
Java™

Wiederholung

- So sieht ein „leeres“ Java-Programm aus

```
public class Programmname {

    public static void main (String[] args) {

        // Hier stehen die Anweisungen

    }

}
```

Wiederholung

- Beispiele für Anweisungen
  - Ausgabe des Textes „Informatik macht Spaß“ auf der Konsole

    ```
    System.out.print("Informatik macht Spaß");
    ```

  - Einlesen einer Zahl vom Benutzer, Speicherung in der Variablen x

    ```
    double x = scanner.nextDouble();
    ```

  - Berechnung der Wurzel von x, Speicherung in der Variablen y

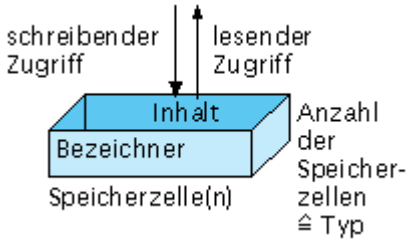
    ```
    double y = Math.sqrt(x);
    ```

Variablen

Variable im Programm

- Grundlegendes Konzept jeder Programmierung
- In der imperativen Programmierung ist der Wert zu jedem Zeitpunkt konkret benennbar
- Komponenten einer Variablen
  - Bezeichner (Name)
  - Adresse (Zugewiesener Ort im Arbeitsspeicher)
  - Wert (Inhalt des Arbeitsspeicher an der Adresse der Variablen)
  - Datentyp
    - Welche Art von Werten kann die Variable annehmen?

Speicherzugriff



- Lesender Zugriff
  - Welchen Wert hat die Variable aktuell?
  - Information wird aus der Speicherzelle ausgelesen
  - Die gespeicherte Information bleibt unverändert
- Schreibender Zugriff
  - Wert der Variablen verändern
  - Information wird in Speicherzelle abgelegt
  - Im Java-Code erfolgt die Zuweisung durch

```
variablenname = wert;
```

Deklaration

- Variable muss zunächst erzeugt werden (Deklaration)
- Datentyp muss festgelegt werden
- Speicherplatz wird entsprechend des Datentyps für die neue Variable bereitgestellt
- Der Variablen muss ein Anfangswert zugewiesen werden (Initialisierung)
- Im Java-Code

```
Datentyp variablenname = initialwert;
```

Variablen in Java

- Elementare Anweisung (mit Semikolon zu beenden)
  - Deklaration einer Variablen (muss initialisiert werden)
    ```
    Datentyp variablenname;
    ```
  - Deklaration und Initialisierung einer Variablen
    ```
    Datentyp variablenname = Wert;
    ```
  - Zuweisung eines Wertes zu einer Variablen
    ```
    variablenname = Ausdruck;
    ```

Variablen in Java

- 1. Beispiel (ganze Zahl)

  - Deklaration einer Variablen (muss initialisiert werden)

```
int jahr;
```

  - Deklaration und Initialisierung einer Variablen

```
int jahr = 2016;
```

  - Zuweisung eines Wertes zu einer Variablen

```
jahr = 2016 + 1;
```

Variablen in Java

- 2. Beispiel (Text)

  - Deklaration einer Variablen (muss initialisiert werden)

```
String vorlesung;
```

  - Deklaration und Initialisierung einer Variablen

```
String vorlesung = "Informatik";
```

  - Zuweisung eines Wertes zu einer Variablen

```
vorlesung = "Informatik WS" + jahr + "/" + (jahr+1);
```

Variablen in Java

- 3. Beispiel (Wahrheitswert)

  - Deklaration einer Variablen (muss initialisiert werden)

```
boolean bestanden;
```

  - Deklaration und Initialisierung einer Variablen

```
boolean bestanden = false;
```

  - Zuweisung eines Wertes zu einer Variablen

```
bestanden = (note <= 4);
```

Variablen in Java

- 4. Beispiel (Komma-Zahl)

  - Deklaration einer Variablen (muss initialisiert werden)
    ```
    double note;
    ```

  - Deklaration und Initialisierung einer Variablen
    ```
    double note = 3.7;
    ```

