CSC171 — Project 2 — Puzzling Arrays

Due: Sunday, November 6th by 1159PM EDT.

For this project, you are asked to write a program which generates puzzles and allows a user to interactively solve them. This will give you an opportunity to practice and demonstrate your understanding of arrays, control flow, encapsulation, and general object oriented design.

Program Requirements

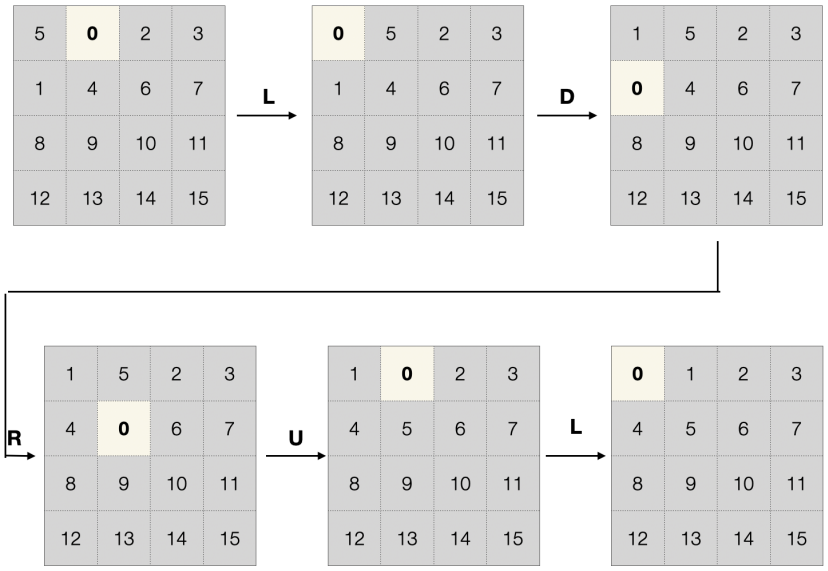
The specific kind of puzzle we will explore is known as a "Sliding Block Puzzle". Sliding block puzzles consist of an NxN grid of movable tiles (aka "blocks"), where each tile is labeled with a separate integer. For this project, you may assume that all puzzles are 4x4, for a total of 16 blocks. One special block is the "blank", which we will represent using zero. The blank can exchange places with its neighbors directly above, below, left, or right. If this were a physical puzzle, you would actually slide the tiles around. Since we are building a software model of a puzzle, *you can think of a move as swapping the zero with and adjacent integer.*

The objective of the puzzle is to find a sequence of moves such that all the non-blank tiles are in-order at the end, with the blank either at the top-left of the board, or at the bottom right. Take a look at Figure 1a and Figure 1b for an example of solving a puzzle through a sequence of moves.

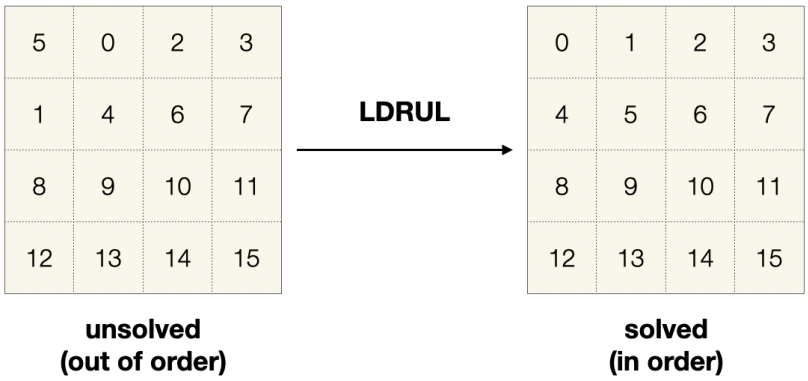
Your submission for this project should be a single Java file for the main `Puzzle` class. Be sure to include a block comment at the top listing your full name, your school email, and whether you worked alone or with a partner. If you worked with a partner, you must include a paragraph describing how you collaborated and who worked on what.

