Sudoku Challenge

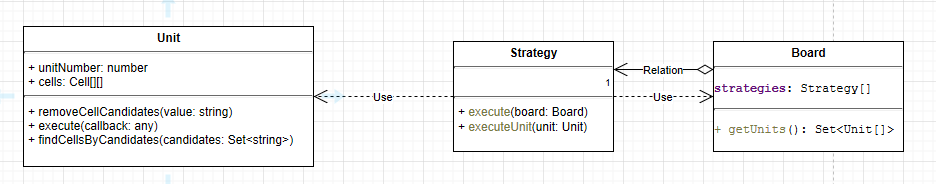
# Overview

This document does not go through the requirements and identify them. It just provide a high level architecture to the solution. The document answer the questions mentioned in the challenge.

# Components

## Overview

The system consists of a Board, Strategy and Unit components. These components uses each other without having any knowledge about each other implementation details or sub classes. This make it easy to add new Units, Strategy or Board components without effecting the other component or need to change system implementation.



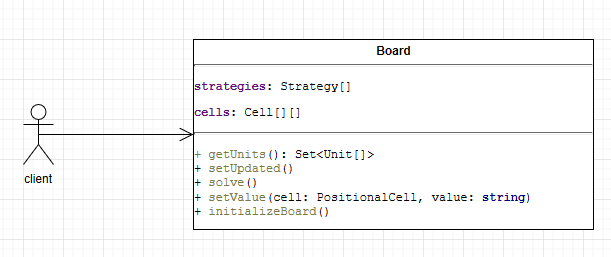
Client sees only the Board and use it to identify and solve Sudoku problem.

## Board component

### Overview

Board components identifies the strategies needed to solve the board, and types of units (row, column, box or etc) used by the board. It also encapsulates board cells and how they are stored.

### Details



setUpdate method is use by board cells to inform the board that some cell had been change. Du to this change Board will run another cycle trying to solve Sudoku.

solve method runs the strategies defined into the strategies array. After that it checks if any of the cells had been change. If that is the case, it runs another time.

The most important thing in setValue method is its cell parameter type. This type is using PositionalCell which means it can be used regardless of how the board is structured, for example in the case of samurai Sudoku. PositionalCell will allow Unit components to access board cells regardless of the Board structure and how these cells are defined. The PositionalCell can be extended to provide that capability. For example it can be extended to SamuraiCell to address samurai board.

Notice in the current implementation PsoitionalCell is not use, but that is defined in the code as TODO and fixed in this architecture document.

getUnits is how the board supply its units to strategy.

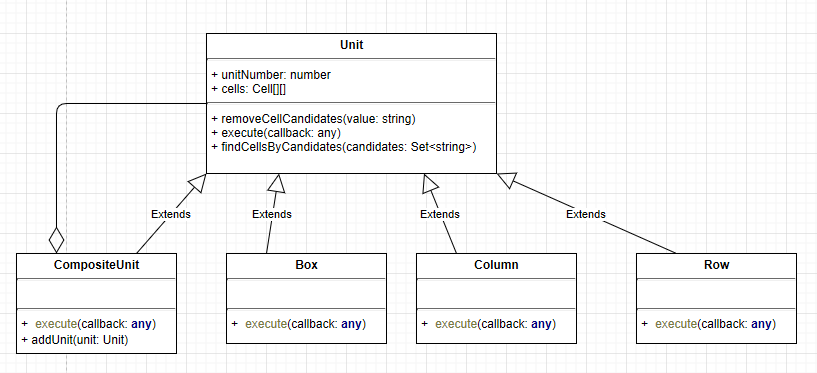
## Unit components

### Overview

These components are used by strategy components to access board cells. They interact with each other by using PositionalCell component. This is how the system encapsulate board structure.

When the board got created it generated its needed units. This method is very important to provide the ability to extend the system to provide different Sudoku boards without the need to change strategy implementation.

### Details



The most important method is the execute method. This allows strategy to execute needed implementation without the need to know about data is structured or define in each of these units. This is fulling functional programming pattern which is a simplified by the use of lambda functions.

The exiting usage of the execute method in the provided code uses row, column and cell instead of using PositionalCell. Every usage of these should be done using the PositionalCell to provide future extensibility and ability to be used with different boards like samurai Sudoku.

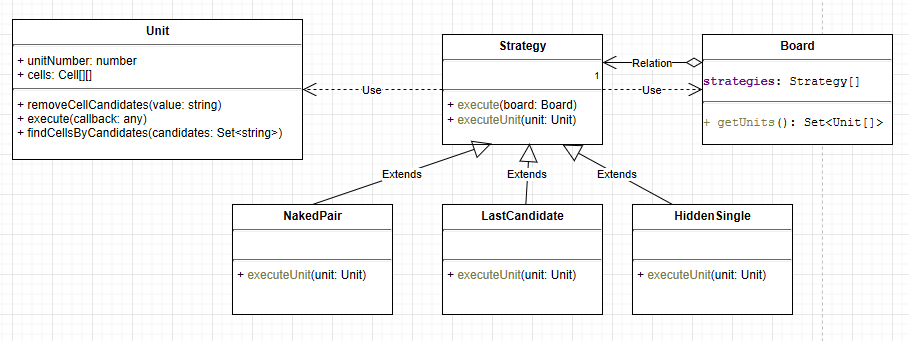
The CompositeUnit defined in the code as DefaultUnit is a composite design pattern. This is where the board or strategy components can build or use dynamic units which consist of one or more exiting unit sub class. This is very important concept used in our architecture to provide flexibility and extensibility.

## Strategy components

### Overview

The strategy components provides needed strategies or algorithms used to solve Sudoku board. Theses strategies follows the strategy design pattern from Gang of Four.

### Details



Strategy components uses the board components to get a set of units used by the board. Every strategy type use these units to apply the strategy and collect information. The method getUnits in the Board component is essential to provide proper encapsulation. This means that strategy does know anything about Unit types used in the Board. Thus changing Units on the board will have no effect on the strategy. Boards in the future can add or remove unit types ad needed by the Board itself.

Strategy component uses Unit component. Units component exposes an execute method which encapsulates the Unit data structure and how data is stored and retrieved. This methods follows functional programming pattern which. This makes Strategy be able to use the unit without any knowledge on how is data is stored or retrieved.

Since Strategy component has no knowledge about Unit implementation or sub classes, Units can be easy added or removed from the system without effecting Strategy components.

The above diagram shows a very important encapsulation by type concept. This means that Strategy does not know anything about Board or Unit sub classes or internal implementation which makes system extensible.

As shown in the above diagram NakedPair, LastCandidate and HidenSingle are different algorithms that implements the strategy abstract class. Other algorithms can also be added to the system to solve and address Sudoku boards. For example on these algorithms are Singles chains, Line intersection, and others.