Übung 11: *Fibonacci-Folge*

Die Fibonacci-Folge ist definiert als:
Jedes Element der Folge ist die Summe der beiden vorangegangenden Elemente.
Wobei die ersten beiden Elemente den Wert 1 haben.

Somit: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Es soll ein Programm erstellt werden, dass vom Benutzer die Anzahl der zu berechnenden Elemente der Fibonacci-Folge erfragt und diese dann entsprechend berechnet und auf die Konsole ausgibt.

```
Anzahl der zu berechnenden Fibonacci-Zahlen (>=2): 6
Fibonacci-Zahlen: 1, 1, 2, 3, 5, 8
```

Übung 12: *Cast*

Da Java eine *typensichere* Sprache ist, sind Zuweisungen von ungleichen Datentyp nicht erlaubt.
Dennoch kann z.B. eine Zuweisung eines *double* in einen *int* erzwungen werden unter Zuhilfenahme des sog. *Cast-Operators*: ***(type)***
Dabei wird mit ***type*** der Ziel-Datentyp spezifiziert.

Beispiel:

```
double franken = 1.25;
double rappenDouble = franken * 100;
int rappenInt = (int) rappenDouble;
```

Es soll ein Programm erstellt werden, dass vom Benutzer einen Franken-Betrag erfragt und dann diesen in Rappen umrechnet und auf die Konsole ausgibt.
Beispiel:

```
Geben Sie einen Franken-Betrag an : 1.25
Umgerechnet in Rappen             = 125
```


Übung 13: *Überlauf*

Es soll eine Variable vom Type **`byte`** definiert und mit 0 initialisiert werden.
Daraufhin soll diese **`byte`**-Variable 300 mal jeweils um 1 inkrementiert und dieser neue Wert auf die Konsole ausgegeben werden.
Was stellt man fest ?

Hinweis:
Eine Zahl kann mit dem sog*. Inkrement-Operator* **++** um den Wert 1 inkrementiert werden.
Beispiel: `meineZahl++;`

Übung 14: *Fakultät*

Es soll ein Programm erstellt werden welches die Fakultät berechnet.
Die Fakultät für eine Zahl *n* ist definiert als: 1 * 2 * 3 * .. n
Die Fakultät von 0 ist 1 und für negative Zahlen nicht bestimmt.
Beispiel: Fakultät von 4 ist 1 * 2 * 3 * 4 ⇨ 24
Es soll nun ein entsprechendes Programm erstellt werden, welches die Zahl *n* von der Konsole einliest, die Fakultät berechnet und das Resultat auf die Konsole ausgibt.

Übung 15: *Quersumme*

Die Quersumme ist definiert als die Summe der einzelnen Ziffern einer Dezimalzahl.
Beispiel: 1234 = 1 + 2 + 3 + 4 ⇨ 10
Es soll nun ein Programm erstellt werden welches die Quersumme einer beliebigen Zahl berechnet.

Hinweise:
Es soll dazu die *Ganzahl-Division* (`/`) und den *Modulo-Operator (%)* verwendet werden.

Übung 16: *Zahlen raten*

Ein Spiel-Programm: das Programm ermittelt eine Zufallszahl zwischen 0 und 100 und der Spieler muss diese Zahl herausfinden.

Ablauf:
Das Programm bestimmt eine Zufallszahl mit:

```
int zufallsZahl = (int) (101 * Math.random());
```

Daraufhin wird der Benutzer nach einer Zahl abgefragt.
Diese Zahl wird mit der Zufallszahl verglichen und wenn sie falsch ist, wird dem Benutzer mitgeteilt, ob die eingegebene Zahl zu klein oder zu gross ist.
Dies wird wiederholt bis die richtige Zahl gefunden wurde.
Am Ende gibt das Programm bekannt, wieviele Versuche nötig waren.

Beispiel:

```
Geben Sie eine Zahl ein: 20
  Sorry, Zahl ist zu gross.
Geben Sie eine Zahl ein: 5
  Sorry, Zahl ist zu klein.
Geben Sie eine Zahl ein: 10

Bingo !

Anzahl Versuche: 3
```

Frage: Wieviele Versuche sind maximal nötig um die richtige Zahl zu finden?

Übung 17: *Primzahlen*

Es soll ein Programm erstellt werden welches alle Primzahlen in einem definierbaren Bereich sucht.

Beispiel:

```
Geben Sie den Bereich ein.
Untere Grenze (>=2): 2
Obere Grenze       : 30
Primzahlen         : 2 3 5 7 11 13 17 19 23 29
```

Hinweis:
Primzahlen sind natürliche Zahlen (ganze Zahlen > 0), die nur durch sich selbst und 1 ohne Rest teilbar sind (wobei 1 keine Primzahl ist).

