# Java Übungsskript

**Teilnehmer:** .....

**Trainer/in:** .....

# Inhaltsverzeichnis

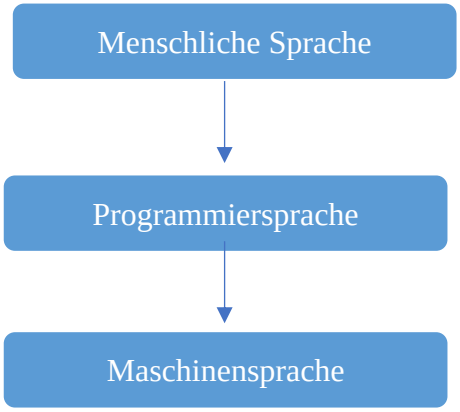
# 1) Einleitung

## 1.1) Grundbegriffe des Programmierens

- Computer (deutsch oft: Rechner): technisches Gerät zur Berechnung, Speicherung, automatischen Verarbeitung von Informationen (allg. Daten).
- Algorithmus: Rechenvorgang nach einem bestimmten (sich wiederholenden) Schema
- Programm: spezielle Handlungsanweisung(en) die für einen Computer verständlich ist/ sind. Ein in Programmiersprache formulierter Algorithmus.
- Hardware: Physische, greifbare Komponenten (z.B.: Prozessor, Grafikkarte, Arbeitsspeicher, …) eines Geräts (z.B. Computer, Monitor, …).
- Software: (im Unterschied zur Hardware) nicht technisch-physikalischer Funktionsbestandteil einer Datenverarbeitungsanlage (wie z. B. Betriebssystem und andere [Computer]-programme).

## 1.2) Was ist eigentlich Programmieren?

Umgangssprachlich spielt man auf einen Computer ein Programm. Dafür müssen einige Bedingungen erfüllt sein wie z.B. eine Übersetzung zwischen menschlicher Sprache (Deutsch, Englisch, etc) auf die Sprache des Computers (Maschinensprache).




Genau hier finden die Programmiersprachen ihre Anwendung. Mithilfe eines sogenannten **Compilers** können Befehle einer Programmiersprache in Maschinensprache umgewandelt werden. Der Compiler muss mit konkreten Informationen gefüttert werden. Für unterschiedliche Programmiersprachen gibt es also jeweils zugehörige Compiler welche von Programmierern geschriebene Programme in Anweisungen für den Computer umwandeln. Beispiele für Programmiersprachen wären z.B.: Java, C, C#, Python, Assembler, uvm… .

## 1.2.1) Vom Java Code zum ausführbaren Programm

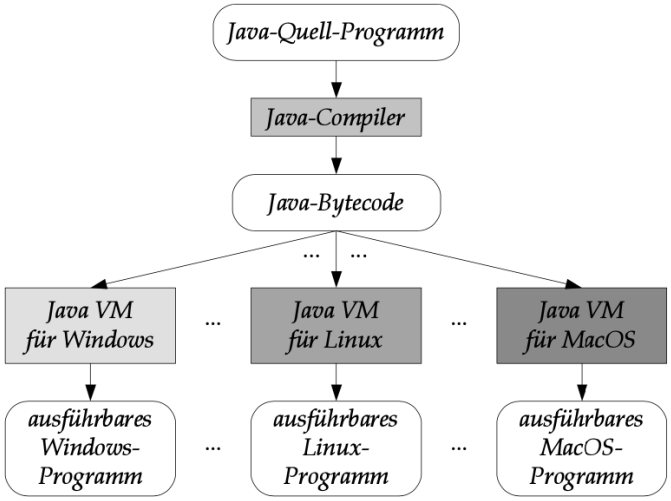



*Abbildung 1: Code zu Programm*

## 1.2.2) Was benötigen wir zum Programmieren?

Um uns das Programmieren von Anwendungen zu erleichtern, gibt es für Java eine sogenannte **IDE** (engl. Integrated Development Environment) (deutsch. Integrierte Entwicklungsumgebung). Solche IDEs umfassen meist alle wichtigen Komponenten wie **Texteditor, Compiler** und **Interpreter**. Für die Codersbay ist (Stand Feb. 2022) JetBrains IntelliJ zum Erstellen aller Applikationen vorgesehen,

Download unter: https://www.jetbrains.com/de-de/idea/download/

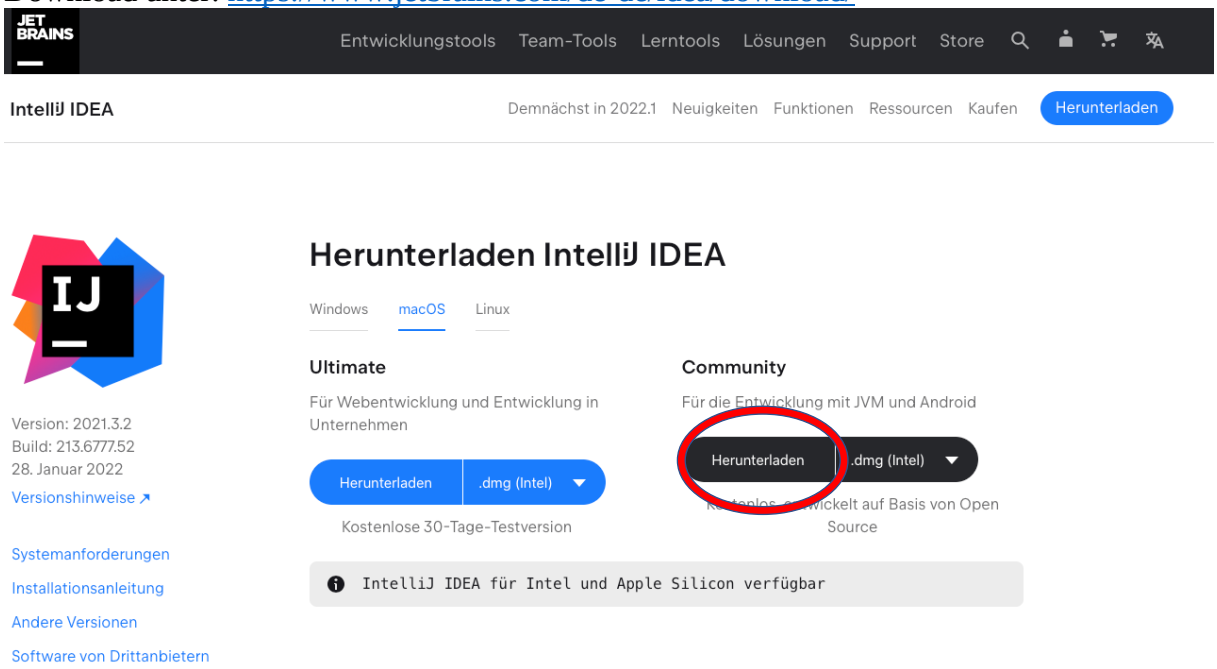



*Abbildung 2: Download Community Edition - JetBrains IntelliJ*

Anleitung zu IntelliJ:
www.youtube.com%2Fwatch%3Fv%3Dc0efB_CKOYo&usg=AOvVaw14yrqfr9LJc7185elMF1Q6

# 2) Erste Schritte in Java

## 2.1) Hello World

Das vermutlich bekannteste Programm unter Entwicklern ist „Hello World“. Es soll die Grundbausteine dafür schaffen, was in den kommenden Kapiteln folgt. Dazu muss ein neues **Projekt** erstellt werden. Unter *New Project* kann ein solches Projekt relativ einfach angelegt werden.

