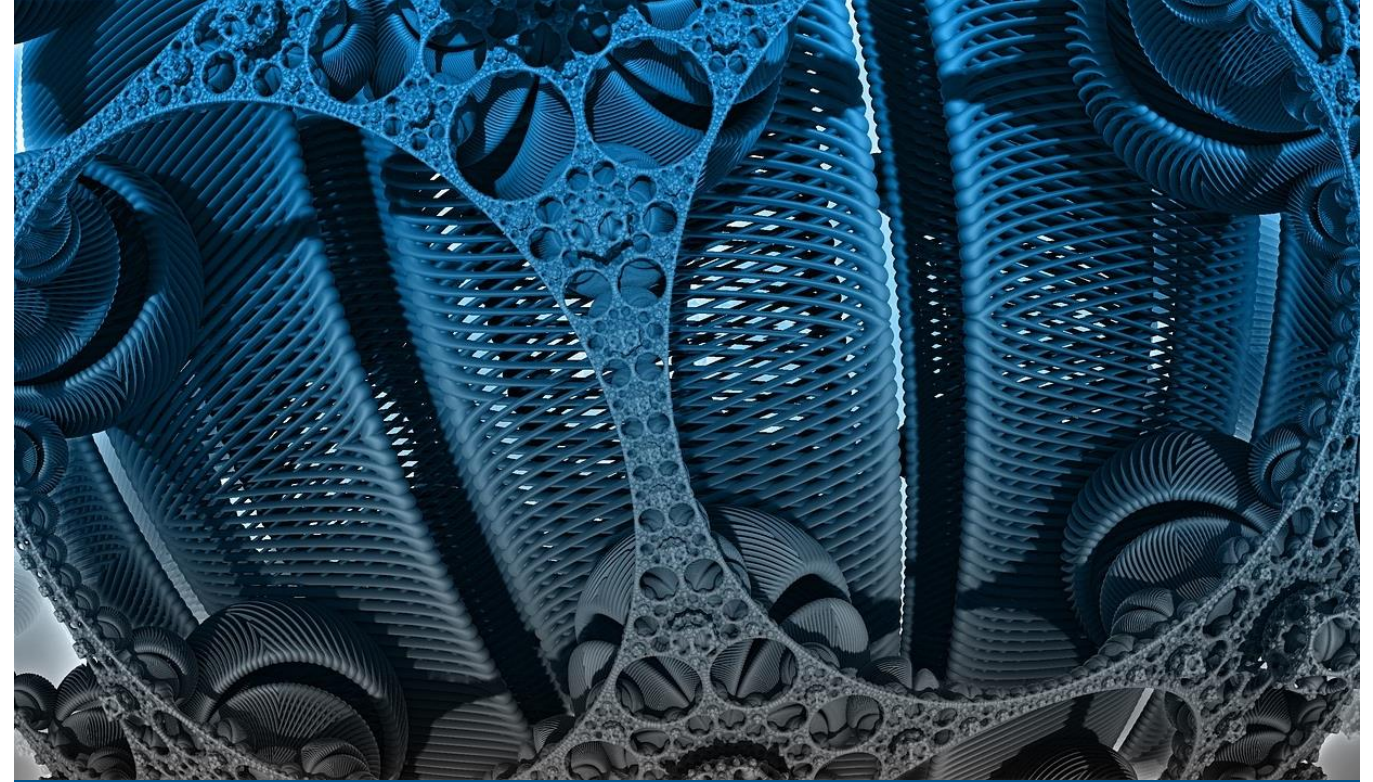




Designing Education
Connecting People

Das erwartet Sie:

- Algorithmen
- Java
- Datenstrukturen



Funktionalität in Anwendungen realisieren

Lernfeld 11a



Die Themen und Lernziele



Algorithmen

Lernziel

Verstehen, worum es bei Algorithmen geht und wo man sie einsetzt



Funktionalität in Anwendungen realisieren

Lernziel

Eine typsichere, objekt-orientierte Programmiersprache beherrschen



Benutzerschnitt- stellen gestalten und entwickeln

Lernziel

Grafische Oberflächen in einer OO-Sprache entwickeln



Software testen

Lernziel

Testfälle formulieren und anwenden



Designing Education
Connecting People

Das erwartet Sie:

- Java
 - Datentypen
 - Operatoren
 - Zeichendarstellung



Funktionalität in Anwendungen realisieren

Lernfeld 11a



Java

Lernziel

Eine typsichere,
objektorientierte
Programmiersprache
beherrschen

Java und die Java Virtual Machine

```
1 import javax.swing.JOptionPane;
2 /**
3  * Ausdruck von 2 Zeilen in einem Dialogfenster
4  */
5 public class Willkommen {
6     public static void main(String... args) {
7         JOptionPane.showMessageDialog(null, "Willkommen zur\nJava Programmierung!");
8     }
9 }
```

Willkommen.java



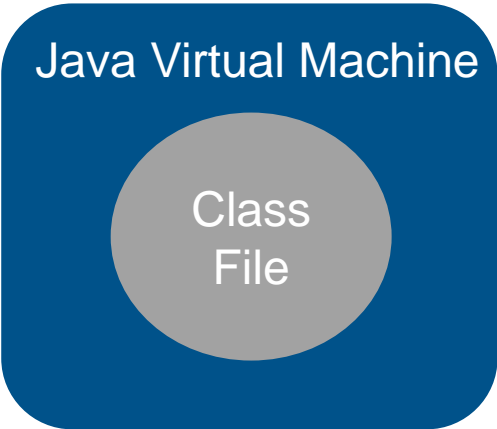
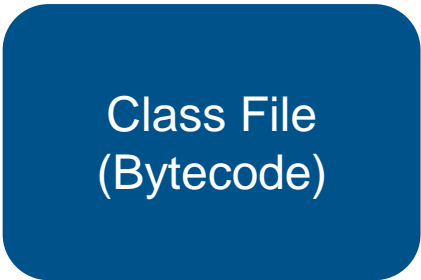
```
Datei Bearbeiten Fenster Hilfe
C:\> javac Willkommen.java
```

Compile



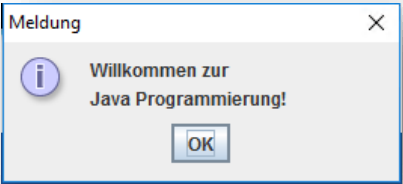
```
1 00000000: <code>
2 00000001: <code>
3 00000002: <code>
4 00000003: <code>
5 00000004: <code>
6 00000005: <code>
7 00000006: <code>
8 00000007: <code>
9 00000008: <code>
```

