Jui-Hung Lu
933293709
Reference:
http://kingxss.iteye.com/blog/2290026
https://en.wikipedia.org/wiki/Eight_queens_puzzle
https://www.semanticscholar.org/paper/Isomorphism-and-the-N-Queens-problem-Cull-Pandey/a468a6a887bc6b96fa576052a14bcfd8c5851875

```
package hw6;

import java.util.HashMap;
import java.util.Map;

public class hw6 {
    private Integer queens;
    // check if any queen in the same col, 1 means yes
    private Integer[] column;
    // check if there is queen in the right skew
    private Integer[] rup;
    // check if there is queen in the left skew
    private Integer[] lup;
    //answer
    private Integer[] queen;
    //Isomorphism
    private Map<String, String> results = new HashMap<String,
String>();
    // the index of Independence
    public static int num;

    public hw6(int queens) {
        this.queens = queens;
        column = new Integer[queens + 1];
        rup = new Integer[(2 * queens) + 1];
        lup = new Integer[(2 * queens) + 1];
        queen = new Integer[queens + 1];

        for (int i = 0; i <= queens; i++) {
            column[i] = queen[i] = 0;
        }

        for (int i = 0; i <= (2 * queens); i++) {
            rup[i] = lup[i] = 0; //initial
        }
```

```
    }

    public void backtrack(int i) {  //using backtrack to find the
normal answer
        if (i > queens) {
            showAnswer();
        } else {
            for (int j = 1; j <= queens; j++) {
                if ((column[j] == 0) && (rup[i + j] == 0) && (lup[i -
j + queens] == 0)) {
                    queen[i] = j;
                    column[j] = rup[i + j] = lup[i - j + queens] = 1;
                      //guess from [1,1], then find the appropriate
                point in row 2, and so on.
                    backtrack(i + 1);
                    column[j] = rup[i + j] = lup[i - j + queens] = 0;
                    // If there is no appropriate point, then return to
                previous level and find another point to search.
                }
            }
        }
    }

    protected void showAnswer() {  //check answer first, if it didn't
show before, print it and store it

        if(!isIndependence(num)) return;
        num++;
        System.out.println("answer" + num + ":");
        for (int y = 1; y <= queens; y++) {
            for (int x = 1; x <= queens; x++) {
                if (queen[y] == x) {
                    System.out.print(" Q");
                } else {
                    System.out.print(" .");
                }
            }
            System.out.println(" ");
        }
        System.out.println();
    }
```

```
    protected boolean isIndependence(int number) {  //check its
rotation and symmetric if show before
        String newSolution = resultToString(queen);
        String flag = results.get(newSolution);

        if (flag != null) {
            return false;
        }

        // check symmetric - left & right
        Integer[] leftRight = new Integer[queen.length];
        // check symmetric - up & down
        Integer[] upDown = new Integer[queen.length];
        // check symmetric - up left - down right
        Integer[] lurd = new Integer[queen.length];
        // check symmetric - up right - down left
        Integer[] ruld = new Integer[queen.length];
        // first  rotation
        Integer[] cw1 = new Integer[queen.length];
        for (int i = 1; i < queen.length; i++) {
            leftRight[i] = queen[queen.length - i];
            upDown[i] = queen.length - queen[i];
            lurd[queen.length - queen[i]] = queen.length - i;
            ruld[queen[i]] = i;
            cw1[queen[i]] = queen.length - i;
        }
        // second  rotation
        Integer[] cw2 = new Integer[queen.length];
        for (int i = 1; i < queen.length; i++) {
            cw2[cw1[i]] = queen.length - i;
        }
        // third  rotation
        Integer[] cw3 = new Integer[queen.length];
        for (int i = 1; i < queen.length; i++) {
            cw3[cw2[i]] = queen.length - i;
        }
        // compare to other 7 solutions RS, RRS, RRRS, MRRRS, RMRRRS,
RRMRRRS, RRRMRRRS
        results.put(newSolution, number + "_self");
        putNewSolution(leftRight, number + "_lr");
        putNewSolution(upDown, number + "_ud");
        putNewSolution(lurd, number + "_lurd");
        putNewSolution(ruld, number + "_ruld");
        putNewSolution(cw1, number + "_cw1");
```

```
            putNewSolution(cw2, number + "_cw2");
            putNewSolution(cw3, number + "_cw3");

            return true;
        }
        protected void putNewSolution(Integer[] temp, String mark) {
            String newSolution = resultToString(temp);
            String flag = results.get(newSolution);

            if(flag == null) {
                results.put(newSolution, mark);
            }
        }
        protected String resultToString(Integer[] result) {
            StringBuilder sb = new StringBuilder();
            for (int i = 1; i < queen.length; i++) {
                sb.append(result[i]);
            }
            return sb.toString();
        }
        public static void main(String[] args) {
          hw6 queen = new hw6(8);  //change the number of n in here
            queen.backtrack(1);
            System.out.print("Total Independence solutions is: ");
            System.out.print(num);
        }

}
```