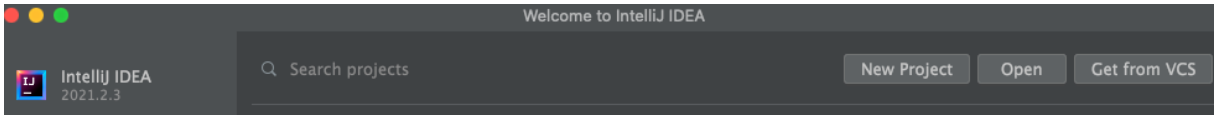



*Abbildung 3: Welcome to IntelliJ IDEA*

Wichtig: Stelle sicher, dass eine „Project SDK“ ausgewählt ist. Sollte noch keine ausgewählt sein, oder installiert sein drücke einfach Download **JDK**. In Zukunft sollte dieser Schritt übersprungen werden können. JDK steht für Java Development Kit

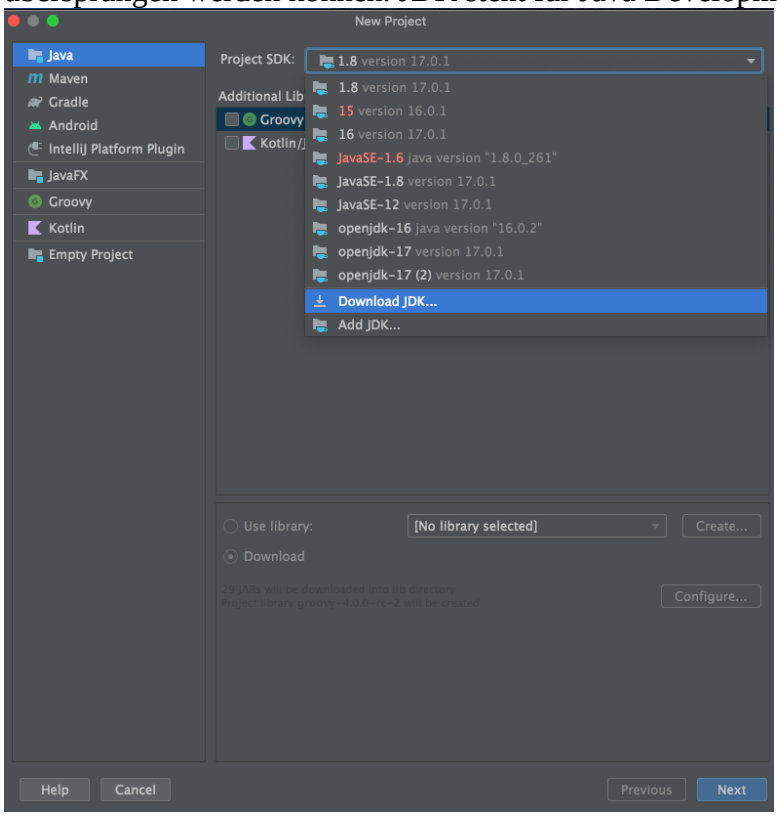



*Abbildung 4: SDK/ JDK herunterladen*

Anschließend muss zweimal *Next* gewählt werden für den nächsten Schritt die Namensvergabe.

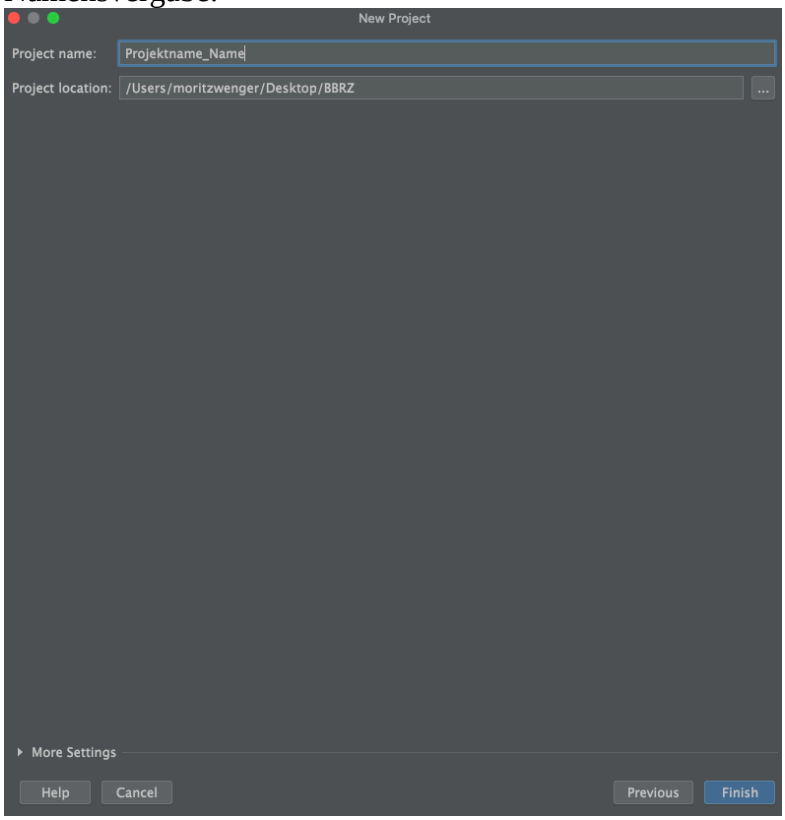



*Abbildung 5: Projekt benennen*

Nachdem der Projektname und der Speicherort gewählt wurde z.B.: HelloWorld_Mustermann kann mithilfe des *Finish* Buttons der Vorgang abgeschlossen werden.
IntelliJ erstellt für jedes Projekt einen eigenen Ordner in welchem sich ein weiterer wichtiger Ordner mit dem Namen *src* befindet. Darin befindet sich dann unsere **Klasse**. Diese wird mit *rechter Maustaste auf src – new – Java Class* erstellt. Der Name der Klasse sollte sinnvoll gewählt werden (z.B.: HelloWorld).

Verbotene Zeichen sind:

- Leerzeichen
- Umlaute
- Sonderzeichen

```
public class Main {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

*Abbildung 6: HelloWorld Programm. Der Name der Klasse ist hier "Main".*

Die geschweiften Klammern kann man sich als Start und Endpunkte der jeweiligen Klasse oder **Methode** vorstellen. Eine Methode ist eine Ansammlung aus Befehlen (Code) welcher immer wieder ausgeführt werden kann. Eine der wichtigsten Methoden für den Anfang ist die Main Methode. Diese beinhaltet das Hauptprogramm und später auch Verweise auf unsere selbst erstellten Methoden. Um die Main Methode etwas schneller einzufügen kann die

automatische Vervollständigung von IntelliJ verwendet werden. Um diese zu nutzen muss ein Stichwort (z.B.: main) eingegeben werden und mithilfe der Tabulator Taste (TAB) wird der Rest automatisch eingefügt.

Wenn nun das Programm ausgeführt wird (grüner Start Knopf) sollte auf der Konsole (unten) Hello World ausgegeben werden.

### 2.1.1) Aufgabe 1:

Erstelle ein neues Java Projekt mit dem Projektnamen HelloWorld (Achte darauf, dass dein Projektname dem obigen Schema entspricht). Erstelle in deinem neuen Projekt unter /src eine neue Java Klasse mit dem Namen HelloWorld.class. Füge nun den Code aus Abbildung 6 in deine Klasse ein und Teste dein Programm.

# 2.2) Syntax

Im vorherigen Kapitel haben wir schon unser erstes eigenes Programm erstellt (Abbildung 6).

## 2.2.1) Erklärung

Jede Codezeile welche wir in Java programmieren wollen muss sich in einer Klasse befinden. In unserem Beispiel haben wir die Klasse HelloWorld_Mustermann getauft (in der Abbildung 6 heißt die Klasse Main). Eine Klasse wird immer mit einem Großbuchstaben als ersten Buchstaben begonnen.
**Achtung:** Java ist sehr empfindlich wenn es um Groß- und Kleinschreibung geht. *MeineKlasse* und *meineklasse* sind zwei völlig verschiedene Klassen. Der Name der Java Datei MUSS mit dem Klassennamen 1:1 übereinstimmen, ansonsten funktioniert das **kompilieren** nicht!

Kompilieren ist das was der Compiler macht, also das Übersetzen von Code in Maschinensprache.

## 2.2.2) Die Main Methode

Die main() Methode ist unumgänglich und man sieht sie in jedem Java Programm. Jeder Code der sich in der Main Methode befindet wird auf jeden Fall ausgeführt. Die Schlüsselwörter public, static, void, sowie String[] args müssen zu diesem Zeitpunkt im Skript noch nicht verstanden werden.

Für jetzt ist es wichtig: Jedes Java Programm hat einen Klassennamen, welcher dem Dateinamen entspricht und eine Main Methode.

## 2.2.3) System.out.println()

In der Main Methode können wir System.out.println() verwenden, um einen Text auf der Konsole auszugeben.

## 2.2.4) Merke:

- Die geschwungenen Klammern {} markieren Anfang und Ende eines Codeblocks.
- Jedes Code Statement muss mit einem Semicolon ; beendet werden.

## 2.3) Aufgabe 2:

Recherchiere zum Thema Syntax und fasse deine Erkenntnisse in einer kurzen Zusammenfassung von 150 – 250 Wörtern zusammen. Erstelle anschließend eine Präsentation zum Thema: Syntax in Java

Geh dabei auf folgende Unterpunkte besonders ein:

- Was ist Syntax?
- Wie unterscheidet sich Syntax zu Semantik?
- Erkläre die Zeichen:
  - {}
  - .
  - ;
- Der Begriff „case-sensitive“

Lies dir folgenden Artikel durch:
https://www.developer.com/design/top-10-java-coding-guidelines/

# 3) Java Kommentare

Java Kommentare oder auch Comments genannt werden verwendet, um den geschriebenen Code zu beschreiben und zu dokumentieren. Das verbessert einerseits die Leserlichkeit des Codes und hilft euch und anderen dabei das programmierte besser zu verstehen. Weiterführend können Kommentare verwendet werden, um Teile des Programms NICHT auszuführen.

Dies ist unter anderem für Testzwecke von Vorteil, wenn man nur bestimmte Teile des Codes ausführen möchte, dann kommentiert man einfach das aus, was man nicht braucht.

## 3.1) Single Line Comments

Einzeilige Kommentare werden in Java mit // angeführt. Alles was in dieser Zeile auf // folgt wird vom Compiler nicht beachtet.

