Jasque Saydyk and Joshua Pollock

Professor Elwakil, Melton, and McCarty

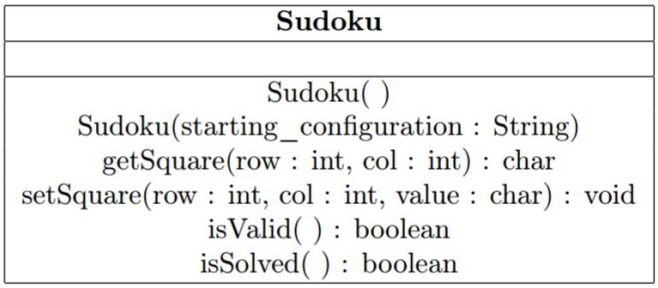
CS 136L Section 3801

28 February 2017

Lab 05 - Sudoku

1. Problem Statement

The problem of this lab is to create a program that when ran simulate a sudoku puzzle. We are given two class files that cannot be edited called “SudokuPlayer$GridListener.class” and “SudokuPlayer.class”. Our job is to create a class Sudoku, that is defined by the UML diagram:



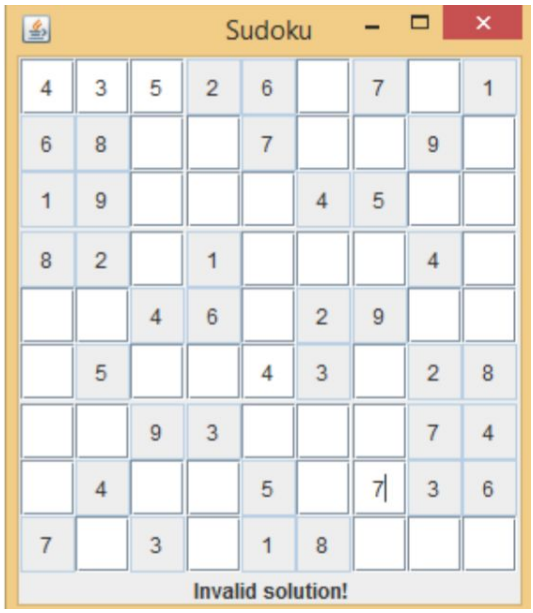
This Sudoku class will check if a sudoku board is valid, which is determined by given constraints. We will be doing this by using iteration, conditional statements, and arrays/list. There are many constraints and requirements for this lab. The constraints are:

- Each row in the sudoku board must not have any repeated numbers
- Each column on the board must not have any repeated numbers
- Each 3 by 3 squares on the board may not have any repeated number
- The isValid() method, and methods that it calls, must be 25 lines of code or less
- Any method, other than the ones given in the UML diagram, may not be public methods

Our features of the problem are defined by the UML and are:

- Sudoku()
- Sudoku(String starting_configuration)
- getSquare(int row, int col)
    - Returns a character
- setSquare(int row, int col, char value)
    - Void
- isValid()
    - Returns a boolean
- isSolved()
    - Returns a boolean

More private features may be implemented later on to fulfill the line limit constraint. An example of the final result of an invalid board is as shown:



2. Planning

For the planning of this project, we decided to use Java Lambda expressions as they were recently introduced to Java and are particularly suited to solving this type of problem involving extensive list manipulation. This is because Java Lambda expressions hail their origin from Scheme and the Functional programming mindset, and has the benefit of completing this task in a minimal number of lines of code and better allows for multiple cores of a computer to be utilized than traditional Object Oriented programing would. If this had failed, we would fall back on using multiple private methods to determine the validity of the columns, rows, and squares using multiple for-each loops.

The only other part of the project that we identified that we would need to do was to make a class that has a series of tests to show the code works in more cases than the test classes given.

3. Implementation and Testing

The implementation was far more difficult than we had initially planned getting into this. Although having experience with Scheme and Functional programming definitely helped in understanding how to get this to work, the two big problems we ran up against was getting the lists to be filtered properly, then figuring out a way to identify duplicates in the list. Figuring out that we were filtering the lists incorrectly took the longest amount of time to figure out, as it was the source of all of the problems we thought had, which is why we changed the 2D array from char to int. We finally corrected that issue after 5 hours of debugging, with the trusted adage of “When in doubt, print it out” ultimately saving use and the lambda approach to the project.

Following that was deciding how to identify when duplicates are in the list, which took us three hours to find answer to. Initially, we attempt to put in some function into the Lambda expression

to do just that, but we ultimately failed in that front. There maybe a solution in implementing an interface for that lambda expression, something that I was in the midst of researching, but I then discovered a far more elegant solution to the problem, a HashSet, or a similar equivalent in Python's words, a dictionary, which returns a true or false if the unique object was able to be put into the data structure.

After those two issues were solved, the rest of the project was straight-forward, with the only nuance that needed to be noted was that since the array was declared as an int array, conversions from string to int and char to int were necessary in the getters and setters of the program. The rest of this section explains and describes the implementation of the lambda expressions in the program.

