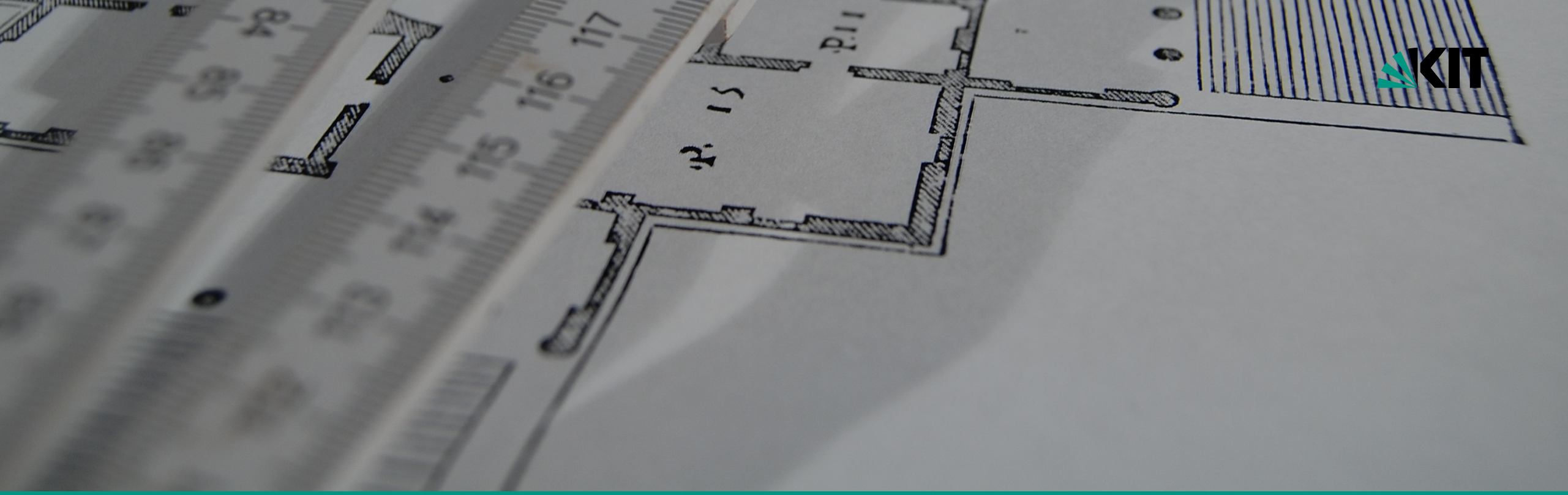

Variablen und Datentypen

Programmieren Tutorium Nr.17

Aleksandr Zakharov | 12. November 2025

Vorab

- Am 03.02.2026 findet die Saalübung statt
  - Übung zur Vorbereitung für die Abschlussaufgaben
- Ab heute könnt ihr eure Abgaben des 1. Übungsblattes machen

Letztes Mal

- **Klassen**
  - Klassen sind "Vorlagen"für Objekten
- **Objekte**
  - Sind gleichartige Instanzen von Klassen
- **Attribute**
  - Sind Eigenschaften eines Objektes und "definierenës
- **Methoden**
  - Definieren das mögliche Verhalten des Objektes

Faustregeln...

- Substantive $\rightarrow$ **Kandidate** für Klassen
- Verben $\rightarrow$ Methoden
- Adjektive $\rightarrow$ eher unwichtig oder Attribute
- "genau X Stück" oder "bis zu X Stück" $\rightarrow$ Array (oder Set/Liste)
- endliche Wertemenge $\rightarrow$ Enum!

Aufgabe 2.1 - Modellierungsaufgaben

Modelliert folgenden Sachverhalt möglichst sinnvoll in Java.

Bei einem Fußballspiel nehmen zwei Mannschaften teil. Eine davon ist die Heimmannschaft, die andere die Gastmannschaft. Ein Spiel findet zu einem bestimmtem Datum in einem Stadion statt. Ein Spiel hat ein Ergebnis. Ein Datum besteht aus Tag, Monat und Jahr. Ein Stadion hat einen Namen, eine Anzahl an Sitzplätzen und eine zugehörige Mannschaft. Eine Mannschaft hat eine Spieleranzahl und einen Namen.

Hinweis: modelliert nur das, was auch im Text steht
Zum Beispiel braucht ihr keine Klasse für die Spieler oder einen Sitzplatz im Stadion zu modellieren.