Willkommen.class



Execute

```
Datei Bearbeiten Fenster Hilfe
C:\> java Willkommen
```



Class File

```
1  xCAxFExBAxBENULNULNUL4NULFS
2  NULENONULDC3BSNULDC4
3  NULNAKNULSYNBELNULETBBELNULCANSOHNULACK<init>SOHNULETX()VSOHNULEOTCodeSOHNULSILineNumberTableSOHNULDC2LocalVaria
bleTableSOHNULEOTthisSOHNULDC1Ltest/Willkommen;SOHNULEOTmainSOHNULSYN([Ljava/lang/String;)VSOHNULEOTargsSOHNULDC3[
Ljava/lang/String;SOHNUL
4  SourceFileSOHNULSILWillkommen.javaFFNULACKNULBELSOHNUL$Willkommen zur
5  Java
   Programmierung!BELNULEMFFNULSUBNULEESC SOHNULSILtest/WillkommenSOHNULDLEjava/lang/ObjectSOHNULETBjavax/swing/JOptionP
aneSOHNULDC1showMessageDialogSOHNUL)(Ljava/awt/Component;Ljava/lang/Object;)VNUL!NULEOTNULENONULNULNULNULNULSTXNUL
SOHNULACKNULBELNULSOHNULBSNULNULNUL/NULSOHNULSOHNULNULNULENO*xB7NULSOHxB1NULNULNULSTXNUL
NULNULNULACKNULSOHNULNULNULDC1NUL
6  NULNULNULFFNULSOHNULNULNULENONULVTNULFFNULNULNULNULNUL
7  NULSONULSOHNULBSNULNULNUL5NULSTXNULSOHNULNULNULBELSOHDC2STXxB8NULETXxB1NULNULNULSTXNULNULNULNUL
8  NULSTXNULNULNULCANULACKNULEMNUL
9  NULNULNULFFNULSOHNULNULNULBELNULSINULDLENULNULNULSOHNULDC1NULNULNULSTXNULDC2
```

Klasse Kreis berechnen



Umfang und Fläche eines Kreises soll berechnet werden.

Vorbereitungen:

EVA?

Eingabedaten? Radius

Verarbeitung? Formel Umfang: $U = 2 * \pi * r$
Fläche: $F = \pi * r^2$

Ausgabe? Umfang und Fläche

Für π den Wert 3.1415926 annehmen

→ Plan?

Kreisberechnung

Deklarationen
double radius, umfang, fläche

Eingabe radius

Berechnungen
umfang = $2 * \pi * \text{radius}$

fläche = $\pi * \text{radius} * \text{radius}$

Ausgabe umfang, fläche

Klasse Kreis berechnen

Kreisberechnung

Deklarationen
double radius, umfang, fläche

Eingabe radius

Berechnungen
 $\text{umfang} = 2 * \pi * \text{radius}$

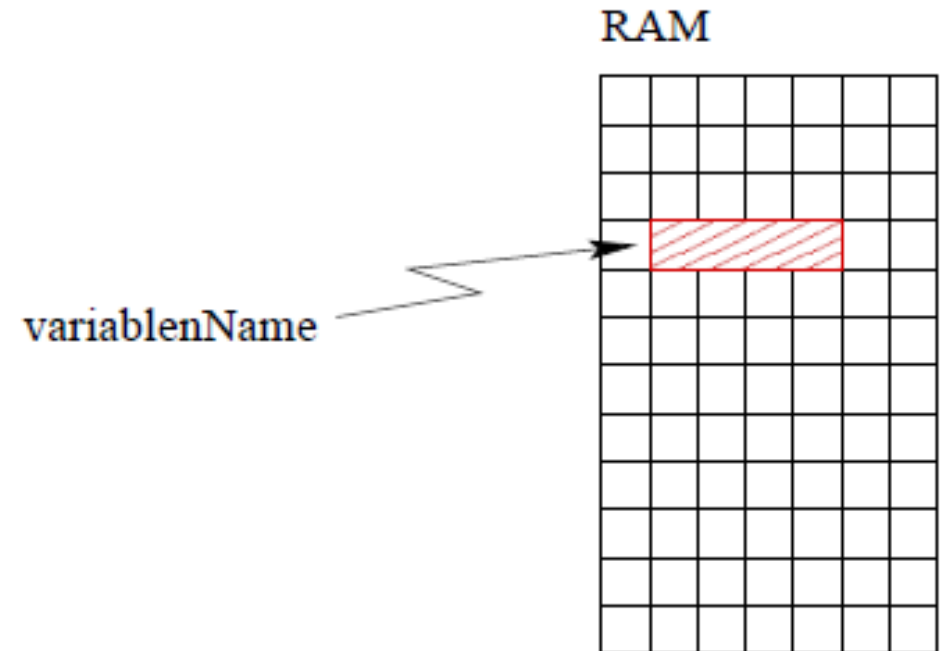
$\text{fläche} = \pi * \text{radius} * \text{radius}$

Ausgabe umfang, fläche

```
15 public static void main(String [] args){
16     // Deklarationen
17
18     double radius, umfang, flaeche;
19     String eingabe;
20
21     // Eingabe
22     eingabe = JOptionPane.showInputDialog
23         ("Geben Sie den Kreisradius ein: ");
24     // Mit einem String kann nicht gerechnet werden
25     // daher muss aus dem String ein numerischer Datentyp
26     // gebildet werden. --> Casten
27
28     radius = Double.parseDouble(eingabe);
29
30     // Berechnungen
31     umfang = 2 * 3.1415926 * radius;
32     flaeche = 3.1415926 * radius * radius;
33
34     // Ausgabe
35
36     JOptionPane.showMessageDialog(null, "Umfang: " + umfang +
37         "\nFläche: " + flaeche);
38 }
```


Datentypen

- Beziehen sich in einer Arithmetik-Applikation auf bestimmte Stellen im Arbeitsspeicher
- Jede Variable hat einen Namen, einen Typ, eine Größe und einen Wert
- Bei der Deklaration am Beginn einer Klassendeklaration werden bestimmte Speicherzellen für die jeweilige Variable reserviert
- Der Name der Variable im Programmcode verweist während der Laufzeit auf die Adressen dieser Speicherzellen



Java- Strict Type/Strong Type

- Streng typisierte Sprachen erzwingen die Typisierung aller Daten, mit denen interagiert wird.

Von nun an können Sie, wann immer Sie *i* verwenden, nur mit ihm als Ganzzahl-Typ interagieren.

Das bedeutet, dass Sie nur mit Methoden verwenden können, die mit ganzen Zahlen arbeiten.

Wie bei Strings können Sie nur als String-Typ mit ihm interagieren.

