Assignment #4: The Tile Game

1 Introduction

In this lab/assignment, you will finish up the tile game by creating all necessary methods to make the game playable. Instructions on methods names, input parameters, returning values and their functionality are given in the next sections.

2 Instructions / Guidelines

2.1 Create NetBeans Project

Create a NetBeans project. You may name it whatever you want. I will reference it as TileApp throughout this document.

2.2 Avoiding Static Methods

To avoid having NetBeans complaining that your methods must be static in nature, please do that small trick that I do in class all the time:

```
public class TileApp {

    public static void main(String[] args) {
        new TileApp();
    }

    public TileApp(){
        // place all your method calls here, not in the main method!
    }

    // define all your methods down here

}
```

2.3 The Game Coding Structure

The game structure is basically a sequence of methods calls. This sequence of calls will be placed in the TileApp() method as shown below. This is given to you, so just copy and paste it inside your TileApp() method. Your job in this lab/assignment is to create the methods used by this sequence.

```
public TileApp() {
    boolean gameOver = false;
    while (!gameOver) {
        printGameOnScreen();
        int chosenTile = getUserSelectedTile();
        int[] tileLocationOnTheBoard = getTileLocation(chosenTile);
        int[] emptyLocationOnTheBoard = getTileLocation(0);
        if (canTileBeMoved(tileLocationOnTheBoard, emptyLocationOnTheBoard)) {
            moveTile(tileLocationOnTheBoard, emptyLocationOnTheBoard);
            if (isGameSolved()) {
                printCongratsMessage();
                gameOver = true;
            }
        }
    }
    printGameOnScreen();  // printing solved board just before closing the game
}
```

2.4 Discussion of each method shown in Section 2.4

2.4.1 Method printGameOnScreen()

Method Name: printGameOnScreen
Method inputs: nothing
Method return: nothing
Method Functionality: Displays the current board on the terminal window. An example is shown below. Pay attention to the vertical alignment of the numbers when implementing this method:

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

2.4.2 Method getUserSelectedTile()

Method Name: getUserSelectedTile
Method inputs: nothing
Method return: an integer number representing the tile game, such as 12
Method Functionality: Asks the use to enter the tile number via keyboard. Checks if the number is valid for the game (number must be between 1 and 15). This method keep asking the tile number until the entered number is valid.
Bonus: if the user by mistake enters a letter along with a number (typo while entering the number, such as j12) your program should not abort with an exception. Instead the method should ask again for another number. You can do this by surrounding your code with Try and Catch as shown below:

```
private int getUserSelectedTile() {
    while (true) {
        try {
            // your code here
        } catch (Exception e) {
            // this line is just to clear the scanner buffer if needed
            // try to keep or remove the following line and see what
            // happens when you enter a bad tile number (such as k2)
            keyboard.nextLine();
        }
    }
}
```

2.4.3 Method getTileLocation ()

Method Name: getTileLocation
Method inputs: an integer number representing the user chosen tile number
Method return: an integer array containing the row and column values of the tile location in the board.
Method Functionality: The method goes over every single row and column (nested FOR loop) and checks where the given tile number is located in the board. As soon as it finds its location, it creates an integer array, place the row and column into that array and returns it.
Hint: The following line of code shows how to create and return an integer array with two numbers.

```
return new int[]{number1, number2}
```

2.4.4 Method canTileBeMoved ()

Method Name: canTileBeMoved

Method inputs: two integer arrays. The first array contains the tile location coordinates (row and col) and the second array contains the empty spot coordinates (row and col).

Method return: returns true if the chosen tile and the empty spot are neighbors. Returns false if they are not.

Method Functionality: The method checks if the empty spot is neighbor of the chosen tile either above, below, .to the right or to the left.

Hint: The empty spot is above the chosen tile if both of them are in the same column, but the empty spot is in the row one value smaller than the row of the tile. The following example exemplifies this logic. You will use this logic for the other 3 possible situations (below, right and left)

```
if ((emptyLocationOnTheBoard[0] == tileLocationOnTheBoard[0] - 1)
    && (emptyLocationOnTheBoard[1] == tileLocationOnTheBoard[1])) {
            return true;
}
```

2.4.5 Method moveTile ()

Method Name: moveTile

Method inputs: Two integer arrays. The first array contains the tile location coordinates (row and col) and the second array contains the empty spot coordinates (row and col).

Method return: returns true if the chosen tile and the empty spot are neighbors. Returns false if they are not.

Method Functionality: This method switches the chosen tile number (let's say tile number 7and the empty spot (0) in the 2D board array.

Hint: Remember two important things when dealing with 2D arrays:

a) To get the tile number value from its location, we can do this:

```
int tileNumber  = gameBoard[its row][its col];
```

b) To set a new value in the 2D board array you can do this:

```
gameBoard[some_row_here][some_col_here] = some_value_here
```

Where:

- `some_value_here` will be the tile number (`tileNumber`) or 0 if it is the empty spot.
- `some_row_value` and `some_col_value` are the location (row and column) of the place in the gameBoard where you want the new value to be placed at.

2.4.6 Method isGameSolved ()

Method Name: isGameSolved
Method inputs: nothing
Method return: returns true if the game has been solved, otherwise it returns false. The game is solved when all the tile numbers are in order.
Method Functionality: This method should go over all the numbers found in gameBoard 2D array and check if they are in the right location (e.g., the tile number 1 should be on row=0, and col=0, and so on).
Hint: Create a second 2D array that has the game solution (`gameSolution`) and then do a nested FOR loop and for each (row, col) compare the numbers between the actual `gameBoard` and the `gameSolution`. At the first non-equal comparison then return false. If after going through all rows and columns there were no differences, then return true.

Place the following 2D arrays at the beginning of your class, so they are treated as global variables

```
int[][] gameBoard = {
    {1, 3, 8, 12},
    {10, 2, 0, 11},
    {6, 7, 13, 9},
    {4, 14, 15, 5}
};

int[][] gameSolution = {
    {1, 2, 3, 4},
    {5, 6, 7, 8},
    {9, 10, 11, 12},
    {13, 14, 15, 0}
};
```

2.4.7 Method printCongratsMessage ()

Method Name: printCongratsMessage
Method inputs: nothing
Method return: prints in the terminal window a congratulation message for winning the game.
Method Functionality: prints in the terminal window a congratulation message for winning the game. Put whatever message you would like to have there.

3 Test your program

With the boardGame described on item 2.4.6, run your program and try to get it solved. It might take few minutes, but it can be done. Be patient!

the end you should have your application up and running. An example of the game being played is shown below: Note: your program shall check for invalid input, such as “20” (as shown below in yellow). Your program also should do nothing if the chosen tile is not close to the empty spot, as shown in green below. It should move only tiles that are close to the zero spot, as shown in blue.

Game Starts…

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

Chose Tile: 20

Invalid Tile Number,

please try again.

Chose Tile: 10

Chose Tile: 6

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

Chose Tile: 15

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

Chose Tile: 5

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

Chose Tile: 5

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

Chose Tile: 4

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

Chose Tile: 2

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

Chose Tile: 3

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

Chose Tile: 1

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

Chose Tile: 5

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

Chose Tile: 3

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

Chose Tile: 2

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

Chose Tile: 4

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

Chose Tile: 3

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

Chose Tile: 15

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

Chose Tile: