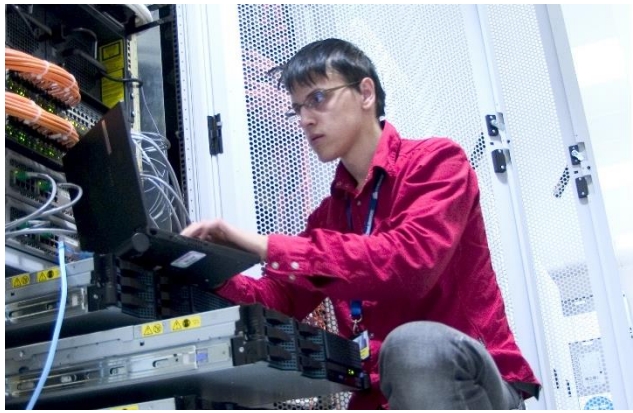
EMIL-VON-BEHRING GYMNASIUM

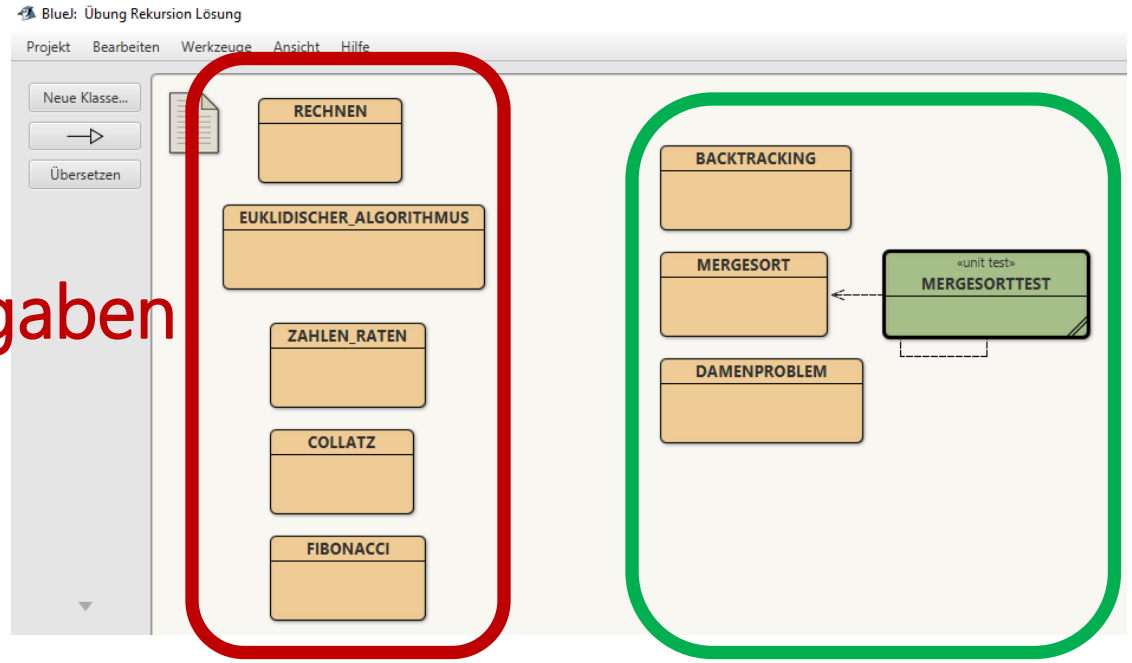
Übungen zur Rekursion



Rechnen mit Rekursion

- Öffne das BlueJ-Projekt „Übungen zur Rekursion“.

Pflichtaufgaben für alle



Optionale Zusatzaufgabe für ganz Schnelle

- Setze die Rechenoperation der Potenz und der Multiplikation mithilfe von Rekursion um! (Aufgabenstellung in der Klasse RECHNEN)

Euklidischer Algorithmus

- Der Euklidischer Algorithmus berechnet den größten gemeinsamen Teiler (GGT) zweier Zahlen. Hierbei gibt es zwei Varianten:
- Implementiere die Methode **einfacherEuklid(int a, int b) : int**, welcher zwei Zahlen a und b übergeben bekommt und den ggT dieser Zahlen zurückgibt.
- Algorithmus Variante 1:
  - Falls beide Zahlen a und b gleich groß sind, ist das der ggT gefunden.
  - Ansonsten ziehe die kleinere der beiden Zahlen von der größeren ab.
  - Wiederhole die Schritte bis a und b gleich groß sind.

a	b
24	56
24	32
24	8
16	8
8	8

Euklidischer Algorithmus mit Modulo %

- Implementiere die Methode **euklid(int a, int b) : int**, welcher zwei Zahlen a und b übergeben bekommt und den ggT dieser Zahlen zurückgibt.
- Variante 2:
  - Falls b = 0 ist, dann ist a der ggT von den ursprünglichen Zahlen a und b.
  - a = b und b ist der Divisionsrest von a durch b -> Wiederholung
- Funktionsweise:

a		b	a%b
1071	= 1 ·	1029	+ 42
1029	= 24 ·	42	+ 21
42	= 2 ·	21	+ 0
21	=	0	<- Abbruch

ggT

Diese Variante des Euklidischen Algorithmus ist schneller, weil anstatt sukzessiv 24-mal die 42 von der 1029 abzuziehen, benötigt man hier nur eine Divisionsrestrechnung!

Zahlen raten

- Denk dir eine Zahl zwischen z. B. 1 und 1000 und der Algorithmus soll systematisch deine Zahl erraten!
- Dies kann folgendermaßen aussehen:
- Wie geht dieser Algorithmus dabei vor?
- Implementiere die Methode **rate(int anfang, int ende) : int**

```
Ist Ihre Zahl groesser als 500? 0:ja, 1: nein
1
Ist Ihre Zahl groesser als 250? 0:ja, 1: nein
0
Ist Ihre Zahl groesser als 375? 0:ja, 1: nein
1
Ist Ihre Zahl groesser als 313? 0:ja, 1: nein
0
Ist Ihre Zahl groesser als 344? 0:ja, 1: nein
1
Ist Ihre Zahl groesser als 329? 0:ja, 1: nein
0
Ist Ihre Zahl groesser als 337? 0:ja, 1: nein
1
Ist Ihre Zahl groesser als 333? 0:ja, 1: nein
1
Ist Ihre Zahl groesser als 331? 0:ja, 1: nein
0
Ist Ihre Zahl groesser als 332? 0:ja, 1: nein
0
Deine gedachte Zahl ist 333
```

Can only enter input while your programming is running

Collatz-Problem

- Das Collatz-Problem beschreibt ein ungelöstes Problem aus der Mathematik: https://de.wikipedia.org/wiki/Collatz-Problem
- Implementiere die Methode **collatz(int n, int i) : int** gemäß der Beschreibung auf Wikipedia. Da die Collatz-Zahlenfolge unendlich lang ist, soll der i-te Wert der Folge zurückgegeben werden.
- Startzahl: n = 19

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	...
n	19	58	29	88	44	22	11	34	17	52	26	13	40	20	10	5	16	8	4	2	1	4	2	1	...

- Du kannst dir natürlich auch per System.out jede Zahl der Folge auf der Konsole ausgeben lassen.

Fibonacci-Folge

- Eine besondere Zahlenfolge in Informatik ist die Fibonacci-Folge https://de.wikipedia.org/wiki/Fibonacci-Folge
- Implementiere die Methode **fibonacci(int i) : int**, wobei die i-te Fibonacci-Zahl folgendermaßen berechnet wird:
  fibonacci(i) = fibonacci (i-1) + fibonacci (i-2)
  und fibonacci(0) = 0 und fibonacci(1) = 1 ist.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	…
	0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	…

  - Optional: Da die Folge sehr schnell ansteigt, kann bereits die 47. Zahl der Folge nicht mehr im Datentyp int gespeichert werden. Nutze die Java-Klasse BigInteger für beliebig große Zahlen.
    https://docs.oracle.com/javase/8/docs/api/java/math/BigInteger.html

Fibonacci-Folge

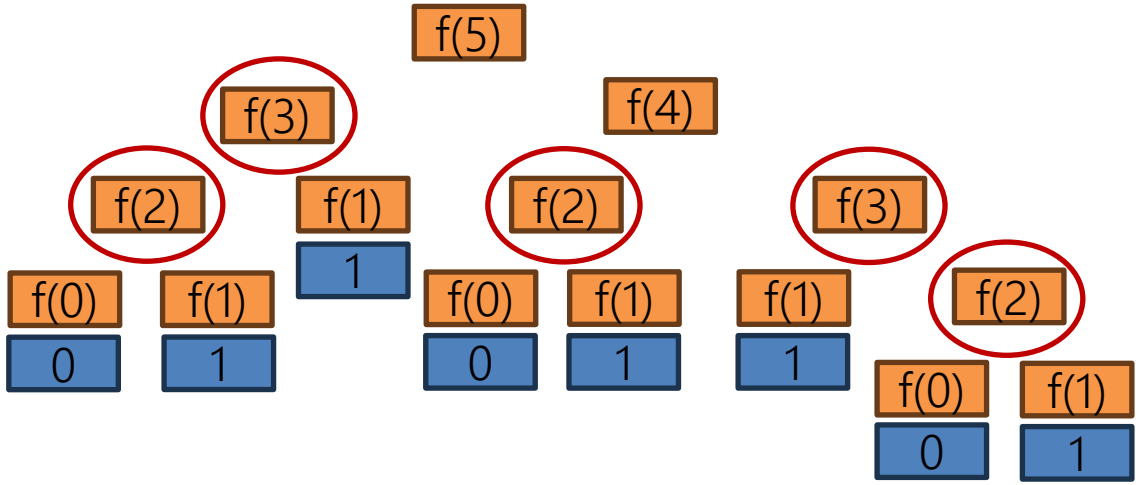
- Lass dir die 46. Fibonacci-Zahl (oder ggf. höher) berechnen.
- Warum dauert das so lange?

- Implementiere die Methode **fibonaccilterativ(int i) : int**, indem du die Fibonacci-Folge mithilfe einer for-Schleife berechnest.
  Tipp: Beginne „unten“ und nutze eine Hilfsvariable.

- Lass dir die 46. Fibonacci-Zahl (oder ggf. höher) **iterativ** berechnen.
- Warum geht das jetzt so schnell?

Fibonacci-Folge rekursiv vs. iterativ

- Iterativ: 46-mal zwei Zahlen addieren, schafft der Computer in Bruchteilen von Sekunden.
- Was ist also bei der rekursiven Variante anders?



f(5) = 0+1+1+0+1+1+0+1 = 5

- Wiederholtes Berechnen desselben Wertes notwendig!

→ Ungünstig für die Laufzeit des Algorithmus (Inf 13.3 Grenzen der Berechenbarkeit)

- Wie aufwändig wird es wohl sein, wenn wir f(25) berechnen wollen?
- Idee für Abhilfe?

Fibonacci-Folge – Dynamisches Programmieren

- Problem: Mehrfachberechnung von Teillösungen → Lösung: Teilergebnisse merken und wiederverwenden!
- Vorteil: Verbesserung der Laufzeit-Effizienz
- Nachteil: mehr Speicherplatz benötigt
- Wie machen wir das jetzt für unsere Fibonacci-Zahlen, wenn wir fibDP(n) berechnen wollen?
  - Array der Größe n+1 **als Attribut** anlegen
  - Basisfälle eintragen
  - Eigentliche Methode zur Berechnung der Fibonacci-Zahlen erstellen
    - Prüfen, ob an Arrayposition der zu berechnende Zahl bereits ein Eintrag steht, dann diesen zurückgeben
    - Wenn nicht, abspeichern des hier berechneten Wertes & rekursiver Aufruf der Fibonacci-Berechnung

Rekursiv vs. Iterativ im Vergleich

- Der iterative Algorithmus ist trotzdem schneller, weil dieser weniger Lese-/Schreibe-Operationen benötigt.
- Merkregel: Die rekursive Variante eines Algorithmus höchstens so schnell wie die iterative Variante eines Algorithmus.
- ABER: Häufig ist die rekursive Variante leichter zu programmieren, wenn die Schwierigkeit des Problems steigt (auch wenn sich das vielleicht für euch im Moment nicht so anfühlt, weil Rekursion neu für euch ist).
- ZUDEM gibt es Algorithmen zu Problemen bei denen man die iterative Variante nur sehr schwer oder gar nicht findet!
  -> siehe optionale Aufgaben (selbstähnliche Figuren, Backtracking)

Selbstähnliche Figuren (in Snap!)

