INTRODUCTION TO COMPUTER SCIENCE II
LAB 7
Sudoku and Stacks

1 A Sudoku solver

Begin by creating a new `lab-7` directory, into which you will copy an already made solver for Sudoku puzzles. Grab source code for that directory:

- **On `remus/romulus`**, perform the following command:
  `$ cp -r ~sfkaplan/public/COSC-112/lab-7/* .`
- **On your own computer**, download and extract this zip file.

You will find the following files:

- `Sudoku.java`: The class that contains the `main()` method. When run, it reads the puzzle from a file (specified by the user at the command line), displays the puzzle, calls the solver, and then displays the solution.
- `BoardPosition.java`: These objects are tied to a specific position on the puzzle (a.k.a., the board). When one is constructed, it finds and immutably sets itself to work with the first empty location (i.e., the first position, in scanned order, that contains a `0`). It can allow the value at its position to be set, it can determine whether the value at its position collides with another on the board,[1] print itself, etc.
- `easy.board`, `medium.board`, and `evil.board`: Three proper sudoku puzzles, given in the form that the `Sudoku` class can read. These files can be opened in a text editor and easily viewed or edited. Their difficulty—that is, the amount of backtracking required to solve them—is described by the name of each.

As provided, this code compiles and correctly solves these puzzles (and, I think, any other—feel free to add your own!).

Your task: Modify the `solve()` method in `Sudoku.java` **not** to use recursion. Instead, your solver must create and maintain its own *stack* to direct the search for a solution. You may, of course, add classes and methods as needed. I also recommend using the standard Java `Stack<E>` class.

[1] That is, does the value at its position also occur in its row, column, or subgrid.

2 How to submit your work

Don't! This lab is for your own exploration.