```
 2  * @(#)UndoManager.java
11
12 package p1;
13
14 import CH.ifa.draw.util.Undoable;
17
18 /**
19  * This class manages all the undoable commands. It keeps track of all
20  * the modifications done through user interactions.
21  *
22  * @version <$CURRENT_VERSION$>
23  */
24 public class UndoManager {
25     /**
26      * Maximum default buffer size for undo and redo stack
27      */
28     public static final int DEFAULT_BUFFER_SIZE = 20;
29
30     /**
31      * Collection of undo activities
32      */
33     private Vector<Undoable> redoStack;
34
35     /**
36      * Collection of undo activities
37      */
38     private Vector<Undoable> undoStack;
39     private int maxStackCapacity;
40
41     public UndoManager() {
42         this(DEFAULT_BUFFER_SIZE);
43     }
44
45     public UndoManager(int newUndoStackSize) {
46         maxStackCapacity = newUndoStackSize;
47         undoStack = new Vector<Undoable>(maxStackCapacity);
48         redoStack = new Vector<Undoable>(maxStackCapacity);
49     }
50
51     public void pushUndo(Undoable undoActivity) {
52         if (undoActivity.isUndoable()) {
53             // If buffersize exceeds, remove the oldest command
54             if (getUndoSize() >= maxStackCapacity) {
55                 undoStack.removeElementAt(0);
56             }
57             undoStack.addElement(undoActivity);
58         }
59         else {
60             // a not undoable activity clears the stack because
61             // the last activity does not correspond with the
62             // last undo activity
63             undoStack = new Vector<Undoable>(maxStackCapacity);
64         }
65     }
66
67     public void pushRedo(Undoable redoActivity) {
68         if (redoActivity.isRedoable()) {
69             // If buffersize exceeds, remove the oldest command
70             if (getRedoSize() >= maxStackCapacity) {
```

```
71                  redoStack.removeElementAt(0);
72              }
73              // add redo activity only if it is not already the last
74              // one in the buffer
75              if ((getRedoSize() == 0) || (peekRedo() != redoActivity)) {
76                  redoStack.addElement(redoActivity);
77              }
78          }
79          else {
80              // a not undoable activity clears the tack because
81              // the last activity does not correspond with the
82              // last undo activity
83              redoStack = new Vector<Undoable>(maxStackCapacity);
84          }
85      }
86
87      public boolean isUndoable() {
88          if (getUndoSize() > 0) {
89              return ((Undoable)undoStack.lastElement()).isUndoable();
90          }
91          else {
92              return false;
93          }
94      }
95
96      public boolean isRedoable() {
97          if (getRedoSize() > 0) {
98              return ((Undoable)redoStack.lastElement()).isRedoable();
99          }
100         else {
101             return false;
102         }
103     }
104
105     protected Undoable peekUndo() {
106         if (getUndoSize() > 0) {
107             return (Undoable) undoStack.lastElement();
108         }
109         else {
110             return null;
111         }
112     }
113
114     protected Undoable peekRedo() {
115         if (getRedoSize() > 0) {
116             return (Undoable) redoStack.lastElement();
117         }
118         else {
119             return null;
120         }
121     }
122
123     /**
124      * Returns the current size of undo buffer.
125      */
126     public int getUndoSize() {
127         return undoStack.size();
128     }
129
```

```
130     /**
131      * Returns the current size of redo buffer.
132      */
133     public int getRedoSize() {
134         return redoStack.size();
135     }
136 
137     /**
138      * Throw NoSuchElementException if there is none
139      */
140     public Undoable popUndo() {
141         // Get the last element - throw NoSuchElementException if there is none
142         Undoable lastUndoable = peekUndo();
143 
144         // Remove it from undo collection
145         undoStack.removeElementAt(getUndoSize() - 1);
146 
147         return lastUndoable;
148     }
149 
150     /**
151      * Throw NoSuchElementException if there is none
152      */
153     public Undoable popRedo() {
154         // Get the last element - throw NoSuchElementException if there is none
155         Undoable lastUndoable = peekRedo();
156 
157         // Remove it from undo collection
158         redoStack.removeElementAt(getRedoSize() - 1);
159 
160         return lastUndoable;
161     }
162 
163     public void clearUndos() {
164         clearStack(undoStack);
165     }
166 
167     public void clearRedos() {
168         clearStack(redoStack);
169     }
170 
171     protected void clearStack(Vector<Undoable> clearStack) {
172         clearStack.removeAllElements();
173     }
174 }
175 
```