CSED332 Assignment 3

Due Thursday, October 19

Background: X-Sudoku

- *X-Sudoku* is a variant of Sudoku (`https://en.wikipedia.org/wiki/Sudoku`). The goal of an X-Sudoku puzzle is to fill numbers from 1 to 9 in empty squares of a 9×9 grid such that
  - $1 \sim 9$ appear exactly once in each row, column and 3×3 box.
  - Each of the two main diagonals contains the numbers 1 through 9 only once.

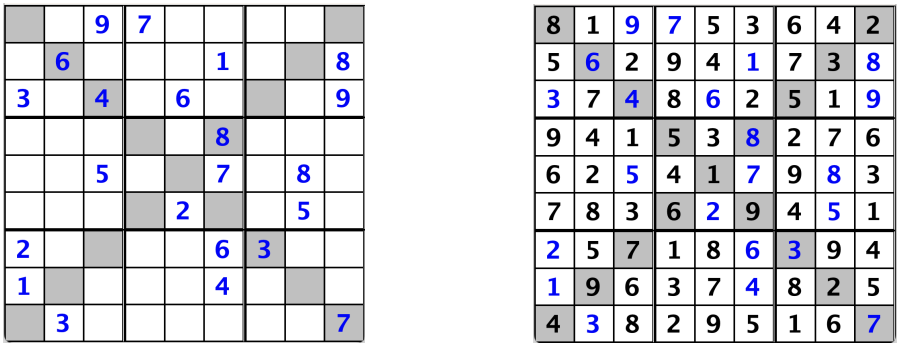

Figure 1: An X-Sudoku puzzle and its solution

Implementing X-Sudoku using Observer

- An X-Sudoku puzzle can be implemented using the Observer pattern. The key classes are `Cell` and `Group`, where the groups observe their cells.
  - A cell has a set of possible numbers that the cell can have, and may have a value. A cell changes by getting or losing a value or possibilities.
  - There is a group for each row, column, 3×3 box, and main diagonal. If one of the members of a group has a particular value, none of its other members can have the value as a possibility.
  - For example, in the unsolved puzzle of Figure 1, the first row of the first column has no value and the set of possibilities $\{5, 8\}$.
- Figure 2 shows the class diagram for this problem. The goal is to implement the five classes in the diagram: `Cell`, `Group`, `Board`, `CellUI`, and `GameUI`.
  - `Subject` and `Observer` are already implemented. A subject notifies an *event* to its observers. An event is an instance of `Event`, and provides additional information about changes.
  - A cell should notify appropriate events to its observers, when particular changes happen. E.g., if a cell has lost all its possibilities, it will notify `ActivationEvent(false)` to its observers.
  - A group receives an event when the value of its cell is set or unset (`NumberEvent(`n`, true)` or `NumberEvent(`n`, false)`). Then, it changes the possibilities of the other cells in the group.
  - A board maintains 9 row groups, 9 column groups, 9 square groups, 2 diagonal groups, and 81 cells. The classes `Board`, `Group`, and `Cell` specify the object-oriented model.

