# Zusammenfassung Prozedurale Programmierung Februar 2019

## Tag 1

### Kommentare

Werden vom Compiler ignoriert.

#### Einzeilig

Eingeleitet mit zwei Schrägstrichen

```
// Das ist ein Kommentar
```

#### Mehrzeilige

Eingeleitet durch Schrägstrich und Sternchen, beendet durch Sternchen und Schrägstich

```
/*
Das ist ein ...
... mehrzeiliger ...
... Kommentar
 */
```

### Ausgaben auf der Konsole

#### Mit Zeilenumbruch

```
System.out.println("Hello World");
```

#### Ohne Zeilenumbruch

```
System.out.print("Hello World ohne Zeilenumbruch");
```

#### Fehlermeldungen

Fehlermeldungen werden auf der Konsole rot dargestellt.

```
System.err.print("Fehlermeldung ohne Zeilenumbruch");
System.err.println("Fehlermeldung mit Zeilenumbruch");
```

## Tag 2

### Sonderzeichen auf der Konsole

#### Backslash

Um einen \ auszugeben, müssen wir zwei Backslashes ("\\") eintragen.

```
System.out.print("Ein Backslash wird ausgegeben: \\");

Ausgabe: Ein Backslash wird ausgegeben \
```

### Neue Zeile

Um eine neue Zeile auszugeben, können wir zusätzlich zu println() auch \n verwenden.

```
System.out.print("Neue Zeile \n");
System.out.print("Test");

Ausgabe:        Neue Zeile
                Test
```

\n kann jederzeit in einem print() oder println() Statement genutzt werden.

```
System.out.print("Das soll in eine Zeile... \\nDas in die andere.");

Das soll in eine Zeile...
Das in die andere.
```

### Unicode Zeichen

https://unicode-table.com/de/

Unicode Zeichen können in einem eigenen String mit \u gefolgt von der Unicode Codierung eingegeben werden.

```
System.out.print("Preis: 239" + "\u20ac");
Preis: 239€
```

## Verzweigungen

### if(Bedingung) {Anweisung}

Die Anweisung wird ausgeführt, wenn die Bedingung erfüllt ist.
Es können beliebig viele if Blöcke vor einem Else Block stehen.

### else if(Bedingung) {Anweisung}

Die Bedingung wird geprüft, wenn die vorangehende Bedingung nicht erfüllt war.
Die Anweisung wir ausgeführt, wenn die aktuelle Bedingung erfüllt ist.
Es können beliebig viele else if Blöcke zwischen einem if und einem else Block stehen.

### else {Anweisung}

Die Anweisung wird ausgeführt, wenn die Bedingung nicht erfüllt ist.
Else kann nie alleine stehen.

# Tag 3

## Operatoren

### \> Größer

```
if(alter > 17)
```

Alles, was größer 17 ist erfüllt die Bedingung.

### \< Kleiner

```
if(alter < 16)
```

Alles, was kleiner 16 ist erfüllt die Bedingung.

### \>= Größer-gleich

```
if(alter >= 18)
```

18 und alles, was größer 18 ist erfüllt die Bedingung.

### \<= Kleiner-gleich

```
if(alter <= 10)
```

10 und alles was, kleiner 10 ist erfüll die Bedingung.

### == Gleich

```
if(alter == 21)
```

NUR 21 erfüllt die Bedingung.

### != Ungleich

```
if(alter != 21)
```

ALLES AUßER 21 erfüllt die Bedingung.

## Datentypen

### String

```
String meinWort = "Hallo";
```

Zeichenkette (32bit)

### int

```
int zahl = 3;
```

Ganzzahl (32bit)

### double

```
double kommaZahl = 3.8;
```

Kommazahl (64bit) !Achtung! Komma wird als Punkt geschrieben

### boolean

```
boolean antwort = false;
```

Kann zwei Werte annehmen, true oder false (wahr oder falsch).

if(antwort) entspricht if(antwort == true) if(!antwort) entspricht if(antwort == false)

## Scanner

Mit dem Scanner können wir Eingaben entgegennehmen.
Vor public class Main ergänzen wir

```
import java.util.Scanner;
```

