COSC 1P03 Lab 1
Sep. 11–15, Fall 2023

Exercise 0

Estimated time: 10 min

1) Download and extract the **`Lab01`** folder from Sakai onto your desktop.
2) Create a folder called **Lab1_`Excercise`** in your `Lab 1` folder on your desktop.
3) Launch IntelliJ.
4) Create a project for the program:
   a. Click New Project on the Welcome screen **OR**
   b. Select **File | New | Project**.
   c. Select Java as your language and select SDK version from the **`Project SDK`** dropdown list. If you don’t have any SDK version yet, select add /download one from the list (version 8 preferably)
   d. Select Next and give the Project Name **`Sudoku`** and select the project location (Your folder **Lab1_Excercise**) and `Finish`**.**
   e. Right-click on the **src** folder in the left-hand side Project tool window and select **New | Package.** Name the new package **`com.sudoku`** and press `Enter`.
   f. Right-click on **`com.sudoku`** and select **New | Java Class**. Name the new class as **`Sudoku`** and press `Enter`.
5) Inside your Sudoku class type `main` and press `Enter` to select the template that inserts the `main()` method declaration.
   a. Call the `println()` method by
   b. Type: `System.out.println("Sudoku Main Test");`
   c. Save the program: Ctrl+S.
6) Click the ▶ symbol from the left side gutter and select ‘**Run Sudoku.main()**’ in the popup.

   a. The run tool window opens at the bottom of the screen and should print, in this case, “`Sudoku Main Test`”
7) **Add Brock Packages** to your project before you add anything.
   a. Select **File | Project Structure** (Ctrl+Shift+Alt+S)
   b. Select Library and click the **+ symbol** to add a new project library.
   c. Select **java** and navigate to your **Brock_Packages** folder and Click OK.
8) If everything is fine till this point, we are ready to start our first Exercise.

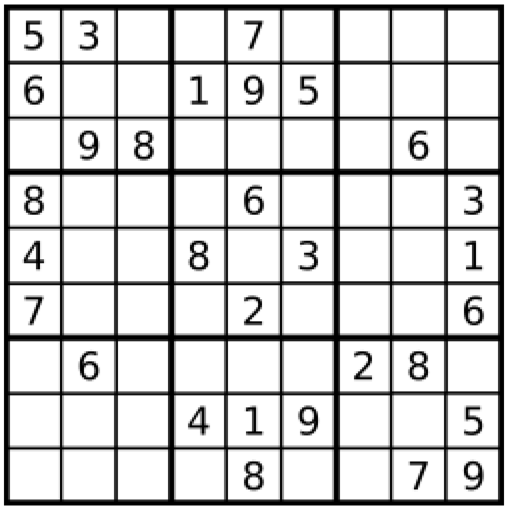
Note 1: The data files for the program are included as `valid.txt`, `invalid.txt` and `puzzle.txt`.
Note 2: There is a sample code in the `ASCIIDataFileSampleCode` folder inside your Lab01 folder. The sample project `ReadAndDisplayExampleProject` will help you to refresh your memory on how to read an `ASCIIDataFile` and how to display text on `ASCIIDisplayer`.

In this lab, we will consider array manipulation.

Sudoku Puzzle

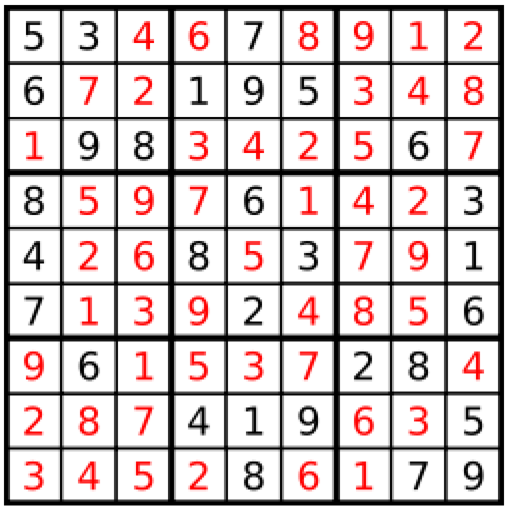
A Sudoku puzzle (https://en.wikipedia.org/wiki/Sudoku) is a 9x9 square into which the numbers from 1 through 9 are entered. A particular puzzle has some numbers prefilled. The object is to fill in the remaining spaces using the numbers 1 through 9 such that each row has the numbers from 1 through 9, each column has the numbers from 1 to 9 and each 3x3 quad has the numbers from 1 through 9.

