

Java Graphics Programming

Assignment - Sudoku

TABLE OF CONTENTS (HIDE)

1. Rules of Game
2. Graphical Display
3. Event Handling
- [4. Hints and Miscellaneous](#)
5. More Credits
6. (Advanced) Subclassing JTextField



1. Rules of Game

You could wiki "Sudoku" to understand the rules of the game.

Sudoku is a single-player mind game. "The objective is to fill a 9×9 grid with digits 1 to 9, so that each column, each row, and each of the nine 3×3 sub-grids (also called "boxes", "blocks", "regions", or "sub-squares") contains all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a unique solution."

2. Graphical Display

Start with the GUI for display. The GUI codes is simple. You can simply use a 9x9 JTextFields arranged in a 9x9 GridLayout.

The steps for producing the display are:

- Set the JFrame's content-pane to 9×9 GridLayout. Create 9×9 JTextFields (called tfCells) and add to the content-pane. The JTextFields shall contain the string "1" to "9" (the number from the puzzle or the number guessed) or empty string (for blank cells).
- Initialize the game by reading in an input puzzle (int[9][9] puzzle) with blank cells (handled by boolean[9][9] masks), and populate the tfCells arrays. Set the non-empty cells to non-editable containing the number from the puzzle; and set the empty cells to editable containing an empty string.

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

Study the following codes. Fill in the main() method ([TODO 1]), run the program, which shall produce the display.

```
import java.awt.*;           // Uses AWT's Layout Managers
import java.awt.event.*;    // Uses AWT's Event Handlers
import javax.swing.*;       // Uses Swing's Container/Components

/**
 * The Sudoku game.
```

```

* To solve the number puzzle, each row, each column, and each of the
* nine 3x3 sub-grids shall contain all of the digits from 1 to 9
*/
public class Sudoku extends JFrame {
    // Name-constants for the game properties
    public static final int GRID_SIZE = 9;    // Size of the board
    public static final int SUBGRID_SIZE = 3; // Size of the sub-grid

    // Name-constants for UI control (sizes, colors and fonts)
    public static final int CELL_SIZE = 60;   // Cell width/height in pixels
    public static final int CANVAS_WIDTH  = CELL_SIZE * GRID_SIZE;
    public static final int CANVAS_HEIGHT = CELL_SIZE * GRID_SIZE;
                                         // Board width/height in pixels
    public static final Color OPEN_CELL_BGCOLOR = Color.YELLOW;
    public static final Color OPEN_CELL_TEXT_YES = new Color(0, 255, 0); // RGB
    public static final Color OPEN_CELL_TEXT_NO = Color.RED;
    public static final Color CLOSED_CELL_BGCOLOR = new Color(240, 240, 240); // RGB
    public static final Color CLOSED_CELL_TEXT = Color.BLACK;
    public static final Font FONT_NUMBERS = new Font("Monospaced", Font.BOLD, 20);

    // The game board composes of 9x9 JTextFields,
    // each containing String "1" to "9", or empty String
    private JTextField[][] tfCells = new JTextField[GRID_SIZE][GRID_SIZE];

    // Puzzle to be solved and the mask (which can be used to control the
    // difficulty level).
    // Hardcoded here. Extra credit for automatic puzzle generation
    // with various difficulty levels.
    private int[][] puzzle =
        {{5, 3, 4, 6, 7, 8, 9, 1, 2},
         {6, 7, 2, 1, 9, 5, 3, 4, 8},
         {1, 9, 8, 3, 4, 2, 5, 6, 7},
         {8, 5, 9, 7, 6, 1, 4, 2, 3},
         {4, 2, 6, 8, 5, 3, 7, 9, 1},
         {7, 1, 3, 9, 2, 4, 8, 5, 6},
         {9, 6, 1, 5, 3, 7, 2, 8, 4},
         {2, 8, 7, 4, 1, 9, 6, 3, 5},
         {3, 4, 5, 2, 8, 6, 1, 7, 9}};
    // For testing, open only 2 cells.
    private boolean[][] masks =
        {{false, false, false, false, false, true, false, false, false},
         {false, false, false, false, false, false, false, false, true},
         {false, false, false, false, false, false, false, false, false},
         {false, false, false, false, false, false, false, false, false},
         {false, false, false, false, false, false, false, false, false},
         {false, false, false, false, false, false, false, false, false},
         {false, false, false, false, false, false, false, false, false},
         {false, false, false, false, false, false, false, false, false},
         {false, false, false, false, false, false, false, false, false}};

    /**
     * Constructor to setup the game and the UI Components
     */
    public Sudoku() {
        Container cp = getContentPane();
        cp.setLayout(new GridLayout(GRID_SIZE, GRID_SIZE)); // 9x9 GridLayout

        // Allocate a common listener as the ActionEvent listener for all the
        // JTextFields
        // ... [TODO 3] (Later) ....

        // Construct 9x9 JTextFields and add to the content-pane
        for (int row = 0; row < GRID_SIZE; ++row) {
            for (int col = 0; col < GRID_SIZE; ++col) {

