Halten Sie beim Programmieren rigoros ALLE aufgestellten Programmierregeln ein. Ihre Programme werden in Hinsicht auf die Einhaltung dieser Regeln kontrolliert.

1. Betrachten Sie folgendes Programmfragment, und stellen Sie die *Einrückungen* richtig ein. Beachten Sie dabei nicht, was die Anweisungen machen:

```
int a = 3; int b = 4; if (a == 3) if (b > a) a = 7; a = 6; if (b == a) { if
(a == 4) b = 0; a = 1; } else if (a == b * 2) while (a <= 10) a = a + 1; else
a = 1; b = a;
```

2. Suchen Sie die *Fehler* im folgenden Programmfragment. *Kennzeichnen* Sie diese. Beachten Sie dabei nicht, was das Programm macht:

```
int a = 3; int b = 7;
boolean fix = a * 3 - b == 7 || false;
if (a = b - 4 && (!fix + 3 == 2 || a < b) {
   b = 3
   if (a < b) {
      int c = 100;
      a = a + 1;
      while (a < a + 10
         a = a + 1;
   }
   c = c + 3;
else
   b = 3;
```

3. Welches *Ergebnis* liefern folgende Bedingungen? Dabei seien die verwendeten Variablen wie angegeben initialisiert worden:

int a = 3; **int** b = 7; **double** d = 3.0; **boolean** flex = false;

	Ergebnis
`!(a == b && !flex ) && !!(a - b < 0)`	
`a == b - 4 && flex != (a == b)`	
`a % b < a || !!flex || a / b == 0`	
`Math.round(((d / (b - 1)) - 0.1)) >= 0 && (b != 7 || !flex)`	
`!flex || a > b || Math.sin(a) > 0 || Math.cos(b) != 0 || a + b > d`	
`!(flex || !flex && flex) || flex`	

4. Geben Sie für jeden der folgenden umgangssprachlich beschriebenen Ausdrücke den entsprechenden *Java-Ausdruck* an. Der Java-Ausdruck soll **true** liefern, wenn die Bedingung zutrifft, und ansonsten **false**. i, j und k sind **int**-Variablen, b, c und d **boolean**-Variablen (versuchen Sie möglichst kurze Ausdrücke zu finden):

i, j und k sind alle verschieden von null	
i ist durch 17 teilbar und echt positiv	
j ist ungerade und liegt zwischen 20 und 40	
k ist entweder Vielfaches von 3 und 5, oder Vielfaches von 5 und 7, oder Vielfaches von 5 und 11	
Genau eines von b, c und d ist **true**	
b, c und d sind alle drei **true** oder alle drei **false**	

5. Übersetzen Sie folgende **while**-Schleife zuerst in eine **for**-Schleife mit *Startanweisung*, *Bedingung* und *Fortschaltungsanweisung*. Dann sollen Sie dieselbe **while**-Schleife in eine **for**-Schleife übersetzen, bei der Startanweisung, Bedingung und Fortschaltungsanweisung leer sind. Die übersetzten **for**-Schleifen sollen dasselbe Ergebnis wie die ursprüngliche **while**-Schleife liefern.

```
double i = 3;
while (i < 10) {
   i = i + 0.5;
   System.out.println(i);
}
```

6. Wie viele *Inkarnationen* der Variable d werden zur *Laufzeit* des Programmes angelegt?

```
for (int i = 0; i < 10; i = i + 1)
   double d = i * i;
```

7. Kann *nach dem Durchlaufen* der obigen Schleife in einer der Schleife folgenden Anweisung auf die Variable i zugegriffen werden?

8. Programmieren Sie die nachfolgenden **while**-Schleifen, welche folgende *Zahlenfolge* ausgeben:

999 996 993 ... 0

```
int i = ____;
while (____________________) {
   System.out.println(i);
   i = _______________
}
```

```
int i = ____;
while (____________________) {
   i = _______________
   System.out.println(i);
}
```

9. Schreiben Sie folgende **for**-Schleifen in **while**-Schleifen um:

```
for (int i = 3; i <= 10; i = i + 1)
   System.out.println(i);
```

```
for (int i = 1000; i > 6; i = i - 3)
   System.out.println(i);
```

10. Lösen Sie das Problem der *Aufgabe 8* durch eine **for**-Schleife.

11. Generieren Sie folgende *Zahlenfolgen* zuerst durch eine **while**- und dann durch eine **do-while**-Schleife. Könnten sie auch durch eine **for**-Schleife generiert werden?

