**CMPSC 111**
**Introduction to Computer Science I**
**Fall 2014**

**Lab 8 for Sections 03 and 04**
**30 October 2014**
**Due Thursday, 6 November by 2:30pm**

## Objectives

To enhance your experience with designing and implementing your own Java classes, including the completion of tasks such as picking the right instance variables and creating both the constructors and the methods. Additionally, to practice using compound `if/else` statements as part of a solution to a real-world problem involving a popular and well-known game.

## General Guidelines for Labs

- **Work on the Alden Hall computers.** If you want to work on a different machine, be sure to transfer your programs to the Alden machines and re-run them before submitting.
- **Update your repository often!** You should `add`, `commit`, and `push` your updated files each time you work on them. I will not grade your programs until the due date has passed.
- **Review the Honor Code policy.** You may discuss programs with others, but programs that are nearly identical to others will be taken as evidence of violating the Honor Code.

## Reading Assignment

To learn more about `if` statement and boolean expressions, review Sections 5.1–5.3; to recall material on constructing classes and methods, read Sections 4.1–4.5. Students who are not familiar with the Sudoku game are encouraged to examine `http://en.wikipedia.org/wiki/Sudoku`.

## Create a New Directory and Two Java Programs

In your own `cs111F2014-<your user name>` repository inside the `labs/` directory, create a directory called `lab8`. Type "`cd lab8`" to move to the new directory. Then, using `gvim`, you should create `SudokuChecker.java` and `SudokuCheckerMain.java` files that will store your solution to this laboratory assignment. Remember to use the Java program template from a previous lab!

## Implementing a Sudoku Checker

Sudoku is a logic-based placement puzzle. The aim of this puzzle is to enter a numerical digit from 1 through 9 in each cell of a 9x9 grid made up of 3x3 subgrids (called the "regions"), starting with various digits given in some cells (called the "givens"). Each row, column, and region must contain only one instance of each valid numeral.

For this lab, you are going to write a Sudoku validator for 4x4 grids made up of 2x2 regions instead of 9x9 grids made up of 3x3 regions. In contrast to the previously explained full grids, Sudoku puzzles that use 4x4 grids only use the numerical digits 1 through 4 instead of 1 through 9.

Here are two correct 4x4 Sudoku grids:

| | | | |
|---|---|---|---|
| 3 | 2 | 4 | 1 |
| 4 | 1 | 3 | 2 |
| 1 | 4 | 2 | 3 |
| 2 | 3 | 1 | 4 |

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 3 | 4 | 1 | 2 |
| 2 | 1 | 4 | 3 |
| 4 | 3 | 2 | 1 |

Notice how each row, column, and region has the individual numbers from 1 through 4 used only once. This means that the values used in each row, column, and region must add up to 10 because of the fact that $(1 + 2 + 3 + 4) = 10$. This is how your program will check the 4x4 Sudoku grids.

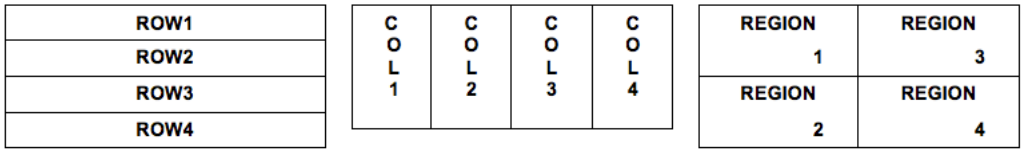
Your program should allow its user to enter their Sudoku grids row-by-row, requiring that they separate each of the four values by a space and that they hit ENTER at the end of the row.

When the user has entered all 16 values into the program, you should output validation checks for each region, row, and column. An example run of "`SudokuChecker`" can be seen below.

```
Welcome to the Sudoku Checker!

This program checks simple and small 4x4 Sudoku grids for correctness.
Each column, row, and 2x2 region contains the numbers 1 through 4 only once.

To check your Sudoku, enter your board one row at a time,
with each digit separated by a space.  Hit ENTER at the end of a row.

Enter Row 1: 3 2 4 1
Enter Row 2: 4 1 3 2
Enter Row 3: 1 4 2 3
Enter Row 4: 2 3 1 4

REG-1:GOOD
REG-2:GOOD
REG-3:GOOD
REG-4:GOOD

ROW-1:GOOD
ROW-2:GOOD
ROW-3:GOOD
ROW-4:GOOD

COL-1:GOOD
COL-2:GOOD
COL-3:GOOD
COL-4:GOOD

SUDOKU GRID IS VALID!
```

Your program does not have to check for the numbers 1 through 4 being used uniquely in each row, column, and region. In other words, don't worry about validating the user's input to make sure they only enter the digits 1, 2, 3, or 4 and only enter them once per row, column, and region. You simply need to check that each row, column, and region adds up to 10. This simplistic type of checking will, however, allow some bad Sudoku grids to be validated as good—this is acceptable.

