

```
1 import components.naturalnumber.NaturalNumber;
2
3
4 /**
5  * Controller class.
6  *
7  * @author Put your name here
8  */
9 public final class NNCalcController1 implements NNCalcController {
10
11     /**
12      * Model object.
13      */
14     private final NNCalcModel model;
15
16     /**
17      * View object.
18      */
19     private final NNCalcView view;
20
21     /**
22      * Useful constants.
23      */
24     private static final NaturalNumber TWO = new NaturalNumber2(2),
25         INT_LIMIT = new NaturalNumber2(Integer.MAX_VALUE);
26
27     /**
28      * Updates this.view to display this.model, and to allow only operations
29      * that are legal given this.model.
30      *
31      * @param model
32      *         the model
33      * @param view
34      *         the view
35      * @ensures [view has been updated to be consistent with model]
36      */
37     private static void updateViewToMatchModel(NNCalcModel model,
38         NNCalcView view) {
39
40         NaturalNumber input = model.bottom();
41         NaturalNumber output = model.top();
42
43         view.updateBottomDisplay(input);
44         view.updateTopDisplay(output);
45         /**
46          * make sure the number would be smaller than the limit of the int
47          */
48         if (input.compareTo(INT_LIMIT) < 0) {
49             view.updatePowerAllowed(true);
50         } else {
51             view.updatePowerAllowed(false);
52         }
53         /**
54          * make sure the number which are subtracted would be correct
55          */
56         if (input.compareTo(output) < 0) {
57             view.updateSubtractAllowed(true);
58         } else {
59             view.updateSubtractAllowed(false);
60         }
61         /**
62          * make sure the number would be above 0
63          */
64     }
```

```
64         if (input.compareTo(TWO.newInstance()) > 0) {
65             view.updateDivideAllowed(true);
66         } else {
67             view.updateDivideAllowed(false);
68         }
69         /*
70          * make sure the number would be bigger than two and smaller than the
71          * limit of the int
72          */
73         if (input.compareTo(TWO) >= 0 && input.compareTo(INT_LIMIT) <= 0) {
74             view.updateRootAllowed(true);
75         } else {
76             view.updateRootAllowed(false);
77         }
78     }
79
80     /**
81      * Constructor.
82      *
83      * @param model
84      *         model to connect to
85      * @param view
86      *         view to connect to
87      */
88     public NNCalcController1(NNCalcModel model, NNCalcView view) {
89         this.model = model;
90         this.view = view;
91         updateViewToMatchModel(model, view);
92     }
93
94     @Override
95     public void processClearEvent() {
96         /*
97          * Get alias to bottom from model
98          */
99         NaturalNumber bottom = this.model.bottom();
100        /*
101         * Update model in response to this event
102         */
103        bottom.clear();
104        /*
105         * Update view to reflect changes in model
106         */
107        updateViewToMatchModel(this.model, this.view);
108    }
109
110    @Override
111    public void processSwapEvent() {
112        /*
113         * Get aliases to top and bottom from model
114         */
115        NaturalNumber top = this.model.top();
116        NaturalNumber bottom = this.model.bottom();
117        /*
118         * Update model in response to this event
119         */
120        NaturalNumber temp = top.newInstance();
121        temp.transferFrom(top);
122        top.transferFrom(bottom);
123        bottom.transferFrom(temp);
124        /*
125         * Update view to reflect changes in model
```

```
126         */
127         updateViewToMatchModel(this.model, this.view);
128     }
129
130     @Override
131     public void processEnterEvent() {
132         /*
133          * Get aliases to top and bottom from model
134          */
135         NaturalNumber top = this.model.top();
136         NaturalNumber bottom = this.model.bottom();
137         /*
138          * Update model in response to this event
139          */
140         top.copyFrom(bottom);
141         /*
142          * Update view to reflect changes in model
143          */
144         updateViewToMatchModel(this.model, this.view);
145     }
146
147     @Override
148     public void processAddEvent() {
149         /*
150          * Get aliases to top and bottom from model
151          */
152         NaturalNumber top = this.model.top();
153         NaturalNumber bottom = this.model.bottom();
154         /*
155          * Update model in response to this event
156          */
157         bottom.add(top);
158         top.clear();
159         /*
160          * Update view to reflect changes in model
161          */
162         updateViewToMatchModel(this.model, this.view);
163     }
164
165     @Override
166     public void processSubtractEvent() {
167         /*
168          * Get aliases to top and bottom from model
169          */
170         NaturalNumber top = this.model.top();
171         NaturalNumber bottom = this.model.bottom();
172         /*
173          * Update model in response to this event
174          */
175         top.subtract(bottom);
176         bottom.transferFrom(top);
177         /*
178          * Update view to reflect changes in model
179          */
180         updateViewToMatchModel(this.model, this.view);
181     }
182
183     @Override
184     public void processMultiplyEvent() {
185         /*
186          * Get aliases to top and bottom from model
187          */
```

```
188     NaturalNumber top = this.model.top();
189     NaturalNumber bottom = this.model.bottom();
190     /*
191      * Update model in response to this event
192      */
193     bottom.multiply(top);
194     top.clear();
195     /*
196      * Update view to reflect changes in model
197      */
198     updateViewToMatchModel(this.model, this.view);
199 }
200
201 @Override
202 public void processDivideEvent() {
203     /*
204      * Get aliases to top and bottom from model
205      */
206     NaturalNumber top = this.model.top();
207     NaturalNumber bottom = this.model.bottom();
208     /*
209      * Update model in response to this event
210      */
211     NaturalNumber tem = new NaturalNumber2();
212     tem = top.divide(bottom);
213     bottom.copyFrom(top);
214     top.transferFrom(tem);
215     /*
216      * Update view to reflect changes in model
217      */
218     updateViewToMatchModel(this.model, this.view);
219 }
220
221 @Override
222 public void processPowerEvent() {
223     /*
224      * Get aliases to top and bottom from model
225      */
226     NaturalNumber top = this.model.top();
227     NaturalNumber bottom = this.model.bottom();
228     /*
229      * Update model in response to this event
230      */
231     top.power(bottom.toInt());
232     bottom.transferFrom(top);
233     /*
234      * Update view to reflect changes in model
235      */
236     updateViewToMatchModel(this.model, this.view);
237 }
238
239 @Override
240 public void processRootEvent() {
241     /*
242      * Get aliases to top and bottom from model
243      */
244     NaturalNumber top = this.model.top();
245     NaturalNumber bottom = this.model.bottom();
246     /*
247      * Update model in response to this event
248      */
249     top.root(bottom.toInt());
```

```
250         bottom.transferFrom(top);
251         /*
252          * Update view to reflect changes in model
253          */
254         updateViewToMatchModel(this.model, this.view);
255     }
256
257     @Override
258     public void processAddNewDigitEvent(int digit) {
259         /*
260          * Get aliases to bottom from model
261          */
262         NaturalNumber bottom = this.model.bottom();
263         /*
264          * Update model in response to this event
265          */
266         bottom.multiplyBy10(digit);
267         /*
268          * Update view to reflect changes in model
269          */
270         updateViewToMatchModel(this.model, this.view);
271     }
272
273 }
274
```