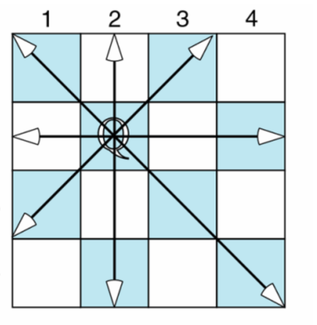
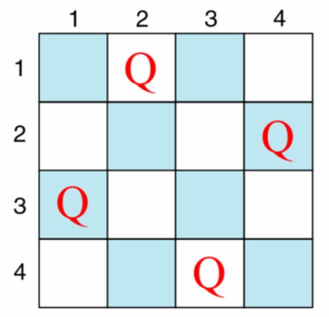
EGCI 221 – Group Project 1 (N Queens)

1. **Problem**: place N queens on N x N chessboard such that no two queens capture each other. A queen can capture another queen who is in her horizontal, vertical, or diagonal directions (as in figure (a)).

Thus, a solution for 4 queens on a 4x4 board would be as figure (b).



(a) Queen capture rules



(b) First four queens solution

2. **Programming**: the program must be written in Java.

2.1 Input

- Ask user for value N (#queens and size of square board); N must be at least 4. Let rows be indexed 1-N from top to bottom, and columns be indexed 1-N from left to right.
- Ask the user to place the first queen on the board.
- The user can also choose to not place the first queen.

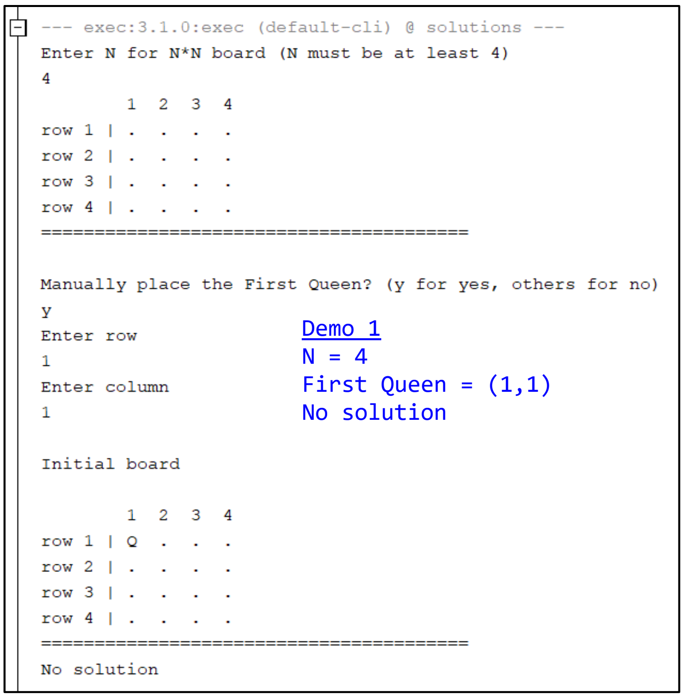
2.2 Output

- Show at least 1 solution to place all N queens on the board.
- It is possible that no solution can be found for the initial queen position given by the user.

2.3 The program must allow the user to try new first-queen placements & new board sizes.

2.4 The program must be able to check user input and handle invalid input without crashing.

2.5 Your source file (.java) must be in folder Project1_XXX where XXX = full ID of the group representative, assuming that this folder is under Maven's "src/main/java/" structure. The first line of all source files must be comments containing English names and IDs of all members.