0.0 0.01 0.02 0.03 ... 0.1

```
double i = _______;
while (____________) {
   System.out.println(i);
   i = ___________
}
```

```
double i = _______;
do {
   System.out.println(i);
   i = ___________
} while (____________);
```

1 3 9 27 ... 6561

```
int i = _______;
while (____________) {
   System.out.println(i);
   i = ___________
}
```

```
int i = _______;
do {
   System.out.println(i);
   i = ___________
} while (____________);
```

12. Wandeln Sie folgende Schleifen in **do-while**-Schleifen um:

```
for (int i = 16; i <= 44; i = i + 1)
   System.out.println(i);
```

```
for (int i = 16; i > 0; i = i - 1)
   System.out.println(i);
```

13. Schreiben Sie ein Programm mit dem Namen Median, welches von drei in beliebiger Reihenfolge eingegebenen ganzen Zahlen die *zweitkleinste Zahl* ermittelt.
Achten Sie darauf, dass Ihr Programm alle Sonderfälle berücksichtigt.
Programmieren Sie dabei die abgebildete Benutzerschnittstelle.

```
Median
======
1. Zahl: 2
2. Zahl: 1
3. Zahl: 2

Der Median lautet 2
```

14. Schreiben Sie ein Programm mit dem Namen GgTEuklid, welches den *größten gemeinsamen Teiler* zweier eingegebener Zahlen nach dem *Algorithmus von Euklid* ermittelt. Informieren Sie sich zuerst selbständig darüber, wie dieser Algorithmus genau funktioniert. Die Benutzerschnittstelle soll wie angegeben gestaltet werden.

```
GgT von Euklid
==============
Erste Zahl: 15
Zweite Zahl: 12

Der größte gemeinsame Teiler lautet 3
```

15. Die Folge der *Fibonacci-Zahlen* ist folgendermaßen definiert:

$$f_n = \begin{cases} 1 & falls\ n = 0\ oder\ n = 1 \\ f_{n-2} + f_{n-1} & falls\ n \geq 2 \end{cases}$$

Der Anfang der Fibonacci-Folge lautet 1, 1, 2, 3, 5, 8, …

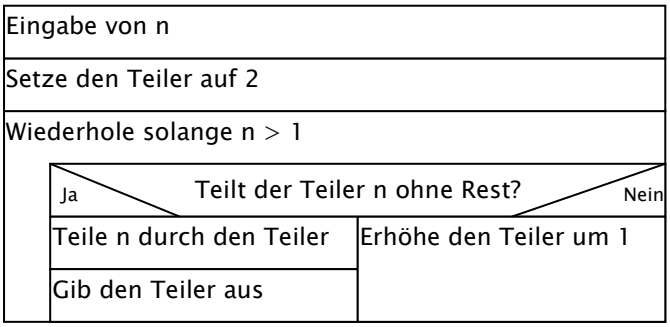
Schreiben Sie ein Programm mit dem Namen `Fibonacci`, das eine Zahl $n \geq 0$ einliest und die entsprechende Fibonacci-Zahl f_n berechnet. Ignorieren Sie arithmetischen Überlauf. Programmieren Sie die angegebene Benutzerschnittstelle.

```
Fibonacci-Zahlen
================
Die wievielte Zahl? -23
Die Zahl muss größer oder gleich Null sein
Die wievielte Zahl? -1
Die Zahl muss größer oder gleich Null sein
Die wievielte Zahl? 23

Die 23. Fibonacci-Zahl lautet 46368
```

