```java
import java.awt.Cursor;

/**
 * View class.
 *
 * @author Nicholas Cheong
 */
public final class NNCalcView1 extends JFrame implements NNCalcView {

    /**
     * Controller object registered with this view to observe user-interaction
     * events.
     */
    private NNCalcController controller;

    /**
     * State of user interaction: last event "seen".
     */
    private enum State {
        /**
         * Last event was clear, enter, another operator, or digit entry, resp.
         */
        SAW_CLEAR, SAW_ENTER_OR_SWAP, SAW_OTHER_OP, SAW_DIGIT
    }

    /**
     * State variable to keep track of which event happened last; needed to
     * prepare for digit to be added to bottom operand.
     */
    private State currentState;

    /**
     * Text areas.
     */
    private final JTextArea tTop, tBottom;

    /**
     * Operator and related buttons.
     */
    private final JButton bClear, bSwap, bEnter, bAdd,
bSubtract, bMultiply,
            bDivide, bPower, bRoot;

    /**
     * Digit entry buttons.
     */
    private final JButton[] bDigits;

    /**
     * Useful constants.
     */
    private static final int TEXT_AREA_HEIGHT = 5,
TEXT_AREA_WIDTH = 20,
            DIGIT_BUTTONS = 10, MAIN_BUTTON_PANEL_GRID_ROWS =
4,
            MAIN_BUTTON_PANEL_GRID_COLUMNS = 4,
SIDE_BUTTON_PANEL_GRID_ROWS = 3,
            SIDE_BUTTON_PANEL_GRID_COLUMNS = 1, CALC_GRID_ROWS
= 3,
            CALC_GRID_COLUMNS = 1;

    /**
     * Default constructor.
     */
    public NNCalcView1() {
        // Create the JFrame being extended

        /*
         * Call the JFrame (superclass) constructor with a
String parameter to
         * name the window in its title bar
         */
        super("Natural Number Calculator");

        // Set up the GUI widgets
--------------------------------------------

        /*
         * Set up initial state of GUI to behave like last
event was "Clear";
         * currentState is not a GUI widget per se, but is
needed to process
         * digit button events appropriately
         */
        this.currentState = State.SAW_CLEAR;

        // TODO: fill in rest of body, following outline in comments

        /*
         * Create widgets
         */

        this.tTop = new JTextArea("", TEXT_AREA_HEIGHT, TEXT_AREA_WIDTH);
        this.tBottom = new JTextArea("", TEXT_AREA_HEIGHT, TEXT_AREA_WIDTH);
        this.bClear = new JButton("clear");
        this.bSwap = new JButton("swap");
        this.bEnter = new JButton("=");
        this.bAdd = new JButton("+");
        this.bSubtract = new JButton("-");
        this.bMultiply = new JButton("x");
        this.bDivide = new JButton("/");
        this.bPower = new JButton("^");
        this.bRoot = new JButton("root");
        // ...
        this.bDigits = new JButton[DIGIT_BUTTONS];
        for (int i = 0; i < this.bDigits.length; i++) {
            this.bDigits[i] = new JButton("" + i);
        }

        // Set up the GUI widgets --------------------------------------------

        /*
         * Text areas should wrap lines, and should be read-only; they cannot be
         * edited because allowing keyboard entry would require checking whether
         * entries are digits, which we don't want to have to do
         */

        this.tTop.setLineWrap(true);
        this.tBottom.setLineWrap(true);

        /*
```

```java
         * Initially, the following buttons should be disabled: divide (divisor
         * must not be 0) and root (root must be at least 2) -- hint: see the
         * JButton method setEnabled
         */

        this.bDivide.setEnabled(false);
        this.bRoot.setEnabled(false);

        /*
         * Create scroll panes for the text areas in case number is long enough
         * to require scrolling
         */

        JScrollPane topScrollPane = new JScrollPane(this.tTop);
        JScrollPane bottomScrollPane = new JScrollPane(this.tBottom);

        /*
         * Create main button panel
         */

        JPanel mainButtonPanel = new JPanel(new GridLayout(
                MAIN_BUTTON_PANEL_GRID_ROWS, MAIN_BUTTON_PANEL_GRID_COLUMNS));

        /*
         * Add the buttons to the main button panel, from left to right and top
         * to bottom
         */

        mainButtonPanel.add(this.bDigits[7]);
        mainButtonPanel.add(this.bDigits[8]);
        mainButtonPanel.add(this.bDigits[9]);
        mainButtonPanel.add(this.bAdd);
        mainButtonPanel.add(this.bDigits[4]);
        mainButtonPanel.add(this.bDigits[5]);
        mainButtonPanel.add(this.bDigits[6]);
        mainButtonPanel.add(this.bSubtract);
        mainButtonPanel.add(this.bDigits[1]);
        mainButtonPanel.add(this.bDigits[2]);
        mainButtonPanel.add(this.bDigits[3]);
        mainButtonPanel.add(this.bMultiply);
        mainButtonPanel.add(this.bDigits[0]);
        mainButtonPanel.add(this.bPower);
        mainButtonPanel.add(this.bRoot);
        mainButtonPanel.add(this.bDivide);

        /*
         * Create side button panel
         */

        JPanel sideButtonPanel = new JPanel(new GridLayout(
                SIDE_BUTTON_PANEL_GRID_ROWS,
SIDE_BUTTON_PANEL_GRID_COLUMNS));

        /*
         * Add the buttons to the side button panel, from left
to right and top
         * to bottom
         */

        sideButtonPanel.add(this.bClear);
        sideButtonPanel.add(this.bSwap);
        sideButtonPanel.add(this.bEnter);

        /*
         * Create combined button panel organized using flow
layout, which is
         * simple and does the right thing: sizes of nested
panels are natural,
         * not necessarily equal as with grid layout