- Öffne Snap! https://snap.berkeley.edu/snap/snap.html
- Öffne Einführung selbstähnliche Figuren und setze dies in Snap um!
- Tipps:
  - Startroutine wie rechts im Bild ->
  - Startwert von 150 ist besser, damit es auf der Zeichenfläche bleibt
  - Der Rekursionsabbruch ist leer, sprich es soll dann einfach nichts gemacht werden
  - Um die Grundfigur zu testen, kann man den Rekursionsabbruch höher wählen z. B. > 100 anstatt >2

- Weitere Übungen zu selbstähnlichen Figuren

Backtracking

- Backtracking (Rücksetzverfahren):
  - Problemlösungsverfahren
  - systematisches Durchsuchen des Suchraums
  - dabei Anwendung von trial-and-error (Versuch-Und-Irrtum)
- Falls für eine „Entscheidung“ mehrere Möglichkeiten existieren:
  - alle Möglichkeiten rekursiv durchprobieren (**rekursiver Aufruf in if-Bedingung bei return**)
    - falls eine Möglichkeit zum Erfolg führt: gut
      - Suche kann abgebrochen werden (außer man will alle Lösungen finden)
    - andernfalls:
      - Entscheidung war wohl falsch…
      - Entscheidung „rückgängig machen“

→ Backtracking findet garantiert eine Lösung, wenn überhaupt eine existiert

- Beispiel für Backtracking: Tiefendurchlauf eines Graphen

Tiefendurchlauf bzw. Tiefensuche im Graphen

- → Siehe Präsentation zum Tiefendurchlauf bzw. Tiefensuche

- → Nachfolgend sind weitere optionale Übungen zur Rekursion, die noch mehr Herausforderung bieten!

Übung zu Backtracking (optional)

- Folgende Methode soll das Feld a (garantiert der Länge 2n und beim ersten Aufruf von außen mit 0 initialisiert) mittels rekursivem Backtracking so mit Zahlen 1 <= x <= n befüllen, dass jedes x genau zweimal in a vorkommt und der Abstand zwischen den Vorkommen genau x ist. Sie soll genau dann true zurückgeben, wenn es eine Lösung gibt. Beispiele:
  - n = 2 → false
  - n = 3 → [3,1,2,1,3,2]
  - n = 4 → [4,1,3,1,2,4,3,2]
  - n = 5 → false
  - n = 6 → false
  - n = 7 → [7,3,6,2,5,3,2,4,7,6,5,1,4,1]
  - n = 8 → [8,3,7,2,6,3,2,4,5,8,7,6,4,1,5,1]
  - n = 9 → false
  - n = 10 → false

How to: nächste Folie

Übung zu Backtracking (optional)

- Wie kommt es zu der Lösung? n = 4 → [4, 1, 3, 1, 2, 4, 3, 2]
- Zahl 4 gesetzt auf Position 0 und 5 → [4,0,0,0,0,4,0,0]
- Zahl 3 auf Position 0 und 4 ↯
- Zahl 3 auf Position 1 und 5 ↯
- Zahl 3 auf Position 2 und 6 → [4,0,3,0,0,4,3,0]
- Zahl 2 auf Position 0 und 3 ↯
- Zahl 2 auf Position 1 und 4 → [4,2,3,0,2,4,3,0]
- Zahl 1 auf Position 0 und 2 ↯
- Zahl 1 auf Position 1 und 3 ↯
- Zahl 1 auf Position 2 und 4 ↯

- Zahl 1 auf Position 3 und 5 ↯
- Zahl 1 auf Position 4 und 6 ↯
- Zahl 1 auf Position 5 und 7 ↯ ->Position 7 ist der letzte Platz, damit keine Möglichkeit gefunden
- Backtracking Zahl 2 zurückgenommen → [4,0,3,0,0,4,3,0]
- Zahl 2 auf Position 2 und 5 ↯
- Zahl 2 auf Position 3 und 6 ↯
- Zahl 2 auf Position 4 und 7 → [4,0,3,0,2,4,3,2]
- Zahl 1 auf Position 0 und 2 ↯
- Zahl 1 auf Position 1 und 3 → [4,1,3,1,2,4,3,2]
- Lösung gefunden, wenn alle gesetzt sind

MergeSort (optional)