  - Zuweisung eines Wertes zu einer Variablen
    ```
    note = (2.0 + 3.7)/2;
    ```

Java – Programm

- Was passiert mit den Variablen?
- Schreibender Zugriff

```
public class Test {
    public static void main (String[] args) {
        int a = 42;
        a = 9;
        int b = 0;
        a = 1;
        b = 2;
        a = 3;
        int c = 17;
        b = 4;
    }
}
```

Java – Programm

- Was passiert mit den Variablen?
- Was wird ausgegeben?

```
public class Test {
   public static void main (String[] args) {
      int x = 1;
      int y = 2;
      int z = x+y;
      x = z;
      z = x*y;
      y = z-1;
      z = 2*(y-x);
      x = z/2;
      System.out.println(x+y);
   }
}
```

Java – Programm

- Dieses Programm berechnet den Benzinverbrauch

```
import java.util.Scanner;
public class Benzinverbrauch {
    public static void main (String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Getanktes Benzin (Liter): ");
        double benzin = scanner.nextDouble();
        System.out.print("Gefahrene Strecke (km): ");
        double strecke = scanner.nextDouble();

        double verbrauch = 100 * benzin / strecke;
        System.out.print("Das Fahrzeug verbraucht ");
        System.out.print(verbrauch);
        System.out.println(" Liter je 100 km.");

        scanner.close();
    }
}
```

Konventionen – Variablenname

- Klein schreiben
- Kurz, aber aussagekräftig
- Keine Umlaute oder Sonderzeichen verwenden
- Schlecht
  - Zählvariablefürdieschleife, blub, JavaistBLÖD, ...
- Gut
  - sum, counter, upperValue, lowerValue, money, ...
- Keine Einheiten
  - Schlecht: meter, km, liter, days, dollar
  - Gut: distance, height, size, capacity, force

Konstanten

- Kein schreibender Zugriff nach Initialisierung
- Konvention: Nur Großbuchstaben
- In Java mit Schlüsselwort „final“

```
static final Datentyp KONSTANTENNAME = wert;
```

- Beispiele (in Java bereits vorhanden)
  - `Math.PI`
  - `Integer.MAX_VALUE`
  - `Double.POSITIVE_INFINITY`

Datentypen

Bits und Bytes

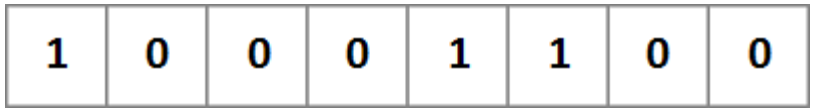
- 1 Bit
  - Kleinste Speicher-Einheit im Rechner
  - z. B. Magnet (+,-)
  - Kann die Werte 0 oder 1 annehmen

- 1 Byte = 8 Bit

Hexadezimale Darstellung

- 4 Bits werden zu einem Symbol zusammengefasst

0000 = 0	1000 = 8
0001 = 1	1001 = 9
0010 = 2	1010 = A
0011 = 3	1011 = B
0100 = 4	1100 = C
0101 = 5	1101 = D
0110 = 6	1110 = E
0111 = 7	1111 = F



8 C

Datentypen in Java

- Primitive Datentypen
  - Boolscher Wahrheitswert
    - boolean (JVM-spezifisch, mind. 1 bit)
  - Unicode-Zeichen (UTF-16)
    - char (16 bit)
  - Ganzzahlen
    - byte (8 bit)
    - short (16 bit)
    - int (32 bit)
    - long (64 bit)
  - Gleitkommazahlen
    - float (32 bit)
    - double (64 bit)

boolean

- Deklaration

```
boolean variable;
```

- Beispiel-Werte

```
variable = false;
```


```
variable = true;
```

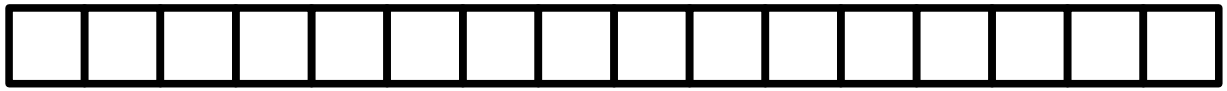
Unicode-Zeichen (UTF-16)