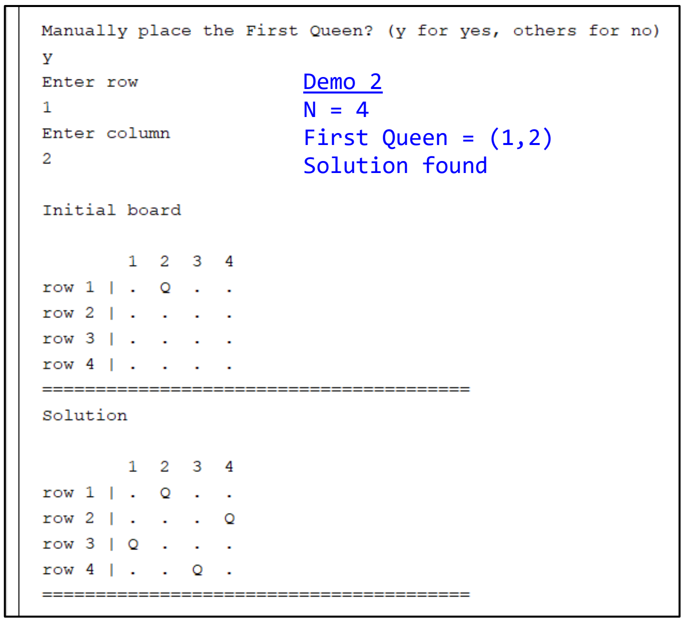
```
--- exec:3.1.0:exec (default-cli) @ solutions ---
Enter N for N*N board (N must be at least 4)
4
            1   2   3   4
row 1 |  .   .   .   .
row 2 |  .   .   .   .
row 3 |  .   .   .   .
row 4 |  .   .   .   .
========================================

Manually place the First Queen? (y for yes, others for no)
y
Enter row
1
Enter column
1

Initial board

            1   2   3   4
row 1 |  Q   .   .   .
row 2 |  .   .   .   .
row 3 |  .   .   .   .
row 4 |  .   .   .   .
========================================
No solution
```

Demo 1
N = 4
First Queen = (1,1)
No solution



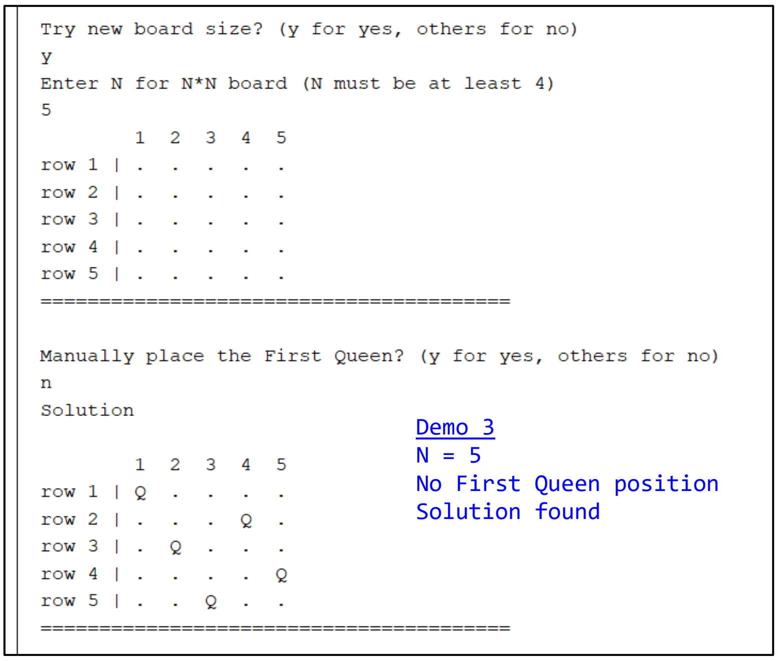
```
Manually place the First Queen? (y for yes, others for no)
y
Enter row
1
Enter column
2

Initial board

            1   2   3   4
row 1 |  .   Q   .   .
row 2 |  .   .   .   .
row 3 |  .   .   .   .
row 4 |  .   .   .   .
========================================
Solution

            1   2   3   4
row 1 |  .   Q   .   .
row 2 |  .   .   .   Q
row 3 |  Q   .   .   .
row 4 |  .   .   Q   .
========================================
```

Demo 2
N = 4
First Queen = (1,2)
Solution found



```
Try new board size? (y for yes, others for no)
y
Enter N for N*N board (N must be at least 4)
5
        1  2  3  4  5
row 1 | .  .  .  .  .
row 2 | .  .  .  .  .
row 3 | .  .  .  .  .
row 4 | .  .  .  .  .
row 5 | .  .  .  .  .
========================================

Manually place the First Queen? (y for yes, others for no)
n
Solution

        1  2  3  4  5
row 1 | Q  .  .  .  .
row 2 | .  .  .  Q  .
row 3 | .  Q  .  .  .
row 4 | .  .  .  .  Q
row 5 | .  .  Q  .  .
========================================
```

Demo 3
N = 5
No First Queen position
Solution found

Design a better user interface by yourself. Good user interface doesn't mean fancy output, but rather how your program is easy to understand & to use even without user manual. For example,

- Can user continue playing without having to restart the program?
- Are the instructions clear enough?
- Do you warn about valid user input and/or handle invalid user input?
- Is the output properly formatted and understandable?