```
// This is a comment
System.out.println("Hello World");
```

*Abbildung 7: Beispiel 1 für Single Line Comments*

```
System.out.println("Hello World"); // This is a comment
```

*Abbildung 8: Beispiel 2 für Single Line Comments*

## 3.2) Multi Line Comments

Mehrzeilige Kommentare werden in Java mit /* begonnen und enden mit */.

```
/* The code below will print the words Hello World
to the screen, and it is amazing */
System.out.println("Hello World");
```

*Abbildung 9: Beispiel für Multi Line Comments*

## 3.3) Comment-Header

Alle Klassen die wir erstellen bekommen von uns einen Comment-Header. Dieser sieht folgendermaßen aus:

```
/*
* Author:
* Date:
* Version:
* Description:
*/
```

Unter Author wird der Name des Erstellers/der Erstellerin verzeichnet.
Date = Erstellungsdatum
Version = Versionsnummer (z.B.: 0.1 oder 1.0.2)
Description: Kurze Beschreibung der Klasse
Optional:
Last modified = zuletzt bearbeitet am:

# 3.4) Aufgabe 3:

Füge deiner HelloWorld Klasse aus Aufgabe 1 Kommentare hinzu.
Achte hierbei auf den Comment-Header.
Wenn du deine Methoden beschriften möchtest (was du solltest) versuch 1 Zeile über der Methode:
/** - ENTER
Dies sollte einen Kommentarblock für diese Methode erstellen.
Dokumentiere mithilfe einzeiliger Kommentare deinen Code.

# 4) Variablen und Datentypen

Variablen sind erstellbare und veränderbare Container oder auch Behälter für Datenwerte. Die Referenz auf den sich in der Variable befindenden Wert ist deren Name. Dadurch kann man eine Variable an beliebigen Orten im Code und auch beliebig oft einsetzen, indem man einfach den Namen der Variable angibt auf welche man zugreifen möchte.
In Java gibt es verscheiden Datendypen z.B.:

- String – speichert Text wie z.B. „Hallo“. Strings beginnen mit “ und enden mit “.
- int – speichert Integer (Ganzzahlen) wie z.B.: 456 oder -456
- float – speichert Gleitkommazahlen wie z.B.: 12.3 oder -12.3
- chat – speichert Zeichen wie z.B.: ‘a‘ oder ‘B‘. Chars beginnen mit ‘ und enden mit ‘.
- boolean – speichert Zustandswerte wie true oder false.

## 4.1) Variablendeklaration (Variablenerstellung)

Um eine Variable zu erstellen, muss man zuerst den Typ, anschließend den Namen spezifizieren. Zuletzt kann man einen Wert zuweisen.
**Syntax:**

```
type variableName = value;
```

*Abbildung 10: Syntax der Variablendeklaration*

Das = Zeichen wird verwendet um der Variable einen Wert zuzuweisen.

### 4.1.1) Beispiele

Erstellen einer Variable welche einen Text speichern soll und diesen auf der Konsole ausgibt.

```
String name = "John";
System.out.println(name);
```

*Abbildung 11: String Variable*

Erstellen einer Variable welche eine Ganzzahl speichert und diesee auf der Konsole ausgibt.

```
int myNum = 15;
System.out.println(myNum);
```

*Abbildung 12: int Variable*

Erstellen einer Variable ohne direkter Wertzuweisung. Zuweisung erfolgt zu einem späteren Zeitpunkt.

```
int myNum;
myNum = 15;
System.out.println(myNum);
```

*Abbildung 13: int Variable – nicht direkte Wertzuweisung*

Das zuweisen eines neuen Wertes auf eine bereits existierende Variable überschreibt den vorherigen Wert:

```
int myNum = 15;
myNum = 20;  // myNum is now 20
System.out.println(myNum);
```

*Abbildung 14: Überschreiben des Wertes einer Variable*

### 4.1.2) Das final-Keyword

Eine Variable die mit dem Schlüsselwort final initialisiert ist wird auch Konstante gennant. Dies wird verwendet um andere (oder dich selbst) daran zu hindern, dass der Wert überschrieben/geändert wird. Daraus wird aus der Variable eine sogenannte read-only-Variable.

```
final int myNum = 15;
myNum = 20;  // will generate an error: cannot assign a value to a final variable
```

*Abbildung 15: Final in Verwendung*

### 4.1.3) Andere Typen

Hier ein kurzes Beispiel für das deklarieren von Variablen mit anderen Datentypen:

```
int myNum = 5;
float myFloatNum = 5.99f;
char myLetter = 'D';
boolean myBool = true;
String myText = "Hello";
```

*Abbildung 16: Variablendaklaration verschiedener Typen*

### 4.1.4) Anzeigen von Variablen

Eine häufig verwendete Methode um Variablen am Monitor anzuzeigen ist die *println()* – Methode.
Um darin Text und Variablen miteinander zu verbinden kann das + - Zeichen verwendet werden.

```
String name = "John";
System.out.println("Hello " + name);
```

*Abbildung 17: Anzeigen von Variablen*

Außerdem kann das + auch verwendet werden um 2 Variablen miteinander zu verbinden.

```
String firstName = "John ";
String lastName = "Doe";
String fullName = firstName + lastName;
System.out.println(fullName);
```

*Abbildung 18: Strings miteinander verbinden*

Für numerische Werte fungiert das + als mathematischer Operator. Hierbei werden die beiden numerischen Variablen (x und y) miteinander addiert.

```
int x = 5;
int y = 6;
System.out.println(x + y); // Print the value of x + y
```

*Abbildung 19: Addition*

### 4.1.5) Deklarieren mehrerer Variablen

Um mehrere Variablen vom **GLEICHEN TYP** zu deklarieren, kann man eine komma-geteilte-Liste verwenden. Hier ein Beispiel:

```
int x = 5, y = 6, z = 50;
System.out.println(x + y + z);
```

*Abbildung 20: Deklaration mehrerer Variablen vom selben Typ*

## 4.2) Java Identifiers

Alle Java Variablen müssen mit eindeutigen Namen identifizierbar sein. Diese eindeutigen Namen werden auch **Identifier** genannt.

Identifier können kurze Namen ( x und y ) oder detailliertere, beschreibende Namen ( wie alter, ergebnis, kontostand, totalVolume ) haben.

