# *Programming II (420-B20-HR)*

# *Assignment 2 – Arrays*

Date assigned: Tues. March 15, 2016

Test Plan, Class Diagram due: **Wed. March 23, 2016 in lab**

Phase I due (CLI Sudoku): **Wed. March 30, 2016 in lab**

Phase II due (Frame Sudoku): **Fri. April 8, 2016**

**Learning Objectives**

Upon successful completion of this assignment, the student will be able to:

1. Design test cases from requirements
2. Design a frame from requirements.
3. Design a class hierarchy from requirements.
4. Use two dimensional arrays.
5. Read from a text file.
6. Use various Swing UI methods

**To be handed in:**

1. A test plan and class diagram should be handed in for review during the lab on Wed, Mar 23. The test plan should be a numbered list of test scenarios that need to be tested. The files, the user interface, and the game logic all need to be considered in the test plan.
2. A Java project called ***username*\_B20\_A02\_PhaseI** should be zipped and uploaded to Moodle on Wed. March 30, 2016, for review during the lab. This should contain the Sudoku game working through the command line interface, without the frame.
3. The final Java project (with frame) called ***username\_*B20\_A02\_Sudoku** should be zipped and uploaded to **Moodle** on Fri. Apr. 8, 2016 andshould include the following in addition to the Java **src** folder:
   1. ***username\_*B20\_A02\_Test\_Plans.docx** containing the updated test plans for the system.
   2. A self-assessment for the assignment. (Use the template from **Moodle**.)

**Programming Style:**

Marks will be deducted for poor style. The following criteria will be used:

* All Java source files must be formatted using the Eclipse formatter. (Right-click and select **Source🡪 Format**.),
* All variables, classes, objects and methods must be named according to Java naming standards.
* Methods, objects and instance variables must be given an appropriate visibility (private or protected for instance variables and objects; private, protected or public for methods.)
* Complex code must be documented with an explanation.

**Organization:**

5 marks of the assignment mark will be for organization. The self-assessment must be completed and included in the assignment folder and the assignment must be:

* + handed in to the correct location,
  + be properly named
  + be complete according to the assignment specifications

