Übersicht

- Grundlagen
- Variablen
- Ausdrücke

Ausdrücke

- Dienen der Berechnung von neuen Werten aus bestehenden Werten
- Werden aus Operanden und Operatoren gebildet
- Beispiel:

```
(1) int i=1;
(2) int j=2;
(3) int k;
(4) k = i + j;
```



- Dabei:
  - Verträglichkeit Operanden, z.B. keine Zeichenkette mit ganzer Zahl addieren
  - Nicht alle Operatoren für alle Datentypen, z.B. keine Division von Zeichenketten

Syntax

- Bilden von Ausdrücken:

```
<Ausdruck> ::= <Literal>
             | <Variablenname>
             | <Funktionsaufruf>
             | "(" <Ausdruck> ")"
             | <unär-Op> <Ausdruck>
             | <Ausdruck> <binär-Op> <Ausdruck>
             | <Ausdruck> <ternär-Op1>
               <Ausdruck> <ternär-Op2>
               <Ausdruck>
```

- `<unär-Op>`: Unärer (einstelliger) Operator
- `<binär-Op>`: Binärer (zweistelliger) Operator
- `<ternär-Op>`: Ternärer (dreistelliger) Operator

Übersicht Operatoren

-

Bezeichnung	Operatoren
Ganzzahl-Arithmetik (int, short, long)	`+, -, *, /, %`
Gleitkomma-Arithmetik (float, double)	`+, -, *, /`
Boolesche Arithmetik (boolean)	`&&, ||, !`
Vergleichsoperatoren	`==, !=, <, >, <=, >=`
Zuweisungsoperatoren	`=, +=, -=, *=, /=, %=`
Inkrement-Operator	`++`
Dekrement-Operator	`--`
Bit-Operatoren	`<<, >>, >>>, &, |, ~, ^, ...`
Spezielle Operatoren	`?:, (type)`

Ganzzahl- und Gleitkommaarithmetik

- Ganzzahl-Arithmetik (Typen `int`, `short` und `long`):
  - +/-/* realisieren die üblichen Grundrechenarten
  - / bildet die ganzzahlige Division
    Bsp.: `9 / 5 = 1` bzw. `26 / 3 = 8`
  - % realisiert die Restbildung (Modulo-Operation)
    Bsp.: `9 % 5 = 4` bzw. `26 % 3 = 2`
- Gleitkomma-Arithmetik (Typen `float` und `double`):
  - +, -, * und / realisieren die üblichen Grundrechenarten
  - Achtung: Bei der Division sind Rundungsfehler möglich!

Boolesche Operatoren (I)

- Vier Operatoren:
  - !: Negation (Logisches NICHT)
  - &&: Konjunktion (Logisches UND)
  - ||: Disjunktion (Logisches ODER)
  - ^: Exklusive Disjunktion (Logisches ENTWEDER ODER)
- Beispiel:

Boolesche Operatoren (II)

- Beispiel:

```
boolean a = true, b = true, c = false;
boolean res;
res = !a;
System.out.println("Nicht a = " + res);
res = a&&b;
System.out.println("a UND c = " + res);
res = a || b;
System.out.println("a ODER b = " + res);
res = a^b;
System.out.println("a XOR b = " + res);
```

führt zur Ausgabe

```
Nicht a = false
a UND c = true
a ODER b = true
a XOR b = false
```

Vergleichsoperatoren

- Operanden können Ganzzahl-, Gleitkommatyp oder char sein
- Resultat ist immer vom Typ `boolean`
- Operatoren:
  - == prüft auf Gleichheit
  - != prüft auf Ungleichheit
  - <, <=, >, >= sind die üblichen Vergleichsoperatoren

Zuweisungsoperator (I)

- Syntax:

  `<Zuweisung> ::= <Variablenname> “=“ <Ausdruck> “;“`

- Beispiele:

```
(1) int i = 0, j = -55; // Initialisierung
(2) i = 69;             // Zuweisung
(3) i = i - 7;
(4) j = j + i*3;
```

- Anmerkung: Keine mathematische Gleichheit!

Zuweisungsoperator (I)

- Notation:

Kurzform	Langform
`i++, ++i`	`i = i + 1`
`i--, --i`	`i = i – 1`
`i += <expr>`	`i = i + (<expr>)`
`i -= <expr>`	`i = i - (<expr>)`
`i *= <expr>`	`i = i * (<expr>)`
`i /= <expr>`	`i = i / (<expr>)`
`i %= <expr>`	`i = i % (<expr>)`

Pre/Post-In/Dekrementoperator

- Steht ++ bzw. - - hinter dem Operanden, wird dieser nach der Berechnung verändert
- Steht ++ bzw. - - vor dem Operanden, wird dieser vor der Berechnung verändert
- Beispiel:

```
(1) int i = 5;
(2) int j = 5;
(3) int x, y;
(4) x = 5 * i++;
(5) y = 5 * ++j;
```

Resultat: x = 25, y = 30

- Zusammenfassung:

	Pre	Post
Inkrement	`++i`	`i++`
Dekrement	`--i`	`i--`

Operator ?

- Operator <op_1> **?** <op_2> **:** <op_3> verlangt drei Operanden:
  - <op_1> muss vom Typ boolean sein
  - <op_2> und <op_3> müssen typkompatibel sein
  - Semantik: Ist der Ausdruck <op_1> wahr, dann wird <op_2> ausgeführt, ansonsten <op_3>
- Beispiel:

```
(1) int i = 5;
(2) int j = 7;
(3) int k = i<j ? i*i : j*j;
(4) int l = i>j ? i*i : j*j;
```

  Resultat: k = 25, l = 49
- Zwar sehr kompakte Darstellung, aber Code schlecht lesbar!
- Daher möglichst nicht verwenden, besser `if`-Anweisung (siehe nächster Abschnitt) benutzen

Typumwandlungen (I)

- Konvertierung zwischen verschiedenen Datentypen:
  - Implizit: Programmierer/in braucht nichts zu machen
  - Explizit: Programmierer/in gibt Umwandlung an
- Implizite Umwandlungen nur von Typ mit kleinerem Wertebereich in Typ mit größerem Wertebereich
- Folgende implizite Umwandlungen:

Ausgangstyp	Automatische Umwandlung nach
`double`	`---`
`float`	`double`
`long`	`double, float`
`int`	`double, float, long`
`char`	`double, float, long, int`
`short`	`double, float, long, int`
`byte`	`double, float, long, int, short`
`boolean`	`---`

Implizite Typumwandlungen

- Beispiele:

```
(1) short i;
(2) int j = i;
(3) long k = j;
(4) float pi = 3.1415;
(5) double d = 2*pi;
(6) char c = ‘s‘;
(7) int x = c;
```

Resultat: x = 155 (ASCII-Code des Zeichens)

Explizite Typumwandlungen

- Programmierer/in gibt Umwandlung von Typen an
- Wird als Casting bezeichnet
- Syntax: `<Cast> ::= “(“ <Typ> “)“`
- Beispiel:

```
float pi = 3.1415;
int i = (int) pi;
```

- Casting wird durchgeführt „so gut es eben geht“, d.h. im Beispiel ist das Resultat i=3 (Informationsverlust)
- Nicht alle Typumwandlungen erlaubt, z.B. keine Umwandlung Boolescher Ausdruck in Ganzzahl

```
boolean a = false;
int i = (int) a;
```

Resultat: Fehlermeldung

Operatorvorrang (I)

- Bei Verknüpfung mehrerer Operatoren: Definierte Reihenfolge des Vorrangs (Präzedenz)
- In Java: Absteigende Reihenfolge gemäß Tabelle auf folgender Folie
- Durch Klammerungen Veränderung der Auswertungsreihenfolge
- Anm.: Operatoren `[]`, `.`, `~`, `instanceof` und `new` werden später eingeführt

Operatorvorrang (II)

- .

Postfix-Operatoren	`[], ., exp++, exp--`
Unäre Operatoren	`++exp, --exp, +exp, -exp, ~, !`
Erzeugung / Typumwandlung	`new, (type)`
Multiplikationsoperatoren	`*, /, %`
Additionsoperatoren	`+, -`
Shift-Operatoren	`<<, >>, >>>`
Vergleichsoperatoren	`<, >, <=, >=, instanceof`
Gleichheitsoperatoren	`==, !=`
Bitweises UND	`&`
Bitweises exklusives ODER	`^`
Bitweises ODER	`|`
Logisches UND	`&&`
Logisches ODER	`||`
Bedingungsoperator	`? :`
Zuweisungsoperatoren	`=, +=, *=, ...`

Assoziativität

- Abarbeitung von Operatoren auf gleicher Präzedenz-Stufe
- Zwei prinzipielle Möglichkeiten:
  - Links-Assoziativität: Von Links nach Rechts auswerten
  - Rechts-Assoziativität: Von Rechts nach Links auswerten
- Regeln:
  - Alle einstelligen Operatoren sind rechts-assoziativ
  - Alle zweistelligen Operatoren mit Ausnahme der Zuweisungen sind links-assoziativ
  - Zuweisungsoperatoren sind rechts-assoziativ
  - Der ternäre Operator ?: ist rechts-assoziativ
- Beispiele:

