Objective(s):

a. To practice enumerating states for searching for goal state stack (dfs)

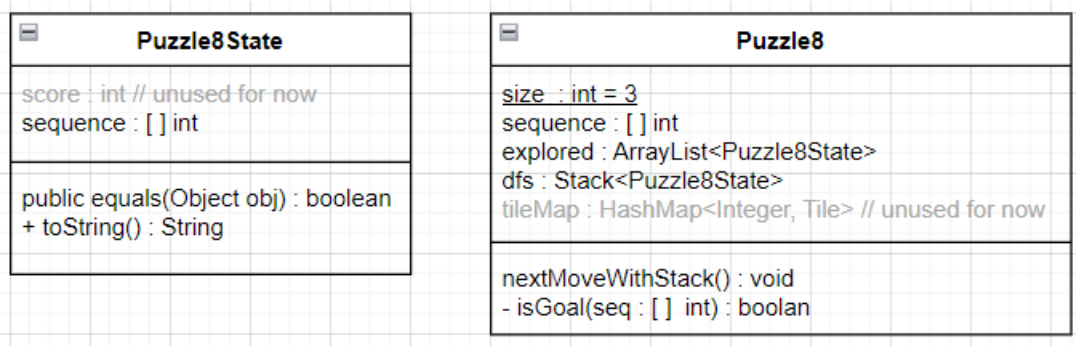
In the 8_Puzzle problem, there are 3x3 grid with 8 numbered tiles and one empty tile (or blank tile). The naïve solution is to apply depth-first-search generating states from the current state to the goal state.

(In AI field, The goal is to rearrange the tiles from a given initial state to a desired goal state using the minimum number of moves.) This is <u>not</u> an AI lab, do not write an AI solution unless stated.

	1	3
4	2	5
7	8	6

1	2	3
4	5	6
7	8	

Given the class diagram below



Let's simply represent the game state with 1D array (called sequence) i.e.

The input of `new int [] {9,0,0,1,0,1,3,0,2,4,1,0,2,1,1,5,1,2,7,2,0,8,2,1,6,2,2}`

becomes `[9, 1, 3, 4, 2, 5, 7, 8, 6]` notice that the index of the face value (1 – 9) is corresponded to row and col with the equation row = idx / size and col =idx % 3.

Task 1 Create Puzzle8's sequence from the input

```
static void demo1() {
    Puzzle8 game = new Puzzle8(new int [] {9,0,0,1,0,1,3,0,2,4,1,0,2,1,1,5,1,2,7,2,0,8,2,1,6,2,2});
    game.displayBoard();
}
```

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

Task 2 The priority now is to try to swap the blank to any eligible directions i.e. given blackPos (index of 9)

(blankPos + 3 < size * size) // south or (blankPos – 3 > -1) // north or (blankPos % 3 < 2) // east or (blankPos % 3 > 0) // west

For example,

`[9, 1, 3, 4, 2, 5, 7, 8, 6]`

Next possible moves are

`[4, 1, 3, 9, 2, 5, 7, 8, 6]`
`[1, 9, 3, 4, 2, 5, 7, 8, 6]`

It is now a proper time to test explored as well. (give Puzzle8State's score attribute 0 for now). This means test whether .contains() works properly such that the generated Puzzle8State will not cause an infinite loop while enumerating the puzzle's moves.

```
static void demo2() {
    Puzzle8 game = new Puzzle8(new int [] {9,0,0,1,0,1,3,0,2,4,1,0,2,1,1,5,1,2,7,2,0,8,2,1,6,2,2});
    // Puzzle8 game = new Puzzle8(new int [] {1,0,0,2,0,1,3,0,2,4,2,2,5,1,0,6,1,1,7,2,0,8,2,1,9,1,2});
    game.generateNextMove();
}
```

```
pushing south 5 [4, 1, 3, 9, 2, 5, 7, 8, 6]
pushing east 3 [1, 9, 3, 4, 2, 5, 7, 8, 6]
true
```

Task3 Follow the depth-first-search process which add the start state to the stack and enter the while loop for popping out the top state and pushing the possible move states until the stack is empty.

