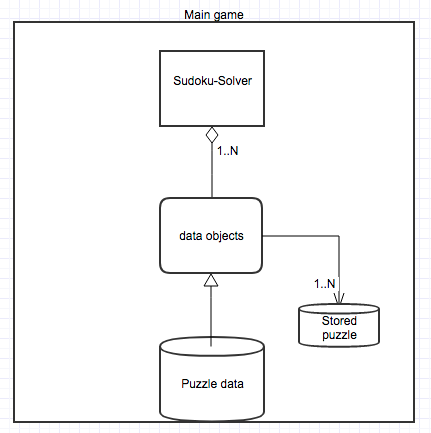
Design Document: Sudoku Solver

By: Sarah Kazi, Zaynab Almoujahed, Kayla Condrella

**Purpose:** The purpose of the project developed is to create an empty Sudoku grid that can then be loaded with a sample puzzle and played. The Sudoku Solver implements helper buttons that let the user know what values are valid and can also provide hints if needed. The purpose of the project is to essentially design a solver for Sudoku, and to allow the user to use the hints to have it solve the puzzle for them. The project can also save a current puzzle’s state to a file, which can then later be loaded at another time. The project will be for users who want to play Sudoku.

The project uses algorithms to resolve cells one at a time until the entire board is solved and also presents a panel on the side to let the user know whether they are currently using the eraser, a helper button, or what their candidates for a specific button is.

 **High Level Entities:**

There are four major high level entities in the project. The overall main game contains the Sudoku solver object, which holds all the contents needed to play the game.

Data objects include all the data we have to play the game, which includes arraylist of buttons, the panels of information we display.

We have the puzzle data which goes into the data objects and loads it onto the Sudoku solver GUI.

We also have information going into a stored puzzle data file, which takes the information on the GUI and stores it in a file.

**Low level design:**

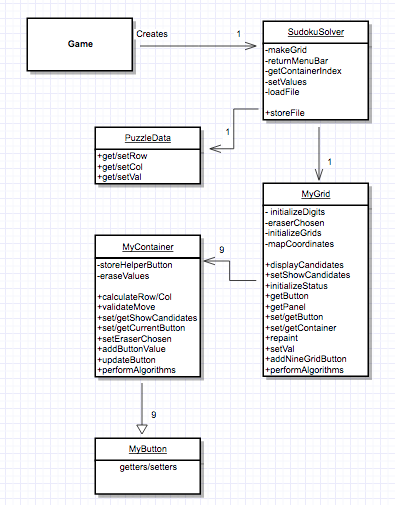
In the Sudoku Solver we have A few low level entities. We have the MyGrid, PuzzleData, MyContainer, and the overall SudokuSolver class.

SudokuSolver contains the creation of the JFrame and initializes the menubar which will set up any actionListeners called in the menu.

PuzzleData is the class that groups the row, col, and val together for our data files.

MyGrid initializes the whole 9x9 grid and calls the instance of nine MyContainers, each which are 3x3 boards. MyGrid calls all the algorithms that will be sent to each container. It also holds the panels that will show us the status and holds the digits at the side that the user will click.

MyContainer holds each 3x3 board where all the action is propagated down to.



**Benefits, assumptions, risks/issues:**

The benefits of doing our way is that most of the Java swing items we use such as container, button, gridlayout has been extended in some way in its own class, myGrid, myButton, myContainer. This way adding any extra functionality to the button or container is easy to implement. We also propagate the actions all the way from the SudokuSolver class down to each button to the myContainer, this way if we want to add another algorithm or add more to the JFrame it can all be done in SudokuSolver and propagated down. The Layout helps make it very simple as we have an underlying 9x9 grid that isn’t seen but keeps track of the 81 buttons, while each 3x3 grid has its own independent functionality.

Some risks/issues is that the separation of the classes can be confusing as an action listener that occurs in the menu bar has to be sent all the way down to the myContainer, so it’s hard to repeat this over and over again.