Lösung 2.1

```java
public class FootballGame {
  FootballTeam homeTeam;
  FootballTeam guestTeam;
  int homePoints;
  int guestPoints;
  FootballStadium   location;
  Date date;
}
public class FootballStadium {
  String name;
  int amountOfSeats;
  FootballTeam homeTeam;
}
```

```java
public class FootballTeam
{
  String name;
  byte playerCount;
}
```

```java
public class Date {
  byte day;
  byte month;
  short year;
}
```

Aufgabe 2.2 - Modellierung Post

Modelliert folgenden Sachverhalt möglichst sinnvoll in Java.

Ein Brief hat Sender und Empfänger, die jeweils Personen sind und wird von einem Postunternehmen zugestellt. Ein Brief kann entweder zugestellt sein oder nicht und hat eine Portogebühr. Eine Person hat Vorund Nachnahme sowie eine Adresse. Eine Adresse besteht aus Straße, Hausnummer, Postleitzahl, Ort und Länderkürzel (hier vereinfachend nur aus einem Buchstaben bestehend). Ein Postunternehmen hat einen Namen und eine 20-stellige Identifikationsnummer.

Lösung 2.2

```java
public class Mail {
  boolean sent;
  Person sender;
  Person receiver;
  MailCompany mailCompany;
  double porto;
}
```

```java
public class MailCompany {
  long id;
  String name;
}
```

```java
public class Person {
  String firstName;
  String name;
  Address address;
}
```

```java
public class Address {
  String street;
  short number;
  //String number;
  short PLZ;
  String place;
  char countryShort;
}
```

Motivation

- letzte Woche: Attribute. Sie haben immer einen Datentyp (in Java)
- Nun lernen wir den Begriff Datentyp näher kennen
- Beispiel aus den Folien: `double neededPower`

Datentypen

Datentypen

- Datentypen bezeichnen Mengen gleichartiger Objekte
- Typ legt die möglichen Werten und Eigenschaften der Variable fest
- Typen bestimmen auch die Operationen, die auf die Variablen ausgeführt werden können

- In Java:
  - 8 primitive Datentypen
  - erweiterbar mit Klassen und Enums

Primitive Datentypen

Datentyp	Werttyp	Speicherplatz	Beispielwerte
boolean	Wahrheitswert	`8 bit`	`true, false`
char	Unicodezeichen	`16 bit`	'B', 'x', '?'
byte	Ganzzahl	`8 bit`	42
short	Ganzzahl	`16 bit`	123
int	Ganzzahl	`32 bit`	32768
long	Ganzzahl	`64 bit`	`12345678901234L`
float	Gleitkommazahl	`32 bit`	0.3F, 0.125F
double	Gleitkommazahl	`64 bit`	0.001, 1e-64

Ganzzahlen

- Überläufe
  - Treten auf, wenn die Zahlen außerhalb des zulässigen Bereichs liegen (mehr dazu in DT)
  - Mit anderen Worten: aufpassen bei großen Zahlen!
  - Zum Beispiel: 2 000 000 000 + 2 000 000 000 = **- 294 967 296**
- Nützliche Operatoren mit Ganzzahlen:
  - Postfix (meistens genutzt): y++, x--
  - Semantik: int a = x++ entspricht int a = x; x = x + 1;
  - Präfix: ++x, --y
  - Semantik: int a = ++x entspricht x = x + 1; int a = x;

Gleitkommazahlen

- Wegen der Struktur der Gleitkommazahlen kommt es häufig zu den Rundungsfehlern!
  - Sie haben eine endliche Genauigkeit
  - Mehr dazu wird im 2. und 3. Semester in "Technische Informatik"gesagt
  - Beispiel: $\sqrt{2} \cdot \sqrt{2} \neq 2$ (nicht aber mathematisch)
- Besondere Werte:
  - NaN (Not a Number)
  - POSITIVE_INFINITY
  - NEGATIVE_INFINITY
- Gleitkommazahlen nicht auf Gleichheit prüfen!

```java
double x;
double y;

if (x == y) { //sehr schlecht, kommt fast nie durch
      //...
}

double epsilon = 0.001; //gewuenschte Genauigkeit
boolean result = Math.abs(x - y) < epsilon; //viel besser!
```