Das Gif sind verlinkt, einfach anklicken!→

- Implementiere ein **rekursives** MergeSort, welches ein übergebenes Zahlen-Array sortiert:
  - Implementiere zunächst die Methode merge(int[] left, int[] right, int[] arr):
    - Diese Methode soll die beiden Arrays left und right zum einem Array arr zusammenfügen. Du kannst davon ausgehen, dass die beiden Arrays left und right bereits sortiert sind und zusammen die gleiche Größe haben wie das Array arr. Somit müssen die Zahlen von left und right nur in das Array arr übertragen werden.
    - Tipp: Nutze zwei Hilfsvariablen, die sich merken, an welcher Stelle, man gerade in left bzw. right ist.
  - Implementiere die Methode mergeSort(int[] arr): Falls der Rekursionsabbruchs-fall nicht greift, soll das Array arr in zwei Arrays left und right mittig aufgeteilt werden (Nutze hierfür die Methode Arrays.copyOfRange(…) Siehe Java API) und ebenfalls mit mergeSort(…) sortiert werden. Anschließend sollen die beiden Arrays left und right mittels merge(…) wieder im Array arr zusammen-gefügt werden.

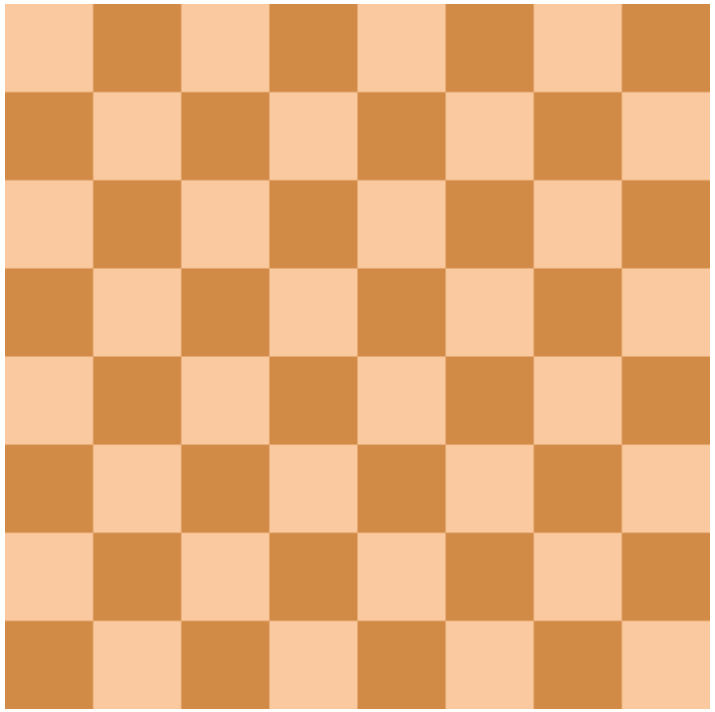
Weiteres Beispiel: Das Damenproblem (optional)

- Das Gif sind verlinkt, einfach anklicken!

- Acht Damen sollen auf einem Schachbrett so aufgestellt werden, dass keine zwei Damen einander gemäß ihren in den Schachregeln definierten Zugmöglichkeiten schlagen können. Für Damen heißt dies konkret: Es dürfen keine zwei Damen auf derselben Reihe, Linie oder Diagonale stehen.
- 92 mögliche Lösungen für das 8x8 Feld – aber wie findet man diese?
- Idee des Backtracking:
  - Algorithmus so lange ausführen, bis man an eine Grenze kommt (hier: es kann keine Dame mehr aufgestellt werden ohne Regeln zu verletzen)
  - Wenn das der Fall ist: zurück zum letzten Schritt, anderen Folgeschritt testen

  → Versuchen, gültige Teillösung zu finden, auf diesem restlichen Weg zum Ziel aufbauen

  → Wenn das nicht möglich ist, versuchen andere Teillösung zu finden



Zentral: Zurücksetzen von bereits getätigten Schritten

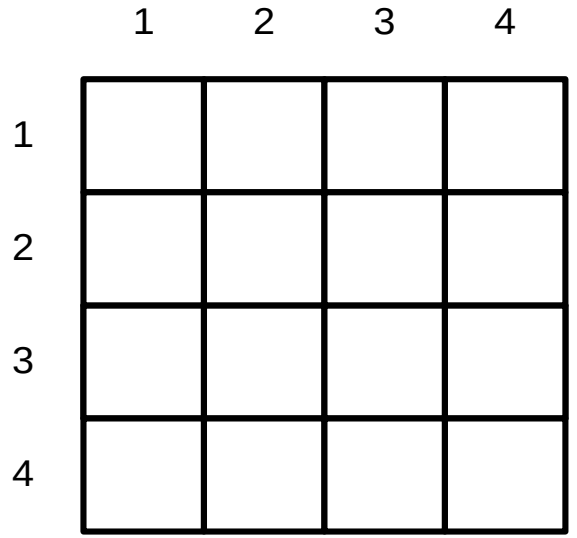
Wie funktioniert das konkret beim Damenproblem?

