

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

```
88
89     /*
90     * Create widgets
91     */
92     this.tTop = new JTextArea("0", TEXT_AREA_HEIGHT,
TEXT_AREA_WIDTH);
93     this.tBottom = new JTextArea("0", TEXT_AREA_HEIGHT,
TEXT_AREA_WIDTH);
94     this.bDigits = new JButton[DIGIT_BUTTONS];
95     for (int i = 0; i < DIGIT_BUTTONS; i++) {
96         String d = Integer.toString(i);
97         this.bDigits[i] = new JButton(d);
98     }
99     this.bClear = new JButton("Clear");
100    this.bSwap = new JButton("Swap");
101    this.bEnter = new JButton("Enter");
102    this.bAdd = new JButton("+");
103    this.bSubtract = new JButton("-");
104    this.bMultiply = new JButton("*");
105    this.bDivide = new JButton("/");
106    this.bPower = new JButton("Power");
107    this.bRoot = new JButton("Root");
108
109    // Set up the GUI widgets
-----
110
111    /*
112    * Text areas should wrap lines, and should be read-only;
they cannot be
113    * edited because allowing keyboard entry would require
checking whether
114    * entries are digits, which we don't want to have to do
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         this.bDivide.setEnabled(false);
129         this.bRoot.setEnabled(false);
130
131         /*
132         * Create scroll panes for the text areas in case number
    is long enough
133         * to require scrolling
134         */
135         JScrollPane topScrollPane = new JScrollPane(this.tTop);
136         JScrollPane bottomScrollPane = new
    JScrollPane(this.tBottom);
137
138         /*
139         * Create main button panel
140         */
141         JPanel mainButtonPanel = new JPanel(new GridLayout(
142             MAIN_BUTTON_PANEL_GRID_ROWS,
    MAIN_BUTTON_PANEL_GRID_COLUMNS));
143
144         /*
145         * Add the buttons to the main button panel, from left to
    right and top
146         * to bottom
147         */
148         mainButtonPanel.add(this.bDigits 7);
149         mainButtonPanel.add(this.bDigits 8);
150         mainButtonPanel.add(this.bDigits 9);
151         mainButtonPanel.add(this.bAdd);
152         mainButtonPanel.add(this.bDigits 4);
153         mainButtonPanel.add(this.bDigits 5);
154         mainButtonPanel.add(this.bDigits 6);
155         mainButtonPanel.add(this.bSubtract);
156         mainButtonPanel.add(this.bDigits 1);
157         mainButtonPanel.add(this.bDigits 2);
158         mainButtonPanel.add(this.bDigits 3);
159         mainButtonPanel.add(this.bMultiply);
160         mainButtonPanel.add(this.bDigits 0);
161         mainButtonPanel.add(this.bPower);
162         mainButtonPanel.add(this.bRoot);
163         mainButtonPanel.add(this.bDivide);
164
```

```
165      /*
166      * Create side button panel
167      */
168      JPanel sideButtonPanel = new JPanel(new GridLayout(
169          SIDE_BUTTON_PANEL_GRID_ROWS,
170          SIDE_BUTTON_PANEL_GRID_COLUMNS));
171
172      /*
173      * Add the buttons to the side button panel, from left to
174      right and top
175      * to bottom
176      */
177      sideButtonPanel.add(this.bClear);
178      sideButtonPanel.add(this.bSwap);
179      sideButtonPanel.add(this.bEnter);
180
181      /*
182      * Create combined button panel organized using flow
183      layout, which is
184      * simple and does the right thing: sizes of nested panels
185      are natural,
186      * not necessarily equal as with grid layout
187      */
188      JPanel combinedPanel = new JPanel(new FlowLayout());
189
190      /*
191      * Add the other two button panels to the combined button
192      panel
193      */
194      combinedPanel.add(mainButtonPanel);
195      combinedPanel.add(sideButtonPanel);
196
197      /*
198      * Organize main window
199      */
200      this.setLayout(new GridLayout(CALC_GRID_ROWS,
201          CALC_GRID_COLUMNS));
202
203      /*
204      * Add scroll panes and button panel to main window, from
205      left to right
206      * and top to bottom
207      */
208      this.add(this.tTop);
```

```
202         this.add(this.tBottom);
203         this.add(combinedPanel);
204
205         // Set up the observers
-----
206
207         /*
208          * Register this object as the observer for all GUI events
209          */
210         this.bAdd.addActionListener(this);
211         this.bSubtract.addActionListener(this);
212         this.bMultiply.addActionListener(this);
213         this.bDivide.addActionListener(this);
214         this.bPower.addActionListener(this);
215         this.bRoot.addActionListener(this);
216         for (int i = 0; i < DIGIT_BUTTONS; i++) {
217             this.bDigits[i].addActionListener(this);
218         }
219         this.bClear.addActionListener(this);
220         this.bSwap.addActionListener(this);
221         this.bEnter.addActionListener(this);
222
223         // Set up the main application window
-----
224
225         /*
226          * Make sure the main window is appropriately sized, exits
this program
227          * on close, and becomes visible to the user
228          */
229         this.pack();
230         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
231         this.setVisible(true);
232
233     }
234
235     @Override
236     public void registerObserver(NNCalcController controller) {
237
238         this.controller = controller;
239
240     }
241
242     @Override
```

```
243     public void updateTopDisplay(NaturalNumber n) {
244         this.tTop.setText(n.toString());
245     }
246
247
248
249     @Override
250     public void updateBottomDisplay(NaturalNumber n) {
251         this.tBottom.setText(n.toString());
252     }
253
254
255
256     @Override
257     public void updateSubtractAllowed(boolean allowed) {
258         this.bSubtract.setEnabled(allowed);
259     }
260
261
262
263     @Override
264     public void updateDivideAllowed(boolean allowed) {
265         this.bDivide.setEnabled(allowed);
266     }
267
268
269
270     @Override
271     public void updatePowerAllowed(boolean allowed) {
272         this.bPower.setEnabled(allowed);
273     }
274
275
276
277     @Override
278     public void updateRootAllowed(boolean allowed) {
279         this.bRoot.setEnabled(allowed);
280     }
281
282
283
284     @Override
285     public void actionPerformed(ActionEvent event) {
286         /*
```

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



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