```
 1 import java.awt.Cursor;
13
14 /**
15  * View class.
16  *
17  * @author Put your name here
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          /*
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          this.tTop = new JTextArea("", TEXT_AREA_HEIGHT, TEXT_AREA_WIDTH);
90          this.tBottom = new JTextArea("", TEXT_AREA_HEIGHT, TEXT_AREA_WIDTH);
91          this.bClear = new JButton("Clear");
92          this.bSwap = new JButton("Swap");
93          this.bEnter = new JButton("Enter");
94          this.bAdd = new JButton("+");
95          this.bSubtract = new JButton("-");
96          this.bMultiply = new JButton("*");
97          this.bDivide = new JButton("/");
98          this.bPower = new JButton("Power");
99          this.bRoot = new JButton("Root");
100         this.bDigits = new JButton[DIGIT_BUTTONS];
101         for (int i = 0; i < DIGIT_BUTTONS; i++) {
102             this.bDigits[i] = new JButton(Integer.toString(i));
103         }
104
105         // Set up the GUI widgets --------------------------------------------
106
107         /*
108          * Text areas should wrap lines, and should be read-only; they cannot be
109          * edited because allowing keyboard entry would require checking whether
110          * entries are digits, which we don't want to have to do
111          */
112         this.tTop.setEditable(false);
113         this.tTop.setLineWrap(true);
114         this.tTop.setWrapStyleWord(true);
115         this.tBottom.setEditable(false);
116         this.tBottom.setLineWrap(true);
117         this.tBottom.setWrapStyleWord(true);
118         /*
119          * Initially, the following buttons should be disabled: divide (divisor
120          * must not be 0) and root (root must be at least 2) -- hint: see the
121          * JButton method setEnabled
122          */
123         this.bDivide.setEnabled(false);
124         this.bRoot.setEnabled(false);
125
126         /*
127          * Create scroll panes for the text areas in case number is long enough
128          * to require scrolling
129          */
130         JScrollPane tTopScrollPane = new JScrollPane(this.tTop);
131         JScrollPane tBottomScrollPane = new JScrollPane(this.tBottom);
132         /*
133          * Create main button panel
134          */
135         JPanel mainButtonPanel = new JPanel(new GridLayout(
```

```
136                 MAIN_BUTTON_PANEL_GRID_ROWS, MAIN_BUTTON_PANEL_GRID_COLUMNS));
137         /*
138          * Add the buttons to the main button panel, from left to right and top
139          * to bottom
140          */
141         final int ten = 10;
142         final int seven = 7;
143         final int four = 4;
144         /*
145          * add 7-10 and +
146          */
147         for (int i = seven; i < ten; i++) {
148             mainButtonPanel.add(this.bDigits[i]);
149         }
150         mainButtonPanel.add(this.bAdd);
151         /*
152          * add 4-7 and -
153          */
154         for (int i = four; i < seven; i++) {
155             mainButtonPanel.add(this.bDigits[i]);
156         }
157         mainButtonPanel.add(this.bSubtract);
158         /*
159          * add 1-3 and +
160          */
161         for (int i = 1; i < four; i++) {
162             mainButtonPanel.add(this.bDigits[i]);
163         }
164         mainButtonPanel.add(this.bMultiply);
165         /*
166          * add 0,power,root and /
167          */
168         mainButtonPanel.add(this.bDigits[0]);
169         mainButtonPanel.add(this.bPower);
170         mainButtonPanel.add(this.bRoot);
171         mainButtonPanel.add(this.bDivide);
172 
173         /*
174          * Create side button panel
175          */
176         JPanel sidePanel = new JPanel(new GridLayout(
177                 SIDE_BUTTON_PANEL_GRID_ROWS, SIDE_BUTTON_PANEL_GRID_COLUMNS));
178         /*
179          * Add the buttons to the side button panel, from left to right and top
180          * to bottom
181          */
182         sidePanel.add(this.bClear);
183         sidePanel.add(this.bSwap);
184         sidePanel.add(this.bEnter);
185 
186         /*
187          * Create combined button panel organized using flow layout, which is
188          * simple and does the right thing: sizes of nested panels are natural,
189          * not necessarily equal as with grid layout
190          */
191         JPanel combinedButtonPanel = new JPanel(new FlowLayout());
192         /*
193          * Add the other two button panels to the combined button panel
194          */
195         combinedButtonPanel.add(mainButtonPanel);
196         combinedButtonPanel.add(sidePanel);
197 
```

```
198         /*
199          * Organize main window
200          */
201         this.setLayout(new GridLayout(CALC_GRID_ROWS, CALC_GRID_COLUMNS));
202         /*
203          * Add scroll panes and button panel to main window, from left to right
204          * and top to bottom
205          */
206         this.add(tTopScrollPane);
207         this.add(tBottomScrollPane);
208         this.add(combinedButtonPanel);
209         // Set up the observers ----------------------------------------------
210 
211         /*
212          * Register this object as the observer for all GUI events
213          */
214         this.bAdd.addActionListener(this);
215         this.bSubtract.addActionListener(this);
