Daniel Williamson

Cole Neubauer

Lab Report #4

1. **Problem Statement**

In this lab we needed to recreate a sudoku game. We were given classes to work with

along with a UML diagram to ensure our classes work together properly.

Required Classes:

Sudoku()

* Constructor with no Parameter

Sudoku(starting\_configuration : String)

* Constructor with a starting configuration of a String to set data set to the String

getSquare(row : int, col : int) : char

* gets square method to retrieve from the data set

setSquare(row : int, col : int, value : char) : void

* sets square method to set in the data set

isValid() : boolean

* checks if the board is valid after an entry

isSolved() : boolean

* checks if the board is solved

Requirements:

* isValid method must be 25 lines or each including methods it calls
* Classes must match class names in the provided UML
* Blank spaces represented with a space
* Must follow the 3 rules for a valid board
* Board must have no empty spaces to be solved

Assumptions:

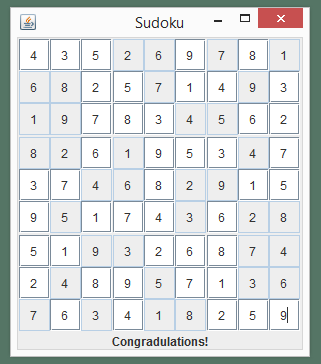
* The isValid function will have other classes called to make it work
* if isSolved returns True it will not check for changes after True

1. **Planning**

We began our planning by deciding whether we should use a single array or a 2D array. As you can tell with our code, we decided on a single array because we as a group had a better understanding of how to use said array. Once we made this decision we began by extracting the numbers, in the SudokuPlayer class file provided to us, and putting them in the correct order on the sudoku frame which was once again provided to us. After being successful we moved onto planning how to check for duplicates in each row and column. Our first thought was to use a loop to add up each row and column and set them equal to 45. However, we figured there were multiple ways to stump this. Like adding 5, 9 times would be equal to 45. So, we decided to check each square against each square for duplicates with multiple for loops with a nested if statement. Next, we began planning for the final check, the checking of the 3x3 square. We decided to keep it simple and made it so we only checked just the very first, top-left square. After being successful we created another method to go through all the 3x3 squares and check for duplicates. And lastly we have our isValid and isSolved to check for any of the duplicate methods returning false. If they don’t, the puzzle is solved!

1. **Implementation and Testing**

To implement the constructor that’s passed through a starting configuration we iterated through the strating\_configuration removing the new lines \n with the replace method replacing with “”. We then used the toCharArray method to turn the resulting string into a char array and set that to an instance variable to use throughout the class methods. Since we used a one dimensional array for getsquare we used the coordinates given and did appropriate math to find the correlating place in our array and having it use that index to return the char. We did the same with setsquare but just appended our array instead of returning a char. For is valid we created private methods to call inside of it to be able be less than 25 lines. We have rowDuplicate which iterates to each row in the board and then to each of the columns in that row to check for duplicates in a row, columnDuplicate iterates to each column and then to each row in that column comparing values to see if there is duplicates. Checking the squares was a harder part of this lab, what we did was find the way to iterate through a single square with checkSquare which has a starting index parameter. Then we made another method, checkAllSquares that iterates to the top left of every square that calls checkSquare and gives it that position index for it parameters. Finally we called isValid in isSolved to first see if the board follows the rules and then checked to see if there was any empty char ‘ ‘ in our array.



1. **Reflection**

This lab was very insightful into the world of arrays. It taught us how to manipulate them to do things we need them to do. Such as, iteration and indexing. In our project we used a single array. However, now that we look back on it, we should have gone with a 2D array to make checking for duplicates a tad easier. Besides that, we believe we had well informed variable names and a well orchestrated layout.