- In Java
  - 16 Bit
  - Jede der 2^16 = 65536 möglichen Belegungen steht für ein bestimmtes Zeichen
- Komplette Tabelle
  - http://unicode-table.com/en/

char

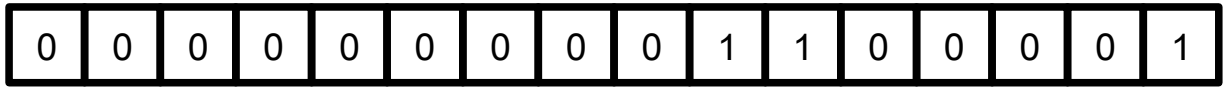
- Deklaration

```
char variable;
```

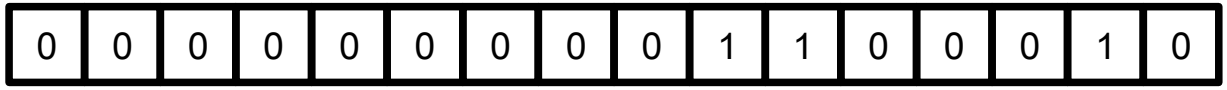


- Beispiel-Werte

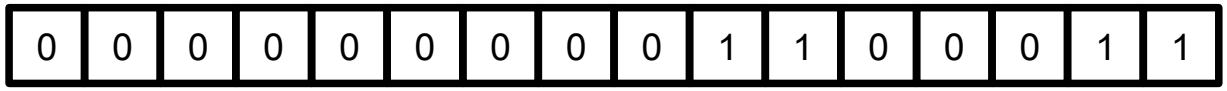
```
variable = 'a';
```



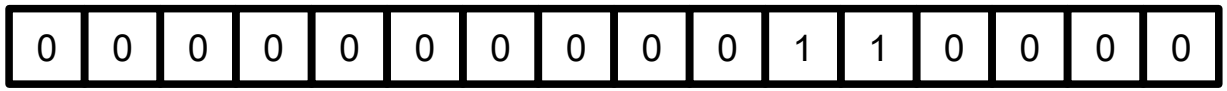

```
variable = 'b';
```



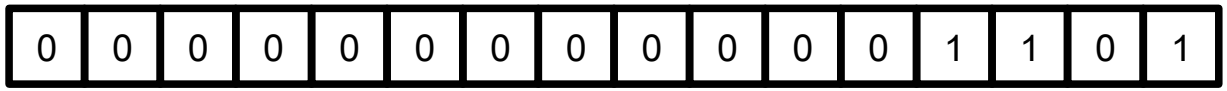

```
variable = 'c';
```




```
variable = '0';
```




```
variable = \n;
```



Ganzzahlen

- N bit
  - 1. bit = Vorzeichen-Bit (vz), entspricht -2^(N-1)
  - 2. bit = 2^(N-2)
  - 3. bit = 2^(N-3)

    …
  - N. bit = 1

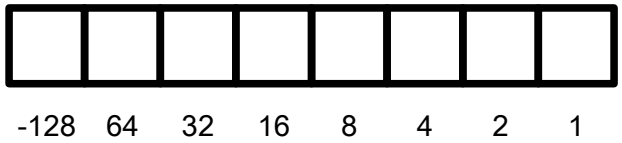
Wert = Summe der gesetzten Bits, wenn vz nicht gesetzt

