```
 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 Ansh Pachauri
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 = new JButton[10];
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       * No argument constructor.
```

```
 70     */
 71    public NNCalcView1() {
 72        // Create the JFrame being extended
 73
 74        /*
 75         * Call the JFrame (superclass) constructor with a
    String parameter to
 76         * name the window in its title bar
 77         */
 78        super("Natural Number Calculator");
 79
 80        // Set up the GUI widgets
    ----------------------------------------
 81
 82        /*
 83         * Set up initial state of GUI to behave like last
    event was "Clear";
 84         * currentState is not a GUI widget per se, but is
    needed to process
 85         * digit button events appropriately
 86         */
 87        this.currentState = State.SAW_CLEAR;
 88
 89        /*
 90         * Create widgets
 91         */
 92        this.tTop = new JTextArea("", TEXT_AREA_HEIGHT,
    TEXT_AREA_WIDTH);
 93        this.tBottom = new JTextArea("", TEXT_AREA_HEIGHT,
    TEXT_AREA_WIDTH);
 94
 95        this.bClear = new JButton("Clear");
 96        this.bSwap = new JButton("Swap");
 97        this.bEnter = new JButton("Enter");
 98        this.bAdd = new JButton("+");
 99        this.bSubtract = new JButton("-");
100        this.bMultiply = new JButton("*");
101        this.bDivide = new JButton("/");
102        this.bPower = new JButton("Power");
```

```
103         this.bRoot = new JButton("Root");
104         for (int i = 0; i < DIGIT_BUTTONS; i++) {
105             this.bDigits[i] = new JButton(String.valueOf(i));
106         }
107 
108         // Set up the GUI widgets
    ----------------------------------------
109 
110         /*
111          * Text areas should wrap lines, and should be read-
    only; they cannot be
112          * edited because allowing keyboard entry would require
    checking whether
113          * entries are digits, which we don't want to have to
    do
114          */
115 
116         this.tTop.setEditable(false);
117         this.tTop.setLineWrap(true);
118         this.tTop.setWrapStyleWord(true);
119         this.tBottom.setEditable(false);
120         this.tBottom.setLineWrap(true);
121         this.tBottom.setWrapStyleWord(true);
122 
123         /*
124          * Initially, the following buttons should be disabled:
    divide (divisor
125          * must not be 0) and root (root must be at least 2) --
    hint: see the
126          * JButton method setEnabled
127          */
128 
129         this.bDivide.setEnabled(false);
130         this.bRoot.setEnabled(false);
131 
132         /*
133          * Create scroll panes for the text areas in case
    number is long enough
134          * to require scrolling
```

```
135         */
136
137         JScrollPane tTopScrollPane = new
    JScrollPane(this.tTop);
138         JScrollPane tBottomScrollPane = new
    JScrollPane(this.tBottom);
139
140         /*
141          * Create main button panel
142          */
143
144         JPanel buttonPanel = new JPanel(new GridLayout(
145                 MAIN_BUTTON_PANEL_GRID_ROWS,
    MAIN_BUTTON_PANEL_GRID_COLUMNS));
146
147         /*
148          * Add the buttons to the main button panel, from left
    to right and top
149          * to bottom
150          */
151
152         buttonPanel.add(this.bDigits[7]);
153         buttonPanel.add(this.bDigits[8]);
154         buttonPanel.add(this.bDigits[9]);
155         buttonPanel.add(this.bAdd);
156         buttonPanel.add(this.bDigits[4]);
157         buttonPanel.add(this.bDigits[5]);
158         buttonPanel.add(this.bDigits[6]);
159         buttonPanel.add(this.bSubtract);
160         buttonPanel.add(this.bDigits[1]);
161         buttonPanel.add(this.bDigits[2]);
162         buttonPanel.add(this.bDigits[3]);
163         buttonPanel.add(this.bMultiply);
164         buttonPanel.add(this.bDigits[0]);
165         buttonPanel.add(this.bPower);
166         buttonPanel.add(this.bRoot);
167         buttonPanel.add(this.bDivide);
168
169         /*
```

```
170          * Create side button panel
171          */
172
173         JPanel sideButtonPanel = new JPanel(new GridLayout(
174                 SIDE_BUTTON_PANEL_GRID_ROWS,
    SIDE_BUTTON_PANEL_GRID_COLUMNS));
175         /*
176          * Add the buttons to the side button panel, from left
    to right and top
177          * to bottom
178          */
179
180         sideButtonPanel.add(this.bClear);
181         sideButtonPanel.add(this.bSwap);
182         sideButtonPanel.add(this.bEnter);
183
184         /*
185          * Create combined button panel organized using flow
    layout, which is
186          * simple and does the right thing: sizes of nested
    panels are natural,
187          * not necessarily equal as with grid layout
188          */
189
190         JPanel combinedButtonPanel = new JPanel(new
    FlowLayout());
191
192         /*
193          * Add the other two button panels to the combined
    button panel
194          */
195
196         combinedButtonPanel.add(buttonPanel);
197         combinedButtonPanel.add(sideButtonPanel);
198
199         /*
200          * Organize main window
201          */
202
```

```
203         this.setLayout(new GridLayout(CALC_GRID_ROWS,
    CALC_GRID_COLUMNS));
204 
205         /*
206          * Add scroll panes and button panel to main window,
    from left to right
207          * and top to bottom
208          */
209 
210         this.add(tTopScrollPane);
211         this.add(tBottomScrollPane);
212         this.add(combinedButtonPanel);
213 
214         // Set up the observers
    ---------------------------------------------
215 
216         /*
217          * Register this object as the observer for all GUI
    events
218          */
219 
220         this.bClear.addActionListener(this);
221         this.bSwap.addActionListener(this);
222         this.bEnter.addActionListener(this);
223         this.bAdd.addActionListener(this);
224         this.bSubtract.addActionListener(this);
225         this.bMultiply.addActionListener(this);
226         this.bDivide.addActionListener(this);
227         this.bPower.addActionListener(this);
228         this.bRoot.addActionListener(this);
229         for (int i = 0; i < DIGIT_BUTTONS; i++) {
230             this.bDigits[i].addActionListener(this);
231         }
232 
233         // Set up the main application window
    --------------------------------
234 
235         /*
236          * Make sure the main window is appropriately sized,
```

```
   exits this program
237          * on close, and becomes visible to the user
238          */
239
240         this.pack();
241         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
242         this.setVisible(true);
243
244     }
245
246     @Override
247     public void registerObserver(NNCalcController controller) {
248
249         this.controller = controller;
250
251     }
252
253     @Override
254     public void updateTopDisplay(NaturalNumber n) {
255
256         String num = n.toString();
257         this.tTop.setText(num);
258
259     }
260
261     @Override
262     public void updateBottomDisplay(NaturalNumber n) {
263
264         String num = n.toString();
265         this.tBottom.setText(num);
266
267     }
268
269     @Override
270     public void updateSubtractAllowed(boolean allowed) {
271
272         this.bSubtract.setEnabled(allowed);
273
274     }
```

```
275
276     @Override
277     public void updateDivideAllowed(boolean allowed) {
278
279         this.bDivide.setEnabled(allowed);
280
281     }
282
283     @Override
284     public void updatePowerAllowed(boolean allowed) {
285
286         this.bPower.setEnabled(allowed);
287
288     }
289
290     @Override
291     public void updateRootAllowed(boolean allowed) {
292
293         this.bRoot.setEnabled(allowed);
294
295     }
296
297     @Override
298     public void actionPerformed(ActionEvent event) {
299         /*
300          * Set cursor to indicate computation on-going; this
    matters only if
301          * processing the event might take a noticeable amount
    of time as seen
302          * by the user
303          */
304
    this.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
305         /*
306          * Determine which event has occurred that we are being
    notified of by
307          * this callback; in this case, the source of the event
    (i.e, the widget
308          * calling actionPerformed) is all we need because only
```

```
   buttons are
309         * involved here, so the event must be a button press;
   in each case,
310         * tell the controller to do whatever is needed to
   update the model and
311         * to refresh the view
312         */
313        Object source = event.getSource();
314        if (source == this.bClear) {
315            this.controller.processClearEvent();
316            this.currentState = State.SAW_CLEAR;
317        } else if (source == this.bSwap) {
318            this.controller.processSwapEvent();
319            this.currentState = State.SAW_ENTER_OR_SWAP;
320        } else if (source == this.bEnter) {
321            this.controller.processEnterEvent();
322            this.currentState = State.SAW_ENTER_OR_SWAP;
323        } else if (source == this.bAdd) {
324            this.controller.processAddEvent();
325            this.currentState = State.SAW_OTHER_OP;
326        } else if (source == this.bSubtract) {
327            this.controller.processSubtractEvent();
328            this.currentState = State.SAW_OTHER_OP;
329        } else if (source == this.bMultiply) {
330            this.controller.processMultiplyEvent();
331            this.currentState = State.SAW_OTHER_OP;
332        } else if (source == this.bDivide) {
333            this.controller.processDivideEvent();
334            this.currentState = State.SAW_OTHER_OP;
335        } else if (source == this.bPower) {
336            this.controller.processPowerEvent();
337            this.currentState = State.SAW_OTHER_OP;
338        } else if (source == this.bRoot) {
339            this.controller.processRootEvent();
340            this.currentState = State.SAW_OTHER_OP;
341        } else {
342            for (int i = 0; i < DIGIT_BUTTONS; i++) {
343                if (source == this.bDigits[i]) {
344                    switch (this.currentState) {
```

```
345                         case SAW_ENTER_OR_SWAP:
346
    this.controller.processClearEvent();
347                             break;
348                         case SAW_OTHER_OP:
349
    this.controller.processEnterEvent();
350
    this.controller.processClearEvent();
351                             break;
352                         default:
353                             break;
354                     }
355                     this.controller.processAddNewDigitEvent(i);
356                     this.currentState = State.SAW_DIGIT;
357                     break;
358                 }
359             }
360         }
361         /*
362          * Set the cursor back to normal (because we changed it
    at the beginning
363          * of the method body)
364          */
365         this.setCursor(Cursor.getDefaultCursor());
366     }
367
368 }
369
```