**Unterlagen Praktische Informatik**

**Klasse 11**

**Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Inhaltsverzeichnis**

[1 Aufbau eine Applikation 3](#_Toc102460321)

[1.1 Methode println() 3](#_Toc102460322)

[Übung: AusgabeZahlen 3](#_Toc102460323)

[Übung: AusgabeAdresse 4](#_Toc102460324)

[Übung: AusgabeWichtelmann 4](#_Toc102460325)

[1.2 Variablen 4](#_Toc102460326)

[Variablen erstellen 4](#_Toc102460327)

[Inhalt einer Variablen lesen 5](#_Toc102460328)

[Übung: Wertezuweisungen 6](#_Toc102460329)

[1.3 Datentypen 7](#_Toc102460330)

[1.4 Typecasting/Typecast (Typenanpassung) 7](#_Toc102460331)

[1.5 Mathematische Standardoperationen 8](#_Toc102460332)

[1.6 Kommentare 9](#_Toc102460333)

[Übung: Notenberechnung 11](#_Toc102460334)

[1.7 Hinweise zur Formatierung 12](#_Toc102460335)

[Übung Benzinverbrauch 12](#_Toc102460336)

[Übung Zinsen 13](#_Toc102460337)

[2 Fallunterscheidungen 14](#_Toc102460338)

[2.1 If -Anweisung 14](#_Toc102460339)

[Übung: Kleinste von 3 Zahlen 15](#_Toc102460340)

[Übung: Berechnung Körpergröße 15](#_Toc102460341)

[2.2 Der Restwert-Operator (Modulo) 16](#_Toc102460342)

[Übung Schaltjahr 16](#_Toc102460343)

[Übung BodyMassIndex 18](#_Toc102460344)

[Übung Notenberechnung 19](#_Toc102460345)

[2.3 Logische Operatoren 21](#_Toc102460346)

[Übung: „and“ und „or“ 22](#_Toc102460347)

[2.4 Switch-Blöcke 23](#_Toc102460348)

[Übung: Grundrechenarten mit zwei Zahlen 24](#_Toc102460349)

[Übung: Bußgeldberechnung 25](#_Toc102460350)

[3 Schleifen 26](#_Toc102460351)

[3.1 While Schleife (kopfgesteuert) 26](#_Toc102460352)

[3.2 Do-While Schleifen 26](#_Toc102460353)

[Übung: Mittelwertberechnung 27](#_Toc102460354)

[Übung: Zinseszinsberechnung 28](#_Toc102460355)

[Übung: Erweiterung Aufgabe Zinseszinsberechnung 28](#_Toc102460356)

[Übung: Kostenvergleich 29](#_Toc102460357)

[Übung: Kennwort-Eingabe 30](#_Toc102460358)

[Übung: Zufallszahlen 31](#_Toc102460359)

[3.3 Die Zählschleife (For Schleife) 32](#_Toc102460360)

[3.4 Break / Continue 33](#_Toc102460361)

[Übung: Zählschleifen 34](#_Toc102460362)

[Übung: Abzahlungsdarlehen 35](#_Toc102460363)

[Übung: Exponential-Funktion 36](#_Toc102460364)

[Übung: Fakultät 36](#_Toc102460365)

[4 Array 37](#_Toc102460366)

[4.1 Eindimensionales Array 37](#_Toc102460367)

[Übung Temperaturen 39](#_Toc102460368)

[Übung Schulnoten mit eindimensionalen Array 40](#_Toc102460369)

[Übung Planeten-Quiz 41](#_Toc102460370)

[Übung Theoretische Aufgabe Min/Max 42](#_Toc102460371)

[Übung Auktion 43](#_Toc102460372)

[Übung Zufallszahlen 44](#_Toc102460373)

[Übung Theoretische Aufgaben: eindimensionales Array 45](#_Toc102460374)

[Übung: Interner Sortieralgorithmen 47](#_Toc102460375)

[Übung Lotto 48](#_Toc102460376)

[Übung: Wortspielquiz 49](#_Toc102460377)

[4.2 Mathematische Funktionen 51](#_Toc102460378)

[4.3 Zweidimensionales Array 52](#_Toc102460379)

[Übung Filialen 53](#_Toc102460380)

[Übung Schulnoten 54](#_Toc102460381)

[Übung TheorieAufgaben zum zweidimensionalen Array 55](#_Toc102460382)

[Übung TicToc 61](#_Toc102460383)

[5 Parameterübergabe an eine Methode 62](#_Toc102460384)

[Übung Parameterübergabe mit Rückgabewert 64](#_Toc102460385)

[Übung „Zahlen runden“ 65](#_Toc102460386)

# Aufbau eine Applikation

Quellcode:

*public class HalloWelt {*

*public static void main(String [] arg) {*

*System.out.println("Hallo Welt");*

*}*

*}*

In der 1. Zeile handelt es sich um eine **Klasse**. Diese bündelt logisch zusammengehörende Code. Dazu aber später erst mehr. Stell sie dir solange wie eine Datei vor, die gewisse Funktionen beinhaltet.

In der 2.Zeile interessiert erst mal nur das Wort „**main**“. Es handelt sich hierbei um die Hauptmethode unseres Programms. Sie gibt an, wo der Programmeinstieg ist und bestimmt den Programmablauf. Alles was nun innerhalb dieser main-Methode steht, wird von oben nach unten abgearbeitet. Eingegrenzt werden alle Anweisungen durch 2 geschweifte Klammern, dies wird als **Block** bezeichnet. Eine **Anweisung** endet immer mit einem Semikolon und ist nichts anders als ein Befehl an den Computer. In unserem Beispiel gibt es nur eine Anweisung: System.out.println („Hallo Welt“); Hierbei handelt es sich um eine **Methode**, die es uns erlaubt etwas auf der Konsole auszugeben. In diesem Fall: Hallo Welt.

Nachdem Abspeichern von HalloWelt.java, kann das Programm kompiliert werden. Ein Bytecode wird hierbei erzeugt mit der Dateierweiterung „class“. Wichtig diese Datei muss im selben Ordner wie die Datei HalloWelt.java gespeichert werden.

**Der Klassenname muss gleich dem Dateinamen sein. Außerdem beginnt er stets mit einem Großbuchstaben.**

## Methode println()

### Übung: AusgabeZahlen

Schreibe ein Java-Programm „AusgabeZahlen“, dass die folgenden Befehle zur Ausgabe bringt: Überlege dir an Hand des Resultats die Unterschiede zwischen den einzelnen Varianten dieser Anweisungen.

System.out.println("00"); *00*

System.out.println(00); *0*

System.out.println(1+2); *3*

System.out.println(+1-2); *-1*

System.out.println("1"+2); *12*

System.out.println("1"+2+3); *123*

System.out.println(1 + " und " + 2); *1 und 2*

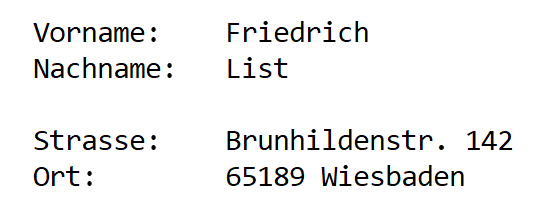
### Übung: AusgabeAdresse

Schreibe ein Programm „AusgabeAdresse“, das deinen Namen und deine Adresse unter Verwendung von Tabulatoren und Zeilenumbrüchen auf dem Bildschirm ausgibt.

Hinweis: \t = Tab \n = neue Zeile \“ = Anführungszeichen

\\= Backslash \‘ = einfaches Anführungszeichen

**Beispiel:**



### Übung: AusgabeWichtelmann

Schreibe das Programm „AusgabeWichtelmann“ zur Ausgabe der folgenden Figur.

*Das Programm soll nur aus einem* ***einzigen*** *System.out.println()-Befehl bestehen.*



## Variablen

* Variablen sind Namen für Speicherorte
* Variablen haben einen Datentyp (z.B. int, string)
* Variablennamen müssen mit einem kleinen Buchstaben beginnen.
* Können Buchstaben, Zahlen und „\_“ - Zeichen beinhalten. Bsp.: alter, autoMarke, auto\_Farbe, spieler12
* Kurze prägnante Namen sind sinnvoll

### Variablen erstellen

Bevor man eine Variable verwenden kann, muss man diese im Code erst bekannt machen. Dies nennt man „Deklaration“.

public class Programm {

public static void main (String [] args){

int alter; *// Grundstruktur der Deklaration: <Datentyp><Bezeichner><;>*

*}*

*}*

**Jetzt schreiben wir einen Wert in die Variable, dies nennt man Initialisierung**

public class Programm {

public static void main (String [] args){

int alter; *// Grundstruktur der Deklaration: <Datentyp><Bezeichner><;>*

alter = 17; // Initialisierung

*}*

*}*

### Inhalt einer Variablen lesen

public class Programm {

public static void main (String [] args){

int alter = 17; *// Deklaration und Initialisierung in einer Zeile*

system.out.println(alter); *//Wert wird auf der Konsole ausgegeben*

*}*

*}*

### Übung: Wertezuweisungen

Analysiere das folgende Java-Programm und beschreibe deren Ablauf Zeile für Zeile. Gebe an, welche Ergebnisse jeweils ausgegeben werden!

public class WerteZuweisungen {

public static void main (String[] args) {

int a;

int b;

int c;

int d;

a = 0;

b = 0;

c = 0;

d = 0;

a = 10;

System.out.println("1. Ausgabe A: " + a + "\tB: " + b + "\tC: " + c + "\tD: " + d );

a = a + 4;

System.out.println("2. Ausgabe A: " + a + "\tB: " + b + "\tC: " + c + "\tD: " + d );

b = 30;

System.out.println("3. Ausgabe A: " + a + "\tB: " + b + "\tC: " + c + "\tD: " + d );

a = b + a + 2;

System.out.println("4. Ausgabe A: " + a + "\tB: " + b + "\tC: " + c + "\tD: " + d );

d = a + b;

System.out.println("5. Ausgabe A: " + a + "\tB: " + b + "\tC: " + c + "\tD: " + d );

c = d + 3 \* b;

System.out.println("6. Ausgabe A: " + a + "\tB: " + b + "\tC: " + c + "\tD: " + d );

c = c + 1;

System.out.println("7. Ausgabe A: " + a + "\tB: " + b + "\tC: " + c + "\tD: " + d );

b = -a;

System.out.println("8. Ausgabe A: " + a + "\tB: " + b + "\tC: " + c + "\tD: " + d );

d = b - a + d;

System.out.println("9. Ausgabe A: " + a + "\tB: " + b + "\tC: " + c + "\tD: " + d );

}

}

## Datentypen

* Definiert die Art der möglichen Inhalte einer Variablen. Bspw. Text, Ganzzahlen, Flieskommazahlen
* Definiert was mit einer Variablen gemacht werden kann. Hier ein paar gängige Datentypen:

|  |  |  |
| --- | --- | --- |
| **Variablentyp** | **Bedeutung** | **Beispiel:** |
| boolean | Logischer Wert, der entweder den Zustand true oder false enthält | boolean b = true;  b = false; |
| Char | Unicode Zeichen | char c = 's'; |
| Byte | Alle ganzzahligen Werte von –128 bis 127 | byte b = 10; |
| Short | Alle ganzzahligen Werte von –32768 bis 32767 | short s = -20; |
| Int | Alle ganzzahligen Werte von –2147483648 bis 2147483647 | int i = 1; |
| Long | Alle ganzzahligen Werte im 64bit – Bereich | long l = -11000029; |
| Float | Alle Fließkommazahlen im 32bit – Bereich | float f = 453627.768; |
| double | Alle Fließkommazahlen im 64bit - Bereich | double d = 34875637.3643645; |
| String | Text |  |

## **Typecasting/Typecast (Typenanpassung**)

Falls man eine int-Zahl durch eine andere int-Zahl dividiert und man einen double-Wert als Ergebnis haben möchte, muss man in Java ein TypCasting durchführen.

Wichtig: Auf die Klammern achten, denn ergebnis = (double) ((note1\*1 +note2\*2 +…)/anzahl) führt zu einem Fehler.

**Automatische Typanpassung:**

Daten eines kleineren Datentyps werden automatisch (implizit) dem größeren Datentyp angepasst. Der Compiler nimmt diese Anpassung selbständig vor.

Explizite Typanpassung: Ein größerer Typ kann einem kleineren Typ nur mit Verlust von Informationen zugewiesen werden.

|  |  |
| --- | --- |
| Vom Typ | In Typ |
| Byte | short, char, int, long, float, double |
| Short | int, long, float, double |
| Char | int, long, float, double |
| Int | long, float, double |
| Long | float, double |
| Float | Double |

Konvertierungen von double in einen long-Wert kann nicht verlustfrei erfolgen.

**Explizite Typanpassung**

Die explizite Anpassung engt einen Typ ein, sodass diese Operation auch narrowing conversion genannt wird. Der gewünschte Typ für eine Typanpassung wird vor den umzuwandelnden Datentyp geschrieben. Der gewollte Datentyp ist geklammert.

int n = (int) 3.1415;

Richtiger Typecast: ergebnis = (double) summe/anzahl\*100

Richtiger Typecast: ergebnis = (double) (note1\*1 +note2\*2 +…)/anzahl

Richtiger Typecast: ergebnis = (int) ( 1.0456 + 2.078)

Falscher Typecast: ergebnis = (double) **(**(note1\*1 +note2\*2 +…)/anzahl**)**

Die äußeren Klammern müssen entfernt werden, sonst wird nur 3.0 bei dem Notendurchschnitt ausgegeben.

Falscher Typecast: int ergebnis = (int) 1.0456 + 2.078

## Mathematische Standardoperationen

Um Datenverarbeitung überhaupt erst durchführen zu können, bedarf es vieler mathematischer Berechnungen, die unter anderem mit den folgenden Operatoren durchgeführt werden können:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Bedeutung** | **Beispiel** |
| + | Addition | i = x+2; |
| - | Subtraktion | i = z –3; |
| \* | Multiplikation | x=5\*2; |
| / | Division | e=9/4; |
| % | Ganzzahliger Rest einer Division/  Modulo | i=10%3; i hat hier nach den Wert 1, weil 3\*3= 9, 10-9= Rest 1 |
| ++ | Inkrementieren | i=0;  i++;  i hat hiernach den Wert 1. |
| -- | Dekrementieren | i=0;  i--;  i hat hiernach den Wert –1. |

## Kommentare

Kommentare sind Texte, die nicht zum Programmcode selbst gehören, aber der Dokumentation des Programmcodes dienen, um dessen Verständnis zu erleichtern. Es gibt in Java drei Arten von Kommentaren:

**Einzeilige Kommentare**

Sie beginnen mit // und enden am Ende der aktuellen Zeile.

Sie sind einzeilig oder nehmen den hinteren Teil einer Zeile bis zum Zeilenende ein.

**Mehrzeilige Kommentare**

Sie beginnen mit /\* und enden mit \*/. Sie können sich über mehrere Zeilen erstrecken.

**Dokumentationskommentare**

Sie beginnen mit /\*\* und enden mit \*/ und können sich ebenfalls über mehrere Zeilen erstrecken.

**Beispiel eines Programmes: Addition zweier Zahlen**

**Programmausgabe:**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Addition zweier Zahlen**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Zahleneingabe:**

**--------------------------------------**

**Bitte geben Sie die 1. Zahl ein: 4**

**Bitte geben Sie die 2. Zahl ein: 5**

**Ausgabe Ergebnis:**

**--------------------------------------**

**4.0 + 5.0 = 9.0**

**Sourcecode:**

*/\**

*Addition zweier Zahlen*

*\*/*

*import java.util.\*; //package java.util wird importiert*

*public class AdditionZahlen {*

*double zahl1; //Deklaration der Variablen*

*double zahl2;*

*double summe;*

*public void eingabe() { // Methode Eingabe der Klasse AdditionZahlen*

*Scanner sc = new Scanner(System.in); //Erzeugt ein Scanner-Objekt für die Eingabe*

*System.out.println("\n\tZahleneingabe: \n\t--------------------------------------" );*

*System.out.print("\tBitte geben Sie die 1. Zahl ein: "); // kein Zeilenumbruch am Ende der Zeile*

*zahl1 = sc.nextDouble();*

*System.out.print("\tBitte geben Sie die 2. Zahl ein: ");*

*zahl2 = sc.nextDouble();*

*System.out.print("\n");*

*}*

*public void verarbeitung() { // Methode Verarbeitung der Klasse AdditionZahlen*

*summe = zahl1 + zahl2;*

*}*

*public void ausgabe() { // Methode Ausgabe der Klasse AdditionZahlen*

*System.out.println("\tAusgabe Ergebnis: \n\t--------------------------------------" ); // Zeilenumbruch am Ende der Zeile*

*System.out.println ("\t" + zahl1 + " + " + zahl2 + " = " + summe);*

*}*

*public static void main (String[] args) {*

*AdditionZahlen az1; // Objekt summe1 wird deklariert*

*az1 = new AdditionZahlen(); // Objekt summe1 wird erzeugt*

*System.out.print("\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");*

*System.out.print("\t\tAddition zweier Zahlen");*

*System.out.print("\n\t\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");*

*az1.eingabe(); // Methoden der Klasse werden aufgerufen*

*az1.verarbeitung();*

*az1.ausgabe();*

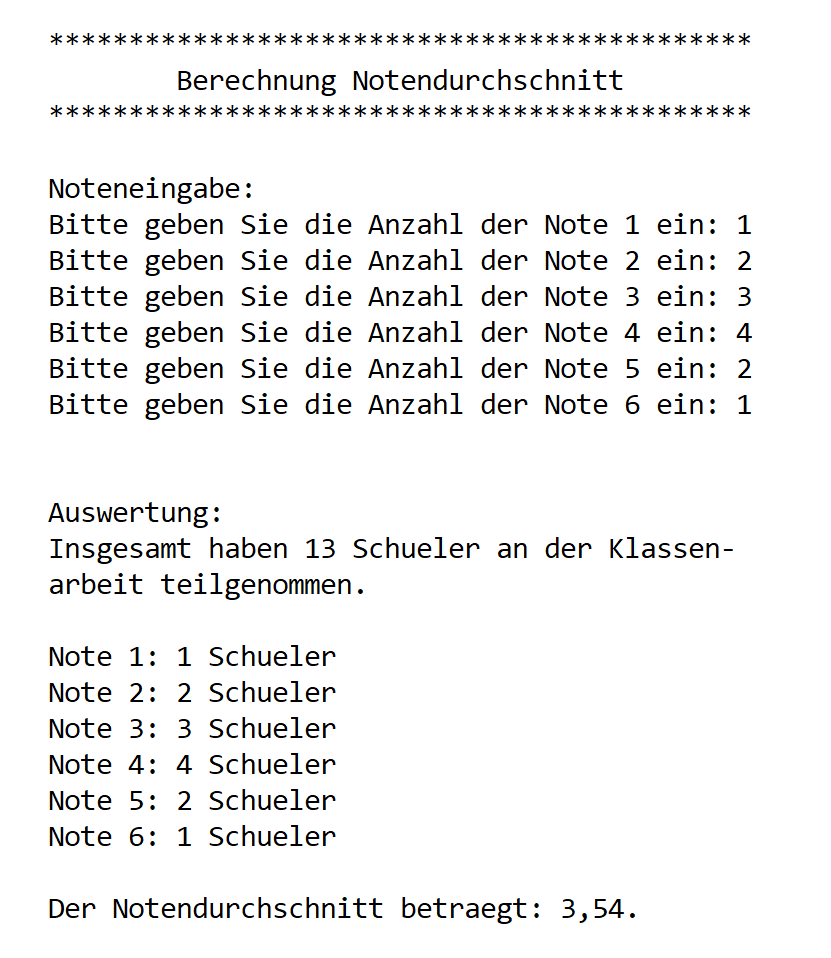
### Übung: Notenberechnung

Du wirst beauftragt, ein Programm zur Berechnung des Notendurchschnitts einer Klassenarbeit mit den Noten 1 bis 6 zu schreiben.

**Anforderungen**:

Zu Beginn des Programms soll der Benutzer die Anzahl der Schüler für die jeweiligen Notenstufen eingeben. Im Anschluss soll der Notendurchschnitt berechnet und ausgegeben werden.

Entwerfe die untenstehende Ein- und Ausgabe inkl. Titel möglichst detailgetreu.



Formatiere den Notendurchschnitt auf genau zwei Dezimalstellen. Gebe neben der Programmdokumentation am Anfang noch mindestens 3 Kommentare in den Source-Code ein!

