

# Klassendefinitionen II

## Lernziele

- Sie können mit Operatoren und Ausdrücken richtig umgehen.
- Sie setzen bedingte Anweisungen korrekt in Ihren Programmen ein.
- Sie können Klassen basierend auf einer Spezifikation und unter Berücksichtigung der Clean Code-Regeln programmieren und diese in BlueJ testen. Dabei implementieren Sie die Methoden so, dass die Parameter auf deren Gültigkeit überprüft werden.

## Aufgabe 1 (auf Papier)

Führen Sie den untenstehenden Code nicht aus. Lösen Sie die Aufgabe auf Papier. In Aufgabe 2 werden Sie Ihre Lösung dann verifizieren können.

Um korrekte Java Programme zu schreiben, ist der Umgang mit Operatoren und Ausdrücken ganz wichtig. Diese Aufgabe bietet Ihnen die Möglichkeit, Ihre Kenntnisse diesbezüglich zu überprüfen. Schreiben Sie bei allen Ausgaben hin, was genau ausgegeben wird, wenn die Methode `ausgeben` aufgerufen wird. Seien Sie präzise und unterscheiden Sie zwischen ganzzahligen und Gleitkommazahlen, indem Sie z.B. 4 oder 4.0 angeben. Falls Sie bei einer Zeile einen Kompilierfehler vermuten, so geben Sie dies bitte ebenfalls an.

```
public void ausgeben() {
    int int1 = 1, int2 = 2, int3 = 3;
    double double1 = 3.0, double2 = 4.0;
    boolean boolean1 = true;

    System.out.println(int1 + int2 + int3--);
    System.out.println(int3);
    System.out.println(--int3);

    int3 = 3;
    System.out.println(int1 + int2 * int3);
    System.out.println((int1 + int2) * int3);

    System.out.println(int1 / int2 + ", " + int3 / int2);
    System.out.println(int1 / double1 + ", " + double2 / int3);
    System.out.println(int1 / 4 + ", " + int1 / 4.0);

    System.out.println(int2 / int3 * double1);
    System.out.println(int2 / (int3 * double1));

    System.out.println(9 % 4 + ", " + 9.0 % 4);

    System.out.println("Ein Hund hat " + 2 + 2 + " Beine");
    System.out.println("Ein Hund hat " + (2 + 2) + " Beine");
    System.out.println("Zwei Hunde haben " + 2 * 4 + " Beine");

    System.out.println(int1 == int2);
    System.out.println(int1 == int3 / int2);
}
```

6  
2  
1  
7  
9  
0, 1  
0.333, 1.333  
0, 0.25  
0.0  
0.2222222222222222  
1, 1.0  
Ein Hund hat 22 Beine  
Ein Hund hat 4 Beine  
Zwei Hunde haben 8 Beine  
false  
true

displayed 16x 3's here

```
System.out.println((int3 <= 3) && (double2 <= 2.999999)); false
System.out.println((int3 < 3) || !(double2 <= 2.999999)); true
System.out.println((int3 <= 3) && !(double2 <= 2.999999)); true

System.out.println(boolean1 = 7 > 6); true
System.out.println(int3 = int1 + int2 == 3); false
System.out.println((int3 = int2 - int1) == 1); false

System.out.println(3 * 1000000); 3000000
System.out.println(3 * 10000000); 30000000
System.out.println(3 * 100000000); 300000000
System.out.println(3 * 1000000000); 3000000000
System.out.println(3 * 1000000000L); -1294967296
System.out.println(3 * 1000000000d); 3.0E9
}
```

compile error

true

-1294967296

3.0E9

## Aufgabe 2

Forken Sie das Projekt: [https://github.engineering.zhaw.ch/prog1-kurs/02\\_Praktikum-2\\_Ausdruck](https://github.engineering.zhaw.ch/prog1-kurs/02_Praktikum-2_Ausdruck).  
Nutzen Sie BlueJ um die eigene Projektkopie auf Ihren Computer zu holen und zu bearbeiten.

Verifizieren Sie Ihre Ausgaben aus Aufgabe 1, indem Sie ein Objekt der Klasse `Ausdruck` erzeugen und die Methode `ausgeben` aufrufen. Haben Sie irgendwo einen Fehler gemacht? Wenn ja, so versuchen Sie genau zu verstehen, wieso Sie falsch lagen.

```
6
2
1
7
9
0, 1
0.3333333333333333, 1.3333333333333333
0, 0.25
0.0
0.2222222222222222
1, 1.0
Ein Hund hat 22 Beine
Ein Hund hat 4 Beine
Zwei Hunde haben 8 Beine
```

```
false
true
false
true
true
true
true
3000000
30000000
300000000
-1294967296
3000000000
3.0E9
```

### Comments:

- `System.out.println((int3 = int2 - int1) == 1);` is true because the variable 'int3' gets assigned to '1', which is equal to 1.
- `3 * 1000000000d` prints out 3.0E9 instead of the whole number
- `3 * 1000000000` prints out -1294967296 for some reason. I do not know why...

## Aufgabe 3

Forken Sie das Projekt: [https://github.engineering.zhaw.ch/prog1-kurs/02\\_Praktikum-2\\_Auto](https://github.engineering.zhaw.ch/prog1-kurs/02_Praktikum-2_Auto).  
Nutzen Sie BlueJ um die eigene Projektkopie auf Ihren Computer zu holen und zu bearbeiten.