Wert = Summe der gesetzten Bits - 2^(N-1), wenn vz gesetzt

byte

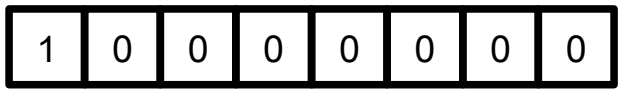
- Deklaration

```
byte variable;
```

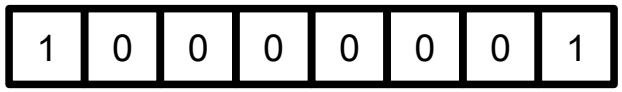


- Beispiel-Werte

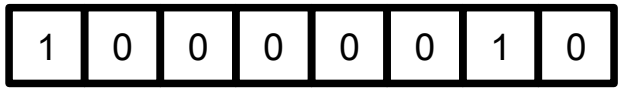
```
variable = -128;
```



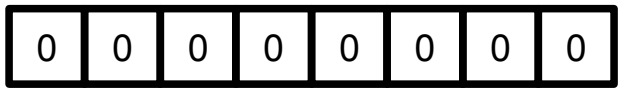

```
variable = -127;
```



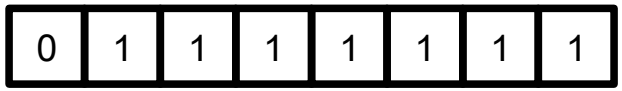

```
variable = -126;
```




```
variable = 0;
```



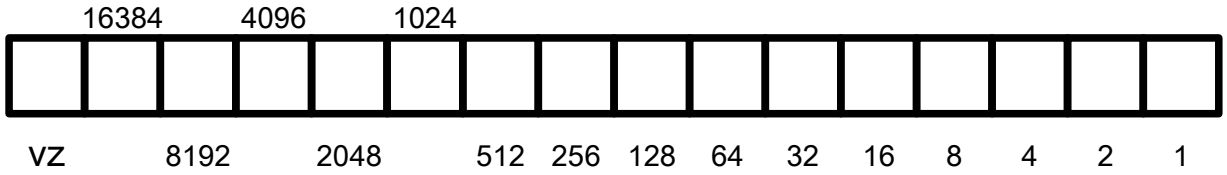

```
variable = 127;
```



short

- Deklaration

```
short variable;
```



- Beispiel-Werte

```
variable = -32768;
```

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

```
variable = -32767;
```

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

```
variable = -32766;
```

1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

```
variable = 0;
```

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

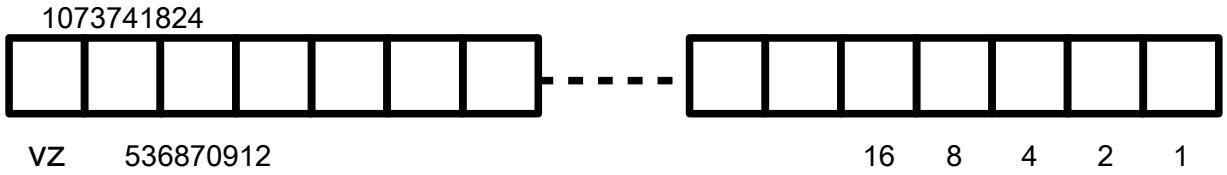
```
variable = 32767;
```

0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

int

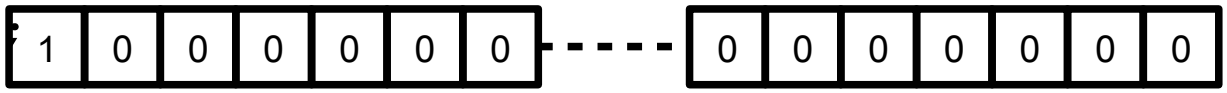
- Deklaration

```
int variable;
```



- Beispiel-Werte

```
variable = -2147483648;
```




```
variable = -2147483647;
```


```
variable = -2147483646;
```

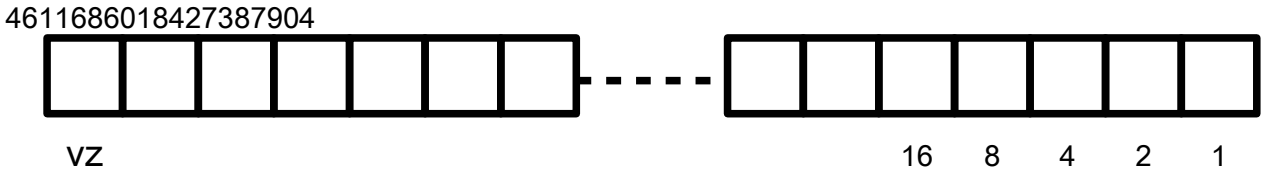

```
variable = 0;
```


```
variable = 2147483647;
```

