COSC 1P03 Assignment 4
Puzzling!

***Due: Mar. 20, 2013 @ 12:00 pm noon** (late date Mar. 23)*

In preparation for this assignment, create a folder called `Assign4` with two subfolders: `PartA` and `PartB` to contain, respectively, your projects for Part A and B below. The data files for this assignment can be downloaded from URL: `http://www.cosc.brocku.ca/Offerings/1P03/A4Files.zip`.

Sudoku

Sudoku is a puzzle in which the solver attempts to fill a 9x9 grid of numbers so that each row, column and square contains all of the numbers 1 through 9. Below is an example. On the left is the puzzle, some squares which already contain numbers. The solver will need to fill in the blanks, until all squares have been filled while adhering to the requirement above. On the right is a solution.

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

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

Humans would solve the puzzle by applying deterministic rules. In general “this number can only go here because putting it anywhere else would violate some rule”. Nine such rules have been defined, and seasoned players apply them intuitively.

Another solution—often referred to as a brute force approach—fills all empty cells with numbers, and then check to see if the “solution” is valid. If not, change a number and check again, going through all possible combinations until a correct solution is found. Given enough time, such an approach will find a solution.

Assignment

A puzzle `P` can be represented as a 9x9 array of integers. Each element is either `0`—representing a “blank” square for which a number has to be filled in—or a number between `1` and `9`—representing a fixed square which cannot be changed.

In any puzzle there are some number (at most 81, but usually fewer) of blank squares for which numbers must be selected to solve the problem. It is easy to count the number of such squares in `P`, call it `s`. Instead of working with the original 2-d array, we could represent a possible solution as a one-dimensional array `A` of length `s`.

For example, suppose P is a 4x4 puzzle (numbers 1-4):

	2		
		4	
4	3		
		1	

The number of blank spaces is 11 and the solution array A would have eleven elements. A possible "solution" in A might be:

1	**2**	**3**	**4**	**1**	**2**	**3**	**4**	**1**	**2**	**3**

representing a "solution" of the puzzle P as:

1	2	**2**	**3**
4	**1**	4	**2**
4	3	**3**	**4**
1	**2**	1	**3**

(Essentially a row-major—row-by-row—filling of the blank spaces in P with the consecutive values from A.) The "solution" to the puzzle (in P) could now be checked for correctness (note: it isn't correct).

We can solve a puzzle by generating all possible combinations of digits 1-9 into A (starting with all 1s and finishing with all 9s) and, each time we have a combination generated, substitute it into P and check to see if the "solution" in valid. When we have a valid solution, we can quit.

We will thus need both a puzzle solver—which generates and checks the combinations—and a solution checker—which determines if a "solution" is valid.

Part A: Solution Validator

Write a program which will read in a Sudoku puzzle solution P and determine if P is a valid solution. It should print, to an `ASCIIDisplayer`, the puzzle solution P and then a message indicating if it is valid or invalid.

Include a method called:

```
private boolean valid ( int[][] P )
```

which will return `true` if P is a valid solution to a Sudoku puzzle and `false` otherwise. The method `valid` must check to ensure each column, row and square contains each of the numbers 1 to 9.

In the `A4Files.zip` folder are files `valid.txt` and `invalid.txt` being a valid and an invalid puzzle solution, respectively.

Part B: Puzzle Solver

Write a program which will read a Sudoku puzzle P and solve it using the brute-force method. You can reuse as much of Part A (especially the method `valid`) as appropriate, but make a copy of the code so you have both parts for submission. Part B should:

1. read the puzzle `P`
2. determine the number of "blank" spaces and create a right-sized array `A` to hold the combinations.
3. execute a recursive method `generate` which generates combinations until it finds one that solves the puzzle (as validated by the method `valid` above).
4. display the solution to the puzzle to an `ASCIIDisplayer`.