Ein Autohändler möchte ein Programm für die Verwaltung seines Lagerbestandes an Neufahrzeugen entwickeln lassen. Ihre Firma erhält den Auftrag und Sie erhalten die Aufgabe, die Klasse Auto zu entwickeln. Implementieren Sie die Klasse gemäss folgender Spezifikation:

- Ein Auto hat eine Marke (z.B. Toyota oder BMW), einen Typ (z.B. Prius oder 320), einen Hubraum in Litern (z.B. 1.8) und einen Motor mit oder ohne Turbo. Zudem hat ein Auto einen Lagerbestand.
- Ein Auto mit Lagerbestand 0 wird erzeugt durch Angabe von Marke, Typ, Hubraum und ob es einen Turbomotor hat. Dabei gelten folgende Regeln:
  - Marke und Typ haben mindestens 3 und höchstens 10 Zeichen. Wird ein ungültiger Wert angegeben, so wird der Wert \_\_\_\_ (drei Tiefstriche) verwendet und eine aussagekräftige Fehlermeldung ausgegeben.
  - Der Hubraum liegt im Bereich 0.5 bis 8 Litern. Wird ein ungültiger Wert angegeben, so wird 0 verwendet und eine aussagekräftige Fehlermeldung ausgegeben.

Bemerkung: Eigentlich möchten wir hier verhindern, dass „ungültige“ Autos erzeugt werden können. Dazu benötigt man aber noch weitere Konzepte, die Sie zu einem späteren Zeitpunkt kennenlernen werden.

- Für Marke, Typ, Hubraum und Turbomotor gibt es jeweils eine Methode, um den Wert zu setzen. Wird ein ungültiger Wert gesetzt, so wird der alte Wert belassen und es wird eine Fehlermeldung ausgegeben.
- Es gibt eine Methode, um den Bestand zu ändern. Der Bestand darf maximal um 10 (negativ oder positiv) geändert werden und darf nicht negativ werden, ansonsten wird eine Fehlermeldung ausgegeben. Nach erfolgter Änderung wird der alte und neue Bestand ausgegeben.
- Es gibt eine Methode, um ein Auto auszugeben. Die Ausgabe soll gemäss den folgenden Beispielen aussehen, wobei sich der Code aus den jeweils ersten drei Zeichen von Marke und Typ, dem Hubraum und optional einem t bei einem Turbomotor zusammensetzt:

```
Mitsubishi Colt, 1.4 Liter  
Code: Mit-Col-1.4  
Lagerbestand: 4
```

```
BMW 330i, 3.0 Liter turbo  
Code: BMW-330-3.0-t  
Lagerbestand: 1
```

Hinweis: `System.out.print(String s)` gibt einen String s ohne Zeilenumbruch aus.

Achten Sie auf die Clean Code-Regeln. Duplizieren Sie keinen Code und achten Sie auf gute Namensgebung. Lagern Sie Teilfunktionen wie z.B. die Berechnung des Codes in eine Methode aus, um das Programm möglichst gut lesbar zu machen. Testen Sie die Klasse nach der Implementierung um sicherzustellen, dass sie korrekt funktioniert und alle Anforderungen erfüllt sind.

## Aufgabe 4 (optional)

Forken Sie das Projekt: [https://github.engineering.zhaw.ch/prog1-kurs/02\\_Praktikum-2\\_Konto](https://github.engineering.zhaw.ch/prog1-kurs/02_Praktikum-2_Konto).  
Nutzen Sie BlueJ um die eigene Projektkopie auf Ihren Computer zu holen und zu bearbeiten.

Im ersten Praktikum haben Sie eine Klasse die ein Bankkonto modelliert, implementiert. Kopieren Sie Ihre Konto-Klasse aus dem ersten Praktikum in dieses Projekt.

Ihre Aufgabe ist es, diese Klasse so zu erweitern, dass fehlerhafte Eingaben erkannt und zurückgewiesen werden. Die Anforderungen lauten wie folgt:

- Der Inhaber eines Kontos muss mindestens 3 und höchstens 20 Zeichen beinhalten. Wenn ein Konto erzeugt wird und dies nicht zutrifft, so wird eine geeignete Fehlermeldung ausgegeben und der Kontoinhaber wird auf „no name“ gesetzt.
- Beim Erzeugen eines Kontos muss der Zinssatz mindestens 0 und höchstens 10 (Prozent) sein. Ist dies nicht der Fall, wird eine geeignete Fehlermeldung ausgegeben und der Zinssatz auf 0 gesetzt.
- Wird der Zinssatz nachträglich geändert, muss er ebenfalls im Bereich 0 bis 10 liegen. Ist dies nicht der Fall, wird eine Fehlermeldung ausgegeben und der Zinssatz nicht verändert.
- Beim Ein- und Auszahlen dürfen nur positive Beträge, maximal aber 10'000 verwendet werden. Ist dies nicht der Fall, wird eine Fehlermeldung ausgegeben und die Ein- bzw. Auszahlung nicht vorgenommen.
- Es darf nicht mehr Geld abgehoben werden, als sich auf dem Konto befindet. Wird dies versucht, so wird eine Fehlermeldung ausgegeben, die auch den aktuellen Kontostand angibt. In diesem Fall wird nichts ausbezahlt.

Achten Sie auch bei dieser Aufgabe darauf, dass Sie keinen Code duplizieren. Sie sollten also nicht einfach bei jedem Konstruktor bzw. Methode direkt den Code zur Fehlererkennung und -behandlung hinzufügen, sondern dies nach Möglichkeit nur einmal implementieren. Beim Betrag könnte es z.B. sinnvoll sein, dass Sie eine private Methode `istBetragGuelutig(...)` schreiben, welche den Betrag prüft und bei Bedarf eine Fehlermeldung ausgibt.

Testen Sie die Klasse nach der Implementierung um sicherzustellen, dass sie korrekt funktioniert und alle Anforderungen erfüllt.