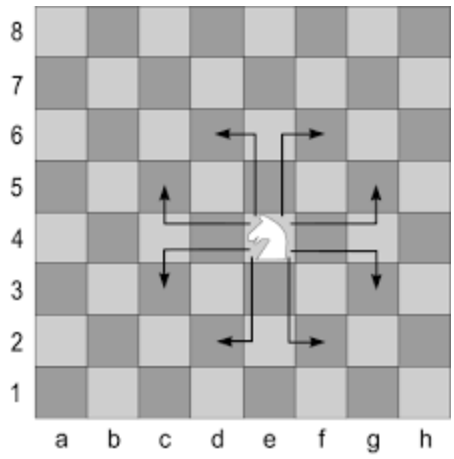
Springer-Problem
"gelöst mit Backtracking und Rekursion"

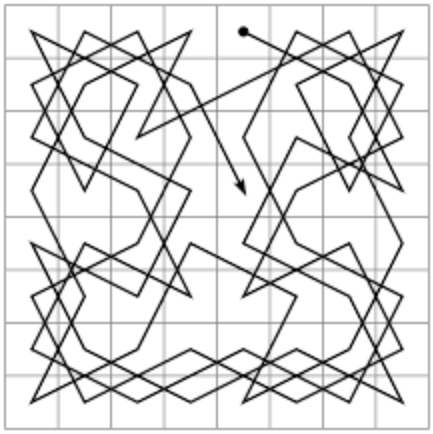
Einleitung:

Spielen Sie Schach?

Neben dem eigentlichen Spiel gibt es viele Knobel-Aufgaben, die sich dem Schachbrett und den Schachfiguren und deren Möglichkeiten bedienen.

So auch die Knobel-Aufgabe mit der Spielfigur Springer.

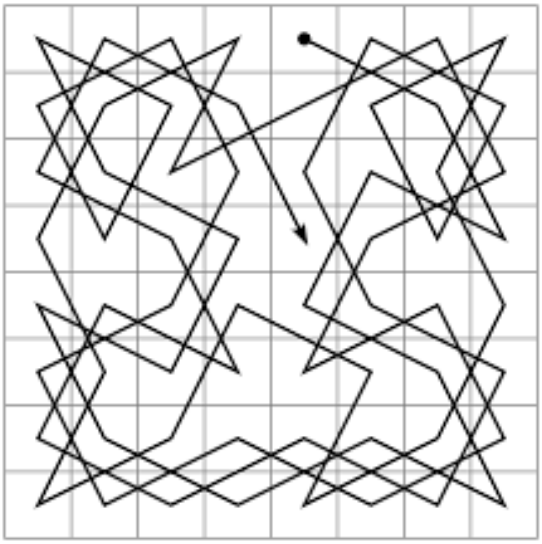




Es muss eine Applikation geschrieben werden, welche für einen Springer auf einem Schachbrett einen Weg findet, der alle Felder genau einmal betritt. Da der Springer ja eben springen kann, zählt jeweils nur das Zielfeld eines Zuges als betreten. (siehe oben)

Dieses Problem kann mit Rekursion und Backtracking gelöst werden.

Ein Bild aus einer Aufgabe:



Ein Bild von meiner Lösung (nur 60 Züge)

```
Springer Problem

 * 54 15 58  1 40  *  *
 * 57 24 41 26 59 32 39
53 14 55 16 33  2 27 60
56 23 52 25 42 17 38 31
13 48 11 34  9 30  3 28
22 51 20 45 18 43  6 37
47 12 49 10 35  8 29  4
50 21 46 19 44  5 36  7
```

Hier das ist Problem ausführlich beschrieben: *DasSpringerProblem_RW_Trier_2013_p1-6.pdf*

Idee:

Im Zentrum steht eine rekursive Methode:

```
/* @param x    x-Koordinate auf dem Spielfeld
 * @param y    y-Koordinate auf dem Spielfeld
 * @param move  Hält fest, der wievielte Sprung es ist.
 *              Der erste Zug wird mit 1 starten.
 *              Hat move 64 erreicht, so sind alle Felder abgedeckt.
 *
 * @return     true, wenn Feld setzen ging oder es das letzte Feld ist.
 */
boolean jump(int x, int y, int move)
```

1. Zu Beginn wird die Methode jump mit *beliebigen Startpositionen für x und y* und dem *move 1* aufgerufen.
2. Die Methode prüft, ob die Werte von x und y sich noch auf dem Spielfeld befinden.
   Falls nicht → return false;
3. Dann wird geprüft, ob das Feld selber nicht schon zuvor betreten wurden.
   Ein freies Feld hat den Wert 0.
   Ein bereits belegtes Feld hat einen Wert != 0
   Wurde das Feld bereits zuvor betreten → return false
4. Dann wird geprüft, ob dies das letzte Feld ist
   Das ist dann der Fall, wenn der *move* das gewünschte Maximum erreicht.
   Falls ja → *move* auf das Feld schreiben, damit man im Spielfeld sieht, wo welcher Zug endet und return true

   ```
   board[x][y] = move;
   ```

   ACHTUNG:
   Bei einem 8x8 Feld wäre das Maximum 64.
   Doch Achtung, die Rechenzeit ist dann riesig grosse.
   Es macht bei ersten Tests durchaus Sinn, ein 8x8 Feld zu haben aber **nur die ersten 20 Züge** rechnen zu lassen.
   Also als maximalen Move nicht 64 sondern zBsp erstmal 20 definieren.