## Hinweise zur Formatierung

Um den Notendurchschnitt auf zwei Stellen formatieren zu können, importiere bitte das Java-Package java.text. (**import java.text.\*;).** Dieses Package umfasst eine Klasse DecimalFormat, die dazu verwendet werden kann, um Zahlen bei der Ausgabe auf dem Bildschirm zu formatieren. DecimalFormat besitzt eine Methode format(), um z.B. Zahlen grundsätzlich mit 2 Kommastellen und einer Währungsbezeichnung auszugeben. Um ein neues Formatobjekt (z.B. df1) zu erstellen, kann z.B. folgende Syntax angewandt werden:

**DecimalFormat df1 = new DecimalFormat("#,##0.00 EUR“).**

Die Ausgabe der formatierten Zahl wird dann stets auf dem Umweg über die Methode format() des Formatobjektes z.B. **df1.format(variablename)** vorgenommen.*Dabei wird die Zahlenvariable an das Objekt übergeben und von dort an die Methode out.println der Klasse System (System.out.println(...)) weitergereicht.* Will man verschiedene Zahlenformate verwenden (etwa für Prozentzahlen), so muss für jedes Format ein neues DecimalFormat-Objekt (z.B. df2) erzeugt werden.

**#,##0.00 Bedeutung:**

* # druckt eine Ziffer, falls nicht anderes angegeben ist.
* 0 wenn vorhanden wird eine Ziffer ausgegeben, andernfalls eine 0
* , Gruppentrennzeichen, sprich Tausendertrennzeichen
* . Dezimaltrennzeichen

### Übung Benzinverbrauch

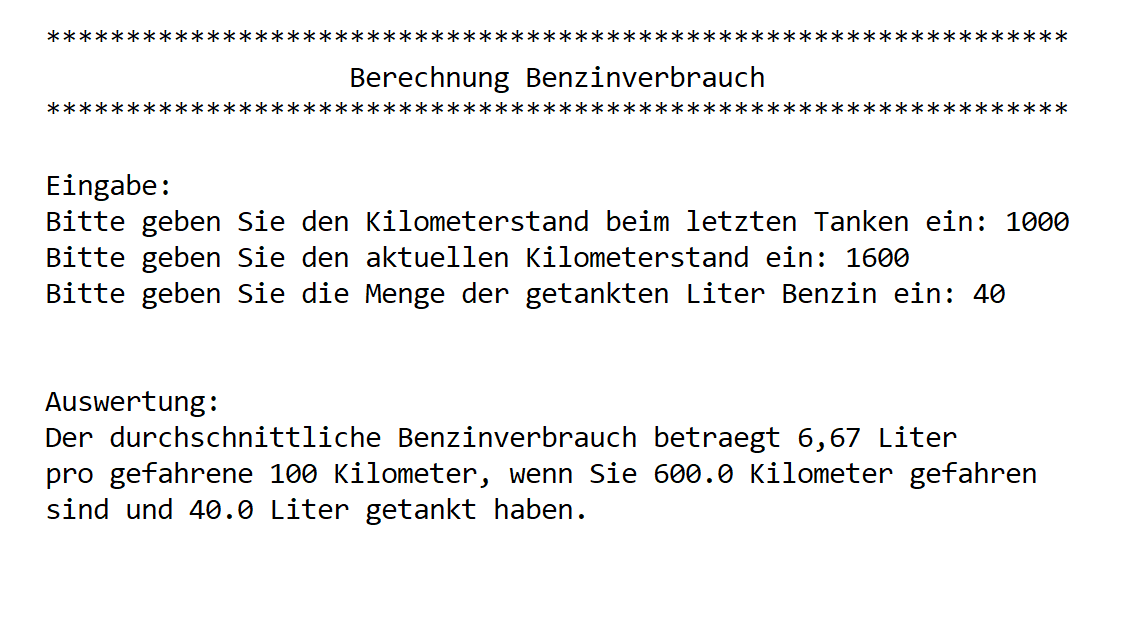
Du wirst beauftragt, ein Programm zur Berechnung des Benzinverbrauchs pro gefahrene 100 Kilometer zu entwickeln.

**Anforderungen**:

Zu Beginn des Programms soll der Benutzer den aktuellen Kilometerstand, den Kilometerstand beim letzten Tanken sowie die Menge der getankten Liter eingeben.

Am Ende des Programms sind die gefahrenen Kilometer sowie der durchschnittliche Verbrauch pro 100 Kilometer auszugegeben.

Entwerfe die unten stehende Ein- und Ausgabe inkl. Titel und copyright möglichst detailgetreu.



* Formatiere den Verbrauch auf genau zwei Dezimalstellen.
* Gib neben der Programmdokumentation am Anfang noch mindestens 3 Kommentare in den Source-Code ein!

### Übung Zinsen

Entwickel ein Struktogramm zur Berechnung der Zinsen.

**Anforderungen**:

Zu Beginn des Programms soll der Benutzer das Kapital, den Zinssatz sowie die Laufzeit (in Jahren) eingeben.

Im Anschluss sollen die Zinsen berechnet und inklusive den Eingabewerten (Kapital, Zinssatz und Laufzeit ausgegeben werden.

Nachdem du das Struktogramm entworfen hast, schreibst du bitte ein entsprechendes Programm mit Berücksichtigung der Namenskonventionen und inkl. Titel.

Formatiere das Kapital und die Zinsen mit 2 Dezimalstellen und dem Zusatz „EUR“, die Laufzeit mit 0 Dezimalstellen und dem Zusatz „Jahr(e)“ und den Zinssatz mit einer Dezimalstelle und dem Zusatz “%“.

Gib neben der Programmdokumentation am Anfang noch mindestens 3 Kommentare in den Source-Code ein!

# Fallunterscheidungen

In Java gibt es zwei Arten von Fallunterscheidungen. If-Anweisungen sind sinnvoll, wenn man richtige Bedingungen formulieren möchte. Soll eine Variable auf mehrere Werte geprüft werden, so empfiehlt sich ein Switch-Block.

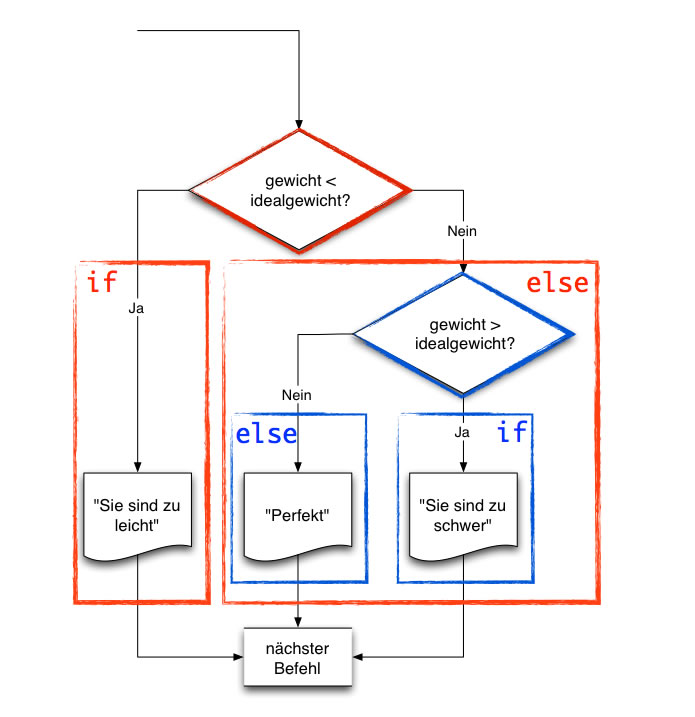
## If -Anweisung

Eine If-Anweisung ist ein Codeblock, welcher nur dann ausgeführt wird, wenn eine bestimmte Bedingung erfüllt wird. Der Aufbau einer if-Anweisung sieht folgendermaßen aus:

|  |  |
| --- | --- |
| Grundsätzlicher Aufbau | Beispiel |
| if (Bedingung1)  {  Befehlsblock1;  }  else if(Bedingung2)  {  Befehlsblock2;  }  else  {  Befehlsblock3;  } | if (gewicht<idealgewicht) {  system.out.println(„Du bist zu leicht“);  }  else if(gewicht >idealgewicht);  {  System.out.println(„Du bist zu schwer“);  else {  system.out.println(„Dein Gewicht ist perfekt“);  } |

Die Bedingung muss erfüllt werden (wahr sein), damit der darauffolgende Codeblock ausgeführt wird. Wenn also die Bedingung 1 erfüllt ist, dann führe den Befehlsblock1, wenn Bedingung 2 wahr ist, dann den Befehlsblock2. Wenn allerdings keine Bedingung erfüllt werden kann, dann wird der letzte Else-Block ausgeführt.

**Struktogramm**

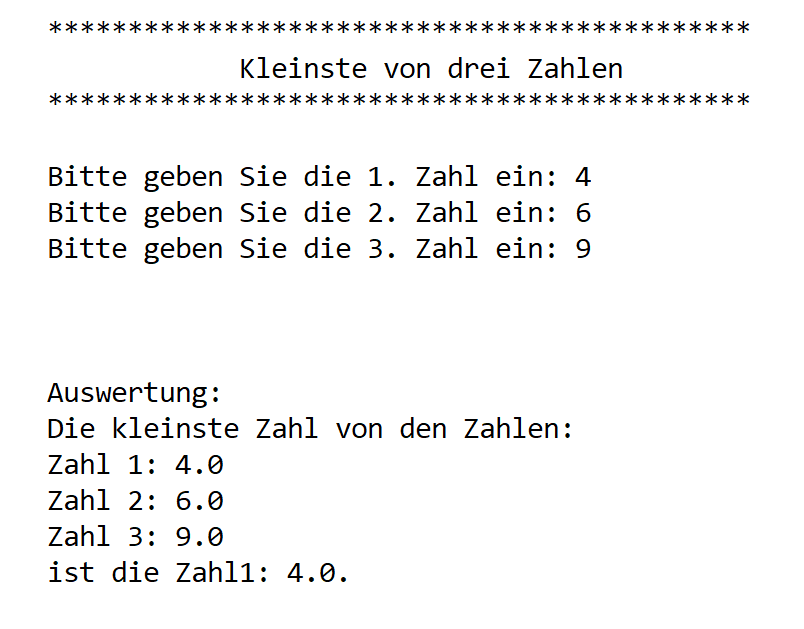


### Übung: Kleinste von 3 Zahlen

Entwirf ein Strukturgramm, welches nach Eingabe von 3 Zahlen, die kleinste der drei Zahlen anzeigt.

2

2



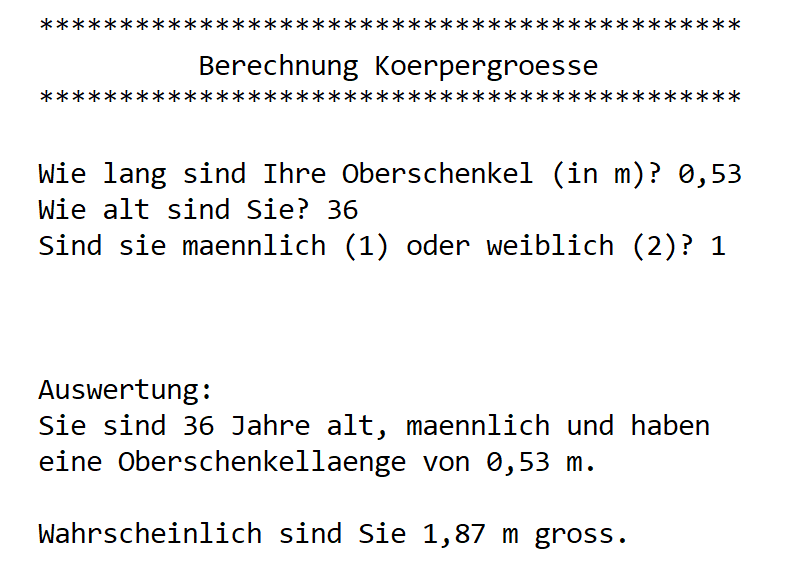
Programmiere im Anschluss das Programm in Java.

### Übung: Berechnung Körpergröße

Entwickel ein Struktogramm und ein Programm, welches die Körpergröße anhand der Oberschenkellänge und des Alters ermittelt.

Anforderungen:

* Zu Beginn des Programms soll der Benutzer seine Oberschenkellänge, sein Geschlecht und sein Alter eingeben.
* Im Anschluss soll die Körpergröße anhand folgender Kriterien ermittelt werden:
* Die empirische und statistische Körpergröße bei Männern beträgt das 2,238-fache der Oberschenkellänge + 0,69089.
* Die empirische und statistische Körpergröße bei Frauen beträgt das 2-fache der Oberschenkellänge + 0,61417.
* Die Körpergröße bei Männern und Frauen nimmt ab einem Alter von 30 Jahren 0,06 cm pro Jahr ab, d.h. bei 31jährigen nimmt die Körpergröße um 0,06 cm, bei 32jährigen um 0,12 cm usw. ab.
* Programmiere die folgende Auswertung möglichst detailgetreu:



Formatiere die Körpergröße so, dass zwei Dezimalstellen angezeigt werden.

## Der Restwert-Operator (Modulo)

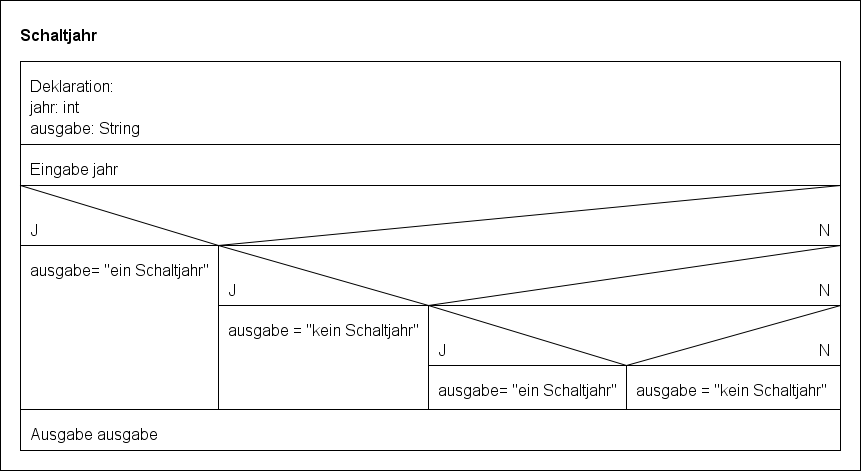
Möchte man den Rest einer Division ermitteln, verwendet man den Modulo-Operator. Das Zeichen des Operators ist %.

Ein Beispiel: 15/4 = 3 Rest 3

Die 4 passt 3-mal in die 15 hinein, weil 3\*4 = 12. 15-12=3. Es bleibt also ein Divisionsrest von 3 über.

### Übung Schaltjahr

Ergänze die Bedingungen in dem nachfolgenden Struktogramm zur Überprüfung der Schaltjahreseigenschaft.



**Hinweis:** Ein Jahr ist ein Schaltjahr, wenn die Jahreszahl durch 4 teilbar ist. Ist die Jahreszahl allerdings auch durch 100 teilbar, so handelt es sich nicht um ein Schaltjahr, es sei denn, sie ist gleichzeitig durch 400 teilbar.

Trage den Quelltext für die Methode auswertung unten ein! Verwende dazu die if-else if–Alternative.

import java.util.\*;

import java.text.\*;

**public class** Schaltjahr {

**int** jahr;

String ergebnis;

**public void** eingabe(){  
 Scanner sc = new Scanner(System.in);

System.out.print("\tBitte geben Sie die Jahreszahl ein: ");

jahresZahl = sc.nextInt();

}

**public void** verarbeitung(){

if (jahresZahl % 4 == 0 && jahresZahl % 100 != 0) {

System.out.print("Das Jahr " + jahresZahl + " ist ein Schaltjahr!“);

} else if (jahresZahl

}

**public void** ausgabe(){

System.out.println(ergebnis);

}

**public static void** main(String[] args) {

Schaltjahr sj1 = **new** Schaltjahr();

sj1.eingabe();

sj1.auswertung();   
 System.out.println(ergebnis);

}

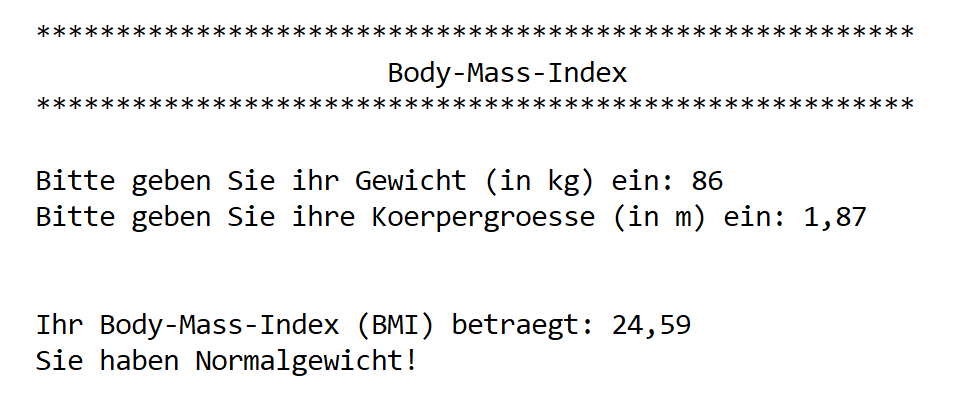
}

### Übung BodyMassIndex

Erstelle ein Struktogramm, welches dem Benutzer anzeigt, ob er unter-, normal- oder übergewichtig ist.

**Anforderungen**:

* Zu Beginn des Programms soll der Benutzer Gewicht und seine Körpergröße eingeben.
* Im Anschluss soll der Body-Mass-Index (bmi) anhand der Formel:  
  bmi = gewicht / koerpergroesse² berechnet werden.
* Beträgt der bmi unter 19 hat die Person Untergewicht.
* Beträgt der bmi zwischen 19 und unter 25 hat die Person Normalgewicht.
* Beträgt der bmi zwischen 25 und unter 30 hat die Person Übergewicht.
* Beträgt der bmi 30 und mehr hat die Person starkes Übergewicht.
* Am Ende des Programms soll z.B. folgende Auswertung erscheinen:



* Formatiere den bmi so, dass zwei Dezimalstellen angezeigt werden.

### Übung Notenberechnung

Die Geschäftsleitung der product GmbH beauftragt dich, ein Programm zur Berechnung von Klassenarbeitsnoten zu schreiben.

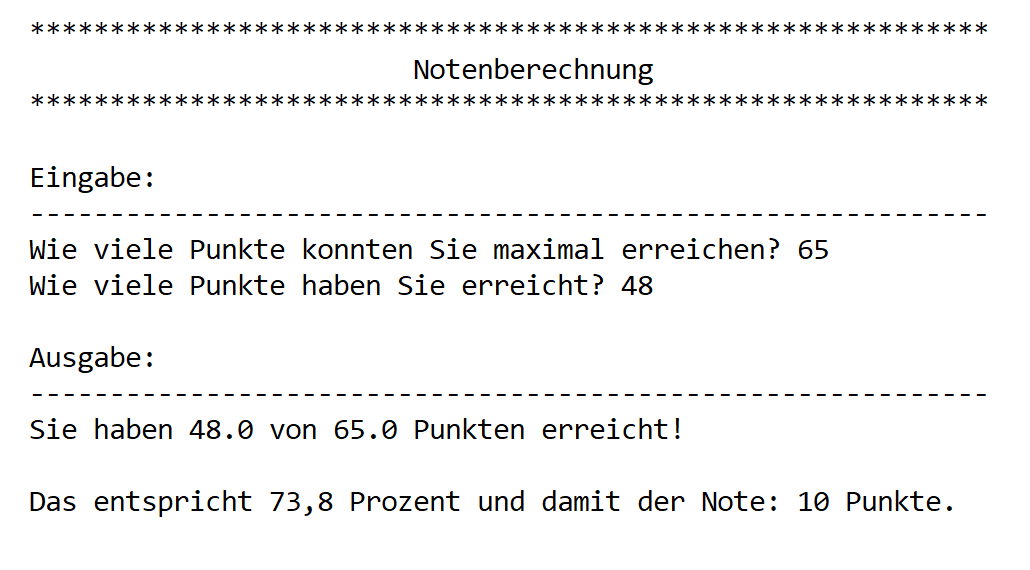
**Anforderungen**:

BD14565_ Zu Beginn des Programms soll der Benutzer die maximale Punktzahl sowie die erreichte Punktzahl der Klassenarbeit eingeben.

BD14565_ Im Anschluss soll die entsprechende Note inklusive der erreichten Prozentzahl angezeigt werden.