```
// Good
int minutesPerHour = 60;

// OK, but not so easy to understand what m actually is
int m = 60;
```

*Abbildung 21: Java Identifier Gut VS. Schlecht*

**REGELN:**

- Namen dürfen Buchstaben, Ziffern, Unterstriche & Dollar Zeichen beinhalten (A1_$)
- Namen müssen mit einem Buchstaben beginnen!
- Namen sollten mit einem Kleinbuchstaben beginnen und dürfen kein Leerzeichen beinhalten!
- Namen können in bestimmten Fällen auch mit $ und _ beginnen.
- Namen sind anfällig auf Fehler bei der Groß- und Kleinschreibung (myVar und myvar sind verschiedene Variablen).
- Reservierte Wörter (wie int oder boolean) dürfen nicht als Namen verwendet werden.

# 4.3) Aufgabe 4:

Erstelle ein neues Java-Projekt mit dem Projektnamen: VariablenUebung.java
Erstelle in deiner Klasse nun für jeden primitiven Datentypen eine Variable und weis ihr direkt einen dir überlassenen Wert hinzu. Gib nun mittels Variablenzugriffs die Werte deiner Variablen Stück für Stück auf der Konsole aus.

Der Output sollte in etwa folgendem Schema entsprechen:
<Variablenname> hat den Wert: <wert>

# 5) Datentypen

## 5.1) Primitive Datentypen

Ein primitiver Datentyp beschreibt die Größe und den Typ von Variablenwerten und hat keine zusätzlichen Methoden. Es gibt acht primitive Datentypen.

| Datentyp | Größe | Beschreibung |
|---|---|---|
| byte | 1 Byte | Ganzzahlen von -128 bis 127 |
| short | 2 Byte | Ganzzahlen von -32.768 bis 32.767 |
| int | 4 Byte | Ganzzahlen von -2.147.483.648 bis 2.147.483.647 |
| long | 8 Byte | Ganzzahlen von -9.223.372.036.854.775.808 bis 9.223.372.036.854.775.807 |
| float | 4 Byte | Kommazahlen bis 7 Dezimalstellen |
| double | 8 Byte | Kommazahlen bis 15 Dezimalstellen |
| boolean | 1 bit | Wahr- oder Falsch-Werte |
| char | 2 Byte | Einzeln er Character oder ASCII Werte |

Also was ist jetzt zu merken?
Primitive Zahlentypen können wir in zwei Gruppen aufteilen.

**Integer Typ:**

- Ganzzahlen
- positiv oder negativ
- keine Dezimalzahlen
- Mögliche Typen sind:
  - byte
  - short
  - **int**
  - long

**Floating Point Typ:**

- Zahlen mit einer oder mehreren Dezimalstellen
- Mögliche Typen sind:
  - float
  - **double**

Die wichtigsten wurden **markiert**.

## 5.2) Nicht primitive Datentypen

Nicht primitive Datentypen werden auch Referenztypen genannt, weil sie auf Objekte zugreifen (reference to objects). Die Hauptunterschiede zwischen primitiven und nicht primitiven Datentypen sind:

- Primitive typen sind vordefiniert. Nicht primitive Typen werden vom Programmierer erstellt (ausgenommen String) und sind nicht in Java definiert.
- Nicht primitive Typen können Methoden aufrufen, um bestimmte Operationen durchzuführen, primitive können dies nicht.
- Primitive Datentypen haben IMMER einen Wert, nicht primitive Datentypen können *null* sein. (null ist ungleich der Zahl 0. Man kann sich unter null auch „nix“ oder die Leere vorstellen)
- Primitive Typen beginnen mit Kleinbuchstaben, nicht primitive mit Großbuchstaben.

# 5.3) Typenkonvertierung

Typenkonvertierung (engl. Type Casting) wird verwendet wenn man den Wert eines primitiven Datentyps auf einen anderen Typ zuweist.

## 5.3.1) Automatisch

Wird verwendet um kleinere Datentypen auf größere Umzuwandeln. Dies passiert automatisch.
Reihenfolge:

`byte` -> `short` -> `char` -> `int` -> `long` -> `float` -> `double`

*Abbildung 22: Reihenfolge Widening Casting*

```
int myInt = 9;
double myDouble = myInt; // Automatic casting: int to double

System.out.println(myInt);      // Outputs 9
System.out.println(myDouble);   // Outputs 9.0
```

*Abbildung 23: Widening Casting. Automatische Typkonvertierung*

## 5.3.2) Manuell

Beim manuellen Konvertieren muss der Datentyp in welche man die Variable umwandeln möchte in runden Klammern angeben.

`double` -> `float` -> `long` -> `int` -> `char` -> `short` -> `byte`

*Abbildung 24: Reihenfolge Narrowing Casting*

```
double myDouble = 9.78d;
int myInt = (int) myDouble; // Manual casting: double to int

System.out.println(myDouble);   // Outputs 9.78
System.out.println(myInt);      // Outputs 9
```

*Abbildung 25: Narrowing Casting. Manuelle Typconvertierung*

# 5.4) Aufgabe 5:

Erstelle eine Ausarbeitung sowie eine anschließende Präsentationn zu den Themen:
Primitive Datentypen in Java
Nicht primitive Datentypen in Java
Typenkonvertierung in Java

Gehe dabei besonders auf folgende Punkte ein:

- Wodurch unterscheiden sich primitive und nicht-primitive Datentypen in Java?
- Wie wird diese Unterscheidung in z.B.: C# (C-Sharp) gehandhabt?
- Wie unterscheidet sich die Typenkonvertierung in Java mit der von C#?

Fasse dies in maximal 400 Wörtern zusammen.
Erstelle die Präsentation als Powerpoint.

# 6) Java Operatoren

## 6.1) Arithmetische Operatoren

| Operator | Name | Beschreibung | Beispiel |
|---|---|---|---|
| + | Addition | Addiert zwei Werte | x + y |
| - | Subtraktion | Subtrahiert einen Wert vom anderen | x – y |
| * | Multiplikation | Multipliziert zwei Werte | x * y |
| / | Division | Dividiert einen Wert durch den anderen | x / y |
| % | Modulo | Gibt den Rest der Division zurück | x % y |
| ++ | Inkrementieren | Erhöht den Wert der Variable um 1 | x++ |
| -- | Dekrementieren | Vermindert den Wert der Variable um 1 | y++ |

## 6.2) Zuweisungsoperatoren

| Operator | Beispiel | Gleich wie |
|---|---|---|
| = | x = 5 | x = 5 |
| += | x += 3 | x = x + 3 |
| -= | x –= 3 | x = x – 3 |
| *= | x *=3 | x = x * 3 |
| /= | x /=3 | x = x / 3 |
| %= | x %=3 | x = x % 3 |
| &= | x &=3 | x = x & 3 |
| \|= | x \|=3 | x = x \| 3 |
| ^= | x ^=3 | x = x ^ 3 |

## 6.3) Vergleichsoperatoren

| Operator | Name | Beispiel |
|---|---|---|
| == | ist gleich | x == y |
| != | ungleich | x != y |
| > | größer als | x > y |
| < | kleiner als | x < y |
| >= | größer oder gleich | x >= y |
| <= | kleiner oder gleich | x <= y |

## 6.4) Logische Operatoren

| Operator | Name | Beschreibung | Beispiel |
|---|---|---|---|
| && | Logisches UND | Ergibt true wenn beide Behauptungen true sind | x < 5 && x < 10 |
| \|\| | Logisches ODER | Ergibt true wenn mindestens eine Behauptung true ist | x < 5 \|\| x < 4 |
| ! | Logisches NICHT | Negiert das Ergebnis | !(x < 5 && x < 10) |

# 86.5) Aufgabe 6:

Erstelle eine kurze Ausarbeitung zu den folgenden Fragen. Die Antwort ist immer True/False/Fehler

A = True
B = False
C = True
X = 5
Y = 8
Z = -4

- Y<= Z
- X < Y
- X % !A
- X != Y && X > Z
- !(Y + Z > X)
- (A && !B) || (!A && C)
- (((X – Z) > Y && A) != B) == C
- A > B

# 7) Strings

Strings werden verwendet, um Text aller Art zu speichern. Worte, ganze Sätze oder nur ein paar Buchstaben, fast alles lässt sich als String darstellen.

Wichtig: Eine String Variable ist vergleichbar mit einer Liste von chars welche mit Apostrophen umgeben ist.

Beispiel:
String begruessung = “Hallo, welche Operation möchten Sie ausführen?“;

Wie bereits in Kapitel 5 angeschnitten, gehören Strings zu den nicht primitiven Datentypen. Das bedeutet sie können Methoden aufrufen. Ein paar wichtige String Methoden sind:

- .length()
- .toUpperCase()
- .toLowerCase()
- .indexOf()

## 7.1) String .length()

Ein String in Java ist eigentlich ein Objekt, welches auf Methoden zugreifen kann. Um zum Beispiel die Länge eines Strings, also z.B. die Anzahl an Buchstaben in einem Wort zu finden, kann die .length() Methode verwendet werden.
Diese liefert die **Anzahl aller Elemente im String** als **Integer!**

*Syntax:*
String variablenName = „ABCDEFGHIJKLMNOPQRSTUVWXYZ“;
System.our.println(“Die Länge des Strings ist: ” + variablenName.length());

## 7.2) String .toLowerCase()/.toUpperCase()

Wandelt einen String in Kleinbuchstaben oder Großbuchstaben um.

*Syntax:*
String variablenName = „ABCDEFGHIJKLMNOPQRSTUVWXYZ“;
System.our.println(variablenName.toLowerCase());
System.out.println(variablenName.toUpperCase());

## 7.3) String .indexOf()

Die indexOf Methode gibt uns den Index (die Position) eines gegebenen Textes zurück. Als Beispiel:

```
String txt = "Please locate where 'locate' occurs!";
System.out.println(txt.indexOf("locate")); // Outputs 7
```

*Abbildung 26: String.indexOf()*

Unser String txt entspricht eine Ansammlung von Wörtern. Unter anderem tritt das Wort *locate* 2 mal auf. Wenn wir nun genau nachsehen liefert uns die indexOf Methode den Wert 7. Warum? Weil der erste Buchstabe des ersten Auftretens von “locate“ das Zeichen mit dem Index 7 ist also das 8. Element (wir fangen bei 0 zu zählen an).

## 7.4) Zusammenfügen von Strings

Allgemein bekannt unter String Concatenation wird der + Operator dazu verwendet um Strings miteinander zu verbinden.

```
String firstName = "John";
String lastName = "Doe";
System.out.println(firstName + " " + lastName);
```

*Abbildung 27: String Concatenation 1*

Man kann auch die Methode **.concat()** verwenden. Hier ein Beispiel:

```
String firstName = "John ";
String lastName = "Doe";
System.out.println(firstName.concat(lastName));
```

*Abbildung 28: String Concatenation 2*

## 7.5) Besondere Zeichen

Da Strings in Anführungszeichen gesetzt werden müssen würde Java bei folgendem Beispiel einen Error generieren:

```
String txt = "We are the so-called "Vikings" from the north.";
```

*Abbildung 29: String Error*

Die Lösung zu diesem Problem sind sogenannte „Escape Character“. Diese werden verwendet um folgende Zeichen zu erzeugen:

| Escape Character | Ergebnis | Beschreibung |
|---|---|---|
| \‘ | ‘ | Einfaches Anführungszeichen |
| \“ | “ | Doppeltes Anführungszeichen |
| \\ | \ | Backslash |
| **\n** | New Line | Beginnt einen neuen Absatz (wie ENTER in Word) |
| **\t** | Tabulator | Fügt einen TabSprung ein |

# 7.6) Aufgabe 7:

Erstelle ein neues Projekt mit dem Projektnamen „StringUebung“.
Verwende nun die Methoden aus 7.1 – 7.5 auf Strings deiner Wahl.
Ändere den String im laufe deiner Tests und **dokumentiere deine Erkenntnisse**.

Gib deine Dokumentation als Strings.pdf ab.

# 8) Java und Mathematik

Um in Java auch andere Rechenoperationen durchzuführen als +, -, * und / wurde natürlich ein schon vorab eine einfachere Lösung dafür bereitgestellt. Diese ist die Java Math Klasse.

## 8.1) Math.max(x,y)

Kann verwendet werden um den höchsten Wert, der angegeben wird, ausfindig zu machen.

Math.max(5, 10); → Gibt 10 zurück.

## 8.2) Math.min(x, y)

Kann verwendet werden um den niedrigsten Wert, der angegeben wird, ausfindig zu machen.

Math.max(5, 10); → Gibt 5 zurück.

## 8.3) Math.sqrt(x)

Berechnet die Quadratwurzel der übergebenen Zahl x.

Math.sqrt(64); → Gibt 8 zurück.

## 8.4) Math.random();

Math.random gibt eine Zufallszahl zwischen 0.0 und 1.0 zurück.
Um mehr Kontrolle über die Zufallszahl zu bekommen, z.B.: Zahl soll zwischen 0 und 100 liege, kann man folgende Formel verwenden:

```
int randomNum = (int)(Math.random() * 101);  // 0 to 100
```

*Abbildung 30: Math.random()*

# 9) Booleans

Sehr oft wird in der Welt der Programmierer ein Datentyp mit dem Namen Boolean verwendet. Bei Anlagensteuerungen, SmartHomes, uvm., tritt der Datentyp boolean auf. Warum das Ganze? Weil boolean nur zwei verschiedene Werte annehmen kann, **true** oder **false**.

Viele Dinge können durch true und false ausgedrückt werden.

- JA / NEIN
- EIN / AUS
- RICHTIG / FALSCH

## 9.1) Deklaration

```
boolean isJavaFun = true;
boolean isFishTasty = false;
System.out.println(isJavaFun);      // Outputs true
System.out.println(isFishTasty);    // Outputs false
```

*Abbildung 31: Beispiel boolean*

Allerdings ist es häufiger der Fall, dass ein boolscher Wert durch einen Ausdruck entsteht.

## 9.2) Boolsche Ausdrücke

Ein boolscher Ausdruck wird in Java durch eine Operation, die einen boolschen Wert (true/false) liefert, erzeugt. Dazu gehören alle **Vergleichsoperatoren** und **logischen Operatoren** (siehe Kapitel 6).
Zum Beispiel:

- 5 < 8 → true
- 5 > 8 → false

```
int x = 10;
int y = 9;
System.out.println(x > y);
```

*Abbildung 32: Beispiel boolscher Ausdruck mit > Operator*

# 10) Entscheidungsanweisungen

Java unterstützt die üblichen logischen Bedingungen der Mathematik

- Kleiner als: a < b
- Kleiner gleich: a <= b
- Größer als: a > b
- Größer gleich: a >= b
- Gleich: a == b
- Ungleich: a != b

Diese Bedingungen können verwendet werden, um unterschiedliche Entscheidungen zu treffen und so Code gezielt auszuführen.

Java beinhaltet dafür folgende Statements

- Verwende **if** um einen Block Code auszuführen, wenn eine **bestimmte** Bedingung **true** ist
- Verwende **else** um einen Block Code auszuführen, wenn die **selbe** Bedingung **false** ist.
- Verwende **else if** um eine neue Bedingung einzuführen sollte die erste **false** sein.
- Verwende **switch** um viele verschiedene Codeblöcke gezielt auszuführen.

## 10.1) Das if – Statement

Verwende **if** um einen Block Code auszuführen, wenn eine **bestimmte** Bedingung **true** ist

Das if MUSS klein geschrieben werden, sonst tritt ein Error auf.

Syntax

```
if (condition) {
  // block of code to be executed if the condition is true
}
```

*Abbildung 33: if - Syntax*

In folgendem Beispiel wird veranschaulicht wie ein if funktioniert.
Im Falle, dass die Bedingung true ist, wird ein Text ausgegeben.

```
if (20 > 18) {
  System.out.println("20 is greater than 18");
}
```

*Abbildung 34: if - Example 1*

Das Ganze funktioniert natürlich auch mit Variablen:

```
int x = 20;
int y = 18;
if (x > y) {
  System.out.println("x is greater than y");
}
```

*Abbildung 35: if - Example 2 ( mit Variablen )*

Im obigen Beispiel verwenden wir 2 Variablen, x und y, um herauszufinden ob x > y stimmt, also ob x größer als y ist, oder nicht. Dies geschieht mit dem > Operator.
Nachdem x dem Wert 20 entspricht und y dem Wert 18, und wir wissen, dass 20 > 18 ist, wird der Code, welcher in den geschwungenen Klammern des ifs steht, ausgeführt.

## 10.2) Das else – Statement

Verwende **else** um einen Block Code auszuführen, wenn die **selbe** Bedingung **false** ist.

### Syntax

```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}
```

*Abbildung 36: else - Syntax*

```
int time = 20;
if (time < 18) {
  System.out.println("Good day.");
} else {
  System.out.println("Good evening.");
}
// Outputs "Good evening."
```

*Abbildung 37: else – Example*

Im obigen Beispiel entspricht time dem Wert 20. Nachdem 20 < 18 false ist wird der if block nicht ausgeführt, aber der else Block. Wenn der Wert time < 18 wäre, würde „Good Day“ ausgegeben werden.

# 10.3) Das else if – Statement

## Syntax

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is false and condition2 is true
} else {
  // block of code to be executed if the condition1 is false and condition2 is false
}
```

*Abbildung 38: else if – Syntax*

```
int time = 22;
if (time < 10) {
  System.out.println("Good morning.");
} else if (time < 20) {
  System.out.println("Good day.");
} else {
  System.out.println("Good evening.");
}
// Outputs "Good evening."
```

*Abbildung 39: else if – Example*

Im obigen Beispiel ist time auf 22 gesetzt. Dadurch ergibt sich:

- Die if Bedingung 20 < 10 ist nicht erfüllt
- Die else if Bedingung 20 < 22 ist auch nicht erfüllt also
- Else wird ausgeführt

# 10.4) Aufgabe 8:

Erstelle ein neues Projekt mit dem Namen „Simple Calculator“.
Erstelle eine zugehörige Klasse Calculator.