For example, the puzzle:



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

has the solution:



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

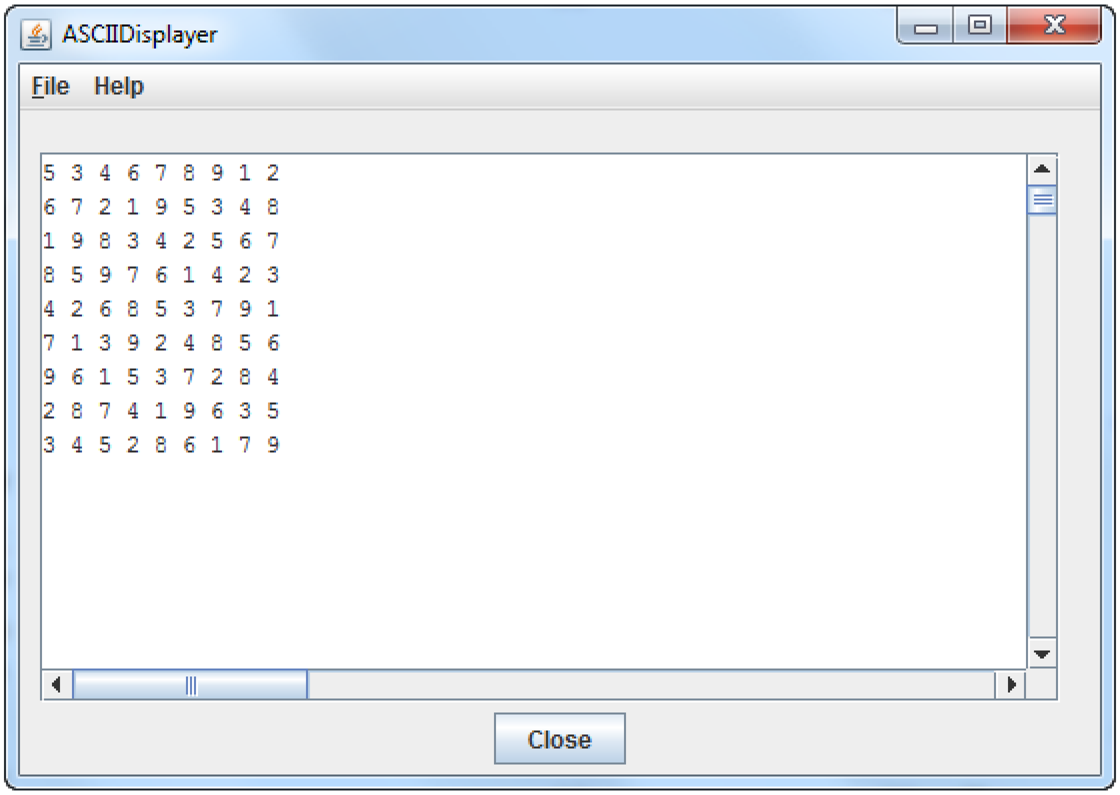
In this lab we will write a program that check to see if a proposed solution to a puzzle is correct. We will complete the program is a number of phases

Exercise 1
Data Representation

Estimated time: 15 minutes

A proposed solution to a puzzle is prepared as an `ASCIIDataFile` consisting of 9 lines of 9 tab delimited integers (1-9). The puzzle can be represented as a 9x9 array of `int`.

As a first phase, write the code to create the array, load it with the values from a file and then display the puzzle on an `ASCIIDisplayer` such as:



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

Write the code to create and load the puzzle as the method:

```
private void loadPuzzle ( ) {
```

and the code to display the puzzle on the displayer as:

```
private void displayPuzzle ( ) {
```

Exercise 2
Checking Rows

Estimated time: 25 minutes

As a method:

```
private void checkRows ( ) {
```

write the code to check that each row in the puzzle includes each of the numbers from 1 through 9. If a row is invalid it should write line(s) to the `ASCIIDisplayer` indicating the which number(s) are missing in the row.

We need to check if each of the rows contains each of the numbers from 1 to 9. If we had a helper method:

```
private boolean findInRow ( int checkFor, int inRow ) {
```

that returns `true` if row `inRow` contains the number `checkFor` we would have the beginning of a solution. Then the method could be called for each row checking for each number.

Write the helper method `findInRow` then write the method `checkRows` and integrate it into your solution.

Exercise 3
Checking Columns

Estimated time: 10 minutes

Checking columns is essentially the same as checking rows, only processing in column-major order. Write the helper method:

```
private boolean findInCol ( int checkFor, int inCol ) {
```

that returns `true` if column `inCol` contains the number `checkFor`, similar to `findInRow`. Then write a method:

```
private void checkColumns ( ) {
```

which checks the columns for each of the numbers like `checkRows`. Integrate the method into your solution.

Exercise 4
Checking Quadrants

Estimated time: 25 minutes

We want something similar to `checkRows` and `checkColumns` for quadrants. A quadrant is a 3x3 set of elements. Each quadrant is rooted at a particular (row,col) position within the array. A helper method:

```
   private boolean findInQuad ( int checkFor, int i, int j ) {
```

could check the 3x3 quadrant rooted at (i.e. top left corner at) `(i,j)` for the number `checkFor`. Write such a method.

The method:

```
   private void checkQuads ( ) {
```

could then use `findInQuad` to check each of the quadrants rooted at: `(0,0)`, `(0,3)`, `(0,6)`, `(3,0)`, `(3,3)`, `(3,6)`, `(6,0)`, `(6,3)`, and `(6,6)` for the numbers 1 through 9. Write this method and integrate it into your solution.