         */

        JPanel combinedPanel = new JPanel(new FlowLayout());

        /*
         * Add the other two button panels to the combined
button panel
         */

        combinedPanel.add(mainButtonPanel);
        combinedPanel.add(sideButtonPanel);

        /*
```

```java
         * Organize main window
         */

        this.setLayout(new GridLayout(CALC_GRID_ROWS, CALC_GRID_COLUMNS));

        /*
         * Add scroll panes and button panel to main window, from left to right
         * and top to bottom
         */

        this.add(topScrollPane);
        this.add(bottomScrollPane);
        this.add(combinedPanel);

        // Set up the observers ----------------------------------------------

        /*
         * Register this object as the observer for all GUI events
         */

        this.bClear.addActionListener(this);
        this.bSwap.addActionListener(this);
        this.bEnter.addActionListener(this);
        this.bAdd.addActionListener(this);
        this.bSubtract.addActionListener(this);
        this.bMultiply.addActionListener(this);
        this.bDivide.addActionListener(this);
        this.bPower.addActionListener(this);
        this.bRoot.addActionListener(this);
        for (int i = 0; i < this.bDigits.length; i++) {
            this.bDigits[i].addActionListener(this);
        }

        // Set up the main application window --------------------------------

        /*
         * Make sure the main window is appropriately sized, exits this program
         * on close, and becomes visible to the user
         */

        this.pack();
        this.setVisible(true);

    }

    @Override
    public void registerObserver(NNCalcController controller) {

        this.controller = controller;

    }

    @Override
    public void updateTopDisplay(NaturalNumber n) {
        this.tTop.setText(n.toString());
    }

    @Override
    public void updateBottomDisplay(NaturalNumber n) {

        this.tBottom.setText(n.toString());

    }

    @Override
    public void updateSubtractAllowed(boolean allowed) {
        this.bSubtract.setEnabled(allowed);
    }

    @Override
    public void updateDivideAllowed(boolean allowed) {

        this.bDivide.setEnabled(allowed);

    }

    @Override
    public void updatePowerAllowed(boolean allowed) {

        this.bPower.setEnabled(allowed);

    }

    @Override
    public void updateRootAllowed(boolean allowed) {

        this.bRoot.setEnabled(allowed);

    }

    @Override
    public void actionPerformed(ActionEvent event) {
        /*
         * Set cursor to indicate computation on-going; this matters only if
         * processing the event might take a noticeable amount of time as seen
         * by the user
         */
        this.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
        /*
         * Determine which event has occurred that we are being notified of by
         * this callback; in this case, the source of the event (i.e, the widget
         * calling actionPerformed) is all we need because only buttons are
         * involved here, so the event must be a button press; in each case,
         * tell the controller to do whatever is needed to update the model and
         * to refresh the view
         */
        Object source = event.getSource();
        if (source == this.bClear) {
            this.controller.processClearEvent();
            this.currentState = State.SAW_CLEAR;
        } else if (source == this.bSwap) {
            this.controller.processSwapEvent();
            this.currentState = State.SAW_ENTER_OR_SWAP;
        } else if (source == this.bEnter) {
            this.controller.processEnterEvent();
            this.currentState = State.SAW_ENTER_OR_SWAP;
        } else if (source == this.bAdd) {
            this.controller.processAddEvent();
```

```java
            this.currentState = State.SAW_OTHER_OP;
        } else if (source == this.bSubtract) {
            this.controller.processSubtractEvent();
            this.currentState = State.SAW_OTHER_OP;
        } else if (source == this.bMultiply) {
            this.controller.processMultiplyEvent();
            this.currentState = State.SAW_OTHER_OP;
        } else if (source == this.bDivide) {
            this.controller.processDivideEvent();
            this.currentState = State.SAW_OTHER_OP;
        } else if (source == this.bPower) {
            this.controller.processPowerEvent();
            this.currentState = State.SAW_OTHER_OP;
        } else if (source == this.bRoot) {
            this.controller.processRootEvent();
            this.currentState = State.SAW_OTHER_OP;
        } else {
            for (int i = 0; i < DIGIT_BUTTONS; i++) {
                if (source == this.bDigits[i]) {
                    switch (this.currentState) {
                        case SAW_ENTER_OR_SWAP:
                            this.controller.processClearEvent();
                            break;
                        case SAW_OTHER_OP:
                            this.controller.processEnterEvent();
                            this.controller.processClearEvent();
                            break;
                        default:
                            break;
                    }
                    this.controller.processAddNewDigitEvent(i);
                    this.currentState = State.SAW_DIGIT;
                    break;
                }
            }
        }
        /*
         * Set the cursor back to normal (because we changed it at the beginning
         * of the method body)
         */
        this.setCursor(Cursor.getDefaultCursor());
    }

}

```