```
 1 import java.awt.Cursor;
 2 import java.awt.FlowLayout;
 3 import java.awt.GridLayout;
 4 import java.awt.event.ActionEvent;
 5
 6 import javax.swing.JButton;
 7 import javax.swing.JFrame;
 8 import javax.swing.JPanel;
 9 import javax.swing.JScrollPane;
10 import javax.swing.JTextArea;
11
12 import components.naturalnumber.NaturalNumber;
13
14 /**
15  * View class.
16  *
17  * @author Shafin Alam
18  */
19 public final class NNCalcView1 extends JFrame implements
    NNCalcView {
20
21     /**
22      * Controller object registered with this view to
    observe user-interaction
23      * events.
24      */
25     private NNCalcController controller;
26
27     /**
28      * State of user interaction: last event "seen".
29      */
30     private enum State {
31         /**
32          * Last event was clear, enter, another operator,
```

```
   or digit entry, resp.
33          */
34         SAW_CLEAR, SAW_ENTER_OR_SWAP, SAW_OTHER_OP,
   SAW_DIGIT
35     }
36
37     /**
38      * State variable to keep track of which event
   happened last; needed to
39      * prepare for digit to be added to bottom operand.
40      */
41     private State currentState;
42
43     /**
44      * Text areas.
45      */
46     private final JTextArea tTop, tBottom;
47
48     /**
49      * Operator and related buttons.
50      */
51     private final JButton bClear, bSwap, bEnter, bAdd,
   bSubtract, bMultiply,
52             bDivide, bPower, bRoot;
53
54     /**
55      * Digit entry buttons.
56      */
57     private final JButton[] bDigits;
58
59     /**
60      * Useful constants.
61      */
62     private static final int TEXT_AREA_HEIGHT = 5,
```

```
   TEXT_AREA_WIDTH = 20,
63             DIGIT_BUTTONS = 10,
   MAIN_BUTTON_PANEL_GRID_ROWS = 4,
64             MAIN_BUTTON_PANEL_GRID_COLUMNS = 4,
   SIDE_BUTTON_PANEL_GRID_ROWS = 3,
65             SIDE_BUTTON_PANEL_GRID_COLUMNS = 1,
   CALC_GRID_ROWS = 3,
66             CALC_GRID_COLUMNS = 1;
67
68     /**
69      * Default constructor.
70      */
71     public NNCalcView1() {
72         // Create the JFrame being extended
73         /*
74          * Call the JFrame (superclass) constructor with a
   String parameter to
75          * name the window in its title bar
76          */
77         super("Natural Number Calculator");
78
79         // Set up the GUI widgets
   ----------------------------------------
80
81         /*
82          * Set up initial state of GUI to behave like last
   event was "Clear";
83          * currentState is not a GUI widget per se, but is
   needed to process
84          * digit button events appropriately
85          */
86         this.currentState = State.SAW_CLEAR;
87         /*
88          * Create widgets
```

```
 89          */
 90         this.tTop = new JTextArea("0", TEXT_AREA_HEIGHT,
    TEXT_AREA_WIDTH);
 91         this.tBottom = new JTextArea("0",
    TEXT_AREA_HEIGHT, TEXT_AREA_WIDTH);
 92         this.bAdd = new JButton("+");
 93         this.bSubtract = new JButton("-");
 94         this.bMultiply = new JButton("*");
 95         this.bDivide = new JButton("/");
 96         this.bPower = new JButton("Power");
 97         this.bRoot = new JButton("Root");
 98         this.bClear = new JButton("Clear");
 99         this.bSwap = new JButton("Swap");
100         this.bEnter = new JButton("Enter");
101         this.bDigits = new JButton[DIGIT_BUTTONS];
102         int i = 0;
103         while (i < DIGIT_BUTTONS) {
104             this.bDigits[i] = new JButton
    (String.valueOf(i));
105             i++;
106         }
107         // Set up the GUI widgets
    ----------------------------------------
108         /*
109          * Text areas should wrap lines, and should be
    read-only; they cannot be
110          * edited because allowing keyboard entry would
    require checking whether
111          * entries are digits, which we don't want to have
    to do
112          */
113         this.tTop.setEditable(false);
114         this.tBottom.setEditable(false);
115         this.tTop.setLineWrap(true);
```

```
116         this.tBottom.setLineWrap(true);
117         /*
118          * Initially, the following buttons should be
    disabled: divide (divisor
119          * must not be 0) and root (root must be at least
    2) -- hint: see the
120          * JButton method setEnabled
121          */
122         if (this.tBottom.getText().equals("0")) {
123             this.bDivide.setEnabled(false);
124         }
125         if (this.tBottom.getText().equals("0")