long

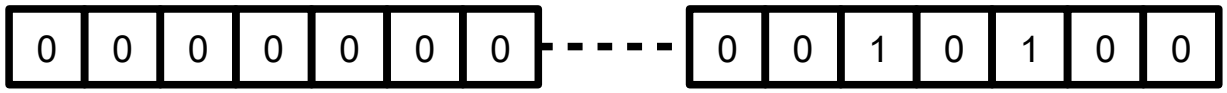
- Deklaration

```
long variable;
```



- Beispiel-Werte

```
variable = 20L;
```




```
variable = 11L;
```


```
variable = -1L;
```


```
variable = -11L;
```

Gleitkommazahlen

- Bits teilen sich auf in
  - Vorzeichen-Bit
    - Ist die Zahl positiv oder negativ?
  - Exponent
    - In welchem Bereich liegen die signifikanten Stellen?
  - Mantisse
    - Das sind die signifikanten stellen

- Bei float
  - 8 bit Exponent – 23 bit Mantisse
- Bei double
  - 11 bit Exponent – 52 bit Mantisse

Gleitkommazahlen

- N Bit Exponent
- M Bit Mantisse

- Berechnung des Wertes
  - Wert = 2^(Exponent + 1 – 2^(N-1)) * (1 + Mantisse/(2^M))

- Beispiel 6.75f

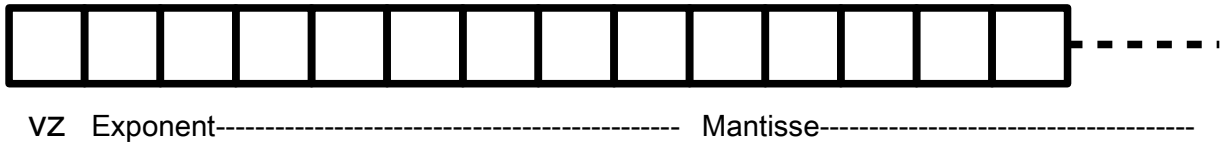
  0 10000001 10110000000000000000000

  \+ 129 2^22 + 2^20 + 2^19

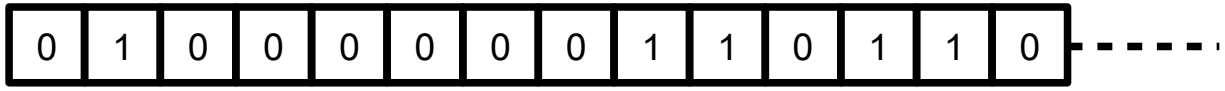
Gleitkommazahlen

- float

```
float variable;
```



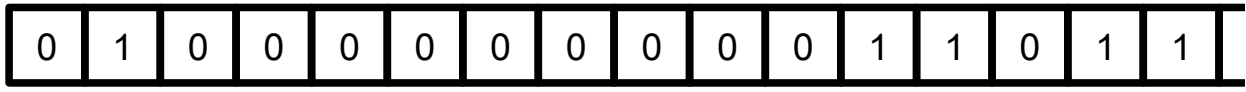

```
variable = 6.75f;
```



- double

```
double variable;
```


```
variable = 6.75;
```



Datentypen in Java

- Primitive Datentypen
  - Boolscher Wahrheitswert
    - boolean (JVM-spezifisch, mind. 1 bit)
  - Unicode-Zeichen (UTF-16)
    - char (16 bit)
  - Ganzzahlen
    - byte (8 bit)
    - short (16 bit)
    - int (32 bit)
    - long (64 bit)
  - Gleitkommazahlen
    - float (32 bit)
    - double (64 bit)

nicht-primitive Datentypen

- String (Zeichenkette)
  - `String variable = "Hallo";`
  - Wir stellen uns mehrere UTF-16 Werte an einander gereiht vor
    - 00000000 01001000 = H
    - 00000000 01100001 = a
    - 00000000 01101100 = l
    - 00000000 01101100 = l
    - 00000000 01101111 = o
- Mehr zu nicht-primitiven Datentypen gegen Ende des Semesters