Turning to the code now to see what the isValid() function does, I have to whole thing wrapped in a for loop to iterate through the rows of the table. Starting with the rows, put all the values in a row into an ArrayList. I do this because it makes the following operations significantly easier, as they were designed to work with lists.

Below that, I make sure that all the values in the row are between 1 to 9. As for the expression, I take the listRow, then I stream() it, which allows the following commands go through the list sequentially. I then apply a filter to that stream, telling it to pull out any -1's it sees. The -1's are the result of convert a blanks space in char to a numeric value, but the reason as to why eludes me, but my best guess in that in hexadecimal, that is the number for a space. The `filter(c -> c != -1)` is the first lambda expression, which states for each element in the array, if it does not equal -1, let it continue to the next expression.

The next section, `allMatch(c -> c >= 1 && c <= 9)` is where the second Lambda expression exists. This says that for each element c in the stream, apply the expression `c >= 1 && c <= 9` to it. If all of the elements match each other, that is the result is true, return true. Once a false is found, stop immediately and return false. The boolean inRangeRow variable holds the result.

Now for the anyRepeatsRow expression, it is the same, save the lambda expression for allMatch is different. What it is doing here is it is adding each element to a HashSet, which is a Hash table or if you are familiar with Python, a dictionary. One property of a dictionary in Python is that every key needs be unique. Java is no exception, and if you look at the documentation for HashSet, the add() method accepts a value to the stored and returns true if it is unique and is stored, and false if it's been rejected. Thus that is what determines if the final state of anyRepeatsRow.

Now, The lambda expression `allMatch(new HashSet<>()::add)` creates a new HashSet of the generic of the line, which is Integer. The "::" double colon operator is a shorthand to say (c -> hashSetVariable.add(c)), which means for each element, run the add method on it from HashSet. The last concern that needs to be addressed is that, since allMatch goes through all of the elements in the list and does the action that is in the Lambda expression, would it overwrite a false result if the last element is a positive result? This is a legitimate concern, but the allMatch

method is lazy in its computation and allows for short circuiting, which means that when it finds a false, it stops computing the rest of the elements of the list and returns that false result.

The rest of the method repeats the above for the Column and Square. For the square, to get the values, there is a private method, getSquare(), which is a giant switch tree that determines the location of every top left square and bottom right square of each 3x3 square in the Sudoku puzzle. It then uses a nested for loop to put the values into a list for use by the lambda functions. This is not the most efficient way of doing this problem, as the computationally fastest way to do it would be to make a series of arrays that held the index values of the squares, making this section of the program operate at constant time rather than $O(n^2)$ that is currently does. But seeing as the theoretical computational speed doesn't matter in this circumstance, it didn't warrant that much effort and the solution we have is what .

The last part of the isValid method is just a giant if statement saying that if any of the columns, rows, or squares failed the test, end it immediately, otherwise continue till true.

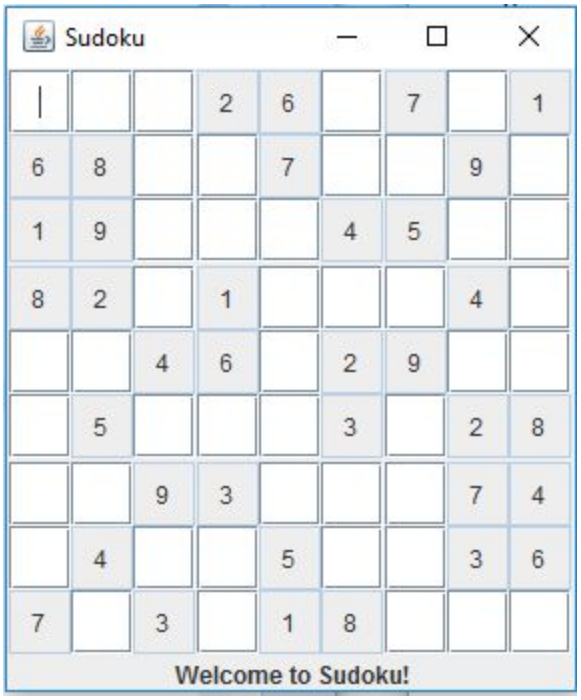
Just to discuss some miscellaneous items, the isValid function is technically 20 lines of code, as the lambda expressions are one line of code, though for readability sake, I've set them as three lines each, making a total of 32 lines of code. The lab instructors, when asked about the specifics on the 25 line limit on the function to determine whether the lambda expressions will be compacted into one line or retain their current form of 3 lines, did not give a definitive answer. However, we believe the 25 line code limit was in place to prevent programmers from using a bunch of for-each loops in the function and to introduce the idea of private functions to divide work in the class. Knowing that, we believe that our hubris in expanding the lambda functions to make them more readable, taking up more lines of code, is within the limits of the assignment.