Rows, columns, and regions are identified as found below:




### `SudokuCheckerMain` Class

To use the `SudokuChecker` class, follow this template for the `SudokuCheckerMain.java` file:

```
public class SudokuCheckerMain
{
     public static void main ( String args[] )
     {
          SudokuChecker checker = new SudokuChecker();
          // TODO: output the required welcome message
          checker.getGrid();
          checker.checkGrid();
     }
}
```

### `SudokuChecker` Class

The basic structure of `SudokuChecker.java` is as follows:

```
import java.util.Scanner;

public class SudokuChecker
{
     // TODO: put private data members here
     // TODO: put constructor here
     // TODO: put getGrid() here
     // TODO: put checkGrid() here
}
```

The UML diagram for `SudokuChecker` is:

| **SudokuChecker** |
|---|
| - w1 : Integer<br>- w2 : Integer<br>- w3 : Integer<br>- w4 : Integer<br>- x1 : Integer<br>- x2 : Integer<br>- x3 : Integer<br>- x4 : Integer<br>- y1 : Integer<br>- y2 : Integer<br>- y3 : Integer<br>- y4 : Integer<br>- z1 : Integer<br>- z2 : Integer<br>- z3 : Integer<br>- z4 : Integer |
| << *constructor* >> SudokuChecker( ) |
| + getGrid ( )<br>+ checkGrid ( ) |

This is a short description of what each data member represents and what each method does:

| | |
|---|---|
| `w1, w2, w3, w4`<br>`x1, x2, x3, x4`<br>`y1, y2, y3, y4`<br>`z1, z2, z3, z4` | Private data members that will store the numbers the user inputs. Variables `w1` through `w4` are for the first row, `x1` through `x4` are for the second row and so on. You should not have any other private data members in your class. |
| SudokuChecker() | This is the constructor. It should initialize each of the private data members to the value of 0, indicating that the user has not yet assigned a value to the data member. |
| getGrid() | This public method should read the Sudoku grid from the user row by row, using spaces between values in each row. |
| checkGrid() | This public method uses the private data members to test whether or not the given values are a valid Sudoku grid.<br>Your method does not have to check for the numbers 1 through 4 being used uniquely in each row, column, and region. In other words, don't worry about validating the user's input to make sure they only enter the digits 1, 2, 3, or 4 and only enter them once per row, column, and region.<br>When producing your output, you must first validate the regions, then the rows, and then the columns. Each region, row, and column validation should appear on a line by itself. |

### Points to Think About

Since we have not talked about programming concepts such as arrays yet—in fact, you should not use them to complete this assignment—you will need to input your 16 values into 16 separate variables. Some suggestions would be `w1`, `w2`, `w3`, and `w4` for the first row and then `x1`, `x2`, `x3`, and `x4` for the second row, and so on. Once you validate that one row is good or not good, the rest of the program should essentially be a matter of "copy and paste", replacing variables where appropriate. The most challenging part of this program is determining whether or not you have a good Sudoku configuration. One suggestion would be to keep track of a variable that counts the number of things that are invalid and, if that variable's value is 0 at the end of the program, you have a valid Sudoku board. Please see the instructor if you have questions about these matters.

### Additional Program Requirements

- Make sure your program prints your name, the lab number, and the date.
- Make sure your program contains the comment header with the Honor Code pledge, your name, lab number, date, and the purpose of the program.
- Make sure your program is documented properly, with descriptive and useful comments throughout your program whenever appropriate.
- Make sure your output is neat (e.g., no missing spaces and no typos) and that your program is neatly formatted (e.g., the indenting is correct).
- **You may not use array structures or loops for this assignment.**

### Required Deliverables

For this assignment you are invited to submit versions of the following deliverables through both the Bitbucket repository and in a printed and signed format.

1. Completed and properly formatted `SudokuChecker.java` and `SudokuCheckerMain.java` files.
2. An output file containing outputs obtained from running `SudokuCheckerMain` five times.

As you complete this step, you should make sure that you created a `lab8/` directory within your Git repository. Then, you can save all of the required deliverables in the `lab8/` directory—please see the course instructor or a teaching assistant if you are not able to create your directory properly.

In addition to turning in signed and printed copies of your code and output, share your source code and the output file with me through your Git repository by correctly using the "`git add`", "`git commit`", and "`git push`" commands. When you are done, please ensure that the Bitbucket Web site has a `lab8/` directory in your repository with the two Java files in the list of deliverables and the `output` file. Please see the instructor if you have questions about assignment submission.

In adherence to the Honor Code, students should complete this assignment on an individual basis. While it is appropriate for students in this class to have high-level conversations about the assignment, it is necessary to distinguish carefully between the student who discusses the principles underlying a problem with others and the student who produces assignments that are identical to, or merely variations on, someone else's work. Deliverables that are nearly identical to the work of others will be taken as evidence of violating the Honor Code.