Literale

- Konstante Ausdrücke,
  um Werte in verschiedenen Datentypen anzugeben
  - boolean (die reservierten Schlüsselworte true und false)

    ```
    true
    false
    ```
  - char (in einfachen Anführungszeichen gesetztes Zeichen oder Sonderzeichen)

    ```
    'x'
    '2'
    \n
    ```
  - String (in normalen Anführungszeichen angegebener Text)

    ```
    "Das ist ein String"
    "22"
    "\n"
    ```

Literale

- Konstante Ausdrücke,
  um Werte in verschiedenen Datentypen anzugeben
  - byte (ganze Zahl mit Y)
    `123y`
  - short (ganze Zahl mit S)
    `123s`
  - int (ganze Zahl)
    `123`
  - long (ganze Zahl mit L)
    `123L`
  - float (Kommazahl mit F)
    `123f`
  - double (Kommazahl mit D)
    `123d`

Casting

- Datentyp verändern
  - Explizit
    - Programmierer schreibt in den Code, dass eine Umwandlung vorgenommen werden soll
    - `(Datentyp) variable`
  - Implizit
    - Rechnung oder Zuweisung macht eine Typumwandlung nötig
    - `3.5 = 2 + 1.5`
      `(double = int + double)`
    - `"WS 2016/17" = "WS " + 2016 + "/" + 17`
      `(String = String + int + String + int)`

Casting

- Diese Variablen akzeptieren implizites Casting
  - boolean – nicht möglich
  - char – nicht möglich
  - byte – nicht möglich
  - short – `byte`
  - int – `byte, short, char`
  - long – `byte, short, int, char`
  - float – `byte, short, int, long, char`
  - double – `byte, short, int, long, float, char`
  - String – `byte, short, int, long, float, double, char, boolean`

Benutzereingaben

- Primitive Datentypen
  - boolean `scanner.nextBoolean();`
  - char – nicht möglich
  - byte `scanner.nextByte();`
  - short `scanner.nextShort();`
  - int `scanner.nextInt();`
  - long `scanner.nextLong();`
  - float `scanner.nextFloat();`
  - double `scanner.nextDouble();`
- Nicht-primitive Datentypen
  - String `scanner.nextLine();`

Operatoren

Subtraktion

- Operator `-`

- Unärer Operator

```
ergebnis = - operand ;
```

- Binärer Operator

```
differenz = minuend - subtrahend ;
```

- Beispiele

```
4 = -(-4)
```

```
5 = 6 - 1
```

Operatoren allgemein

- Operator **#**

- Unäre Operatoren

```
ergebnis = operand # ;
ergebnis = # operand ;
```

- Binäre Operatoren

```
ergebnis = operand1 # operand2 ;
```

Operatoren in Java

- Arithmetische Operatoren
  - Zahl = Zahl # Zahl
- Relationale Operatoren
  - Wahrheitswert = Zahl # Zahl
- Logische Operatoren
  - Wahrheitswert = Wahrheitswert # Wahrheitswert
- Bitweise Operatoren
  - Zahl = Zahl # Zahl
- String-Konkatenation
- Bedingungs-Operator
- Zuweisungs-Operator

Arithmetische Operatoren

- Werden auf numerische Variablen angewendet
  - Char werden zu Integer gecastet
  - Bei binären Operatoren und unterschiedlichen Datentypen der Operanden wird der niedrigere Operand auf den höheren Operanden vor der Berechnung implizit gecastet
- Datentyp des Ergebnisses entspricht dem Datentyp der Operanden (ggf. nach Typecast)
  - Modulo-Operator kann nur auf Ganzzahlen angewendet werden
  - Division oder Modulo durch 0 wirft bei Ganzzahlen einen Fehler
- Binär: `+ - * / %`
- Unär: `+ -`

