

A Rekursion

- Löse alle Beispiel rekursiv. Nicht rekursive Lösungen erzielen 0 Punkte.
- Repository-Name: **sew3_sem1**, Package: ue04_Rekursion.
- Wähle für die Java-Dateien sinnvolle Namen wie z.B.: SierpinskiTriangle, EightQueens, ProjectEuler_031_CoinSums, ...

A.1 Sierpinski Dreieck

Zeichne das Sierpinski-Dreieck (`DrawSierpinski.java`).

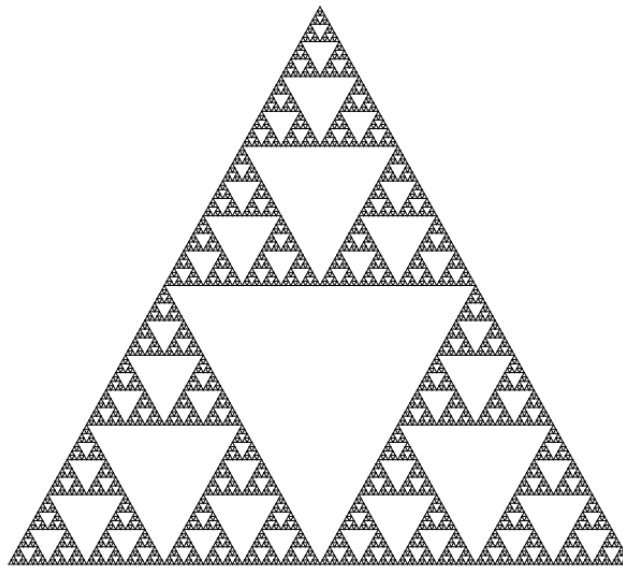


Abbildung 1: Sierpinski-Dreieck (siehe auch: <https://de.wikipedia.org/wiki/Sierpinski-Dreieck>)

- Gehe von der Java-Datei »**DemoSimpleDrawing.java**« aus.
- Die eigentliche rekursive Zeichen-Methode soll »**void drawSierpinskiTriangle**« heißen und die folgenden Parameter haben:
 - `GraphicsContext gc`: die Zeichenfläche
 - `int size`: Abbruchkriterium (Anzahl der Pixel der kürzesten Kantenlänge, die noch gezeichnet werden soll).
 - Koordinaten für die 3 Eckpunkte (`double`) des großen Dreiecks
- das *große* Dreieck besteht aus drei *kleineren* Dreiecken – diese kann man durch einen **rekursiven** Aufruf zeichnen.
 - Wichtig: Die kürzeste Seite, die noch gezeichnet wird, darf die Länge `size` nicht unterschreiten.
- Beispiel: Die Seitenlänge des großen Dreiecks soll 580 Pixel sein.

Hinweise:

- das Dreieck kann auch *schief* sein
- es hat schon *jemand* ein Klasse für Punkte geschrieben (`Point2D`, Teil von JavaFX)

A.2 Damenproblem¹

Dateiname: `EightQueens.java`

¹<https://de.wikipedia.org/wiki/Damenproblem>

Schreibe ausgehend vom Acht-Türmeproblem ein Programm, das alle Lösungen ermittelt, wie 8 Damen auf einem Schachbrett so aufgestellt werden können, dass sie sich nicht gegenseitig schlagen können.

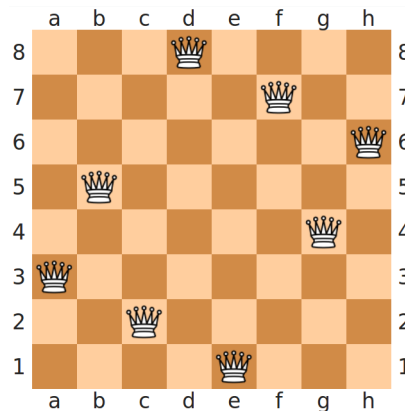


Abbildung 2: eine Lösung des Damenproblems (Quelle: <https://de.wikipedia.org/wiki/Damenproblem>)

- Miss die Zeit in Millisekunden, die dein Programm zum Lösen braucht.
- Gestalte deine Lösung so flexibel, dass die Dimension des Schachbretts frei gewählt werden kann (z.B. ein 10x10 Schachbrett).
- Teste nun beginnend beim 1x1 Schachbrett bis zu welcher Dimension dein Programm die Anzahl der Lösungen in endlicher Zeit findet (schneller als 20 Minuten).
- **Bonusaufgabe 1:** Bestimme die Anzahl der eindeutigen Lösungen. D.h. ohne Drehungen und Spiegelungen.
- **Bonusaufgabe 2:** Löse das Superdamenproblem und/oder das Springerproblem.