Füge deiner Klasse 3 Attribute für Zahl1, Zahl2 und die Rechenoperation hinzu.
Lies für jede deiner Variablen einen zugehörigen Wert vom Benutzer ein. Wir gehen davon aus, dass die Eingaben gültig sind.

Verwende if/else if/else um zwischen den Grundrechungsarten +, -, * und / zu unterscheiden. Anschließend soll das jeweilige Ergebnis auf der Konsole ausgegeben werden.

Achtung: Eine Division durch 0 soll nicht möglich sein!

Außerdem soll bei falscher Rechenoperation eine Warnung an den Benutzer ausgegeben werden („Ungültige Operation“).

Das Programm soll nur eine Rechnung durchführen können (keine Schleifen).

# 10.5) Das switch – Statement

Verwende **switch** um viele verschiedene Codeblöcke gezielt auszuführen.

## Syntax

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

*Abbildung 40: switch – Syntax*

Und so funktionierts:

- Der **Ausdruck** (expression) des switch wird **einmalig** evaluiert (überprüft)
- Der Wert des Ausdrucks wird mit jedem **case** verglichen
- Wenn es eine **Übereinstimmung** gibt wird der zugehörige Code Block ausgeführt
- **Break** und **default** sind optional. Diese Schlüsselwörter werden noch erklärt.

Das folgende Beispiel verwendet die Zahl des Wochentags um den Namen des zugehörigen Tages auszugeben ( 1 für Montag, usw…)

```
int day = 4;
switch (day) {
  case 1:
    System.out.println("Monday");
    break;
  case 2:
    System.out.println("Tuesday");
    break;
  case 3:
    System.out.println("Wednesday");
    break;
  case 4:
    System.out.println("Thursday");
    break;
  case 5:
    System.out.println("Friday");
    break;
  case 6:
    System.out.println("Saturday");
    break;
  case 7:
    System.out.println("Sunday");
    break;
}
// Outputs "Thursday" (day 4)
```

*Abbildung 41: Switch – Example*

## 10.6) Das break Schlüsselwort

Wenn Java ein **break** erreicht, verlässt des den **switch** Block. Dies verhindert das ausführen von weiterem Code innerhalb des **switch** Blocks. Wenn eine Übereinstimmung getroffen wurde und ein **case** ausgeführt wird ist es Zeit für eine Pause (engl. break). Es gibt dann nichts mehr zum ausführen.

## 10.7) Das default Schlüsselwort

Das **default** Schlüsselwort spezifiziert einen bestimmten Code der ausgeführt werden soll, wenn **KEINE ÜBEREINSTIMMUNG** gefunden werden kann.

```
int day = 4;
switch (day) {
  case 6:
    System.out.println("Today is Saturday");
    break;
  case 7:
    System.out.println("Today is Sunday");
    break;
  default:
    System.out.println("Looking forward to the Weekend");
}
// Outputs "Looking forward to the Weekend"
```

**Wichtig:**
Das default Statement wird als **letztes** Statement in einem switch Block deklariert und benötigt **kein break**.

# 10.8) Aufgabe 9:

Erweitere dein zuvor erstelltes Programm nun um folgende Dinge:

- Verwende Refactor-Rename um das Projekt umzubenennen (Rechte Maustaste auf den Projektordner) und nenne es „Advanced Calculator“

- Lies eine weitere Zahl3 ein.
- Die if/else Fallunterscheidung der Rechenoperation soll durch switch/case ersetzt werden. Weiters sollen folgende Berechnungsarten ermöglicht werden:

- Finde die kleinste eingegebene Zahl (min)
- Finde die größte eingegebene Zahl (max)
- Quadratwurzel von Zahl1 und Zahl2 (sqrt)
- Mittelwert aller eingelesenen Zahlen (mean)

- Lies dir das Kapitel 11.1 durch und stelle die Möglichkeit mehrer Berechnungen mithilfe von Schleifen dar.
- Überlege dir dafür eine geeignete Variante (Unendlich Rechnungen bis abgebrochen wird, Anzahl der Rechnungen wird vorher eingelesen, etc…) und entscheide dich für eine davon.

# 11) Schleifen

Schleifen können einen Block Code zyklisch ausführen, solange eine bestimmte Bedingung erfüllt ist. Schleifen sind sehr praktisch, weil sie Zeit sparen, Fehler vermindern und den Code um einiges leserlicher machen.

## 11.1) While Schleife

Die while Schleife führt einen Code aus, solange die Bedingung true ist.

**Syntax**

```
while (condition) {
  // code block to be executed
}
```

*Abbildung 42: While – Syntax*

Mit solch einer Konstruktion können wir z.B.: einen Teil des Programms x – mal ausführen. Sagen wir, wir möchten einen Block Code 5 Mal ausführen:

```
int i = 0;
while (i < 5) {
  System.out.println(i);
  i++;
}
```

*Abbildung 43: While 5 Iterationen*

Der obrige Code führt das System.out.println genau solange aus, solange i < 5 ist. Allerdings erhöht sich i nach jedem Durchlauf der Schleife mithilfe von i++. Die Ausgabe sollte also die Zahlen 0, 1, 2, 3 und 4 ergeben.

## 11.2) Do/While Schleife

Die do/while Schleife ist eine Variante der While Schleife. Diese führt den Code Block einmalig aus, bevor überprüft wird ob die Wiederholbedingung true ist. Anschließend wird der Code Block so lange wiederholt wie die Bedingung true ist.

**Syntax**

```
do {
  // code block to be executed
}
while (condition);
```

*Abbildung 44: Do/While – Syntax*

```java
int i = 0;
do {
  System.out.println(i);
  i++;
}
while (i < 5);
```

*Abbildung 45: Do While 5 Iterationen*

## 11.3) For Schleife

Wenn du genau weißt wie oft du durch einen Codeblock Iterieren möchtest, kannst du anstatt von While Schleifen eine For Schleife verwenden.

### Syntax

```java
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

**Statement 1:** Wird einmalig ausgeführt bevor der Codeblock ausgeführt wird.

**Statement 2:** Entspricht der Bedingung. Solange sie erfüllt ist wird wiederholt.

**Statement 3:** Wird nach JEDEM Durchlauf der Schleife einmal ausgeführt.

```java
for (int i = 0; i < 5; i++) {
  System.out.println(i);
}
```

*Abbildung 46: Beispiel For - Loop*

```java
for (int i = 0; i <= 10; i = i + 2) {
  System.out.println(i);
}
```

*Abbildung 47: Beispiel 2 For – Loop*

Tippe die beiden Beispiele aus Abbildung 46 und 47 mal in deine IDE ab und notiere hier deine Ergebnisse:

## 11.4) For-Each Schleife

Eine weitere Variante der for-Schleife ist die for-each Schleife. Sie wird häufig zum Durchkämmen von **Arrays** (siehe Kapitel 13) verwendet. Die direkte Übersetzung in die Deutsche Sprache wäre: „Für jedes Element in der Liste *Blablabla* führe den folgenden Code Block aus:“