Sie können sie mit anderen Strings verketten, ausdrucken usw. Aber auch wenn es das Zeichen «4» enthält, können Sie nicht zu einer Ganzzahl hinzufügen, ohne eine Funktion zu verwenden, um den String in einen Ganzzahl-Typ zu konvertieren.

```
int i = 3;  
String s = "4";  
int ergebnis = s + i;
```

incompatible types: String cannot be converted to int

(Alt-Enter shows hints)

```
String text = s + i;  
System.out.println("Text: " + text);
```

```
run:  
Text: 43
```

Datentypen

- **Logischer Datentyp**

Wird als *boolean* bezeichnet und kann die Werte *true* und *false* annehmen.

- **Integraler Datentyp**

Der ganzzahlige Datentyp in Java mit seinen Untervarianten *byte*, *short*, *int* und *long*.

- **Gleitkoma-Datentyp**

Für das Speichern reeller Zahlen sind in Java die beiden IEEE 754 Gleitkommatypen *float* und *double* definiert.

- **Zeichen-Datentyp**

Ein Zeichen wird in Java auf Basis des Unicode-Zeichensatzes interpretiert und wird in dem Datentyp *char* gespeichert .

Datentypen

Datentyp	Größe	Werte	Bemerkungen
boolean	1 Byte	true, false	
char	2 Byte	'\u0000' bis '\uFFFF'	Unicode, 2^{16} Zeichen, <u>stets in Apostrophs (!)</u>
byte	1 Byte	-128 bis +127	$-2^7, -2^7 + 1, \dots, 2^7 - 1$
short	2 Byte	-32 768 bis +32 767	$-2^{15}, -2^{15} + 1, \dots, 2^{15} - 1$
int	4 Byte	-2 147 483 648 bis +2 147 483 647	$-2^{31}, -2^{31} + 1, \dots, 2^{31} - 1$ Beispiele: 123, -23, 0x1A, 0b110
long	8 Byte	-9 223 372 036 854 775 808 bis +9 223 372 036 854 775 807	$-2^{63}, -2^{63} + 1, \dots, 2^{63} - 1$ Beispiele: 123L, -23L, 0x1AL, 0b110L
float	4 Byte	$\pm 1.4\text{E-}45$ bis $\pm 3.4028235\text{E+}38$	$[\approx \pm 2^{-149}, \approx \pm 2^{128}]$ Beispiele: 1.0f, 1.F, .05F, 3.14E-5F
double	8 Byte	$\pm 4.9\text{E-}324$ bis $\pm 1.7976931348623157\text{E+}308$	$[\pm 2^{-1074}, \approx \pm 2^{1024}]$ Beispiele: 1.0, 1., .05, 3.14E-5

Variablen Deklaration

- Eine Variable wird deklariert mit dem Datentyp und dem Variablennamen.
- Die Anweisung wird mit Semikolon abgeschlossen.
- Der Wertebereich erstreckt sich von -2^{31} bis $2^{31}-1$ und belegt 32 Bit im RAM.

```
Datentyp VARIABLENAME ;
```

```
int m;
```

```
int x,y,z;
```

Variablennamen

Allgemeine Konventionen

Ähnlich wie in der "realen Welt" sollte man sich bei der Programmierung ebenfalls um einen guten Ton bemühen.

- es ist ein Unterschied ob ich alles klein und zusammenschreibe
ODER
- `_`leserlich_formatiere_sodass_mein_Code_besser_nachvollzogen_werden_kann

Grundlegende Regel hierbei ist die Verständlichkeit und leichte Nachvollziehbarkeit des verfassten Codes (und die Kommentare nicht vergessen).

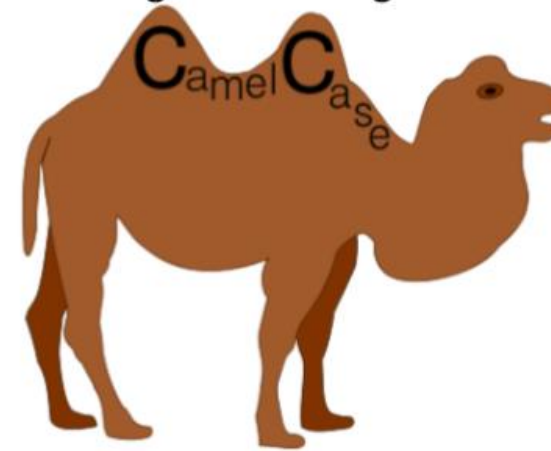
Aus diesem Grund wurden die Java-Code-Konventionen geschaffen.

Variablennamen

Wenn ein Bezeichnername aus mehreren Wörtern besteht,
werden zur besseren Lesbarkeit alle Wortanfänge in Großbuchstaben geschrieben.

eineKleineMaus
anzahlAllerKinder

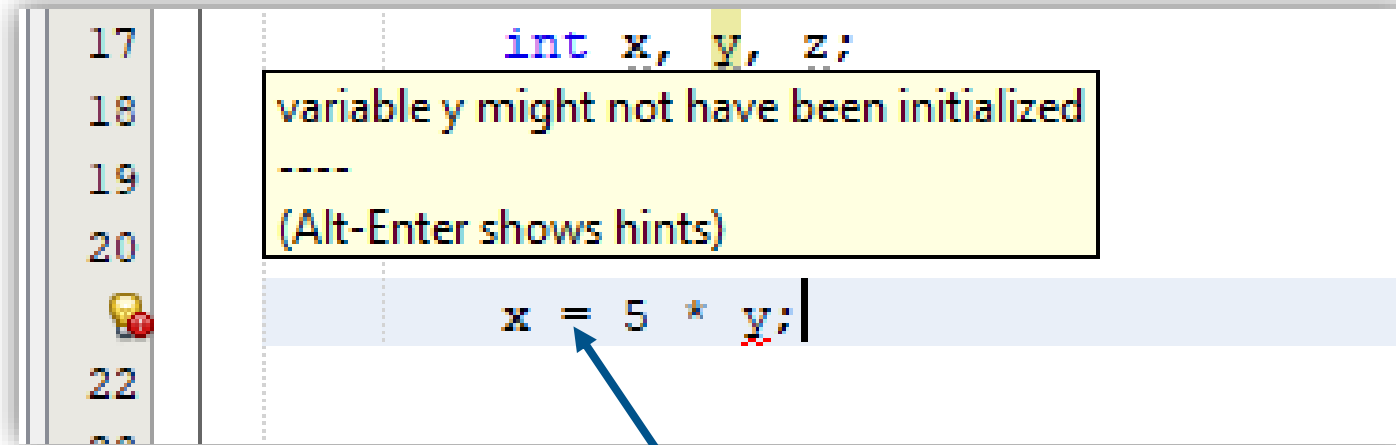
"CamelCase" wg. der höckerartigen Erhöhungen im Namen



Initialisierung

Die Variable x hat nun einen Wert, der in dem reservierten RAM-Bereich gespeichert wird.
Die Variable y wurde bereits deklariert, aber noch nicht initialisiert.

