Overview

The aim of this assignment was to produce a program that can be used by the user to check the validity of very simplified Sudoku puzzles. The program should be able to take a puzzle as an input, and check whether this puzzle is valid or not, in terms of both the format of the puzzle (number of rows and columns), the values within the puzzle and whether there are any duplicate values. There are two possible block types for these puzzles, being numerical puzzles (3 by 3 grid, for numbers 1 to 9) and textual puzzles (5 by 5 grid, for alphabetic characters from A to Y), and have the validation of all aspects work for both cases.

Design

I decided to split the program into 4 different classes:

Assignment2.java:
This class is where the main method of the program is stored, and where the user is asked what type of block they would like to input. After this, based on the response of the user, the program will create a selectedBlock object of the correct type, or end the program (if invalid input). After the object has been created, the intialise() method is called and then the checkStructure() method is called, both of which are located in the Block class.

Block.java:
I have decided to make this an abstract class, due to the use of a method within it, that contains no body, as well as not wanting to have this class being instantiable at all during the program's execution. If this was able to be instantiated, there would be no constructor available, and therefore would break the program.
This class contains all the main program code that makes up the logic of this program. These classes include the initalise() and checkStructure() method, which are further explained later in this design section. In short the initialise() method is what checks whether the format provided is valid, and if so, adds the values to a 2d array, which are checked later in the checkStructure() method if they are legal. There is also an abstract method of checkValsValidity() which is abstract so that it can be still called within the block class, while being overridden by it's same name counterparts within the subclasses.

NumBlock.java:
This is a subclass of the Block class that gets initialised when the user selects the number block option. This therefore sets the block size as 3 and sets the array length as 3 by 3 in the constructor. The rest of the logic works as normal in the block class, until reaching the section where the values are needed to be checked if they are in the correct range of 1 to 9, where the checkValsValidity() method is called through overriding, and if all valid, the value is input into the full 2D array, cellValues.

TextBlock.java:
This is a subclass of the Block class that gets initialised when the user selects the text block option. The block size is set as 5 and the array length as 5 by 5 in the constructor. The rest of the logic works as normal in the block class, until reaching the section where the values are needed to be checked if they are in the correct range of A to Y, where the checkValsValidity() method is called through overriding, and if all valid, the value is input into the full 2D array.

I decided to have both the NumBlock and TextBlock class to inherit from the Block class. Using this method (in contrast to having the TextBlock class inherit from the NumBlock class) allows the ability to add more different blocks in future use cases, such as characters from another language, without worrying about the program breaking or less seamless addition of a new block type. This is also easier to understand for another programmer when looking into how the logic of the program works.

To hold the actual values of the block inputted, I have decided to use character as the data type, for both the NumBlock and TextBlock instances. The reason for this is that it is possible to compare two characters using their ASCII values, so it is possible to state that 'C' is greater than 'A' as it has a greater ASCII value. Storing and integer as a character is also applicable, as the comparison operator will still work as required, as '1' is still

considered equal to '1' while not being considered equal to '6'. This allows a very efficient method of inheritance, without having to worry about the code being duplicated more times than necessary.

The UML diagram representing the relationships between these classes and the attributes and methods within these classes is shown below:

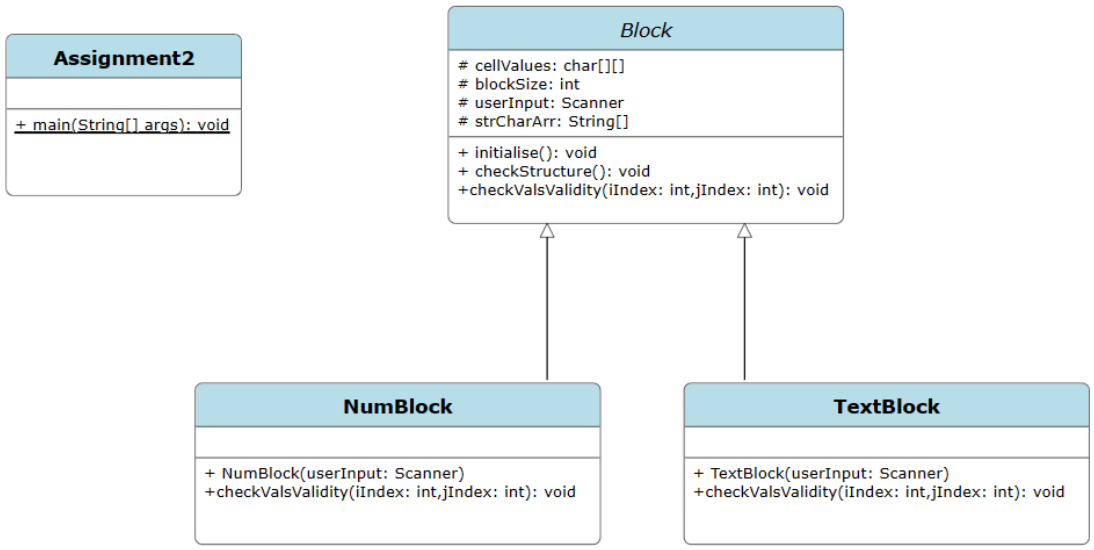

Figure 1: UML Diagram of Assignment2[1]