A.3 Projekt Euler: Coin sums (Problem 31²)

Dateiname: Euler.java

In England the currency is made up of pound, £, and pence, p, and there are eight coins in general circulation:

1p, 2p, 5p, 10p, 20p, 50p, £1 (100p) and £2 (200p).

It is possible to make £2 in the following way:

$1 \times £1 + 1 \times 50p + 2 \times 20p + 1 \times 5p + 1 \times 2p + 3 \times 1p$

How many different ways can £2 be made using any number of coins?

A.4 Projekt Euler: Longest Collatz sequence (Problem 14³)

The following iterative sequence is defined for the set of positive integers:

$n \rightarrow n/2$ (n is even) $n \rightarrow 3n + 1$ (n is odd)

Using the rule above and starting with 13, we generate the following sequence:

$13 \rightarrow 40 \rightarrow 20 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$

It can be seen that this sequence (starting at 13 and finishing at 1) contains 10 terms. Although it has not been proved yet (Collatz Problem), it is thought that all starting numbers finish at 1.

²<https://projecteuler.net/problem=31>

³<https://projecteuler.net/problem=14>

Which starting number, under one million, produces the longest chain?

NOTE: Once the chain starts the terms are allowed to go above one million.

A.5 Zahl Hoch

Dateiname: `Power.java`

Viele(9) Multiplikationen:

$$2^{10} = 2 * 2 * \dots$$

Oder: durch geschicktes Zusammenfassen – 4 Multiplikationen

$$\begin{aligned} 2^{10} &= 2^5 * 2^5 \\ 2^5 &= 2 * 2^2 * 2^2 \\ 2^2 &= 2^1 * 2^1 \\ 2^1 &= 2 \end{aligned}$$

Schreibe die Methode `zahlHoch(int x, int n)` die (rekursiv) nach der zweiten Methode rechnet.

Löse: wenn man im Jahre 0 einen Euro mit 1% verzinst angelegt hätte – wie viel wäre das heute.

A.6 Sudoku

Dateiname: `Sudoku.java`

Das Programm soll in der Lage sein:

1. Sudokus aus einer Datei zu lesen
2. sie zu lösen,
3. die Lösung zu speichern
4. die Sudokus (in einem für den Menschen sinnvollen Darstellung) anzuzeigen.
5. in einem Ordner Lösungen für alle Sudoku-Files erzeugen

A.6.1 Sudoku aus einer Datei lesen

Lies eine Datei ein und speichere die Sudoku-Angaben sinnvoll ab.

In einer Datei können mehrere Sudokus stehen. Beispiel die Datei »easy.sudoku«:

```
easy_01: .....71..15....7..39...8.5..2.5...8.6...291.....1.6.....2.6.....192.....53
easy_02: .12..64...9.....4.7..2..1.....427.59.....6.3....9745.....3.....6..4.59
easy_03: .....3.5...8.2...84..7....2...34...6..8...4.3.5.97.62.5.....4..3..9...7...5.
```

- In jeder Zeile steht genau ein Sudoku.
- Zuerst steht der Name des Sudokus, danach das eigentliche Sudoku mit genau 81 Zeichen (den Ziffern 1-9 und dem Punkt)
- Ein Punkt steht für ein leeres Feld.
- Je neun Zeichen bilden eine Zeile. Die Zeilen sind nicht voneinander getrennt.

Damit ergibt sich für das Sudoku »easy_01«:

```
+---+---+---+
|   | 7|1 |
|15 |   |7 |
|39 |   |8 5|
+---+---+---+
| 2| 5 | 8|
```

```

| 6 | 2|91 |
|  | 1| 6 |
+---+---+---+
|  | 2 |6 |
|  |19|2 |
|  |  |53|
+---+---+---+

```

A.6.2 Löse alle eingelesenen Sudokus

- Bearbeite nun der Reihe nach alle Sudokus:
 - Gib vor dem Lösen den Namen und das ungelöste Sudoku wie in der vorigen Grafik in die Konsole aus.
 - Löse das Sudoku.
 - Gib die Lösung in die Konsole aus.
- Speicher nun alle Lösungen gemeinsam in der Datei »sudoku.solved«.
 - Dabei sollen die Lösungen an die Datei angehängt werden (append!).
 - Jedes Sudoku steht in einer eigenen Zeile mit dem Format: SudokuName;angabe;Lösung

A.6.3 Lösen aller Sudokus in einem Ordner

Löse nun alle Sudokus in einem beliebigen Ordner und speichere alle Lösungen gemeinsam »sudoku.solved«.

Tipp:

```

for (Path path: Files.newDirectoryStream(Paths.get("resources/ue09_Sudokus"), "*.sudoku")) {
    BufferedReader in = Files.newBufferedReader(path);
    // Beten, lesen, lösen, appenden
}

```