```

```

tfCells[row][col] = new JTextField(); // Allocate element of array
cp.add(tfCells[row][col]);           // ContentPane adds JTextField
if (masks[row][col]) {
    tfCells[row][col].setText("");    // set to empty string
    tfCells[row][col].setEditable(true);
    tfCells[row][col].setBackground(OPEN_CELL_BGCOLOR);

    // Add ActionEvent listener to process the input
    // ... [TODO 4] (Later) ...
} else {
    tfCells[row][col].setText(puzzle[row][col] + "");
    tfCells[row][col].setEditable(false);
    tfCells[row][col].setBackground(CLOSED_CELL_BGCOLOR);
    tfCells[row][col].setForeground(CLOSED_CELL_TEXT);
}
// Beautify all the cells
tfCells[row][col].setHorizontalAlignment(JTextField.CENTER);
tfCells[row][col].setFont(FONT_NUMBERS);
}
}

// Set the size of the content-pane and pack all the components
// under this container.
cp.setPreferredSize(new Dimension(CANVAS_WIDTH, CANVAS_HEIGHT));
pack();

setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // Handle window closing
setTitle("Sudoku");
setVisible(true);
}

/** The entry main() entry method */
public static void main(String[] args) {
    // [TODO 1] (Now)
    // Check Swing program template on how to run the constructor
    .....
}

// Define the Listener Inner Class
// ... [TODO 2] (Later) ...
}

```

3. Event Handling

Next, we shall program the event handling.

We shall use a common instance of a Named Inner Class (called `InputListener`) as the `ActionEvent` listener for "all" the editable `JTextFields`. Hence, in the `actionPerformed()`, we need to identify the particular `JTextField` (in terms of `row` and `col`) that trigger the event. You could use the `ActionEvent.getSource()` method to retrieve the source object that has fired the event and compare the object with all the 9×9 `JTextFields`.

Place the inner class at [TODO 2].

```

// [TODO 2]
// Inner class to be used as ActionEvent listener for ALL JTextFields
private class InputListener implements ActionListener {

    @Override
    public void actionPerformed(ActionEvent e) {
        // All the 9*9 JTextFilededs invoke this handler. We need to determine

```

```

// which JTextField (which row and column) is the source for this invocation.
int rowSelected = -1;
int colSelected = -1;

// Get the source object that fired the event
JTextField source = (JTextField)e.getSource();
// Scan JTextFields for all rows and columns, and match with the source object
boolean found = false;
for (int row = 0; row < GRID_SIZE && !found; ++row) {
    for (int col = 0; col < GRID_SIZE && !found; ++col) {
        if (tfCells[row][col] == source) {
            rowSelected = row;
            colSelected = col;
            found = true; // break the inner/outer loops
        }
    }
}

/*
 * [TODO 5]
 * 1. Get the input String via tfCells[rowSelected][colSelected].getText()
 * 2. Convert the String to int via Integer.parseInt().
 * 3. Assume that the solution is unique. Compare the input number with
 *    the number in the puzzle[rowSelected][colSelected]. If they are the same,
 *    set the background to green (Color.GREEN); otherwise, set to red (Color.RED).
 */

/*
 * [TODO 6] Check if the player has solved the puzzle after this move.
 * You could update the masks[][] on correct guess, and check the masks[][] if
 * any input cell pending.
 */
}
}

```

In Sudoku.java's constructor:

1. Declare and allocate a common instance called listener of the InputListener class:

```

// [TODO 3]
InputListener listener = new InputListener();

```

2. All editable JTextField shall add this common instance as its ActionListener:

```

// [TODO 4]
tfCells[row][col].addActionListener(listener); // For all editable rows and cols

```

Continue the actionPerformed() to check if the input number is acceptable in that particular (rowSelected, colSelected) ([TODO 5] and [TODO 6]).

