Bereich: Kontrollstrukturen

Schaltjahr	Schwierigkeit: ★☆☆☆☆
Package: `de.dhbwka.java.exercise.control`	**Klasse:** `LeapYear`

Aufgabenstellung:

Schreiben Sie ein Programm `LeapYear`, das eine Jahreszahl von der Konsole einliest und dann bestimmt, ob es sich bei diesem Jahr um ein Schaltjahr handelt!
Das Ergebnis soll auf der Konsole ausgegeben werden.

Ein Jahr ist ein Schaltjahr, wenn es durch vier teilbar ist, nicht aber wenn es durch 100 teilbar ist, es sei denn, es ist durch 400 teilbar.

Beispieldialog:

```
Welches Jahr soll auf Schaltjahr geprüft werden? 2018
2018 ist kein Schaltjahr

Welches Jahr soll auf Schaltjahr geprüft werden? 2020
2020 ist ein Schaltjahr

Welches Jahr soll auf Schaltjahr geprüft werden? 2000
2000 ist ein Schaltjahr

Welches Jahr soll auf Schaltjahr geprüft werden? 2100
2100 ist kein Schaltjahr
```

Bereich: Kontrollstrukturen

Temperaturtabelle	Schwierigkeit: ★☆☆☆☆
Package: `de.dhbwka.java.exercise.control`	**Klasse:** `TemperatureTable`

Aufgabenstellung:

Erstellen Sie ein Programm zur Berechnung und Ausgabe einer Temperaturtabelle von Fahrenheit (f) nach Celsius (c)!
Die Tabelle soll bei 0° F beginnen und bei 300° F enden. Die Abstände der Stützstellen sollen 20° F betragen.
Der funktionale Zusammenhang von *c* und *f* ist gegeben durch die Formel:

$$c = (5 / 9) * (f - 32)$$

Beispielausgabe:

```
Fahrenheit | Celsius
-----------+--------
  0        | -17.8
...
300        | 148.9
```

Bereich: Kontrollstrukturen

Wildbestand	Schwierigkeit: ★☆☆☆☆
Package: `de.dhbwka.java.exercise.control`	**Klasse:** `Deers`

Aufgabenstellung:

Der Wildbestand eines Forstes umfasst zu Beginn 200 Hirsche.

Die jährliche Vermehrung beträgt 10% des Jahresanfangsbestands; im Herbst werden 15 Stück zum Abschuss freigegeben.

Wie entwickelt sich der Bestand in den nächsten Jahren?

Erstellen Sie ein Programm `Deers`, das den jährlichen Wildbestand ermittelt und ausgibt, bis dieser mindestens die Zahl 300 erreicht hat!

Beispielausgabe:

```
1: 205 Hirsche
2: 210 Hirsche
3: 216 Hirsche
4: 222 Hirsche
5: 229 Hirsche
6: 236 Hirsche
7: 244 Hirsche
8: 253 Hirsche
9: 263 Hirsche
10: 274 Hirsche
11: 286 Hirsche
12: 299 Hirsche
13: 313 Hirsche
```

Bereich: Kontrollstrukturen

Einmaleins	Schwierigkeit: ★☆☆☆☆
Package: `de.dhbwka.java.exercise.control`	**Klasse:** `MultiplicationTable`

Aufgabenstellung:

Entwickeln Sie ein Programm `MultiplicationTable`, welches das kleine Einmaleins in Tabellenform (10x10-Tabelle) berechnet und ausgibt!

Beispielausgabe:

```
   1   2   3   4   5   6   7   8   9  10
   2   4   6   8  10  12  14  16  18  20
   3   6   9  12  15  18  21  24  27  30
   4   8  12  16  20  24  28  32  36  40
   5  10  15  20  25  30  35  40  45  50
   6  12  18  24  30  36  42  48  54  60
   7  14  21  28  35  42  49  56  63  70
   8  16  24  32  40  48  56  64  72  80
   9  18  27  36  45  54  63  72  81  90
  10  20  30  40  50  60  70  80  90 100
```

Bereich: Kontrollstrukturen

Aufsummieren	Schwierigkeit: ★☆☆☆☆
Package: `de.dhbwka.java.exercise.control`	**Klasse:** `AddUp`

Aufgabenstellung:

Entwickeln Sie ein Programm `AddUp`, das beliebig viele positive Zahlen von der Konsole einliest und diese Zahlen aufsummiert!

Verwenden Sie als Abbruchkriterium die Eingabe einer negativen Zahl (diese soll nicht mehr hinzugezählt werden).
Am Ende soll die Summe aller eingegebenen positiven Zahlen ausgegeben werden.

Realisieren Sie `AddUp` einmal mit einer `while`-Schleife und einmal mit einer `do-while`-Schleife!

