NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

Practice Paper 4 for Practical Assessment
CS1010J — Programming Methodology

Time Allowed: 1.5 hours

INSTRUCTIONS TO CANDIDATES

1. This assessment paper consists of 2 exercises on 5 pages.
2. This is an **OPEN BOOK** assessment. You may bring in any printed material and a calculator, but **not** other electronic devices such as laptops, iPads or thumb-drives.
3. In line with the university rules, any form of communication with other students, or the use of unauthorised materials is considered cheating and you are liable to the disciplinary action.
4. The first 10 minutes of the assessment is reserved for you to read questions and design algorithms. You are not allowed to type your programs in this period of time.
5. You are to use **DrJava** or **VS Code** to write your programs. No other IDEs are allowed.
6. You may assume that all input data are valid; hence no input validation is needed.
7. Once finish, you may submit your programs to **CodeCrunch** for automatic grading.
8. You are **NOT** allowed to use any Java language syntax not covered in weeks 1-9 (e.g., string functions or recursion).

ALL THE BEST!

Example Rough Marking Scheme for an Exercise of 80 Marks

1. Style: 10 marks
   - Write program description and student number in the program header.
   - Write a short and meaningful description for every method (except main).
   - Apply consistent indentation and good naming of variables
2. Correctness and design: 70 marks
   - (**In actual PA**) Graders will manually go through your program and award method by method.
3. Additional penalties:
   - Program is not compilable: deduct 5 marks.
   - Break coding restrictions: deduct 5 - 20 marks, depending on severity.

Exercise 1: Common Element [20 marks]

You are given two arrays each contains distinct integers sorted in descending order. There is exactly one integer common to both arrays. You are to find out this common element.

For example, the common element in arrays {9, 7, 5, 3, 1} and {10, 9, 8} is 9.

Input/output statements are given in the skeleton program **CommonElement.java** so that you won't worry about output format. You task is to fill in the **`findCommon()`** method. Note that you won't receive full mark if your **`findCommon()`** method engages nested loop.

Two sample runs are shown below with user's input shown in **bold**.

Sample run #1

```
Enter the number of elements in 1st array: 5
Enter elements: 9 7 5 3 1
Enter the number of elements in 2nd array: 3
Enter elements: 10 9 8
Common element is: 9
```

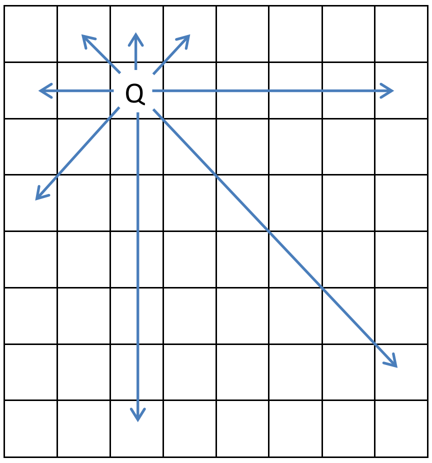
Sample run #2

```
Enter the number of elements in 1st array: 2
Enter elements: 3 2
Enter the number of elements in 2nd array: 4
Enter elements: 2 1 0 -1
Common element is: 2
```

Exercise 2: N Queens [80 marks]

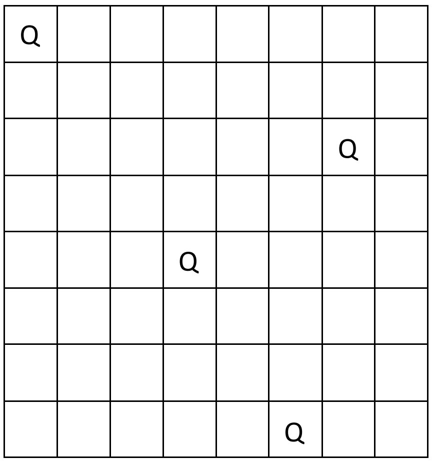
In the field of Artificial Intelligence (AI) there is a classic problem called *n-queens* puzzle, where a number of chess queens are arranged on an *n***n* chess board so that no two queens can attack each other.

A chess queen may attack any other queen that is directly above, below, on the left, on the right or diagonally from itself. An example is shown below.



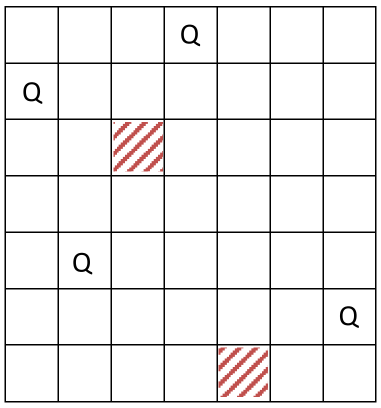
In the diagram above, the arrows show the squares that the queen can attack. Thus if there were another queen in any of those squares, this would be a "bad" arrangement. On the other hand, a "good" arrangement is one where no queen can attack any other queen. Obviously, in a good arrangement, no two queens share the same row, column, or diagonal.

The diagram below shows a good arrangement of 4 queens on an 8*8 board.



In this exercise, you are to check if a given chess board is a good arrangement. If so, further check if two more queens can be placed on the board and still results in a good arrangement.

For example, two more queens (shown as shadowed squares) can be placed on the following 7×7 board with 4 queens and still remains in a good arrangement.



Write in the skeleton program **Queens.java** and complete the following four methods:

- **`int[][] readBoard()`** that reads a **`board`** of size **`n*n`** and returns it. The **`board`** is filled with integers, with 0s representing empty squares and 1s representing queens. You may assume **`n`** > 3 and **`board`** contains at least 1 queen.
- **`boolean canAttack(int[][] board, int r, int c)`** that checks if a queen on row **`r`** and column **`c`** can attack any other queen. It returns true if so, or false otherwise.
- **`boolean isGoodArrangement (int[][] board)`** that checks if **`board`** is a good arrangement. It returns true if so, or false otherwise.
- **`boolean twoMoreQueens(int[][] board)`** that takes a **`board`** in a good arrangement as parameter, checks if two more queens can be placed on the **`board`** and still results in a good arrangement. It returns true if so, or false otherwise.

Take note of the following coding restrictions:

1. You must **not** change the method headers given in the skeleton program.
2. You are **not** allowed to change the **`main()`** method as well.
3. You may write additional methods or change **`board`** as necessary.

Three sample runs of the program are given below with user's input shown in **bold**. For your convenience of testing, test inputs are also appended at the back of the skeleton program.

Sample run #1

```
Enter the size of the board: 7
Enter the 7*7 board:
0 0 0 1 0 0 0
1 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 1 0 0 0 0 0
0 0 0 0 0 0 1
0 0 0 0 0 0 0
Good arrangement
Two more queens can be placed on the chess board
```

Sample run #2

```
Enter the size of the board: 4
Enter the 4*4 board:
1 0 0 0
0 0 0 1
0 0 0 0
0 0 0 0
Good arrangement
Cannot place two more queens on the chess board
```

Sample run #3

```
Enter the size of the board: 6
Enter the 6*6 board:
0 0 0 1 0 0
0 1 0 0 0 0
0 0 0 0 1 0
0 0 0 0 0 1
0 0 0 0 0 0
0 0 1 0 0 0
Bad arrangement
```

=== END OF PAPER ===