```
 1
 2 import java.util.*;
 3 import java.io.*;
 4
 5 /**
 6  * Melody controls Nodes and Play Songs.
 7  *
 8  * <ul>
 9  * <li> Name: Melody.java
10  * <li> Description: Melody
11  * <li> Class: Java 145
12  * <li> Instructor: Ken Hang
13  * <li> Date: Feb 4 2015
14  * </ul>
15  *
16  * @author  Hai H Nguyen (Bill)
17  * @version Winter 2015
18  */
19
20 public class Melody {
21     private Queue<Note> notes;
22
23     private double length;
24
25     private int size;
26
27     public static int repeat = 1;   // How many time all melody should
28                                     // rewind a repeat section
29
30     /**
31      * Main Constructor
32      * Initialize Node Queue and Length
33      */
34     public Melody(){
35         notes = new LinkedList<Note>();
36
37         length = 0;
38
39         size = 0;
40     }
41
42     /**
43      * Scan melody files
44      * @param in       Scanner object with path to file
45      */
46     public void input (Scanner in){
47         while (in.hasNext()){
48             double duration = in.nextDouble();
49
50             Pitch node = Pitch.valueOf(in.next());
51
52             Note nextNote = (node.equals(Pitch.R)) ?
53                     (new Note (duration, node, in.next().equals("true"))):
54                     (new Note (duration, node, in.nextInt(),
55                             Accidental.valueOf(in.next()),
56                             in.next().equals("true")));
57
58             notes.add (nextNote);
59
60             length += nextNote.getDuration();
61         }
62
63         size = notes.size();
64     }
65
66     /**
67      *  Print the Song, one note per line
68      * @param out       PrintStream object to print out
```

```
 69      */
 70     public void output (PrintStream out){
 71         for (int i = 0 ; i < size; ++i){
 72             notes.add(notes.peek());
 73 
 74             out.println(notes.remove().toString());
 75         }
 76     }
 77 
 78     /**
 79      * Change The tempo of the Melody by a Factor
 80      * @param tempo     Factor to alter the Current tempo
 81      */
 82     public void changeTempo (double tempo){
 83         length *= tempo;
 84 
 85         for (int i = 0; i < size; ++i){
 86             notes.peek().setDuration(notes.peek().getDuration() * tempo);
 87 
 88             notes.add(notes.remove());
 89         }
 90     }
 91 
 92     /**
 93      * Get the Length of the Melody
 94      * @return      Length of the Melody
 95      */
 96     public double getLength (){
 97         return length;
 98     }
 99 
100     /**
101      * Reverse the Order of nodes in the Melody
102      */
103     public void reverse(){
104         Stack<Note> noteStack = new Stack<Note>();
105 
106         while (!notes.isEmpty()) {
107             noteStack.push(notes.remove());
108         }
109 
110         while (!noteStack.isEmpty()){
111             notes.add(noteStack.pop());
112         }
113     }
114 
115     /**
116      * Append the given other Melody to the Current melody
117      * @param other     Melody to be appended
118      */
119     public void append (Melody other){
120         Queue<Note> noteQueue = new LinkedList<Note>(other.notes); // Preserve
121 
122         for (int i = 0; i < other.size; ++i){
123             notes.add(noteQueue.remove());
124         }
125 
126         length += other.length;
127 
128         size = notes.size();
129     }
130 
131     /**
132      * Play the melody
133      */
134     public void play(){
135         Queue<Note> noteQueue = new LinkedList<Note>(notes); // Preserve main Queue
136 
```

```
137         play(noteQueue, repeat);
138     }
139 
140     /**
141      * Play the melody at a given moment
142      * @param time      The time to start playback the melody
143      */
144     public void play (double time){
145         Queue<Note> noteQueue = new LinkedList<Note>(notes); // Preserve main Queue
146 
147         while (time > 0){
148             time -= noteQueue.remove().getDuration();
149         }
150 
151         play(noteQueue, repeat);
152     }
153 
154     /**
155      * Play method which accept a queue of node and
156      * a number which indicate how many time to repeat a loop
157      * @param noteQueue     Queue of Node to Play
158      * @param repeat        How many time to repeat the loop
159      */
160     private void play (Queue<Note> noteQueue, int repeat){
161         while(!noteQueue.isEmpty()){ // Preserve User's Input
162             if (noteQueue.peek().isRepeat()) {
163                 Queue<Note> repeatQueue = new LinkedList<Note>();
164 
165                 do {        // Add First, then Check Empty and not Repeat
166                     repeatQueue.add(noteQueue.remove());
167                 } while ((!noteQueue.isEmpty()) &&
168                         (!noteQueue.peek().isRepeat()));
169 
170                 if (!noteQueue.isEmpty()) { // Preserve Musician's Input
171                     repeatQueue.add(noteQueue.remove());
172                 }
173 
174                 for (int i = 0; i <= repeat; ++i){
175                     for (int j = 0; j < repeatQueue.size(); ++j) {
176                         play(repeatQueue, true);
177                     }
178                 }
179             } else {
180                 play(noteQueue, false);
181             }
182         }
183     }
184 
185     /**
186      * Play method which rewind if noteQueue is a repeat section
187      * and play the first node of the noteQueue
188      * @param noteQueue     Queue to Play
189      * @param repeating     Indicate if it is repeating or not
190      */
191     private void play (Queue<Note> noteQueue, boolean repeating){
192         if (repeating) {
193             noteQueue.add(noteQueue.peek());        //Rewind, Size preserved
194         }
195 
196         noteQueue.remove().play();
197     }
198 }
```