Programming Project 4

Due Sunday, April 21 at 11:59pm EDT

I. Overview

The purpose of this homework is to give you practice designing and creating a complete program.

II. Code Readability (20% of your project grade)

To receive the full readability marks, your code must follow the following guidelines:

- All variables (fields, parameters, local variables) must be given appropriate and descriptive names.
- All variable and method names must start with a lowercase letter. All class names must start with an uppercase letter.
- The class body should be organized so that all the fields are at the top of the file, the constructors are next, the non-static methods next, and the static methods at the bottom with the main method last.
- There should not be two statements on the same line.
- All code must be properly indented (see Appendix F of the Lewis book for an example of good style). The amount of indentation is up to you, but it should be at least 2 spaces, and it must be used consistently throughout the code.
- You must be consistent in your use of {, }. The closing } must be on its own line and indented the same amount as the line containing the opening {.
- There must be an empty line between each method.
- There must be a space separating each operator from its operands as well as a space after each comma.
- There must be a comment at the top of the file that **is in proper JavaDoc format** and includes both your name and a description of what the class represents. The comment should include tags for the author. (See Appendix J of the Lewis book of pages 226-234 if the Evans and Flanagan book.)
- There must be a comment directly above each method (including constructors) that **is in proper JavaDoc format** and states *what* task the method is doing, not how it is doing it. The

comment should include tags for any parameters, return values and exceptions, and the tags should include appropriate comments that indicate the purpose of the inputs, the value returned, and the meaning of the exceptions.

- There must be a comment directly above each field that, in one line, states what the field is storing.
- There must be a comment either above or to the right of each non-field variable indicating what the variable is storing. Any comments placed to the right should be aligned so they start on the same column.
- There must be a comment above each loop that indicates the purpose of the loop. Ideally, the comment would consist of any preconditions (if they exist) and the subgoal for the loop iteration.
- Any code that is complicated should have a short comment either above it or aligned to the right that explains the logic of the code.

III. Program Testing (20% of your project grade)

tcs94_CSDS132_Testing4

You are to write a test report that indicates the *types* of tests needed to thoroughly test your project. The tests should demonstrate that all parts of your code behave correctly. Any unit of your program involving conditional statements will need tests that go through each branch of the execution. Any unit of your program involving loops will need tests that cover the "test 0, test 1, test many" and "test first, test middle, test last" guidelines. Your testing report should *not* list the actual tests and results.

You are to have a JUnit test class or classes that implement as many of the tests as you can. You should have comments, names, or other indicators in your JUnit tests that easily link the JUnit tests back to the testing report.

The testing report must be separate from the JUnit class. In most companies, the testing document will be written in a style that allows both programmers and non-programmers to read it and recognize whether all the needed test cases were included.

Note that you will not be able to (easily) test methods involving user input or screen output with JUnit. For these parts of your program, your testing report should indicate the specific tests you did to test these routines.

Hint 1: Make lots of helper methods for each of the different parts of the game. That will make it much easier to design tests for your game, and it will make the testing shorter! You can make the

methods public or private. In the lecture on Java reflection, you will learn how to write tests for private methods.

Hint 2: The JavaFX GUI components are not simple to create outside of a JavaFX application. As a result, it will be easier to write JUnit tests if you have the game mechanics done in helper methods that do not directly use GUI components. If fact, I **strongly recommend** that you have the game mechanics implemented in helper methods that do not use GUI components. For example, you can use a 2D array of ints to represent the game board and play the entire game on the 2D array of int. You can then, after each move, transfer the int values to the GUI. It is much easier to both code the mechanics and to test when working on arrays of int that arrays of JavaFX Button.

IV. Java Programming (55% of your grade)

For this project you will implement our version of the game 2048. Our version of the game will differ slightly from the original in that it will include "diagonal" moves.

Rules of the game:

The game is played on a grid of tiles, typically 4 tiles by 4 tiles. Each tile may be blank or contain an integer. The game starts with all tiles blank except one tile that contains the integer 1, chosen randomly.

A player makes a "move" by choosing a direction and all the non-black tiles slide in that direction. In the normal game, the possible directions are "up", "down", "right", and "left", but our game will also add the diagonal directions "up-left", "up-right", "down-left", and "down-right".

Here is how a slide "left" works. The rest act similarly. In each row, each non-blank tile, starting at the left most non-blank tile, is slid as far as it can go to the left. It stops when it either hits the edge of the board or another non-blank tile. However, if the tile "hits" another tile with the same value, the two tiles are "merged" by adding the sliding tile's value to the tile that it "hits".