Casting

- Ihr könnt die Zahlen-Datentypen aufeinander "casten"(als Variablen eines anderen Datentyps zu nutzen)
- Häufig geschieht dies mit Fehlern
- Wenn ihr in manchen Richtungen die Typen castet, so entstehen **Informationsverluste**. Informationsverlust**freie** Konversionen:

$$\texttt{byte} \rightarrow \texttt{short} \rightarrow \texttt{char} \rightarrow \texttt{int} \rightarrow \texttt{long} \rightarrow \texttt{float} \rightarrow \texttt{double}$$
$$\leftarrow \qquad \leftarrow \qquad \leftarrow \qquad \leftarrow \qquad \leftarrow \qquad \leftarrow$$

Benötigt zusätzliches (nicht impliziertes) Casten mit (Datentyp) vor dem Wert

Der Datentyp String

- Datentyp für Zeichenketten (ähnlich zu GBI)
  - Ist schon kein primitiver Datentyp
  - UTF-16 Zeichen Unterstützung
  - Maximale Länge: 2 147 483 647 Zeichen

- Vergleich mit `"=="` funktioniert nicht, denn:
  - In der Variable `String a = "..."` liegt nur die Speicheradresse
  - Beim Vergleichen mit `"=="` werden nur die "Links"verglichen
  - $\Rightarrow$ Nutzung von **equals()** erforderlich
  - Also: "something".equals("something")

Der Datentyp String

- Erzeugen

```
String a = "Something"; //String with content
String b = new String(); //Empty string
String c = ""; //Also an empty string
```

- Konkatenieren (wie in GBI aber anstatt · schreibt man +)
  - `String added = a + "-"+ c;`

- Zugriff auf Methoden: stringName.methodName()

Nützliche Methoden von String

- **`length()`**: gibt die Länge des Strings zurück
- **`substring(anfang, ende)`**: gibt Teilwort von *anfang* bis *ende* zurück
  - z.B.: "abcdefeinWortfghijk".substring(6, 13) = "einWort"
- **`trim()`**: entfernt alle Leerzeichen und andere chars <= \u0020 am Anfang
  - $"\quad \text{example Text} \quad ".\text{trim}() = "\text{example Text}"$
- **`toLowerCase()/toUpperCase()`**: gibt gleichen String in Großbuchstaben zurück
  (für ein bestimmtes Zeichen **`toLowerCase(position)`**)
- **`equalsIgnoreCase(anderesWort)`**: Vergleicht zwei Strings ohne Groß- und Kleinschreibung zu beachten
  - "CAPSLOCK".equalsIgnoreCase("capslock") = true
- **`replace(zuErsetzen, ersatz)`**: Ersetzt alle teilWörter im String mit dem Ersatzwort
  - "HeORlloOR ORwoORrld!".replace("OR", "") = "Hello world!"

Nützliche String Hilfsklassen

`StringBuilder`

- Hilft bei mehrfacher Konkatenation von Teilstrings. Man benutzt sie meistens dann, wenn mehrfache Konkatenation von String-Objekten benötigt ist
- Beispiel:

```
StringBuilder builder = new StringBuilder();
builder.append("first Word");
builder.append("second Word");
```

`StringJoiner`

- Ähnlich zu StringBuilder, aber benötigt ein angegebenes Zeichen, das bei jeder Konkatenation zwischen den letzten zwei "Wörtern"eingesetzt wird

Datentyp Enum

- Ein Enumerationstyp bietet die Möglichkeit, einen Satz bennanter Konstanten gleicher Obermenge zu definieren
- Das `enum`-Schlüsselwort wird zum Deklarieren einer Enumeration verwendet
- Jedes Aufzählungsobjekt erbt von der Spezialklasse Enum und sind im Grunde nicht andres als statische Objekte

```
enum Name {/*List of values separated by a comma*/}
```

Beispiel für enum

- Quelltext

```
public class EnumTest {
    public static void main(String[] args) {
        Day currentDay;
        currentDay = Day.FRIDAY;
        System.out.println(currentDay);
        System.out.println(currentDay = Day.FRIDAY);
    }
}
enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURSDAY
}
```

Datentypen Übersicht

- In Java üblichen Datentypen:
  - **boolean** für Aussagenlogik
  - **char** für einzelne Zeichen
  - **int** für Ganzzahlen
  - **double** für Gleitkommazahlen
  - **enum** für Aufzählungen
  - **String** für Zeichenketten
- Bei Rechneroperationen verwendet der Java-Compiler standartmäßig
  - **int** für alle Untertypen von Ganzzahlen
  - **double** für Gleitkommazahlen
- Um ein Literal des Typs **long** oder **float** zu verwenden, muss ein **'L'** oder **'F'** hinter der Zahl angehängt werden

Übersicht Datentypen

Problem

- Wie kann ich meine eigene Datentypen definieren?
- Wie werden dieses Datentypen beschrieben?

Lösung

- Neue Datentypen werden mit neuen Klassen definiert
  - Beispiel: `String`-Klasse
- Operationen auf den neuen Datentyp werden mit Methoden dieser Klasse definiert

Hinweis

Das var-Schlüsselwort

In den letzten Java-Versionen gibt es das Keyword **var**. Dieser ersetzt alle Datentypen (bspw. kann eine Variable mit "`var myVariable = 5;`" definiert)
Obwohl es scheint schneller und leichter auf diese Weise Code zu schreiben, ist es als unschön und wird als schlechter Stil gesehen $\rightarrow$ Punktabzug bei Übungsblättern und Abschlussaufgaben

Operatoren in Java

Arithmetische und Vergleichs-Operationen

Präzedenz	Operator	Beschreibung

1	`+ -`
	`++x, x++`
	`-x, x-`
		Vorzeichen
		Inkrement
		Dekrement

2	`x * y`
	`x / y`
	`x % y`
		Multiplikation
		Division
		Modulo

3	`x + y` `x - y`	Addition, Subtraktion

5	`x < y`
	`x <= y`
	`x > y`
	`x >= y`
		Größenvergleiche

6	`x == y` `x != y`	Gleichheit, Ungleichheit

Operationen auf Ganzenzahlen

Präzedenz	Operator	Beschreibung
1	$\sim$x	Bitweises Komplement (NOT)
4	x << y	Linksshift
	x >> y	Rechtsshift (mit Vorzeichen)
	x >>> y	Rechtsshift (ohne Vorzeichen)

Operationen auf Wahrheitswerten

Präzedenz	Operator	Beschreibung
7	x & y	Bitweises Und (AND)
8	x ^ y	Bitweises Entweder-Oder (XOR)
9	x | y	Bitweises Oder (OR)
1	!x	Negation (NOT)
10	x && y	Sequenzielles Und (AND)
11	x || y	Sequenzielles Oder (OR)

Übersicht Operatoren

Präzedenz	Operator	Bedeutung
1	++	Prä- oder Postinkrement
	--	Prä- oder Postdekrement
	+ -	Vorzeichen
	~	bitweises Komplement
	!	logische Negation
	(Type)	Typumwandlung
2	* / %	Multiplikation, Division, Rest
3	+ -	Addition, Subtraktion
	+	String-Verkettung
4	<<	Linksschieben
	>>	Rechtsschieben, Vorzeichenbit nachziehen
	>>>	Rechtsschieben, Nullen nachziehen
5	< <=	kleiner, kleiner oder gleich
	> >=	größer, größer oder gleich
6	==	gleich
	!=	ungleich
7	&	bitweises Und
8	^	logisches Exklusiv-Oder
	^	bitweises Exklusiv-Oder
9	|	bitweises Oder
10	&&	logisches Und
11	||	logisches Oder
12	=	Wertzuweisung
12	*= /= %=	kombinierte Wertzuweisung
	+= -=	
	<<= >>= >>>=	
	&= ^= |=	

Präzedenz

- Je kleiner der Wert, desto stärker bindet der Operator!
- Im Zweifel der Präzedenz nutzt Klammern!
- Leerzeichen spielen *keine* Rolle

Division und Modulo

- Bei der Division mit Ganzzahlen wird alles ab dem Komma abgeschnitten
- Ganzzahlen
  - $4 / 3 = 1$
  - $1 / 3 = 0$
- Gleitkommazahlen
  - $4.0 / 3.0 = 1.3333333333333333$
  - $1.0 / 3.0 = 0.3333333333333333$
- Ganzzahlen
  - $4 \,\%\, 3 = 1$
  - $1 \,\%\, 3 = 1$
- Gleitkommazahlen
  - $4.0 \,\%\, 3.0 = 1.0$
  - $1.0 \,\%\, 3.0 = 1.0$
- Die Division durch Null erzeugt meist einen Laufzeitfehler
  - $1 / 0$ löst Programmabsturz aus (ArithmeticException)
  - $1.0 / 0.0$ wird zu Double.POSITIVE_INFINITY
  - $-1.0 / 0.0$ wird zu Double.NEGATIVE_INFINITY
  - $0.0 / 0.0$ wird zu Double.NaN

Übung zu Präzedenz

Was ergibt . . .?

