Goals Apply and use the Observer design pattern; construct a simple GUI application.

Preparation

Download links and other details can be found via the course web page:

`http://www.win.tue.nl/~wstomv/edu/sc-hse/`

1. In the book *Introduction to Programming Using Java* by David Eck, reread §6.1, 6.5; read §6.6; 7.3; 9.1 (Recursion); 11.2, and browse §13.4.1 (Model-View-Controller).
2. In the (e)book *Programming in the Large with Design Patterns* by Eddie Burris, read Chapter 9 (Observer Pattern).
3. Review the slides of Lecture 9.

Assignments

1. Inspect the feedback in peach[3] on your work for Series 8.
2. *Simple Kakuro Helper* Develop a simple GUI application to assist in solving Kakuro puzzles (see Fig. 1). In such a puzzle, each cell is to be filled by one number in the range 1 through 9 (a *digit*). Like a crossword puzzle, multiple horizontally or vertically adjacent empty cells belong together, and they are called an *entry*. The number of cells in an entry is called its *length*.

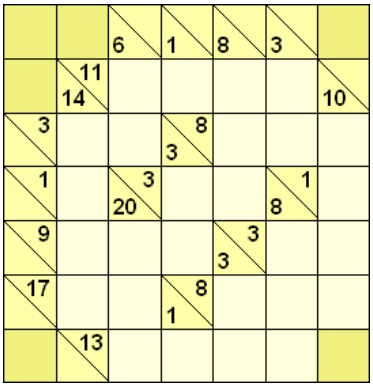

Figure 1: Example Kakuro puzzle

In the case of Kakuro puzzles, an entry contains *no duplicate digits*. Each entry has a *specification*. The specification of a Kakuro entry consists of a

number, which is the *sum* of the (unique) digits to be put in the adjacent cells of the entry.

In the example puzzle above, the number 9 in the leftmost column of the fifth row from the top is the specification of the horizontal entry with length 3, consisting of the three empty cells to its right. The sum 9 can be obtained by three different combinations of distinct digits:

- $1 + 2 + 6$
- $1 + 3 + 5$
- $2 + 3 + 4$

Of course, these digits can be entered in different orders.

The program you are to implement will not solve such puzzles, but given the sum s and length n of an entry, the program can list all combinations of n distinct digits with sum s (see Fig. 2), and additional information.

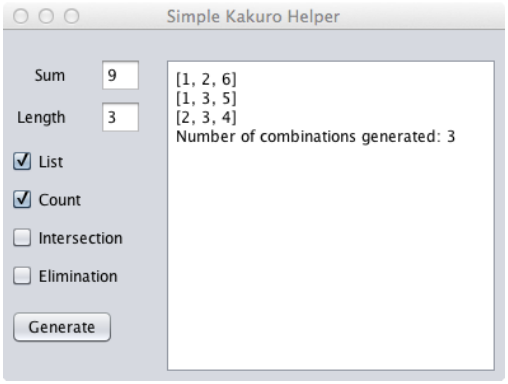

Figure 2: Screenshot of application for sum 9 and length 3

The source code must be organized as follows (still no Java packages; that is, all files reside in the default package; also see the given skeleton code):

- **class** `KakuroCombinationGenerator`, an observable generator with method **void** `generate(`**int** `s, `**int** `n)` that generates all combinations, triggering an event on all registered listeners for each generated combination. N.B. It does not store all generated combinations.
- **class** `KakuroCombinationGeneratorTest`, a few test cases, that you are strongly recommended to expand.
- **interface** `GeneratorListener`, for observers.
- **class** `Counter` **implements** `GeneratorListener`, a listener that counts.

- **class** `Intersector` **implements** `GeneratorListener`, a listener that calculates the intersection of all combinations (or of their complements); that is, to calculate the set of numbers that is common to all combinations generated, or the set of numbers that appear in none of the combinations (see Fig. 3).
- **class** `MainFrame` **extends** `JFrame`, the main GUI window, holding some labels, text fields (for input), check boxes (for options), a button and a text area (for output).

  When the Generate button is clicked, the program registers the user-selected listeners and invokes the generator.

  The **class** `Lister` **implements** `GeneratorListener` is an inner class of `MainFrame`, so that it can easily append generated combinations to the text area on the right.

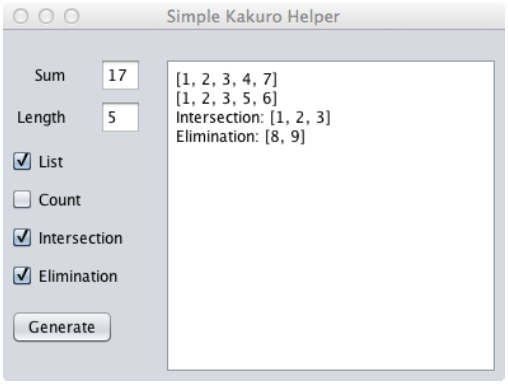

Figure 3: Screenshot of application for sum 17 and length 5, showing intersections

Feel free in choice of tool to develop the GUI. `Swing` is recommended; *NetBeans* has a graphical designer for `Swing` GUIs. Submit work to peach[3].

Please, do not remove the `//# ... TODO` *markers from given skeleton code.* You can substitute your own `MainFrame`, but do adhere to the proposed layout, including your name, group number, date, and the cut line.

Hints

- Convert a `String s` to an **int** with `Integer.parseInt(s)`. Do this in a **try** statement, and catch `NumberFormatException`, to present an error message in the text area.
- The type `Set` offers a `toString` method to convert a set to a `String`, producing e.g. `"[1, 2, 6]"` for a `Set<Integer>` value.
- Append a line containing `String s` to a text area with the method call `jTextArea.append(s + "\n")`.
- Clear a text area by `jTextArea.setText("")`.

Deadline: Thursday 20 November 23:00 (Moscow time)

