

Programmieraufgabe 1

Grundsätzliches

- In dieser Aufgabe übst du den Umgang mit Methoden, Variablen, Operationen, Schleifen und Arrays.
- Eine Aufgabe zählt nur als vollständig, wenn sie Testcode aufweist.
- Passe auf die Formulierungen auf: 'zurückgeben' heisst als `return`-Wert, 'ausgeben' heisst auf der Konsole mit `print()`-Befehlen.
- Mache dir Notizen, was alles anders ist als in Python (und anderen Programmiersprachen, die du allenfalls kennst). Wir werden dies von Zeit zu Zeit zusammen diskutieren.

Methoden und Variablen

Ressourcen

- `Main.java` und `Calculations.java`
- [W3Schools](#): Seiten "Java Syntax" bis und mit "Java If...Else" sowie den ganzen Block "Java Methods". Es ist sinnvoll, diese verstanden zu haben.
- [GeeksForGeeks](#): "Basics of Java", "Operators in Java" und erster Teil von "Flow Control in Java"

Aufgabenstellungen

a) Schreibe eine Methode `hours()`, die eine ganzzahlige Anzahl Sekunden entgegennimmt und sie in Stunden, Minuten und Sekunden umrechnet und auf der Konsole ausgibt. Die Eingabe 1234 soll z.B. zur Ausgabe `0:20:34` führen. Teste dein Programm mit vernünftigen Eingabewerten, zum Beispiel mit 0, 59, 60, 100, 3600 und 4000. Stelle auch sicher, dass Ausgaben wie `1:3:11` zu `1:03:11` umgewandelt werden (möglichst elegant mit ternärem Operator).

b) Abstand zwischen zwei Punkten: Schreibe eine Methode `distance()`, die die x- und y-Koordinaten zweier Punkte als `int`-Argumente entgegennimmt und den Abstand zwischen ihnen berechnet und zurückgibt. Hinweis: Die ganzzahlige Wurzel einer Zahl `x` kannst du mit `Math.sqrt(x)` berechnen.

Schleifen

Ressourcen

- `Powers.java`
- [W3Schools](#): Seiten "Java Switch" bis und mit "Java Break/ Continue".
- [GeeksForGeeks](#): "Flow Control in Java"

Aufgabenstellungen

c) Schreibe eine Methode `primeFactorisation()`, die eine positive `int`-Zahl `n` in ihre Primfaktoren zerlegt. Die Methode soll die Zerlegung auf der Konsole ausgeben, z.B. für das Argument `12` soll auf der Konsole die Zeile `Prime facorisation of 12 = 2 * 2 * 3` ausgegeben werden.

d) Berechnung von Pi:

Schreibe eine Methode `pi()`, die einen Integer `digits` entgegennimmt und Pi auf mindestens `digits` Stellen nach dem Komma angenähert zurückgibt. Dabei soll die [Madhava-Leibniz-Reihe](#) gebraucht werden.

Tipp: Mit `Math.abs()` berechnest du den Betrag einer Zahl.

Zusatzaufgabe: Wie viele Stellen sind möglich? Was sind die limitierenden Faktoren, und kannst du sie umgehen/ ausschalten?

Arrays

Ressourcen

- `SerialHello.java` und `TicTacToe.java`
- `ArrayTests.java`: Variablen, die für Arrays stehen, sind Objektvariablen (d.h. Zeigervariablen). Für den Unterschied siehe [GeeksForGeeks: Primitive data type vs. Object data type](#).
- [W3Schools](#): "Java Arrays"
- [GeeksForGeeks](#): "Arrays in Java"

Aufgabenstellungen

e) Schreibe ein Programm `sort()`, das die Werte in einem `double`-Array aufsteigend sortiert und das sortierte Array zurückgibt. Eine ineffiziente, aber einfache Möglichkeit ist, benachbarte Elemente zu betrachten und jeweils zu vertauschen, wenn sie falsch zueinander stehen.