isGoal() implementation should not be a problem (testing whether the (state) sequence is `[1,2,3,4,5,6,7,8,9]`).

```
static void demo3() {
    Puzzle8 game = new Puzzle8(new int [] {9,0,0,1,0,1,3,0,2,4,1,0,2,1,1,5,1,2,7,2,0,8,2,1,6,2,2});
    game.nextMoveWithStack();
    System.out.println(game.explored.size());
    System.out.println(x:"partial explored state");
    for (Puzzle8State s : game.explored) {
        if (s.sequence[0] == 1 && s.sequence[1] == 2 && s.sequence[2] == 3 && s.sequence[3] == 4)
            System.out.println(Arrays.toString(s.sequence));
    }
    System.out.println(x:"note that the program terminates prior to pushing goal state into explored!!");
}
```

Adjust your code output as you want (what is your estimates the number of states generated given the example) to track the algorithm progress.

```
number of pop invocation = 36605 stack size = 25555 explored size = 62160
number of pop invocation = 36606 stack size = 25555 explored size = 62161
number of pop invocation = 36607 stack size = 25556 explored size = 62163
number of pop invocation = 36608 stack size = 25556 explored size = 62164
from isGoal [1, 2, 3, 4, 5, 6, 7, 8, 9]
found goal [1, 2, 3, 4, 5, 6, 7, 8, 9] let's terminate the loop
partial explored state
[1, 2, 3, 4, 9, 5, 7, 8, 6]
[1, 2, 3, 4, 8, 5, 9, 7, 6]
[1, 2, 3, 4, 7, 8, 9, 5, 6]
[1, 2, 3, 4, 7, 8, 5, 9, 6]
[1, 2, 3, 4, 9, 8, 5, 7, 6]
[1, 2, 3, 4, 7, 8, 5, 6, 9]
[1, 2, 3, 4, 7, 9, 5, 6, 8]
[1, 2, 3, 4, 9, 7, 5, 6, 8]
[1, 2, 3, 4, 6, 7, 5, 9, 8]
[1, 2, 3, 4, 7, 5, 9, 6, 8]
[1, 2, 3, 4, 9, 5, 6, 7, 8]
[1, 2, 3, 4, 6, 8, 9, 7, 5]
[1, 2, 3, 4, 9, 6, 5, 8, 7]
[1, 2, 3, 4, 9, 5, 8, 6, 7]
[1, 2, 3, 4, 9, 6, 7, 5, 8]
[1, 2, 3, 4, 5, 6, 7, 9, 8]
[1, 2, 3, 4, 6, 9, 7, 5, 8]
[1, 2, 3, 4, 9, 8, 7, 6, 5]
[1, 2, 3, 4, 6, 8, 7, 9, 5]
[1, 2, 3, 4, 8, 9, 7, 6, 5]
[1, 2, 3, 4, 8, 5, 7, 6, 9]
[1, 2, 3, 4, 7, 6, 9, 8, 5]
[1, 2, 3, 4, 7, 6, 8, 9, 5]
[1, 2, 3, 4, 9, 6, 8, 7, 5]
[1, 2, 3, 4, 7, 6, 8, 5, 9]
[1, 2, 3, 4, 7, 9, 8, 5, 6]
[1, 2, 3, 4, 9, 7, 8, 5, 6]
[1, 2, 3, 4, 5, 7, 8, 9, 6]
[1, 2, 3, 4, 5, 9, 7, 8, 6]
note that the program terminates prior to pushing goal state into explored!!
```

Remark To prevent this exploration of states generated, compute the sum of Manhattan Distance sum of Tile t1 to t8 and store the score with in Puzzle8State instance. Now, instead of using a stack, you could simple use a priority queue to let the queue poll() the minimum score state to proceed the searching of goal state. (i.e. we are not trying back-track here to obtain the solution.). With priority queue, the generated states should be much less than of a stack because there should always be a state closer to goal be generated and be poll()ed.

```
class Tile {
    int face;
    int correctRow;
    int correctCol;
    Tile(int id) {
        face = id;
        correctRow = (id - 1)/3;
        correctCol = (id - 1)%3;
    }
    public String toString() {
        return "I am " + face;
    }
    int manhattanDistance(int curRow, int curCol) {
        return Math.abs(curRow - correctRow) + Math.abs(curCol - correctCol);
    }
}
```

One straightforward way to compute the sum of the Manhattan Distance is through the use of tileMap<Integer, Tile> which allows you quickly obtain the tile's correct coordination to compute each tile's distance.

(You do not have to submit this informed search algorithm on Manhattan summation because it is beyond the course's learning objectives. However, I encourage you to challenge your skills.)

Submission: Puzzle8_XXYYYY java

Due date: TBA