- Wir reduzieren erstmal auf ein kleineres Feld…

→ 4 Damen auf dem 4x4-Feld unterbringen, sodass sie sich nicht gegenseitig schlagen können

Vorgehen:

- für jede Zeile führe aus:
  - prüfe ob Feld von bereits gesetzter Dame bedroht ist
  - falls Feld nicht bedroht ist
    - setze Dame dort hin
    - setze Bedrohungen, die von gesetzter Dame ausgehen
    - falls letzte Spalte erreicht
      - gib die Lösung aus
    - falls letzte Spalte noch nicht erreicht
      - Führe Vorgehen erneut in der nächsten Spalte durch
    - zum Schluss lösche die Bedrohung und die Dame



Wie funktioniert das konkret beim Damenproblem?

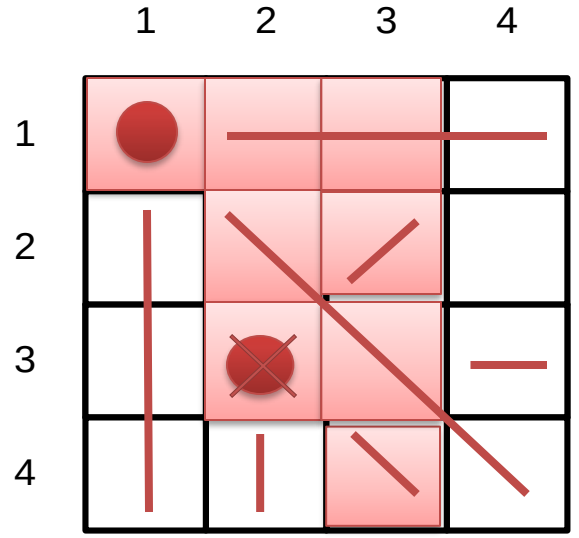
- Wir reduzieren erstmal auf ein kleineres Feld…

→ 4 Damen auf dem 4x4-Feld unterbringen, sodass sie sich nicht gegenseitig schlagen können

Vorgehen:

- für jede Zeile führe aus:
  - prüfe ob Feld von bereits gesetzter Dame bedroht ist
  - falls Feld nicht bedroht ist
    - setze Dame dort hin
    - setze Bedrohungen, die von gesetzter Dame ausgehen
    - falls letzte Spalte erreicht
      - gib die Lösung aus
    - falls letzte Spalte noch nicht erreicht
      - Führe Vorgehen erneut in der nächsten Spalte durch
  - zum Schluss lösche die Bedrohung und die Dame

Fortsetzen des vorherigen Aufrufs des Vorgehens



usw.… bis Lösung gefunden wurde, dann wird sie graphisch ausgegeben

Nochmal zum Nachvollziehen: https://www.youtube.com/watch?v=B3Vr1r3J8iI

Übungsaufgabe N-Damenproblem (optional)

- Implementiere das N-Damenproblem:
  - Nutze hierfür das Backtracking-Algorithmusmuster ähnlich zu der Aufgabe auf der vorherigen Folie.
  - Deinem Programm sollte ein Wert übergeben werden können, sodass man die „Größe des Schachfelds“ und damit auch die Anzahl der Damen bestimmen kann.
  - Tipp: Es bietet sich in jedem Fall an eine Methode zu schreiben die überprüft, ob eine Dame auf einem gewissen Feld stehen darf.
  - Tipp: Schreibe eine Methode den aktuellen Stand und/oder das Ergebnis auf der Konsole entsprechend dem Schachmuster ausgibt. Dies erleichtert das Verifizieren der Lösung und das Debugging.
  - Beispielausgabe für das 4-Damenproblem:

```
0 1 0 0
0 0 0 1
1 0 0 0
0 0 1 0
```