CSCI-C212/A592 – Intro to Software Systems
Spring 2017

SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

Assignment 5

Due by 4/21/2017, Friday midnight through Canvas

Instructions:

- Review the requirements given below and Complete ***all*** parts from below. Please submit all files through Canvas.
- The grading scheme is provided on Canvas. Be sure that your code includes everything required in the grading rubric.

N-Puzzle Problem

The problem. The 8-puzzle problem is a puzzle invented and popularized by Noyes Palmer Chapman in the 1870s. It is played on a 3-by-3 grid with 8 square blocks labeled 1 through 8 and a blank square. Your goal is to rearrange the blocks so that they are in order, using as few moves as possible. **You are permitted to slide blocks horizontally or vertically into the blank square. For the programming assignment, the blank square is represented by a square with 0 as its value.** The following shows a sequence of legal moves from an *initial board* (left) to the *goal board* (right).

```
    1  3        1     3        1  2  3        1  2  3        1  2  3
 4  2  5   =>   4  2  5   =>   4     5   =>   4  5      =>   4  5  6
 7  8  6        7  8  6        7  8  6        7  8  6        7  8

 initial        1 left          2 up          5 left          goal
```

Board Abstract Data Type

Your programming task: Implement the Board class according to the assignment specifications and the template provided. The Solver class is the class you run to test your board class and determine if an input board is solvable. I have added the test cases for you in the main method of the Solver class. The Solver class uses the A* algorithm, which is used in artificial intelligence algorithms, to solve the 8-puzzle problem. **You do not need to make any changes to any of the other classes, only implement the Board class.** The Solver class relies on some other data structures provided. You do not need to be concerned with the MinPQ data structure provided, that is just used by the solver class. You will need to use the Stack (which you have implemented before) in the Board class. The Stack class has been provided for this assignment and uses Generic types as ArrayList do, so you will need to initialize it the same.
Stack<Board> stack = new Stack<>();

All you need to use from the stack class is *stack.push(Board b)* to push all the neighboring boards on to the Stack and return the Stack.

You can test your board class methods as you are writing the methods, but when it is complete, running the Solver class will run the A* algorithms on the test cases provided with the assignment.

You do need to implement two heuristic functions that the Solver class uses to determine what number to swap with the blank square. The Solver class looks at all the possible moves in the boards current state, then uses one of these heuristics to determine the best board to choose.

We consider two priority functions: The board described below is an intermediate board state from the initial puzzle as it moving to a goal board.

- *Hamming priority function.* The number of blocks in the wrong position. Intuitively, a search node with a small number of blocks in the wrong position is close to the goal, and we prefer a search node that have been reached using a small number of moves.
- *Manhattan priority function.* The sum of the Manhattan distances (sum of the vertical and horizontal distance) from the blocks to their goal positions.

  For example, the Hamming and Manhattan priorities of the initial search node below are 5 and 10, respectively.

```
 8  1  3        1  2  3    1  2  3  4  5  6  7  8    1  2  3  4  5  6  7  8
 4     2        4  5  6     ----------------------    ---------------------
 7  6  5        7  8       1  1  0  0  1  1  0  1    1  2  0  0  2  2  0  3

 initial          goal         Hamming = 5          Manhattan = 10
```

Board and Solver data types. Organize your program by creating an immutable data type Board with the following API:

```
public class Board {
  // construct a board from an N-by-N array of blocks
  // (where  blocks[i][j] = block in row i, column j)
  public Board(int[][] blocks)

  // board dimension N
  public int dimension()

  // number of blocks out of place
  public int hamming()

  // sum of Manhattan distances between blocks and goal
  public int manhattan()

  // is this board the goal board?
  public boolean isGoal()
```

```
   // a board that is obtained by exchanging any pair of blocks
   public Board twin()

   // does this board equal y?
   public boolean equals(Object other)

   // all neighboring boards – all the possible moves that can be made from the current board.
   // the moves that can be made are swapping the empty block with all neighbor blocks
   // a neighbor block is any inbound block that is one index above, below, left, or right of the
   // blank square
   public Stack<Board> neighbors()

   // string representation of this board
   public String toString()

   public static void main(String[] args) // unit tests
}
```

Possible Questions:
Is 0 a block? No, 0 represents the blank square. Do not treat it as a block when computing either the Hamming or Manhattan priority functions.

I'm a bit confused about the purpose of the twin() method. Don't get bogged down about the details of this function. The Solver class uses a twin board to complete the algorithm. The Left most Board below is the initial board, and the rest are all valid twin boards. You just need to swap to numbers positions (not including the 0 block).

```
    1  3       3  1       1  3       1  3       1  3       6  3
 4  2  5    4  2  5    2  4  5    4  5  2    4  2  5    4  2  5
 7  8  6    7  8  6    7  8  6    7  8  6    8  7  6    7  8  1

  board      twin       twin       twin       twin       twin
```

Running the Solver class with a correctly implemented Board class will output the following:

```
--- Testing puzzle04 ---
Minimum number of moves = 4
3
 1  0  3
 4  2  5
 7  8  6

3
 1  2  3
 4  0  5
 7  8  6

3
 1  2  3
 4  5  0
 7  8  6
```

```
3
 1  2  3
 4  5  6
 7  8  0


--- Testing puzzle3x3-unsolvable.txt ---
No solution possible

--- Testing puzzle01 ---
Minimum number of moves = 1
2
 1  2
 3  0
```