```
 1 import java.awt.Cursor;
13
14 /**
15  * View class.
16  *
17  * @author Yakob Getu
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       * 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
 93         this.tTop = new JTextArea("", TEXT_AREA_HEIGHT,
    TEXT_AREA_WIDTH);
 94         this.tBottom = new JTextArea("",
    TEXT_AREA_HEIGHT, TEXT_AREA_WIDTH);
 95
 96         this.bClear = new JButton("Clear");
 97         this.bSwap = new JButton("Swap");
 98         this.bEnter = new JButton("Enter");
 99         this.bAdd = new JButton("+");
100         this.bSubtract = new JButton("-");
101         this.bMultiply = new JButton("*");
102         this.bDivide = new JButton("/");
103         this.bPower = new JButton("Power");
104         this.bRoot = new JButton("Root");
105
106         this.bDigits = new JButton[DIGIT_BUTTONS];
107
108         for (int i = 0; i < DIGIT_BUTTONS; i++) {
109             this.bDigits[i] = new
    JButton(Integer.toString(i));
110         }
111
```

```
112         // Set up the GUI widgets
    ----------------------------------------
113
114         /*
115          * Text areas should wrap lines, and should be
    read-only; they cannot be
116          * edited because allowing keyboard entry would
    require checking whether
117          * entries are digits, which we don't want to
    have to do
118          */
119         this.tTop.setEditable(false);
120         this.tTop.setLineWrap(true);
121         this.tTop.setWrapStyleWord(true);
122         this.tBottom.setEditable(false);
123         this.tBottom.setLineWrap(true);
124         this.tBottom.setWrapStyleWord(true);
125
126         /*
127          * Initially, the following buttons should be
    disabled: divide (divisor
128          * must not be 0) and root (root must be at least
    2) -- hint: see the
129          * JButton method setEnabled
130          */
131         this.bDivide.setEnabled(false);
132         this.bRoot.setEnabled(false);
133
134         /*
135          * Create scroll panes for the text areas in case
    number is long enough
136          * to require scrolling
137          */
138         JScrollPane tTopScrollPane = new
    JScrollPane(this.tTop);
139         JScrollPane tBottomScrollPane = new
    JScrollPane(this.tBottom);
140
141         /*
```

```
142         * Create main button panel
143         */
144        JPanel mainButtonPanel = 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        mainButtonPanel.add(this.bDigits[7]);
152        mainButtonPanel.add(this.bDigits[8]);
153        mainButtonPanel.add(this.bDigits[9]);
154        mainButtonPanel.add(this.bAdd);
155        mainButtonPanel.add(this.bDigits[4]);
156        mainButtonPanel.add(this.bDigits[5]);
157        mainButtonPanel.add(this.bDigits[6]);
158        mainButtonPanel.add(this.bSubtract);
159        mainButtonPanel.add(this.bDigits[1]);
160        mainButtonPanel.add(this.bDigits[2]);
161        mainButtonPanel.add(this.bDigits[3]);
162        mainButtonPanel.add(this.bMultiply);
163        mainButtonPanel.add(this.bDigits[0]);
164        mainButtonPanel.add(this.bPower);
165        mainButtonPanel.add(this.bRoot);
166        mainButtonPanel.add(this.bDivide);
167
168        /*
169         * Create side button panel
170         */
171        JPanel sideButtonPanel = new JPanel(new
    GridLayout(
172                SIDE_BUTTON_PANEL_GRID_ROWS,
    SIDE_BUTTON_PANEL_GRID_COLUMNS));
173
174        /*
175         * Add the buttons to the side button panel, from
```

```
    left to right and top
176          * to bottom
177          */
178         sideButtonPanel.add(this.bClear);
179         sideButtonPanel.add(this.bSwap);
180         sideButtonPanel.add(this.bEnter);
181 
182         /*
183          * Create combined button panel organized using
    flow layout, which is
184          * simple and does the right thing: sizes of
    nested panels are natural,
185          * not necessarily equal as with grid layout
186          */
187         JPanel combinedButtonPanel = new JPanel(new
    FlowLayout());
188 
189         /*
190          * Add the other two button panels to the
    combined button panel
191          */
192         combinedButtonPanel.add(mainButtonPanel);
193         combinedButtonPanel.add(sideButtonPanel);
194 
195         /*
196          * Organize main window
197          */
198         this.setLayout(new GridLayout(CALC_GRID_ROWS,
    CALC_GRID_COLUMNS));
199 
200         /*
201          * Add scroll panes and button panel to main
    window, from left to right
202          * and top to bottom
203          */
204         this.add(tTopScrollPane);
205         this.add(tBottomScrollPane);
206         this.add(combinedButtonPanel);
207 
```

```
208         // Set up the observers
    ----------------------------------------
209
210         /*
211          * Register this object as the observer for all
    GUI events
212          */
213         this.bClear.addActionListener(this);
214         this.bSwap.addActionListener(this);
215         this.bEnter.addActionListener(this);
216         this.bAdd.addActionListener(this);
217         this.bSubtract.addActionListener(this);
218         this.bMultiply.addActionListener(this);
219         this.bDivide.addActionListener(this);
220         this.bPower.addActionListener(this);
221         this.bRoot.addActionListener(this);
222         this.bDigits[0].addActionListener(this);
223         this.bDigits[1].addActionListener(this);
224         this.bDigits[2].addActionListener(this);
225         this.bDigits[3].addActionListener(this);
226         this.bDigits[4].addActionListener(this);
227         this.bDigits[5].addActionListener(this);
228         this.bDigits[6].addActionListener(this);
229         this.bDigits[7].addActionListener(this);
230         this.bDigits[8].addActionListener(this);
231         this.bDigits[9].addActionListener(this);
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         this.pack();
240
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
241         this.setVisible(true);
```

```
242
243     }
244
245     //Update UI components according to model state
    changes.
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         this.tTop.setText(n.toString());
257
258     }
259
260     @Override
261     public void updateBottomDisplay(NaturalNumber n) {
262
263         this.tBottom.setText(n.toString());
264
265     }
266
267     @Override
268     public void updateSubtractAllowed(boolean allowed) {
269
270         this.bSubtract.setEnabled(allowed);
271
272     }
273
274     @Override
275     public void updateDivideAllowed(boolean allowed) {
276
277         this.bDivide.setEnabled(allowed);
278
```

```
279     }
280 
281     @Override
282     public void updatePowerAllowed(boolean allowed) {
283 
284         this.bPower.setEnabled(allowed);
285 
286     }
287 
288     @Override
289     public void updateRootAllowed(boolean allowed) {
290 
291         this.bRoot.setEnabled(allowed);
292 
293     }
294 
295     //Handle action events from UI elements.
296     @Override
297     public void actionPerformed(ActionEvent event) {
298         /*
299          * Set cursor to indicate computation on-going;
    this matters only if
300          * processing the event might take a noticeable
    amount of time as seen
301          * by the user
302          */
303 
    this.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CUR
    SOR));
304         /*
305          * Determine which event has occurred that we are
    being notified of by
306          * this callback; in this case, the source of the
    event (i.e, the widget
307          * calling actionPerformed) is all we need
    because only buttons are
308          * involved here, so the event must be a button
    press; in each case,
309          * tell the controller to do whatever is needed
```

```
    to update the model and
310          * to refresh the view
311          */
312         Object source = event.getSource();
313         if (source == this.bClear) {
314             this.controller.processClearEvent();
315             this.currentState = State.SAW_CLEAR;
316         } else if (source == this.bSwap) {
317             this.controller.processSwapEvent();
318             this.currentState = State.SAW_ENTER_OR_SWAP;
319         } else if (source == this.bEnter) {
320             this.controller.processEnterEvent();
321             this.currentState = State.SAW_ENTER_OR_SWAP;
322         } else if (source == this.bAdd) {
323             this.controller.processAddEvent();
324             this.currentState = State.SAW_OTHER_OP;
325         } else if (source == this.bSubtract) {
326             this.controller.processSubtractEvent();
327             this.currentState = State.SAW_OTHER_OP;
328         } else if (source == this.bMultiply) {
329             this.controller.processMultiplyEvent();
330             this.currentState = State.SAW_OTHER_OP;
331         } else if (source == this.bDivide) {
332             this.controller.processDivideEvent();
333             this.currentState = State.SAW_OTHER_OP;
334         } else if (source == this.bPower) {
335             this.controller.processPowerEvent();
336             this.currentState = State.SAW_OTHER_OP;
337         } else if (source == this.bRoot) {
338             this.controller.processRootEvent();
339             this.currentState = State.SAW_OTHER_OP;
340         } else {
341             for (int i = 0; i < DIGIT_BUTTONS; i++) {
342                 if (source == this.bDigits[i]) {
343                     switch (this.currentState) {
344                         case SAW_ENTER_OR_SWAP:
345 
    this.controller.processClearEvent();
346                             break;
```

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