BD14565_ Berücksichtige bei der Vergabe der Note folgende Prozentverteilung:

|  |  |
| --- | --- |
| ab | Noten |
| 95% | **15** |
| 90% | **14** |
| 85% | **13** |
| 80% | **12** |
| 75% | **11** |
| 70% | **10** |
| 65% | **9** |
| 60% | **8** |
| 55% | **7** |
| 50% | **6** |
| 45% | **5** |
| 40% | **4** |
| 33% | **3** |
| 27% | **2** |
| 20% | **1** |
| 0% | **0** |



**Hinweise:**

* Programmiere die Auswahl mit der if-else if-Alternative.
* Verwende möglichst wenige Ausgaben-Zeilen.

## Logische Operatoren

Um Vergleiche oder logische Verknüpfungen durchführen zu können, bedarf es einiger Operatoren, die in der folgenden Tabelle aufgeführt sind (i ist hierbei eine **boolsche** Variable, die den Wert true oder false annehmen kann).

|  |  |  |
| --- | --- | --- |
| **Operator** | **Bedeutung** | **Beispiel** |
| < | ist kleiner als | i=5<6;  i hat hiernach den Wert true. |
| > | ist größer als | i = 5>6;  i hat hiernach den Wert false. |
| <= | ist kleiner gleich als | i = 5<=5;  i hat hiernach den Wert true. |
| >= | ist größer gleich als | i = 5 >=5;  i hat hiernach den Wert true. |
| == | ist gleich | i = 5==6;  i hat hiernach den Wert false. |
| != | ist ungleich | i = 5 !=6;  i hat hiernach den Wert true. |
| && | logisches UND | i = ((5>6)&&(6>5));  i hat hiernach den Wert false. |
| || | logisches ODER | i = ((5>6)||(6>5));  i hat hiernach den Wert true. |
| ! | Nicht – Operator | I = !(5>6);  I hat hiernach den Wert true. |

### Übung: „and“ und „or“

Welche Ausgaben liefert das folgende Programm, wenn man an der markierten Stelle jeweils die Programmzeilen a bis h einsetzt?

1. if (y <= 2 && y == 3) {
2. if (y <= 2 || y == 3) {
3. if (y <= 3 || y != 2 && y >= 4) {
4. if ((y <= 3 || y != 2 ) && y >= 4) {
5. if (x < y && w > z && x == 0 && w!=2 || x >= -1){
6. if (x != 2 && y > 3) {
7. *true false -> false* if (z >= 1 || z <= 3){
8. *true false -> true true für alle Zahlen, daher eine unsinnige Bedingung*if (y <= 1 && y >=6 ){

*false false -> false*

false für alle Zahlen, daher eine unsinnige Bedingungpublic class Aufgabe {

public static void main (String args[]) {

int w = 3;

int x;

int y = 3;

int z = 4;

// **Hier die Programmzeilen a, b, c usw. einsetzen!**

System.out.println("true!" );

}

**else** {

System.out.println("false!" );

}

}

}

***Hinweise***

*Operatoren vorstellen:*

*mathematische Operatoren: + - \* \*

*vergleichende Operatoren:*

*gleich: ==*

*größer (gleich): > =>*

*kleiner (gleich): < =<*

*ungleich: !=*

*logische Oerpatoren:*

*AND: && (alle Bedingungen müssen erfüllt sein)*

*OR: || (eine Bedingung muss erfüllt sein)*

*Beispiel: if (name == „Meier“ && plz = 65189)*

*Nach Vorstellung der Operatoren verschachtelte if … else-Struktur mit logischen Operatoren für die kleinste Zahl entwickeln lassen, max. 2 if-else-Bedingungen.*

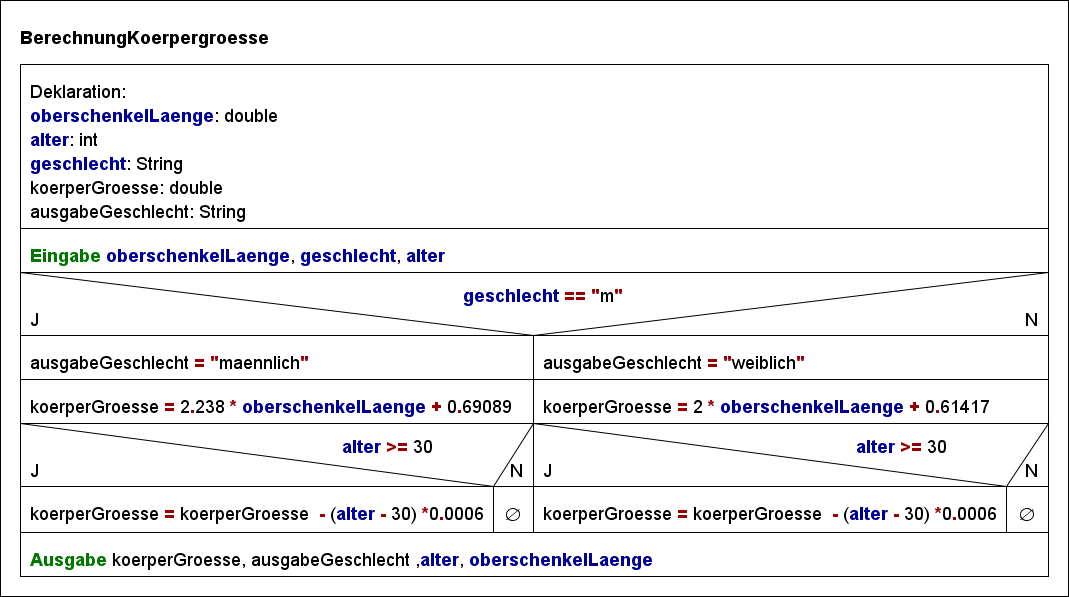
***Beispiel: zahl1=2, zahl2 = 2***

*if(zahl1 ==1 && zahl2 ==2 || zahl2 >10)*

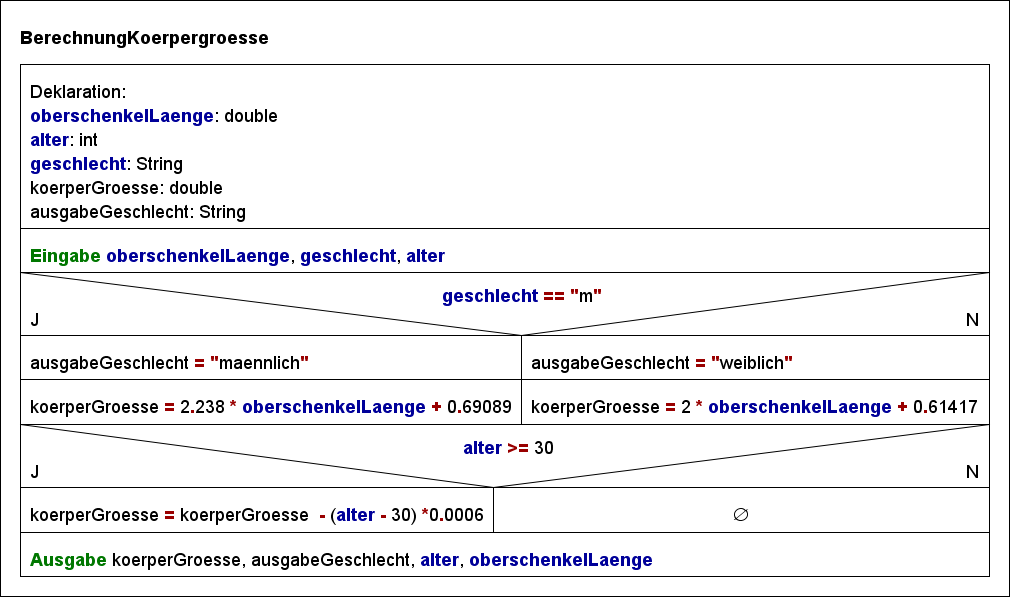
*false true false -> false*

*if(zahl1 ==1 && (zahl2 == 2 || zahl2 > 10))*

*false true false -> true*

**

***Optimale Lösung:***

**

## Switch-Blöcke

Eine weiter Art der Fallunterscheidung stellen die sogenannten „Switch-Blöcke“ dar. Ein Switch-Block prüft eine Variable auf verschieden Werte. Diese Werte sind die sogenannten „Cases“ (Fälle), unter welchen entsprechenden Anweisungen befinden, die dann ausgeführt werden, wenn die zu prüfende Variable denselben Wert beinhaltet wie ein „Case“.

Beispiel:

int month = 3;

switch(month) {

case 1: System.out.println(„Januar“);break;

case 2: System.out.println(„Februar“);break;

case 3: System.out.println(„März“);break;

default: System.out.println(„kein gültiger Monat!“); break;

}

Wichtig: Nach jedem case sollte das Schlüsselwort „break“ mit angefügt werden. Sollte man dies nicht tun, so werden alle nachfolgenden cases auch noch ausgeführt, und zwar solange bis alle nachfolgenden cases ausgeführt wurden oder das erste Mal das Schlüsselwort „break“ erschienen ist. Wenn kein case erfüllt wird, dann wird einfach aus dem Switch-Block gesprungen und es geschieht nichts, es sei denn man hat einen „default“-Case definiert. Der „default“- case wird immer als letzte Alternative ausgeführt.

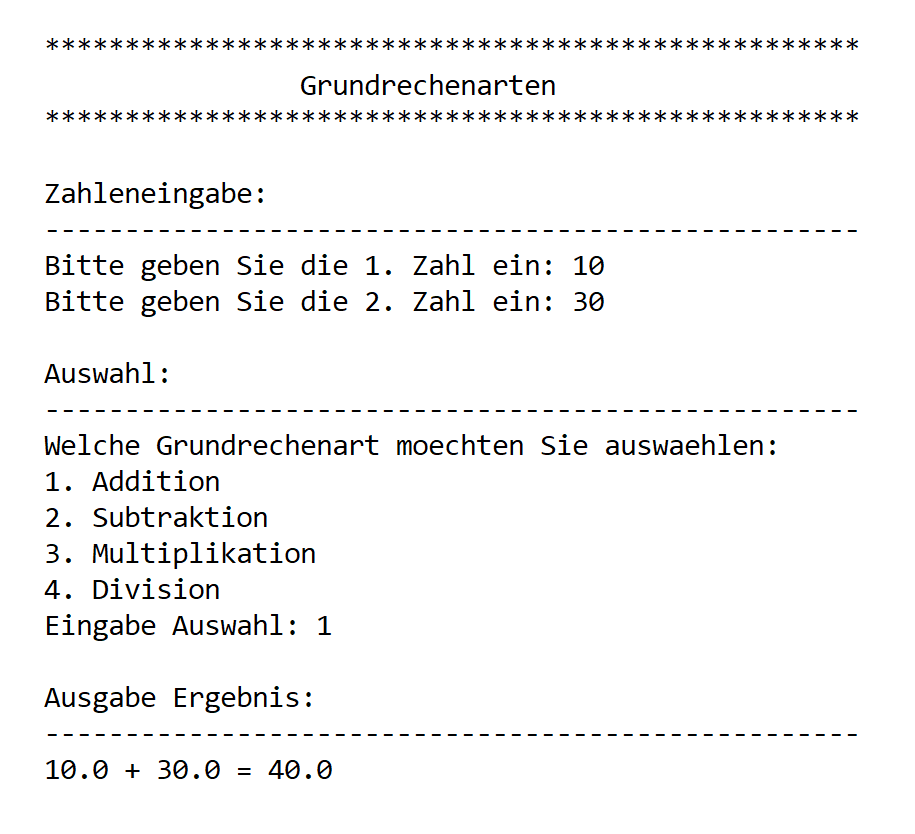
### Übung: Grundrechenarten mit zwei Zahlen

Sie sollen ein Programm entwickeln, bei dem der Benutzer nach Eingabe von zwei Zahlen auswählen kann, welche Grundrechenart er mit den beiden Zahlen ausführen möchte.

**Anforderungen**:

Nach Eingabe der Zahlen soll der Benutzer eine der Grundrechenarten Addition, Subtraktion, Multiplikation und Division auswählen.

Im Anschluss soll das entsprechende Ergebnis angezeigt werden.



Gibt der Benutzer eine andere, nicht definierte Kennziffer ein, soll die Ausgabe „Sie haben eine falsche Kennziffer eingegeben! Das Programm wird abgebrochen“ erscheinen.

Programmiere die Auswahl mit der switch-Alternative.

### Übung: Bußgeldberechnung

Du sollst ein Bußgeldberechnungsprogramm schreiben. Das Programm soll die Geschwindigkeitsüberschreitung abzüglich einer Toleranz berechnen und dem Betrachter das Bußgeld, die Strafpunkte sowie das Fahrverbot in Monaten anzeigen. Dabei sind folgende Angaben zu berücksichtigen.

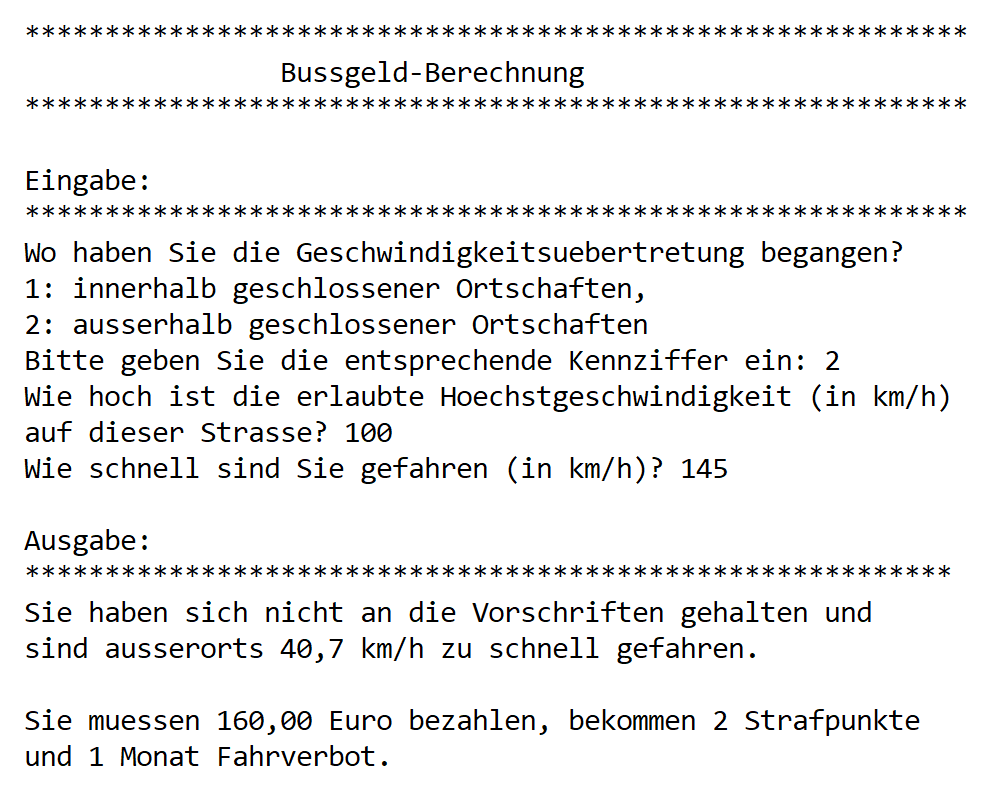
**Innerhalb geschlossener Ortschaften (gilt auch für 30 km/h-Zone!)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Geschwindigkeitsüberschreitung** | **Bußgeld** | **Strafpunkte** | **Fahrverbot** |
| bis 10 km/h | 30,00 EUR |  |  |
| über 10 bis 15 km/h | 50,00 EUR |  |  |
| über 15 bis 20 km/h | 70,00 EUR | 1 Strafpunkt |  |
| über 20 bis 25 km/h | 115,00 EUR | 1 Strafpunkt |  |
| über 25 bis 30 km/h | 180,00 EUR | 1 Strafpunkt |  |
| über 30 bis 40 km/h | 260,00 EUR | 2 Strafpunkte | 1 Monat Fahrverbot |
| über 40 bis 50 km/h | 400,00 EUR | 2 Strafpunkte | 1 Monat Fahrverbot |
| über 50 bis 60 km/h | 600,00 EUR | 2 Strafpunkte | 2 Monate Fahrverbot |
| über 60 bis 70 km/h | 700,00 EUR | 2 Strafpunkte | 3 Monate Fahrverbot |
| über 70 km/h | 800,00 EUR | 2 Strafpunkte | 3 Monate Fahrverbot |

**Außerhalb geschlossener Ortschaften (z.B. Landschaftstraßen, Autobahnen)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Geschwindigkeits- überschreitung** | **Bußgeld** | **Strafpunkte** | **Fahrverbot** |
| bis 10 km/h | 20,00 EUR |  |  |
| über 10 bis 15 km/h | 40,00 EUR |  |  |
| über 15 bis 20 km/h | 60,00 EUR |  |  |
| über 20 bis 25 km/h | 100,00 EUR | 1 Strafpunkt |  |
| über 25 bis 30 km/h | 150,00 EUR | 1 Strafpunkt |  |
| über 30 bis 40 km/h | 200,00 EUR | 2 Strafpunkt | 1 Monat Fahrverbot |
| über 40 bis 50 km/h | 320,00 EUR | 2 Strafpunkte | 1 Monat Fahrverbot |
| über 50 bis 60 km/h | 480,00 EUR | 2 Strafpunkte | 1 Monat Fahrverbot |
| über 60 bis 70 km/h | 600,00 EUR | 2 Strafpunkte | 2 Monate Fahrverbot |
| über 70 km/h | 700,00 EUR | 2 Strafpunkte | 3 Monate Fahrverbot |

Von dem gemessenen Geschwindigkeitswert werden als Toleranz 3 km/h abgezogen, wenn das Fahrzeug unter 100 km/h fährt, ab 100 km/h werden 3 % als Toleranz abgezogen.

**Anforderungen**:

* Programmiere die   
  Fallunterscheidungen mit der switch-case und der if-  
  else if-Alternative.
* Die Toleranzwerte sind als   
  Konstanten zu definieren.
* Verwende möglichst   
  wenige Ausgabezeilen.
* Unterscheide bei der   
  Ausgabe der Monate   
  zwischen „Monat“ und   
  „Monate“ bzw. „Strafpunkte“   
  und „Strafpunkt“.
* Die Strafpunkte und das Fahrverbot sind als Integer-Werte zu deklarieren.

# Schleifen

Sehr oft kommt es vor, dass man bestimmte Teile des Programmcodes mehr als nur einmal ausführen möchte. Hierfür gibt es 4 verschiedene Arten von Schleifen.

## While Schleife (kopfgesteuert)

Die While Schleife ermöglicht uns einen Codeblock solange auszuführen, wie eine bestimmte Bedingung erfüllt wird. Sie ist sozusagen eine If-Anweisung, die sich von selbst immer wieder wiederholt.

|  |  |
| --- | --- |
| Grundsätzlicher Aufbau | Beispiel |
| while (Bedingung)  {  Befehlsblock;  } | Int zahl = 0;  While (zahl<5){  System.out.println(zahl);  zahl++;  } |

Im Beispiel besagt die Bedingung, dass die Zahl kleiner als 5 sein muss. Jedes Mal, wenn alle Anweisungen innerhalb des Schleifenblocks ausgeführt worden sind, wird geprüft, ob die Bedingung noch erfüllt ist. Ergibt diese Prüfung ein „wahr“, wird der Schleifenblock ein weiteres Mal durchlaufen. In diesem Beispiel wird die Schleife 5-mal durchlaufen.

## Do-While Schleifen

Im Gegensatz zur While Schleife wird hier die Bedingungsüberprüfung am Ende der Schleife durchgeführt. Man spricht hier von **kopfgesteuerten und fußgesteuerten Schleifen.**

|  |  |
| --- | --- |
| Solange | Befehlsblock |
| Bedingung wahr |

|  |  |
| --- | --- |
| Solange | Bedingung wahr |
| Befehlsblock |

Bei der While Schleife (kopfgesteuert) wird die Abbruchbedingung im Schleifenkopf notiert und somit vor Ausführung des ersten Schleifendurchlaufs überprüft. Dies bewirkt, dass die in der Schleife notierten Anweisungen nur ausgeführt werden, wenn die Abbruchbedingung bei Erreichen der Schleife true ergibt. Ist dies nicht der Fall, so wird die komplette Schleife übersprungen.  
Das 'Gegenmodell', ist die [do-while-Schleife](https://javabeginners.de/Schleifen_und_Verzweigungen/do_-while_-Schleife.php) (fußgesteuert), bei der die Anweisungen mindestens einmal ausgeführt werden.

|  |  |
| --- | --- |
| Grundsätzlicher Aufbau | Beispiel |
| Do  {  Befehlsblock;  }  while (Bedingung); | Int zahl = 0;  do  {  System.out.println(zahl);  zahl++;  }  while (zahl<5); |

**Unterschied:**

Würden wir in unserem Beispiel den Wert der Variable Zahl statt 0 auf 10 setzten, so käme bei der While Schleife keine Ausgabe auf die Konsole, während bei der Do-While Schleife eine 10 ausgegeben würde.