   ```
   public int MAX_MOVES = 2;
   ```

5. So und wenn das alles nichtzutreffend war, dann muss man
   a. An diesem Feld den move eintragen
      ```
      board[x][y] = move;
      ```
   b. Die Möglichkeiten für Folge-Felder prüfen
      → Es gibt acht unterschiedliche Möglichkeiten
      → Für die erste Möglichkeit *jump* aufrufen, wobei der move eins höher sein muss.
      War das erfolgreich → return true
      War es nicht erfolgreich… die nächste Möglichkeit aufrufen.
      und so weiter

      War keine Möglichkeit erfolgreich, dann geht das aktuelle Feld nicht.
      Backtracking machen…
      → Feld wieder freigeben
      ```
      board[i][j] = FIELD_FREE;
      ```
      und return false

Es folgen Tipps für die Implementierung:

Ein paar Tipps zur Implementierung

Schritt 1:

- Starten Sie mit dem leeren **Spielbrett** board und lassen Sie es zeichnen
  Das board ist das Attribut in der Klasse SpringerProblem:

  private int[][] board
  → zweidimensionalen Array von int

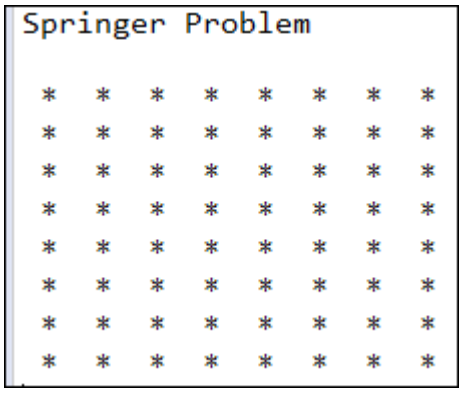
  Es werden später zweistellige Zahlen in das Feld geschrieben, deshalb habe ich ein formatierte Ausgabe verwendet.

  System.out.printf("%2d ", solver.getBoard()[i][j]);

- Die Elemente des Arrays können die Werte FIELD_FREE später die Nummer des Zuges (move) haben.

  ```
  private static final int FIELD_FREE = 0;      //Platz frei
  ```

  Alle Felder des Boards werden mit ***FIELD_FREE*** initialisiert.

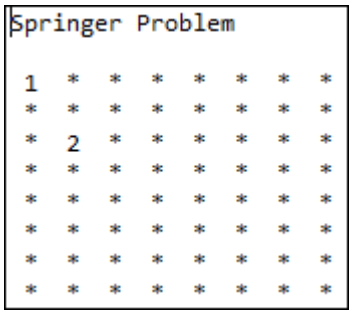


Schritt 2:

Schreiben Sie nun die rekursive Methode *jump*, die noch ungeprüft einen Springer setzt.
Vorerst nur mal zwei Züge. (Solange move <=2)

```
private int MAX_MOVES = 2;
```

Das sieht dann zBsp für die ersten beiden Sprünge so aus, wenn ich auf dem Feld 0 0 starte.




```
16          System.out.println("Springer Problem");
17          int startX = 0;
18          int startY = 0;
19          System.out.println();
20
21          if (solver.jump(startX, startY, 1))
```

Schritt 2:

Bauen Sie jump gemäss der Beschreibung auf der Seite 2 weiter aus.

ACHTUNG:

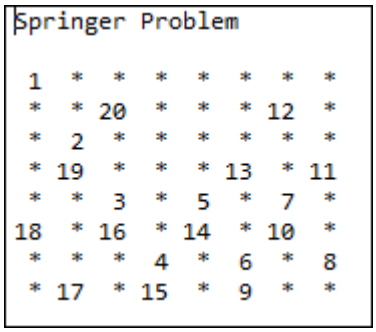
Bei einem 8x8 Feld wäre das Maximum 64.
Doch Achtung, die Rechenzeit ist dann riesig gross.

Es macht bei ersten Tests durchaus Sinn, ein 8x8 Feld zu haben aber nur die ersten 20 Züge rechnen zu lassen.

Also jump mal zu Beginn nur mit *max-move = 20* laufen lassen!!

Mein Rechner hat es innert nützlicher first knapp auf *max-move = 60* geschafft.
Den *max-move = 64* hat der Computer auch nach etlichen Minuten nicht gelöst.

Hier sehen Sie wie die ersten 20 Züge bei mir aussehen:



Und hier 40 Züge… das Feld wird langsam voller