The recursive method:

```
private boolean generate ( int i )
```

solves the puzzle `P` (an instance variable), returning `true` when it has done so. Each recursive call fills in a value for element `i` of `A` (attempting numbers `1-9` in sequence) and uses recursion to fill in element `i+1` (and so on). When the method is called to fill in position `n` (being the length of `A`), a complete combination exists in `A`. It substitutes the values from `A` into the "blank" spots in `P` and calls `valid` to determine it the combination represents a valid solution, returning `true` if so and `false` if not. At position `i`, after the recursion to position `i+1` returns, if a valid solution was found it returns with `true`. Otherwise it tries the next value (i.e. of `1-9`) in sequence. If there are no more values to try (i.e. already tried `9`), it returns `false`.

In the `A4Files.zip` folder are files `easy.txt` and `hard.txt` being a puzzles that are easy (i.e. won't take too long to solve) and hard (will take some time to solve). Note: the brute-force method is very inefficient ($O(10^n)$) so use the easy puzzle for testing until you believe you have a working solution.

Submission:

Details regarding preparation and submission of assignments in COSC 1P03 are found on the COSC 1P03 website at URL: `http://www.cosc.brocku.ca/Offerings/1P03/AssignGuide.pdf`. This document includes a discussion of assignment preparation, programming standards, evaluation criteria and academic conduct (including styles for citation) in addition to the detailed assignment submission process copied below. **Part of the marks for the assignment will be awarded for programming standards.**

To prepare and submit the assignment electronically from the lab, follow the procedure below:

1. Ensure your folder (say `Assign4`) for the assignment is stored on your `Z:` drive.
2. Print (to `PDFCreator`) the `.java` file of your classes for Part A and Part B using the names *`classname`*`.pdf` where *`classname`* is the name of the class. Save the `.pdf` files at the **top level** of the project folder (i.e. directly within `Assign4`).
3. Run your program for Part A using both files `valid.txt` and `invalid.txt` and save the images of the `ASCIIDisplayer` (`File/Save Image of Window`…) as `PartAValid.pdf` and `PartAInvalid.pdf`, respectively, at the **top level** (i.e. in the `Assign4` folder).
4. Run your program for Part B using the file `hard.txt` and save the image of the `ASCIIDisplayer` (`File/Save Image of Window`…) as `PartB.pdf` at the **top level** (i.e. in the `Assign4` folder).
5. Run PuTTY by selecting `PuTTY` under `All Programs` in the `Start` menu.
6. Double-click `sandcastle` in the `Load, save` … entry.

7. Enter your Brock userid and press the `Enter` key.
8. Once you have the `sandcastle%` prompt, navigate to your project directory for your assignment (say `Assign4`).
   Here are a few useful commands (press `Enter` after typing the command):

`ls –l`	- list files in current directory
`cd <directory name>`	- changes to the specified subdirectory (note, do not include the <>) e.g. `cd Assign4`
`cd ..`	- go up 1 directory level

   Note: If your file or folder names include spaces or special characters, you have to enclose the name in quotes, e.g. `cd "COSC 1P03"`.
9. Once you have confirmed you are in the correct project directory, type the command `submit1p03` and follow the instructions. It is important to note that the script will copy everything from the current directory and its subfolders to the 1P03 electronic drop box. It is important you are in the correct directory when you run the script. The script will confirm what you have submitted.
10. Log off sandcastle by typing `logout`.

For help in submitting an assignment from home see the COSC Help Center at URL: `http://www.cosc.brocku.ca/help/esubmit`.

DrJava

The folder from which you do the electronic submission should contain the project folder, including all files (including those supplied) relevant to the project—the `.java` and `.class` files for the assignment and `.pdf` files for program listings and output.

Other platforms

If you are using an IDE other than DrJava to prepare your assignment at home, you must copy your code into DrJava to create new project(s) and then compile and run and prepare the submission as above. Your electronic submission must only include DrJava project folders and the `.pdf` files as described.