### Übung: Mittelwertberechnung

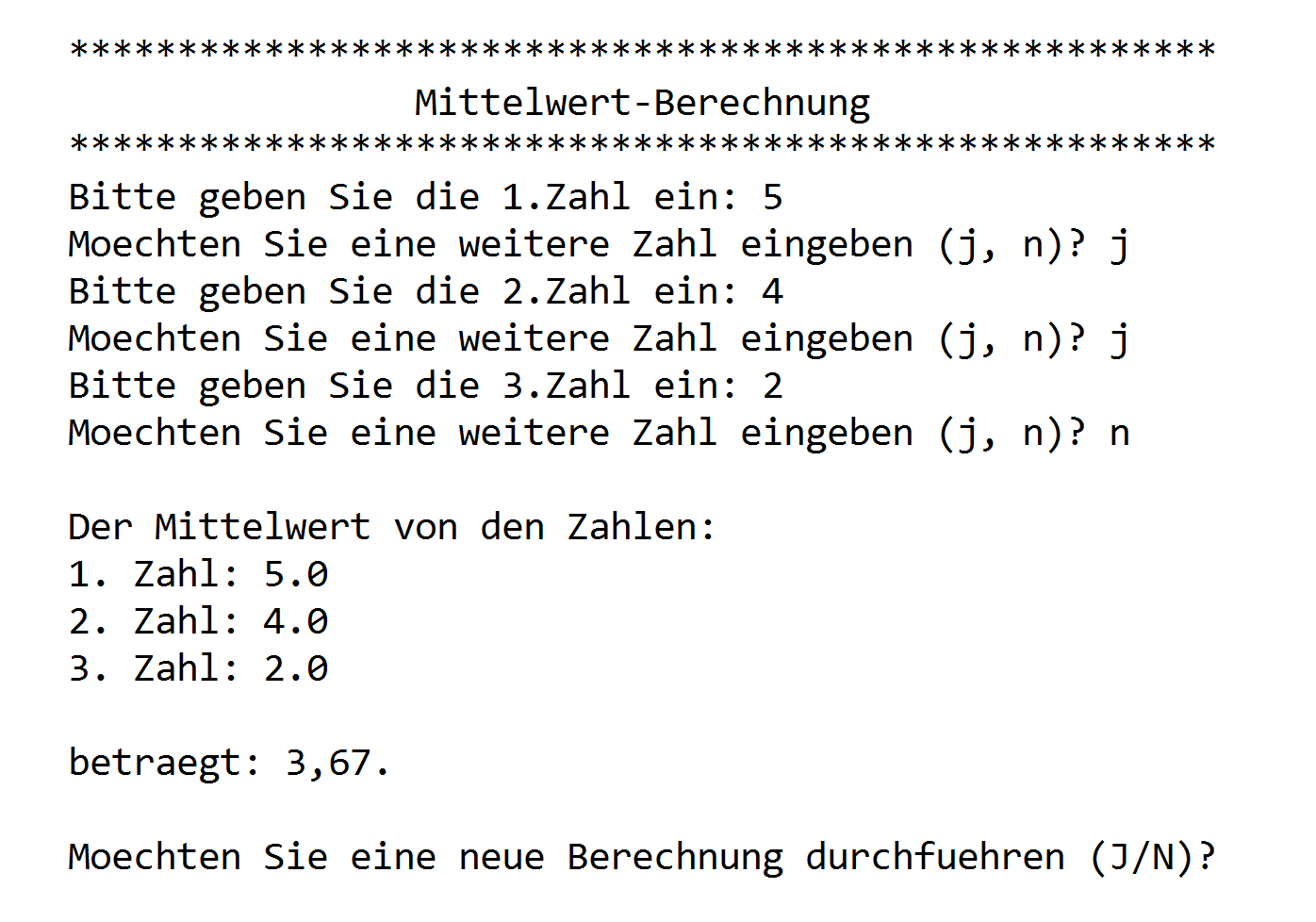
**Anforderungen**:

* Beim Start des Programms soll der Benutzer solange zur Eingabe einer Zahl aufgefordert werden, bis er auf die Frage „Möchten Sie eine weitere Zahl eingeben?“ „Nein“ eingibt.
* Am Ende des Programms sind die Summe der eingegebenen Zahlen und der Mittelwert auszugeben.

Passe das Programm zur Mittelwertberechnung wie folgt an:

**Anforderungen**:

* Der Benutzer soll am Ende des Programms gefragt werden, ob er das Programm wiederholen möchte. Wählt er „J“ oder „j“ aus, soll das Programm wiederholt werden.
* Der Mittelwert ist auf zwei Stellen hinter dem Komma zu begrenzen.



### Übung: Zinseszinsberechnung

Entwerfe ein Struktogramm zur Zinseszinsberechnung.

**Anforderungen**:

* Zu Beginn des Programms soll der Benutzer das Kapital, den Zinssatz und die Anlagedauer in Jahren eingeben.
* Im Anschluss soll mit Hilfe der Zinsformel: zinsen = kapital\*zinssatz/100 die Zinsen für die gesamte Anlagedauer berechnet werden. Beachte hierbei, dass die Zinsen jährlich ausgeschüttet werden und damit das Kapital erhöhen.
* Am Ende des Programms soll dem Benutzer das Startkapital, der Zinssatz, die Anlagedauer, die Zinsen, die sich im Laufe der Anlagedauer angesammelt haben und das Endkapital angezeigt werden.
* Der Benutzer soll am Ende des Programms gefragt werden, ob er das Programm wiederholen möchte.

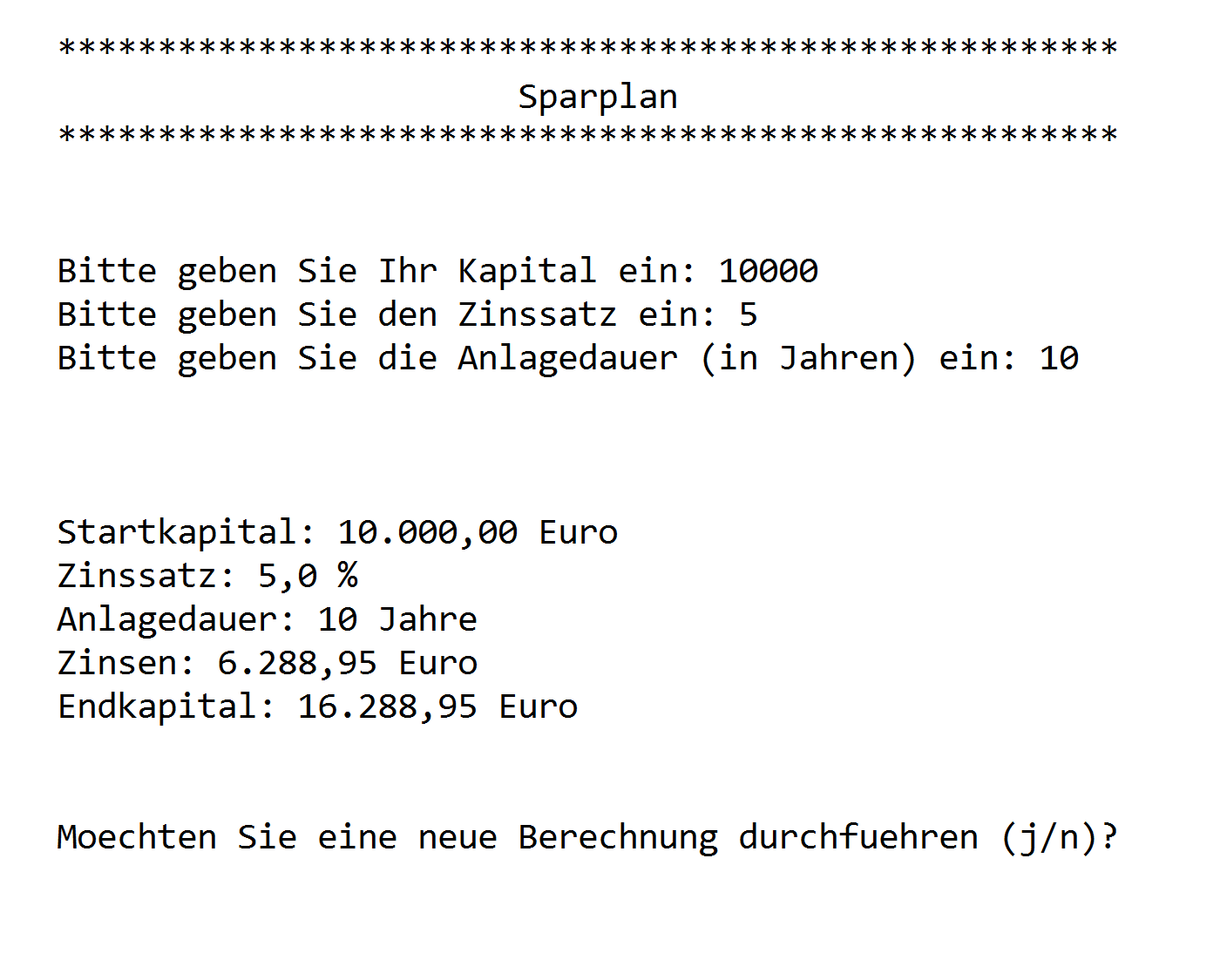
**Beispiel**: Bei einem Startkapital von 10.000,00 EUR ergibt sich bei einem Zinssatz von 5 % und einer Anlagedauer von 10 Jahren ein Endkapital von 16.288,95 EUR.

### Übung: Erweiterung Aufgabe Zinseszinsberechnung

Codiere das Struktogramm aus der Aufgabe 1 in Java.

**Weitere Anforderungen**:

* Unterscheide bei der Ausgabe zwischen „Jahr“ und „Jahre“ und formatiere die Zahlen entsprechend.



***Hinweise zur Vorgehensweise***

* *Struktogramm mit den Schülern gemeinsam entwickeln*
* *Source-Code-Syntax an der Tafel zeigen*
* *Schüler die Aufgabe 1 programmieren lassen*
* *Für die Aufgabe 2 noch zwei Hinweise geben:*
* *Aufruf von Methoden in den Methoden (this.verarbeitung())*
* *Wiederholung der Klasse in der main-zeigen*
* *Aufgabe 2 anhand der Aufgabe 1 gemeinsam an der Tafel entwickeln*
* *Besonderheiten: static Variablen, Aufsummierung der Werte im Wiederholungsfall,*

**

### Übung: Kostenvergleich

Die Geschäftsleitung der product GmbH plant die Erschließung eines neuen Absatzgebietes. Zur Organisation des Vertriebs gibt es zwei Alternativen:

**Alternative 1: Handlungsreisende**

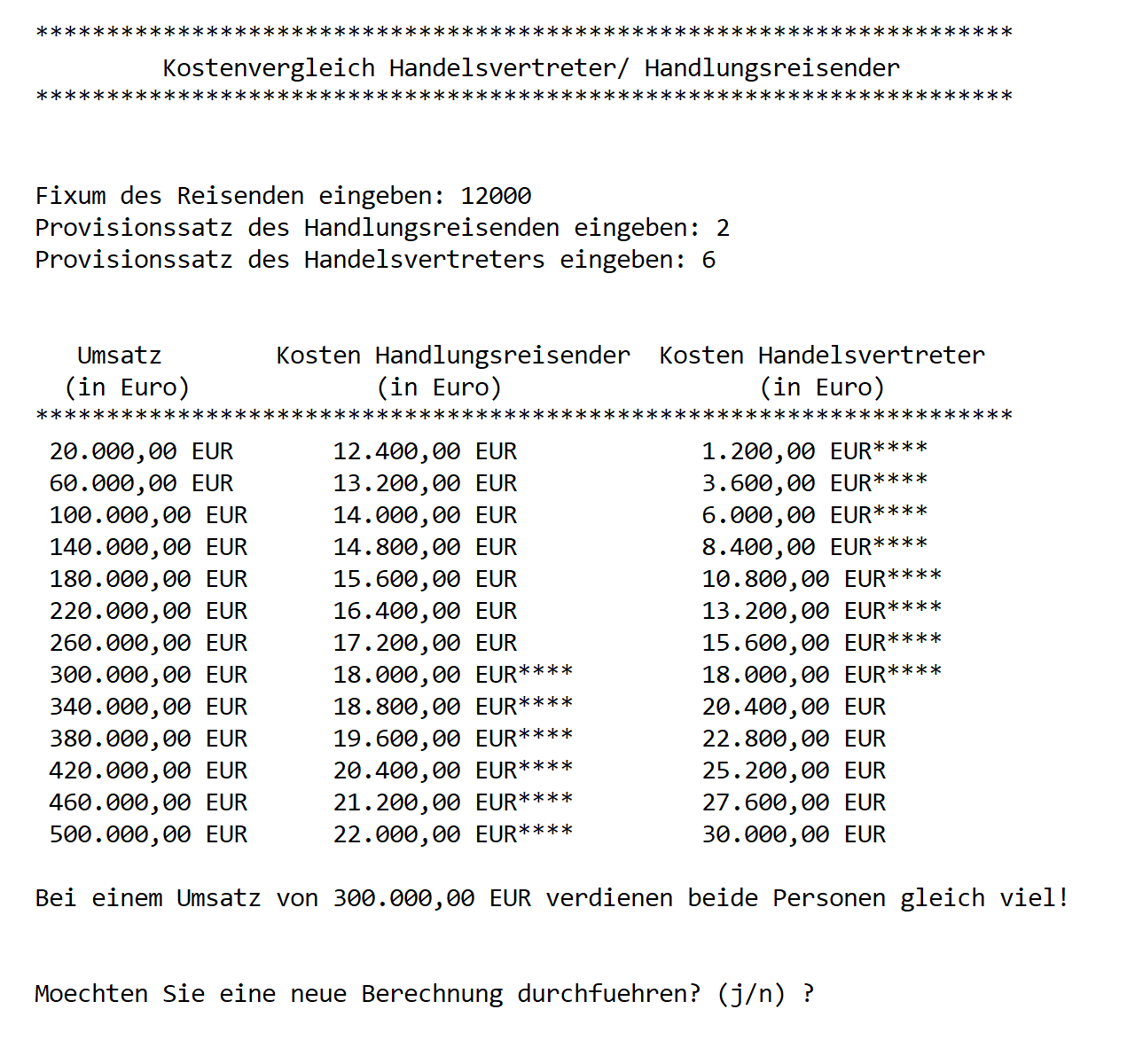
Handlungsreisende sind fest beim Unternehmen angestellt. Das Gehalt der Reisenden setzt sich aus einem Festbetrag (Fixum) und einem umsatzabhängigen Provisionsanteil zusammen.

**Alternative 2: Handelsvertreter**

Handelsvertreter sind selbständige Kaufleute und arbeiten ausschließlich auf Provisionsbasis. Das Unternehmen müsste die Handelsvertreter nur dann bezahlen, wenn auch tatsächlich Ware verkauft wurde. Allerdings ist diese Provision deutlich höher als für Reisende, denn der Handelsvertreter trägt sein eigenes Unternehmerrisiko.

**Anforderungen**:

* Um dem Unternehmen die Entscheidung für eine der Alternativen zu erleichtern, soll ein Java-Programm entwickelt werden, dass in einer Tabelle die Kosten für Handlungsreisende und Handelsvertreter in Abhängigkeit vom Umsatz gegenüberstellt.
* In der Tabelle soll der Umsatz bei 20.000,00 EUR beginnen und in Schritten von 40.000,00 EUR auf maximal 500.000,00 EUR gesteigert werden. Die für jeden Umsatz günstigere Alternative wird mit \*\*\* hinter dem Wert der Kosten gekennzeichnet.
* Für einen Handlungsreisenden mit einem Fixum von 12.000,00 EUR und einer Provision von 2% einerseits und einer Provision für einen Handelsvertreter von 6 % andererseits ergibt sich die folgende Ausgabe:



* Schreibe anschließend ein Programm zur Lösung dieses Problems und programmiere die obenstehende Ausgabe möglichst detailgetreu.

### Übung: Kennwort-Eingabe

Du wirst beauftragt, ein Programm für einen Login-Vorgang zu schreiben.

**Anforderungen**

* Beim Start des Programms soll der Benutzer nach seinem Kennwort (z.B. “FLS“ gefragt werden.
* Gibt der Benutzer das richtige Kennwort ein, soll die Meldung „Sie haben sich erfolgreich angemeldet!“ erscheinen.
* Gibt der Benutzer beim 3. Versuch immer noch ein falsches Kennwort ein, soll die Meldung „Ihre Benutzerkennung wird gesperrt!“ erscheinen. Im Anschluss wird das Programm abgebrochen.
* Das Kennwort „FLS“ sowie die maximale Anzahl der Versuche sind als Konstanten zu definieren.



**Hinweise:**

* Die Anzahl der Versuche und das richtige Kennwort sollen als Konstanten festgelegt werden.
* Auf die Methode verarbeitung() soll verzichtet werden.
* Die Meldungen „Das Kennwort ist leider falsch!“ bzw. „Sie haben noch x von 3 Versuchen!“ können in der eingabe()-Methode ausgegeben werden.
* Die Meldungen „Sie haben sich erfolgreich angemeldet!“ bzw. „Ihre Benutzerkennung wird gesperrt!“ sind in der ausgabe()-Methode zu implementieren.
* Ein String-Vergleich geht nicht mit dem Operator ==, sondern benötigt die String-Methode equals().
* Beispiel: String text = “FLS“;
* if(text.equals(“FLS“)==true){ //True kann in diesem Fall auch weggelassen werden.

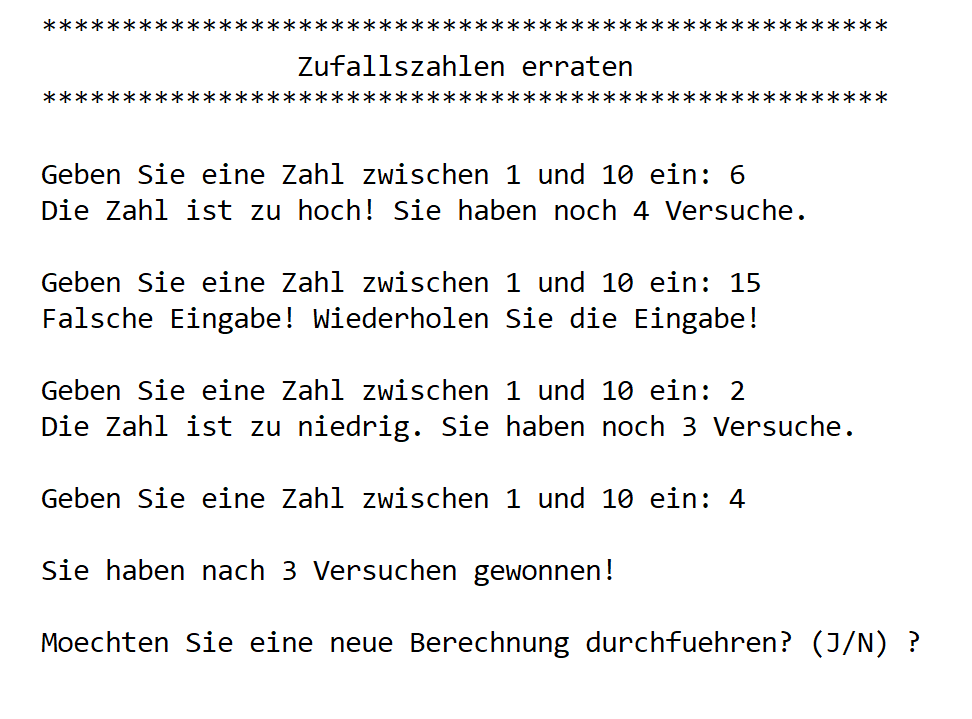
}

### Übung: Zufallszahlen

Du wirst beauftragt, ein Spiel zum Erraten von Zufallszahlen zu schreiben.

**Anforderungen**:

* Der Computer ermittelt über einen Zufallszahlengenerator eine Geheimzahl zwischen 1 und 10. Danach wird der Benutzer aufgefordert, seinerseits eine Zahl einzugeben, die der Computer mit der Zufallszahl vergleicht. Hat der Benutzer die Geheimzahl erraten, erhält er eine Erfolgsmeldung. Ist die Zahl kleiner bzw. größer als die Geheimzahl, erhält der Benutzer eine entsprechende Meldung und kann erneut eine Zahl eingeben.
* Insgesamt soll der Benutzer 5 Versuche zum Erraten der Zahl haben.