```
for (type variableName : arrayName) {
  // code block to be executed
}
```

*Abbildung 48: For-Each Syntax*

# Aufgabe 10:

Erstelle ein neues Projekt mit dem Projektnamen LoopsUebung. Deine Applikation soll dir das kleine 1x1 in folgendem Schema ausgeben:

1er Reihe:
1 x 1 = 1
1 x 2 = 2
1 x n = n
…
1 x 10 = 10

_____
2er Reihe:
2 x 1 = 2
2 x 2 = 4
2 x n = n
…
2 x 10 = 20

_____
USW…
…
…
…
10 x 10 = 100

Verwende dazu nicht mehr als 20 Zeilen Code!!!

# 12) Break/Continue

## 12.1) Break

Das break Statement, wie bereits in **switch** angeschnitten, wird einerseits verwendet um aus einem Switch Statement zu „hüpfen“ und andererseits auch um aus einer Schleife zu „hüpfen“. Das folgende Beispiel zeigt, wie aus einer Schleie gesprungen werden kann wenn ein gewisser Wert erreicht wurde (z.B. 4).

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    break;
  }
  System.out.println(i);
}
```

*Abbildung 49: Java Break*

## 12.2) Continue

Das continue Statement bricht nur eine einzige Iteration (in der Schleife), wenn eine Bedingung auftritt und fährt mit der nächsten Iteration fort (darum „continue“).
Folgendes Beispiel überspringt den Durchlauf der Schleife wenn der Wert 4 auftritt:

```
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  System.out.println(i);
}
```

*Abbildung 50: Java Continue*

# Aufgabe 11:

Erstelle ein neues Java Projekt mit dem Projektnamen „BreakContinueUebung“.
Frage den Benutzer um Zahleneingaben aber speichere diese als String ab. Convertiere deinen String in ein dir sinnvoll erscheinendes Zahlenformat und addiere alle umgewandelten Zahlen zyklisch zusammen.
Der Vorgang wird beendet und die Ausgabe des Ergebnisses erfolgt sobald der User statt einer Zahl den String „ende“ oder „stop“ eingibt. Groß-Kleinschreibung sollen bei der Eingabe keinen Unterschied machen also „EnDE“ ist gleich gültig wie „eNDe“.

# 13) Arrays

Arrays werden verwendet, um mehrere Werte in einer einzigen Variable zu speichern, anstatt dafür separate Variablen anzulegen. Um ein Array zu deklarieren, lege den Datentyp der Variable mit **eckigen Klammern** an.

```
String[] cars;
```

*Abbildung 51: Array vom Typ String mit dem Namen "cars"*

Um nun dem Array Werte zuzuweisen, wird eine kommagetrennte Liste von Werten in geschwungenen Klammern verwendet.

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

*Abbildung 52: Array vom Typ String mit 4 Elementen*

```
int[] myNum = {10, 20, 30, 40};
```

*Abbildung 53: Array vom Typ Integer mit 4 Elementen*

## 13.1) Zugriff auf Elemente eines Arrays

Man kann auf die Elemente eines Arrays zugreifen, indem man auf den Index des gesuchten Elements zugreift. Der Index eines Arrays mit 4 Elementen beginnt mit 0 und endet mit 3, (0, 1, 2, 3 → 4 Elemente). Um also auf das erste Element einer Liste zuzugreifen verwendet man den Index 0:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars[0]);
// Outputs Volvo
```

*Abbildung 54: Elementzugriff*

Man spricht wörtlich „Drucke das 0. Element aus dem Array cars“ oder „Greife auf das Element in cars zu, welches den Index 0 hat“.

## 13.2) Ändern von Elementen eines Arrays

Um einen bestimmten Wert in einem Array zu ändern, greift man wieder mithilfe des Index auf diese Position zu, und überschreibt den vorherigen Wert.

```
cars[0] = "Opel";
```

*Abbildung 55: Überschreiben/Ändern von Elementen eines Arrays*

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
cars[0] = "Opel";
System.out.println(cars[0]);
// Now outputs Opel instead of Volvo
```

*Abbildung 56: Ändern Beispiel*

## 13.3) Länge eines Arrays

Um herauszufinden, wie lang ein Array ist (wieviele Elemente sich darin befinden), kann die length property verwendet werden:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars.length);
// Outputs 4
```

*Abbildung 57: Array.length*

## 13.4) Durchkämmen eines Arrays

Man kann ein Array mithilfe eine for – oder for – each Schleife durchkämmen. Hier ein Beispiel für beide Möglichkeiten:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (int i = 0; i < cars.length; i++) {
  System.out.println(cars[i]);
}
```

*Abbildung 58: For Array*

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars) {
  System.out.println(i);
}
```

*Abbildung 59: For Each Array*

# 13.5) Aufgabe 12:

Erweitere dein Advanced Calculator Projekt aus dem Kapitel Entscheidungsanweisungen um folgende Dinge:

Teil 1:

- Benenne dein Projekt um („Calculator with Arrays“).
- Nun sollen eine fixe Anzahl an Berechnungen (5-mal) ermöglicht werden. Verwende dazu deine Schleife und beende sie, sobald die Anzahl an möglichen Berechnungen erreicht wurde. Speichere alle deine Ergebnisse in ein Array ab und gib dieses vollständig nach jeder Berechnung aus.

Teil 2:

- Es sollen beliebig viele Berechnungen möglich sein. Dazu benötigst du eine Abbruchbedingung (break) für deine Schleife.
- Überprüfe vor jeder Berechnung, ob der Benutzer rechnen möchte oder nicht.
- Das Array soll nun 5 Plätze haben.
- Weiters soll ein neuer Case „show results“ geschaffen werden. Dieser wird als Rechenoperation behandelt. Wird dieser Case aufgerufen, werden die 5 **aktuellsten** Ergebnisse auf deiner Konsole ausgegeben. Um dies zu testen solltest du mehr als 5 Berechnungen durchführen.
  Um diesen Teil der Aufgabe zu erfüllen musst du dir überlegen, wie du algorithmischen die Plätze in deinem Array nachrückst.
  - Hilfestellung:
    int zahlen[] = {23, 44, 98, -5};
    // Neue eingelesene Zahl ist 7.
    // Das Array sieht nun so aus: {7, 23, 44, 98}.

# PHASE 2) Aufgabe 13

Frage deinen Trainer nach dem StyleGuide.

# 14) Methoden

Methoden sind ein wichtiger Bestandteil von Java-Programmen und ermöglichen es, wiederkehrende Aufgaben in einem einzelnen, wiederverwendbaren Block von Anweisungen zu organisieren. Sie erleichtern das Schreiben, Lesen und Warten von Code, indem sie es ermöglichen, komplexe Prozesse in kleinere, leicht verständliche Schritte zu unterteilen.

**Vorab:**

In Java gibt es zwei Hauptmodifikatoren für Methoden: **"public" und "private"**.

- "**public**" bedeutet, dass die Methode von **jeder Klasse** aus aufgerufen werden kann, unabhängig davon, ob sie in dem gleichen Paket oder einem anderen Paket ist. Eine Methode mit dem Modifikator "public" ist also öffentlich zugänglich.
- "**private**" bedeutet, dass die Methode nur innerhalb der Klasse aufgerufen werden kann, in der sie definiert ist. Eine Methode mit dem Modifikator "private" ist also nur **innerhalb der Klasse** sichtbar und nicht von außerhalb der Klasse zugänglich.

Diese Modifikatoren werden verwendet, um den Zugriff auf Methoden zu steuern und sicherzustellen, dass nur autorisierte Teile des Codes auf bestimmte Methoden zugreifen können.

Ein weiteres Wichtiges Keyword ist „**static**“. Dies wird verwendet um beispielsweise Methoden als „statische Methoden“ zu deklarieren.

- Statische Methoden können direkt aufgerufen werden, ohne dass eine Instanz der Klasse (ein Objekt) erstellt werden muss. Sie kann aufgerufen werden, indem der Klassenname verwendet wird, gefolgt von dem Namen der Methode. Ein Beispiel für eine statische Methode wäre Math.max()
- Mehr dazu aber später…

# 14.1) Definition und Aufruf von Methoden

Das Definieren einer Methode ist leichter als man denkt. Methoden werden nach folgendem Schema erstellt:

<Zugriffstyp> <Statisch oder nicht> <Methodentyp> <Methodenname> <Runde Klammern>

Hier ein Beispiel für richtig erstellte Methoden:

```
2  public class HelloWorld
3  {
       Moritz Wenger *
4      public static void main(String[] args)
5      {
6          int a = 4;
7          int b = 5;
8          System.out.println("Addition ergibt: " + add(a, b));
9
10     }
11
12     /**
13      * Die Methode add erwartet sich 2 Werte als Übergabewerte nämlich int x, und int y.
14      * Diese beiden Variablennamen x und y sind NUR für diese Methode add relevant.
15      * @param x entspricht einer Ganzzahl, die in diesem Beispiel mit dem Wert von int a also 4 befüllt wird
16      * @param y entspricht einer Ganzzahl, die in diesem Beispiel mit dem Wert von int b also 5 befüllt wird
17      * @return gibt den Wert x + y also 9 zurück. Bildlich gesprochen wandert dieser Wert an die Stelle wo die Methode
18      *          aufgerufen wurde also Zeile 8.
19      */
        new *
20     public static int add(int x, int y){
21         return x + y;
22     }
23 }
```

*Abbildung 60: Methodenerstellung Basics*

## 14.2) Methoden - Troubleshooting

Es ist wichtig, dass du dich beim Erstellen und Aufrufen von Methoden ausprobierst. Die meisten Fehler, die beim Coden mit Methoden zu Beginn auftreten sind folgende:

- Die Methode, die man aufrufen will, wurde nicht erstellt
- Die Methode, die man aufrufen will, ist nicht static obwohl sie static sein müsste
  - Dazu muss bei der Deklaration ein static eingefügt werden siehe Zeile 20.
- Die Methode erwartet sich mehr Parameter als sie übergeben bekommt
  - Hier gilt es: Wenn die Methode n Werte erwartet, müssen bei JEDEM Aufruf auch n Werte übergeben werden.
- Die Methode erwartet sich weniger Parameter als sie übergeben bekommt
  - Hier gilt es: Wenn die Methode n Werte erwartet, müssen bei JEDEM Aufruf auch n Werte übergeben werden.
- Die Datentypen der Übergebenen Parameter stimmen nicht überein
  - Hier auch die Reihenfolge beachten!
- Die Methode wurde zwar erstellt wird aber nie aufgerufen
  - Führt zwar zu keinem Error, aber es geht trotzdem oft nicht gut aus
- Die Methode gibt einen Wert zurück, obwohl sie vom Typ void ist
  - Entferne das return aus der Methode.
- Die Methode gibt keinen Wert zurück, obwohl sie nicht vom Typ void ist
  - Füge ein return am Ende der Methode ein
- Die Methode wurde außerhalb der Klasse erstellt
  - Achte auf deine {}

## Aufgabe 14.1

Frage deinen Trainer um die Aufgabe PV-System.
Der StyleGuide gilt für diese Aufgabe noch nicht.
Arbeite die Aufgabe aus in dem du NUR die Main Methode verwendest.
Teste deine Ausgaben.

## Aufgabe 14.2

Verwende dein Wissen über Methoden und lagere Programmteile in Methoden aus, um den StyleGuide einzuhalten. Kommentiere deinen Code sinngemäß (siehe StyleGuide).

# 15) Objektorientierte Programmierung

Die Grundprinzipien der OOP sind für manche etwas schwerer nachzuvollziehen. Da es die perfekte Erklärung nicht gibt, schon gar nicht in Worten, verlinke ich hier einige Quellen, die zur Recherche dieser Themen hergenommen werden können.

ACHTUNG:
Bitte diese Ausarbeitung ernst nehmen es ist mitunter eines der wichtigsten Kapitel im Lehrplan und das Konzept der OOP sollte nach diesem Kurs verstanden und angewandt werden können.

**Englisch:**
https://www.w3schools.com/java/java_oop.asp
https://www.w3schools.com/java/java_classes.asp
https://www.w3schools.com/java/java_class_attributes.asp
https://www.w3schools.com/java/java_class_methods.asp

**Deutsch:**
https://programmieren-starten.de/blog/objektorientierte-programmierung/

https://studyflix.de/informatik/objektorientiertes-programmieren-i-423
https://studyflix.de/informatik/objektorientiertes-programmieren-ii-425

## 15.1) OOP Aufgabe 1

Erstelle eine Klasse Car
Die Klassenattribute von Car sind:

- make – die Marke des Fahrzeugs
- model – das Model des Fahrzeugs
- year – das Baujahr des Fahrzeugs

Die Klassenmethoden von Car sind:

- start() – Gibt „Car has started“ auf der Konsole aus
- accelerate(int speed) – Gibt „Car is accelerating to <*SPEED*> km/h“ aus
- stop() – gibt „Car has stopped“ aus

Erstelle eine Klasse Main

- Erstelle in der main Methode ein neues Objekt des Typen Car und nenne es myCar
- Setze die Objektattribute von myCar auf beliebige Werte.
- Gib die Objektattribute anschließend auf der Konsole aus
- Rufe die 3 Methoden start, accelerate und stop auf

Versuche anschließend ein zweites Car Objekt zu erstellen, belege die Objektattribute mit anderen Werten als bei myCar und gib die Outputs wie vorhin aus.

# 15.2) Der Konstruktor

Der Konstruktor in Java ist im Grunde nur eine besondere Art von Methode, die verwendet wird um beim Erstellen eines Objekts, dessen Objektattribute auf bestimmte Werte zu setzen. Der Konstruktor wird automatisch beim Erstellen des Objekts aufgerufen.

**WICHTIG:**

**Der Konstruktor hat den GLEICHEN Namen wie die Klasse und keinen Typen.**

Java stellt **2 Typen von Konstruktoren** zur Verfügung:

- Der Default Konstruktor
  - Keine Übergabewerte
  - **Wird automatisch vom Compiler erstellt**, wenn kein anderer Konstruktor definiert wird.
  - Initialisiert die Attribute eines Objekts mit Default-Werten

```
public class Car {
    // member variables
    String make;
    String model;
    int year;

    // default constructor
    Car() {
        make = "unknown";
        model = "unknown";
        year = 0;
    }
}
```

*Abbildung 61: Default-Constructor*

- Parametrisierter Konstruktor
  - 1 oder mehrere Übergabeewerte
  - Wird verwendet um die Attribute eines Objekts mit spezifischen Werten zu initialisieren.

```
public class Car {
    // member variables
    String make;
    String model;
    int year;

    // parameterized constructor
    Car(String make, String model, int year) {
        this.make = make;
        this.model = model;
        this.year = year;
    }
}
```

*Abbildung 62: Parameterized Constructor*

# 15.3) this – Keyword

Das Keyword „this“ ist eine **Referenz auf das aktuelle Objekt,** oder auf das Objekt, welches einer Methode oder einem Konstruktor zugehörig ist.
Es **unterscheidet** zwischen Klassenattributen und lokalen Variablen mit dem **gleichen Namen** und kann auch verwendet werden um andere Konstruktoren in der gleichen Klasse aufzurufen.

In Abbildung 62 wird *THIS* verwendet, um als Referenz auf das zu erstellende Objekt zu dienen.

**Erklärung:**
Möchte ich mehrere Objekte der Klasse Car erstellen (also mehrere Autos) diese sollen aber unterschiedliche Hersteller und Modeltypen haben, dann referenziert *THIS* immer nur auf das **aktuell zu erstellende und initialisierende Objekt.**

Damit lassen sich auch hunderte oder tausende Auto Objekte erstellen, die sich alle im Model, Hersteller und Baujahr unterscheiden, ohne, dass jedes Auto manuell diese Werte eingetragen bekommen muss (da es beim Konstruktor für jedes Auto gleich korrekt initialisiert wird, abhängig von den Werten die ich als Übergabeparameter übergebe)

# 15.4) Konstruktor Aufgabe 1

Erweitere deine Klasse Car um einen solchen Parametrisierten Konstruktor und rufe diesen beim Erstellen der Objekte korrekt auf. Sollten dabei Fehlermeldungen aufscheinen Empfehle ich das Kapitel Methoden – Troubleshooting nochmal anzusehen.

Erstelle nun **30 Autos** mit einer **Schleife** und rufe nach dem Erstellen die zuvor erstellten Methoden in derselben Reihenfolge auf.
Damit die Attribute der einzelnen Autos auch unterschiedlich sind, verwende jeweils ein Array mit **10 Herstellern, 5 Modellen und 20 Baujahren** und ruf den Konstruktor beim Erstellen eines Autos mit **zufälligen Werten** aus deinen Arrays auf. Verwende dafür **random.nextInt().**

Sobald du einen Output erfolgreich schaffst kopiere dir den Konsolenoutput per Hand und speichere ihn in eine Word File. Vergleiche so deine Outputs wenn du das Programm 2 oder 3 mal startest und schreibe deine Erkenntnisse nieder.