Zusatzaufgabe: Implementiere ein [effizienteres Sortierverfahren](#) (oder mehrere).

f) Ein magisches Quadrat ist ein Zahlenquadrat, in dem die Summe jeder Zeile, jeder Spalte und der beiden Diagonalen jeweils den gleichen Wert ergibt. Schreibe eine Methode `isMagicSquare`, die quadratisches Array von ganzen Zahlen als Argument entgegennimmt und zurückgibt, ob das Array ein magisches Quadrat ist.

```
{ { 12, 6, 15, 1 },
  { 13, 3, 10, 8 },
  { 2, 16, 5, 11 },
  { 7, 9, 4, 14 }
}
```

ist ein magisches Quadrat

*g) [Conways Game of Life](#): Das Spiel basiert auf der folgenden Idee: Die Spielwelt besteht aus einer Matrix von Zellen, die entweder leben oder tot sind. Jede Zelle hat 8 Nachbarn, Randzellen haben die Zellen des gegenüberliegenden Randes als Nachbarn. Aus der momentanen Zellenpopulation kann man die Population in der nächsten Generation durch folgende Regeln berechnen:

- Hat eine tote Zelle genau 3 lebende Nachbarn, erwacht sie zum Leben.
- Hat eine lebende Zelle 2 oder 3 lebende Nachbarn, bleibt sie am Leben.
- Alle anderen Zellen sterben.

Aufgabe:

- Programmiere die Aufgabe in einer separaten Klasse `GameOfLife`.

- Schreibe eine Methode `createRandom()`, das zwei Zahlen `n` und `m` entgegennimmt und eine zufällig gefüllte `boolean`-Matrix der Dimension `m*n` zurückgibt.
- Schreibe eine Methode `getNextGeneration()`, das eine zweidimensionale `boolean`-Matrix `cells` entgegennimmt und aus der übergebenen Matrix die Matrix der nächsten Generation berechnet und zurückgibt.
- Schreibe eine Methode `printCells()`, das eine zweidimensionale `boolean`-Matrix `cells` entgegennimmt und auf der Konsole darstellt. Dabei sollen lebende Zellen mit `@` dargestellt werden und tote entweder mit Leerschlag oder mit einem Punkt.
- Die `main`-Methode soll Zahlen `m`, `n` und `numberOfGenerations` definieren (wähle selbst sinnvolle Werte), eine `m*n`-Generation erzeugen und sie auf der Konsole ausgeben. Danach soll sie `numberOfGenerations` Generationen berechnen und ebenfalls ausgeben. Zwischen den Generationen kann mit `TimeUnit.SECONDS.sleep(1);` eine Pause gemacht werden.

Rekursion

Ressourcen

- `Recursion.java`
- [W3Schools](#): Seite "Java Recursion".

Aufgabenstellungen

Für die folgenden Probleme gäbe es grundsätzlich auch iterative (d.h. schleifenbasierte) Lösungen. Sie sollen aber allesamt mit Rekursion gelöst werden - wenn möglich ohne Hilfsmethoden.

Falls nichts Anderes erwähnt, soll das Resultat zurückgegeben werden.

h) Euklidischer Algorithmus: Damit kann von zwei natürlichen Zahlen `x` und `y` der ggT berechnet werden:

`ggT(24, 81):`

`81 / 24 -> Rest 9`

`24 / 9 -> Rest 6`

`9 / 6 -> Rest 3 <= ggT`

`6 / 3 -> Rest 0`

=> der letzte Rest ungleich 0 ist der ggT

Programmiere eine rekursive Methode `gcd()` (greatest common divisor), die diesem Algorithmus folgt.

Achtung: Wie viel Code brauchst du wirklich für die Fallunterscheidung, welches das grössere Argument ist?

i) Spiegeln einer Zeichenfolge: Schreibe ein Programm `verlan()`, das eine Zeichenfolge invertiert.

Tipps:

- Für einen String `s` kann mit `char c = s.charAt(n)` das Zeichen an der `n`-ten Stelle ausgelesen werden. Zeichen vom Typ `char` können danach ganz normal mit Hilfe von `+` mit `Strings` verbunden werden.
- Benutze `.substring()`
- Es soll kein Array gebraucht werden.

j) Türme von Hanoi: Mache dich mit dem Problem vertraut und entwickle eine Methode `move(height, fromPeg, toPeg, otherPeg)`, die Schritt für Schritt ausgibt, welche Scheibe von welchem Stab auf welchen verschoben wird.

Zusatzaufgabe: Stelle jeden Schritt graphisch auf der Konsole dar.

*k) Implementiere ein Sortierverfahren deiner Wahl rekursiv. Für welche Verfahren ist das überhaupt sinnvoll möglich?