16. Schreiben Sie ein Programm mit dem Namen `Primfaktoren`, das die *Primfaktorenzerlegung* einer Zahl $n \geq 2$ ermittelt. Die Primfaktoren sollen in *steigender Größe* ausgegeben werden. Ignorieren Sie arithmetischen Überlauf. Halten Sie die angegebene Benutzerschnittstelle ein.

```
Primfaktorzerlegung
===================
Geben Sie die Zahl ein: 0
Zahl muss größer als 1 sein
Geben Sie die Zahl ein: 1668

Die Zerlegung lautet:
2 * 2 * 3 * 139 = 1668
```

Das unten abgebildete Struktogramm zeigt einen Entwurf des Programms:



17. Eine Zahl nennt man eine *„Perfekte oder vollkommene Zahl“*, wenn sie gleich der Summe aller ihrer ganzzahligen Teiler ist (einschließlich der 1, ohne sie selbst). Die ersten beiden Perfekten Zahlen sind

    6 = 3 + 2 + 1 und
    28 = 14 + 7 + 4 + 2 + 1.

    Berechnen Sie die ersten *fünf* Perfekten Zahlen im Programm mit dem Namen `PerfekteZahlen`.

18. Schreiben Sie ein Programm mit dem Namen `UmrechnungNachZehn`, das Zahlen aus einem fremden Zahlensystem mit der Basis b in das Zehnersystem umrechnet. Zur Vereinfachung gilt $b < 10$. Eine Zahl n in einem fremden Zahlensystem mit der Basis b wird folgendermaßen umgerechnet:

    $2041_5 = 2 \cdot 5^3 + 0 \cdot 5^2 + 4 \cdot 5^1 + 1 \cdot 5^0 = 271_{10}$
    $11111110_2 = 254_{10}$

```
Umrechnung ins Zehnersystem
===========================
Geben Sie die Zahl ein: 2041
Geben Sie die Basis ein: 16
Basis muss zwischen 2 und 9 liegen
Geben Sie die Basis ein: 5

Die Zahl im Zehnersystem lautet 271
```

19. Schreiben Sie dann ein Programm mit dem Namen `UmrechnungVonZehn`, das eine Zahl aus dem Zehnersystem in ein fremdes Zahlensystem mit der Basis b umrechnet (wieder gilt $b < 10$). Gehen Sie analog zur vorigen Aufgabe vor. Halten Sie die angegebene Benutzerschnittstelle ein.

```
Umrechnung vom Zehnersystem
===========================
Geben Sie die Zahl ein: 271
Geben Sie die Basis ein: 16
Basis muss zwischen 2 und 9 liegen
Geben Sie die Basis ein: 5

Die Zahl im 5-ersystem lautet 2041
```

20. Schreiben Sie ein Programm mit dem Namen Zahlenraten, bei welchem sich der Computer eine Zahl zwischen 0 und 1000 ausdenkt und der Benutzer dann diese Zahl erraten muss. Der Computer gibt ihm den Hinweis, ob die eingegebene Zahl zu groß oder zu klein ist.

    Halten Sie die angegebene Benutzerschnittstelle ein.

```
Zahlenraten
===========
Ich habe mir eine Zahl im Intervall [0,1000]
ausgedacht. Versuchen Sie diese zu erraten
Ihr Tipp: 500
Ihre Zahl ist zu groß
...
Ihr Tipp: 101
Ihre Zahl ist zu klein
Ihr Tipp: 105
Mein Kompliment! Sie haben die Zahl gefunden
```

21. Schreiben Sie dann ein Programm mit dem Namen ZahlenratenUmgekehrt, bei welchem die obigen Rollen vertauscht werden und der Benutzer sich eine Zahl zwischen 0 und 1000 ausdenkt, die der Computer dann erraten soll.

    Wiederum sollen Sie die angegebene Benutzerschnittstelle einhalten.