* Schreibe ein Programm zur Lösung dieses Problems und programmiere die obenstehende Ausgabe möglichst detailgetreu.
* Definiere die Anzahl der Versuche, die Ober- und Untergrenze der Zufallszahlen als Konstanten.
* Entwirf vorher ein Struktogramm.

**Hinweise:**

* Zur Generierung der Zufallszahl verwende bitte die mathematische Funktion

zufallszahl = (int) ((obergrenze-untergrenze+1)\*Math.random()+untergrenze).

* Die rnd-Methode der Klasse Math gibt einen Wert zurück, der kleiner als 1, aber größer gleich Null ist.
* Ein import der Klasse Math ist nicht nötig.
* Neu: Variable boolean (wahr, falsch) gewonnen = true -> if (gewonnen) bzw. if(!gewonnen)

## Die Zählschleife (For Schleife)

Um einen Befehlsblock n-mal durchlaufen zu lassen, ist es nicht notwendig, den Befehlsblock n - mal in den Programmcode zu schreiben. Man kann hierfür Schleifen benutzen. Ist n ganzzahlig, so eignet sich besonders die for - Schleife.

|  |  |
| --- | --- |
| **Grundlegende Struktur** | **Beispiel** |
| For (Anweisung1; Bedingung; Anweisung2)  {  Befehlsblock;  } | for(int i=0; i < 5; i++) {  System.out.println(i); |

**Teilbereich 1: int i = 0; //Initialisierung der Zählervariable**

Die Zählervariable ist er Kern der for-Schleife. Sie enthält immer die Zahl des aktuellen Durchlauf und zählt sozusagen, wie oft die Schleife sich schon wiederholt hat. Man kann die Zählervariable auch in Schleifenkörper lesen, um damit bestimmt Operationen durchzuführen.

Im Normalfall nennt an die Zählervariable einfach „i“. Hat man eine weitere verschachtelte for-Schleife in einer for-Schleife, dann nennt man die Zählervariable in dieser in der Regel „j“. Es ist also auch nach Belieben möglich, verschiedene Schleifen ineinander zu verschachteln.

**Teilbereich 2: i < 5; Bedingung zum weiteren Ausführen**

Dieses Element ist ziemlich einfach zu verstehen. Es handelt sich dabei einfach um einen booleschen Ausdruck, welcher bestimmt ob ein weiterer Durchlauf der Schleife stattfinden wird oder nicht. Wenn der Ausdruck „wahr“ ergibt, dann wird die Schleife noch einmal ausgeführt.

**Teilbereich 3: Manipulation der Zählervariable nach jedem Durchlauf**

In diesem Teilbereich inkrementiert oder dekrementiert man die Zählervariable. Diese Operation wird immer am Ende eines Schleifendurchlaufs ausgeführt.

## Break / Continue

Um in Schleifen flexibler navigieren zu können existieren die beiden Schlüsselwörter break und continue.

break:

Das Schlüsselwort break bricht eine Struktur sofort ab. Beispiel:

..

for (int i = 0; i < 10; i++)

{

System.out.println (i);

if ( i== 5)

break;

}

Das Programm gibt die Zahlen 0 bis 5 aus und verlässt dann die Zählschleife.

continue:

Das Schlüsselwort continue sorgt dafür, dass eine Schleife sofort wieder an den Anfang der Schleife springt. Beispiel:

for (int i = 0; i < 10; i++)

{

if ( i> 5)

continue;

System.out.println (i);

}

Das Programm gibt nur die Zahlen von 0 bis 5 aus.

### Übung: Zählschleifen

Analysiere die folgenden Java-Programme und gib an, welche Ergebnisse jeweils in denSystem.out.println()-Anweisungen ausgegeben werden!

1

public class ForSchleife1 {

public static void main (String[] args) { *for(initialisierung,laufbedingung, zählweite)*

for (int i=1; i<=10; i++) { ***neu****: i wird nur in der Schleife deklariert!!*

System.out.print(i + “ “); *1 2 3 4 5 6 7 8 9 10*

}

System.out.println("\n“ + i); ***Fehler****, da i nicht mehr gültig, sonst 11.*

} *Deklaration der Zählvariable in der*

} *üblichen Form*

2

public class ForSchleife2 {

public static void main (String[] args) { *for(initialisierung,laufbedingung, zählweite)*

for (int i=1; i<=3; i++) {

for (int j=1; j<=5; j++) { *1 2 3 4 5 (i=1, j=1,2..)*

System.out.print(i\*j + "\t“); *2 4 6 8 10 (i=2, j=2,4..)*

}   
System.out.println(); *3 6 9 12 15 (i=3, j=3,6..)*

}

*innere Schleife läuft fünfmal*

}

}

3

public class ForSchleife3 {

public static void main (String[] args) { *mehrere Zählvariablen*

for (int i=0, j=10; i<j; i++, j--) { ***es werden beide Variable i und j initalisiert***

System.out.println(“i: “ + i +“ j: “ + j); *i: 0 j: 10*

} *i: 1 j: 9*

} *i: 2 j: 8*

} *i: 3 j: 7*

*i: 4 j: 6*

public class PlusMinus {  *i++ bzw. i+=1, Inkrementoperator erhöht den Wert*

4

*i-- bzw. i-=1, Dekrementoperator vermindert den Wert*

public static void main (String[] args) {

int i = 4711; *kein final, Variable kann überschrieben werden*

System.out.println(i++ + “ “ + (i + 1) + “ “ + i); *Ergebnis: 4711 4713 4712*

System.out.println(i++ + “ “ + ++i + “ “ + i); *Ergebnis: 4712 4714 4714*

System.out.println(i-- + “ “ + --i + “ “ + i); *Ergebnis: 4714 4712 4712*

}

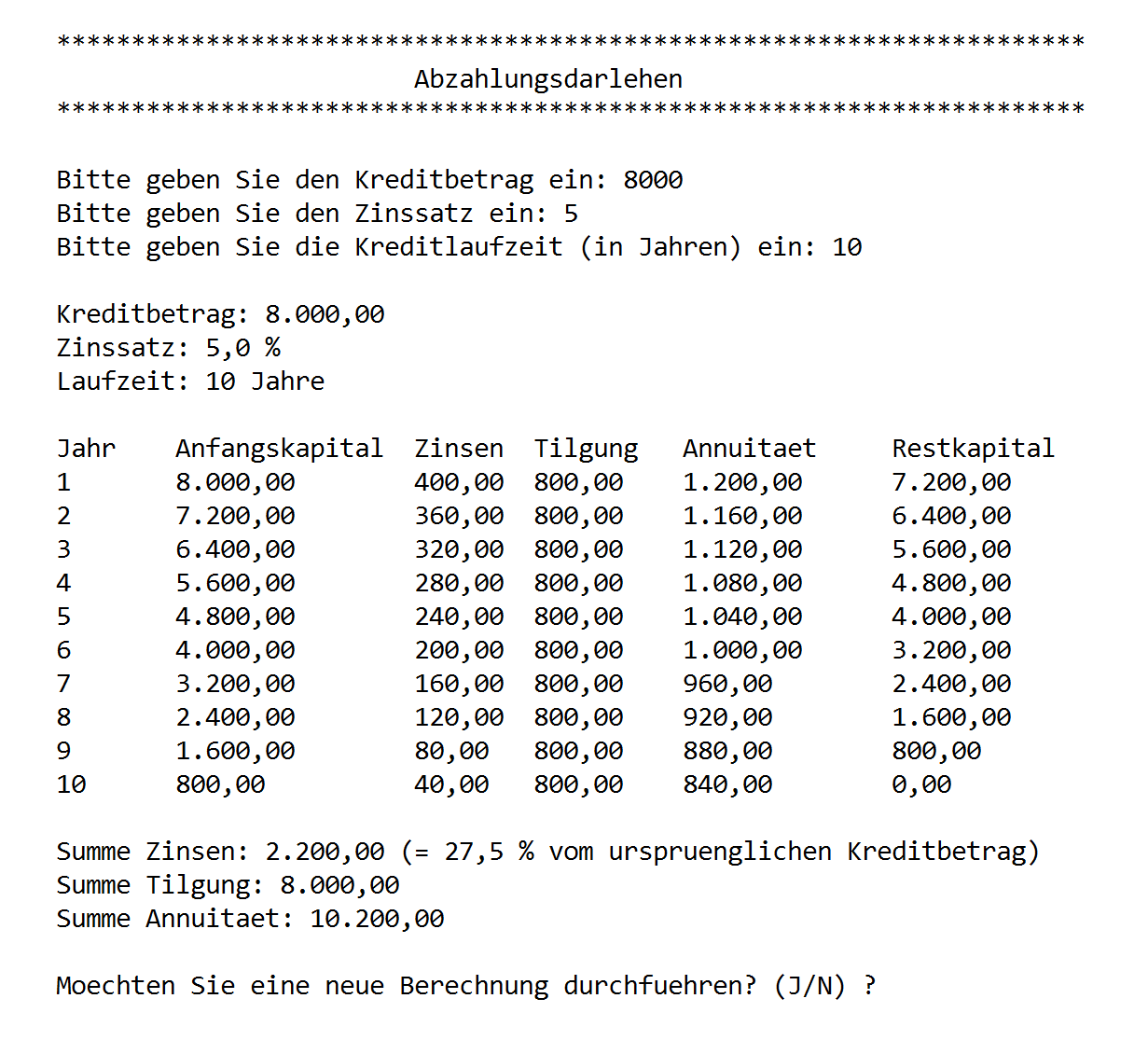
}

*Präfix-Opterator, Operator wird angewendet, bevor ihr Wert in eine Anweisung eingesetzt wird.*

*Postfix-Operator benutzt den Variableninhalt zuerst in einer Anweisung und erhöht ihn danach.*

### Übung: Abzahlungsdarlehen

Entwickle ein Struktogramm für ein Kredittilgungsplan. Eingabefelder sind der Kreditbetrag, die Laufzeit sowie der Zinssatz. Codieren Sie im Anschluss das Programm in Java.



**Hinweise:**

* Nur die Tilgung verringert den Kreditbetrag. Die Tilgung soll als gleichmäßiger Betrag über die Laufzeit verteilt werden.
* Die Zinsen berechnen sich von dem jeweiligen Kreditbetrag.
* Die Annuität entspricht der Summe aus Zinsen und Tilgung und ist der Betrag, der an die jeweilige Bank überwiesen wird.

**Zusatzaufgabe:**

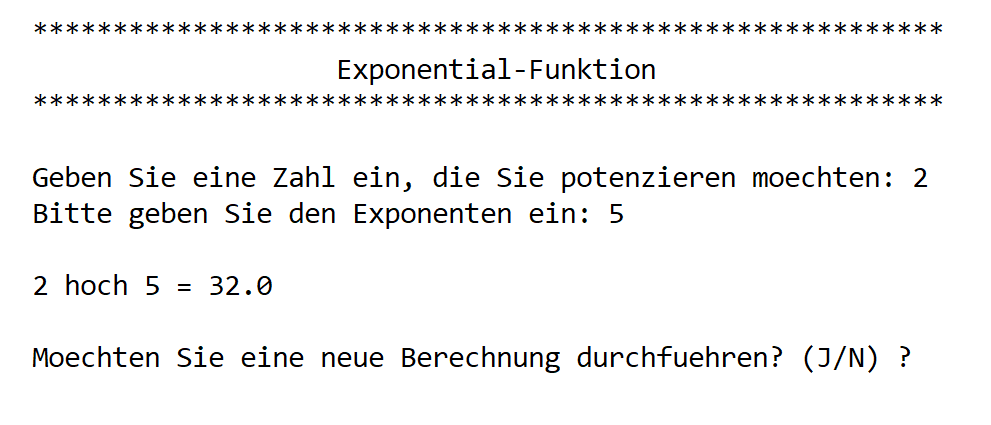
Sorge dafür, dass der Benutzer nur eine Laufzeit von nicht mehr als 10 Jahren eingeben kann. Gibt der Benutzer eine Laufzeit von über 10 Jahren oder weniger als 1 Jahr ein, dann soll eine Fehlermeldung „Die Laufzeit beträgt minimal 1 und maximal 10 Jahre!“ angezeigt und der Benutzer zur erneuten Eingabe der Laufzeit aufgefordert werden.

**Hinweis**:

* Verwende für die maximale Laufzeit eine Konstante.

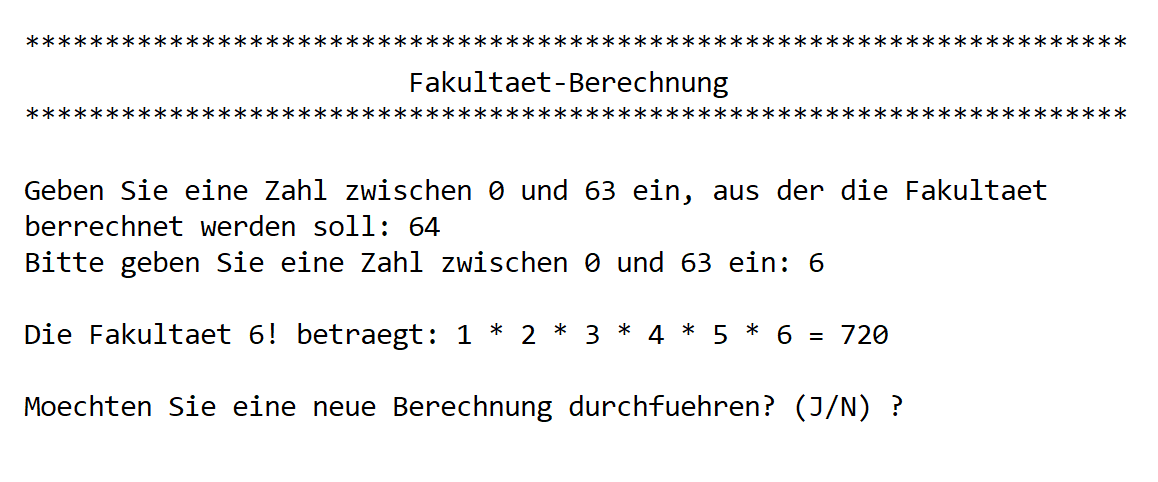
### Übung: Exponential-Funktion

Du wirst beauftragt, ein Struktogramm zur Potenzierung von Zahlen zu entwickeln. Im Anschluss soll das Struktogramm implementiert werden.



### Übung: Fakultät

Entwirf ein Struktogramm zur Berechnung der Fakultät.



* Berücksichtige bei der Programmierung, dass 0! definitorisch dem Wert 1 entspricht.
* Verwende für die Variable ergebnis den Datentyp long, um eine Fakultät bis 63! zu ermöglichen.
* Definiere für die maximale Fakultät (hier: 63) eine Konstante.

# Array

Arrays ermöglichen, genau wie Variablen, die Speicherung von Daten zur Laufzeit. Der Unterschied liegt darin, dass ein Array mehrere Werte gleichzeitig beinhalten kann. Man unterscheidet zwischen

eindimensionalen und mehrdimensionalen Arrays. Die eindimensionalen Arrays kann man sich wie eine Liste vorstellen. Bei mehrdimensionalen Arrays kommt es darauf an, von vielen Dimensionen die Rede ist. Wir beschränken uns hier auf ein- und zweidimensionale Arrays. Ein zweidimensionales Array ist vergleichbar mit einer Tabelle.

|  |  |
| --- | --- |
|  | **Namen** |
| *Zeile 0* | Peter |
| *Zeile 1* | Tom |
| *Zeile 2* | Tim |
| *Zeile 3* | Ali |

|  |  |  |
| --- | --- | --- |
|  | *Spalte 0* | *Spalte 1* |
| *Zeile 0* | **Vorname** | **Nachname** |
| *Zeile 1* | Peter | Müller |
| *Zeile 2* | Tom | Baum |
| *Zeile 3* | Tim | Schmidt |
| *Zeile 0* | Ali | Koch |

1: eindimensional 2: zweidimensional

## Eindimensionales Array

**Dieser eindimensionale Array legt man folgendermaßen an:**

import java.util.\*;

*public class Program {*

*public static void main (String[] args) {*

*String[] namen = new String[4]; // Array wird definiert*

*}*

*}*

Zuerst gibt man den Datentyp an, hier ein String. Hierbei müssen zwei eckige Klammern angefügt werden, damit diese Variable als ein Array erkannt wird. Darauf folgt der Name des Arrays, hier Namen. Mit Hilfe des Schlüsselworts „new“ wird eine Instanz eines Objektes erstellt. Danach müssen wir noch einmal den Datentyp des Arrays angeben, gefolgt von einem weiteren eckigen Klammerpaar. In die Klammer wird aber nun eine Zahl eingetragen, diese legt die Größe es Arrays fest.

Klasse Array aus dem Package java.util stellt mit fill eine Methode bereit, mit der alle Elemente eines Feldes einfach initialisiert werden können.

**Array Werte zuweisen:**

*namen[0] = „Peter“;*

*namen[1] = „Tom“;*

*namen[2] = „Tim“;*

*namen[3] = „Ali“;*

Die erste verfügbare Stelle in einem Array ist über den Index 0 zu erreichen. Deshalb ist **letzte Stelle im Array immer um 1 kleiner als die Größe des Arrays.** Somit hat das letzte Element des Feldes den Index *feld.length – 1* !

Anschließend kann man über den Index in den eckigen Klammern auf die einzelnen Feldelemente zugreifen.

**Array Wert lesen: Ausgabe:**

*System.out.println(namen[1]); Tom*

***Alternative Wertzuweisung:***

Wenn ein Array direkt mit Werten belegt werden soll, so gibt es in Java eine Abkürzung, die ein Array Objekt anlegt und zugleich mit Werten belegt.

*String [] namen = {„Peter”, „Tom”, “Tim”, “Ali”};*

***Sortierung des Array***

Primitive Typen

Für die primitiven Datentypen wird in aufsteigender natürlicher Ordnung sortiert:

int[] beispielInt = {4, 5, 1, 7, 2, 9};

Arrays.sort(beispielInt);

System.out.println(beispielInt[0]));

// Ausgabe: [1]

## 

String

String[] beispielString = {"Gustav", "Anton", "Heinrich"};

Arrays.sort(beispielString);

System.out.println(beispielString[0]));

// Ausgabe: [Anton]

### Übung Temperaturen

Analysiere die folgenden Java-Programme und gib an, welche Ergebnisse jeweils in den System.out.println()-Anweisungen ausgegeben werden!