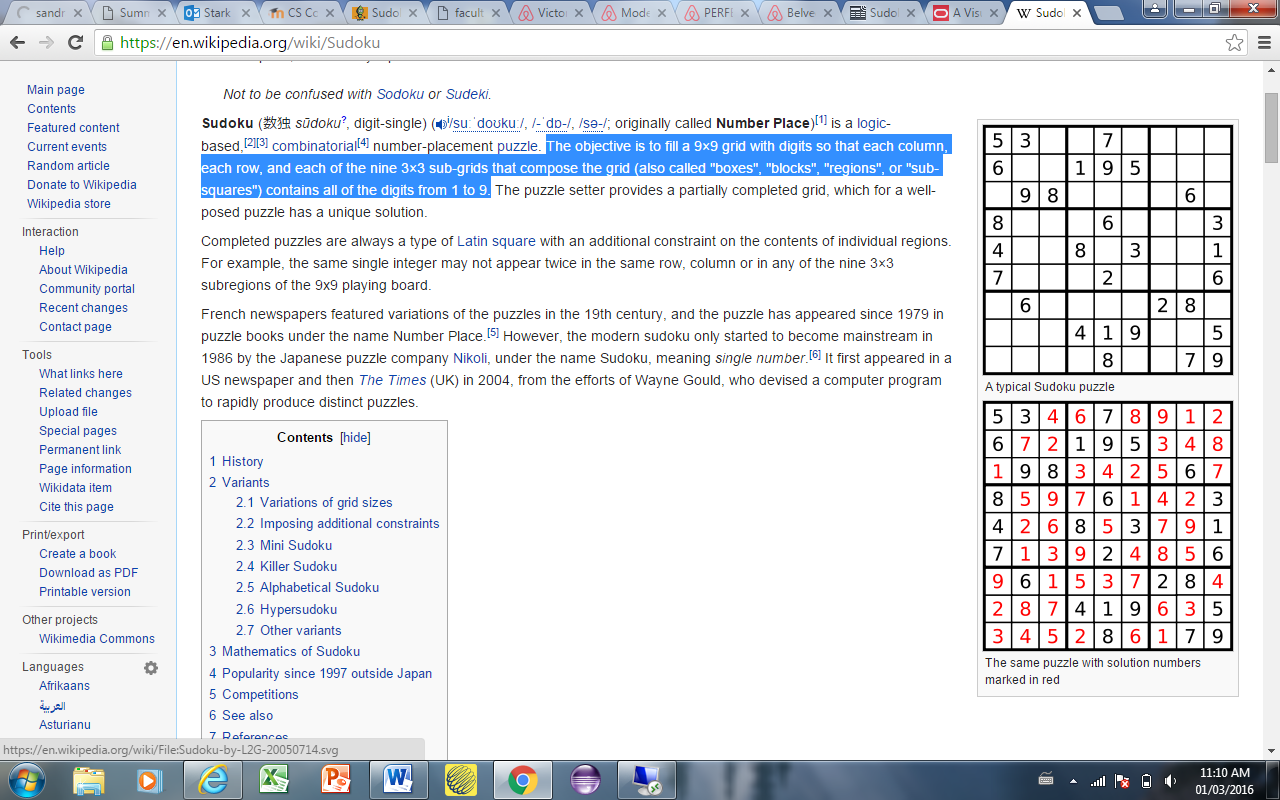
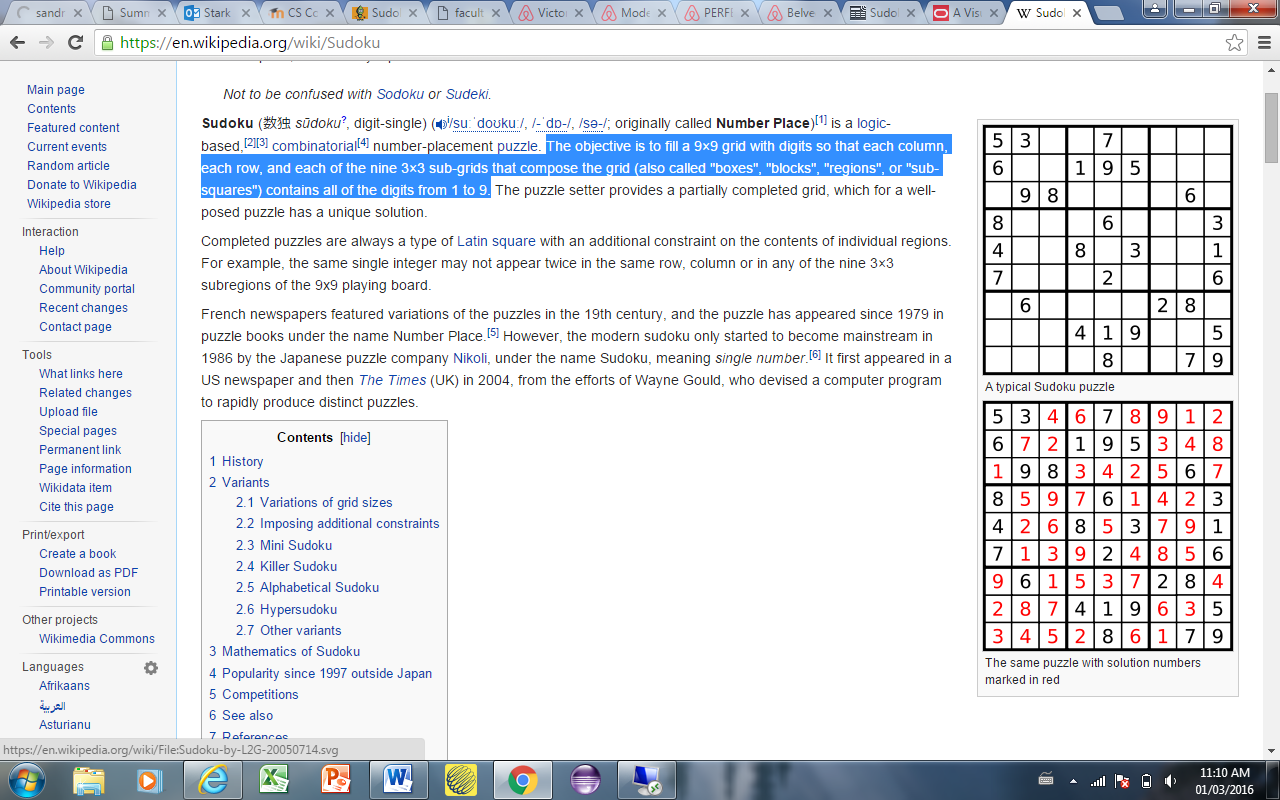
***Marking Scheme:***