126                 || this.tBottom.getText().equals("1")) {
127             this.bRoot.setEnabled(false);
128         }
129         /*
130          * Create scroll panes for the text areas in case
    number is long enough
131          * to require scrolling
132          */
133         JScrollPane topScrollPane = new JScrollPane
    (this.tTop);
134         JScrollPane bottomScrollPane = new JScrollPane
    (this.tBottom);
135         /*
136          * Create main button panel
137          */
138         JPanel mainButtonPanel = new JPanel(new GridLayout
    (
139                 MAIN_BUTTON_PANEL_GRID_ROWS,
    MAIN_BUTTON_PANEL_GRID_COLUMNS));
140         /*
141          * Add the buttons to the main button panel, from
    left to right and top
```

```
142          * to bottom
143          */
144         mainButtonPanel.add(this.bDigits[7]);
145         mainButtonPanel.add(this.bDigits[8]);
146         mainButtonPanel.add(this.bDigits[9]);
147         mainButtonPanel.add(this.bAdd);
148         mainButtonPanel.add(this.bDigits[4]);
149         mainButtonPanel.add(this.bDigits[5]);
150         mainButtonPanel.add(this.bDigits[6]);
151         mainButtonPanel.add(this.bSubtract);
152         mainButtonPanel.add(this.bDigits[1]);
153         mainButtonPanel.add(this.bDigits[2]);
154         mainButtonPanel.add(this.bDigits[3]);
155         mainButtonPanel.add(this.bMultiply);
156         mainButtonPanel.add(this.bDigits[0]);
157         mainButtonPanel.add(this.bPower);
158         mainButtonPanel.add(this.bRoot);
159         mainButtonPanel.add(this.bDivide);
160         /*
161          * Create side button panel
162          */
163         JPanel sideButtonPanel = new JPanel(new GridLayout
    (
164                 SIDE_BUTTON_PANEL_GRID_ROWS,
    SIDE_BUTTON_PANEL_GRID_COLUMNS));
165         /*
166          * Add the buttons to the side button panel, from
    left to right and top
167          * to bottom
168          */
169         sideButtonPanel.add(this.bClear);
170         sideButtonPanel.add(this.bSwap);
171         sideButtonPanel.add(this.bEnter);
172         /*
```

```
173         * Create combined button panel organized using
    flow layout, which is
174         * simple and does the right thing: sizes of
    nested panels are natural,
175         * not necessarily equal as with grid layout
176         */
177        JPanel combinedButtonPanel = new JPanel(new
    FlowLayout());
178        /*
179         * Add the other two button panels to the combined
    button panel
180         */
181        combinedButtonPanel.add(mainButtonPanel);
182        combinedButtonPanel.add(sideButtonPanel);
183        /*
184         * Organize main window
185         */
186        this.setLayout(new GridLayout(CALC_GRID_ROWS,
    CALC_GRID_COLUMNS));
187        /*
188         * Add scroll panes and button panel to main
    window, from left to right
189         * and top to bottom
190         */
191        this.add(topScrollPane);
192        this.add(bottomScrollPane);
193        this.add(combinedButtonPanel);
194        // Set up the observers
    -------------------------------------------
195        /*
196         * Register this object as the observer for all
    GUI events
197         */
198        this.bClear.addActionListener(this);
```

```
199         this.bSwap.addActionListener(this);
200         this.bEnter.addActionListener(this);
201         this.bAdd.addActionListener(this);
202         this.bSubtract.addActionListener(this);
203         this.bMultiply.addActionListener(this);
204         this.bDivide.addActionListener(this);
205         this.bPower.addActionListener(this);
206         this.bRoot.addActionListener(this);
207         int j = 0;
208         while (j < DIGIT_BUTTONS) {
209             this.bDigits[j].addActionListener(this);
210             j++;
211         }
212         // Set up the main application window
    ---------------------------------
213         /*
214          * Make sure the main window is appropriately
    sized, exits this program
215          * on close, and becomes visible to the user
216          */
217         this.pack();
218         this.setVisible(true);
219         this.setDefaultCloseOperation
    (JFrame.EXIT_ON_CLOSE);
220     }
221 
222     @Override
223     public void registerObserver(NNCalcController
    controller) {
224         /*
225          * Register argument as observer/listener of this;
    this must be done
226          * first, before any other methods of this class
    are called.