3. This puzzle is generally known as "N Queens" puzzle. There are many approaches to solving it. But you're required to use backtracking with either explicit or implicit stack in this project. Recursive depth-first-search method is counted as using (runtime) stack implicitly.

- You can search and use any algorithm or even pieces of code, provided that the sources are properly acknowledged.
- These sources must be from your own research e.g. paper, textbook, Internet, etc.
- But using classmates as references is considered cheating.

4. **Report**: describe at least the following

4.1 Short user manual

4.2 Stack (including ArrayDeque, runtime stack) for backtracking + other data structures

- Explain how stack is used in your backtracking algorithm. Give the name of your (explicit) stack, point out where it is in your program, and what values are kept in it.
    - Backtracking must always work with stack. Even if you use recursive method, you still have to explain the above points. Do further research on how runtime stack works with recursive method and give explanation in the context of N Queens solution.
    - Stacks used in trivial tasks, that don't help the algorithm going forward or backward (see 4.3), will be counted only as other data structures in the next bullet.
- Other data structures e.g. ArrayList, LinkedList, etc. Be specific about each data structure object: give the name of the object, point out where it is in your program, reasons for using it, and what values are kept in it.

4.3 Backtracking algorithm to find at least 1 solution. Source code listing without any explanation isn't counted as a proper report. Explain the following points:

- Forwarding steps
    - The starting point of your search for valid queen placements e.g. from the first queen position, from (1, 1), from (N, N), etc.
    - The order of your search direction e.g. up-down-left-right, left-right-up-down, etc.
    - The calculation in each step + values added to/removed from your stack in each step

- Backtracking steps
    - Conditions that trigger backtracking
    - The calculation in each step + values added to /removed from your stack in each step
- How does the algorithm conclude that there is no solution?

4.4 Demonstrate how your algorithm works until a solution is found or not found, using board size and first queen positions in demo 1 and demo 2.

- In each demo, visualize board & stack content in every forwarding step and every backtracking step. Don't conjure up the moves. Your demo must match what your program actually does and your explanation in 4.3. Note that there can be >=1 solutions to the puzzle. Yours may be different from the given examples.

4.5 Any limitation of your program. Convenient limitation isn't counted as proper limitation. For example, don't give a limitation that the program will crash if N is negative just because you are too lazy to check for valid input.

4.6 References or declaration of originality

- If you design the data structures/algorithms and write the whole program by yourself, explicitly declare that everything is done by yourself. But if it is found later that this is not true, it will be counted as cheating.
- Otherwise, valid references/URLs must be given. The URLs must point to the code you take from, and I can access it without having to log in the website.
- A report without valid references or declaration of originality will get point deduction.

*** The project can be done in a group of <=4 students. But each group must do it by themselves. **Everyone involved in cheating will get ZERO point.**

Grading

Programming (10 points)

2 points	Correct solution (N=4, N>4) without first queen placement
2 points	Correct solution or no solution (N=4, N>4) given first queen placement
2 points	User interface + exception handling

- This part must be your own effort. Even if 2 groups take algorithms or pieces of code from the same source, it would be impossible to have the same/similar coding for the user interface. Therefore, same (or too similar) user interface coding is considered cheating

4 points	Programming + OOP with Java collection APIs

- Although this is not a programming course, your program should still be well structured. Excessive use of static methods as C-style functions is strongly discouraged.
- Even if you use pieces of code from the Internet. Convert it to proper OOP with Java collection APIs.

Report (10 points)

3 points	Stack for backtracking + other data structures
5 points	Backtracking algorithm + 2 demos
2 points	Manual, limitation, reference/declaration of originality, others (writing, format, etc.)

<u>Submission</u>

1. A report in <u>only 1 PDF file</u>. The front page must contain names and IDs of all members.
2. Source files (*.java).
3. File readme.txt containing English names + IDs of all members.
4. Put all files in a folder Project1_XXX (see 2.5) and zip the folder.
    - The group representative submits the whole project to Google Classroom.
    - The other group members submit only readme.txt to Google Classroom.

Late submission

• -0.5 point	for <1 week late
• -1 point	for each 1 full week late