|  |  |
| --- | --- |
|  | **Out of** |
| Test Case Plan – includes tests for the CLI, the Frame, the file and the game. | 15 |
| Class Diagram – all classes and methods included; correct use of access. | 10 |
| SudokuInterface java code – can enter a filename; board is displayed after each move; can enter a move and have it validated with an error message; can undo last move; can quit the game at any time; | 25 |
| SudokuGame java code – code divided correctly between classes; constructors, accessors, mutators; two dimensional array used; read in board from file; illegal move identified; undo last move;. | 45 |
| SudokiFrame java code - all frame components work; can enter a file name; can enter a move and have it validated with a pop-up error message; can undo last move; exit functionality; | 35 |
| Correct execution against requirements; thoroughly tested against test cases. | 15 |
| Organization | 5 |
| **Total** | **150** |

# Problem Specification

Sudoku is a logic based, number placement puzzle. The board is a 9 x 9 grid with 81 squares. Some of the squares are already filled in at the start of the puzzle. The player must fill in the rest of the squares by observing these 3 rules.

1. You must place the numbers 1 – 9 in each row without repeating a number.
2. You must place the numbers 1 – 9 in each column without repeating a number.
3. You must place the numbers 1 – 9 in each of the marked 3 x 3 boxes without repeating a number.



Phase 1 of your program will include the Sudoku game being played from a command line interface. Phase 2 will include the Sudoku game being played from a frame.

There are lots of websites with puzzles with solutions, such as <http://www.sudokuessentials.com/easy_sudoku.html>

**Phase 1** of your program must include the following functionality:

1. The initial Sudoku puzzle board should be read in from a file. The ~ is used to delimit the fields. An asterix (\*) indicates a blank square in the puzzle. A sample sudoku.txt file for the puzzle above is provided:

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~

1. The player should be prompted for the filename at the start of the game. If no filename is entered, the file sudoko.txt should be used as the default value.
2. The board should be displayed to the user. To play, the player enters a square number and the value to put in that square. For example, if the player wants to enter a 4 in row 1 column 3, then they should specify 1,3 and the value 4. It is up to you to decide how to prompt the user for the input. Once the move has been accepted, the program should display the updated board.
3. Your program must validate input. The user should only be able to enter the numbers 1 – 9 as a move, a row or a column.
4. Your program should identify illegal moves. If the player enters a number that is already in either that row, column or 3 by 3 square, then the user should be informed of the error and the move should not be made on the board.
5. Your program does not need to validate whether a move is correct. For example, if the value 1 is entered in row 1 column 3, it is not illegal, but it is not correct. In this case, the player will eventually lose the game because they have made a mistake.
6. The player can only modify empty squares. Once there is a number in a square (either because it was there to start with, or because of a move the player made), then that square cannot be change.
7. Your program should allow the player to undo the last move. Only the last move can be undone. If the player undid their last move, and then tried to undo the move again before making another move, then this should not be allowed.
8. The player should be able to quit the game at any time.
9. **You must use a two-dimensional array to store the game board.**
10. You must have a **SudokuGame** class and a **SudokuInterface** class. The **SudokuInterface** class will be your command line interface for Phase 1.