    ZUSATZAUFGABE: Erweitern Sie das Programm so, dass es erkennt, wenn der Benutzer schwindelt und versucht das Programm zu täuschen.

```
Umgekehrtes Zahlenraten
=======================
Suchen Sie sich eine Zahl im Intervall [0, 1000]
aus. Ich werde sie finden.
Mein Tipp: 500
Zahl kleiner (0), größer (1), gefunden (2): 0
Mein Tipp: 250
...
Zahl kleiner (0), größer (1), gefunden (2): 1
Mein Tipp: 271
Zahl kleiner (0), größer (1), gefunden (2): 2
Ich habe die Zahl gefunden!
```

22. Schreiben Sie ein Programm mit dem Namen DreieckEigenschaften, welches die drei Seiten eines Dreiecks einliest und die nachfolgend angegebenen *Eigenschaften* des Dreiecks der Reihe nach ermittelt.

    Dabei sollen Sie die angegebene Benutzerschnittstelle programmieren und das Programm so erstellen, dass auch mehrere Dreiecke der Reihe nach analysiert werden können.

    - **Unmögliches Dreieck:** Dieses entsteht, wenn eine Seite länger als die beiden anderen Seiten ist.
    - **Umfang**
    - **Fläche:** Zur Ermittlung der Fläche sollen die die *„Heronsche Flächenformel“* verwenden. Informieren Sie sich zuerst selbständig darüber, wie diese lautet.
    - **Gleichseitiges Dreieck**
    - **Gleichschenkeliges Dreieck**
    - **Rechtwinkeliges Dreieck:** Wenn das Dreieck rechtwinkelig ist, sollen Sie ebenfalls kontrollieren, ob das Dreieck ein *„Pythagoreisches Dreieck“* ist. Das bedeutet, dass alle drei Seiten ganzzahlig sind. Weiters sollen Sie für ein rechtwinkeliges und pythagoreisches Dreieck die *Hypotenuse* ermitteln.

```
Eigenschaften eines Dreiecks
============================
Seite a: 5
Seite b: 4
Seite c: 3

Umfang: 12.0
Fläche: 6.0
Pythagoreisches Dreieck
Hypothenuse: 5.0

Nochmal (0), Beenden (1): 0

Seite a: 3
Seite b: 3
Seite c: 10
Unmögliches Dreieck

Nochmal (0), Beenden (1): 1
```

23. Mathematiklehrer ziehen es vor, zur Schularbeit nur *pythagoreische Dreiecke* zu verwenden, deren Seiten ganzzahlig sind – z. B. Seitenlängen: 3, 4 und 5. Schreiben Sie ein Programm mit dem Namen PythagoreischeTripel, das alle solchen Dreiecke bis zu einer maximalen Seitenlänge von 1000 ermittelt.

	A	B	C
1	Pythagoreische Tripel		
2	Seite a	Seite b	Seite c
3	3	4	5
4	5	12	13
5	6	8	10
6	7	24	25
7	8	15	17
8	9	12	15
9	9	40	41
10	10	24	26

Dabei soll jedes Dreieck in der Liste genau einmal vorkommen, d. h. es soll in der Liste keine doppelten Dreiecke geben.

Das Programm benötigt keine Benutzerführung, es sollen aber die ermittelten Dreiecke in eine mit *Excel* bearbeitbare Datei mit dem Namen `PythagoreischeTripel.csv` (Abk. *Character Separated Values*) geschrieben werden. Das bedeutet, dass die einzelnen Spalten durch ein bestimmtes Trennzeichen – z. B. den Strichpunkt (;) – getrennt werden. Fügen Sie in die Datei – wie angegeben – noch eine sinnvolle Beschriftung der Tabelle und der Spalten ein.

Damit die Ausgabebefehle (`System.out.println`) in eine Datei umgeleitet werden, sollen Sie folgendes Programmfragment benutzen:

```
try {
   // Umleiten der Standardausgabe in die Datei welche unter dem angegebenen
   // Laufwerk und Pfad gespeichert wird. Ist die Datei dort nicht vorhanden,
   // wird sie angelegt
   System.setOut(new java.io.PrintStream("H:\\PythagoreischeTripel.csv"));
} catch (java.io.FileNotFoundException e) {
   // Es ist ein Fehler beim Erstellen oder Öffnen der Datei aufgetreten
   System.out.println("Fehler beim Erstellen der Datei");
}
```