*Stellen Sie sich vor, Sie müssten die Temperaturen für jeden Tag eines Monats speichern. Mit den Techniken, die Sie bisher kennen gelernt haben, müssten Sie x-Variablen anlegen.*

*Ein* ***Feld*** *ist eine Sammlung von Daten ein und derselben Typs, die man unter einem gemeinsamen Namen ansprechen kann.*

***double [] temperatur*** *bzw. double temperatur[]. 1. Variante ist besser, denn Sie macht deutlich, dass termperatur der Name und double[] der Typ ist (double[]-Array). 2. ist für unverbesserliche C++-Programmierer.*

a

*import java.util.\*; Klasse Array aus dem Package java.util stellt mit fill eine Methode bereit, mit der alle Elemente eines Feldes einfach initialisiert werden können.*

public class ArrayEinfuehrungTemperaturen {

*final int ANZAHL\_TAGE = 31; final* *verhindert, dass der Wert der Konstante geändert wird.*

double[] temperatur; *Arrays sind bei Java auch Objekte, die deklariert …*

*und erzeugt werden müssen.*

*// double[] temperatur = new double[31]; ANZAHL-TAGE: Definition der Feldgröße*

public void start() {

temperatur = new double[31]; *Erzeugung und Initalisierung des Arrays,* ***Trennung ist gut da bei Wdh. Das Objekt neu erzeugt wird und alle Werte wieder 0 sind***

System.out.println(temperatur.length); *da temperatur ein Objekt ist, haben Sie auch Methoden. Length ist die Größe des Arrays, Anzahl der möglichen Elemente werden ausgegeben.*

***Arrays.fill(temperatur,0.0);*** *durch Array.fill werden alle Elemente des im ersten Parameter angegebenen Arrays mit dem zweiten Wert gefüllt.*

temperatur[0] = 25.0; *Initialisierung eines Elementes des Feldes erfolgt über den Index, der in eckigen Klammern steht. Sein niedrigster Wert ist 0*.

temperatur[1] = 33.2; *höchster Index ist immer lenght-1*

temperatur[3] = 18.7;

System.out.println(temperatur[3]); *Ergebnis: 18.71*

System.out.println(temperatur[2]); *:* ***null nicht 0!!!,*** *Ergebnismit fill 0.0 \**

System.out.println(temperatur[31]); *ExceptionOutOfBounds*

*\* Wäre temperatur ein String muss dieser initialisiert werden, sonst* ***null, Zahlen in Anführungszeichen setzen und dies zeigen***

}

public static void main (String[] args) {

ArrayEinfuehrungTemperaturen aet = new ArrayEinfuehrungTemperaturen();

aet.start();

}

}

b

*import java.util.\*;*

public class TestArray {  *i*

int[] beispiel = {23, 9, 7,43, 93, 9,12};

public void start() { *Anweisung erzeugt ein Objekt mit 7 Elemente, ohne dass man den Befehl Operator new explizit angeben muss. Wenn ein Array direkt mit Werten belegt werden soll, so gibt es in Java eine Abkürzung, die ein Array-Objekt anlegt und zugleich mit Werten belegt.*

*String[] beispiel = {„A“,“B“,“C“};*

System.out.println(beispiel[0]); *23*

System.out.println(beispiel[6]); *12*

*Array.sort(beispiel); Methode sort der klasse java.util.Arrays ordnet die Zahlen in steigender Reihenfolge im Feld an.*

*System.out.println(beispiel[0]); 7*

*System.out.println(beispiel[6]); 92*

}

public static void main (String[] args) {

TestArray ta = new TestArray();

ta.start();

}

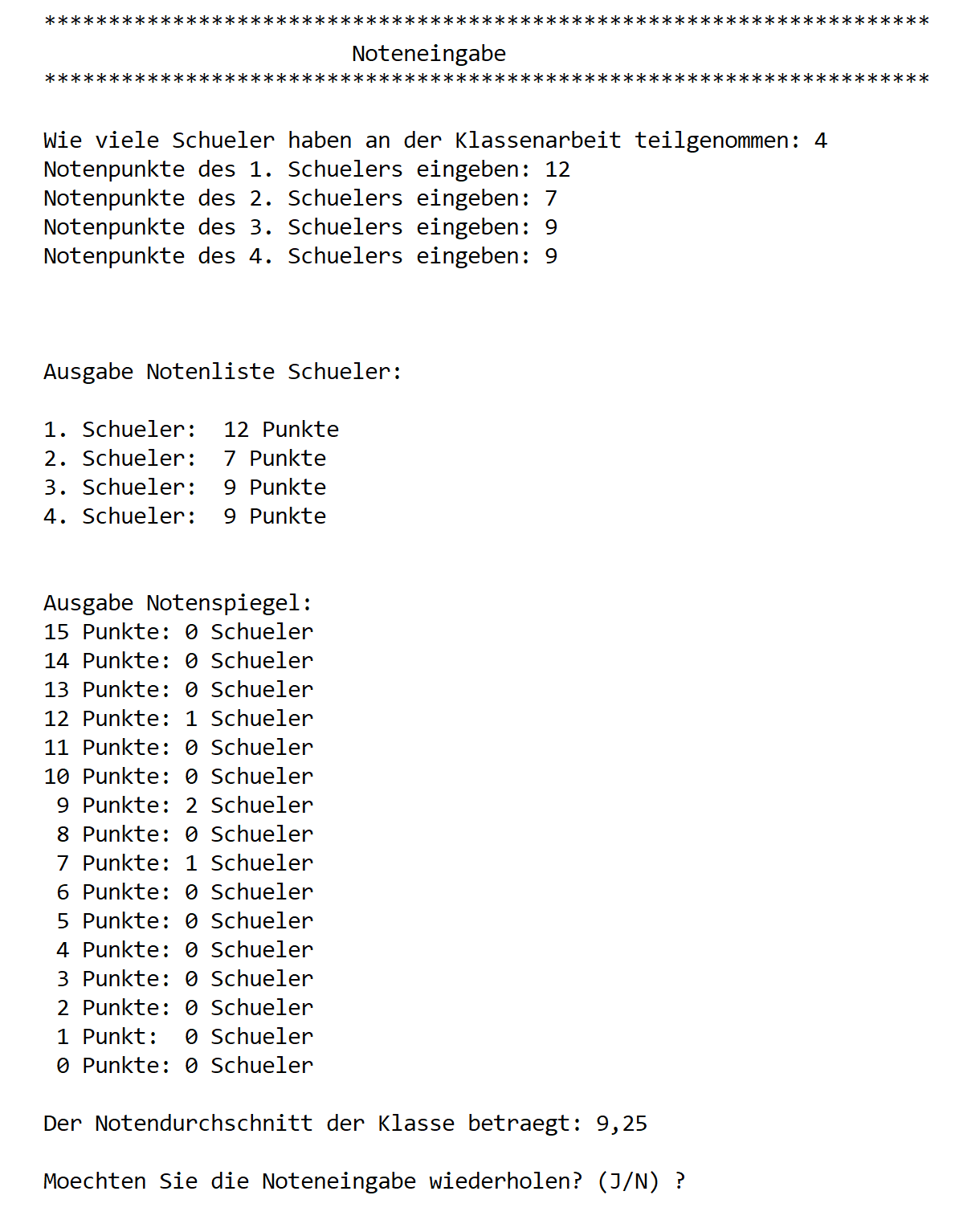
}

### Übung Schulnoten mit eindimensionalen Array

Du wirst beauftragt, ein Programm zur Noteneingabe zu programmieren.

**Anforderungen**

* Der Benutzer gibt zunächst die Anzahl der Schüler sowie die einzelnen Notenpunkte ein.
* Danach soll das Programm den Notenspiegel und den Notendurchschnitt ermitteln.
* Abschließend werden die Schülerpunkte sowie der Notenspiegel und der Notendurchschnitt auf dem Bildschirm ausgegeben.
* Verwende die Methoden eingabe(), notendurchschnitt(), notenspiegel(), und ausgabe().

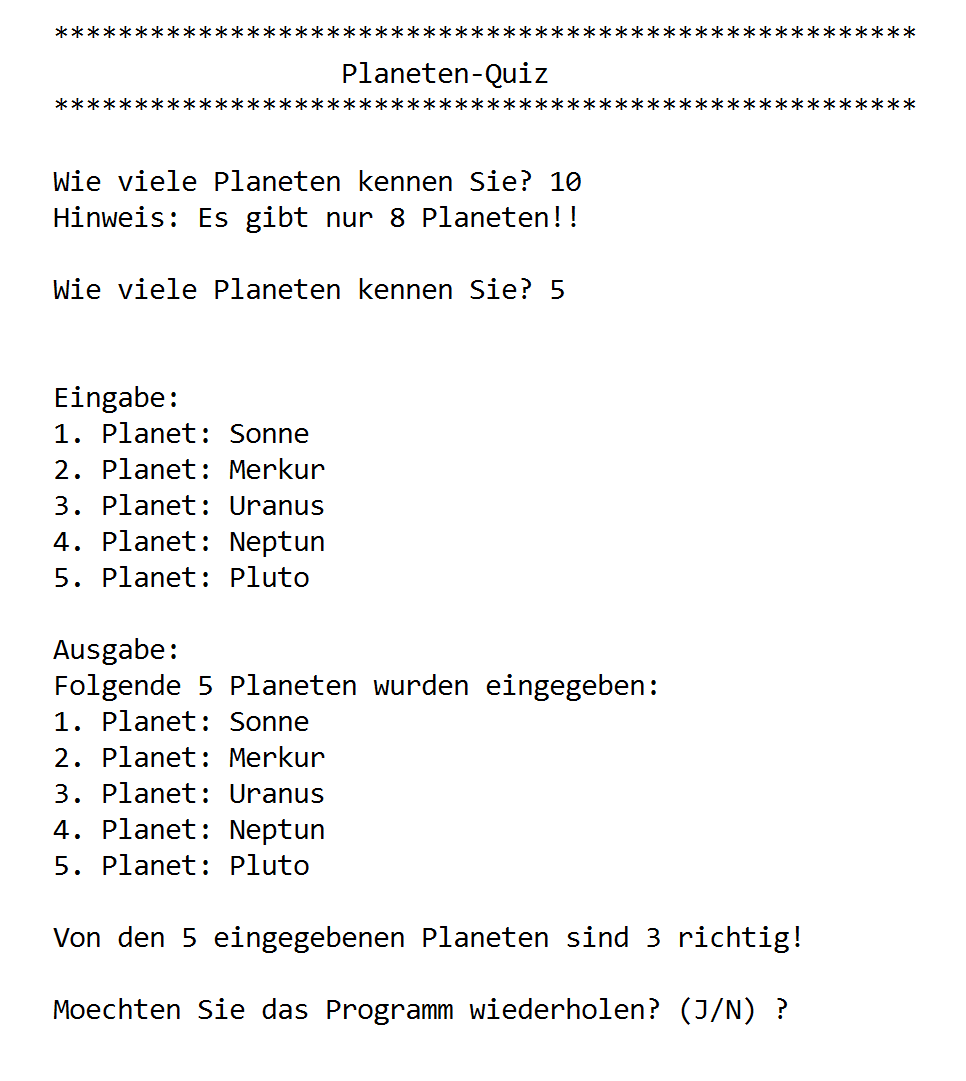


### Übung Planeten-Quiz

Du wirst beauftragt, ein Planeten-Quiz-Programm in Java zu schreiben.

**Anforderungen**:

* Der Benutzer soll zu Beginn des Programms gefragt werden, wie viele Planeten er kennt bzw. eingeben möchte.
* Nach der Eingabe sollen im Anschluss alle Planetennamen inkl. einer Auswertung ausgegeben werden.



**Anforderungen**

* Die richtigen Planeten sollen in einer Array-Konstanten deklariert werden. Beispiel: String[] planetenLoesungen = {„Merkur“, „Uranus“, …..};
* Es sollen maximal drei Instanzvariablen im Kopf der Klasse deklariert werden. Alle anderen Variablen sind in den Methoden zu deklarieren.
* Verwende für Eingabe, Verarbeitung und Ausgabe jeweils eine Methode.

**Zusatzaufgabe**

* Verändere die eingabe()-Methode so, dass der Benutzer keine doppelten Eingaben tätigen darf, d.h. er darf den Planeten Erde nicht mehrmals eingeben.
* Gibt der Benutzer einen Planeten doppelt ein, soll der Hinweis „Dieser Planet wurde schon eingegeben!“ erscheinen.

### Übung Theoretische Aufgabe Min/Max

Implementiere ein Programm zur Ermittlung der höchsten und kleinsten Zahl aus dem Array.

public class MinMax{

int[] zahlen = { 5,6, 8, 1, - 10, 8, 7, 6};

int max;

int min;

public void ermitteleMax(){

*max = zahlen[0];*

*for(int i =0; i < zahlen.length, i++){*

*if(zahlen[i] > max){*

*max = zahlen[i];*

*} }*

}

public void ermitteleMin(){

*min = zahlen[0];*

*for(int i =0; i < zahlen.length, i++){*

*if(zahlen[i] < min){*

*min = zahlen[i];*

*}*

*}*

}

### Übung Auktion

Implementiere ein Programm zur Ermittlung des höchsten Angebots einer Versteigerung in Java.

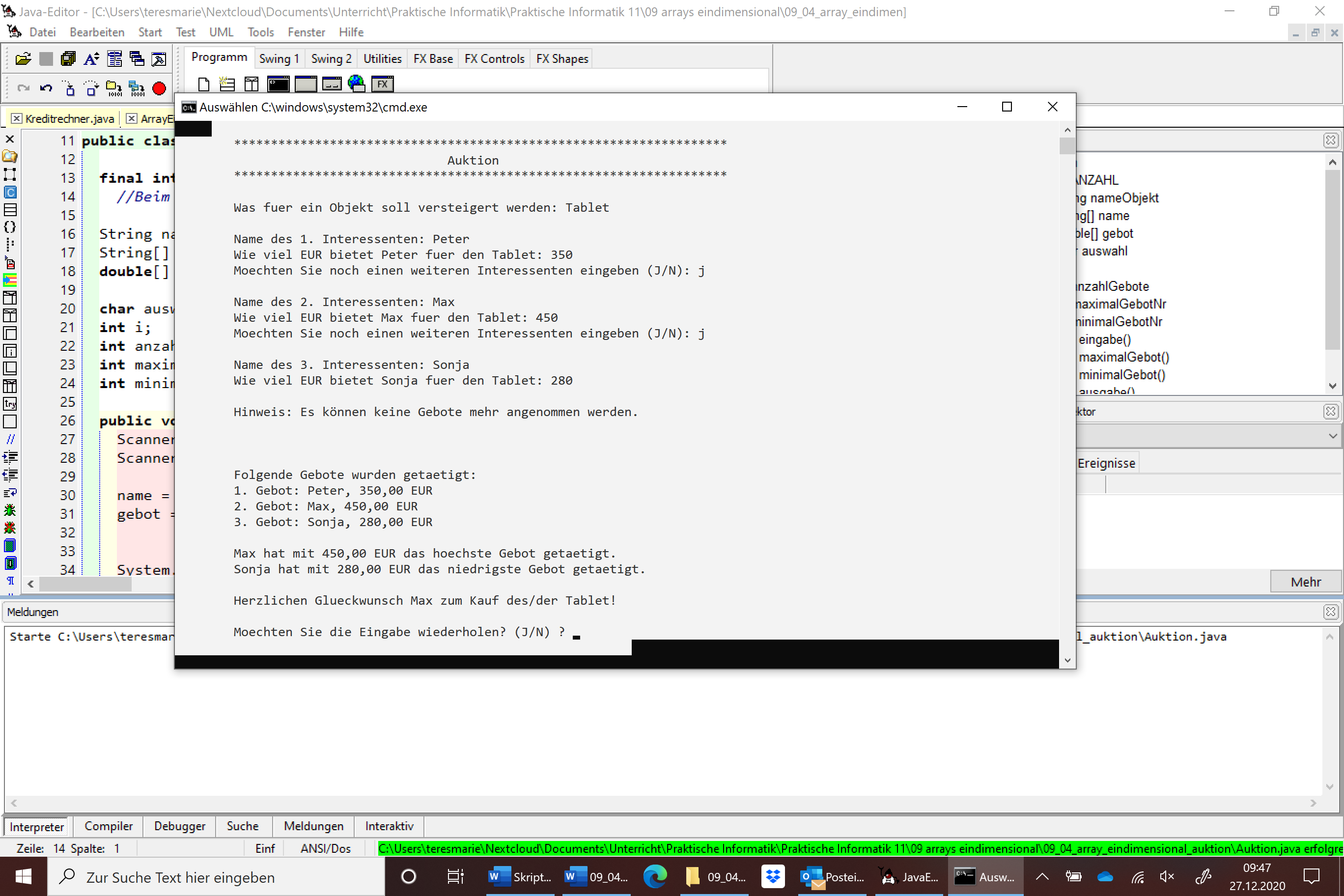
**Anforderungen**:

Zu Beginn des Programms soll der Anwender nach dem Objekt der Versteigerung gefragt werden.

Im Anschluss soll der Anwender den Namen und die Höhe des Angebots der Interessenten eingeben.

Nach jeder Eingabe soll der Anwender gefragt werden, ob er noch einen weiteren Interessenten eingeben möchte.

Nach der Ausgabe sollen die Namen und Angebote aller Interessenten sowie das höchste und niedrigste Angebot ausgegeben werden.



**Zusatzanforderung**:

Sorge dafür, dass bei Erreichen der Arraygrenze der Bieter statt der Frage „Möchten Sie noch einen weiteren Interessenten eingeben?“ der Hinweis „Es können keine Gebote mehr angenommen werden!“ erscheint.

### Übung Zufallszahlen

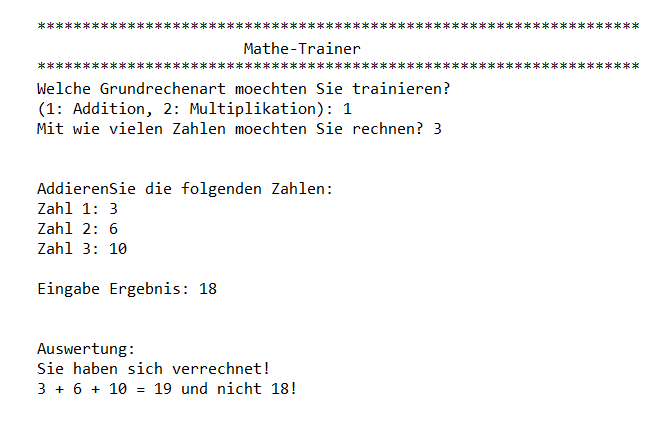
Du wirst beauftragt, ein Mathe-Trainings-Programm zu implementieren.

**Anforderungen**:

Zu Beginn des Programms soll der Benutzer auswählen, welche Grundrechenart er trainieren und mit vielen Zahlen er rechnen möchte.

Im Anschluss sollen ihm zufällig ausgewählte Zahlen angezeigt werden, die er addieren oder multiplizieren soll. Das Ergebnis ist dann von dem Benutzer einzugeben.

Am Ende des Programms soll dem Benutzer eine Erfolgsmeldung angezeigt werden: Hat er richtig gerechnet, soll ihm zum richtigen Ergebnis gratuliert werden, hat er falsch gerechnet, soll ihm das richtige Ergebnis mitgeteilt werden.



Programmiere das Programm mit folgenden Methoden:

public void eingabe()

public void ziehungZufallsZahlenBerechneLoesung()

public void auswertung()

### Übung Theoretische Aufgaben: eindimensionales Array

**Aufgabe 1:**

Die Funktion der Methode **verschiebeZahlenNachLinks()** besteht darin, die Werte der Array-Variablen um je ein Element nach links zu verschieben. Dazu soll ein und derselbe Array verwendet werden! Das erste Element der Liste soll dabei auf die letzte Position des Arrays verschoben werden.

**Beispiel** (für ein Array zahlen[] mit 4 Werten):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Index: | (0) | (1) | (2) | (3) |
| Ursprünglicher Variableninhalt: | 5 | 26 | 13 | 11 |
| Inhalt nach Verschiebung: | 26 | 13 | 11 | 5 |

Die Methode soll so entworfen werden, dass sie auch für kürzere bzw. längere Arrays gültig ist.

**Aufgabe 2:**

Gegeben sind die folgenden beiden Arrays:

int[] zahlen = new int[10];

int[] zahlen2 = new int[10];

Schreibe eine Methode **uebertrageZahlenUmgekehrteReihenfolge()**, welche die Zahlen im Array zahlen[] in das Array zahlen2[] überträgt. Beachte dabei, dass die Inhalte des Array zahlen[] in umgekehrter Reihenfolge übertragen werden.

**Aufgabe 3:**

Die Funktion der Methode **verschiebeZahlenNachRechts()** besteht darin, die Werte der Array-Variablen um je ein Element nach rechts zu verschieben. Das letzte Element der Liste soll dabei auf die erste Position des Arrays verschoben werden. Bei der Verschiebung ist nur das Array zahlen[] und kein weiteres Array zu verwenden.

**Beispiel** (für ein Array zahlen[] mit 4 Werten):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Index: | (0) | (1) | (2) | (3) |
| Ursprünglicher Variableninhalt: | 5 | 26 | 13 | 11 |
| Inhalt nach Verschiebung: | 11 | 5 | 26 | 13 |

Die Methode soll so entworfen werden, dass sie auch für kürzere bzw. längere Arrays gültig ist.

**Aufgabe 4:**

Die Funktion der Methode **verschiebeZahlenZweiPositionenNachRechts** () besteht darin, jeden zweiten Wert der Array-Variablen (beginnend bei dem Index 0) um je zwei Elemente nach rechts zu verschieben. Das letzte Element der Liste soll dabei auf die erste Position des Arrays verschoben werden. Bei der Verschiebung ist nur das Arrray zahlen[] und kein weiteres Array zu verwenden!

**Beispiel** (für ein Array zahlen[] mit 7 Werten):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Index: | (0) | (1) | (2) | (3) | (4) | (5) | (6) |
| Ursprünglicher Variableninhalt: | **5** | 26 | **13** | 11 | **8** | 13 | **9** |
| Inhalt nach Verschiebung: | **9** | 26 | **5** | 11 | **13** | 13 | **8** |

Die Methode soll so entworfen werden, dass sie auch für kürzere bzw. längere Arrays gültig ist. Arrays mit einer geraden Länge müssen nicht berücksichtigt werden.

**Aufgabe 5:**

**Aufgabe 3**

Implementiere die Methode **ermitteleAnzahlPrimzahlen()**. Die Methode soll die Anzahl der Primzahlen in dem Array zahlen[] ermitteln.

**Hinweise:**

Eine Primzahl ist eine natürliche Zahl, die größer als 1 und ausschließlich durch sich selbst und durch 1 teilbar ist.

Im Array int[] zahlen = {1, 2, 3, 4 , 5, 6, 7, 8, 9} existieren 4 Primzahlen.