$10 * 5 + 2$	$\rightarrow 62$
$(\text{double})(5 + 8)/10$	$\rightarrow 1.3$
$(5 + 8)/10$	$\rightarrow 1$
1^2	$\rightarrow 3$
true && $-9 > 10$	$\rightarrow$ false
$1 < 2 \,\|\| 6 + 9$	$\rightarrow$ Compiler-Fehler
!(false && true)	$\rightarrow$ true
1 & 14	$\rightarrow 0$
$3 = 5$ && true	$\rightarrow$ Compiler-Fehler
$(2 * 10)$ % 5	$\rightarrow 0$
$(5 + 4$ / 9	$\rightarrow$ Compiler-Fehler

Fehlende Operationen

- Java unterstützt keine schwierigen mathematischen Operationen . . .
  - . . .oder doch?
- Lösung: das Math-Package
  - Dadrin könnt ihr viele Operationen wie Math.sqrt(double), Math.round(double) finden
  - Außerdem sind π und e als Math.PI und Math.E definiert

Variablen

Variablen und Datentypen

- In Java haben alle Variablen einen Datentyp
- Der Datentyp muss zur Compile-Zeit bekannt sein
- Der Typenprüfung wird vom Compiler und von der JVM überprüft
- Die erlaubten Operationen hängen vom jeweiligen Datentyp ab
- Variablen sind Platzhalter für Werte eines Datentyps

Variablen und Zuweisungen

- **Deklaration:** Legt Name und Typ fest
  - `int x;`
- **Zuweisung:** Setzen eines Wertes
  - `x = 4;`
- **Initialisierung:** Kombination von Deklaration und Zuweisung
  - `int y = 2;`
- **Änderung:** Setzen eine (neuen) Wertes
  - `int z = x * 10 + y;`
  - `z = -1 * z;`
  - `Z++;`

Konstanten Deklaration

- Analog zu Variablendeklaration:
- Für lokale Variablen
  - Verwendung des Zusatzes "final"
  - Schema: **`final Typ name = Wert;`**

Beispiel:

```
public static void foo() {
final String data = "Hello World!";
System.out.println(data);
}
```

Konstant Deklaration

- Klassenkonstanten
  - unterliegen der Konvention Großbuchstaben zu verwenden
  - Schema: `static final Typ NAME = Wert;`

Beispiel:

```
public static void foo() {
static final float GRAVITATION_EARTH = 9.80665f;
static final float PI = 3.14159265f;
}
```

null-Wert

In Java existieren zwei Typen der Daten, die gespeichert werden:

- Die primitiven: sie werden in ihren eigenen "Speicherzellen" gespeichert
- Und die "Referenz"-Datentypen: sie werden nur referenziert
  - in diesen Variablen wird nur der Link zu dem tatsächlichen Wert gespeichert
- Daher existiert der sogenannte null-Wert. Objektvariablen können ihn annehmen
  - z.B.: `Object obj = null;`

- Die Referenz `null` steht für "nichts gespeichert" oder "kein Objekt"
  - $\rightarrow$ kein Zugriff auf die Eigenschaften oder Methoden über diese Variable
  - es können auch NullPointerExceptions auftreten

- Die Objektidentität wird dann erst mit dem Aufruf von `new` erstellt

Variablenüberdeckung

Erlaubt (wenn auch unschön)

```
Class MyClass {
    int number = 0;
    public void method() {
        int number;
        number = 3;
        print(number);
    {
}
```

Nicht erlaubt

```
Class MyClass {
    int number;
    String number;    public void
method() {
        int word;
        String word;
    {
}
```

Was ist richtig?

- int a = 9; → "9"
- b = 4; → Compiler-Fehler
- int c, d; → c und d nur deklariert
- boolean true = 1; → Compiler-Fehler
- a = 3; → "3"
- boolean x = (a == b); → "false"
- (boolean y = a) == b; → Compiler-Fehler
- boolean z = (true == false); → "false"
- char letter1 = a; → Compiler-Fehler
- char letter2 = 9; → "	" (Tabstopp)
- char letter3 = (char) a; → "	" (Tabstopp)
- char letter4 = '9'; → "9"