CSci 1103 Fall 2019
Programming Project 3
Implementing a 15-Puzzle Game
Due: Friday, November 1st at 11:59 PM

In this project you build upon the code you wrote in Lab 7 to implement a working 15-Puzzle computer game. Your program will allow a user to solve the 15-Puzzle by making repeated tile moves until the puzzle is in a solved configuration. When the puzzle is solved, the program should print a congratulatory message and terminate. There should also be a way to stop the program gracefully mid-game. From your lab, you will utilize: your `puzzle` array(s), `initSolution()`, `printPuzzle()` and `puzzleEqual()` as well as code provided for you. A big part of this project is integrating code you have already written, with code provided, and with code you will write.

You may choose to either work with a partner or work alone for this project. If you choose to work with a partner, only one submission is required. However, pay close attention the **submission requirements** listed at the end of the project writeup.

Your program will consist of several components, each of which can be implemented as a separate method. We will provide for you a `main()` method and methods `swap()` and `swapper()` which should be used *without* modification.

There are *two* methods that you will need to write. They are described below.

initializer(int[][] p) – This method will initialize a 15-puzzle to a “random” configuration. It should accept one parameter, your `puzzle` array.

Approach: Begin with a puzzle initialized to a solved puzzle using `initSolution()` from your lab. A random configuration of the 15-puzzle will be created by making 10,000 random swaps of the open spot and a tile adjacent to it.

Outline:

1. Using `initSolution()`, begin by initializing the given parameter array `p` to a solved 15-Puzzle.

2. Since there will be 10,000 random swaps of the open spot and an adjacent tile, create a variable to keep track of how many swaps have been made so far. Call this variable `swapCount`.

3. Additionally, it should be ensured that a new swap does not undo the previous swap which just occurred. Create a variable to keep track of the tile that was previously swapped. Call this variable `prevTileSwapped`.

4. Set up a `while` loop that will attempt to make a swap of the open spot and an adjacent tile with each iteration. It should iterate as long as 10,000 swaps have not yet been made.

5. To pick a random tile to swap the open spot with, use `randomInt()` with an appropriate range (i.e. 1-15). Obviously not every randomly generated tile will be adjacent to the open spot and thus would not be a valid swap. Luckily, the `swapper()`method provided for you can help determine if the swap is valid or not. The method `swapper()` takes two parameters, puzzle `p` and tile `n`, it will attempt to swap tile `n` with the open spot in puzzle `p`. If the swap is successful, it will return `true`. Otherwise, it will return `false` and no swap occurs. Note that `swapper()` will not ensure the previous swap is left untouched, an additional condition must be used to ensure that. Now if the swap is successful *and* does not undo the previous swap, both `swapCount` and `prevTileSwapped` should be updated accordingly.

6. After the `while` loop finishes, 10,000 random swaps will have been made and the puzzle `p` is now in a "random" and solvable configuration.

playLoop(int[][] p, int[][] sol)– This method will *repeatedly* allow the user to move a tile by selecting a number between 1 and 15, inclusive. It accepts two parameters, your `puzzle` array and the `solution` array.

Approach: The input is only a number—no direction to move the tile is needed, since there is only one open spot on the board in which to move a tile. Only a tile that is adjacent to the open spot can be moved. Therefore, a legal move must not only be in the range 1-15, inclusive, but it must also be a tile that is adjacent to the open spot. Use `TextIO` to get a tile number from the user.

Outline:

1. Print out the `puzzle` array so that the user can see the current configuration of the puzzle (using `printPuzzle()` from lab).

2. Prompt the user for an input value between 1 and 15, inclusive, using `TextIO`. Also, allow input of -1 to quit early.

3. Call method `swapper()` (which is provided for you) to verify that the input is in the range 1-15, inclusive, verify that the tile is adjacent to the open spot, and swap the tile selected with the open spot. If the tile selected is out of range or not adjacent to the open spot, `false` is returned and no tile movement is performed; otherwise, `true` is returned and the tile selected is swapped with the open spot.

4. The `puzzle` array should be checked to see if the solution has been reached. Use `puzzleEqual()` to compare your `puzzle` array with the `solution` puzzle.

5. Repeat steps one through four above until the puzzle is solved or a -1 is entered.

6. A congratulatory note should be printed when the puzzle is solved, and a simple "Bye" should be printed if -1 is entered.

A sample dialog of the running puzzle is shown below:

```
Welcome to the 15-Puzzle

 1      2      3      9
 6      5     13     15
 7     10     11     14
 4      0      8     12

Enter tile to move:
10

 1      2      3      9
 6      5     13     15
 7      0     11     14
 4     10      8     12

Enter tile to move:
? 7

 1      2      3      9
 6      5     13     15
 0      7     11     14
 4     10      8     12

Enter tile to move:
? 4

 1      2      3      9
 6      5     13     15
 4      7     11     14
 0     10      8     12

Enter tile to move:
? 10

 1      2      3      9
 6      5     13     15
 4      7     11     14
10      0      8     12

Enter tile to move:
? 8

 1      2      3      9
 6      5     13     15
 4      7     11     14
10      8      0     12

Enter tile to move:
? 12

 1      2      3      9
 6      5     13     15
 4      7     11     14
10      8     12      0

Enter tile to move:
? -1
Bye
```

Use the code and template from Puzzle.java to complete your assignment.

Submitting

If you’re working with a partner, only one submission is required. *However, please follow the submission instructions below. Failure to do so could result in you or your partner not receiving points for this assignment.*

1. Make sure **both** of your names, student IDs, and x500s are on the top of each file you submit.
2. Make sure the name of your zip file includes **both** of your x500s. *e.g. If John Smith (whose email is smith012@umn.edu) is working with Jane Doe (whose email is doe4147@umn.edu), John would upload "smith012_doe4147_project3.zip". Jane does not need to upload anything since only one submission is required.*
3. If you are the one submitting the assignment, leave a **comment** on your submission with the name of your partner.

Please submit a .zip file, conforming to the naming specifications of past projects, containing: **Puzzle.java**, and **TextIO.java**.

If you’re working alone, just submit as you did for previous projects.