There are two methods in this program that are called within the main method, the initialise() method and the checkStructure() method, both of which are within the Block class.

The initialise() method is used to check if the format of the input is correct. First, the program checks if a next line has been input at all, and if not, returns the relevant error and stops the program. It takes the input line by line, and splits this into a separate one-dimensional array as a string using a comma as a delimiter. After this, the length of the array is checked if it is the required length. If not, an error is output, and the program is halted. After this, the value of the array is checked, whether it is within the correct range. This is done by calling the checkValsValidity() method, which due to method overriding, calls the method that is in the relevant subclass. This method trims the value being checked, saves this in the correct data type (int for NumBlock, char for TextBlock), and then checks if this value is within the correct range provided for that class. If not, an error is output, and the program halted.
If valid, then the value being saved is saved onto the cellValues 2D array. For the NumBlock class, this is saved by adding the number to the character '0', and converting to character, before saving this to the cellValues 2d array in the correct location (as parameters are passed into the method to indicate the location to be saved to). If the addition to the character '0' is not included, then integer is considered as an ASCII value, and converted to the equivalent character. For the numbers 1 to 9, this is not given a character, and so the character saved would be blank. This process occurs line by line, and if all aspects of the format is correct, then 'Valid format, ' is output, and the checkStructure() method is called.

The checkStructure() method is used to check whether the values input are all legally available, judging by sudoku rules. The values of the cellValues() array are iterated through, where each value is compared. The first comparison made is to check whether there are any recurrences of values in a row. If so, an error message is output, and the program is halted. After this, the values of the column are checked for any recurrances within the column. If there are, the loop continues, but a Boolean value of columnRepeated is given the value of true. This is because if there is ever an instance where there is still a repeated value in a row, this is given more precedence as an error over other possible errors. If one instance of the column being repeated is found, then the column values are not compared again. Within all of this, the values within the entire 2D array cellValues()

are saved onto a separate 1D array. This is not done if a recurrence is detected in the column (as an error will be output regardless after the iteration ends). After the iteration ends (and so rows are all valid), the value of columnRepeated is checked, and if true, an error is output, and the program is halted. If not, then the 1D array is searched through, checking for any instances where the same value appears more than once in the entire block. If this is the case, then an error is output, and the program is halted. If not, then all the validation has passed and the output would be “valid structure.”, and the program would be completed.

Testing

Firstly, I checked whether the sample data provided by stacscheck worked alongside my program, producing the result as is required. The image below shows the result of these tests, which were successful as was required:

```
Testing CS1002 Assignment 2
- Looking for submission in a directory called 'Solution': Already in it!
* BUILD TEST - build-all : pass
* ...
* ...
23 out of 23 tests passed
```

Figure 2: Stacscheck Tests all passed.

Having confirmed that the previous tests had passed, I had decided to attempt my own test, to ensure that all possible use cases for uses are determined for. These tests include those that are not already tested by Stacscheck, and so include those with inputs of normal inputs, extreme inputs and erroneous inputs. There was a total of 20 tests, and the table of tests have been included below:

For the Section of Program, it is as follows:

1. Numerical Puzzles.
2. Textual Puzzles.

Test Number	Section Of Program	Type of Test	Input Values	Expected Output	Final Output	Test Result
1	Menu	Normal	n	Enter cell values:	Enter cell values:	Pass
2	Menu	Normal	t	Enter cell values:	Enter cell values:	Pass
3	Menu	Erroneous	A	Invalid block type.	Invalid block type.	Pass
4	1	Normal	2,3,4 5,6,7 8,9,1	Valid format, valid structure.	Valid format, valid structure.	Pass
5	1	Extreme	7,3,8 5,1, 9 6, 4 ,2	Valid format, valid structure.	Valid format, valid structure.	Pass
6	1	Erroneous	7,3,8 5,1,9 6,4,82	Invalid format: value: 82.	Invalid format: value: 82.	Pass
7	1	Erroneous	7,3 5,1,9 6,4,2	Invalid format: number of cells in row.	Invalid format: number of cells in row.	Pass
8	1	Erroneous	1,2,1	Valid format,	Valid format,	Pass