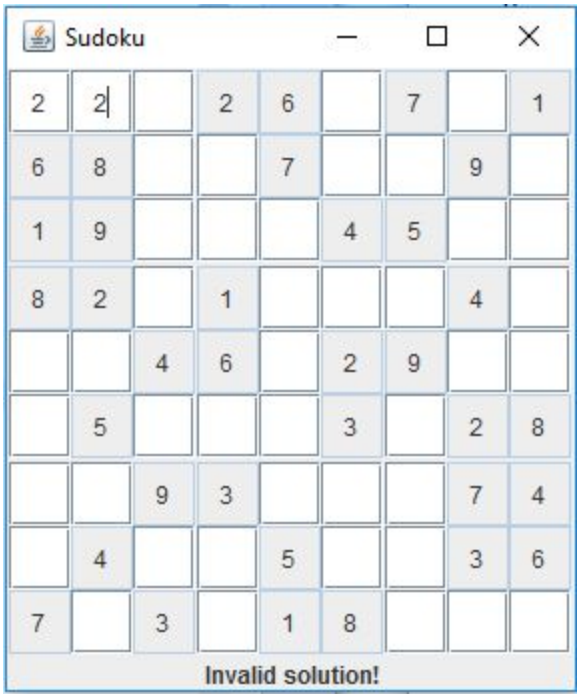
Below are results from the tests we made to ensure the program worked correctly and the results from the given classes.

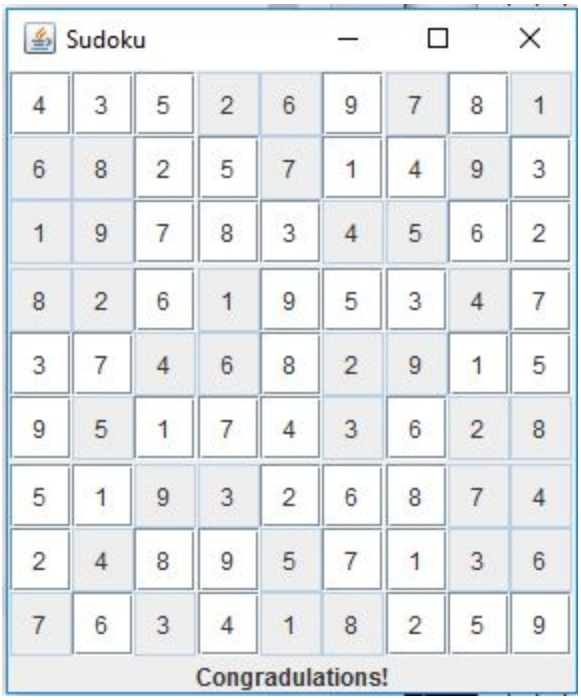
```
2|2|2|2|2|2|2|2|2|
2|2|2|2|2|2|2|2|2|
2|2|2|2|2|2|2|2|2|
2|2|2|2|2|2|2|2|2|
2|2|2|2|2|2|2|2|2|
2|2|2|2|2|2|2|2|2|
2|2|2|2|2|2|2|2|2|
2|2|2|2|2|2|2|2|2|
2|2|2|2|2|2|2|2|2|
Testing SetSquare 0,8

Testing Get Square in 9th column on first row
9

Testing isValid()
false

Testing isSolved()
false
```

```
-|-|-|2|6|-|7|-|1|
6|8|-|-|7|-|-|9|-|
1|9|-|-|-|4|5|-|-|
8|2|-|1|-|-|-|4|-|
-|-|4|6|-|2|9|-|-|
-|5|-|-|-|3|-|2|8|
-|-|9|3|-|-|-|7|4|
-|4|-|-|5|-|-|3|6|
7|-|3|-|1|8|-|-|-|
Testing SetSquare 0,8

Testing Get Square in 9th column on first row
9

Testing isValid()
false

Testing isSolved()
false
```

```
4|3|5|2|6|9|7|8|1|
6|8|2|5|7|1|4|9|3|
1|9|7|8|3|4|5|6|2|
8|2|6|1|9|5|3|4|7|
3|7|4|6|8|2|9|1|5|
9|5|1|7|4|3|6|2|8|
5|1|9|3|2|6|8|7|4|
2|4|8|9|5|7|1|3|6|
7|6|3|4|1|8|2|5|9|

Testing Get Square in 9th column on first row
1

Testing isValid()
true

Testing isSolved()
true
```







4. Reflection

From a productivity perspective, implementing lambda expressions was not good, as it consumed too much time to implement. If this was a product for a company that was not designing it to use multiple threads of a computer, this would have been more quickly done with 10 similar for each loops in their own private methods, with the isValid() method tying them all together. Despite that, this was a fun learning opportunity to try out a set of tools I've heard a lot about, and from this point forward, I will be using more lambda expressions in my code to

replace for loops whenever I can to get better adjusted with lambda expressions and the limitations.