Arithmetische Operatoren

- Additions-Operator `+`
  - Bildet die Summe zweier Zahlen
- Subtraktions-Operator `-`
  - Bildet die Differenz zweier Zahlen
- Multiplikations-Operator `*`
  - Bildet das Produkt zweier Zahlen
- Divisions-Operator `/`
  - Bildet den Quotienten zweier Zahlen
- Modulo-Operator `%`
  - Berechnet den Rest, der bei der Division von ganzen Zahlen bleibt

Arithmetische Operatoren

- Beispiele

```
8 = 2 + 6

2.6 = 2 * 1.3

1.5 = 5.5 - 4

1 = 8 / 5

1.6 = 8.0 / 5

3 = 8 % 5

7 = + 7

-7 = - 7
```

Relationale Operatoren

- Werden auf numerische Variablen angewendet
  - Char werden zu Integer gecastet
  - Bei binären Operatoren und unterschiedlichen Datentypen der Operanden wird der niedrigere Operand auf den höheren Operanden vor der Berechnung implizit gecastet
- Datentyp des Ergebnisses ist boolean
  - Gleichheits-Relation nicht mit Zuweisungs-Operator verwechseln!
  - Der exakte Vergleich von Gleitkommazahlen ist wegen Rundungsfehlern mit Vorsicht zu genießen!
- Binär: `==` `!=` `<` `>` `<=` `>=`

Relationale Operatoren

- Gleich `==`
- Ungleich `!=`
- Kleiner als `<`
- Größer als `>`
- Kleiner als oder gleich `<=`
- Größer als oder gleich `>=`

Relationale Operatoren

- Beispiele

```
false = 2 == 6

true = 2 != 1.3

false = 5.5 < 4

true = 8 > 5

false = 8 <= 5

true = 7 >= 7
```

Logische Operatoren

- Werden auf boolsche Variablen angewendet
- Datentyp des Ergebnisses ist boolean
- Binär: `&& || ^ & | == !=`
- Unär: `!`

Logische Operatoren

- Negation `!`
- Short-Circuit Und `&&`
- Short-Circuit Oder `||`
- Exklusives Oder `^`
- Und `&`
- Oder `|`
- Gleich `==`
- Ungleich `!=`

Logische Operatoren

- Negation `!`
  - `false = !true`
  - `true = !false`
- Short-Circuit Und `&&`
  - `true = true && true`
  - `false = true && false`
  - `false = false && true`
  - `false = false && false`
- Short-Circuit Oder `||`
  - `true = true || true`
  - `true = true || false`
  - `true = false || true`
  - `false = false || false`

Logische Operatoren

- XOR `^`
  - `false = true ^ true`
  - `true = true ^ false`
  - `true = false ^ true`
  - `false = false ^ false`
- AND OR `& |`
  - Siehe entsprechender Short-Circuit Operator
- Gleich `==`
  - Wie XOR, nur anders herum
- Ungleich `!=`
  - Siehe XOR

Logische Operatoren

- Mit den Operatoren `!`, `&&` und `||` kann man alle anderen logischen Operatoren ersetzen
- Beispiele für Äquivalenzen
  - `A ^ B  = A & !B | B & !A`
  - `A == B  = A & B | !A & !B`
- Gesetze von De Morgan
  - `!(A & B)  = !A | !B`
  - `!(A | B)  = !A & !B`
- Distributivgesetze
  - `(A|B) & C  = A&C | B&C`
  - `A&B | C  = (A|C) & (B|C)`

Bitweise Operatoren

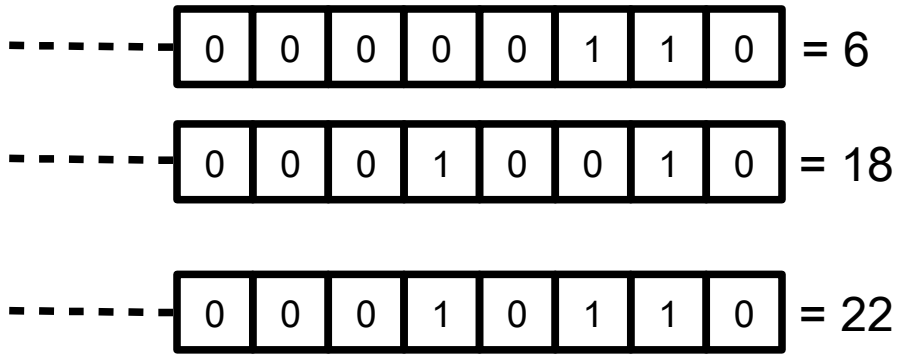
- Werden auf ganzzahlige Variablen angewendet
  - Char werden zu Integer gecastet
  - Bei binären Operatoren und unterschiedlichen Datentypen der Operanden wird der niedrigere Operand auf den höheren Operanden vor der Berechnung implizit gecastet
- Datentyp des Ergebnisses entspricht dem Datentyp der Operanden (ggf. nach Typecast)
- Binär: `&` `|` `^` `>>` `>>>` `<<`
- Unär: `~`