In der public static void Main ergänzen wir

```
Scanner derScanner = new Scanner(System.in);
```

Um Strings auszulesen, nutzen wir:

```
String meinString = derScanner.nextLine();
```

Für Ganzzahlen:

```
int meineZahl = derScanner.nextInt();
```

Für Kommazahlen:

```
double meinKommazahl = derScanner.nextDouble();
```

## Schleifen

### For-Schleife

Häufig werden wir Teile unseres Programm öfter ausführen müssen.
Um nicht alles kopieren zu müssen, können wir uns Schleifen zu nutze machen.
Der allgemeine Aufbau einer For-Schleife sieht folgendermaßen aus:

```
for (Zählvariable mit Anfangswert; Bedingung; Schrittweite) {
    // Anweisung, die wiederholt werden soll
}
```

Diese können wir beliebig oft ineinander schachteln, wie wir es beim Zeichnen eines Rechteckes nutzen.

# Tag 4

Bearbeitung des Projekts Rechteck
Verschachtelte Zähler um Recht

# Tag 5

## Ausgelagerten Methoden

Wir können Teile unseres Programms zur besseren Struktur auslagern und bei Bedarf wieder aufrufen.

### Auslagern

```
static void methodenName() { Anweisungen, die ausgeführt werden sollen. }
```

### Aufruf in der main Methode

```
public static void main(String[] args) {
    methodenName();
}
```

# Tag 6

## printf

Mit System.out.printf() können wir Formatierungen auf unsere Ausgaben anwenden.
Ein toller Artikel der Beuth Hochschule:
http://public.beuth-hochschule.de/~grude/PrintfAppletHilfe.html

# Tag 7

## while-Schleife

Führt den Anweisungsblock solange aus, wie die Bedingung im Schleifenkopf erfüllt ist.

```
while(Bedingung) {
    Anweisungen
}
```

Mit break; können wir jederzeit aus der Schleife springen. Praktisch um beispielsweise ganze Programme nach dem Durchlauf zu wiederholen und kontrolliert zu beenden.

## switch case

Bietet sich an, wenn wir wir mehrere explizite Werte prüfen möchten.
Ein Switch Case kann nur einzelne Werte prüfen, keine Wertebereiche!

```
switch(zahl) {
    case 4:
        // ... Anweisungen
        break;
    case 6:
    case 7:
        // ... Anweisungen
        break;
    default:
        // ... Wird ausgeführt, wenn kein Case weiter oben bereits
               ausgeführt wurde.
        // Entsprich also dem else{} Block bei einer if/else Verzweigung.
    break;
}
```

# Tag 8

## Verzögerungen mit Thread.sleep

Mit Thread.sleep(1000) können wir unser Programm um 1 sec in der Ausführung unterbrechen.
In den Runden Klammern steht die Anzahl der Millisekunden.

```
try
{
    Thread.sleep(5000);   //   Programm wird für 5 Sekunden angehalten
}
catch (InterruptedException e)
{
    e.printStackTrace();  //   Sollten Fehler auftreten,
                              werden diese ausgegeben
}
```

Wichtig! Thread.sleep(1000); muss in einen try-catch Block gebaut werden, da es Fehler auswerfen kann, welche wir auffangen müssen!

## Modulo Operator

Mit % (Modulo) können wir den Rest einer Division berechnen.
10 % 7 = 3
10 / 7 = 1 Rest 3
Der Rest ist hierbei relevant, dieser wird angegeben.
Der Modulo Operator (%) kann wie jeder anderer arithmetische Operator (+,-,*,/) verwendet werden.

## Arrays

Ein toller Artikel über Arrays finden wir hier:
https://javabeginners.de/Arrays_und_Verwandtes/Array_deklarieren.php
Oder im Buch Java für Dummies ab Seite 340, Kapitel 11 Arrays verwenden

# Tag 9

## Verknüpfte Bedingungen

Seither haben wir in einer if-Abfrage lediglich eine Bedingung geprüft

```
if(zahl == 5) { ... }
```

Wir können jedoch mehr als nur eine Bedingung setzen.

### UND Verknüpfung

Mit doppeltem kaufmännischem Und (&&) können wir eine UND Verknüpfung erstellen.
Diese ergibt true (wahr), wenn beide Bedingungen erfüllt sind.

```
int zahl = 5
if(zahl > 2 && zahl < 10) { ... }
// Der Anweisungsblock würde ausgeührt werden,
// da die Bedingungen erfüllt sind.
// 5 ist größer zwei UND kleiner 10.
if(zahl > 8 && zahl < 10) { ... }
// Der Anweisungsblock würde  NICHT ausgeührt werden,
// da die Bedingungen erfüllt sind.
// 5 ist NICHT größer 8, aber kleiner 10.
```

### ODER Verknüpfung

Mit zwei senkrechten Strichen können wir ODER Verknüpfungen einleiten.
Diese ergibt true, wenn mindestens eine Bedingung erfüllt ist.

```
int  zahl = 5
if(zahl > 8 || zahl < 10) { ... }
// Der Anweisungsblock würde ausgeührt werden,
// da eine der Bedingungen erfüllt ist.
// 5 ist NICHT größer 8, ABER kleiner 10.
```

# Tag 10

## Methoden mit Parameter

Java für Dummies - Seite 214ff

## Methoden mit Rückgabetyp

Java für Dummies - Seite 216ff

## Methoden mit Parameter und Rückgabetyp

Methoden können gleichzeitig Parameter annehmen und einen Rückgabewert besitzen. Das eine schließt das andere nicht aus!

# Tag 11

## Zustandsautomaten

Ein Zustandsautomat ist eine Verbindung von while und switch mitderen Hilfe wir Zustände modellieren können.
Zwischen Zuständen kann beliebig oft gesprungen werden, sie können beliebig oft wiederholt werden.
Eine außerhalb der while Schleife definierte Zustandsvariable entscheidet über den aktuellen Zustand, also den Fall, der ausgeführt werden soll.
Um zwischen den Zuständen zu wechseln, müssen wir nur die Variable zustand verändern.
Sobald der aktuelle Fall (case) beendet wird, wird im nächsten Durchlauf der dem Zustand entsprechende Fall ausführt.

```
int zustand = 0;
while (zustand < 3) {
    switch (zustand) {
        case 0:
            // Starte Programm
            // Ausgabe: Willkommen bei TicTacToe V1.1
            zustand = 1;
            // Zustand wird auf 1 gesetzt
            // =>      Im nächsten Durchlauf wird der nächste Zustand
                       ausgeführt!
            break;
        case 1:
            // Führe Programm aus
            // Tic Tac Toe Spiel
            zustand = 2;
            // Zustand wird auf 2 gesetzt
            break;
```

```
        case 2:
             // Beende Programm
             // Nachfragen, ob beendet werde soll
             // Wenn ja, zustand = 3
             // (3 liegt außerhalb unserer Grenze der While Schleife,
             // daher wird der Automat beendet)
             // Wenn nein, dann zustand = 1
             // (Programm wird nochmal gestartet)
             break;
    }
}
```

## Zustandsautomat mit konstanten Bezeichnern

Um unserem Zustandsautomat mehr Aussagekraft zu verleihen, können wir zu Beginn unseres Programms konstante Bezeichner definieren und später über diese auf die entsprechenden Zustände zugreifen.
Die Definition derer erfolgt vor der Main Methode, also innerhalb der Klasse, aber außerhalb irgendwelcher Methoden.
Wir nutzen dazu static final int, da wir eine Variale erstellen, die über alle Methoden verfügbar (static) und außerdem nicht veränderbar (final) und ganzzahlig (int) sein soll.

```
// Definition innerhalb von public class, aber außerhalb von Methoden
static final int SP1WIN = 4;
static final int SP2WIN = 5;


// Innerhalb switch(zustand)
// statt case 4:
case SP1WIN:
    ...
break;


// Innerhalb von Methoden (also überall sonst in unserem Programm)
// statt zustand = 5;
zustand = SP2WIN;
```