Some useful methods of JTextField are as follows. You can check the Java API for more methods.

```

setBackground(Color c) // Set the background color of the component
setForeground(Color c) // Set the text color of the JTextField
setFont(Font f) // Set the font used by the JTextField
setHorizontalAlignment(int align); // align: JTextField.CENTER, JTextField.LEFT, JTextField.RIGHT

```

Common colors are defined via constants such as Color.RED, Color.GREEN, Color.BLUE, and etc.

4. Hints and Miscellaneous

- This is a moderately complex program. You need to use the graphics debugger under Eclipse/NetBeans

to debug your program logic.

- Check the JDK API on the classes and methods available under Swing.
- You can use the following static method to pop up a dialog box with a message:

```
JOptionPane.showMessageDialog(null, "Congratulation!");
```

- To check the sub-grid, you could use `InputRow/3` and `InputCol/3` to get the `subGridRow` and `subGridCol`. The (row, col)'s for the sub-grid are (`subGridRow*3 + 0|1|2`, `subGridCol*3 + 0|1|2`).
- There are many ways to automatically generate a new puzzle, e.g.,
 - Start with a solved puzzle, add a random number between 1-9 to all the cells and modulo 9.
 - Start with a solved puzzle, swap rows/columns among 1-2-3, 4-5-6, 7-8-9.
 - [TODO]
- There are many ways to set the difficulty level, e.g.,
 - Control the number of unrevealed cells.
 - [TODO]

5. More Credits

- A good Sudoku engine shall accept any "valid" number at the time of input (no duplicate in row, column and sub-grid), but signal a conflict whenever it is detected. Highlight the conflicting cells.
- Re-organize your codes, with methods such as `initGame()`, `getPuzzle()`, etc. Better still, apply your OO knowledge to write a separate class called `Puzzle`, and etc.
- Beautify your graphical interface, e.g., color, font, layout, etc.
- Choice of puzzles and difficulty levels.
- Create a status bar (`JTextField` at the south zone of `BorderLayout`) to show the messages (e.g., number of cells remaining) (google "java swing statusbar").
- Create a menu bar for options such as "File" ("New Game", "Reset Game", "Exit"), "Options", and "Help" (Use `JMenuBar`, `JMenu`, and `JMenuItem` classes).
- Timer (pause/resume), score, progress bar.
- A side panel for command, display, strategy?
- Automatic puzzle generation with various difficulty level.
- The sample program processes `ActionEvent` of the `JTextField`, which requires user to push the ENTER key. Try `KeyEvent` with `keyTyped()` handler; or other means that does not require pushing of ENTER key.
- Sound effect, background music, enable/disable sound?
- High score and player name?
- Hints and cheats (reveal a cell, or reveal all cells with number n)?
- Choice of display "theme"?
- Use of images and icons?
-

6. (Advanced) Subclassing JTextField

The `javax.swing.JTextField` does not contain identifiers (such as row and column). You can actually create your subclass of `JTextField` to provide the additional functionality, which will greatly simplify your programming. E.g.,

```
import javax.swing.JTextField;

public class Cell extends JTextField {
    // Package access
    int row, col;        // row and col of this JTextField

    // More?! For examples,
    int numberPuzzle;    // The puzzle number
    boolean isOpened;    // to be solved?
    int numberGuess;     // The number entered/display
    boolean isValid;
}
```

REFERENCES & RESOURCES

Latest version tested: JDK 1.8.0_66

Last modified: April, 2016

Feedback, comments, corrections, and errata can be sent to Chua Hock-Chuan (ehchua@ntu.edu.sg) | [HOME](#)