Bitweise Operatoren

- Bitweise Negation `~`
  - Vertauscht Nullen und Einsen in der Bit-Darstellung der Zahl
- Bitweises AND, OR, XOR `& | ^`
  - Bildet bitweises Und, Oder, exklusives Oder (1=true, 0=false)
- Bitshifts `>> >>> <<`
  - Verschiebt alle Bits nach links oder rechts (mit oder ohne Vorzeichen-Bit). Der 2. Operand gibt an, um wie viele Bits verschoben wird

Bitweise Operatoren

- Beispiel: `6 | 18`



String-Konkatenation

- Wird auf zwei Strings angewendet
  - Was kein String ist, wird zu einem String implizit gecastet
  - Char werden zum entsprechenden String gecastet (`'x'` wird zu `"x"`)
  - Numerische Werte werden zu den entsprechenden Zahlenliteralen als String gecastet (`123` wird zu `"123"`)
- Datentyp des Ergebnisses ist ebenfalls String
  - Er besteht aus dem ersten und dem zweiten Operanden zusammengesetzt
- Binär: `+`
- Beispiel: `"Hochschule" = "Hoch" + "schule"`

Bedingungs-Operator

- Wird auf einen boolschen Ausdruck und zwei weitere Variablen, die beide den gleichen Typ haben, angewendet
  - Falls die beiden Variablen nicht den gleichen Typ haben, wird der niedrigere auf den höheren implizit gecastet
  - Ergebnis ist die erste Variable, falls der boolsche Ausdruck wahr ist, ansonsten ist das Ergebnis die zweite Variable
- Datentyp des Ergebnisses entspricht dem Datentyp der beiden Variablen (ggf. nach Typecast)
- Allgemein: `ergebnis = ausdruck ? operand1 : operand2;`
- Beispiel: `"Nein" = (2 > 3) ? "Ja" : "Nein"`

Zuweisungs-Operator

- Das Gleichheitszeichen weist einen Wert einer Variablen zu
- Das Ergebnis des Zuweisungsoperators ist der Wert, der nach der Zuweisung in der Variablen steht
- Allgemein: `ergebnis = variable = ausdruck;`
- Beispiel: `System.out.print(x = 4);`

Abkürzungen

- Zuweisung mit Operation
  - Kurz: `variable #= wert;`
  - Lang: `variable = variable # wert;`
  - Resultat ist der Wert der Variablen nach der Veränderung
- Inkrement und Dekrement
  - Präfix
    - Kurz: `++variable; --variable;`
    - Lang: `variable = variable+1; variable = variable-1;`
    - Resultat ist der Wert der Variablen nach der Veränderung
  - Postfix
    - Kurz: `variable++; variable--;`
    - Lang: `variable = variable+1; variable = variable-1;`
    - Resultat ist der Wert der Variablen VOR der Veränderung!

Auswertungsreihenfolge

- Welche Operation wird zuerst ausgeführt?
  - Unäre Operatoren werden immer zuerst ausgeführt
  - Punkt vor Strich
  - Und vor Oder
- Vollständige Liste: Java ist auch eine Insel
  - Kapitel 2.4.9
- Nutzen Sie Klammern!! ( )

Hausaufgaben

2.3 Datentypen, Typisierung, Variablen und Zuweisungen

2.4 Ausdrücke, Operanden und Operatoren