Beispieldialog:

```
Zahl eingeben (<0 für Abbruch): 4
Zahl eingeben (<0 für Abbruch): 3
Zahl eingeben (<0 für Abbruch): 7
Zahl eingeben (<0 für Abbruch): 2
Zahl eingeben (<0 für Abbruch): -1
Summe: 16
```

Bereich: Kontrollstrukturen

Schuhgrößen	Schwierigkeit: ★★☆☆☆
Package: `de.dhbwka.java.exercise.control`	**Klasse:** `ShoeSize`

Aufgabenstellung:

Für die Umrechnung der in Deutschland üblichen Schuhgrößen in Zentimeter gilt die folgende Beziehung:

*Schuhgröße = Zentimeter * 1,5*

Entwickeln Sie ein Programm `ShoeSize`, das eine Tabelle der folgenden Art berechnet und ausgibt!

```
Zentimeter     | Größe
---------------+------
19,33 - 20,00 | 30
20,00 - 20,67 | 31
20,67 - 21,33 | 32
21,33 - 22,00 | 33
22,00 - 22,67 | 34
22,67 - 23,33 | 35
23,33 - 24,00 | 36
24,00 - 24,67 | 37
24,67 - 25,33 | 38
25,33 - 26,00 | 39
26,00 - 26,67 | 40
26,67 - 27,33 | 41
27,33 - 28,00 | 42
28,00 - 28,67 | 43
28,67 - 29,33 | 44
29,33 - 30,00 | 45
30,00 - 30,67 | 46
30,67 - 31,33 | 47
31,33 - 32,00 | 48
32,00 - 32,67 | 49
```

Bereich: Kontrollstrukturen

Babylonisches Wurzelziehen (Heronverfahren)	Schwierigkeit: ★★☆☆☆
Package: `de.dhbwka.java.exercise.control`	**Klasse:** `Babylon`

Aufgabenstellung:

Das Babylonische Wurzelziehen (oft auch Heron-Verfahren) ist ein alter iterativer Algorithmus zur Bestimmung einer rationalen Näherung der Quadratwurzel einer Zahl. Es ist ein Spezialfall des Newton-Verfahrens.

Die Iterationsvorschrift lautet:

$$x_{n+1} = \frac{x_n + \frac{a}{x_n}}{2}$$

Hierbei steht *a* für die Zahl, deren Quadratwurzel bestimmt werden soll. Der Startwert x_0 der Iteration kann, solange er nicht gleich 0 (Null) ist, beliebig festgesetzt werden, wobei zu beachten ist, dass negative Werte gegen die negative Quadratwurzel konvergieren.

Implementieren Sie das Babylonische Wurzelziehen so, dass eine Zahl eingelesen und deren Wurzel auf 6 Stellen hinter dem Komma genau berechnet wird (bzw. bis der Betrag von $x_{n+1} - x_n$ kleiner als 10^{-6} ist)! Geben Sie zur Kontrolle die Werte der einzelnen Berechnungsschritte aus!

Hinweis:
Zur Berechnung des Betrages können Sie die Methode `Math.abs()` benutzen.

Beispieldialog:

```
Wurzel aus welcher Zahl ziehen? 25
xn: 1.0
xn: 13.0
xn: 7.461538461538462
xn: 5.406026962727994
xn: 5.015247601944898
xn: 5.000023178253949
xn: 5.000000000053722

Die Wurzel aus 25.0 ist 5.0
```

Bereich: Kontrollstrukturen

Zahlenraten	Schwierigkeit: ★★☆☆☆
Package: `de.dhbwka.java.exercise.control`	**Klasse:** `NumberGuess`

Aufgabenstellung:

Schreiben Sie eine Java-Applikation `NumberGuess`, die ein einfaches Ratespiel implementiert!

Bei diesem Ratespiel muss der Benutzer eine zufällig erzeugte Zahl zwischen 1 und 100 erraten.
Als Hinweis bekommt er jeweils angezeigt, ob er zu hoch oder zu niedrig getippt hat.
Die Anzahl der Versuche, die er benötigt, wird mitgezählt.

Der Benutzer soll folgendes eingeben können:

- Spielername
- Jeweils die nächste zu tippende Zahl
- Endabfrage, ob die Applikation beendet werden (Eingabewert: „`0`“)
  oder ein weiteres Spiel durchgeführt werden soll (Eingabewert: „`1`“).

Das Programm soll nach Eingabe eines Tipps prüfen, ob die aktuelle Eingabe höher, niedriger oder gleich der gesuchten Zahl ist.
Es soll dann die Nummer des Versuches ausgeben und ob der Tipp zu hoch, zu niedrig oder korrekt war.

Hinweis: Mit `Math.random()` können reelle Zufallszahlen aus dem Bereich [0,1) erzeugt werden.

Beispieldialog:

```
Wie ist Dein Name? Donald
Donald, rate eine Zahl [1-100]: 50
Versuch 1: 50 ist zu hoch.
Donald, rate eine Zahl [1-100]: 25
Versuch 2: 25 ist zu hoch.
Donald, rate eine Zahl [1-100]: 12
Versuch 3: 12 ist zu hoch.
Donald, rate eine Zahl [1-100]: 6
Versuch 4: 6 ist zu niedrig.
Donald, rate eine Zahl [1-100]: 9
Versuch 5: 9 ist zu hoch.
Donald, rate eine Zahl [1-100]: 7
Versuch 6: 7 ist korrekt.
Was möchtest Du tun?
0 - Das Spiel beenden
1 - Das Spiel fortsetzen 0
```