Das merkt der Compiler bei folgendem Code-Ausschnitt und meldet:



The screenshot shows a code editor with a line of code: `int x, y, z;` on line 17 and `x = 5 * y;` on line 20. A yellow tooltip box is displayed over the code, containing the text: `variable y might not have been initialized`, followed by `----` and `(Alt-Enter shows hints)`. A blue arrow points from the word `y` in the assignment statement to a blue box below the code.

```
17      int x, y, z;  
18  
19  
20      x = 5 * y;  
22
```

Wertzuweisung

Deklaration und Initialisierung

Als **Literal** werden Werte bezeichnet, die im Code als feste Größe bzw. als Initialwert stehen.

```
// Wertzuweisungen :
boolean geschlossen = true;
char addsign = '@';
byte oktett = 127;
short breite = 345;
int anzahl = 475643;
long sandkoerner = 2345123567756845L; // L als letztes Zeichen
// für long-Literal

float zinsSatz = 3.5F; // F als letztes Zeichen für float-Literal
double ertrag = 2436.12;
double kapital = 1546312.56;
double schulden = 3_215_410.25; // Unterstrich als Tausender
// Trennzeichen ist besser lesbar
```

Casting

= Umwandlung eines Datentyps in einen anderen.

auch:

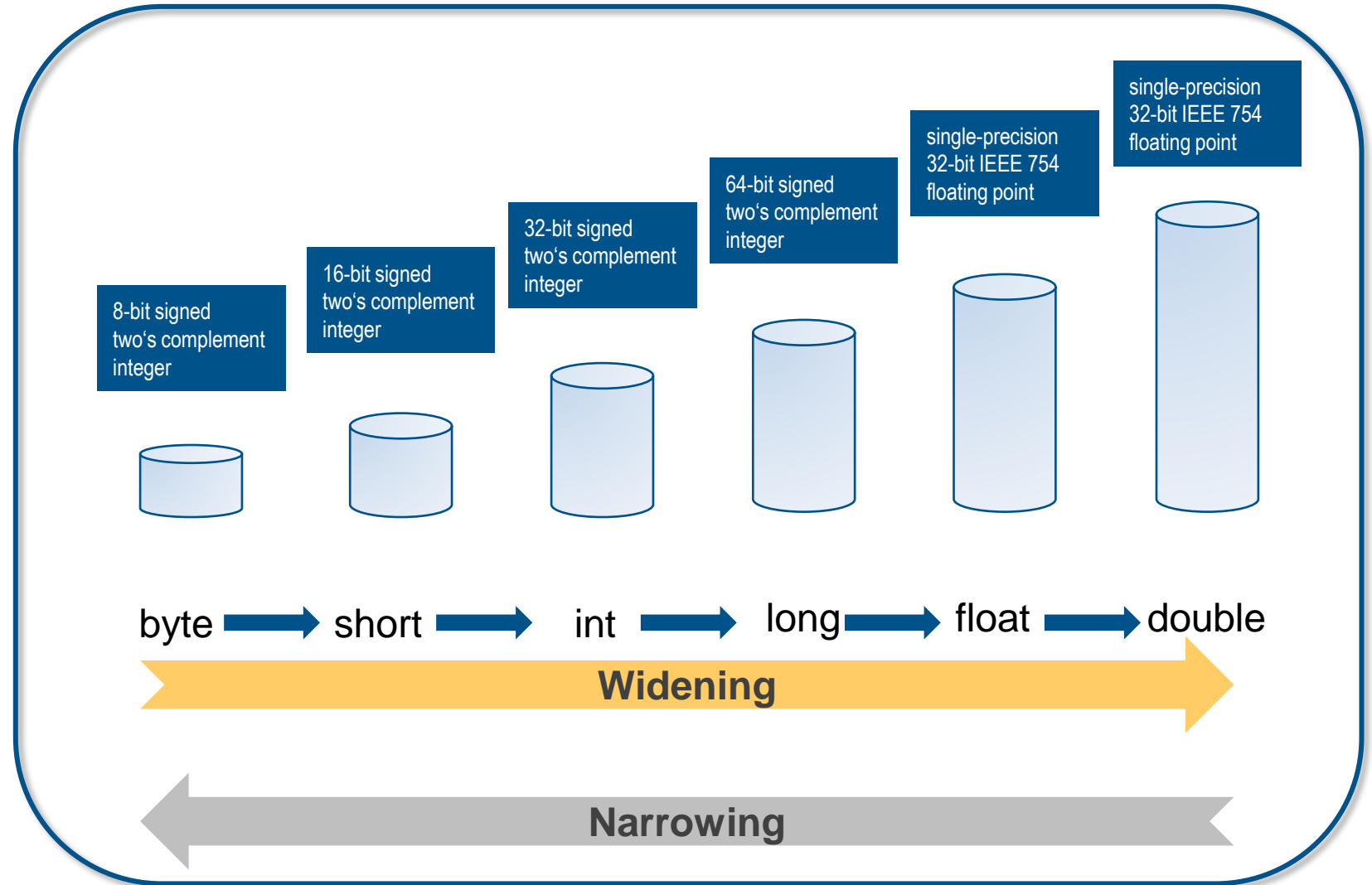
Typumwandlung,
Typanpassung,
Konvertierung

Widening:

Converting a lower data type into higher data type

Narrowing:

Converting a higher data type into lower data type
(may lose precision during conversion)



Casting

Implizit

- Typumwandlung geschieht „automatisch“ (Widening)
- Daten des kleineren Datentyps werden durch den Compiler automatisch dem größeren angepasst
→ nur dann, wenn kein Datenverlust!

```
double d = 1;
```

1 ist ein `int`-Literal, wird automatisch in 1.0 umgewandelt

```
int zeichenWert = 'A';
```

Der ASCII-Code des Buchstabens A = 65 wird als `int` zugewiesen

Explizit

- Einen Wert eines gegebenen Datentyps in einen anderen explizit konvertieren (Narrowing)

```
double x = 3.1415;  
int n = (int) x;
```

- Die Nachkommastellen werden ignoriert, n hat nun den Wert 3
- Der gewünschte Zieldatentyp wird **in Klammern** (*Datentyp*) vor den zu konvertierenden Ausdruck, der Variable oder dem Literal formuliert.

Darstellungsformen von Zeichen und Text

Ein Zeichenkodierungsschema (Character set) definiert einen Code, mit dem Zeichen maschinenlesbar werden

- ASCII-Zeichensatz
- ISO-8859-Norm
- Unicode
 - UTF-8
 - UTF16
 - UTF-32

Darstellungsformen von Zeichen und Text

ISO 8859-1 / ISO-8859-15

- An den Codepunkten 01 bis 1F stehen in ASCII, ISO-8859-1 und Unicode nicht-darstellbare Steuerzeichen. Sie sind deshalb unterdrückt
- Deutsche Umlaute existieren (Ü → Entity:Ü)
- Währungssymbol € - noch nicht enthalten
 - Während moderne Browser die "ISO8859-1 zu Windows-1252"-Ersetzung beherrschen, tun dies die meisten E-Mail-Programme nicht
 - Das echte Euro-Zeichen steht in Unicode an Position 20AC
- Wird nicht mehr aktiv weiterentwickelt

ISO 8859

-1	Latin-1, Westeuropäisch
-2	Latin-2, Mitteleuropäisch
-3	Latin-3, Südeuropäisch
-4	Latin-4, Nordeuropäisch
-5	Kyrillisch
-6	Arabisch
-7	Griechisch
-8	Hebräisch
-9	Latin-5, Türkisch
-10	Latin-6, Nordisch
-11	Thai
-12	(existiert nicht)
-13	Latin-7, Baltisch
-14	Latin-8, Keltisch
-15	Latin-9, Westeuropäisch
-16	Latin-10, Südosteuropäisch

Zeichensätze

Quelle: https://de.wikipedia.org/wiki/ISO_8859

Darstellungsformen von Zeichen und Text

- Ein Java-Programm besteht aus einer Folge von Unicode-Zeichen.
- Der Unicode-Zeichensatz fasst eine große Zahl internationaler Zeichensätze zusammen und integriert sie in einem einheitlichen Darstellungsmodell.
- Da die 256 verfügbaren Zeichen eines 8-Bit-Wortes bei weitem nicht ausreichen, um die über 30.000 unterschiedlichen Zeichen des Unicode-Zeichensatzes darzustellen, ist ein Unicode-Zeichen 2 Byte, also 16 Bit, lang.
- Der Unicode ist mit den ersten 128 Zeichen des ASCII- und mit den ersten 256 Zeichen des ISO-8859-1-Zeichensatzes kompatibel.
- Die Integration des Unicode-Zeichensatzes geht in Java so weit, dass neben *String*- und *char*-Typen auch die literalen Symbole und Bezeichner der Programmiersprache im Unicode realisiert sind.
- Es ist daher ohne weiteres möglich, Variablen- oder Klassennamen mit nationalen Sonderzeichen oder anderen Symbolen zu versehen.

Unicode

Unicode → UTF-8/UTF-16/UTF-32

- Codierung globaler Kommunikation
- 1. Version 1990
- Ist in den ersten 128 Zeichen deckungsgleich mit ASCII
- UTF-8 kann einfache Zeichen, die dem Zeichenvorrat von ASCII angehören, als 8-Bit-Wert speichern
- UTF-16 speichert ein Zeichen als 16-Bit-Wert

Unicode Codepos.	Zeichen	UTF-8 (hex.)	Name
U+0000		00	<control>
U+0001		01	<control>
U+0002		02	<control>
U+0003		03	<control>
U+0004		04	<control>
U+0005		05	<control>

U+003D	=	3d	EQUALS SIGN
U+003E	>	3e	GREATER-THAN SIGN
U+003F	?	3f	QUESTION MARK
U+0040	@	40	COMMERCIAL AT
U+0041	A	41	LATIN CAPITAL LETTER A
U+0042	B	42	LATIN CAPITAL LETTER B
U+0043	C	43	LATIN CAPITAL LETTER C
U+0044	D	44	LATIN CAPITAL LETTER D
U+0045	E	45	LATIN CAPITAL LETTER E
U+0046	F	46	LATIN CAPITAL LETTER F
U+0047	G	47	LATIN CAPITAL LETTER G
U+0048	H	48	LATIN CAPITAL LETTER H
U+0049	I	49	LATIN CAPITAL LETTER I

Unicode

- Der Unicode-Standard nutzt das Präfix »U+«, gefolgt von Hexadezimalzahlen.
- Prinzipiell umfasst der Bereich 1.114.112 mögliche Codepunkte, von U+0000 bis U+10FFFF.
- Obwohl Java intern alle Zeichenfolgen in Unicode kodiert, ist es ungünstig, Klassennamen zu wählen, die Unicode-Zeichen enthalten.
- Einige Dateisysteme speichern die Namen im alten 8-Bit-ASCII-Zeichensatz ab, sodass Teile des Unicode-Zeichens verloren gehen.
- Werden Texte ausgetauscht, sind sie üblicherweise UTF-8-kodiert.
Bei Webseiten ist das ein guter Standard.
UTF-16 ist für Dokumente seltener, wird aber häufiger als interne Textrepräsentation genutzt.
So verwenden zum Beispiel die JVM und die .NET-Laufzeitumgebung intern UTF-16.

Glyph	A	B	東	᠔
Unicode-Codepoint	U+0041	U+00DF	U+6771	U+10400
UTF-32	00000041	000000DF	00006771	00010400
UTF-16	0041	00DF	6771	D801 DC00
UTF-8	41	C3 9F	E6 9D B1	F0 90 90 80

Quelle: https://openbook.rheinwerk-verlag.de/javainsel/04_001.html#u4.1.3

Unicode

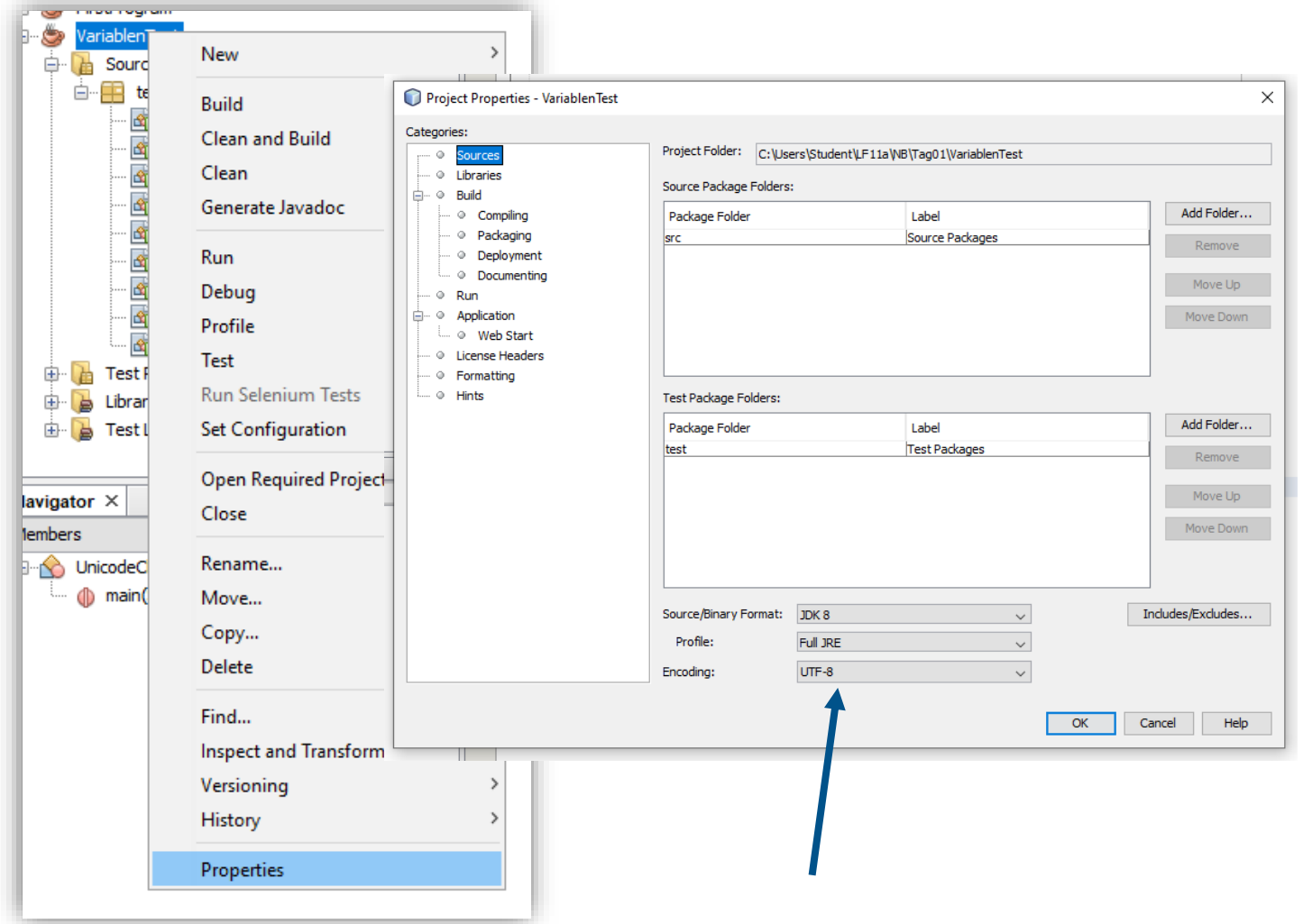
- \u maskiert die nachfolgenden Zeichen als Kodierung in einem Unicode-Zeichensatz
- Das Unicode-Zeichen wird mit Hilfe von Hexadezimalzahlen kodiert

```
public class UnicodeChar {  
    public static void main(String[] args) {  
  
        char buchstabe = 'A';  
        char ziffer = 65;  
        char unicode = '\u0041';  
  
        System.out.println("Buchstabe: " + buchstabe);  
        System.out.println("Ziffer: " + ziffer);  
        System.out.println("Unicode: " + unicode);  
  
    }  
}
```

```
run:  
Buchstabe: A  
Ziffer: A  
Unicode: A
```

Unicode in Netbeans

- Rechtsklick auf das Projekt.
- Auswahl von Properties im Kontextmenü.
- Die Kategorie Sources ist aktiv.
- Das Kombinationsfeld Encoding auf der Seite rechts unten gibt Auskunft über den genutzten Zeichensatz



Unicode in Netbeans

- ... und Microsoft Eingabeaufforderung
- Die Microsoft Eingabeaufforderung unterstützt standardmäßig nur ASCII-Zeichencode (siehe <http://www.ascii-code.com/>).
- Der Befehl chcp zeigt die aktuelle genutzte Codepage an.
- Der Befehl chcp 1252 stellt die aktuelle Codepage auf „West European Latin“ um.
- Der Befehl chcp 65001 (siehe Kapitel 2) stellt die aktuelle Codepage auf „UTF-8“ Encoding um.

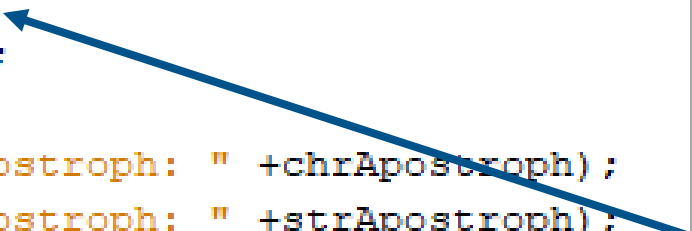
```
C:\Users\Student\LF11a>chcp  
Aktive Codepage: 65001.
```

Escape Sequenzen (Fluchtsymbole)

Um spezielle Zeichen, etwa den Zeilenumbruch oder Tabulator, in einen *String* oder *char* setzen zu können, stehen Escape-Sequenzen [128] zur Verfügung.

Dienen z. B. als Steuerzeichen für den Drucker

```
char chrApostroph = '\\';  
String strApostroph = "\"";  
  
System.out.println("chrApostroph: " +chrApostroph);  
System.out.println("strApostroph: " +strApostroph);
```



```
run:  
chrApostroph: '  
strApostroph: '"
```

Zeichen	Bedeutung
\b	Rückschritt (Backspace)
\n	Zeilenschaltung (Newline)
\f	Seitenumbruch (Formfeed)
\r	Wagenrücklauf (Carriage Return)
\t	horizontaler Tabulator
\"	doppeltes Anführungszeichen
'	einfaches Anführungszeichen
\\	Backslash

Escape Sequenzen (Fluchtsymbole)

- Ein Apostroph begrenzt ein Wert vom Typ *char*.
Das Anführungszeichen wird als ASCII-Zeichen behandelt.
- Ein Anführungszeichen begrenzt ein *String*.
Das Anführungszeichen als Zeichen muss in einem String maskiert werden.

```
char chrAnfuhrungszeichen = '';  
String strAnfuhrungszeichen = "\"";  
  
System.out.println("chrAnfuhrungszeichen: " +chrAnfuhrungszeichen);  
System.out.println("strAnfuhrungszeichen: " +strAnfuhrungszeichen);
```

```
run:  
chrAnfuhrungszeichen: "  
strAnfuhrungszeichen: "
```

Datentyp String

Eine Variable vom Typ *String* (ist kein primitiver Datentyp, sondern eine Klasse) muss auch initialisiert werden.

```
String text = ""; // Initialisierung mit einem leeren Text
```

```
String name;      // Deklaration  
name = "Java";    // Initialisierung
```

```
String name = "Java";
```

Deklaration und Initialisierung in einem

Datentyp String

Klasse `String`

Objekt mit dem Bezeichner `name`

```
String name = new String();
```

```
name = "Java";
```

Konstruktor

Erzeugungsoperator

Wertzuweisung

Klasse String

<code>name.charAt(i)</code>	Liefert ein Zeichen aus einem String-Objekt
<code>name.substring(i,j)</code>	Liefert einen Teilstring aus einem String-Objekt
<code>name.length()</code>	Liefert die Anzahl der gültigen Zeichen in einem String-Objekt
<code>name.equals(xxx)</code>	Prüft, ob der Inhalt des String-Objektes xxx ist
<code>name.compareTo(xxx)</code>	Prüft, ob der Inhalt des String-Objektes lexikalisch vor oder nach xxx kommt
<code>name.toUpperCase ()</code>	Liefert einen String, in dem alle Zeichen von name als Großbuchstaben vorliegen
<code>name.toLowerCase()</code>	Liefert einen String, in dem alle Zeichen von name als Kleinbuchstaben vorliegen
<code>Integer.parseInt(name)</code>	Enthält das String-Objekt Zeichen, die als int -Zahl interpretiert werden kann, so liefert diese Methode der Klasse Integer den int -Wert des ursprünglichen Strings.

[Auszug der Methoden, der Klasse String]

Die Methode `.charAt(n)`

```
String text = "JAVA-IST-TOLL";
```

J	A	V	A	-	I	S	T	-	T	O	L	L
0	1	2	3	4	5	6	7	8	9	10	11	12

```
char zeichen = text.charAt(7);  
  
System.out.println("Zeichen: " + zeichen);
```

```
run:  
Zeichen: T
```

Die Methode `.substring(i,j)`

```
String text = "JAVA-IST-TOLL";
```

J	A	V	A	-	I	S	T	-	T	O	L	L
0	1	2	3	4	5	6	7	8	9	10	11	12

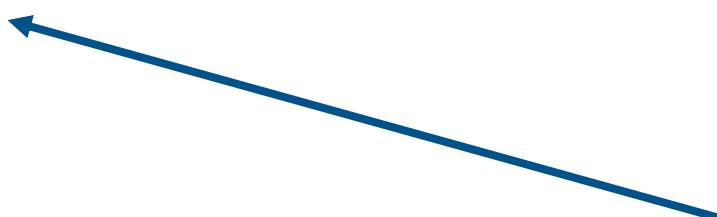
```
String teil = text.substring(3,7);  
System.out.println("Ausgabe: " +teil);
```

```
run:  
Ausgabe: A-IS
```

Die Methode .substring(i)

```
String text = "JAVA-IST-TOLL";
```

J	A	V	A	-	I	S	T	-	T	O	L	L
0	1	2	3	4	5	6	7	8	9	10	11	12



```
String teil2 = text.substring(2);  
System.out.println("Ausgabe: " + teil2);
```

```
run:  
Ausgabe: VA-IST-TOLL
```

Der Teilstring beginnt mit dem Zeichen am angegebenen Index und reicht bis zum Ende dieses Strings oder bis zum Ende-Index – 1, wenn das zweite Argument angegeben wird.

Die Methode .length()

```
String text = "JAVA-IST-TOLL";
```

J	A	V	A	-	I	S	T	-	T	O	L	L
0	1	2	3	4	5	6	7	8	9	10	11	12

```
int laenge = text.length();  
System.out.println("Ausgabe: " + laenge);
```

```
run:  
Ausgabe: 13
```

Die Methode .equals(xxx)

```
String text1 = "JAVA-IST-TOLL";  
String text2 = "Java-ist-toll";  
  
boolean gleich = text1.equals(text2);  
  
System.out.println("Gleich: " + gleich);
```

```
run:  
Gleich: false
```

Die Methode `.compareTo(xxx)`

Der Inhalt von `name1` kommt alphabetisch nach dem Inhalt von `name2`

```
public class Sortierung {  
    public static void main(String[] args) {  
        String name1 = "Günter";  
        String name2 = "Christine";  
  
        int i = name1.compareTo(name2);  
    }  
}
```

→ Die Variable `i` enthält nun:

- einen Wert kleiner als 0, wenn `name1` nach alphabetischer Sortierung vor `name2` kommt
- 0, wenn `name1` und `name2` gleich sind (d. h., dass `name1.equals(name2)` true ergibt)
- einen Wert größer als 0, wenn `name1` nach alphabetischer Sortierung nach `name2` kommt

Die Methode `.toUpperCase()`/ `.toLowerCase`

Beachten Sie, dass Strings in Java unveränderbar sind und der Aufruf von `toUpperCase` einen neuen String erzeugt.

Mit anderen Worten, der *text* bleibt beim Aufruf von `UpperCase` unverändert.

```
public class UpperLower {  
    public static void main(String[] args) {  
        String text = "JAvA-ist toll";  
  
        String gross= text.toUpperCase();  
        String klein = text.toLowerCase();  
  
        System.out.println("Ausgabe: " + gross + " " + klein);  
    }  
}
```

Genau wie bei `toUpperCase` ändert auch `toLowerCase` den Wert von *text* nicht.

```
run:  
Ausgabe: JAVA-IST TOLL java-ist toll
```

Die Methode Integer.parseInt(xxx)

```
public class StringToInteger {  
    public static void main(String[] args) {  
        String text1 = "1234";  
        String text2 = "5678";  
  
        String textgesamt= text1+text2;  
        System.out.println("Ausgabe: " + textgesamt );  
    }  
}
```

```
run:  
Ausgabe: 12345678
```

Diese Pluszeichen
haben
unterschiedliche
Funktionalität

```
public class StringToInteger {  
    public static void main(String[] args) {  
        String text1 = "1234";  
        String text2 = "5678";  
  
        String textgesamt= text1+text2;  
        System.out.println("Ausgabe: " + textgesamt );  
  
        int zahl1 = Integer.parseInt(text1);  
        int zahl2 = Integer.parseInt(text2);  
        int ergebnis = zahl1 + zahl2;  
  
        System.out.println("Ausgabe: "+ ergebnis);  
    }  
}
```

```
run:  
Ausgabe: 12345678  
Ausgabe: 6912
```

Kompetenzcheck



Welche Aussagen sind richtig?