			4,5,6 7,8,9	invalid structure: 1 repeated in row 1.	invalid structure: 1 repeated in row 1.	
9	1	Erroneous	1,2,3 4,5,6 1,8,9	Valid format, invalid structure: 1 repeated in column 1.	Valid format, invalid structure: 1 repeated in column 1.	Pass
10	1	Erroneous	1,2,3 4,5,6 7,1,9	Valid format, invalid structure: 1 repeated in block.	Valid format, invalid structure: 1 repeated in block.	Pass
11	1	Erroneous	1,2,1 4,5,6 7,5,9	Valid format, invalid structure: 1 repeated in row 1.	Valid format, invalid structure: 1 repeated in row 1.	Pass
12	2	Normal	B,C,D,E,F G,H,I,J,K L,M,N,O,P Q,R,S,T,U V,W,X,Y,A	Valid format, valid structure.	Valid format, valid structure.	Pass
13	2	Extreme	B,H,N,T,A G,M, S ,Y, V L,R,X,Q,W C,I,O, U ,F D, J ,P,E,K	Valid format, valid structure.	Valid format, valid structure.	Pass
14	2	Erroneous	A,B,C,D,Z F,G,H,I,J K,L,M,N,O P,Q,R,S,T U,V,W,X,Y	Invalid format: value: Z.	Invalid format: value: Z.	Pass
15	2	Erroneous	A,BC,C,D,E F,G,H,I,J K,L,M,N,O P,Q,R,S,T U,V,W,X,Y	Invalid format: value: BC.	Invalid format: value: BC.	Pass
16	2	Erroneous	A,B,C,D F,G,H,I,J K,L,M,N,O P,Q,R,S,T U,V,W,X,Y	Invalid format: number of cells in row.	Invalid format: number of cells in row.	Pass
17	2	Erroneous	A,B,C,D,A F,G,H,I,J K,L,M,N,O P,Q,R,S,T U,V,W,X,Y	Valid format, invalid structure: A repeated in row 1.	Valid format, invalid structure: A repeated in row 1.	Pass
18	2	Erroneous	A,B,C,D,E F,G,H,I,J K,L,M,N,O P,Q,R,S,T A,V,W,X,Y	Valid format, invalid structure: A repeated in column 1.	Valid format, invalid structure: A repeated in column 1.	Pass
19	2	Erroneous	A,B,C,D,E F,G,H,I,J,	Valid format, invalid	Valid format, invalid structure:	Pass

			K,L,M,N,O P,Q,R,S,T U,V,A,X,Y	structure: A repeated in block.	A repeated in block.	
20	2	Erroneous	A,B,C,A,E F,G,H,I,J K,L,M,N,O P,Q,R,S,T U,V,A,X,Y	Valid format, invalid structure: A repeated in row 1.	Valid format, invalid structure: A repeated in row 1.	Pass

Within my tests, I decided not to tests the functionality of my program that can check whether or not the number of rows provided is correct. This is because when I do this, the program appears to be waiting for the next input as explained in the assignment specification. However, this functionality is tested by Stacscheck, has passed the tests regarding this error case, and therefore should be eligible.

All the tests that I had assigned has passed, and so I believe that the development of the program has been completed, with the program working as intended. The provided testing should be sufficient to cover the multiple use cases that may occur during the usage of this program, with a reasonable amount of testing data that should cover these use cases. This testing therefore should provide an accurate assessment of how the program may be used in real-world environment.

Evaluation

As shown previously, all the stacscheck tests had passed, as well as the tests that I had performed myself. Including this, all of the sections have been completed, with the requisite of needing to incorporate inheritance in a valid method. I believe that any code reuse has been decreased to a substantially minimum value, and therefore completed the assignment given.

Conclusion

Overall, this assignment was enjoyable, with more planning involved within the design section. The incorporation of inheritance, and the plan to reuse as little code as possible was a refreshing challenge, involving the steps of decomposition and abstraction. This project also helped me to understand the importance of time management within a project, all of which are skills that can be incorporated elsewhere as required.

If given more time, there are a few things I would have done differently for this assignment. One such example would be incorporate the usage of a Hashset to be able to compare the values to check for whether they are valid, by using the hashset to have a time complexity of O(1) for lookup times . Another method I could have used to solve this issue (particularly with checking if the value appears twice within the entire block), was the Collections.frequency() method, which would decrease a number of lines of code, and also make the code easier to follow for external users.

References

[1] Used https://app.smartdraw.com/ for UML Class Diagram