Your program should have a main method which reads a puzzle from the user and then reads a move sequence (as a single token). Your program should them simulate the moves and inform the user of the result. If the sequence solves the puzzle, print "success", if the sequence is valid but does not solve the puzzle, print "failure", if there are any invalid moves print "invalid".

In the case of success or failure, the program should print the final board configuration. In the case of an invalid move, the program should print the last valid board configuration before the invalid move, and a message indicating how many steps were completed before the problem occurred.



(a) Example of a puzzle solution, in steps.



(b) Example of the same puzzle solution, end to end.

Required Methods

Your class must support the following methods:

- `public Puzzle copy();` - return a new instance of a `Puzzle` with an identical board to the current instance.
- `public void display();` - display the board as a 4x4 grid of integers. Four per line, with each integer taking 4 spaces (i.e., use `%4d` with `printf`).
- `public void doMove(char m);` - change the state of the board to reflect performing move `m` where `m` $\in$ {'L', 'R', 'D', 'U'}.
- `public boolean validMove(char m);` - return true if the given move is valid, and false otherwise. (I.e., check if the zero tile is against the border.)
- `public boolean solved();` - return true if the board is in the solved configuration, false otherwise.
- `public Puzzle(int[][] board);` - a constructor which builds an instance of a `Puzzle` with a board matching the supplied parameter. Note that this creates a new internal array, and does not alias the parameter.
- `public boolean checkSolution(char[] moves);` - a method which determines if the given sequence of moves results in a solution. If the final configuration is not a solution, or if any of the intermediate moves are invalid, this method should return false. Note that if the board is "temporarily" solved but then unsolved during the move sequence, the method should still return false.

You are free to create other methods as well, but consider whether they should be public or private. In particular, you may want to write a method for directly reading a `Puzzle` from the user, but that behavior is left for you to decide. All instance variables must be private.

Input Data Format

A puzzle configuration will always consist of 16 integers given across four in the same order — left to right, top to bottom. The user should be expected to enter the puzzle configuration with each row of integers as a separate line. (E.g., each row of the puzzle should correspond to one line of input.) The integers should be separated by whitespace, but they are not required to be any specific number of spaces or tabs. (Hint: this means you should be using `Scanner.nextInt()`.)

A move is represented using the motion of the blank tile, which can either move Up, Down, Left, or Right. A move sequence will be represented as a String of letters - with no spaces - where each letter is either U, D, L, or R. E.g., the string "UURD" corresponds to the move sequence: up, up, right, down.

Collaboration

You may choose to complete this assignment either on your own (i.e., solo) or with a partner. Students working as a team will receive the same grade. Groups of more than two students are not allowed. If you work with a partner, you must ensure that names and emails for both students are listed as comments at the top of each and every Java file. Moreover, your source file comments must describe the contributions of each team member. If you work with a partner, only ONE person should submit the source code, and the other partner should submit a plain text file stating who they worked with, and who is making the submission.

Academic Honesty

The idea of sliding block puzzles is very old — dating back at least to 1880. There are several puzzle solvers online, and probably even a few Java implementations which are similar in scope to this project. *Searching online for solutions (e.g., via github) is STRICTLY FORBIDDEN and would be considered as a significant instance of academic dishonesty.* Any submission of plagiarized code (even with modifications) will be reported to the board of academic honesty and penalties will be applied.

All that being said, you are welcome to research the general concept of sliding block puzzles on the internet. Although you are not asked to write a program to SOLVE the puzzles, there are a variety of very interesting techniques to make that happen, and if you are so inclined then you are encouraged to take a look. Where we draw the line is looking for CODE solutions. Reading about the 15 puzzle online? Totally OK! Looking for Java (or other language) source code? Definitely NOT allowed!

Grading

All submissions will be graded according to the following approximate rubric:

Points	**Category**
10	Encapsulation
10	Data Input
10	`copy()`
10	`display()`
15	`doMove()`
10	`validMove()`
10	`solved()`
10	`Puzzle(int[][] board)`
15	`checkSolution(char[] moves)`

Revisions

1. Oct 16 - First revision.

Examples