```

```
227          */
228         this.controller = controller;
229 
230     }
231 
232     @Override
233     public void updateTopDisplay(NaturalNumber n) {
234         /*
235          * Updates top operand display based on
    NaturalNumber provided as
236          * argument.
237          */
238         this.tTop.setText(String.valueOf(n));
239 
240     }
241 
242     @Override
243     public void updateBottomDisplay(NaturalNumber n) {
244         /*
245          * Updates bottom operand display based on
    NaturalNumber provided as
246          * argument.
247          */
248         this.tBottom.setText(String.valueOf(n));
249 
250     }
251 
252     @Override
253     public void updateSubtractAllowed(boolean allowed) {
254         /*
255          * Updates display of whether subtract operation
    is allowed.
256          */
257         this.bSubtract.setEnabled(allowed);
```

```
258
259     }
260
261     @Override
262     public void updateDivideAllowed(boolean allowed) {
263         /*
264          * Updates display of whether divide operation is
    allowed.
265          */
266         this.bDivide.setEnabled(allowed);
267
268     }
269
270     @Override
271     public void updatePowerAllowed(boolean allowed) {
272         /*
273          * Updates display of whether power operation is
    allowed.
274          */
275         this.bPower.setEnabled(allowed);
276
277     }
278
279     @Override
280     public void updateRootAllowed(boolean allowed) {
281         /*
282          * Updates display of whether root operation is
    allowed.
283          */
284         this.bRoot.setEnabled(allowed);
285
286     }
287
288     @Override
```

```
289     public void actionPerformed(ActionEvent event) {
290         /*
291          * Set cursor to indicate computation on-going;
    this matters only if
292          * processing the event might take a noticeable
    amount of time as seen
293          * by the user
294          */
295         this.setCursor
    (Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
296         /*
297          * Determine which event has occurred that we are
    being notified of by
298          * this callback; in this case, the source of the
    event (i.e, the widget
299          * calling actionPerformed) is all we need because
    only buttons are
300          * involved here, so the event must be a button
    press; in each case,
301          * tell the controller to do whatever is needed to
    update the model and
302          * to refresh the view
303          */
304         Object source = event.getSource();
305         if (source == this.bClear) {
306             this.controller.processClearEvent();
307             this.currentState = State.SAW_CLEAR;
308         } else if (source == this.bSwap) {
309             this.controller.processSwapEvent();
310             this.currentState = State.SAW_ENTER_OR_SWAP;
311         } else if (source == this.bEnter) {
312             this.controller.processEnterEvent();
313             this.currentState = State.SAW_ENTER_OR_SWAP;
314         } else if (source == this.bAdd) {
```

```
315             this.controller.processAddEvent();
316             this.currentState = State.SAW_OTHER_OP;
317         } else if (source == this.bSubtract) {
318             this.controller.processSubtractEvent();
319             this.currentState = State.SAW_OTHER_OP;
320         } else if (source == this.bMultiply) {
321             this.controller.processMultiplyEvent();
322             this.currentState = State.SAW_OTHER_OP;
323         } else if (source == this.bDivide) {
324             this.controller.processDivideEvent();
325             this.currentState = State.SAW_OTHER_OP;
326         } else if (source == this.bPower) {
327             this.controller.processPowerEvent();
328             this.currentState = State.SAW_OTHER_OP;
329         } else if (source == this.bRoot) {
330             this.controller.processRootEvent();
331             this.currentState = State.SAW_OTHER_OP;
332         } else {
333             for (int i = 0; i < DIGIT_BUTTONS; i++) {
334                 if (source == this.bDigits[i]) {
335                     switch (this.currentState) {
336                         case SAW_ENTER_OR_SWAP:
337 
    this.controller.processClearEvent();
338                             break;
339                         case SAW_OTHER_OP:
340 
    this.controller.processEnterEvent();
341 
    this.controller.processClearEvent();
342                             break;
343                         default:
344                             break;
345                     }
```

```
346
   this.controller.processAddNewDigitEvent(i);
347                     this.currentState = State.SAW_DIGIT;
348                     break;
349                 }
350             }
351         }
352         /*
353          * Set the cursor back to normal (because we
   changed it at the beginning
354          * of the method body)
355          */
356         this.setCursor(Cursor.getDefaultCursor());
357     }
358
359}
360
```