`a = 8 * 9 / 5;` entspricht `(8*9)/5`, weil die zweistelligen Operatoren `*` und `/` links-assoziativ sind

`a = b = c + 3;` entspricht `a = (b = (c + 3));`, weil die Zuweisung rechts-assoziativ ist

Zurück zum Ausgangsproblem

- Ausgehend von Startwert und Endwert Bildung der Summe ganzer Zahlen zwischen diesen beiden Werten
- Gelernte Sprachmittel:
  - Variablen zum Ablegen von Start- und Endwert bzw. eventuell Zwischenwerten
  - Operatoren zum Aufsummieren
  - Bedingungen zum Überprüfen, wie weit schon gezählt wurde
- Was noch fehlt: Kontrollstrukturen zum kontrollierten Hochzählen vom Start- bis zum Endwert
- Einführung im nächsten Abschnitt

Zusammenfassung (I)

- Grundlagen:
  - Java-Programme (Übersetzung, Ausführung, Bytecode, JVM)
  - Zeichensätze (ASCII, Unicode)
  - Kommentare
  - Token
  - Schlüsselworte
  - Bezeichner
- Variablen:
  - Motivation
  - Datentypen (boolean, char, String, byte, short, int, long, float, double)
  - Initialisierung
  - Konstanten
  - Felder
  - Aufzählungen

Zusammenfassung (II)

- Ausdrücke:
  - Motivation
  - Definition
  - Operatoren
  - Präzedenz
  - Assoziativität

Aufgaben (I)

- **Aufgabe 1**

  Erstellen Sie ein Java-Programm, das den Text „Hallo Welt“ auf der Textkonsole ausgibt.

  Übersetzen Sie das Programm und führen Sie es aus.

- **Aufgabe 2**

  Erstellen Sie ein Java-Programm, in dem drei Integer-Variablen a1, a2 und a3 deklariert und mit Datenwerten belegt werden. Ihr Programm soll folgende Forderungen erfüllen:

  a) Es soll das arithmetische Mittel dieser Werte berechnen und ausgeben.
  b) Es soll ausgeben, ob a1 < a2 < a3 gilt.
  c) Es soll ausgeben, ob a1 ein ganzzahliges Vielfaches von a2 ist.
  d) Es soll a3 unter Verwendung der Bitmanipulations-Operatoren und des Additions-Operators invertieren und das Ergebnis ausgeben.

Aufgaben (II)

e) Es soll a3 mit dem größten positiven Integer-Wert belegen und diesen Wert ausgeben.
f) Welchen Wert erhalten Sie, wenn sie diesen größten positiven Integer-Wert um 1 erhöhen?
g) a2 soll mit 0xFFFFFFFF belegt werden und dieser Wert mittels Bitmanipulationsoperatoren in den in e) ermittelten größten positiven Integer-Wert umgewandelt werden.

Anmerkung: if-Anweisungen o.ä. sind in der gesamten Aufgabe 2 verboten. Die kennen wir ja noch nicht!

Aufgaben (III)

- **Aufgabe 3**

  Erstellen Sie ein Java-Programm, in dem drei Integer- und eine Boolesche Variable dekla riert werden. Weisen Sie den Integer-Variablen Datenwerte zu und lassen Sie die Funktion

$$g : \text{int} \times \text{int} \times \text{int} \rightarrow boolean$$

$$g(a,b,c) \mapsto \begin{cases} true & falls\, a^2 + b^2 = c^2 \\ false & sonst \end{cases}$$

  berechnen. Geben Sie das Ergebnis aus!

  Variieren Sie die Werte für die Variablen und untersuchen Sie, ob Ihr Programm stets die gewünschten Ergebnisse liefert.

Aufgaben (IV)

- **Aufgabe 4**

  Gegeben sei das folgende Programmfragment:

```
int i = 5;
int j = 3;
boolean b = false;
```

  Welche Rückgabe (Datentyp und Wert) liefern die nachfolgenden Ausdrücke für die oben genannte Variablenbelegung?

  Welche Datenwert beinhaltet Variable i nach der Ausführung von f) und g)?

  Gehen Sie dabei davon aus, dass i vor jeder dieser Ausführungen den Wert 5 enthält.

  Bitte überlegen Sie diese Antworten erst „auf dem Papier“, bevor Sie es in Java implementieren!

Aufgaben (V)

a) `(!((i<j) && b))`

b) `i/j`

c) `(float) (i/j)`

d) `(float) i/j`

e) `(float) i / (float) j`

f) `((i++ == 5) || (--i == 5))`

g) `((i++ == 5) | (--i == 5))`