A sample run of the program:

Welcome to Heritage Sudoku.

Please enter the filename for your puzzle: easySudoku.txt

easySukoku.txt does not exist.

Please enter the filename for your puzzle: mediumSudoku.txt

mediumSudoku.txt does not have the correct file format.

Please enter the filename for your puzzle: hardSudoku.txt

Type Q at any time to exit the game. Type U at any time to undo your last move

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

Enter square number (row, column) -> 1,3

Enter value: 4

5 3 4 | \* 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

Enter square number (row, column) -> 8,8

Enter value: 3

5 3 4 | \* 7 \* | \* \* \*

6 \* \* | 1 9 5 | \* \* \*

\* 9 8 | \* \* \* | \* 6 \*

---------------------

8 \* \* | \* 6 \* | \* \* 3

4 \* \* | 8 \* 3 | \* \* 1

7 \* \* | \* 2 \* | \* \* 6

---------------------

\* 6 \* | \* \* \* | 2 8 \*

\* \* \* | 4 1 9 | \* 3 5

\* \* \* | \* 8 \* | \* 7 9

Enter square number (row, column) -> 8,10

Invalid column number. Please try again.

Enter square number (row, column) -> 0,5

Invalid row number. Please try again.

Enter square number (row, column) -> 8,9

Invalid move. That location already has a value. Please try again.

Enter square number (row, column) -> 1,4

Enter value: 7

Invalid move. There is already a 7 in that row/column/square. Please try again.

5 3 4 | \* 7 \* | \* \* \*

6 \* \* | 1 9 5 | \* \* \*

\* 9 8 | \* \* \* | \* 6 \*

---------------------

8 \* \* | \* 6 \* | \* \* 3

4 \* \* | 8 \* 3 | \* \* 1

7 \* \* | \* 2 \* | \* \* 6

---------------------

\* 6 \* | \* \* \* | 2 8 \*

\* \* \* | 4 1 9 | \* 3 5

\* \* \* | \* 8 \* | \* 7 9

**Phase 2** of your program must include the following functionality:

1. The file functionality remains the same, but rather than prompting the user for a filename, you will need to design the filename input into your frame.
2. The **SudokuGame** class should not change in Phase 2. The game rules remain the same. You must have the same **SudokuGame** class from Phase 1, and a **SudokuFrame** class.
3. It is up to you to design your frame. You can have the user enter their move in a separate “move” text box, or you can have the user enter their move right in the square, or some other design.
4. When an error in input is detected (either because of invalid input (not 1 – 9) or because of an illegal move), a pop-up window indicating the error message should be displayed.
5. A Menu should exist that has the following options:
6. Undo last move
7. Help – this should display a box that has instructions for the game.
8. About – this should display a box with you’re the title of your program, your name, year, and Heritage College
9. Exit – this should quit the application

### Your tasks:

1. Create test plans for testing the **SudokuGame** class, the **SudokuInterface** class and the **SudokuFrame** class. This should be a numbered list of the scenarios to be tested. The scenarios should describe in a sentence what to test and the expected result.
2. Design the system. Draw a class diagram that includes all classes involved, including your frame class. .
3. Get approval of your frame design and test plans before you start coding.
4. Code the **SudokuGame** and **SudokuInterface** classes according to the specifications. Test as you go.
5. Completely test your program using the test plans you developed in the first step.
6. Hand in Phase 1.
7. Design the frame. This can be a hand drawn design.
8. Code the **SudokuFrame** class according to the specifications. Test as you go.
9. Completely test your program using the test plans you developed in the first step.
10. Document your code with comments; format your code.
11. Complete the self-assessment.