- a. Variablen werden zur temporären Speicherung von Daten verwendet.
- b. Bei der Deklaration von Variablen muss kein Datentyp angegeben werden.
- c. Die Initialisierung einer Variablen ist die erste Zuweisung eines Wertes.
- d. Nach dem compile eines Java-Quellcodes liegt eine entsprechende Java-Datei im Maschinencode vor.
- e. Ziel der Einführung vom Unicode war es, einen einheitlichen Standard für die Darstellung aller bekannten Zeichen zu schaffen.
- f. Variablen- oder Klassennamen können mit nationalen Sonderzeichen oder anderen Symbolen versehen werden.
- g. Variable vom Typ String kann mit einer Variable vom Typ int verknüpft werden.

Übung



Übung

Java Programm

Escape-Sequenzen

Operatoren und deren Prioritäten

Operator	Operation	Präzedenz
()	Klammern	werden zuerst ausgewertet. Sind Klammern von Klammern umschlossen, werden sie von innen nach außen ausgewertet.
*, /, %	Multiplikation, Division, Modulus	werden als zweites ausgewertet
+, -	Addition, Subtraktion	werden zuletzt ausgewertet

- Werden in einer Anweisung mehrere Operatoren verwendet, so werden sie nach einer durch ihre Präzedenz festgelegten Reihenfolge ausgeführt. Eine solche Präzedenz ist z.B. „Punkt vor Strich“.
- Operatoren hängen von den Datentypen ihrer Operanden ab

Operatoren - Arithmetisch

- Die Variable `x` ist vom Typ `double`. `14 / 4` sind aber Integer-Literale, also wird
- 1 Ganzzahl durch Ganzzahl geteilt und das Ergebnis ist 3, eine Integer-Zahl. Erst durch die Zuweisung zur Variablen `x` wird implizit in `double` gecastet.
 - 2 Integer-Literal mit einem Double-Literal in einem Ausdruck wird implizit auf den genauesten Datentyp des Ausdruck gecastet, hier `double` des Literals `4.0`. Es wird also `14.0 / 4.0` gerechnet. Ergebnis ist 3.5.
 - 3 Durch explizites Casting des Literals `14` wird auch das Literal `4` implizit auf `double` gecastet.
 - 4 Der Divisionsoperator `/` hat zwei Integer-Operanden. Daraus ergibt sich eine ganzzahlige Division. Diese ganzzahlige Ergebnis 3 wird der Integer-Variablen `n` zugewiesen.
 - 5 Den Rest der ganzzahligen Division erhält man mit der modulo-Operation `%`. Da mathematisch `14 / 4 = 3` Rest 2; ist `14 % 4` eben 2.

In der Praxis sind diese Effekte nicht immer offensichtlich, für bestimmte Anwendungsfälle allerdings durchaus nützlich.

Beispiel.

```
double x, y, z;
int n, m;

// Ergebnisse

x = 14 / 4;           // x = 3.0    1
y = 14 / 4.0;        // y = 3.5    2
z = (double) 14 / 4;  // z = 3.5    3
n = 14 / 4;           // n = 3      4
m = 14 % 4;           // m = 2      5
```

Datentyp double

Um die beschriebenen Effekte zu vermeiden, gibt es mehrere Möglichkeiten, ein Literal als double-Typ zu formulieren.

- ❶ Die Schreibweise `e1` ist die wissenschaftliche Schreibweise mit Exponent.
Bedeutet $1.4 * 10^1$, also $1.4 * 10 = 14$

```
double s = 14.0, t = 14., u = 14d;
```

```
double v = 1.4e1;
```

❶

Kombinierte Operatoren

Zuweisungs- operator	Beispiel	Bedeutung	Wert für c, wenn <code>int c=11, x=4</code>
<code>+=</code>	<code>c += x;</code>	<code>c = c + x;</code>	15
<code>-=</code>	<code>c -= x;</code>	<code>c = c - x;</code>	7
<code>*=</code>	<code>c *= x;</code>	<code>c = c * x;</code>	44
<code>/=</code>	<code>c /= x;</code>	<code>c = c / x;</code>	2
<code>%=</code>	<code>c %= x;</code>	<code>c = c % x;</code>	3

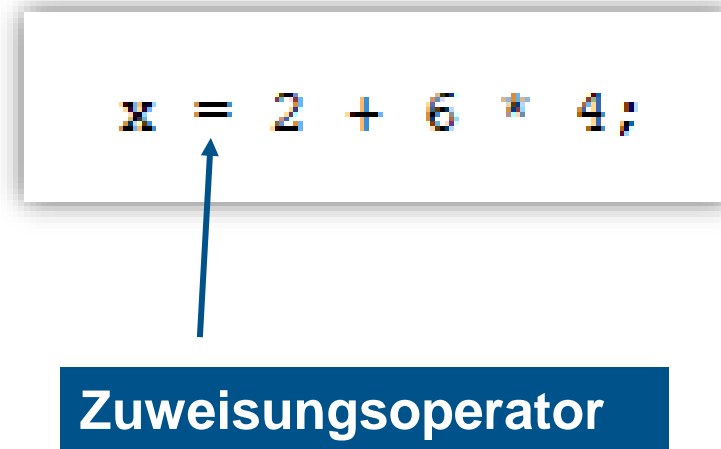
Die Anweisung "c += 4" bewirkt, dass der beim Ausführen der Anweisung aktuelle Wert von c, beispielsweise 11, um 4 erhöht wird und den alten Wert überschreibt.

Zuweisungsoperator

- Der Variablen x wird der Wert, der sich aus dem Ausdruck auf der rechten Seite ergibt, also 26 zugewiesen.

(Java rechnet Punkt- vor Strichrechnung!)

- Der Zuweisungsoperator ist ein Operator mit zwei Operanden, ein sogenannter binärer Operator.
- Die erstmalige Wertzuweisung wird auch Initialisierung der Variablen genannt.



Kompetenzcheck



Welche Aussagen sind richtig?

- a. Variablen werden zur temporären Speicherung von Daten verwendet.
- b. Bei der Deklaration von Variablen muss kein Datentyp angegeben werden.
- c. Die Initialisierung einer Variablen ist die erste Zuweisung eines Wertes.
- d. Nach dem compile eines Java-Quellcodes liegt eine entsprechende Java-Datei im Maschinencode vor.
- e. Ziel der Einführung vom Unicode war es, einen einheitlichen Standard für die Darstellung aller bekannten Zeichen zu schaffen.
- f. Variablen- oder Klassennamen können mit nationalen Sonderzeichen oder anderen Symbolen versehen werden.
- g. Variable vom Typ String kann mit einer Variable vom Typ int verknüpft werden.