For example, if the tiles in one row are {0, 1, 0, 2} (with 0 representing a blank), the result of sliding left should be {1, 2, 0, 0} as first the 1 slides left until it gets to the left most edge, and the 2 is then slid left until it hits the 1. The remaining two tiles in that row are now blank. If the tiles in one row are {0, 1, 0, 1}, the result of sliding left should be {2, 0, 0, 0} as the left 1 slides left until it gets to the edge of the board, and the second 1 slides left until it hits the 1 causing it to merge with the 1. Now the leftmost tile stores the 2 (the result of the merge) and the rest of the tiles in that row are blank. Since you slide the leftmost non-zero tile first, if a row has {0, 1, 1, 1, 1}, the result

of sliding left should be {2, 2, 0, 0, 0} and if the row has {0, 1, 1, 2, 0}, the result of sliding left should be {4, 0, 0, 0, 0}.

After the tiles are slid and merged in the desired direction, then one of the blank tiles on the board is chosen at random and a 1 placed on that tile.

What you must do:

Important: Read the instructions for the testing part of this project and especially the hints for testing a JavaFX application. You need to design your code so that it is easy to test. If you put off writing the testing code to the end of the project, you may need to rewrite your code to get testing to work.

Designing your program

You are to create a class called SlideGame. The SlideGame class extends the JavaFX Application class.

Create the board

You are to create a board by making a two-dimensional grid of Buttons. This simplest way to to that is to create a GridPane instance that will hold the buttons, and add the GridPane to the JavaFX Scene.

You can then create a 2-dimensional array of Buttons and add them to the GridPane using the add method of GridPane with the appropriate values so that the location of a button in the array corresponds to its location in the display grid. (You are welcome to change this, but you should have a two-dimensional array in your program somewhere that stores the current board situation.)

Create the initial board

All of the buttons should be blank, but choose one button at random and set its text to display the number 1.

Respond to button clicks

You should create an EventHandler for (some of) the buttons. Recall that an EventHandler has a method handle that is called every time the button is pressed. The handle method has a single parameter, ActionEvent e.

If a button on the left column is clicked, but not one of the corner buttons, every "tile" should slide horizontally to the left (see the slide rule explanation above).

If a button on the right column is clicked, but not one of the corner buttons, every "tile" should slide horizontally to the right.

If a button on the top row is clicked, but not one of the corner buttons, every "tile" should slide vertically to the top.

If a button on the bottom row is clicked, but not one of the corner buttons, every "tile" should slide vertically to the bottom.

If the button in the top right corner is clicked, every "tile" should slide diagonally up and to the right.

If the button in the top left corner is clicked, every "tile" should slide diagonally up and to the left.

If the button in the bottom right corner is clicked, every "tile" should slide diagonally down and to the right.

If the button in the bottom left corner is clicked, every "tile" should slide diagonally down and to the left.

Clicking any other button of the grid should have no affect.

If the click resulting in at least one game tile sliding, after the sliding is done, choose a blank "tile" at random and change that tile to now have a 1. If no tile moved as a result of the click, then do not add a 1 to the board.

Step 5: Add a main method

The SlideGame class should have a main method that launches the game. With a main method, you should be able to play your game by typing:

java SlideGame

in the Interactions pane (or outside of DrJava). You should also allow the user to enter two arguments for the board size. For example:
java SlideGame should start a game with a 4x4 grid, and
java SlideGame 4 6 should start a game with 4 rows and 6 columns. If the user enters something other than realistic numbers, your code should do something appropriate, but not crash.

Do *Not* adjust the game board size in the main method directly. Instead, pass the command line arguments to the launch method. Then in the start method, you can access them using the getParameters().getRaw() method of Application. Once the start method has the parameters, use them to set the game board dimensions.

Hint: Do not try to code everything at once! Code things one piece at atime and then test that method. For example, code just the slide left method on arrays of int, or code just the ability to display a grid of buttons. If you try to code multiple things at once, small errors could cascade into big errors.

Extra Credit: (10%)

If you decide to do the extra credit, *you must state in the Canvas comments what you did*. Don't make us hunt through your code to figure out what extra you did.

Make further improvements to the aesthetics and play of the game. Extra credit will be awarded to improvements that require coding challenge, a lot of work with the API, or creativity. *Exception, you may not change the behavior of the slide methods nor change the behavior of clicking the different buttons around the grid.*. **Note:** Do not look up online 2048 games and copy their code or style. This extra credit is for you to have some fun and be creative. It is an academic integrity offense if you copy code or ideas from online versions of the game.