216         this.bMultiply.addActionListener(this);
217         this.bPower.addActionListener(this);
218         this.bRoot.addActionListener(this);
219         this.bDivide.addActionListener(this);
220         this.bClear.addActionListener(this);
221         this.bSwap.addActionListener(this);
222         this.bEnter.addActionListener(this);
223         for (int i = 0; i < DIGIT_BUTTONS; i++) {
224             this.bDigits[i].addActionListener(this);
225         }
226         // Set up the main application window -------------------------------
227 
228         /*
229          * Make sure the main window is appropriately sized, exits this program
230          * on close, and becomes visible to the user
231          */
232         this.pack();
233         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
234         this.setVisible(true);
235     }
236 
237     @Override
238     public void registerObserver(NNCalcController controller) {
239 
240         this.controller = controller;
241 
242     }
243 
244     @Override
245     public void updateTopDisplay(NaturalNumber n) {
246 
247         this.tTop.setText(n.toString());
248 
249     }
250 
251     @Override
252     public void updateBottomDisplay(NaturalNumber n) {
253 
254         this.tBottom.setText(n.toString());
255 
256     }
257 
258     @Override
259     public void updateSubtractAllowed(boolean allowed) {
```

```
260
261         this.bSubtract.setEnabled(allowed);
262
263     }
264
265     @Override
266     public void updateDivideAllowed(boolean allowed) {
267
268         this.bDivide.setEnabled(allowed);
269
270     }
271
272     @Override
273     public void updatePowerAllowed(boolean allowed) {
274
275         this.bPower.setEnabled(allowed);
276
277     }
278
279     @Override
280     public void updateRootAllowed(boolean allowed) {
281
282         this.bRoot.setEnabled(allowed);
283
284     }
285
286     @Override
287     public void actionPerformed(ActionEvent event) {
288         /*
289          * Set cursor to indicate computation on-going; this matters only if
290          * processing the event might take a noticeable amount of time as seen
291          * by the user
292          */
293         this.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
294         /*
295          * Determine which event has occurred that we are being notified of by
296          * this callback; in this case, the source of the event (i.e, the widget
297          * calling actionPerformed) is all we need because only buttons are
298          * involved here, so the event must be a button press; in each case,
299          * tell the controller to do whatever is needed to update the model and
300          * to refresh the view
301          */
302         Object source = event.getSource();
303         if (source == this.bClear) {
304             this.controller.processClearEvent();
305             this.currentState = State.SAW_CLEAR;
306         } else if (source == this.bSwap) {
307             this.controller.processSwapEvent();
308             this.currentState = State.SAW_ENTER_OR_SWAP;
309         } else if (source == this.bEnter) {
310             this.controller.processEnterEvent();
311             this.currentState = State.SAW_ENTER_OR_SWAP;
312         } else if (source == this.bAdd) {
313             this.controller.processAddEvent();
314             this.currentState = State.SAW_OTHER_OP;
315         } else if (source == this.bSubtract) {
316             this.controller.processSubtractEvent();
317             this.currentState = State.SAW_OTHER_OP;
318         } else if (source == this.bMultiply) {
319             this.controller.processMultiplyEvent();
320             this.currentState = State.SAW_OTHER_OP;
321         } else if (source == this.bDivide) {
```

```
322             this.controller.processDivideEvent();
323             this.currentState = State.SAW_OTHER_OP;
324         } else if (source == this.bPower) {
325             this.controller.processPowerEvent();
326             this.currentState = State.SAW_OTHER_OP;
327         } else if (source == this.bRoot) {
328             this.controller.processRootEvent();
329             this.currentState = State.SAW_OTHER_OP;
330         } else {
331             for (int i = 0; i < DIGIT_BUTTONS; i++) {
332                 if (source == this.bDigits[i]) {
333                     switch (this.currentState) {
334                         case SAW_ENTER_OR_SWAP:
335                             this.controller.processClearEvent();
336                             break;
337                         case SAW_OTHER_OP:
338                             this.controller.processEnterEvent();
339                             this.controller.processClearEvent();
340                             break;
341                         default:
342                             break;
343                     }
344                     this.controller.processAddNewDigitEvent(i);
345                     this.currentState = State.SAW_DIGIT;
346                     break;
347                 }
348             }
349         }
350         /*
351          * Set the cursor back to normal (because we changed it at the beginning
352          * of the method body)
353          */
354         this.setCursor(Cursor.getDefaultCursor());
355     }
356
357 }
358
```