n	1	2	3	4	5	6	7	8	9
U	1	0	0	1	2	1	6	12	46
D	1	0	0	2	10	4	40	92	352

```
=======================================
This is one of the Independence of 1*1
Q
There is 1 solution for normal answer
And 1 solutions for Independence
=======================================
This is no solution of 2*2
```

```
=====================================
This is no solution of 3*3
```

```
=====================================
This is one of the Independence of 4*4
. Q . .
 . . . Q
 Q . . .
 . . Q .
```
There are 2 solutions for normal answer
And only 1 solutions for Independence
```
=====================================
This is one of the Independence of 5*5
answer1:
 Q . . . .
 . . Q . .
 . . . . Q
 . Q . . .
 . . . Q .
```
There are 10 solutions for normal answer
But only 2 solutions for Independence
```
=====================================
This is one of the Independence of 6*6
answer1:
 . Q . . . .
 . . . Q . .
 . . . . . Q
 Q . . . . .
 . . Q . . .
 . . . . Q .
```
There are 4 solutions for normal answer
But only 1 solutions for Independence
```
=====================================
This is one of the Independence of 7*7
answer1:
 Q . . . . . .
 . . Q . . . .
 . . . . Q . .
 . . . . . . Q
```

```
 . Q . . . . .
 . . . Q . . .
 . . . . . Q .
```
There are 40 solutions for normal answer
But only 6 solutions for Independence

```
=======================================
This is one of the Independence of 8*8
answer1:
 Q . . . . . . .
 . . . . Q . . .
 . . . . . . . Q
 . . . . . Q . .
 . . Q . . . . .
 . . . . . . Q .
 . Q . . . . . .
 . . . Q . . . .
```
There are 92 solutions for normal answer
But only 12 solutions for Independence
```
=======================================
This is one of the Independence of 9*9
answer1:
 Q . . . . . . . .
 . . Q . . . . . .
 . . . . . Q . . .
 . . . . . . . Q .
 . Q . . . . . . .
 . . . Q . . . . .
 . . . . . . . . Q
 . . . . . Q . . .
 . . . . Q . . . .
```
There are 352 solutions for normal answer
But only 46 solutions for Independence

Report How To Run:

This code is running in eclipse.

At the beginning
Make a n*n board
Using backtracking to find all the answer.
About the backtracking, it uses deep search to find. It will guess from [1,1], then find the appropriate point in row 2, and so on.
If there is no appropriate point, then return to previous level and find another point to search.

When find each answer, go to the Independence function to check if the answer is independent or not

In the Independence function, it will generate its three rotation and four rotation of symmetric. For example, if S is a solution and M stands for mirror image, R for rotate by 90 degrees, then the 8 solutions isomorphic to S are S, RS, RRS, RRRS, MRRRS, RMRRRS, RRMRRRS, RRRMRRRS
Therefore, for each solution we get, we need to check its other 7 solutions is already stored or not.

Then check if it didn't show before, return sure and print it and put it in hash map.