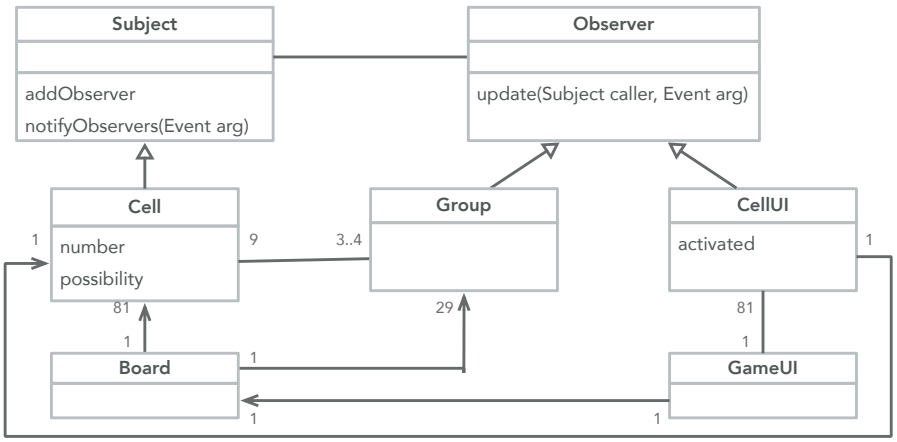

Figure 2: Class diagram for even/odd Sudoku

- `GameUI` and `CellUI` implements a simple GUI, as shown in Fig. 3. The class `GameUI` defines the top-level container. `CellUI` observes a single cell and defines an interface for the cell.
  - If a number is written in an empty `CellUI`, it tries to update the value of the related cell. If successful, the number is retained in the `CellUI`; otherwise, the `CellUI` is emptied again.
  - If a cell loses all its possibilities (`ActivationEvent(false)`), because other cells in the same group are filled, the corresponding `CellUI` is *deactivated* and marked with red borders.
  - If a number is removed from `CellUI`, other cells in the same group can restore a possibility. If a deactivated cell gets a possibility (`ActivationEvent(true)`), the `CellUI` is *activated*.

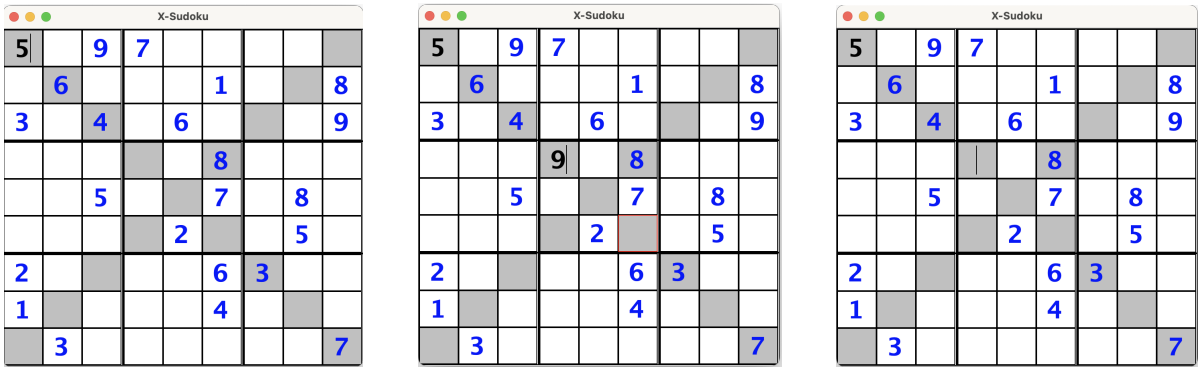

Figure 3: X-Sudoku GUI

General Instruction

- The `src/main` directory contains the skeleton code. You should implement all classes and methods with *TODO* in the above classes.
- The `src/test` directory contains some test methods for non-GUI classes in `BoardTest.java`. Your code will be graded by Gradle, using extra test cases written by teaching staff.
- Your code must follow the Model-View-Controller architectural pattern. In particular, the model classes (`Board`, `Group`, and `Cell`) should *not* depend on GUI classes.
- The command `gradle jar` will create a jar file in the `build/libs` directory, which can be executed using the command: `java -jar homework3-1.0-SNAPSHOT.jar`.
- Do not modify the existing interfaces, the class names, and the signatures of the public methods. You can add other methods or member variables if you want.

Turning in

1. Create a private project with name homework3 in `https://csed332.postech.ac.kr`, and clone the project on your machine.
2. Commit your changes in your homework3 project, push them to the remote repository. Tag your project with submitted. We will use the tagged version of your project for grading.

Reference

- Java Swing Tutorial: `https://www.javatpoint.com/java-swing`
- Using Swing Components: `https://docs.oracle.com/javase/tutorial/uiswing/components`
- Laying Out Components: `https://docs.oracle.com/javase/tutorial/uiswing/layout/`
- Writing Event Listeners: `https://docs.oracle.com/javase/tutorial/uiswing/events`