Der Modulo hilft euch hier.

*Sortieren in umgekehrter Reihenfolge: mit sort() sortieren, mit reverse() umzudrehen.*

### Übung: Interner Sortieralgorithmen

**BubbleSort: Sortieren durch Vertauschen**

Eines der bekanntesten *, wenn auch kein besonders effizientes* Sortierverfahren ist BubbleSort. Der Name ist aus der Vorstellung abgeleitet, dass sich *bei einer vertikalen Anordnung der Elemente der Folge* verschieden große, aufsteigende Blasen („Bubbles“) wie in einer Flüssigkeit von allein sortieren, da die größeren Blasen die kleineren überholen.

**Quellcode:**

public void sortieren () {

for(int i=0; i<sortieren.length - 1; i++) {

for(int j=0; j<sortieren.length-1-i; j++) {

if(sortieren[j]>sortieren[j+1]) {

int temp = sortieren[j];

sortieren[j]=sortieren[j+1];

sortieren[j+1]=temp;

}

}

}

}

**Aufgabe:**

Deine Aufgabe ist es zu verstehen, wann was beim BubbleSort passiert.

Gegeben ist folgendes Array {5,3,7,2,1}. Notiere für jeden Durchlauf, wie das Array danach aussieht.

**Sortiervorgang:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| i= | j= |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

### Übung Lotto

Du wirst beauftragt ein Lotto-Simulations-Programm zu schreiben.

**Anforderungen:**

Zu Beginn des Spiels soll der Benutzer die Möglichkeit haben sechs Zahlen zwischen 1 und 49 einzugeben.

Danach werden über ein Zufallszahlenmechanismus sechs Gewinnzahlen gezogen und mit den eingegebenen Zahlen verglichen.

Das Programm soll die gezogenen Zahlen und die Anzahl der Treffer auf dem Bildschirm ausgeben.

Sowohl bei der Eingabe wie auch bei der Ziehung der Zahlen darf es keine Wiederholung von Werten geben. Sollte der Anwender eine Zahl doppelt eingeben wollen, so soll das Programm die Eingabe verweigern und mit einem Fehlerhinweis reagieren. Der Anwender muss dann eine andere Zahl auswählen.

Bei der zufallsgesteuerten Ziehung entfällt die Fehlermeldung, jedoch muss der Ziehungsprozess so oft wiederholt werden, bis sechs verschiedene Zahlen gefunden wurden.

Die getippten und gezogenen Zahlen sollen in aufsteigender Reihenfolge auf dem Bildschirm erscheinen.

Verwende für jeden Vorgang Eingabe, Sortierung, Ziehung der Zufallszahlen, Ermittlung der richtigen Zahlen und Ausgabe eine eigene Methode.

Erstelle für die Methode Eingabe, Sortierung, Ermittlung der richtigen Zahlen auch ein Struktogramm.

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

Tipp: Wenn ihr Probleme mit der Sortierung habt, dann öffnet die Übung BubbleSort, da findet ihr die Lösung. Aber natürlich nur im Notfall. 😉

**Zusatzaufgabe Gewinnausschüttung:**

Erweitere das Programm um eine Gewinnausschüttung:

Zu Beginn des Programms soll der Benutzer gefragt werden, wie viel er auf sein virtuelles Geldkonto einzahlen möchte.

Bei jeder Ziehung ist das Geldkonto des Benutzers mit 1,50 EUR zu belasten.

Für das Lottospiel gilt folgende Gewinnausschüttung:

|  |  |
| --- | --- |
| **Anzahl der Treffer** | **Gewinnausschüttung** |
| 1 | 0,00 EUR |
| 2 | 3,00 EUR |
| 3 | 10,00 EUR |
| 4 | 500,00 EUR |
| 5 | 10.000,00 EUR |
| 6 | 1.000.000,00 EUR |

Der gewonnene Geldbetrag soll dem Geldkonto gutgeschrieben werden.

### Übung: Wortspielquiz

Sie sollen ein Wortspiel-Quiz in Java programmieren.

**Anforderungen**:

Das Wortspiel soll folgende Fragen enthalten, die mit einem Zufallsgenerator ausgewählt werden.

|  |  |
| --- | --- |
| **Frage** | **Lösung** |
| Sie wird kürzer, je länger man zieht | Zigarre |
| Dieser Ring ist nicht rund | Boxring |
| Dieser Baum hat keine Blätter | Purzelbaum |
| Dieses Tal will jeder haben | Kapital |
| Ein Zahn, der nicht beißt | Löwenzahn |
| Ein Stuhl ohne Beine | Dachstuhl |

Nach jeder Lösungseingabe soll eine Erfolgsmeldung ausgegeben werden.

Der Benutzer soll das Programm wiederholen können.

Am Ende des Programms ist dem Spieler anzuzeigen, wie viele Fragen er richtig gelöst hat.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

**Hinweise:**

Verwende für die Fragen und die Lösungen jeweils eine Array-Variable.

Verwende für die Auswahl der Fragen, die Eingabe und die Auswertung eigenständige Methoden (wie z.B. auswahlFragen(), eingabeAntwort(), auswertung(), ausgabeGesamtauswertung())

**Zusatzaufgabe**:

Programmiere das Programm so um, dass jede Frage nur einmal angezeigt wird.

Das Programm ist zu beenden, wenn der Benutzer nicht noch einmal spielen möchte bzw. wenn jede Frage einmal angezeigt wurde.

**Hinweis:**

Speicher die bereits gezogenen Indizes der Fragen in einer Array-Variablen gezogeneZahlen ab und kontrolliere bei jeder Ziehung, ob der gezogene Index in der Array-Variablen schon vorhanden ist. Falls ja, muss noch einmal gezogen werden.

## Mathematische Funktionen

Analysiere das folgende Java-Programm und gebe an, welche Ergebnisse jeweils in den System.out.println()-Anweisungen ausgegeben werden!

public class MatheMatischeFunktionen {

public static void main(String[] args) {

int a,b;

double c,d,e,f,g;

a=3;

b=2;

c=Math.pow(a,b);

System.out.println("\t1: " + c); *9*

d = c \* Math.PI;

System.out.println("\t2: " + d); *28.2743338823*

e=3.889;

System.out.println("\t3: " + Math.round(e)); *4*

System.out.println("\t4: " + Math.rint(e\*10)/10); *3,9*

System.out.println("\t5: " + Math.rint(e\*100)/100); *3,89*

f=3.111;

System.out.println("\t6: " + Math.ceil(f\*10)/10); *3,2*

System.out.println("\t7: " + Math.ceil(f\*100)/100); *3.12*

System.out.println("\t8: " + Math.floor(e\*10)/10); *3,8*

System.out.println("\t9: " + Math.floor(e\*100)/100); *3,88*

System.out.println("\t10: " + Math.min(a, b)); *2*

g = b - a;

System.out.println("\t11: " + Math.abs(g)); *1*

System.out.println("\t12: " + Math.max(a, b)); *3*

System.out.println("\t13: " + Math.sqrt(2)); *1,4132135623*

}

}

**Hinweise:**

* Die Funktion round() rundet auf die nächste Ganzzahl vom Typ int oder long auf oder ab.
* Die rint()-Funktion lässt sich auch bei Dezimalzahlen vom Typ double einsetzen und wird für das kaufmännische Runden mit Nachkommastellen verwendet.
* Die Methode ceil() und floor() dienen zum Aufrunden bzw. Abrunden und liefern die nächst höhere bzw. niedrigere Ganzzahl, wenn die Zahl nicht schon eine ganze Zahl ist.
* Die Methode pow() liefert den Wert der Potenz xy.
* Die Methode sqrt() steht für square root (=Quadratwurzel) und liefert die Quadratwurzel   
  von x.

## Zweidimensionales Array

Erzeugung eines zweidimensionalen Arrays:

Import java.util\*;

*public class Program {*

*public static void main (String[] args) {*

*String [] [] namen = new String[4][2]; // Array wird definiert*

*}*

*}*

Der Unterschied zum eindimensionalen Array liegt darin, dass wir zwei aufeinanderfolgenden eckige Klammerpaare benötigen. Auf der rechten Seite des Zuweisungsoperator geben wir dann wieder konkrete Ganzzahlen zwischen den eckigen Klammern ein, um die Größe des Arrays festzulegen. Die erste Zahl steht für die Anzahl der Zeilen, die zweite für die Anzahl der Spalten.

|  |  |
| --- | --- |
| Peter | Müller |
| Tom | Baum |
|  |  |
|  |  |

**Array Werte zuweisen:**

*namen[0][0] = „Peter“;*

*namen[0][1] = „Müller“;*

*namen[1][0] = „Tom“;*

*namen[1][1] = „Baum“;*

**Array Wert lesen: Ausgabe:**

*System.out.println(namen[0][0] + “ ‘‘ + namen[0][1]); Peter Müller*

**Man kann auch hier die Werte direkt mitgeben:**

*String [][] personen = new String [][] {*

*{ Peter, Müller, Lehrer},*

*{Tom, Baum, Schüler}*

*};*

**Array vollständig auf die Konsole ausgeben:**

for( int zeile = 0; zeile < personen.length; zeile ++){

for (int spalte = 0; spalte <personen[0].length; spalte ++ ){

System.out.print(personen[zeile][spalte]);

}

System.out.println();}

### Übung Filialen

Ein Unternehmen betreibt zwei Filialen. Zur Darstellung der Geschäftsentwicklung wirst du von der Unternehmensleitung beauftragt, eine tabellarische Übersicht zu erstellen, aus der sich die Umsätze pro Filiale und Quartal, die Gesamtumsätze jeder Filiale, die Gesamtumsätze eines jeden Quartals und schließlich der Gesamtumsatz des Unternehmens ablesen lassen.

**Anforderungen**:

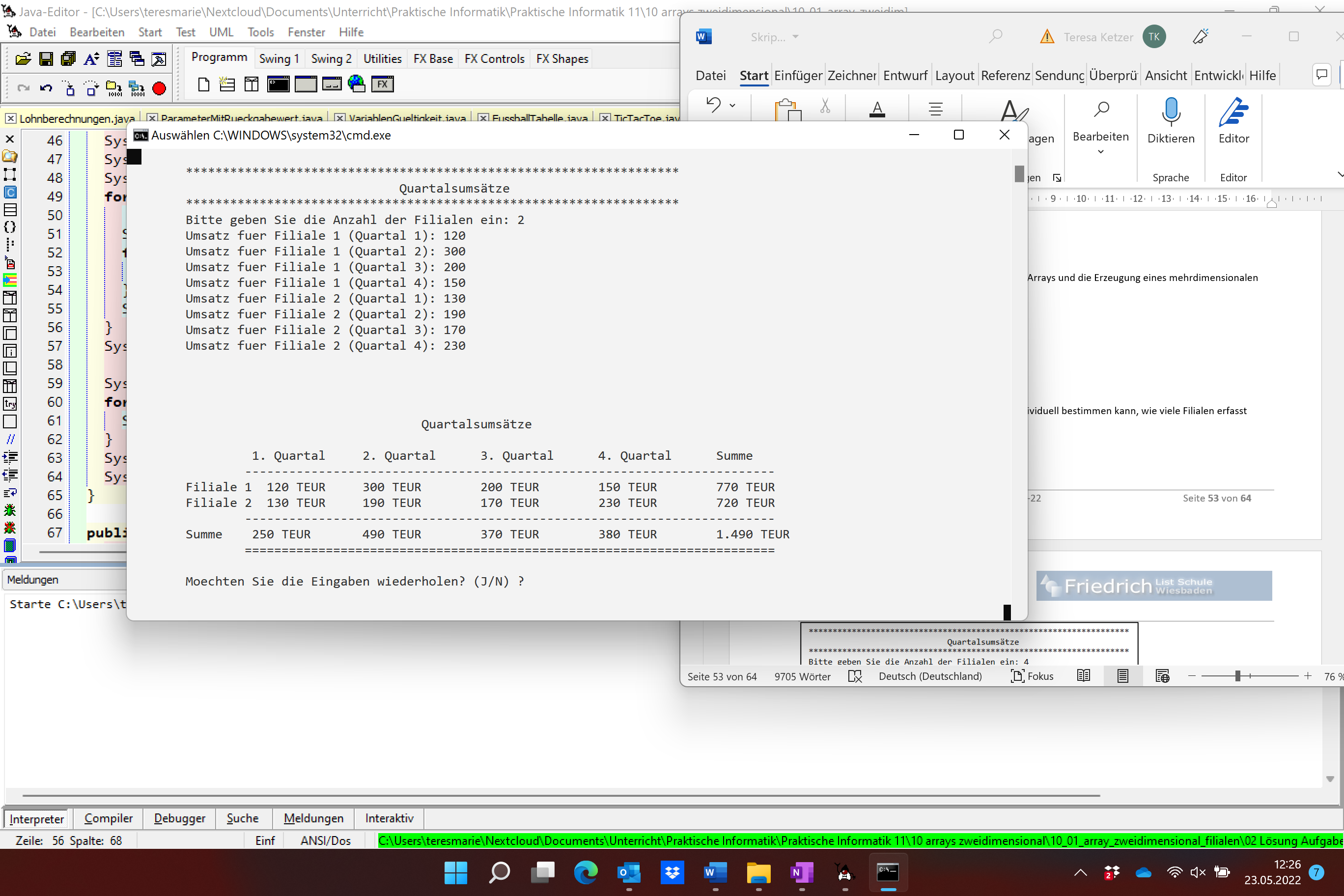
Erstelle eine Java-Anwendung, die nach Eingabe der 8 Quartalswerte die untenstehende Darstellung auf dem Bildschirm ausgibt.

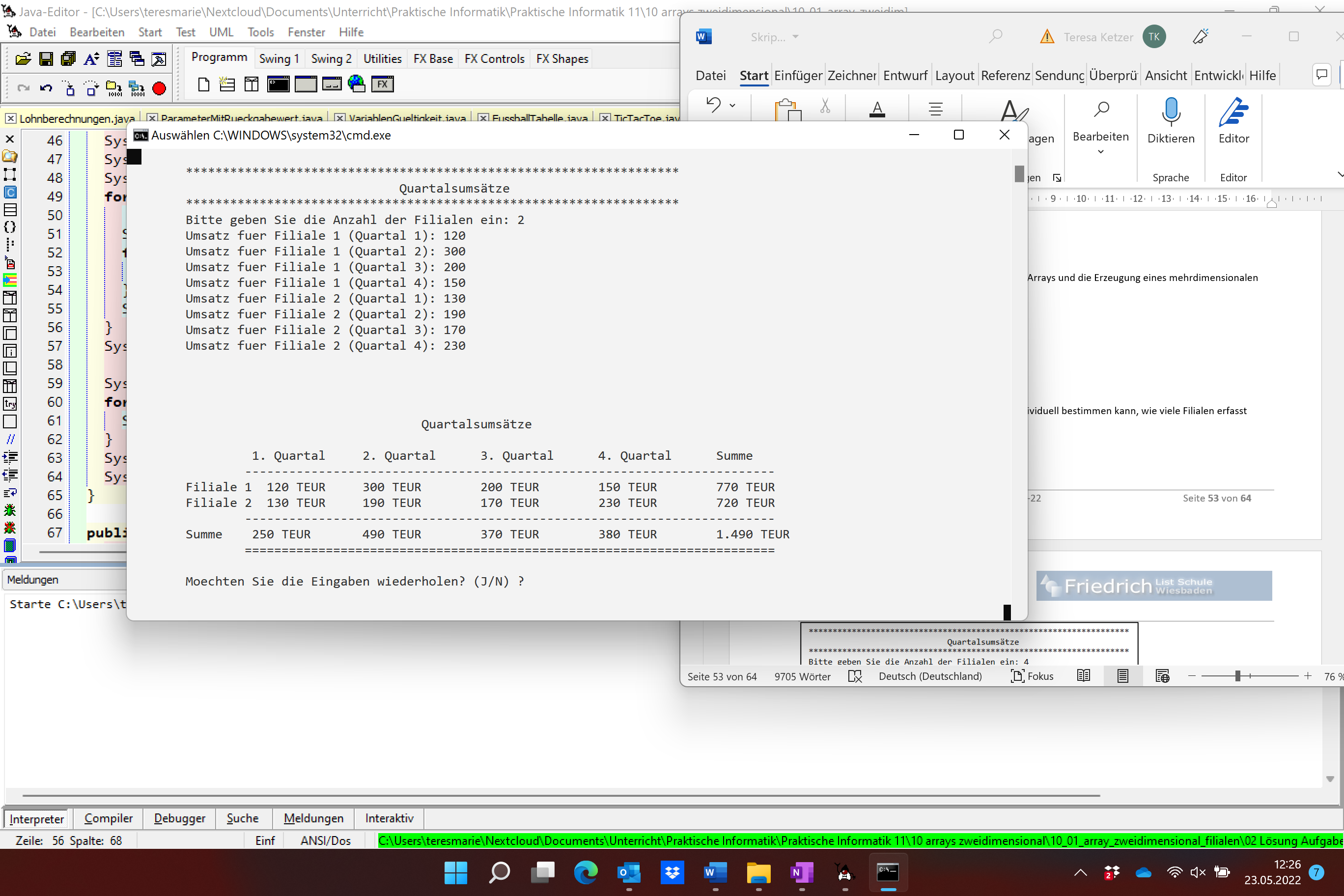
Die Verwaltung der Eingabe- und Ausgabedaten soll über mehrdimensionale Array-Variablen umgesetzt werden.

Um nicht unnötigen Arbeitsspeicherplatz zu verschwenden, belege in den Arrays auch die 0-Positionen.

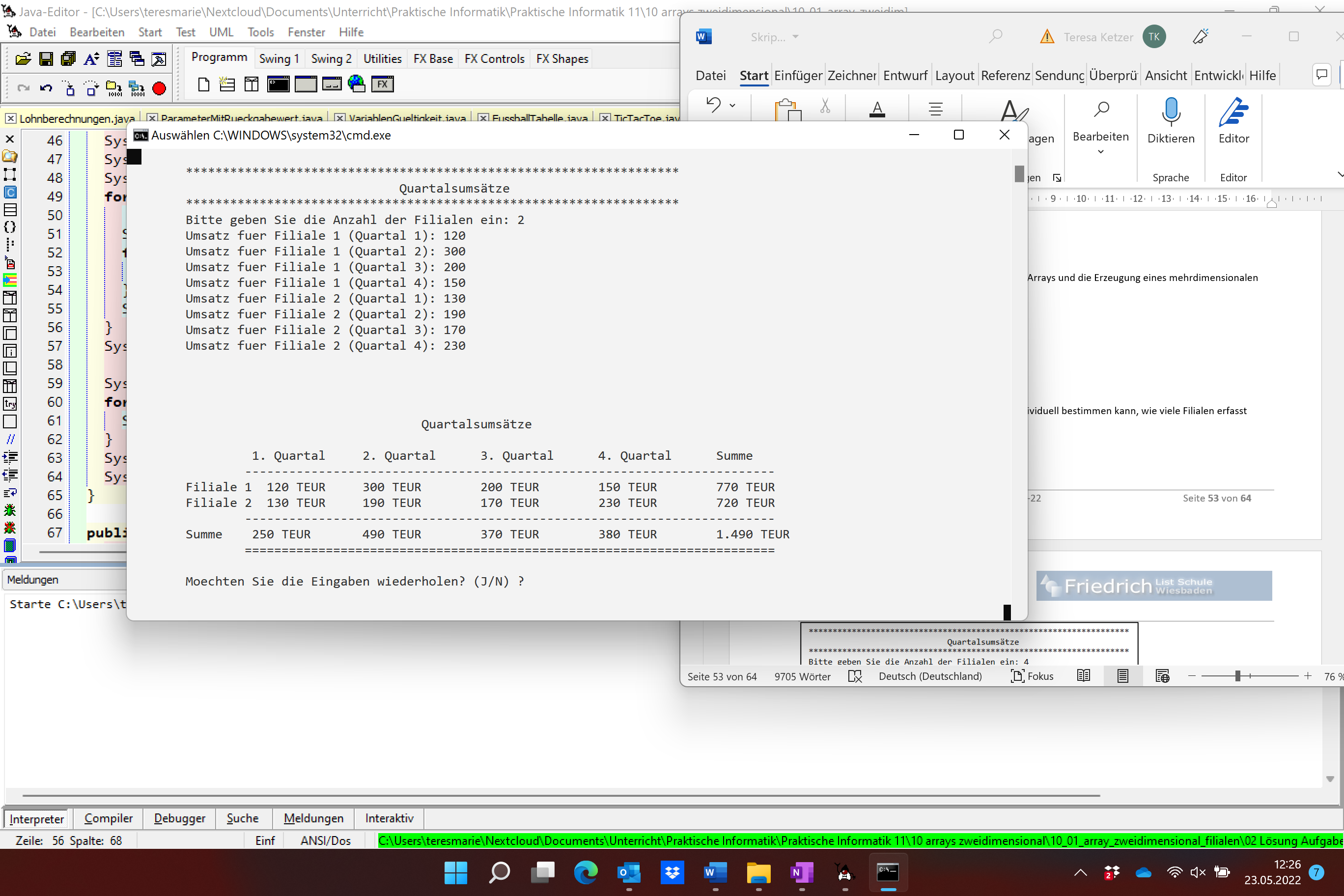
Der Benutzer soll am Ende entscheiden, ob er das Programm wiederholen möchte.

**Eingabe:**





**Ausgabe:**



**Hinweise:**

Syntax für die Deklaration des mehrdimensionalen Arrays und die Erzeugung eines mehrdimensionalen Array-Objekts:   
double[][] umsatz;   
umsatz= new double[2][4];

oder

double[][] umsatz = new double [2][4];

**Zusatzaufgabe:**

Ändere das Programm so ab, dass der Benutzer individuell bestimmen kann, wie viele Filialen erfasst werden sollen.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

### Übung Schulnoten

Mittels eines Java-Programms sollen die Zeugnisnoten für einzelne Schüler ermittelt werden. Die Schüler haben drei Unterrichtsfächer (z.B. Deutsch, Englisch, Geschichte). In jedem der Fächer werden zwei Klausuren geschrieben und es wird am Ende des Halbjahres eine mündliche Note vergeben. Aus den beiden Arbeitsnoten wird die schriftliche Gesamtnote als Mittelwert berechnet. Die Zeugnisnote des jeweiligen Faches wird wiederum als Mittelwert aus der schriftlichen Gesamtnote und der mündlichen Note ermittelt.

**Anforderungen**:

Erstelle eine Java-Anwendung, die nach Eingabe der Schulnoten die untenstehende Darstellung auf dem Bildschirm ausgibt.

Die einzelnen Fächer und Notentypen sind in zwei eindimensionalen Array-Variablen im Programmkopf zu definieren.

Die Verwaltung der Eingabe- und Ausgabedaten soll über mehrdimensionale Array-Variable umgesetzt werden.

Um nicht unnötigen Arbeitsspeicherplatz zu verschwenden, belege in den Arrays auch die 0-Positionen.

Stelle sicher, dass die schriftliche Note und die Zeugnisnote entsprechend gerundet werden.

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

**Hinweise:**

Syntax für die Deklaration und Festlegung der Fächer und Klausurtypen im Programmkopf:   
String faecher[]={“Deutsch“,“Englisch“,“Geschichte“}; bzw.  
String notenTypen[] = {“1. Klausur“,“2. Klausur“, “schriftliche Note“, “mündliche Note“, “Zeugnisnote“};

Um auf die richtigen Ergebnisse zu kommen, musst du die schriftliche Note mit der Math.round()-Methode runden.

**Zusatzaufgabe:**

Stelle sicher, dass der Benutzer nur gültige Notenpunkte eingibt. *Frage: Was kann der Dau alles eingeben und wie kann dies behandelt werden.*

### Übung TheorieAufgaben zum zweidimensionalen Array

1. Gegeben ist folgendes zweidimensionales Array zahlen[][] = new int [5][6]:

**Index j**

0 1 2 3 4 5

0 (1 2 3 4 5 6)

1 (1 2 3 4 5 6)

2 (1 2 3 4 5 6)

**Index i** 3 (1 2 3 4 5 6)

4 (1 2 3 4 5 6)

Implementiere eine Methode **public void berechneSummen()**, welche die Summe der einzelnen Zeilen (Variable: summeZeilen[]) und Spalten (Variable: summeSpalten[]) sowie die Gesamtsumme (Variable: summeGesamt) der Arrayvariablen zahlen[][] berechnet.

Die Methode ist so zu programmieren, dass sie bei einer Wiederholung des Programms auch einwandfrei funktioniert.

*public void berechneSummen(){*

*summeZeilen[] = new int[zahlen.length]*

*summeSpalte = new int[zahlen[0].length];*

*summeGesamt =0*

*for (int i=0 ;i < zahlen.length; i++ ) {*

*for (int j = 0 ;j < zahlen[0].length;j++ ) {*

*summeZeilen[i] = summeZeilen[i] + zahlen[i][j];*

*summeSpalten[j] = summeSpalten[j] + zahlen[i][j];*

*}*

*summeGesamt = summeGesamt + summeZeilen[i];*

*}*

*}*

|  |  |
| --- | --- |
|  |  |

1. Gegeben ist folgendes zweidimensionales Array zahlen[][]:

**Index j**

0 1 2 3 4 5

0 (**1** 2 3 4 5 **6**)

1 (1 **2** 3 4 **5** 6)

2 (1 2 **3** **4** 5 6)

**Index i** 3 (1 2 **3** **4** 5 6)

4 (1 **2** 3 4 **5** 6)

5 (**1** 2 3 4 5 **6**)

Implementiere die Methode **berechneSummeDiagonaleObenLinksNach UntenRechts()**,welche die Summe der Zahlen in der Diagonalen von links oben nach rechts unten berechnet.

Die Methode soll so entworfen werden, dass sie auch für kürzere bzw. längere Arrays gültig ist und bei einer Programmwiederholung auch richtig ausgeführt wird. Auch bei kürzeren bzw. längeren Arrays sind die Längen der ersten und der zweiten Dimension identisch.

***Alternative zur Variante ”eine for-Schleife”***

*public void berechneSummeDiagonaleObenLinksNachUntenRechts(){*

*summe = 0;*

*for (int i= 0;i < zahlen.length; i++) {*

*summe += zahlen[i][i]);*

*}*

*}*

*public void berechneSummeDiagonaleObenLinksNachUntenRechts(){*

*summe = 0;*

*for (int i= zahlen.length-1;i >= 0;i--) {*

*summe += zahlen[i][i]);*

*}*

*}*

***Alternative zur Variante ”zwei for-Schleifen”***

*public void berechneSummeDiagonaleObenLinksNachUntenRechts(){*

*summe = 0;*

*for (int i= 0;i < zahlen.length; i++) {*

*for (int j= 0;j < zahlen[0].length; j++) {*

*if(i== j){*

*summe += zahlen[i][j]);*

*}*

*}*

*}*

*}*

*Übugn*

1. Programmiere die Methode **berechneSummeDiagonaleUntenLinksNach ObenRechts()**,welche die Summe der Zahlen in der Diagonalen von links unten nach rechts oben berechnet.

***Alternative zur Variante ”eine for-Schleife”***

*public void berechneSummeDiagonaleUntenLinksNachObenRechts(){*

*summe = 0;*

*for (int i = 0;i < zahlen.length;i++) {*

*summe+= zahlen[tabelle.length-1-i][i]);*

*}*

*}*

***Alternative zur Variante ”zwei for-Schleifen”***

*public void berechneSummeDiagonaleUntenLinksNachObenRechts(){*

*summe = 0;*

*for (int i=0 ;i < zahlen.length; i++) {*

*for (int j = 0 ;j < zahlen[0].length;j++) {*

*if ((j+i)==zahlen.length-1){*

*summe += zahlen[i][i]);*

*}*

*}*

*}*

1. Gegeben ist folgendes zweidimensionales Array zahlen[][]:

**Index j**

0 1 2 3 4 5

0 (21 10 23 24 25 26)

1 (16 17 18 19 14 16)

2 (51 52 53 54 55 56)

**Index i** 3 (31 42 32 44 45 36)

4 (99 92 93 94 75 96)

5 (91 13 73 74 95 76)

Implementiere die Methode **berechneKleinsteZahl()**,welche die kleinste Zahl eines Arrays berechnet und am Ende mit den Indexen ausgibt.

Beispiel für eine Ausgabe: „Die kleinste Zahl im Array beträgt 10 und ist an der Position i=0 und j=1 zu finden!“

Die Methode soll so entworfen werden, dass sie auch für kürzere bzw. längere Arrays gültig ist und bei einer Programmwiederholung auch richtig ausgeführt wird. Der Methode soll auch funktionieren, wenn negative Zahlen im Array vorhanden sind.

1. Gegeben ist folgendes zweidimensionales Array zahlen[][]:

**Index j**

0 1 2 3 4 5

0 (21 12 23 24 25 26)

1 (16 17 18 19 10 16)

2 (13 13 13 13 13 13)

**Index i** 3 (31 42 32 44 45 36)

4 (11 92 93 94 75 96)

5 (91 12 73 74 95 76)

Überprüfe in einer Methode **pruefeGleicheZahlenInEinerZeile(),** ob in einer Zeile des Arrays zahlen[][] nur gleiche Zahlen stehen.

Wenn dies in einer Zeile der Fall ist, soll z.B. die Ausgabe „Es stehen in der 3. Zeile im Array nur gleiche Zahlen!“ auf dem Bildschirm erscheinen.

Die Methode soll so entworfen werden, dass sie auch für kürzere bzw. längere Arrays gültig ist und bei einer Programmwiederholung auch richtig ausgeführt wird.

1. Ermittle in einer Methode ermitteleMaximumDerDiagonale() den maximalen Wert der zahlen[][]-Diagonale von links oben nach rechts unten.

**Index j**

0 1 2 3 4 5

0 (**1** 2 3 4 5 6)

1 (1 **2** 3 4 5 6)

2 (1 2 **8** 4 5 6)

**Index i** 3 (1 2 3 **4** 5 6)

4 (1 2 3 4 **5** 6)

5 (1 2 3 4 5 **6)**

Die größte Zahl sowie die beiden Indizes sind am Ende der Methode auszugeben, z.B. „Die Zahl [2] [2] ist die größte Zahl und beträgt: 8“

Die Methode soll so entworfen werden, dass sie natürlich auch für Arrays mit negativen Zahlen bzw. auch für kürzere bzw. längere Arrays gültig ist.

1. Ermittle in einer Methode ermitteleMaximumDerDiagonale() den maximalen Wert der zahlen[][]-Diagonale von links unten nach rechts oben.*public void ermitteleMaximumDerDiagonale(){ //Aufgabe 2 int max = zahlen[0][0]*

*int x = 0;*

*int y =0;*

*for (int i = 0;i < zahlen.length;i++) {*

*if(max < zahlen[i][i]){*

*max = zahlen[i][i];*

*x = i;*

*y = i;*

*}*

*}*

*System.out.println("\n\nDie Zahl [" + x + "][" + y + "] ist die größte Zahl und beträgt: " + max);*

*}*

*Auf*

**Index j**

0 1 2 3 4 5

0 (1 2 3 4 5 **6**)

1 (1 2 3 4 **1** 6)

2 (1 2 3 **4** 5 6)

**Index i** 3 (1 2 **8** 4 5 6)

4 (1 **2** 3 4 5 6)

5 (**7** 2 3 4 5 6)

Die größte Zahl sowie die beiden Indizes sind am Ende der Methode auszugeben, z.B. „Die Zahl [3] [2] ist die größte Zahl und beträgt: 8“

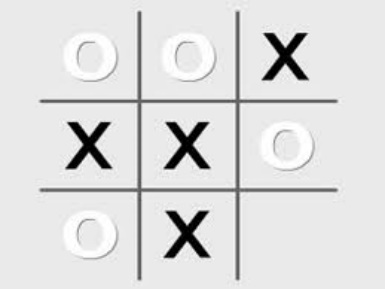
Die Methode soll so entworfen werden, dass sie natürlich auch für Arrays mit anderen Zahlen bzw. auch für kürzere bzw. längere Arrays gültig ist.

### Übung TicToc

Programmiere eine Simulation des Spieles „Tic Tac Toe“.

**Spielablauf**:

Tic Tac Toe ist ein klassisches, einfaches Zweipersonen-Strategiespiel, dessen Geschichte sich bis ins 12. Jahrhundert v. Chr. zurückverfolgen lässt.

[](http://images.google.de/imgres?imgurl=http://kean.foryounet.de/games/game_pic/tictactoe2.jpg&imgrefurl=http://kean.foryounet.de/%3Finh%3Dspiele&usg=__e-yJmd7R1m0Vj7_KYG5wIrlGauA=&h=300&w=400&sz=10&hl=de&start=63&um=1&tbnid=XZzxwra4D1cUsM:&tbnh=93&tbnw=124&prev=/images%3Fq%3Dtic%2Btac%2Btoe%2Bspiel%26ndsp%3D18%26hl%3Dde%26rlz%3D1T4GFRD_deDE312DE312%26sa%3DN%26start%3D54%26um%3D1)Auf einem 3×3-Felder großen Spielfeld machen die beiden Spieler abwechselnd ihre Zeichen (ein Spieler Kreuze (X), der andere Kreise (O)).

Der Spieler, der als erstes drei seiner Zeichen in einer Reihe, Spalte oder einer der beiden Hauptdiagonalen setzen kann, gewinnt.

Wenn allerdings beide Spieler optimal spielen, kann keiner gewinnen, und es kommt zu einem Unentschieden.

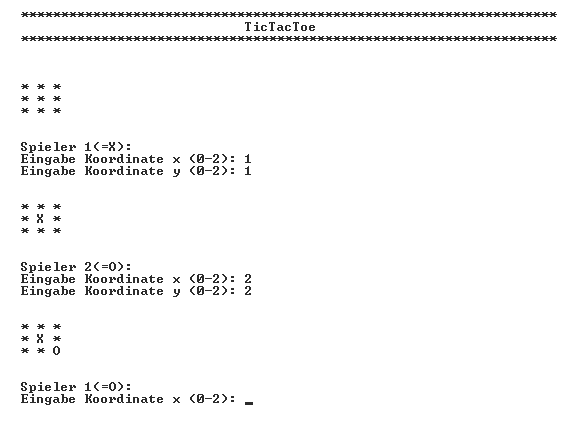
**Anforderungen:**

Nach Ausgabe des mit Sternchen (\*) gefüllten Spielfeldes sollen die Spieler 1 und 2 solange abwechselnd ein X bzw. O durch Eingabe der Koordinaten x und y setzen bis ein Spieler gewonnen hat oder alle Felder befüllt sind.

Ein Feld, welches von einem Spieler mit einem X bzw. O bereits belegt wird, darf nicht noch einmal ausgewählt werden.

Definiere das Spielfeld als zweidimensionale Array-Variable **char[][] spielFeld**

Verwende die Methoden starteSpiel(), setzeSpielstein(), ausgabeSpielfeld() und auswertungTreffer()



**Zusatzaufgabe**

Erweiterw das TicTacToe-Programm so, dass die Spielfeldgröße von dem Benutzer vorher festgelegt werden kann, so dass das Spielen auf einem 4×4-, 5×5- usw. großen Spielfeld auch möglich ist.

# Parameterübergabe an eine Methode

Eines der Grundprinzipien moderner Programmiersprachen ist das modulare Erstellen von Programmteilen. Um dieses Konzept konsequent umzusetzen, müssen unabhängige Methoden zwischen den Programmierern vereinbart werden.

Es muss festgelegt werden:

* Was soll eine Methode leisten?
* Welche Daten müssen der Methode zur Erfüllung ihrer Aufgaben bereitgestellt werden?
* In welcher Form soll die Methode ihr Ergebnis zur weiteren Verarbeitung bereitstellen?

Sind diese Punkte vereinbart, dann braucht sich der Programmierer ausschließlich damit zu beschäftigen, wie er die von ihm erwartete Leistung umsetzt. Er kennt die Voraussetzung (Bereitstellung der Daten), und er kennt die Anforderungen an seine Methode(Art und Form des Ergebnisses).

Innerhalb seiner Methode wird er lokale Variable verwenden, mit denen er das Ergebnis seines Programmteils ermittelt. Was in anderen Methoden bei anderen Programmierern passiert, interessiert ihn nicht, denn die Programmschnittstellen wurden ja zuvor geklärt.

**Grundaufbau einer Methode**

public class Program {

int x = 3;

int y = 2;

public static void main(String []args){

plus(x,y);



}

public static void plus(int i, int j) {

i = i + j;

System.out.println(i);



}

***Vorgehensweise:***

**Definition der Methode im Detail:**

1.Zeile (**Methodenkopf**): Schlüsselwort **public**: Dies ist ein sogenannter Sichtbarkeitsmodifizierer und bedeutet, dass jeder von überall auf die Methode zugreifen kann. Mehr dazu später.

Schlüsselwort **static**: Dadurch wird die gesamte Methode statisch, sprich sie ist nicht an ein Objekt gebunden. Dazu auch später mehr.

Schlüsselwort **void**: bedeutet, die Methode besitzt kein Rückgabewert. Ein Rückgabewert ist ein Ergebnis, dass eine Methode am Ende der Ausführung an den Aufrufer zurückgibt. Wenn du eine Rückgabewert haben möchtest, schreibst du statt void einfach den gewünschten Datentyp, denn der Rückgabewert besitzt. (Beispiel siehe unten)

Nach dem Rückgabetyp folgt der **Methodenbezeichner**, sprich der Name der Methode. (Hier plus)

Es folgt ein rundes Klammerpaar, in welchem man die Parameter Variablen deklariert. Die Initialisierung der Variablen erfolgt beim Methodenaufruf. Hier 2 Parameter vom Typen Integer mit dem Bezeichner i und j.

2. – 3. Zeile (**Methodenblock**): Hier werden nun innerhalb der geschweiften Klammern die Anweisungen hineingeschrieben. In diesem Beispiel existieren 2 Anweisungen. In der 1. Anweisung werden die Zahlen 3 und 2 addiert. In der 2. Anweisungen wird das Ergebnis auf der Konsole ausgegeben.

**Was wird übergeben?**

Sind die Übergabeparameter von **primitiven** Datentypen (z.B. integer, double) dann erhält eine Methode **Kopien** dieser Variablen. Das bedeutet, im Falle plus(), dass nach Beendigung der Methode die Variable x immer noch den ursprünglichen Wert besitzt. Die Addition hat nur innerhalb der Methode mit den Kopien stattgefunden.

Bei der Übergabe von **Objekten** (z.B. array) wird allerdings die **Referenz** auf den Speicherbereich der Objekte übergeben. Damit verändert die Methode auch die ursprünglichen Daten.

**Ausnahme: String wird immer als Kopie übergeben!**

**Rückgabewert von Methoden**

Soll eine Methode Daten dauerhaft verändern, oder ein Ergebnis nach außen weitergeben, dann muss im Methodenkopf ein Rückgabewert vereinbart werden.   
public class Program {

int x = 3;

int y = 2;

int z;

public static void main(String []args){

z = plus(x,y);

System.out.println(z); }

public int plus(int i, int j) {

int result = i + j;



return result; }

In unserem Beispiel tauschen wir im Methodenkopf das Wort void gegen int. Dies bedeutet, dass der Rückgabewert dieser Methode vom Typ Integer ist. Der Methodenkörper besteht wieder aus 2 Anweisungen. Die 1. Anweisung ist die Initialisierung einer Variable namens result vom Typ int. Ihr wird die Summe aus i und j zugewiesen. In der 2. Anweisung wird dieses Ergebnis dann als Rückgabewert zurückgeben. Das Schlüsselwort lautet hier **return**.

Der Methodenaufruf ist so etwas wie ein Platzhalter für den Rückgabewert. Man kann den Aufruf also mit dem "=" -Operator zuweisen. Nach der Zuweisung des Rückgabewerts in die Variable z, wird der Wert 5 auf der Konsole ausgeben.

Man kann auch als Parameter für Print() direkt den Aufruf übergeben.

System.out.print(plus(x,y));

### Übung Parameterübergabe mit Rückgabewert

1. Erstelle eine Klasse mit einer Methode **public boolean teilbarkeit(int zahl, int divisor)** mittels der geprüft werden kann, ob eine Zahl durch eine andere glatt teilbar ist. Die Klasse soll eine eingabe() und ausgabe()-Methode enthalten.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

1. Erweitere das Programm aus Aufgabe a, indem du alle möglichen Teilerzahlen einer einzugebenden Zahl ausgeben lässt. Als Rückgabewert soll hier ein Array zurückgegeben werden

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Die Kennung der Methode soll wie folgt aussehen: **public int[] teilbarkeit(int zahl)**

### Übung „Zahlen runden“

Erstelle ein Programm, das eine Methode public double runden(double zahl, int stellen) benutzt, die eine Dezimalzahl auf eine bestimmte Anzahl von Stellen nach dem Komma kaufmännisch rundet.

**Hinweis:**

Für 2 Stellen nach dem Komma gilt: ((int) ((zahl\*100)+0,5))/100

Um die Formel flexibler zu gestalten, ist hier die Formel Math.pow(Basis, Exponent) hilfreich.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung