5-6-2004

Melodia: An Online Web-based Program for Teaching Melodic and Rhythmic Dictation

Aaron Tauchen

Melodia: An Online Web-based Program for Teaching Melodic and Rhythmic Dictation

Aaron Tauchen

Honors Senior Project
May 6, 2004

Honors Senior Project
Approval

Form 3 – Submit with completed thesis. All signatures must be obtained.

Name of candidate: Aaron Tauchen

Department: Music

Degree: B.A. Music / B.S. Computer Science

Full title of project: Melodia: An Online Web-based Program for Teaching Melodic and Rhythmic Dictation

---

Approved by:

Project Advisor	Date 5/6/04
Department Chair	Date 5/6/04
Honors Program Director for Honors Council	Date 5/11/04

Abstract

Computer Assisted Instruction (CAI) has been used to teach various musical skills since nearly the advent of computers in the late 1960s. As of recently, there is trend for Internet based CAI programs that can run within a web browser. These programs provide very specialized instruction conveniently available through the Internet. While there are several examples of such programs, no program exists which gives a comprehensive approach to music theory instruction through a drill-and-practice type curriculum.

The objective of this project is to help bridge that gap by developing an on-line, browser-based, drill-and-practice program to assist in the teaching of melodic and rhythmic dictation skills. Because the intended audience for such a program will be music students at either the advanced high school level or at the college level, this program must be user-friendly and easy to use. The user will interact with the program through a GUI implemented using Java Applet technology. The program will quiz the users by playing a MIDI file through the users browser. Then, the user will attempt to "write" down the what they heard in music notation through the use of an interactive Graphical User Interface (GUI). The GUI interface will be implemented using Java Applet technology

Introduction

On of the skills required by formal music instruction involves ear-training. Ear-training is basically the memorization of certain audible characteristics to be able to distinguish intervals, rhythms, and harmonies audibly. For the average student, much practice is needed to be able to memorize and distinguish between different intervals and harmonies. Traditionally, this practice involves two people. One person plays the ear-training exercise on the piano while the student, without looking, guesses what the was played. This sort of practice can be tedious and time consuming for both people involved.

To aid in the practice of ear-training exercises, computers have been used as early as the 1960s to assist in such practices. Early computers programs were crude and only available on big mainframe computers and only at large universities that had access to such computers. As computers have become cheaper, smaller, and more advanced, more and more people have had access to computer assisted instruction techniques to teach ear-training. In recent years, there is a trend for Internet based, online computer assisted instruction to practice ear-training skills. While there are very successful examples of these programs, there currently lacks a program that could provide a comprehensive, drill-and-practice type curriculum to assist in the instruction of ear-training.

The objective of Melodia software is to provided a music theory student to practice ear training exercises in a challenging, yet convenient manner. The aim of such software is not to replace in-class instruction or instruction provided by a human instructor, but rather provide an easy, convenient way for the student to receive additional practice. To accomplish these goals, Melodia provides a user-friendly, easily-accessible, extensive computer based solution for developed of ear training skills.

Melodia – An Overview

This section presents an overview of the development and functionality of Melodia.

Java Applet Technology

Melodia is programed completely in Java using Java Applet Technology. Applets are Java's solution to developing robust programs that run completely within a web browser. These programs can run in any browser on any computer operating system as long as the a version of the Java Browser Plug-In is installed. Melodia requires that version 1.4.1 of the Java Run Time is installed on the host computer. The run time is provided as a free download from Java's website.

Java was chosen as the programming language for several reasons. First, Melodia had to run within a web browser. This rules out the user of any compiled language such as Visual Basic or C++. Next, Melodia required the ability to open and play MIDI files from the server. Java

provides an ample API for reading and playing MIDI files. The determining factor was Java's graphics ability. Melodia required vector graphics to be drawn and updated within the program. Currently, Java is the only choice for drawing and updating vector graphics within a web browser.

User Interface

When a users first opens Melodia in a browser, the user sees the main program screen (see Figure 1). This screen allows the user to select and play a quiz, or exit the program. To select a quiz, the user selects the quiz name from a drop-down combo box. A brief description of the quiz is displayed in a text box on the right of the screen. Currently, there is only one fully functional quiz, but the process to update quizzes is fairly simple. This will be discussed in more detail later.

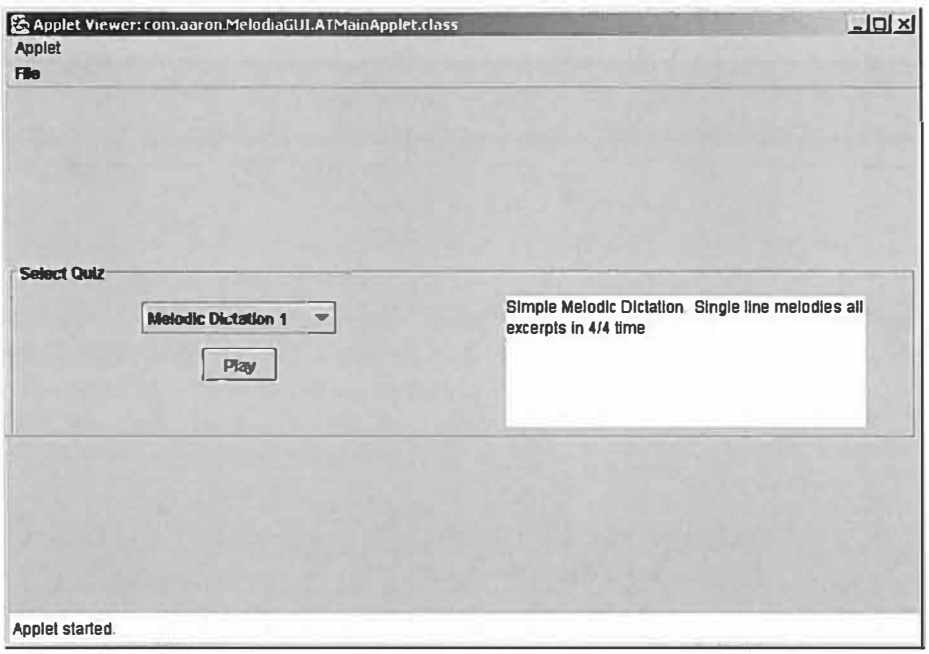

Figure 1. Screen shot of Melodia's main screen.

Upon selecting a quiz, a quiz dialog opens in a separate dialog. This dialog can be resized and dragged around the screen depending on the users preference. There are five main functional areas on the quiz dialog. In the upper right corner, Melodia shows the users basic statistics about the game such as the player name, question number, and score. In the lower right corner, Melodia displays the controls to navigate between questions in the quiz. This area contains the 'Next,' 'Previous,' and 'Done' buttons. Making use of most of the screen's center is the main workspace for entering melodies while taking the quizzes. Initially, the user is presented with a blank staff on which he or she will enter the notes. To enter notes, the users selects the rhythm of the note from a group of buttons directly below the staff and

simply clicks on the staff. The program records where the user clicks on the staff and then adds that note. The pitch of the note is determined by finding the closest line or space on the staff from which the user clicked. The horizontal location of the note is determined by where the user clicks relative to other notes. If a user wants to add a new note to the end of a measure, the users simply clicks between the last note of the measure and the final measure bar line. If the user wants to insert a new note between two existing notes, the users can click in between two of the existing notes and the note will be inserted, causing all other notes to be shifted over. The final functional area of the quiz dialog is the play back controls. These controls allow the user to play, pause, or stop and midi quiz file.

Once the user has heard the MIDI file for this particular question and entered the notes he or she thinks is correct, the user clicks on the 'Check Answer' button. This button will compare the melody that the user has entered with the melody that was played back in the MIDI file and determines whether the user entered the correct answer. It compares the answers by comparing note by note, rhythm by rhythm. If any pitch is incorrect or any note has the wrong rhythm, the program will score the question as an incorrect answer. If all the rhythms and pitches match the quiz MIDI file exactly, the answer is scored as correct.

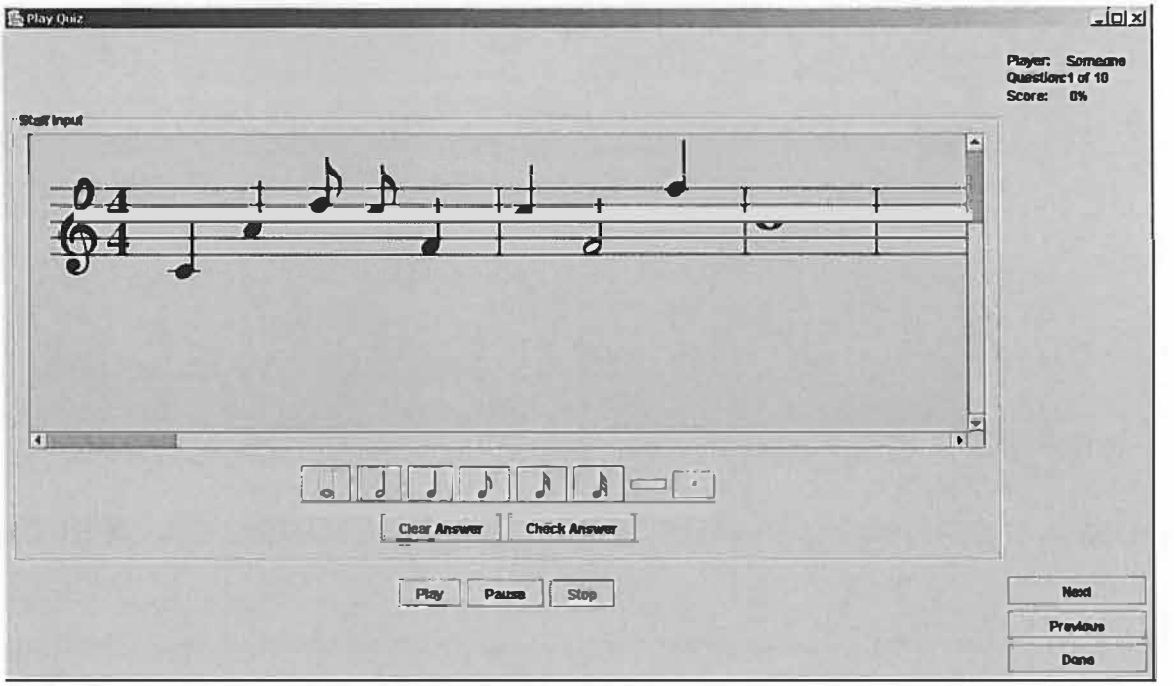

Figure 2. Screen shot of Melodia's quiz screen.

Quizzes

A game usually consists of ten questions. These questions are usually drawn from a pool of about fifty possible questions and presented to the user randomly. This ensures that the user will not memorize specific melodies. Once a quiz is complete, the program shows the user the final score for the quiz and takes the user back to the main screen.

The quiz files are made up of only MIDI files. This allows to create new quizzes and new questions to a quiz quite easily. For example, if an instructor wanted a customized module that focused only on dotted quarter note rhythms, he or she could simple create a number of MIDI files that focused on these rhythms. Then all the instructor would need to do is copy these MIDI files into Melodia's quiz file directory, tell Melodia to create a new quiz, and, presto, it is done.

Conclusion

Currently, Melodia is available on the University of Alabama in Huntsville's Music Technology website. In its current state, a user can play one melodic dictation game and receive wrong or right answers for each question. Certainly, many more future upgrades are need to make Melodia a viable instructional tool for use in a music theory class. Among these include improving the user interface, creating more quizzes, and more user testing of the program. Although Melodia is not in a finished product state, it can be used by music theory students for additional practice in the ear-training curriculum. Ear-training is not a skill that comes natural to music students, but must be learned over long hours of rigorous practice. Melodia is that tool which makes these hours of practice more efficient and more fun.

Note: As of May 2004, the current version of Melodia is available at http://www.uah.edu/music/aaron/

Appendix I

Appendix I contains a complete listing of the Java source code for Melodia. The code is organized into 4 packages and 23 separate files. The file names is shown in the header information of the print-out and the package name is the first line of code in each file.

```
 1 /*
 2  * Created on Jan 23, 2004
 3  *
 4  * To change the template for this generated file go to
 5  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 6  */
 7 package com.aaron.MelodiaGUI;
 8
 9 import com.aaron.StaffGraphic.*;
10 import com.aaron.MidiParser.*;
11 import javax.swing.JFrame;
12 import javax.swing.border.EtchedBorder;
13 import java.awt.geom.AffineTransform;
14
15 import javax.swing.*;
16 import java.awt.*;
17 import java.awt.event.*;
18 import java.util.*;
19 import com.aaron.MidiReader.*;
20 import javax.sound.midi.*;
21 import java.net.URL;
22
23 /**
24  * This is the main frame for doing all the quizes.  It displays the
25  * main quiz controls, play back controls, and the score for the current
26  * quiz
27  *
28  * @author Aaron Tauchen
29  */
30 public class ATQuizFrame extends JFrame {
31
32         /** link to the parent applet */
33         protected JApplet parent;
34
35         /** The MIDI files from which to choose from for the quiz */
36         protected Vector midiFiles;
37         /** The sequence of the current midifile being used for this question */
38         protected Sequence curMidiFile;
39         /** A midi sequence to play the midi files */
40         protected Sequencer sequencer;
41         /** The current note length to be adding when you click on the scrollpane*/
42         protected int lastNoteButtonPressed = 2;
43         /** The current question number */
44         protected int currentQuizNum = 0;
45         /** The index of the midifile in {@link #midiFiles midiFiles}*/
46         protected int quizFileNum = 0;
47         /**  */
48         protected int lastQuizNum = -1;
49         /**  */
50         protected int numofQuestion = 0;
51         protected int numofCorrect = 0;
52
53         protected ATStaff ansStaff;
54         protected ATStaff userStaff;
55
56         protected ATDrawStaff ansDrawStaff;
57         protected ATDrawStaff userDrawStaff;
58
59         // GUI Controls
60         protected JLabel lblPlayer;
61         protected JLabel lblQuestion;
62         protected JLabel lblScore;
63
64         protected JScrollPane scrScrollStaff;
65         protected DrawingPane panStaff;
66
67         protected JButton cmdPlay;
68         protected JButton cmdPause;
69         protected JButton cmdStop;
70
71         protected JButton cmdNext;
```

```
72          protected JButton cmdPrevious;
73          protected JButton cmdDone;
74
75          protected JButton cmdCheckAnswer;
76          protected JButton cmdClearAnswer;
77
78          protected JButton[] cmdAddNotes;
79          protected JToggleButton cmdDotted;
80
81          public class DrawingPane extends JPanel {
82                  protected void paintComponent(Graphics g) {
83                          super.paintComponent(g);
84                          //Graphics2D g2 = (Graphics2D) scrScrollStaff.getGraphics();
85                          Graphics2D g2 = (Graphics2D) g;
86                          g2.addRenderingHints(
87                                  new RenderingHints(RenderingHints.KEY_ANTIALIASING, RenderingHints.VAL
88                          g2.transform(new AffineTransform(1, 0, 0, -1, 0, 0));
89
90                          try {
91                                  g2 = ansDrawStaff.draw(g2);
92                                  g2 = userDrawStaff.draw(g2);
93                          }
94                          catch (NullPointerException e) {
95                          }
96
97                  }
98          }
99
100         public ATQuizFrame(JApplet parent) {
101                 super("Play Quiz");
102                 this.parent = parent;
103
104                 midiFiles = new Vector();
105
106                 addControls();
107                 addEvents();
108                 sequencer = getDefaultSequencer();
109
110                 //double startx = 20, starty = -50;
111                 userDrawStaff = new ATDrawStaff(20, -50);
112                 userDrawStaff.scale(1.5, 1.5);
113                 userDrawStaff.isVisible(true);
114
115                 ansDrawStaff = new ATDrawStaff(20, -200);
116                 ansDrawStaff.scale(1.5, 1.5);
117                 ansDrawStaff.isVisible(false);
118
119         }
120
121         public void setNumberofQuestions(int num) {
122                 numofQuestion = num;
123         }
124
125         private void addEvents() {
126
127                 cmdAddNotes[0].addActionListener(new ActionListener() {
128                         public void actionPerformed(ActionEvent evt) {
129                                 cmdAddNotes_Clicked(evt, 0);
130                         }
131                 });
132                 cmdAddNotes[1].addActionListener(new ActionListener() {
133                         public void actionPerformed(ActionEvent evt) {
134                                 cmdAddNotes_Clicked(evt, 1);
135                         }
136                 });
137                 cmdAddNotes[2].addActionListener(new ActionListener() {
138                         public void actionPerformed(ActionEvent evt) {
139                                 cmdAddNotes_Clicked(evt, 2);
140                         }
141                 });
142                 cmdAddNotes[3].addActionListener(new ActionListener() {
```

```
143                     public void actionPerformed(ActionEvent evt) {
144                             cmdAddNotes_Clicked(evt, 3);
145                     }
146             });
147             cmdAddNotes[4].addActionListener(new ActionListener() {
148                     public void actionPerformed(ActionEvent evt) {
149                             cmdAddNotes_Clicked(evt, 4);
150                     }
151             });
152             cmdAddNotes[5].addActionListener(new ActionListener() {
153                     public void actionPerformed(ActionEvent evt) {
154                             cmdAddNotes_Clicked(evt, 5);
155                     }
156             });
157
158             cmdDone.addActionListener(new ActionListener() {
159                     public void actionPerformed(ActionEvent evt) {
160                             cmdDone_Clicked(evt);
161                     }
162             });
163             scrScrollStaff.addMouseListener(new MouseAdapter() {
164                     public void mouseClicked(MouseEvent e) {
165                             scrScrollStaff_MouseClicked(e);
166                     }
167             });
168             cmdPlay.addActionListener(new ActionListener() {
169                     public void actionPerformed(ActionEvent evt) {
170                             cmdPlay_Clicked(evt);
171                     }
172             });
173             cmdPause.addActionListener(new ActionListener() {
174                     public void actionPerformed(ActionEvent evt) {
175                             cmdPause_Clicked(evt);
176                     }
177             });
178             cmdStop.addActionListener(new ActionListener() {
179                     public void actionPerformed(ActionEvent evt) {
180                             cmdStop_Clicked(evt);
181                     }
182             });
183             cmdNext.addActionListener(new ActionListener() {
184                     public void actionPerformed(ActionEvent evt) {
185                             cmdNext_Clicked(evt);
186                     }
187             });
188             cmdPrevious.addActionListener(new ActionListener() {
189                     public void actionPerformed(ActionEvent evt) {
190                             cmdPrevious_Clicked(evt);
191                     }
192             });
193
194             cmdCheckAnswer.addActionListener(new ActionListener() {
195                     public void actionPerformed(ActionEvent evt) {
196                             cmdCheckAnswer_Clicked(evt);
197                     }
198             });
199             cmdClearAnswer.addActionListener(new ActionListener() {
200                     public void actionPerformed(ActionEvent evt) {
201                             cmdClearAnswer_Clicked(evt);
202                     }
203             });
204
205     }
206
207     private void addControls() {
208             //addMenus();
209
210             //              Create a panel and add components to it.
211             JPanel mainPane = new JPanel(new GridBagLayout());
212             JPanel URPane = new JPanel(new GridLayout(3, 2));
213
```

```
214                 GridBagConstraints c = new GridBagConstraints();
215                 c.fill = GridBagConstraints.BOTH;
216
217                 // Blank Spacers
218                 mainPane.add(new JLabel());
219                 mainPane.add(new JLabel());
220                 mainPane.add(new JLabel());
221
222                 // Labels for players score
223                 JLabel label1 = new JLabel("Player:");
224                 URPane.add(label1);
225
226                 lblPlayer = new JLabel("Someone");
227                 URPane.add(lblPlayer);
228
229                 JLabel label3 = new JLabel("Question:");
230                 URPane.add(label3);
231
232                 lblQuestion = new JLabel("");
233                 URPane.add(lblQuestion);
234
235                 JLabel label5 = new JLabel("Score:");
236                 URPane.add(label5);
237
238                 lblScore = new JLabel("100%");
239                 URPane.add(lblScore);
240                 c.gridwidth = GridBagConstraints.REMAINDER;
241                 c.insets = new Insets(20, 5, 5, 20);
242                 mainPane.add(URPane, c);
243
244                 // Scroll Pane for staff view
245                 panStaff = new DrawingPane();
246                 panStaff.setPreferredSize(new Dimension(6000, 1000));
247                 panStaff.setBackground(Color.CYAN);
248
249                 JPanel panStaffInput = new JPanel(new BorderLayout(5, 10));
250                 scrScrollStaff =
251                         new JScrollPane(panStaff, JScrollPane.VERTICAL_SCROLLBAR_ALWAYS, JScrollPane.H
252                 scrScrollStaff.setPreferredSize(new Dimension(50, 600));
253                 //scrScrollStaff.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
254
255                 JLabel space = new JLabel();
256                 space.setPreferredSize(new Dimension(5, 0));
257                 panStaffInput.add(space, BorderLayout.EAST);
258                 JLabel space2 = new JLabel();
259                 space2.setPreferredSize(new Dimension(5, 0));
260                 panStaffInput.add(space2, BorderLayout.WEST);
261
262                 panStaffInput.add(scrScrollStaff, BorderLayout.CENTER);
263                 panStaffInput.setBorder(
264                         BorderFactory.createTitledBorder(BorderFactory.createEtchedBorder(EtchedBorder
265
266                 //Add add Note buttons
267                 cmdAddNotes = new JButton[7];
268                 JPanel panAddNotes = new JPanel(new FlowLayout());
269                 for (int i = 0; i < cmdAddNotes.length; i++) {
270                         //cmdAddNotes[i] = new JButton(String.valueOf(i));
271                         cmdAddNotes[i] = new JButton();
272                         panAddNotes.add(cmdAddNotes[i]);
273                 }
274
275                 cmdAddNotes[0].setIcon(new ImageIcon(parent.getImage(parent.getCodeBase(), "pics/1note
276                 cmdAddNotes[1].setIcon(new ImageIcon(parent.getImage(parent.getCodeBase(), "pics/2note
277                 cmdAddNotes[2].setIcon(new ImageIcon(parent.getImage(parent.getCodeBase(), "pics/4note
278                 cmdAddNotes[3].setIcon(new ImageIcon(parent.getImage(parent.getCodeBase(), "pics/8note
279                 cmdAddNotes[4].setIcon(new ImageIcon(parent.getImage(parent.getCodeBase(), "pics/16not
280                 cmdAddNotes[5].setIcon(new ImageIcon(parent.getImage(parent.getCodeBase(), "pics/32not
281                 //cmdAddNotes[0].setIcon(new ImageIcon("C:\\Program Files\\eclipse\\workspace\\Melodia
282                 //cmdAddNotes[1].setIcon(new ImageIcon("C:\\Program Files\\eclipse\\workspace\\Melodia
283                 //cmdAddNotes[2].setIcon(new ImageIcon("C:\\Program Files\\eclipse\\workspace\\Melodia
284                 //cmdAddNotes[3].setIcon(new ImageIcon("C:\\Program Files\\eclipse\\workspace\\Melodia
```

```
285             //cmdAddNotes[4].setIcon(new ImageIcon("C:\\Program Files\\eclipse\\workspace\\Melodia
286             //cmdAddNotes[5].setIcon(new ImageIcon("C:\\Program Files\\eclipse\\workspace\\Melodia
287
288             cmdDotted = new JToggleButton("*");
289             panAddNotes.add(cmdDotted);
290
291             JPanel panExtraButtons = new JPanel(new FlowLayout());
292             cmdClearAnswer = new JButton("Clear Answer");
293             panExtraButtons.add(cmdClearAnswer);
294
295             cmdCheckAnswer = new JButton("Check Answer");
296             panExtraButtons.add(cmdCheckAnswer);
297
298             JPanel panBottom = new JPanel(new GridLayout(0, 1));
299
300             panBottom.add(panAddNotes);
301             panBottom.add(panExtraButtons);
302
303             panStaffInput.add(panBottom, BorderLayout.PAGE_END);
304
305             c.weightx = 1.0;
306             c.weighty = 1.0;
307             //c.gridheight = 1;
308             c.gridwidth = GridBagConstraints.RELATIVE;
309             c.insets = new Insets(0, 3, 10, 0);
310
311             mainPane.add(panStaffInput, c);
312
313             //c.gridwidth = GridBagConstraints.REMAINDER;
314             //c.gridwidth = GridBagConstraints.REMAINDER;
315             //mainPane.add(new JButton("NEW2"), c);
316
317             /*c.weightx = 0;
318             c.weighty = 0;
319             c.gridheight = 2;
320             c.gridwidth = 1;
321
322             c.gridwidth = GridBagConstraints.REMAINDER;
323             mainPane.add(new JLabel());
324             mainPane.add(new JLabel());
325             mainPane.add(new JLabel());
326             mainPane.add(new JLabel());
327             */
328
329             //c.gridwidth = 1;
330             //c.weighty = 1;
331
332             JPanel panPlayBack = new JPanel(new FlowLayout());
333
334             cmdPlay = new JButton("Play");
335             panPlayBack.add(cmdPlay);
336
337             cmdPause = new JButton("Pause");
338             panPlayBack.add(cmdPause);
339
340             cmdStop = new JButton("Stop");
341             panPlayBack.add(cmdStop);
342
343             //c.anchor = GridBagConstraints.LINE_START;
344
345             c.weightx = 0;
346             c.weighty = 0;
347             c.gridx = 0;
348             c.gridy = 2;
349             c.insets = new Insets(0, 0, 0, 0);
350             mainPane.add(panPlayBack, c);
351
352             JPanel panNavigation = new JPanel(new GridLayout(0, 1, 5, 5));
353
354             cmdNext = new JButton("Next");
355             panNavigation.add(cmdNext);
```

```
356
357                 cmdPrevious = new JButton("Previous");
358                 panNavigation.add(cmdPrevious);
359
360                 cmdDone = new JButton("Done");
361                 panNavigation.add(cmdDone);
362
363                 c.gridwidth = 1;
364                 c.gridx = 3;
365                 c.gridy = 2;
366                 c.insets = new Insets(3, 3, 3, 3);
367                 mainPane.add(panNavigation, c);
368
369                 //              Make it the content pane.
370                 mainPane.setOpaque(true);
371                 this.setContentPane(mainPane);
372
373         }
374
375         //Event Handlers
376         private void cmdDone_Clicked(ActionEvent evt) {
377                 this.setVisible(false);
378         }
379
380         private void cmdAddNotes_Clicked(ActionEvent evt, int index) {
381                 lastNoteButtonPressed = index;
382         }
383         private void scrScrollStaff_MouseClicked(MouseEvent e) {
384
385                 //Graphics2D g = (Graphics2D) scrScrollStaff.getGraphics();
386                 //g.addRenderingHints(new RenderingHints(RenderingHints.KEY_ANTIALIASING, RenderingHin
387                 //g.transform( new AffineTransform(1,0,0,-1,0,0));
388
389                 JScrollBar hsb = scrScrollStaff.getHorizontalScrollBar();
390                 JScrollBar vsb = scrScrollStaff.getVerticalScrollBar();
391
392                 //lblScore.setText(String.valueOf(hsb.getValue()) + " : " + String.valueOf(e.getX()));
393                 double x = e.getX() + hsb.getValue();
394                 double y = e.getY() + vsb.getValue();
395
396                 int pitch = userDrawStaff.getPitchFromY(y);
397                 ATBeatStruct bs = userDrawStaff.getBeatFromX(x);
398                 ATNote newNote = createNextNote(pitch, bs);
399                 int[] newBeat = getNextNoteBeat(bs, userStaff.getTimeSignature());
400
401                 if (bs.doIAdd == true) {
402                         userStaff.addNote(newBeat[0], newBeat[1], newBeat[2], newNote);
403                 }
404                 else {
405                         userStaff.insertNote(newBeat[0], newBeat[1], newBeat[2], newNote);
406                 }
407
408                 userDrawStaff.clear();
409                 userDrawStaff.add(userStaff);
410
411                 //s.add(d,75,20);
412                 //s.add(d, e.getX(), e.getY());
413
414                 //g.fill(MusicalSymbols.trebleCleff());
415
416                 //ansDrawStaff.add(ansStaff);
417
418                 //g = s.draw(g);
419
420                 panStaff.repaint();
421
422         }
423
424         protected void cmdPlay_Clicked(ActionEvent evt) {
425                 if (sequencer.getTickPosition() != 0 && lastQuizNum == quizFileNum) {
426                         sequencer.start();
```

```
427                 }
428                 else {
429
430                         Sequence s = openMidiFile((URL) midiFiles.get(quizFileNum));
431                         try {
432                                 lastQuizNum = quizFileNum;
433                                 sequencer.setSequence(s);
434                                 sequencer.start();
435                         }
436                         catch (Exception e) {
437                                 // Error
438                         }
439                 }
440
441         }
442         protected void cmdPause_Clicked(ActionEvent evt) {
443                 sequencer.stop();
444         }
445         protected void cmdStop_Clicked(ActionEvent evt) {
446                 sequencer.stop();
447                 sequencer.setTickPosition(0);
448
449         }
450         protected void cmdNext_Clicked(ActionEvent evt) {
451                 currentQuizNum = currentQuizNum < numofQuestion - 1 ? currentQuizNum + 1 : currentQuiz
452                 updateQuestion();
453         }
454         protected void cmdPrevious_Clicked(ActionEvent evt) {
455                 currentQuizNum = currentQuizNum == 0 ? 0 : currentQuizNum - 1;
456                 updateQuestion();
457         }
458
459         protected void cmdCheckAnswer_Clicked(ActionEvent evt) {
460                 if (ansStaff.equals(userStaff)) {
461                         JOptionPane.showMessageDialog(this, "Correct Answer");
462                         numofCorrect++;
463                 }
464                 else {
465                         JOptionPane.showMessageDialog(this, "Bad Answer");
466                 }
467
468                 ansDrawStaff.isVisible(true);
469                 panStaff.repaint();
470         }
471
472         protected void cmdClearAnswer_Clicked(ActionEvent evt) {
473                 userStaff = ansStaff.getFirstNote();
474                 userDrawStaff.clear();
475                 userDrawStaff.add(userStaff);
476                 panStaff.repaint();
477         }
478
479         public void setQuizMidiFiles(Vector mFiles) {
480                 midiFiles = mFiles;
481
482         }
483
484         private void updateQuestion() {
485                 // lblQuestion
486                 int num = currentQuizNum + 1;
487                 int outOf = numofQuestion;
488                 lblQuestion.setText(String.valueOf(num) + " of " + String.valueOf(outOf));
489                 int score = 0;
490                 try {
491                         score = (int) Math.round(((double)numofCorrect / num) * 100);
492                 } catch(Exception e) {
493                         score = 0;
494                 }
495                 lblScore.setText(" " + score + "%");
496
497                 // answerStaff
```

```
498                 int size = midiFiles.size();
499                 quizFileNum = (((int) Math.floor(Math.random() * 1000)) % size);
500                 ATMidiParser mp = new ATMidiParser((URL) midiFiles.get(quizFileNum));
501
502                 ansStaff = mp.getStaff();
503                 userStaff = ansStaff.getFirstNote();
504                 ansDrawStaff.clear();
505                 userDrawStaff.clear();
506                 ansDrawStaff.add(ansStaff);
507                 userDrawStaff.add(userStaff);
508                 ansDrawStaff.isVisible(false);
509                 panStaff.repaint();
510
511         }
512
513         /* (non-Javadoc)
514          * @see java.awt.Component#show()
515          */
516         public void show() {
517                 // TODO Auto-generated method stub
518                 updateQuestion();
519                 super.show();
520         }
521
522         private Sequence openMidiFile(URL fname) {
523                 try {
524                         //File myMidiFile = new File(fname);
525                         //                  Construct a Sequence object, and
526                         //                  load it into my sequencer.
527                         Sequence mySeq = MidiSystem.getSequence(fname);
528
529                         //sequencer.setSequence(mySeq);
530                         return mySeq;
531                 }
532                 catch (Exception e) {
533                         return null;
534                 }
535
536         }
537         private Sequencer getDefaultSequencer() {
538                 Sequencer sequencer;
539                 // Get default sequencer.
540                 try {
541                         sequencer = MidiSystem.getSequencer();
542                         if (sequencer == null) {
543                                 // Error -- sequencer device is not supported.
544                                 // Inform user and return...
545                                 return null;
546                         }
547                         else {
548                                 // Acquire resources and make operational.
549                                 sequencer.open();
550                                 return sequencer;
551                         }
552                 }
553                 catch (Exception e) {
554                         // Error -- send message
555                         return null;
556                 }
557         }
558
559         private ATNote createNextNote(int pitch, ATBeatStruct bs) {
560                 int addNoteLen = 0;
561                 switch (lastNoteButtonPressed) {
562                         case 0 :
563                                 addNoteLen = ATDrawConstant.WHOLE;
564                                 break;
565                         case 1 :
566                                 addNoteLen = ATDrawConstant.HALF;
567                                 break;
568                         case 2 :
```

```
569                             addNoteLen = ATDrawConstant.QUARTER;
570                             break;
571                     case 3 :
572                             addNoteLen = ATDrawConstant.EIGHTH;
573                             break;
574                     case 4 :
575                             addNoteLen = ATDrawConstant.SIXTEENTH;
576                             break;
577                     case 5 :
578                             addNoteLen = ATDrawConstant.THIRTYSECOND;
579                             break;
580                     case 6 :
581                             addNoteLen = ATDrawConstant.SIXTYFORTH;
582                             break;
583             }
584             int dotted = 0;
585             if (cmdDotted.isSelected()) {
586                     dotted = 1;
587             }
588             return new ATNote(addNoteLen, ATNote.TYPE_NOTE, pitch, dotted);
589
590     }
591     /**
592      *
593      * @param bs
594      * @return [0] -- int Measure, [1] -- int MajorBeat, [2] -- int MinorBeat
595      */
596     private int[] getNextNoteBeat(ATBeatStruct bs, ATTimeSignature ts) {
597             int measure = bs.measure;
598             int majorBeat = bs.majorBeat;
599             int minorBeat = bs.minorBeat;
600             int length = bs.length;
601
602             if (bs.majorBeat == 0 && bs.minorBeat == 0) {
603                     majorBeat = 1;
604                     minorBeat = 0;
605                     int[] retval = new int[3];
606                     retval[0] = measure;
607                     retval[1] = majorBeat;
608                     retval[2] = minorBeat;
609                     return retval;
610
611             }
612
613             int accountedFor = 0;
614             int denomLength = ts.getDenominator().getLength();
615             while (length >= denomLength) {
616                     majorBeat++;
617                     length -= denomLength;
618                     accountedFor += denomLength;
619             }
620             while (majorBeat > ts.getNumerator()) {
621                     measure++;
622                     majorBeat -= ts.getNumerator();
623             }
624
625             minorBeat += (int) ((((double) length / (double) denomLength) - (int) Math.floor(lengt
626
627             while (minorBeat >= 240) {
628                     majorBeat++;
629                     minorBeat -= 240;
630             }
631
632             while (majorBeat > ts.getNumerator()) {
633                     measure++;
634                     majorBeat -= ts.getNumerator();
635             }
636
637             int[] retval = new int[3];
638             retval[0] = measure;
639             retval[1] = majorBeat;
```

```
640                 retval[2] = minorBeat;
641                 return retval;
642
643         }
644
645 }
```

```
 1 /*
 2  * Created on Jan 23, 2004
 3  *
 4  * To change the template for this generated file go to
 5  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 6  */
 7 package com.aaron.MelodiaGUI;
 8
 9 import javax.swing.JApplet;
10 import javax.swing.border.EtchedBorder;
11 import javax.swing.*;
12 import java.awt.*;
13 import java.awt.event.*;
14 import java.util.*;
15 import java.net.URL;
16
17 /**
18  * This is the Main Applet class.
19  *
20  * @author Aaron Tauchen
21  */
22 public class ATMainApplet extends JApplet {
23
24         /** Control */
25         protected JPanel panSelectQuiz;
26         /** Control */
27         protected JButton cmdPlayQuiz;
28         /** Control */
29         protected JComboBox cboSelectQuiz;
30         /** Control */
31         protected JTextArea txtQuizDesc;
32
33         protected String[] quizNames = {
34          "Melodic Dictation 1",
35          "Rhythmic Dictation 1"
36     };
37
38     protected String[] quizDescriptions = {
39          "Simple Melodic Dictation.  Single line melodies all excerpts in 4/4 time",
40          "Simple Rhythmic Dications.  Under construction"
41     };
42
43
44
45
46         public void init() {
47                 addMenus();
48                 addControls();
49                 addEvents();
50         }
51
52         private void addControls() {
53                 JPanel mainPane = new JPanel(new GridBagLayout());
54                 GridBagConstraints c = new GridBagConstraints();
55                 c.fill = GridBagConstraints.HORIZONTAL;
56
57                 JPanel panSelectQuiz = new JPanel(new GridLayout(0, 2));
58                 //JPanel panLeftSelect = new JPanel(new GridLayout(0,1));
59                 JPanel panLeftSelect = new JPanel();
60                 JPanel panRightSelect = new JPanel(new FlowLayout());
61
62                 panLeftSelect.setLayout(new BoxLayout(panLeftSelect, BoxLayout.PAGE_AXIS));
63                 cboSelectQuiz = new JComboBox(quizNames);
64                 cboSelectQuiz.setAlignmentX(Component.CENTER_ALIGNMENT);
65                 cboSelectQuiz.setMaximumSize(new Dimension(150, 25));
66
67                 cmdPlayQuiz = new JButton("Play");
68                 cmdPlayQuiz.setAlignmentX(Component.CENTER_ALIGNMENT);
69
70                 panLeftSelect.setLayout(new BoxLayout(panLeftSelect, BoxLayout.PAGE_AXIS));
71                 panLeftSelect.add(Box.createRigidArea(new Dimension(0, 10)));
```

```
72                panLeftSelect.add(cboSelectQuiz);
73                panLeftSelect.add(Box.createRigidArea(new Dimension(0, 10)));
74                panLeftSelect.add(cmdPlayQuiz);
75
76                txtQuizDesc = new JTextArea("");
77                txtQuizDesc.setPreferredSize(new Dimension(275, 100));
78                txtQuizDesc.setEditable(false);
79                txtQuizDesc.setLineWrap(true);
80                txtQuizDesc.setWrapStyleWord(true);
81                txtQuizDesc.append(quizDescriptions[0]);
82                panRightSelect.add(txtQuizDesc);
83
84                panLeftSelect.setAlignmentX(Component.CENTER_ALIGNMENT);
85                panRightSelect.setAlignmentY(Component.CENTER_ALIGNMENT);
86                panSelectQuiz.add(panLeftSelect);
87                panSelectQuiz.add(panRightSelect);
88                panSelectQuiz.setBorder(
89                        BorderFactory.createTitledBorder(BorderFactory.createEtchedBorder(EtchedBorder
90
91                c.weightx = 1.0;
92                mainPane.add(panSelectQuiz, c);
93
94                //              Make it the content pane.
95                mainPane.setOpaque(true);
96                this.setContentPane(mainPane);
97
98        }
99
100       private void addEvents() {
101               cmdPlayQuiz.addActionListener(new ActionListener() {
102                       public void actionPerformed(ActionEvent evt) {
103                               cmdPlayQuiz_Clicked(evt);
104                       }
105               });
106
107               cboSelectQuiz.addItemListener(new ItemListener() {
108                       public void itemStateChanged(ItemEvent e) {
109                               cboSelectQuiz_ItemChange(e);
110                       }
111               });
112
113       }
114
115       private void addMenus() {
116               JMenuBar menuBar = new JMenuBar();
117               this.setJMenuBar(menuBar);
118
119               JMenu fileMenu = new JMenu("File");
120               menuBar.add(fileMenu);
121
122               JMenuItem fileItem = new JMenuItem("Exit");
123               fileMenu.add(fileItem);
124
125       }
126
127
128       /**
129        * Event handler for button cmdPlayQuiz
130        * @param evt The event
131        */
132       private void cmdPlayQuiz_Clicked(ActionEvent evt) {
133               ATQuizFrame f = new ATQuizFrame(this);
134               f.setSize(1050, 700);
135               f.setQuizMidiFiles(getQuizMidiFiles(1));
136               f.setNumberofQuestions(10);
137               f.show();
138
139       }
140
141       private void cboSelectQuiz_ItemChange(ItemEvent e) {
142               if(e.getStateChange() == java.awt.event.ItemEvent.SELECTED) {
```

```
143                     String s = (String) e.getItem();
144                     for(int i = 0; i < quizNames.length; i++) {
145                             if(s.equals(quizNames[i])) {
146                                     txtQuizDesc.setText(quizDescriptions[i]);
147                             }
148                     }
149
150
151                     }
152             }
153
154         /**
155          * Load a vector with the quiz files for the given gameID
156          * @param gameID ID of the game which to get the quiz files for
157          * @return A vector of {@link java.net.URL URL} objects that contain file name for the MIDI fi
158          */
159         private Vector getQuizMidiFiles(int gameID) {
160                 Vector files = new Vector();
161
162                 try {
163                         /*      files.add(new URL("./midifiles/test1.mid"));
164                                 files.add(new URL("./midifiles/test2.mid"));
165                                 files.add(new URL("./midifiles/test3.mid"));
166                                 files.add(new URL("./midifiles/test4.mid"));
167                                 files.add(new URL("./midifiles/test5.mid"));
168                                 files.add(new URL("./midifiles/test6.mid"));
169                                 files.add(new URL("./midifiles/test7.mid"));
170                                 files.add(new URL("./midifiles/test8.mid"));
171                                 files.add(new URL("./midifiles/test9.mid"));
172                                 files.add(new URL("./midifiles/test10.mid"));
173                                 files.add(new URL("./midifiles/test11.mid"));
174                                 files.add(new URL("./midifiles/test12.mid"));
175                                 files.add(new URL("./midifiles/test13.mid"));
176                                 files.add(new URL("./midifiles/test14.mid"));
177                                 files.add(new URL("./midifiles/test15.mid"));
178                                 files.add(new URL("./midifiles/test16.mid"));
179                                 files.add(new URL("./midifiles/test17.mid"));
180                                 files.add(new URL("./midifiles/test18.mid"));
181                                 files.add(new URL("./midifiles/test19.mid"));
182                                 files.add(new URL("./midifiles/test20.mid"));
183                         */
184
185                         files.add(new URL(getCodeBase() + "midifiles/test1.mid"));
186                         files.add(new URL(getCodeBase() + "midifiles/test2.mid"));
187                         files.add(new URL(getCodeBase() + "midifiles/test3.mid"));
188                         files.add(new URL(getCodeBase() + "midifiles/test4.mid"));
189                         files.add(new URL(getCodeBase() + "midifiles/test5.mid"));
190                         files.add(new URL(getCodeBase() + "midifiles/test6.mid"));
191                         files.add(new URL(getCodeBase() + "midifiles/test7.mid"));
192                         files.add(new URL(getCodeBase() + "midifiles/test8.mid"));
193                         files.add(new URL(getCodeBase() + "midifiles/test9.mid"));
194                         files.add(new URL(getCodeBase() + "midifiles/test10.mid"));
195                         files.add(new URL(getCodeBase() + "midifiles/test11.mid"));
196                         files.add(new URL(getCodeBase() + "midifiles/test12.mid"));
197                         files.add(new URL(getCodeBase() + "midifiles/test13.mid"));
198                         files.add(new URL(getCodeBase() + "midifiles/test14.mid"));
199                         files.add(new URL(getCodeBase() + "midifiles/test15.mid"));
200                         files.add(new URL(getCodeBase() + "midifiles/test16.mid"));
201                         files.add(new URL(getCodeBase() + "midifiles/test17.mid"));
202                         files.add(new URL(getCodeBase() + "midifiles/test18.mid"));
203                         files.add(new URL(getCodeBase() + "midifiles/test19.mid"));
204                         files.add(new URL(getCodeBase() + "midifiles/test20.mid"));
205
206                         /*
207                         files.add(new String("./midiFiles/test1.mid"));
208                         files.add(new String("./midiFiles/test2.mid"));
209                         files.add(new String("./midiFiles/test3.mid"));
210                         files.add(new String("./midiFiles/test4.mid"));
211                         files.add(new String("./midiFiles/test5.mid"));
212                         files.add(new String("./midiFiles/test6.mid"));
213                         files.add(new String("./midiFiles/test7.mid"));
```

```
214                         files.add(new String("./midiFiles/test8.mid"));
215                         files.add(new String("./midiFiles/test9.mid"));
216                         files.add(new String("./midiFiles/test10.mid"));
217                         files.add(new String("./midiFiles/test11.mid"));
218                         files.add(new String("./midiFiles/test12.mid"));
219                         files.add(new String("./midiFiles/test13.mid"));
220                         files.add(new String("./midiFiles/test14.mid"));
221                         files.add(new String("./midiFiles/test15.mid"));
222                         files.add(new String("./midiFiles/test16.mid"));
223                         files.add(new String("./midiFiles/test17.mid"));
224                         files.add(new String("./midiFiles/test18.mid"));
225                         files.add(new String("./midiFiles/test19.mid"));
226                         files.add(new String("./midiFiles/test20.mid"));
227                         */
228
229                         /*
230                         files.add(new String("./midiFiles/test1.mid"));
231                         files.add(new String("C:\\Program Files\\eclipse\\workspace\\MelodiaGUI\\midiF
232                         files.add(new String("C:\\Program Files\\eclipse\\workspace\\MelodiaGUI\\midiF
233                         files.add(new String("C:\\Program Files\\eclipse\\workspace\\MelodiaGUI\\midiF
234                         files.add(new String("C:\\Program Files\\eclipse\\workspace\\MelodiaGUI\\midiF
235                         files.add(new String("C:\\Program Files\\eclipse\\workspace\\MelodiaGUI\\midiF
236                         files.add(new String("C:\\Program Files\\eclipse\\workspace\\MelodiaGUI\\midiF
237                         files.add(new String("C:\\Program Files\\eclipse\\workspace\\MelodiaGUI\\midiF
238                         files.add(new String("C:\\Program Files\\eclipse\\workspace\\MelodiaGUI\\midiF
239                         files.add(new String("C:\\Program Files\\eclipse\\workspace\\MelodiaGUI\\midiF
240                         files.add(new String("C:\\Program Files\\eclipse\\workspace\\MelodiaGUI\\midiF
241                         files.add(new String("C:\\Program Files\\eclipse\\workspace\\MelodiaGUI\\midiF
242                         files.add(new String("C:\\Program Files\\eclipse\\workspace\\MelodiaGUI\\midiF
243                         files.add(new String("C:\\Program Files\\eclipse\\workspace\\MelodiaGUI\\midiF
244                         files.add(new String("C:\\Program Files\\eclipse\\workspace\\MelodiaGUI\\midiF
245                         files.add(new String("C:\\Program Files\\eclipse\\workspace\\MelodiaGUI\\midiF
246                         files.add(new String("C:\\Program Files\\eclipse\\workspace\\MelodiaGUI\\midiF
247                         files.add(new String("C:\\Program Files\\eclipse\\workspace\\MelodiaGUI\\midiF
248                         files.add(new String("C:\\Program Files\\eclipse\\workspace\\MelodiaGUI\\midiF
249                         files.add(new String("C:\\Program Files\\eclipse\\workspace\\MelodiaGUI\\midiF
250                         */
251                 }
252                 catch (Exception e) {
253                 }
254                 return files;
255
256         }
257
258 }
```

```
 1 /*
 2  * Created on Mar 3, 2004
 3  *
 4  * To change the template for this generated file go to
 5  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 6  */
 7 package com.aaron.MidiParser;
 8
 9 import com.aaron.MidiReader.*;
10 import javax.sound.midi.*;
11 import java.util.*;
12 import java.text.ParseException;
13 import java.net.URL;
14
15 /**
16  * @author SoundBooth
17  *
18  * To change the template for this generated type comment go to
19  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
20  */
21 public class ATMidiParser {
22         private static final int MINTICKS = 240; // Number of ticks in a beat
23
24         private ATStaff midiStaff;
25
26         private class midiNotes {
27                 public long length;
28                 public long time;
29                 public int pitch;
30         }
31
32         private class midiTimeSig {
33                 public int num;
34                 public int den;
35         }
36
37         private class MidiNoteOn {
38                 int velocity;
39                 int note;
40                 public MidiNoteOn(ShortMessage midiMessage) {
41                         velocity = midiMessage.getData2();
42                         note = midiMessage.getData1();
43                 }
44                 public int getVelocity() {
45                         return velocity;
46                 }
47                 public int getNote() {
48                         return note;
49                 }
50         }
51
52         private class MidiTimeSignature{
53                 int numerator;
54                 int denominator;
55                 public MidiTimeSignature(byte[] midiMessage) {
56                         numerator = midiMessage[3];
57                         denominator = midiMessage[4];
58                 }
59                 public int getNumerator() {
60                         return numerator;
61                 }
62                 public int getDenominator() {
63                         return denominator;
64                 }
65
66         }
67
68         public ATMidiParser(URL fileName) {
69                 midiStaff = Midi2Staff(fileName);
70         }
71
```

```
72          public ATStaff getStaff() {
73                  return midiStaff;
74          }
75
76
77          private ATStaff Midi2Staff(URL fileName) {
78                  String staffString = new String();
79
80
81                  //System.err.println("Filename: " + fileName.toString());
82                  Sequence seq = openMidiFile(fileName);
83
84
85                  //int format  -- No equivalent in java
86                  Track[] t_all = seq.getTracks();
87                  int numtracks = t_all.length;
88                  Vector notesinTrack = new Vector();
89                  notesinTrack.clear();
90
91                  for (int i = 0; i < t_all.length; i++) {
92                          long curTrackTime = 0;
93                          Track t = t_all[i];
94
95                          //System.err.println("Track: " + i);
96
97                          for (int j = 0; j < t.size(); j++) {
98                                  MidiEvent ev = t.get(j);
99                                  byte evbytes[] = ev.getMessage().getMessage();
100                                 //for(int bloop = 0; bloop < evbytes.length ; bloop++) {
101                                 //      System.err.print(evbytes[bloop]);
102                                 //}
103                                 //System.err.println(ev.getMessage().toString());
104
105                                 curTrackTime = ev.getTick();
106                                 // Get message number for event type
107                                 int evType = ev.getMessage().getStatus();
108
109                                 if (evType >= 0x90 && evType <= 0x9f) { //Case for NOTE-ON
110                                         MidiNoteOn no = new MidiNoteOn((ShortMessage) ev.getMessage())
111                                         if (no.getVelocity() != 0) {
112                                                 int k = j;
113                                                 long noteLength = 0;
114                                                 while (k < t.size() - 1) {
115                                                         k++;
116                                                         MidiEvent tempEv = t.get(k);
117                                                         if (tempEv.getMessage().getStatus() == evType)
118                                                                 MidiNoteOn nextOn = new MidiNoteOn((Sh
119                                                                 if (nextOn.getNote() == no.getNote())
120                                                                         noteLength = tempEv.getTick()
121                                                                         break;
122                                                                 }
123                                                         }
124                                                         if (tempEv.getMessage().getStatus() >= 0x80 &&
125                                                                 MidiNoteOn nextOn = new MidiNoteOn((Sh
126                                                                 if (nextOn.getNote() == no.getNote())
127                                                                         noteLength = tempEv.getTick()
128                                                                         break;
129                                                                 }
130                                                         }
131                                                 }
132                                                 midiNotes newNote = new midiNotes();
133                                                 newNote.pitch = no.getNote();
134                                                 newNote.time = ev.getTick();
135                                                 newNote.length = noteLength;
136                                                 notesinTrack.add(newNote);
137                                         }
138                                 }
139                                 if (evType == 0xFF) { // Case for extra messages
140                                         byte[] bmesg = ev.getMessage().getMessage();
141                                         if (bmesg[1] == 0x58) { //Case for Timesignature
142                                                 MidiTimeSignature ts = new MidiTimeSignature(ev.getMes
```

```
143                                     int fileNum = ts.getNumerator();
144                                     int fileDen = ts.getDenominator();
145                                     int num = fileNum;
146                                     int den = (int) Math.pow(2, 6 - fileDen);
147                                     midiTimeSig time = new midiTimeSig();
148                                     time.den = den;
149                                     time.num = num;
150
151                                     try {
152                                             Object tempO = notesinTrack.get(0);
153                                             if (!tempO.getClass().getName().equals("midiTi
154                                                     notesinTrack.insertElementAt(time, 0);
155                                             }
156                                     }
157                                     catch (ArrayIndexOutOfBoundsException e) {
158                                             notesinTrack.insertElementAt(time, 0);
159                                     }
160                             }
161
162                     }
163
164             }
165
166     }
167
168
169
170     String trackStr = new String();
171     ATTimeSignature timSig = new ATTimeSignature(4,new ATNoteDuration(ATNoteDuration.QUART
172
173     Enumeration enumNoteTrack = notesinTrack.elements();
174     while(enumNoteTrack.hasMoreElements()) {
175             trackStr = "";
176             Object obj = enumNoteTrack.nextElement();
177             String objType = obj.getClass().getName();
178             if(objType.equals("com.aaron.MidiParser.ATMidiParser$midiTimeSig")) {
179                     midiTimeSig mSig = (midiTimeSig) obj;
180                     ATNoteDuration nd = new ATNoteDuration(mSig.den);
181                     timSig = new ATTimeSignature(mSig.num, nd);
182                     trackStr = trackStr.concat("T=" + timSig.getNumerator() + ":" + timSig
183
184             }
185             if(objType.equals("com.aaron.MidiParser.ATMidiParser$midiNotes")) {
186                     midiNotes mNote = (midiNotes) obj;
187                     int measure = 1;
188                     int minorBeat = 0;
189                     int majorBeat = 1;
190                     int length;
191                     int dotted = 0;
192
193                     int beatUnit = seq.getResolution();
194                     double exactTime = mNote.time / beatUnit;
195                     majorBeat = (int) Math.floor(exactTime) + 1;
196                     while(majorBeat > timSig.getNumerator()) {
197                             measure++;
198                             majorBeat -= timSig.getNumerator();
199                     }
200
201                     //(int) ((((double)inNote.Length / (double) timeSig.Denominator) - (do
202                     minorBeat = (int) ((((double) mNote.time / (double) beatUnit) - (int)
203                     double test =  Math.pow(2, log2((double) mNote.length / beatUnit) + 3)
204                     int Notelength = (int) Math.round(Math.pow(2,log2((double) mNote.lengt
205                     // key=Measure:MajorBeat:MinorBeat|Type,Length,Pitch,Dotted,Tie,IsDupl
206
207                     // length -- Normalize whole note length , EX:1,2,4,8
208                     // diff   -- How much is not accounted for
209                     // accountedLen -- How much is accounted for
210                     //
211
212                     length = Math.min((int) Math.pow(2,(long)Math.floor(log2(Notelength) +
213                     // ! TODO - Handle case of double wholenotes
```

```
214                             //MessageBox.Show("Notelength = " + Notelength.ToString() + "\nlength
215                             int diff = Notelength - length;
216                             int accountedLen = length;
217                             int sumLength = length;
218                             //MessageBox.Show("diff = " + diff.ToString());
219                             while(diff >= 1 && diff >= sumLength / 2)
220                             {
221                                     dotted++;
222                                     diff = diff - (sumLength / 2);
223                                     sumLength /= 2;
224                                     accountedLen += sumLength;
225
226                             }
227                             //MessageBox.Show("After dotted adjust note loop");
228
229
230                             String tied = new String("null");
231                             if (diff > 0) tied = "T";
232
233                             trackStr += "N=" + measure + ":" + majorBeat + ":" + minorBeat+ "|Note
234                                     + (new ATNoteDuration(length)).toString() + "," + mNote.pitch
235
236                             // initialize so we know that this is the first time through this loop
237                             //int tieMajorBeat = -1;
238                             //int tieMinorBeat = -1;
239                             //int tieMeasure = -1;
240
241                             while(diff > 0)
242                             {
243                                     //accountedLen;
244
245                                     ///int tieMajorBeat = majorBeat + (int) Math.floor(accountedLe
246                                     int tieMeasure = measure;
247                                     ///while(tieMajorBeat   > timSig.getNumerator())
248                                     ///{
249                                     ///     tieMeasure++;
250                                     ///     tieMajorBeat -= timSig.getNumerator();
251                                     ///}
252                                     int tieMajorBeat = majorBeat;
253
254                                     ///int tieMinorBeat = (int) Math.round(minorBeat + ((accounted
255                                     int tieMinorBeat = (int) Math.round(((double) accountedLen / t
256                                     while(tieMinorBeat >= MINTICKS)
257                                     {
258                                             tieMajorBeat++;
259                                             tieMinorBeat -= MINTICKS;
260                                     }
261                                     int tieLength= Math.min((int) Math.pow(2,Math.floor(log2(diff)
262                                     diff = diff - tieLength;
263                                     accountedLen = accountedLen + tieLength;
264                                     tied = "null";
265                                     if (diff > 0) tied = "T";
266
267                                     trackStr += "N=" + tieMeasure + ":" + tieMajorBeat + ":" + tie
268                                             + (new ATNoteDuration(tieLength)).toString() + "," + m
269                             }
270                             //MessageBox.Show("After tie adjust note loop");
271
272
273
274
275                     }
276
277                     staffString = staffString.concat(trackStr);
278
279             }
280
281
282
283             ATStaff retStaff = new ATStaff();
284             try {
```

```
285                         retStaff.AddNotesbyString(staffString);
286                         return retStaff;
287                 } catch (ParseException e) {
288                         return retStaff;
289                 }
290
291
292
293         }
294
295         private double log2(double x) {
296                 return 1.4426950408889634 * Math.log(x);
297         }
298
299         private Sequence openMidiFile(URL fname) {
300                 try {
301                    //File myMidiFile = new File(fname);
302 //                 Construct a Sequence object, and
303 //                 load it into my sequencer.
304                         Sequence mySeq = MidiSystem.getSequence(fname);
305
306                 //sequencer.setSequence(mySeq);
307                         return mySeq;
308                 }
309                 catch (Exception e) {
310                         return null;
311                 }
312
313
314
315         }
316
317 }
318
319
320
321 /*
322  *
323  *
324  *               private struct midiNotes
325                 {
326                         public long length;
327                         public long time;
328                         public int pitch;
329                 }
330
331                 struct midiTimeSig
332                 {
333                         public int num;
334                         public int den;
335                 }
336
337
338 public static string Midi2SheetMusic(MidiSequence seq)
339 {
340         int     format = seq.Format;
341         int     tracks = seq.NumberOfTracks;
342         //int division = seq.Division;
343         ArrayList notesinTrack = new ArrayList();
344
345         MidiTrack [] t = seq.GetTracks();
346
347         string retStr = "";
348         foreach (MidiTrack r in t)
349         {
350                 notesinTrack.Clear();
351                 long curTrackTime =     0;
352                 MidiEventCollection     c =     r.Events;
353                 //nextNote.majorBeat = 0;
354                 //nextNote.minorBeat = 0;
355                 //nextNote.measure = 1;
```

```
356             //MessageBox.Show("Before event loop");
357
358             int     i =     0;      // loop for     each event in track
359             for(i = 0; i < c.Count; i++)
360             {
361                     MidiEvent ev = c[i];
362                     curTrackTime += c[i].DeltaTime;
363                     string evType = ev.GetType().ToString();
364                     switch(evType)
365                     {
366                             case "Toub.Sound.Midi.NoteOn":
367                                     NoteOn no =     (NoteOn) ev;
368                                     if (no.Velocity != 0)
369                                     {
370                                             int     j =     i;
371                                             long noteLength = 0;
372                                             while(j < c.Count)
373                                             {
374                                                     j++;
375                                                     noteLength += c[j].DeltaTime;
376                                                     if(c[j].GetType().ToString() == "Toub.Sound.Mi
377                                                     {
378                                                             NoteOn nextOn = (NoteOn) c[j];
379                                                             if(nextOn.Note == no.Note)
380                                                             {
381                                                                     break;
382                                                             }
383                                                     }
384                                                     if(c[j].GetType().ToString() == "Toub.Sound.Mi
385                                                     {
386                                                             NoteOff nextOn = (NoteOff) c[j];
387                                                             if(nextOn.Note == no.Note)
388                                                             {
389                                                                     break;
390                                                             }
391
392
393                                                     }
394                                             }
395                                             midiNotes newNote = new midiNotes();
396                                             newNote.pitch = no.Note;
397                                             newNote.time = curTrackTime;
398                                             newNote.length = noteLength;
399                                             notesinTrack.Add(newNote);
400
401
402                                     }
403                                     break;
404                             case "Toub.Sound.Midi.TimeSignature":
405                                     Toub.Sound.Midi.TimeSignature ts = (Toub.Sound.Midi.TimeSignat
406                                     int fileNum = ts.Numerator;
407                                     int fileDen = ts.Denominator;
408                                     int num = fileNum;
409                                     int den = (int) Math.Pow(2, 5 - fileDen);
410                                     midiTimeSig time = new midiTimeSig();
411                                     time.den = den;
412                                     time.num = num;
413                                     if(notesinTrack.Count < 1 || notesinTrack[0].GetType().ToStrin
414                                     {
415                                             notesinTrack.Insert(0,time);
416                                     }
417                                     break;
418
419
420
421                             default:
422                                     break;
423                     }
424             }
425
426             //MessageBox.Show("After event loop");
```

```
427
428
429                 string trackStr = "";
430                 Aaron.SheetMusic1.TimeSignature timSig = new Aaron.SheetMusic1.TimeSignature(4, NoteDu
431                 //MessageBox.Show("Before note loop");
432
433                 foreach(object Obj in notesinTrack)
434                 {
435                         string objType = Obj.GetType().ToString();
436                         switch( objType)
437                         {
438                                 case "MelodicDictationCtrl.MelodicDictationCtrl+midiTimeSig":
439                                    midiTimeSig mSig = (midiTimeSig) Obj;
440                                    NoteDurationEnum nd = new NoteDurationEnum();
441                                    nd = (NoteDurationEnum) mSig.den;
442
443                                    timSig = new Aaron.SheetMusic1.TimeSignature(mSig.num, nd);
444                                    trackStr += "T=" + timSig.Numerator + ":" + timSig.Denominator
445                                    break;
446                                 case "MelodicDictationCtrl.MelodicDictationCtrl+midiNotes":
447
448                                    midiNotes mNote = (midiNotes) Obj;
449                                    int     measure = 1;
450                                    int     minorBeat =     0;
451                                    int     majorBeat =     1;
452                                    int length;
453                                    int dotted = 0;
454
455                                    // !NEED TO GET TIME SIG FROM MIDI FILE
456                                    //Aaron.SheetMusic1.TimeSignature       timSig = new Aaron.She
457                                    int     beatUnit = seq.Division;
458                                    double exactTime = mNote.time / beatUnit;
459                                    majorBeat =     (int) Math.Floor(exactTime) + 1;
460                                    while(majorBeat > timSig.Numerator)
461                                    {
462                                            measure++;
463                                            majorBeat -= timSig.Numerator;
464                                    }
465                                    //MessageBox.Show("After majorBeat adjust note loop");
466
467                                    //(int) ((((double)inNote.Length / (double) timeSig.Denominato
468                                    minorBeat = (int) ((((double) mNote.time / (double) beatUnit)
469                                    double test =  Math.Pow(2,Math.Log((double) mNote.length / bea
470                                    int Notelength = (int) Math.Round(Math.Pow(2,Math.Log((double)
471                                    // key=Measure:MajorBeat:MinorBeat|Type,Length,Pitch,Dotted,Ti
472
473                                    // length -- Normalize whole note length , EX:1,2,4,8
474                                    // diff   -- How much is not accounted for
475                                    // accountedLen -- How much is accounted for
476                                    //
477
478                                    length = Math.Min((int) Math.Pow(2,Math.Round(Math.Log(Notelen
479                                    // ! TODO - Handle case of double wholenotes
480                                    //MessageBox.Show("Notelength = " + Notelength.ToString() + "\
481                                    int diff = Notelength - length;
482                                    int accountedLen = length;
483                                    int sumLength = length;
484                                    //MessageBox.Show("diff = " + diff.ToString());
485                                    while(diff >= 1 && diff >= sumLength / 2)
486                                    {
487                                            dotted++;
488                                            diff = diff - (sumLength / 2);
489                                            sumLength /= 2;
490                                            accountedLen += sumLength;
491
492                                    }
493                                    //MessageBox.Show("After dotted adjust note loop");
494
495
496                                    string tied = "null";
497                                    if (diff > 0) tied = "T";
```

```
498
499                                     trackStr += "N=" + measure + ":" + majorBeat + ":" + minorBeat
500                                             + (NoteDurationEnum) length + "," + mNote.pitch + ","
501
502                                     while(diff > 0)
503                                     {
504                                             //accountedLen;
505                                             int tieMajorBeat = majorBeat + (int) Math.Floor(accoun
506                                             int tieMeasure = measure;
507                                             while(tieMajorBeat      > timSig.Numerator)
508                                             {
509                                                     tieMeasure++;
510                                                     tieMajorBeat -= timSig.Numerator;
511                                             }
512                                             int tieMinorBeat =  minorBeat + (accountedLen - (int)
513                                             while(tieMinorBeat > 120)
514                                             {
515                                                     tieMajorBeat++;
516                                                     tieMinorBeat -= 120;
517                                             }
518                                             int tieLength= Math.Min((int) Math.Pow(2,Math.Floor(Ma
519                                             diff = diff - tieLength;
520                                             tied = "null";
521                                             if (diff > 0) tied = "T";
522
523                                             trackStr += "N=" + tieMeasure + ":" + tieMajorBeat + "
524                                                     + (NoteDurationEnum) tieLength + "," + mNote.p
525                                     }
526                                     //MessageBox.Show("After tie adjust note loop");
527                                     break;
528                             }
529
530
531
532
533
534                     }
535
536                     retStr += trackStr;
537
538
539
540
541
542             }
543             //MessageBox.Show("After note loop");
544
545             return retStr;
546 }
547
548
549 */
```

```
 1 /*
 2  * Created on Feb 12, 2004
 3  *
 4  * To change the template for this generated file go to
 5  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 6  */
 7 package com.aaron.MidiReader;
 8
 9 /**
10  * @author aaron
11  *
12  * To change the template for this generated type comment go to
13  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
14  */
15 public class ATTimeSignature {
16         protected int numerator;
17         protected ATNoteDuration denominator;
18
19         public ATTimeSignature(int num, ATNoteDuration denom) {
20                 numerator = num;
21                 denominator = denom;
22         }
23
24         public int getNumerator() { return numerator; }
25         public ATNoteDuration getDenominator() { return denominator; }
26
27         public boolean equals(Object o) {
28                 if (o == null
29                         || !o.getClass().getName().equals(this.getClass().getName()))
30                         return false;
31                 ATTimeSignature n = (ATTimeSignature) o;
32                 return (this.numerator == n.getNumerator()) && (this.denominator.equals(n.getDenominat
33         }
34
35         public int hashCode() {
36                 return numerator ^ denominator.getLength();
37
38         }
39
40 }
41
42
43 /*
44 public class TimeSignature
45 {
46         protected int numerator;
47         protected NoteDurationEnum denominator;
48         public TimeSignature(int numer, NoteDurationEnum denom)
49         {
50                 numerator = numer;
51                 denominator = denom;
52         }
53
54         public int Numerator
55         {
56                 get
57                 {
58                         return numerator;
59                 }
60         }
61         public NoteDurationEnum Denominator
62         {
63                 get
64                 {
65                         return denominator;
66                 }
67         }
68
69         public override bool Equals(Object obj)
70         {
71                 if (obj == null || this.GetType() != obj.GetType()) return false;
```

```
 72                 TimeSignature t = (TimeSignature) obj;
 73                 return (this.denominator == t.denominator) && (this.numerator == t.numerator);
 74         }
 75
 76         public static bool Equals(TimeSignature t1, TimeSignature t2)
 77         {
 78                 Object obj1, obj2;
 79                 obj1 = t1;
 80                 obj2 = t2;
 81                 if(obj1 == null && obj2 == null) return true;
 82                 if(obj1 == null) return false;
 83                 if(obj2 == null) return false;
 84                 return (t1.numerator == t2.numerator) && (t1.denominator == t2.denominator);
 85         }
 86
 87         public static bool operator==(TimeSignature t1, TimeSignature t2)
 88         {
 89                 return Equals(t1,t2);
 90         }
 91         public static bool operator!=(TimeSignature t1, TimeSignature t2)
 92         {
 93                 return !Equals(t1,t2);
 94         }
 95
 96         public override int GetHashCode()
 97         {
 98                 return numerator ^ (int) denominator;
 99         }
100
101
102 }
103 */
```

```
 1 /*
 2  * Created on Feb 18, 2004
 3  *
 4  * To change the template for this generated file go to
 5  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 6  */
 7 package com.aaron.MidiReader;
 8
 9 import java.util.*;
10 import java.text.ParseException;
11
12 /**
13  * @author aaron
14  *
15  * To change the template for this generated type comment go to
16  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
17  */
18 public class ATStaff {
19
20         private class NextNoteData {
21                 public NextNoteData(int majBeat, int minBeat, int meas)
22                 {
23                         majorBeat = majBeat;
24                         minorBeat = minBeat;
25                         measure = meas;
26                 }
27
28                 public int majorBeat;
29                 public int minorBeat;
30                 public int measure;
31         }
32
33         protected Vector measures;  //0 indexed
34         protected NextNoteData nextNote;
35         protected int keySig; // = null if time/key varies within staff
36         protected ATTimeSignature timeSig;   // set this only if time/key remain constant throughout
37
38
39         public ATStaff()
40         {
41                 //
42                 // TODO: Add constructor logic here
43                 //
44                 measures = new Vector();
45                 nextNote = new NextNoteData(1,1,1);
46                 keySig = ATKeySig.UnKnown;
47                 timeSig = null;
48         }
49         public ATStaff(int numOfMeasures)
50         {
51                 measures = new Vector();
52                 nextNote = new NextNoteData(1,1,1);
53
54                 for (int i = 0; i < numOfMeasures; i++)
55                 {
56                         measures.add(i, new ATMeasure(measures.size()));
57                 }
58         }
59
60         public void addMeasures(int num)
61         {
62                 for (int i = 0; i < num; i++)
63                 {
64                         measures.add(new ATMeasure(measures.size()));
65                 }
66         }
67
68         public void addMeasure(ATMeasure m)
69         {
70                 measures.add(m);
71                 m.setMeasureNumber(measures.size());
```

```
72          }
73
74          public ATMeasure getMeasure(int index)
75          {
76                  return (ATMeasure) measures.get(index + 1);
77          }
78
79          /**
80           *
81           * @return Iterator of ATMeasure objects
82           */
83          public Iterator iterator() {
84                  return measures.iterator();
85          }
86
87          public int getKeySignature() {
88                  return keySig;
89          }
90
91          public void setKeySignature(int key)
92          {
93                  Iterator i = measures.iterator();
94                  while(i.hasNext()) {
95                          ATMeasure m = (ATMeasure) i.next();
96                          m.setKeySignature(key);
97                  }
98          }
99
100         public ATTimeSignature getTimeSignature() {
101                 return timeSig;
102         }
103
104         public ATStaff getFirstNote() {
105                 boolean foundNote = false;
106
107                 ATStaff retval = new ATStaff();
108                 retval.setKeySignature(this.getKeySignature());
109                 retval.setTimeSignature(this.getTimeSignature());
110                 Iterator i = measures.iterator();
111                 while(!foundNote && i.hasNext()) {
112                         ATMeasure m = (ATMeasure) i.next();
113                         Iterator h = m.iterator();
114                         while(!foundNote && h.hasNext()) {
115                                 if(h.hasNext()) {
116                                    ATNoteBeat a = (ATNoteBeat) h.next();
117                                    retval.addNote(1,a.getMajorBeat(),a.getMinorBeat(),a.getNote());
118                                    foundNote = true;
119                                 }
120
121                         }
122
123                 }
124
125                 return retval;
126         }
127
128         public void setTimeSignature(ATTimeSignature time)
129         {
130                 Iterator i = measures.iterator();
131                 while(i.hasNext()) {
132                         ATMeasure m = (ATMeasure) i.next();
133                         m.setTimeSignature(time);
134                 }
135                 timeSig = time;
136         }
137
138         public void addNote(int measure, int majBeat, int minBeat, ATNote inNote)
139         {
140                 int i = measures.size();
141                 while (measures.size() <= measure)
142                 {
```

```
143                     measures.add(new ATMeasure(measures.size()));
144             }
145
146             ATMeasure m = (ATMeasure) measures.remove(measure);
147             m.addNote(inNote,majBeat,minBeat);
148             measures.add(measure, m);
149     }
150
151     public void insertNote(int measure, int majBeat, int minBeat, ATNote inNote) {
152             int i = measures.size();
153             while (measures.size() <= measure)
154             {
155                     measures.add(new ATMeasure(measures.size()));
156             }
157
158             ATMeasure m = (ATMeasure) measures.remove(measure);
159             m.insertNote(inNote,majBeat,minBeat);
160             measures.add(measure, m);
161
162     }
163
164
165     public void AddNextNote(ATNote inNote)
166     {
167             int tMajBeat, tMinBeat;
168             if (measures.size() <= nextNote.measure )
169             {
170                     measures.add(new ATMeasure(measures.size(), timeSig, keySig));
171             }
172             ATMeasure m = (ATMeasure) measures.get(nextNote.measure);
173             m.addNote(inNote,nextNote.majorBeat,nextNote.minorBeat);
174             measures.add(nextNote.measure, m);
175
176             int [] retout = m.computeTimeElapsed(inNote);
177             tMajBeat = retout[0];
178             tMinBeat = retout[1];
179             nextNote.majorBeat += tMajBeat;
180             nextNote.minorBeat += tMinBeat;
181             while (nextNote.majorBeat > m.getTimeSignature().getNumerator())
182             {
183                     nextNote.majorBeat -= m.getTimeSignature().getNumerator();
184                     nextNote.measure++;
185                     measures.add(new ATMeasure(measures.size(), timeSig,keySig));
186             }
187
188
189
190     }
191
192     private void checkStringErrors(String[] tokens) throws ParseException {
193             ATNoteDuration length;
194             int type;
195             int pitch;
196             int dotted;
197             boolean isDuplet;
198             boolean isTriplet;
199
200             char token0 = tokens[0].charAt(0);
201
202             switch (token0) {
203                     case 'N' :
204                             if (!ATNoteDuration.isValid(tokens[5]))
205                                     throw new ParseException("Invalid Note Length Specifier: " +
206                             if (!ATNote.isValidType(tokens[4]))
207                                     throw new ParseException("Invalid Note Type Specifier: " + to
208
209                             try {
210                                     pitch = Integer.parseInt(tokens[6]);
211                             }
212                             catch (Exception e) {
213                                     throw new ParseException("Invalid Note Pitch Specifier: " + t
```

```
214                                 }
215
216                                 try {
217                                         dotted = Integer.parseInt(tokens[7]);
218                                 }
219                                 catch (Exception e) {
220                                         throw new ParseException("Invalid Note Dotted Specifyer: " +
221                                 }
222
223                                 if (tokens[9].compareToIgnoreCase("TRUE") == 0) {
224                                         isDuplet = true;
225                                 }
226                                 if (tokens[9].compareToIgnoreCase("FALSE") == 0) {
227                                         isDuplet = false;
228                                 }
229                                 else {
230                                         throw new ParseException("Invalid Note isDuplet Specifyer: "
231                                 }
232
233                                 if (tokens[10].compareToIgnoreCase("TRUE") == 0) {
234                                         isTriplet = true;
235                                 }
236                                 if (tokens[10].compareToIgnoreCase("FALSE") == 0) {
237                                         isTriplet = false;
238                                 }
239                                 else {
240                                         throw new ParseException("Invalid Note isTriplet Specifyer: "
241                                 }
242                                 break;
243
244                         case 'T' :
245
246                                 try {
247                                         int num = Integer.parseInt(tokens[1]);
248                                 }
249                                 catch (Exception e) {
250                                         throw new ParseException("Invalid Time Signature Numerator: "
251                                 }
252
253                                 if (!ATNoteDuration.isValid(tokens[2]))
254                                         throw new ParseException("Invalid Time Signature Denominator:
255
256                                 break;
257
258                 }
259
260         }
261
262
263
264 // CUTSTART
265
266
267         public void AddNotesbyString(String noteStr)  throws ParseException
268         {  // Format for string
269            /* Notes on format string
270                 * Measure -- Integer
271                 * MajorBeat -- Integer
272                 * MinorBeat -- Int32
273                 * Type -- "Note" or "Rest"
274                 * Length -- NoteDurationEnum
275                 * Dotted -- Int32 range: 0 - 4
276                 * Tie -- "T" the next note is the note it is tied to
277                 */
278
279             //key=Measure:MajorBeat:MinorBeat|Type,Length,Pitch,Dotted,Tie,IsDuplet,IsTriplet;
280
281                 noteStr = noteStr.toUpperCase();
282                 noteStr = noteStr.replaceAll(" ","");
283                 noteStr = noteStr.replaceAll("\n","");
284                 noteStr = noteStr.replaceAll("\f","");
```

```
285                 noteStr = noteStr.replaceAll("\r","");
286
287
288
289                 String[] lines = noteStr.split(";");
290                 //int i = 0;
291                 for( int i = 0; i < lines.length; i++)
292                 {
293                         String line = lines[i];
294                         if(line.length() <= 1) continue;
295                         String[] tokens = new String[11];
296                         tokens = line.split("[=:,\\|]");
297                         //tokens.GetType();
298                         char token1 = tokens[0].charAt(0);
299                         switch (token1)
300                         {
301                                 case 'N':
302                                         //e = (NoteDurationEnum) Enum.Parse(new NoteDurationEnum().Ge
303                                         if(tokens.length != 11) throw new ParseException("Invalid Num
304                                         checkStringErrors(tokens);
305
306                                         int noteType = ATNote.TYPE_NOTE;
307                                         if(tokens[4].equalsIgnoreCase("Note")) {
308                                                 noteType = ATNote.TYPE_NOTE;
309                                         }
310                                         if(tokens[4].equalsIgnoreCase("Rest")) {
311                                                 noteType = ATNote.TYPE_REST;
312                                         }
313
314
315
316                                         ATNote note = new ATNote(ATNoteDuration.convertFromString(tok
317                                         if(tokens[8].compareToIgnoreCase("T") == 0) {
318                                                 // Get tied note
319                                                 Integer loopAdvance = new Integer(i);
320                                                 note.setTie(getTies(loopAdvance,lines,i));
321                                                 i = loopAdvance.intValue();
322                                         }
323                                         note.isDuplet = (tokens[9].compareToIgnoreCase("TRUE") == 0)
324                                         note.isTriplet = (tokens[10].compareToIgnoreCase("TRUE") == 0
325
326                                         // add code for tied notes
327                                         this.addNote(Integer.parseInt(tokens[1]),Integer.parseInt(tok
328
329                                         break;
330                                 case 'T':
331
332                                         ATTimeSignature tSig = new ATTimeSignature(Integer.parseInt(t
333                                         this.timeSig = tSig;
334                                         break;
335                                 default:
336                                         throw new ParseException("Invalid data in token: " + tokens[0
337
338                         }
339
340
341
342                         /*CSharp.Utils.Tokenizer t = new CSharp.Utils.Tokenizer();
343                         leftPiece = t.GetToken(line,"=|");
344                         rightPiece = t.MoveToNext();
345                         CSharp.Utils.Tokenizer tLeft = new CSharp.Utils.Tokenizer();
346                         tokens[0] = string.Copy(tLeft.GetToken(leftPiece,"=:"));
347                         for(int i = 1; i <= 3; i++)
348                         {
349                                 tokens[i] = string.Copy(tLeft.MoveToNext());
350                         }
351                         CSharp.Utils.Tokenizer tRight = new CSharp.Utils.Tokenizer();
352                         tokens[4] = string.Copy(tRight.GetToken(rightPiece,","));
353                         for(int i = 5; i <= 10; i++)
354                         {
355                                 tokens[i] = string.Copy(tRight.MoveToNext());
```

```
356                         }
357                         */
358
359
360                 }
361
362         }
363
364         private ATNote getTies(Integer newI, String[] lines, int curI)
365         {
366                 if(curI + 2 >= lines.length)
367                 {
368                         newI = new Integer(curI);
369                         return null;
370                 }
371                 String line = lines[++curI];
372                 String[] tokens = new String[11];
373                 tokens = line.split("[=:,\\|]");
374                 try {
375                         checkStringErrors(tokens);
376                 } catch (ParseException e) {
377
378                 }
379
380                 int noteType = ATNote.TYPE_NOTE;
381                 if(tokens[4].equalsIgnoreCase("Note")) {
382                         noteType = ATNote.TYPE_NOTE;
383                 }
384                 if(tokens[4].equalsIgnoreCase("Rest")) {
385                         noteType = ATNote.TYPE_REST;
386                 }
387                 ATNote note = new ATNote(ATNoteDuration.convertFromString(tokens[5]).getLength(),note
388
389                 note.isDuplet = (tokens[9].compareToIgnoreCase("TRUE") == 0) ? true : false;
390                 note.isTriplet = (tokens[10].compareToIgnoreCase("TRUE") == 0) ? true : false;
391
392                 if(tokens[8] == "T")
393                         note.setTie(getTies(newI,lines,curI));
394                 else {
395                         newI = new Integer(curI);
396                 }
397                 return note;
398
399
400
401         }
402
403 /*
404         private void checkStringErrors(string[] tokens)
405         {
406                 NoteDurationEnum length;
407                 Note.NoteTypeEnum type;
408                 int pitch;
409                 int dotted;
410                 bool isDuplet;
411                 bool isTriplet;
412
413                 switch( tokens[0] )
414                 {
415                         case "N":
416                                 try
417                                 {
418                                         length = (NoteDurationEnum) Enum.Parse(new NoteDurationEnum()
419                                 }
420                                 catch (System.Exception)
421                                 {
422                                         throw new InvalidDataInTokenException("Invalid Note Length Sp
423                                 }
424                                 try
425                                 {
426                                         type = (Note.NoteTypeEnum) Enum.Parse(new Note.NoteTypeEnum()
```

```
427                                 }
428                                 catch (System.Exception)
429                                 {
430                                         throw new InvalidDataInTokenException("Invalid Note Type Spec
431                                 }
432                                 try
433                                 {
434                                         pitch = Int32.Parse(tokens[6]);
435                                 }
436                                 catch (System.Exception)
437                                 {
438                                         throw new InvalidDataInTokenException("Invalid Note Pitch Spe
439                                 }
440
441                                 try
442                                 {
443                                         dotted = Int32.Parse(tokens[7]);
444                                 }
445                                 catch (System.Exception)
446                                 {
447                                         throw new InvalidDataInTokenException("Invalid Note Dotted Sp
448                                 }
449                                 try
450                                 {
451                                         isDuplet = Boolean.Parse(tokens[9]);
452                                 }
453                                 catch (System.Exception)
454                                 {
455                                         throw new InvalidDataInTokenException("Invalid Note isDuplet
456                                 }
457                                 try
458                                 {
459                                         isTriplet = Boolean.Parse(tokens[10]);
460                                 }
461                                 catch (System.Exception)
462                                 {
463                                         throw new InvalidDataInTokenException("Invalid Note Dotted Sp
464                                 }
465                                 break;
466                         case "T":
467                                 try
468                                 {
469                                         int num = Int32.Parse(tokens[1]);
470                                 }
471                                 catch (System.Exception)
472                                 {
473                                         throw new InvalidDataInTokenException("Invalid Time Signature
474                                 }
475                                 try
476                                 {
477                                         NoteDurationEnum en = (NoteDurationEnum) Enum.Parse(new NoteD
478                                 }
479                                 catch (System.Exception)
480                                 {
481                                         throw new InvalidDataInTokenException("Invalid Time Signature
482                                 }
483                                 break;
484
485                 }
486
487         }
488
489
490         private Note getTies(out int newI, string [] lines, int curI)
491         {
492                 if(curI + 2 >= lines.GetLength(0))
493                 {
494                         newI = curI;
495                         return null;
496                 }
497                 string line = lines[++curI];
```

```
498                 string [] tokens = new string[11];
499                 tokens = line.Split(new Char[]{'=',':','|',','});
500                 checkStringErrors(tokens);
501                 Note note = new Note((NoteDurationEnum) Enum.Parse(new NoteDurationEnum().GetType(),t
502                 note.IsDuplet = Boolean.Parse(tokens[9]);
503                 note.IsTriplet = Boolean.Parse(tokens[10]);
504                 if(tokens[8] == "T")
505                         note.Tie = getTies(out newI,lines,curI);
506                 else {
507                         newI = curI;
508                 }
509                 return note;
510
511
512
513         }
514
515 */
516
517
518
519
520
521
522
523
524         /* (non-Javadoc)
525          * @see java.lang.Object#equals(java.lang.Object)
526          */
527         public boolean equals(Object o) {
528                 // TODO Auto-generated method stub
529                 if (o == null
530                         || !o.getClass().getName().equals(this.getClass().getName()))
531                         return false;
532                 ATStaff s = (ATStaff) o;
533                 if(this.timeSig != s.timeSig) return false;
534                 if(this.measures.size() != s.measures.size()) return false;
535                 ATMeasure m1, m2;
536                 for(int i = 0; i < this.measures.size(); i++)
537                 {
538                         m1 = (ATMeasure) this.measures.get(i);
539                         m2 = (ATMeasure) s.measures.get(i);
540                         if(!m1.equals(m2)) return false;
541                 }
542                 return true;
543
544         }
545
546         /* (non-Javadoc)
547          * @see java.lang.Object#hashCode()
548          */
549         public int hashCode() {
550                 // TODO Auto-generated method stub
551                 int retval = 0;
552                 Iterator i = measures.iterator();
553                 while(i.hasNext()) {
554                         ATMeasure m = (ATMeasure) i.next();
555                         retval ^= m.hashCode();
556                 }
557                 return retval;
558
559         }
560
561         /* (non-Javadoc)
562          * @see java.lang.Object#toString()
563          */
564         public String toString() {
565                 // TODO Auto-generated method stub
566                 String ret = new String();
567                 int i = 0;
568                 if( timeSig != null)
```

```
569                     ret.concat("T=" + String.valueOf(timeSig.getNumerator()) + ":" + timeSig.getD
570
571             Iterator it = measures.iterator();
572             while(it.hasNext()) {
573                     ATMeasure m = (ATMeasure) it.next();
574                     if( m.getNoteCount() > 0)
575                     {
576                             ret = ret.concat(m.toString() + "\n");
577                             i++;
578                     }
579             }
580             return ret;
581     }
582
583
584 }
585
586
587 /*
588 {
589
590     // Rules for Notes:
591     /* length will only be one of the NoteDurationEnums
592      * dotted fields and tied fields will be used
593      * When inputting notes by string, the first note of a tie will have
594      *  a T in the tied field.  This will mean that it is to be tied to the very next
595      *  Note in the string.
596      *
597      * The tied field of class Note will be a linked list of Notes that is is tied to.
598      *   The isTiedTo field denotes if the note is a recipient of a tie.
599      */
600      /*
601     public struct NextNoteData
602                     {
603                             public NextNoteData(int majBeat, int minBeat, int meas)
604                             {
605                                     majorBeat = majBeat;
606                                     minorBeat = minBeat;
607                                     measure = meas;
608                             }
609
610                             public int majorBeat;
611                             public int minorBeat;
612                             public int measure;
613                     }
614
615     /// <summary>
616     /// Summary description for Class2.
617     /// </summary>
618     public class Staff
619     {
620
621             protected ArrayList measures;  //0 indexed
622             protected NextNoteData nextNote;
623             protected KeySignatureEnum keySig; // = null if time/key varies within staff
624             protected TimeSignature timeSig;   // set this only if time/key remain constant throu
625
626
627             public Staff()
628             {
629                     //
630                     // TODO: Add constructor logic here
631                     //
632                     measures = new ArrayList();
633                     nextNote = new NextNoteData(1,1,0);
634                     keySig = KeySignatureEnum.UnKnown;
635                     timeSig = null;
636             }
637             public Staff(int numOfMeasures)
638             {
639                     measures = new ArrayList();
```

```
640                     nextNote = new NextNoteData(1,1,0);
641
642                     for (int i = 0; i < numOfMeasures; i++)
643                     {
644                             measures[i] = new Measure(measures.Count);
645                     }
646             }
647
648             public void addMeasures(int num)
649             {
650                     for (int i = 0; i < num; i++)
651                     {
652                             measures.Add(new Measure(measures.Count));
653                     }
654             }
655
656             public void addMeasure(Measure m)
657             {
658                     measures.Add(m);
659                     m.MeasureNumber = measures.Count - 1;
660             }
661
662             public Measure getMeasure(int index)
663             {
664                     return (Measure) measures[index + 1];
665             }
666
667             public void SetKeySignature(KeySignatureEnum key)
668             {
669                     foreach (Measure m in measures)
670                     {
671                             m.KeySignature = key;
672                     }
673             }
674
675             public void SetTimeSignature(TimeSignature time)
676             {
677                     foreach (Measure m in measures)
678                     {
679                             m.TimeSignature = time;
680                     }
681             }
682
683             public void addNote(int Measure, int MajBeat, int MinBeat, Note inNote)
684             {
685                     int i = measures.Count;
686                     while (measures.Count <= Measure)
687                     {
688                             measures.Add(new Measure(measures.Count));
689                     }
690
691                     Measure m = (Measure) measures[Measure];
692                     m.addNote(inNote,MajBeat,MinBeat);
693                     measures[Measure] = m;
694             }
695
696
697
698             public void AddNextNote(Note inNote)
699             {
700                     int tMajBeat, tMinBeat;
701                     if (measures.Count <= nextNote.measure )
702                     {
703                             measures.Add(new Measure(measures.Count, timeSig,keySig));
704                     }
705                     Measure m = (Measure) measures[nextNote.measure];
706                     m.addNote(inNote,nextNote.majorBeat,nextNote.minorBeat);
707                     measures[nextNote.measure] = m;
708
709                     m.ComputeTimeElapsed(out tMajBeat,out tMinBeat,inNote);
710                     nextNote.majorBeat += tMajBeat;
```

```
711                     nextNote.minorBeat += tMinBeat;
712                     while (nextNote.majorBeat > m.TimeSignature.Numerator )
713                     {
714                             nextNote.majorBeat -= m.TimeSignature.Numerator;
715                             nextNote.measure++;
716                             measures.Add(new Measure(measures.Count, timeSig,keySig));
717                     }
718
719
720
721             }
722
723             public void AddNotesbyString(string noteStr)
724             {  // Format for string
725                /* Notes on format string
726                     * Measure -- Integer
727                     * MajorBeat -- Integer
728                     * MinorBeat -- Int32
729                     * Type -- "Note" or "Rest"
730                     * Length -- NoteDurationEnum
731                     * Dotted -- Int32 range: 0 - 4
732                     * Tie -- "T" the next note is the note it is tied to
733                     */
734                     /*
735                // key=Measure:MajorBeat:MinorBeat|Type,Length,Pitch,Dotted,Tie,IsDuplet,IsTriplet
736                     noteStr = noteStr.ToUpper();
737                     noteStr = noteStr.Replace(" ","");
738                     noteStr = noteStr.Replace("\n","");
739                     noteStr = noteStr.Replace("\f","");
740                     noteStr = noteStr.Replace("\r","");
741
742
743
744                     string [] lines = noteStr.Split(';');
745                     //int i = 0;
746                     for( int i = 0; i < lines.GetLength(0); i++)
747                     {
748                             string line = lines[i];
749                             if(line.Length <= 1) continue;
750                             string [] tokens = new string[11];
751                             tokens = line.Split(new Char[]{'=',':','|',','});
752                             tokens.GetType();
753                             switch (tokens[0])
754                             {
755                                     case "N":
756                                             //e = (NoteDurationEnum) Enum.Parse(new NoteDurationE
757                                             if(tokens.GetLength(0) != 11) throw new InvalidNumber
758                                             checkStringErrors(tokens);
759
760                                             Note note = new Note((NoteDurationEnum) Enum.Parse(ne
761                                             if(tokens[8] == "T")
762                                                     note.Tie = getTies(out i,lines,i);
763                                             note.IsDuplet = Boolean.Parse(tokens[9]);
764                                             note.IsTriplet = Boolean.Parse(tokens[10]);
765                                             // add code for tied notes
766                                             this.addNote(Int32.Parse(tokens[1]),Int32.Parse(token
767
768                                             break;
769                                     case "T":
770
771                                             Aaron.SheetMusic1.TimeSignature tSig = new Aaron.Shee
772                                             this.TimeSignature = tSig;
773                                             break;
774                                     default:
775                                             throw new InvalidDataInTokenException("Invalid data i
776                                             break;
777
778                             }
779
780
781
```

```
782                             /*CSharp.Utils.Tokenizer t = new CSharp.Utils.Tokenizer();
783                             leftPiece = t.GetToken(line,"=|");
784                             rightPiece = t.MoveToNext();
785                             CSharp.Utils.Tokenizer tLeft = new CSharp.Utils.Tokenizer();
786                             tokens[0] = string.Copy(tLeft.GetToken(leftPiece,"=:"));
787                             for(int i = 1; i <= 3; i++)
788                             {
789                                     tokens[i] = string.Copy(tLeft.MoveToNext());
790                             }
791                             CSharp.Utils.Tokenizer tRight = new CSharp.Utils.Tokenizer();
792                             tokens[4] = string.Copy(tRight.GetToken(rightPiece,","));
793                             for(int i = 5; i <= 10; i++)
794                             {
795                                     tokens[i] = string.Copy(tRight.MoveToNext());
796                             }
797                             *//*
798
799                     }
800
801             }
802
803             private void checkStringErrors(string[] tokens)
804             {
805                     NoteDurationEnum length;
806                     Note.NoteTypeEnum type;
807                     int pitch;
808                     int dotted;
809                     bool isDuplet;
810                     bool isTriplet;
811
812                     switch( tokens[0] )
813                     {
814                             case "N":
815                                     try
816                                     {
817                                             length = (NoteDurationEnum) Enum.Parse(new NoteDurati
818                                     }
819                                     catch (System.Exception)
820                                     {
821                                             throw new InvalidDataInTokenException("Invalid Note L
822                                     }
823                                     try
824                                     {
825                                             type = (Note.NoteTypeEnum) Enum.Parse(new Note.NoteTy
826                                     }
827                                     catch (System.Exception)
828                                     {
829                                             throw new InvalidDataInTokenException("Invalid Note T
830                                     }
831                                     try
832                                     {
833                                             pitch = Int32.Parse(tokens[6]);
834                                     }
835                                     catch (System.Exception)
836                                     {
837                                             throw new InvalidDataInTokenException("Invalid Note P
838                                     }
839
840                                     try
841                                     {
842                                             dotted = Int32.Parse(tokens[7]);
843                                     }
844                                     catch (System.Exception)
845                                     {
846                                             throw new InvalidDataInTokenException("Invalid Note D
847                                     }
848                                     try
849                                     {
850                                             isDuplet = Boolean.Parse(tokens[9]);
851                                     }
852                                     catch (System.Exception)
```

```
853                                     {
854                                             throw new InvalidDataInTokenException("Invalid Note i
855                                     }
856                                     try
857                                     {
858                                             isTriplet = Boolean.Parse(tokens[10]);
859                                     }
860                                     catch (System.Exception)
861                                     {
862                                             throw new InvalidDataInTokenException("Invalid Note D
863                                     }
864                                     break;
865                             case "T":
866                                     try
867                                     {
868                                             int num = Int32.Parse(tokens[1]);
869                                     }
870                                     catch (System.Exception)
871                                     {
872                                             throw new InvalidDataInTokenException("Invalid Time S
873                                     }
874                                     try
875                                     {
876                                             NoteDurationEnum en = (NoteDurationEnum) Enum.Parse(n
877                                     }
878                                     catch (System.Exception)
879                                     {
880                                             throw new InvalidDataInTokenException("Invalid Time S
881                                     }
882                                     break;
883
884                     }
885
886             }
887
888
889             private Note getTies(out int newI, string [] lines, int curI)
890             {
891                     if(curI + 2 >= lines.GetLength(0))
892                     {
893                             newI = curI;
894                             return null;
895                     }
896                     string line = lines[++curI];
897                     string [] tokens = new string[11];
898                     tokens = line.Split(new Char[]{'=',':','|',','});
899                     checkStringErrors(tokens);
900                     Note note = new Note((NoteDurationEnum) Enum.Parse(new NoteDurationEnum().Get
901                     note.IsDuplet = Boolean.Parse(tokens[9]);
902                     note.IsTriplet = Boolean.Parse(tokens[10]);
903                     if(tokens[8] == "T")
904                             note.Tie = getTies(out newI,lines,curI);
905                     else {
906                             newI = curI;
907                     }
908                     return note;
909
910
911
912             }
913             public override int GetHashCode()
914             {
915                     int retval = 0;
916                     foreach(Measure m in measures)
917                     {
918                             retval ^= m.GetHashCode();
919                     }
920                     return retval;
921             }
922
923
```

```
924             public override bool Equals(Object obj)
925             {
926                     if (obj == null || this.GetType() != obj.GetType()) return false;
927                     Staff s = (Staff) obj;
928                     if(this.TimeSignature != s.TimeSignature) return false;
929                     if(this.measures.Count != s.measures.Count) return false;
930                     Measure m1, m2;
931                     for(int i = 0; i < this.measures.Count; i++)
932                     {
933                             m1 = (Measure) this.measures[i];
934                             m2 = (Measure) s.measures[2];
935                             if(m1 != m2) return false;
936                     }
937                     return true;
938
939             }
940
941             public static bool Equals(Staff s1, Staff s2)
942             {
943                     Object obj1, obj2;
944                     obj1 = s1;
945                     obj2 = s2;
946                     if(obj1 == null && obj2 == null) return true;
947                     if(obj1 == null) return false;
948                     if(obj2 == null) return false;
949
950                     if(s1.TimeSignature != s2.TimeSignature) return false;
951                     if(s1.measures.Count != s2.measures.Count) return false;
952                     Measure m1, m2;
953                     for(int i = 0; i < s1.measures.Count; i++)
954                     {
955                             m1 = (Measure) s1.measures[i];
956                             m2 = (Measure) s2.measures[i];
957                             if(m1 != m2) return false;
958                     }
959                     return true;
960
961
962             }
963
964
965             public static bool operator==(Staff s1, Staff s2)
966             {
967                     return Equals(s1,s2);
968             }
969
970             public static bool operator!=(Staff s1, Staff s2)
971             {
972                     return !Equals(s1,s2);
973             }
974
975
976
977             new public string ToString()
978             {
979                     string ret = "";
980                     int i = 0;
981                     if( timeSig != null)
982                             ret += "T=" + timeSig.Numerator.ToString() + ":" + timeSig.Denominato
983                     foreach (Measure m in measures)
984                     {
985                             //ret += "Measure " + (i++);
986                             if( m.NoteCount > 0)
987                             {
988                                     ret += m.ToString() + "\n";
989                                     i++;
990                             }
991                     }
992                     return ret;
993             }
994
```

```
 995                 public KeySignatureEnum KeySignature
 996                 {
 997                         get
 998                         {
 999                                 return keySig;
1000                         }
1001                         set
1002                         {
1003                                 keySig = value;
1004                         }
1005                 }
1006
1007                 public TimeSignature TimeSignature
1008                 {
1009                         get
1010                         {
1011                                 return timeSig;
1012                         }
1013                         set
1014                         {
1015                                 timeSig = value;
1016                         }
1017                 }
1018
1019         }
1020 }
1021 */
```

```
 1 /*
 2  * Created on Jan 12, 2004
 3  *
 4  * To change the template for this generated file go to
 5  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 6  */
 7 package com.aaron.MidiReader;
 8
 9
10
11 /**
12  * @author aaron
13  *
14  * To change the template for this generated type comment go to
15  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
16  */
17 public final class ATNoteDuration {
18         public static final int WHOLE = 64;
19         public static final int HALF = 32;
20         public static final int QUARTER = 16;
21         public static final int EIGHTH = 8;
22         public static final int SIXTEENTH = 4;
23         public static final int THIRTYSECOND = 2;
24         public static final int SIXTYFORTH = 1;
25
26         private int len;
27
28         public ATNoteDuration(int length) {
29                 len = length;
30         }
31
32         public int getLength() {
33                 return len;
34         }
35
36         /* (non-Javadoc)
37          * @see java.lang.Object#toString()
38          */
39         public String toString() {
40                 // TODO Auto-generated method stub
41                 switch(len) {
42                         case WHOLE:
43                                 return new String("Whole");
44                         case HALF:
45                                 return new String("Half");
46                         case QUARTER:
47                                 return new String("Quarter");
48                         case EIGHTH:
49                                 return new String("Eigthth");
50                         case SIXTEENTH:
51                                 return new String("Sixteenth");
52                         case THIRTYSECOND:
53                                 return new String("ThirtySecond");
54                         case SIXTYFORTH:
55                                 return new String("SixtyForth");
56                 }
57                 return new String();
58         }
59
60         public static ATNoteDuration convertFromString(String s) {
61                 if(s.equalsIgnoreCase("Whole")) {
62                         return new ATNoteDuration(WHOLE);
63                 }
64                 if(s.equalsIgnoreCase("Half")) {
65                         return new ATNoteDuration(HALF);
66                 }
67                 if(s.equalsIgnoreCase("Quarter")) {
68                         return new ATNoteDuration(QUARTER);
69                 }
70                 if(s.equalsIgnoreCase("Eigthth")) {
71                         return new ATNoteDuration(EIGHTH);
```

```
72                  }
73                  if(s.equalsIgnoreCase("Sixteenth")) {
74                          return new ATNoteDuration(SIXTEENTH);
75                  }
76                  if(s.equalsIgnoreCase("ThirtySecond")) {
77                          return new ATNoteDuration(THIRTYSECOND);
78                  }
79                  if(s.equalsIgnoreCase("SixtyForth")) {
80                          return new ATNoteDuration(SIXTYFORTH);
81                  }
82
83                  return new ATNoteDuration(0);
84
85
86          }
87
88          public static boolean isValid(String s) {
89
90                  if(s.equalsIgnoreCase("Whole")) {
91                          return true;
92                  }
93                  if(s.equalsIgnoreCase("Half")) {
94                          return true;
95                  }
96                  if(s.equalsIgnoreCase("Quarter")) {
97                          return true;
98                  }
99                  if(s.equalsIgnoreCase("Eigthth")) {
100                         return true;
101                 }
102                 if(s.equalsIgnoreCase("Sixteenth")) {
103                         return true;
104                 }
105                 if(s.equalsIgnoreCase("ThirtySecond")) {
106                         return true;
107                 }
108                 if(s.equalsIgnoreCase("SixtyForth")) {
109                         return true;
110                 }
111
112                 return false;
113         }
114
115         public static int convertToTimeSigDenominator(ATNoteDuration nd) {
116                 int ndLen = nd.getLength();
117                 int ndRoot = (int)log2(ndLen);
118                 return (int) Math.floor(Math.pow(2,6 - ndRoot));
119
120
121         }
122
123         private static double log2(double x) {
124                 return 1.4426950408889634 * Math.log(x);
125         }
126
127
128 }
```

```
 1 /*
 2  * Created on Jan 12, 2004
 3  *
 4  * To change the template for this generated file go to
 5  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 6  */
 7 package com.aaron.MidiReader;
 8
 9 /**
10  * @author aaron
11  *
12  * To change the template for this generated type comment go to
13  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
14  */
15 public class ATNoteBeat extends Object implements Comparable  {
16         protected ATNote note;
17         protected int majorBeat;
18         protected int minorBeat;
19
20         public ATNoteBeat(ATNote InNote, int MajorBeat, int MinorBeat)
21         {
22                 note = InNote;
23                 majorBeat = MajorBeat;
24                 minorBeat = MinorBeat;
25         }
26
27         public ATNote getNote() {
28                 return note;
29         }
30         public int getMajorBeat() {
31                 return majorBeat;
32         }
33         public int getMinorBeat() {
34                 return minorBeat;
35         }
36
37         public int compareTo(Object o)
38         {
39                 if (!o.getClass().getName().equals(this.getClass().getName()))
40                         throw (new ClassCastException());
41                 if (o.equals(this))
42                         return 0;
43
44                 ATNoteBeat nb = (ATNoteBeat) o;
45                 if(this.majorBeat == nb.majorBeat && this.minorBeat == nb.minorBeat && this.note.compa
46                 {
47                         return 0;
48                 }
49
50                 if(this.majorBeat < nb.majorBeat)
51                         return -1;
52                 else if(this.majorBeat > nb.majorBeat)
53                         return 1;
54                 else
55                 {
56                         if (this.minorBeat < nb.minorBeat)
57                                 return -1;
58                         else if (this.minorBeat > nb.minorBeat)
59                                 return 1;
60                         else
61                         {
62                                 return this.note.compareTo(nb.getNote());
63                         }
64                 }
65
66         }
67
68         public boolean equals(Object obj) {
69                 if (obj == null || !obj.getClass().getName().equals(this.getClass().getName())) return
70                 ATNoteBeat n = (ATNoteBeat) obj;
71                 return (this.majorBeat == n.majorBeat) && (this.minorBeat == n.minorBeat) && (this.not
```

```
72
73          }
74
75          public int hashCode() {
76                  return (majorBeat << 5) ^ (minorBeat << 3) ^ note.hashCode();
77
78          }
79
80
81          /* (non-Javadoc)
82           * @see java.lang.Object#toString()
83           */
84          public String toString() {
85                  // TODO Auto-generated method stub
86                  return majorBeat + ":" + minorBeat + "|" + note.toString();
87          }
88
89  }
90
91  /*
92  protected class NoteBeat : System.IComparable
93  {
94          protected Note note;
95          protected int majorBeat;
96          protected int minorBeat;
97
98          public NoteBeat(Note InNote, int MajorBeat, int MinorBeat)
99          {
100                 note = InNote;
101                 majorBeat = MajorBeat;
102                 minorBeat = MinorBeat;
103         }
104
105         public Note Note
106         {
107                 get
108                 {
109                         return note;
110                 }
111         }
112         public int MajorBeat
113         {
114                 get
115                 {
116                         return majorBeat;
117                 }
118         }
119         public int MinorBeat
120         {
121                 get
122                 {
123                         return minorBeat;
124                 }
125         }
126
127         public int CompareTo(Object obj)
128         {
129                 if(obj.GetType() != this.GetType()) throw(new System.ArgumentException());
130                 if(obj == this) return 0;
131
132                 NoteBeat nb = (NoteBeat) obj;
133                 if(this.majorBeat == nb.majorBeat && this.minorBeat == nb.minorBeat && this.note.Compa
134                 {
135                         return 0;
136                 }
137
138                 if(this.majorBeat < nb.majorBeat)
139                         return -1;
140                 else if(this.majorBeat > nb.majorBeat)
141                         return 1;
142                 else
```

```
143                 {
144                         if (this.minorBeat < nb.minorBeat)
145                                 return -1;
146                         else if (this.minorBeat > nb.minorBeat)
147                                 return 1;
148                         else
149                         {
150                                 return this.note.CompareTo(nb.Note);
151                         }
152                 }
153
154         }
155
156         public override int GetHashCode()
157         {
158                 return (majorBeat << 5) ^ (minorBeat << 3) ^ note.GetHashCode();
159         }
160
161         public override bool Equals(Object obj)
162         {
163                 if (obj == null || this.GetType() != obj.GetType()) return false;
164                 NoteBeat n = (NoteBeat) obj;
165                 return (this.majorBeat == n.majorBeat) && (this.minorBeat == n.minorBeat) && (this.not
166         }
167
168         public static bool Equals(NoteBeat nb1, NoteBeat nb2)
169         {
170                 Object obj1, obj2;
171                 obj1 = nb1;
172                 obj2 = nb2;
173                 if(obj1 == null && obj2 == null) return true;
174                 if(obj1 == null) return false;
175                 if(obj2 == null) return false;
176                 return (nb1.majorBeat == nb2.majorBeat) && (nb1.minorBeat == nb2.minorBeat) && (nb1.no
177         }
178
179
180
181         public static bool operator==(NoteBeat nb1, NoteBeat nb2)
182         {
183                 return Equals(nb1,nb2);
184         }
185         public static bool operator!=(NoteBeat nb1, NoteBeat nb2)
186         {
187                 return !Equals(nb1,nb2);
188         }
189         new public string ToString()
190         {
191                 return majorBeat + ":" + minorBeat + "|" + note.ToString();
192         }
193 }
194 */
```

```
 1 /*
 2  * Created on Jan 12, 2004
 3  *
 4  * To change the template for this generated file go to
 5  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 6  */
 7 package com.aaron.MidiReader;
 8
 9 /**
10  * @author aaron
11  *
12  * To change the template for this generated type comment go to
13  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
14  */
15 public class ATNote implements Comparable {
16
17         public static final int TYPE_NOTE = 1;
18         public static final int TYPE_REST = 2;
19
20         protected ATNoteDuration length;
21         protected int type;
22         protected int dotted;
23         protected int pitch;
24         protected ATNote tie;
25         protected boolean isTiedTo;
26         protected ATNote beam;
27         protected boolean isTriplet;
28         protected boolean isDuplet;
29
30         /** Creates a new instance of ATNote */
31         public ATNote() {
32                 this(ATNoteDuration.QUARTER, TYPE_NOTE, 60, 0);
33         }
34
35         public ATNote(int length, int type, int pitch) {
36                 this(length, type, pitch, 0);
37         }
38
39         public ATNote(int length, int type, int pitch, int dotted) {
40                 this.length = new ATNoteDuration(length);
41                 this.type = type;
42                 if (this.type == TYPE_REST) {
43                         this.pitch = -1;
44                 } else {
45                         this.pitch = pitch;
46                 }
47                 this.dotted = dotted;
48                 this.tie = null;
49                 this.isTiedTo = false;
50                 this.beam = null;
51                 this.isTriplet = false;
52                 this.isDuplet = false;
53         }
54
55         public int getLength() {
56                 return this.length.getLength();
57         }
58         public void setLength(int value) {
59                 this.length = new ATNoteDuration(value);
60         }
61
62         public int getType() {
63                 return this.type;
64         }
65         public void setType(int value) {
66                 this.type = value;
67         }
68
69         public int getDotted() {
70                 return this.dotted;
71         }
```

```
 72          public void setDotted(int value) {
 73                  //! add code to make max number of dots
 74                  this.dotted = value;
 75          }
 76
 77          public int getPitch() {
 78                  return this.pitch;
 79          }
 80          public void setPitch(int value) {
 81                  //! Add code to make 1 <= pitch <= 127
 82                  if (this.type != TYPE_REST)
 83                          pitch = value;
 84          }
 85
 86          public ATNote getBeam() {
 87                  return this.beam;
 88          }
 89          public void setBeam(ATNote value) {
 90                  beam = value;
 91          }
 92
 93          public boolean isTriplet() {
 94                  return this.isTriplet;
 95          }
 96          public void setTriplet(boolean value) {
 97                  isTriplet = value;
 98          }
 99
100          public boolean isDuplet() {
101                  return isDuplet;
102          }
103          public void setDuplet(boolean value) {
104                  isDuplet = value;
105          }
106
107          public ATNote getTie() {
108                  return tie;
109          }
110          public void setTie(ATNote n) {
111                  tie = n;
112          }
113
114          public int compareTo(Object o) {
115                  if (!o.getClass().getName().equals(this.getClass().getName()))
116                          throw (new ClassCastException());
117                  if (o.equals(this))
118                          return 0;
119
120                  ATNote n = (ATNote) o;
121                  if (this.pitch < n.pitch)
122                          return -1;
123                  else if (this.pitch > n.pitch)
124                          return 1;
125                  else {
126                          if (this.getTotalLength() < n.getTotalLength())
127                                  return -1;
128                          else if (this.getTotalLength() > n.getTotalLength())
129                                  return 1;
130                          else
131                                  return 0;
132                  }
133          }
134
135          public int getTotalLength() {
136                  int retval = 0;
137                  retval += length.getLength();
138                  int tdot = dotted;
139                  int tlength = length.getLength();
140                  while (tdot > 0) {
141                          tlength /= 2;
142                          retval += tlength;
```

```
143                     tdot--;
144             }
145
146             if (tie != null) {
147                     retval += tie.getTotalLength();
148             }
149             return retval;
150     }
151
152     public boolean equals(Object o) {
153             if (o == null
154                     || !o.getClass().getName().equals(this.getClass().getName()))
155                     return false;
156             ATNote n = (ATNote) o;
157             return (this.type == n.type)
158                     && (this.getTotalLength() == n.getTotalLength())
159                     && (this.pitch == n.pitch);
160     }
161
162     public int hashCode() {
163             return pitch ^ length.getLength() ^ dotted;
164
165     }
166
167
168     public String toString() {
169             String retStr = new String("");
170             String strType = new String(String.valueOf(this.type));
171             String strLength = new String(String.valueOf(this.length));
172             String strPitch = new String(String.valueOf(this.pitch));
173             String strDotted = new String(String.valueOf(this.dotted));
174             String strTie = (tie == null) ? new String("null") : new String("T");
175             String strIsTriplet = new String(String.valueOf(this.isTriplet));
176             String strIsDuplet = new String(String.valueOf(this.isDuplet));
177
178             retStr = strType + "," + strLength + "," + strPitch + "," + strDotted + "," + strTie +
179
180             return retStr;
181
182     }
183
184     public static boolean isValidType(String s) {
185             if(s.equalsIgnoreCase("Note")) {
186                     return true;
187             }
188             if(s.equalsIgnoreCase("Rest")) {
189                     return true;
190             }
191             return false;
192     }
193 }
194
195 /*          public override int GetHashCode()
196             {
197                     return pitch ^ (int) length ^ dotted;
198             }
199
200             public static bool Equals(Note n1, Note n2)
201             {
202                     Object obj1, obj2;
203                     obj1 = n1;
204                     obj2 = n2;
205                     if(obj1 == null && obj2 == null) return true;
206                     if(obj1 == null) return false;
207                     if(obj2 == null) return false;
208                     return (n1.type == n2.type) && (n1.getTotalLength() == n2.getTotalLength()) &&
209             }
210
211             public static bool operator==(Note n1, Note n2)
212             {
213 //                  if(n1.type == n2.type && n1.getTotalLength() == n2.getTotalLength()) return tr
```

```
214 //                      return false;
215                         return Note.Equals(n1,n2);
216
217                 }
218
219                 public static bool operator!=(Note n1, Note n2)
220                 {
221
222                         return !Note.Equals(n1,n2);
223                 }
224
225
226 */
227
228 /* (non-Javadoc)
229  * @see java.lang.Comparable#compareTo(java.lang.Object)
230  */
```

```
 1 /*
 2  * Created on Feb 12, 2004
 3  *
 4  * To change the template for this generated file go to
 5  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 6  */
 7 package com.aaron.MidiReader;
 8
 9 import java.util.*;
10
11 /**
12  * @author aaron
13  *
14  * To change the template for this generated type comment go to
15  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
16  */
17 public class ATMeasure {
18         protected TreeSet notelist;
19         protected ATTimeSignature timeSig;
20         protected int key;
21         protected int measureNumber;
22
23         protected int totalLength;
24
25         public ATMeasure(int MeasNum)
26         {
27                 //
28                 // TODO: Add constructor logic here
29                 //
30                 timeSig = new ATTimeSignature(4, new ATNoteDuration(ATNoteDuration.QUARTER));
31                 key = ATKeySig.CMajor;
32                 notelist = new TreeSet();
33                 measureNumber = MeasNum;
34                 totalLength = 0;
35         }
36         public ATMeasure(int MeasNum, ATTimeSignature tSig)
37         {
38                 if (tSig == null)
39                 {
40                         tSig =  new ATTimeSignature(4, new ATNoteDuration(ATNoteDuration.QUARTER));
41                 }
42
43                 notelist = new TreeSet();
44                 timeSig = tSig;
45                 key = ATKeySig.CMajor;
46                 measureNumber = MeasNum;
47                 totalLength = 0;
48
49         }
50         public ATMeasure(int MeasNum, ATTimeSignature tSig, int inKey)
51         {
52                 if (tSig == null)
53                 {
54                         tSig =  new ATTimeSignature(4, new ATNoteDuration(ATNoteDuration.QUARTER));
55                 }
56                 if (inKey == ATKeySig.UnKnown)
57                 {
58                         inKey = ATKeySig.CMajor;
59                 }
60                 notelist = new TreeSet();
61                 timeSig = tSig;
62                 key = inKey;
63                 measureNumber = MeasNum;
64                 totalLength = 0;
65
66         }
67
68
69         //measurenumber
70         //keysig
71         //notecount
```

```
72
73          public int getMeasureNumber() {
74                  return measureNumber;
75          }
76          public void setMeasureNumber(int num) {
77                  measureNumber = num;
78          }
79          public int getKeySignature() {
80                  return key;
81          }
82          public void setKeySignature(int keySig) {
83                  key = keySig;
84          }
85          public int getNoteCount() {
86                  return notelist.size();
87          }
88          public ATTimeSignature getTimeSignature() {
89                  return timeSig;
90          }
91          public void setTimeSignature(ATTimeSignature time) {
92                  timeSig = time;
93          }
94          public boolean isMeasureComplete() {
95                  return timeSig.getNumerator() * timeSig.getDenominator().getLength() == totalLength;
96          }
97
98          /**
99           *
100         * @return An iterator of ATNoteBeat objects contained in the measure
101         */
102         public Iterator iterator() {
103                 return notelist.iterator();
104         }
105
106         public void addNote(ATNote iNote, int majBeat, int minBeat)
107         {
108                 // make sure beat is within bounds
109                 if(majBeat < 1 || majBeat > timeSig.getNumerator()) ;// THROW EXCEPTION
110                 if(minBeat < 0 || minBeat > 119) ;// THROW EXCEPTION
111
112                 notelist.add(new ATNoteBeat(iNote,majBeat,minBeat));
113                 totalLength += iNote.getLength();
114
115         }
116         public void insertNote(ATNote iNote, int majBeat, int minBeat) {
117                 // make sure beat is within bounds
118                 if(majBeat < 1 || majBeat > timeSig.getNumerator()) ;// THROW EXCEPTION
119                 if(minBeat < 0 || minBeat > 119) ;// THROW EXCEPTION
120
121                 Object[] o = notelist.toArray();
122                 ATNoteBeat[] nbs = new ATNoteBeat[o.length];
123                 for(int k = 0; k < o.length; k++) {
124                         nbs[k] = (ATNoteBeat) o[k];
125                 }
126
127                 int i = 0;
128                 while(i < nbs.length && nbs[i].getMajorBeat() < majBeat ) {
129                         i++;
130                 }
131                 while(i < nbs.length && nbs[i].getMinorBeat() < minBeat && nbs[i].getMajorBeat() <= ma
132                         i++;
133                 }
134
135                 if(i == nbs.length) {
136                         addNote(iNote, majBeat, minBeat);
137                         return;
138                 }
139
140                 for(int j = i; j < nbs.length; j++) {
141                         int[] beats = computeNewTime(nbs[j].getMajorBeat(), nbs[j].getMinorBeat(), iNo
142                         nbs[j] = new ATNoteBeat(nbs[j].getNote(), beats[0], beats[1]);
```

```
143                 }
144
145                 notelist = new TreeSet();
146                 for(i = 0; i < nbs.length; i++) {
147                         notelist.add(nbs[i]);
148                 }
149                 addNote(iNote, majBeat, minBeat);
150
151
152         }
153
154         /**
155          *
156          * @param inNote
157          * @return 2 member array [0] = major Beat, [1] = minorBeat
158          */
159         public int[] computeTimeElapsed(ATNote inNote)
160         {
161                 int[] retval = new int[2];
162                 retval[0] = (int) Math.floor((double) inNote.getTotalLength() / (double) timeSig.getDe
163                 retval[1] = (int) ((((double)inNote.getTotalLength() / (double) timeSig.getDenominator
164                 return retval;
165         }
166
167         public int[] computeNewTime(int majorBeat, int minorBeat, int length) {
168                 int[] retval = new int[2];
169
170                 int majLength = (int) Math.floor(length / timeSig.getDenominator().getLength());
171                 retval[0] = majorBeat + majLength;
172                 int leftOver = length - (majLength * timeSig.getDenominator().getLength());
173                 retval[1] = (int) Math.floor((double)leftOver * 15);
174                 retval[1] += minorBeat;
175                 while(retval[1] >= 240) {
176                         retval[0] += 1;
177                         retval[1] -= 240;
178
179                 }
180                 return retval;
181         }
182
183         /* (non-Javadoc)
184          * @see java.lang.Object#hashCode()
185          */
186         public int hashCode() {
187                 int retval = 0;
188                 Iterator item = notelist.iterator();
189
190                 while(item.hasNext())
191                 {
192                         retval ^= item.next().hashCode();
193                 }
194                 return retval;
195         }
196
197         /* (non-Javadoc)
198          * @see java.lang.Object#equals(java.lang.Object)
199          */
200         public boolean equals(Object o) {
201                 // TODO Auto-generated method stub
202                 if (o == null
203                         || !o.getClass().getName().equals(this.getClass().getName()))
204                         return false;
205
206                 ATMeasure m = (ATMeasure) o;
207                 if(this.notelist.size() != m.notelist.size()) return false;
208                 ATNoteBeat n1, n2;
209                 Iterator i1 = this.notelist.iterator();
210                 Iterator i2 = m.notelist.iterator();
211                 while(i1.hasNext() && i2.hasNext()) {
212                         n1 = (ATNoteBeat) i1.next();
213                         n2 = (ATNoteBeat) i2.next();
```

```
214                     if(!n1.equals(n2)) return false;
215             }
216             return true;
217     }
218
219
220
221     /* (non-Javadoc)
222      * @see java.lang.Object#toString()
223      */
224     public String toString() {
225             // TODO Auto-generated method stub
226             String ret = new String();
227             Iterator i = notelist.iterator();
228             while(i.hasNext()) {
229                     ATNoteBeat nb = (ATNoteBeat) i.next();
230                     ret = ret.concat("N=" + measureNumber + ":" + nb.toString() + "\n");
231             }
232             return ret;
233     }
234
235 }
236
237 /*
238     protected ArrayList notelist;
239     protected TimeSignature timeSig;
240     protected KeySignatureEnum key;
241     protected int measureNumber;
242
243     /* Status:
244      *    MaxBeat - the major beat of the measure based on time signature
245      *    MinBeat - a portion of a beat.  valid numbers are 0-119. therefore
246      *              1/8note = 60
247      *              1/16note = 30
248      *              1/32note = 15
249      *                              1/4triplet = 40
250      *              1/8triplet = 20
251      *
252
253
254     public Measure(int MeasNum)
255     {
256             //
257             // TODO: Add constructor logic here
258             //
259             timeSig = new TimeSignature(4,NoteDurationEnum.Quarter);
260             key = KeySignatureEnum.CMajor;
261             notelist = new ArrayList();
262             measureNumber = MeasNum;
263     }
264     public Measure(int MeasNum, TimeSignature tSig)
265     {
266             if (tSig == null)
267             {
268                     tSig = new TimeSignature(4,NoteDurationEnum.Quarter);
269             }
270
271             notelist = new ArrayList();
272             timeSig = tSig;
273             key = KeySignatureEnum.CMajor;
274             measureNumber = MeasNum;
275
276     }
277     public Measure(int MeasNum, TimeSignature tSig, KeySignatureEnum inKey)
278     {
279             if (tSig == null)
280             {
281                     tSig = new TimeSignature(4,NoteDurationEnum.Quarter);
282             }
283             if (inKey == KeySignatureEnum.UnKnown)
284             {
```

```
285                     inKey = KeySignatureEnum.CMajor;
286             }
287             notelist = new ArrayList();
288             timeSig = tSig;
289             key = inKey;
290             measureNumber = MeasNum;
291
292     }
293
294
295
296
297     public void addNote(Note iNote, int majBeat, int minBeat)
298     {
299             // make sure beat is within bounds
300             if(majBeat < 1 || majBeat > timeSig.Numerator) ;// THROW EXCEPTION
301             if(minBeat < 0 || minBeat > 119) ;// THROW EXCEPTION
302
303             notelist.Add(new NoteBeat(iNote,majBeat,minBeat));
304     }
305
306     public override int GetHashCode()
307     {
308             int retval = 0;
309             foreach(NoteBeat nb in notelist)
310             {
311                     retval ^= nb.GetHashCode();
312             }
313             return retval;
314     }
315
316     public override bool Equals(Object obj)
317     {
318             if (obj == null || this.GetType() != obj.GetType()) return false;
319             Measure m = (Measure) obj;
320             if(this.notelist.Count != m.notelist.Count) return false;
321             NoteBeat n1, n2;
322             for(int i = 0; i < this.notelist.Count; i++)
323             {
324                     n1 = (NoteBeat) this.notelist[i];
325                     n2 = (NoteBeat) m.notelist[i];
326                     if (n1 != n2) return false;
327             }
328             return true;
329
330     }
331
332     public static bool Equals( Measure m1, Measure m2)
333     {
334             Object obj1, obj2;
335             obj1 = m1;
336             obj2 = m2;
337             if(obj1 == null && obj2 == null) return true;
338             if(obj1 == null) return false;
339             if(obj2 == null) return false;
340             NoteBeat n1, n2;
341             if(m1.notelist.Count != m2.notelist.Count) return false;
342             for(int i = 0; i < m1.notelist.Count; i++)
343             {
344                     n1 = (NoteBeat) m1.notelist[i];
345                     n2 = (NoteBeat) m2.notelist[i];
346                     if (n1 != n2) return false;
347             }
348             return true;
349
350     }
351
352
353     public static bool operator==(Measure m1, Measure m2)
354     {
355             return Equals(m1,m2);
```

```
356         }
357
358         public static bool operator!=(Measure m1, Measure m2)
359         {
360                 return !Equals(m1,m2);
361         }
362
363
364         new public string ToString()
365         {
366                 string ret = "";
367                 notelist.Sort();
368                 foreach (NoteBeat nb in notelist)
369                 {
370                         ret += "N=" + measureNumber + ":" + nb.ToString() + "\n";
371                 }
372                 return ret;
373         }
374
375         public int MeasureNumber
376         {
377                 get
378                 {
379                         return measureNumber;
380                 }
381                 set
382                 {
383                         measureNumber = value;
384                 }
385         }
386
387
388         public int NoteCount
389         {
390                 get
391                 {
392                         return notelist.Count;
393                 }
394         }
395
396                 public TimeSignature TimeSignature
397         {
398                 get
399                 {
400                         return timeSig;
401                 }
402                 set
403                 {
404                         timeSig = value;
405                 }
406         }
407
408         public KeySignatureEnum KeySignature
409         {
410                 get
411                 {
412                         return key;
413                 }
414                 set
415                 {
416                         key = value;
417                 }
418         }
419
420         public void ComputeTimeElapsed(out int majBeat, out int minBeat, Note inNote)
421         {
422                 majBeat = (int) System.Math.Floor((double) inNote.getTotalLength() / (double) timeSig.
423                 minBeat = (int) ((((double)inNote.getTotalLength() / (double) timeSig.Denominator) - (
424         }
425 }
426
```

```
427 public class TimeSignature
428 {
429         protected int numerator;
430         protected NoteDurationEnum denominator;
431         public TimeSignature(int numer, NoteDurationEnum denom)
432         {
433                 numerator = numer;
434                 denominator = denom;
435         }
436
437         public int Numerator
438         {
439                 get
440                 {
441                         return numerator;
442                 }
443         }
444         public NoteDurationEnum Denominator
445         {
446                 get
447                 {
448                         return denominator;
449                 }
450         }
451
452         public override bool Equals(Object obj)
453         {
454                 if (obj == null || this.GetType() != obj.GetType()) return false;
455                 TimeSignature t = (TimeSignature) obj;
456                 return (this.denominator == t.denominator) && (this.numerator == t.numerator);
457         }
458
459         public static bool Equals(TimeSignature t1, TimeSignature t2)
460         {
461                 Object obj1, obj2;
462                 obj1 = t1;
463                 obj2 = t2;
464                 if(obj1 == null && obj2 == null) return true;
465                 if(obj1 == null) return false;
466                 if(obj2 == null) return false;
467                 return (t1.numerator == t2.numerator) && (t1.denominator == t2.denominator);
468         }
469
470         public static bool operator==(TimeSignature t1, TimeSignature t2)
471         {
472                 return Equals(t1,t2);
473         }
474         public static bool operator!=(TimeSignature t1, TimeSignature t2)
475         {
476                 return !Equals(t1,t2);
477         }
478
479         public override int GetHashCode()
480         {
481                 return numerator ^ (int) denominator;
482         }
483
484
485 }
486
487 */
```

```
 1 /*
 2  * Created on Jan 12, 2004
 3  *
 4  * To change the template for this generated file go to
 5  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 6  */
 7 package com.aaron.MidiReader;
 8
 9 /**
10  * @author aaron
11  *
12  * To change the template for this generated type comment go to
13  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
14  */
15 public final class ATKeySig {
16         public static final int UnKnown = -99;
17         public static final int CMajor = 0;
18         public static final int AMinor = 0;
19         public static final int FMajor = -1;
20         public static final int DMinor = -1;
21         public static final int BbMajor = -2;
22         public static final int GMinor = -2;
23         public static final int EbMajor = -3;
24         public static final int CMinor = -3;
25         public static final int AbMajor = -4;
26         public static final int FMinor = -4;
27         public static final int DbMajor = -5;
28         public static final int BbMinor = -5;
29         public static final int GbMajor = -6;
30         public static final int EbMinor = -6;
31         public static final int CbMajor = -7;
32         public static final int AbMinor = -7;
33         public static final int FSharpMajor = 6;
34         public static final int FDSharpMinor = 6;
35         public static final int BMajor = 5;
36         public static final int GSharpMinor = 5;
37         public static final int EMajor = 4;
38         public static final int CSharpMinor = 4;
39         public static final int AMajor = 3;
40         public static final int FSharpMinor = 3;
41         public static final int DMajor = 2;
42         public static final int BMinor = 2;
43         public static final int GMajor = 1;
44         public static final int EMinor = 1;
45
46 }
```

```
 1 /*
 2  * Created on Apr 11, 2004
 3  *
 4  * To change the template for this generated file go to
 5  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 6  */
 7 package com.aaron.MidiReader;
 8
 9 /**
10  * @author aaron
11  *
12  * To change the template for this generated type comment go to
13  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
14  */
15 public class ATBeatStruct {
16         public int measure = 0;
17         public int majorBeat = 0;
18         public int minorBeat = 0;
19         public int length;
20         public boolean doIAdd;
21
22 }
```

```
 1 /*
 2  * Created on Feb 2, 2004
 3  *
 4  * To change the template for this generated file go to
 5  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 6  */
 7 package com.aaron.StaffGraphic;
 8
 9 import java.awt.Graphics2D;
10 import java.util.Map;
11
12 /**
13  * @author SoundBooth
14  *
15  * To change the template for this generated type comment go to
16  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
17  */
18 public class ATDrawCleff implements ATMusicSymbol {
19         protected int clefType;
20
21     public ATDrawCleff(int type) {
22         clefType = type;
23     }
24         /* (non-Javadoc)
25          * @see com.aaron.StaffGraphic.ATMusicSymbol#getType()
26          */
27         public String getType() {
28                 // TODO Auto-generated method stub
29                 return new String("ATDrawCleff");
30         }
31
32         /* (non-Javadoc)
33          * @see com.aaron.StaffGraphic.ATMusicSymbol#draw(java.awt.Graphics2D, double, double)
34          */
35         public Graphics2D draw(Graphics2D g, double x, double y) {
36                 // TODO Auto-generated method stub
37                 switch(clefType) {
38                         case ATDrawConstant.CLEF_TREBLE:
39                                 return MusicalSymbols.cleffTreble(g,x,y);
40                         case ATDrawConstant.CLEF_BASS:
41                                 return MusicalSymbols.cleffBass(g,x,y);
42                 }
43                 return g;
44         }
45
46         /* (non-Javadoc)
47          * @see com.aaron.StaffGraphic.ATMusicSymbol#setAttributes(java.util.Hashtable)
48          */
49         public void setAttributes(Map m) {
50                 // TODO Auto-generated method stub
51
52         }
53
54         /* (non-Javadoc)
55          * @see com.aaron.StaffGraphic.ATMusicSymbol#getAttributes()
56          */
57         public Map getAttributes() {
58                 // TODO Auto-generated method stub
59                 return null;
60         }
61
62 }
```

```
 1 /*
 2  * Created on Jan 22, 2004
 3  *
 4  * To change the template for this generated file go to
 5  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 6  */
 7 package com.aaron.StaffGraphic;
 8
 9 /**
10  * @author aaron
11  *
12  * To change the template for this generated type comment go to
13  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
14  */
15 public final class ATDrawConstant {
16         public static final String STR_TYPE = "Type";
17         public static final String STR_LENGTH = "Length";
18         public static final String STR_PITCH = "Pitch";
19         public static final String STR_DOTTED = "Dotted";
20         public static final String STR_TOTALLENGTH = "Totallength";
21         public static final String STR_MEASURENUM = "MeasureNumber";
22
23
24
25         public static final int WHOLE = 64;
26         public static final int HALF = 32;
27         public static final int QUARTER = 16;
28         public static final int EIGHTH = 8;
29         public static final int SIXTEENTH = 4;
30         public static final int THIRTYSECOND = 2;
31         public static final int SIXTYFORTH = 1;
32
33         public static final int NOTE = 1;
34         public static final int REST = 0;
35
36         public static final int CLEF_TREBLE = 0;
37         public static final int CLEF_BASS = 1;
38
39         public static final int FLAT = 1;
40         public static final int NATURAL = 2;
41         public static final int SHARP = 3;
42
43
44
45 }
```

```
 1 /*
 2  * Created on Mar 30, 2004
 3  *
 4  * To change the template for this generated file go to
 5  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 6  */
 7 package com.aaron.StaffGraphic;
 8
 9 import java.awt.Graphics2D;
10
11
12 import java.awt.geom.AffineTransform;
13 import java.awt.Rectangle;
14 import java.util.*;
15
16 /**
17  * @author SoundBooth
18  *
19  * To change the template for this generated type comment go to
20  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
21  */
22 public class ATDrawExtra implements ATMusicSymbol {
23         public static final int TYPE_LEDGERLINE = 1;
24         public static final int TYPE_BARLINE = 2;
25
26         int symbol;
27         TreeMap data;
28
29         public ATDrawExtra(int symbol) {
30                 data = new TreeMap();
31                 // default values
32                 data.put("STAFFSIZE", new Integer(0));
33                 this.symbol = symbol;
34         }
35         /* (non-Javadoc)
36          * @see com.aaron.StaffGraphic.ATMusicSymbol#getType()
37          */
38         public String getType() {
39                 // TODO Auto-generated method stub
40                 switch(symbol) {
41                         case TYPE_LEDGERLINE:
42                                 return  new String("ATDrawExtra-LedgerLine");
43                         case TYPE_BARLINE:
44                                 return  new String("ATDrawExtra-BarLine");
45                 }
46                 return  new String("ATDrawExtra");
47
48         }
49
50         /* (non-Javadoc)
51          * @see com.aaron.StaffGraphic.ATMusicSymbol#draw(java.awt.Graphics2D, double, double)
52          */
53         public Graphics2D draw(Graphics2D g, double x, double y) {
54                 // TODO Auto-generated method stub
55                 switch(symbol) {
56                         case TYPE_LEDGERLINE:
57                                 return ledgerLine(g,x,y);
58                         case TYPE_BARLINE:
59                                 double staffSize = ((Double) data.get("STAFFSIZE")).doubleValue();
60                                 return barLine(g,x,y,staffSize);
61
62                 }
63                 return g;
64         }
65
66         /* (non-Javadoc)
67          * @see com.aaron.StaffGraphic.ATMusicSymbol#setAttributes(java.util.Hashtable)
68          */
69          /**
70           * Attribute:  MEASURENUMBER    -- the number of the measure after the barline
71           */
```

```
 72          public void setAttributes(Map m) {
 73                  // TODO Auto-generated method stub
 74                  Iterator keys = m.keySet().iterator();
 75                  while(keys.hasNext()) {
 76                          Object o = keys.next();
 77                          data.put(o,m.get(o));
 78                  }
 79
 80
 81          }
 82
 83          /* (non-Javadoc)
 84           * @see com.aaron.StaffGraphic.ATMusicSymbol#getAttributes()
 85           */
 86          public Map getAttributes() {
 87                  // TODO Auto-generated method stub
 88                  return data;
 89          }
 90
 91
 92          private Graphics2D ledgerLine(Graphics2D g, double x, double y) {
 93                  AffineTransform oldAT = g.getTransform();
 94                  g.translate(x, y + .5);
 95                  //g.scale(1.4,1.4);
 96
 97                  Rectangle r = new Rectangle();
 98                  r.setRect(-10,0, 20, .05);
 99                  g.fill(r);
100                  g.setTransform(oldAT);
101                  return g;
102          }
103          private Graphics2D barLine(Graphics2D g, double x, double staffTop, double staffSize) {
104                  AffineTransform oldAT = g.getTransform();
105                  g.translate(x, staffTop);
106                  //g.scale(1.4,1.4);
107
108                  Rectangle r = new Rectangle();
109                  r.setRect(0,staffSize, .5, staffSize * 4);
110                  g.fill(r);
111                  g.setTransform(oldAT);
112                  return g;
113          }
114
115
116  }
```

```
 1 package com.aaron.StaffGraphic;
 2 import java.awt.Graphics2D;
 3 import java.util.*;
 4
 5 import java.awt.geom.AffineTransform;
 6
 7 import java.awt.geom.Arc2D;
 8
 9
10 /*
11  * Created on Jan 22, 2004
12  *
13  * To change the template for this generated file go to
14  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
15  */
16
17 /**
18  * @author aaron
19  *
20  * To change the template for this generated type comment go to
21  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
22  */
23 public class ATDrawNote implements ATMusicSymbol {
24
25         protected int length;
26         protected int type;
27         protected int dotted;
28         protected int pitch;
29         protected ATDrawNote tie;
30         protected boolean isTiedTo;
31         protected ATDrawNote beam;
32         protected boolean isTriplet;
33         protected boolean isDuplet;
34
35
36         public ATDrawNote() {
37                 Map m = new TreeMap();
38                 m.put(ATDrawConstant.STR_TYPE, new Integer(ATDrawConstant.NOTE));
39                 m.put(ATDrawConstant.STR_LENGTH, new Integer(ATDrawConstant.QUARTER));
40                 m.put(ATDrawConstant.STR_PITCH, new Integer(60));
41                 m.put(ATDrawConstant.STR_DOTTED, new Integer(0));
42
43                 setAttributes(m);
44
45         }
46
47         public ATDrawNote(int length, int type, int dotted) {
48                 Hashtable h = new Hashtable();
49                 h.put(ATDrawConstant.STR_TYPE, new Integer(type));
50                 h.put(ATDrawConstant.STR_LENGTH, new Integer(length));
51                 h.put(ATDrawConstant.STR_PITCH, new Integer(pitch));
52                 h.put(ATDrawConstant.STR_DOTTED, new Integer(dotted));
53                 setAttributes(h);
54
55         }
56
57         public ATDrawNote(Hashtable h) {
58                 setAttributes(h);
59         }
60
61         /* (non-Javadoc)
62          * @see ATMusicSymbol#getType()
63          */
64         public String getType() {
65                 return new String("ATDrawNote");
66         }
67
68         /* (non-Javadoc)
69          * @see ATMusicSymbol#draw(java.awt.Graphics2D)
70          */
71         public Graphics2D draw(Graphics2D g, double x, double y) {
```

```
72                  // TODO Auto-generated method stub
73                  switch(length) {
74                          case ATDrawConstant.THIRTYSECOND:
75                                  g = MusicalSymbols.noteThirtysecond(g, x, y);
76                                  break;
77                          case ATDrawConstant.SIXTEENTH:
78                              g = MusicalSymbols.noteSixteenth(g, x, y);
79                              break;
80                          case ATDrawConstant.EIGHTH:
81                                  g =  MusicalSymbols.noteEighth(g, x, y);
82                                  break;
83                          case ATDrawConstant.QUARTER:
84                                  g =  MusicalSymbols.noteQuarter(g, x, y);
85                                  break;
86                          case ATDrawConstant.HALF:
87                                  g =  MusicalSymbols.noteHalf(g, x, y);
88                                  break;
89                          case ATDrawConstant.WHOLE:
90                                  g =  MusicalSymbols.noteWhole(g, x, y);
91                                  break;
92
93                  }
94                  if( dotted > 0) {
95                          g = drawDot(g, x, y);
96
97                  }
98                  return g;
99          }
100
101         /* (non-Javadoc)
102          * @see ATMusicSymbol#setAttributes(java.util.Hashtable)
103          */
104         public void setAttributes(Map m) {
105                 // TODO Auto-generated method stub
106                 Iterator keys = m.keySet().iterator();
107                 while(keys.hasNext()) {
108                         String key = (String) keys.next();
109                         if(key.equals(ATDrawConstant.STR_TYPE)){
110                                 Integer t = (Integer) m.get(key);
111                                 type = t.intValue();
112                         }
113                         if(key.equals(ATDrawConstant.STR_LENGTH)){
114                                 Integer l = (Integer) m.get(key);
115                                 length = l.intValue();
116                         }
117                         if(key.equals(ATDrawConstant.STR_PITCH)){
118                                 Integer p = (Integer) m.get(key);
119                                 pitch = p.intValue();
120                         }
121                         if(key.equals(ATDrawConstant.STR_DOTTED)) {
122                                 Integer d = (Integer) m.get(key);
123                                 dotted = d.intValue();
124                         }
125                 }
126
127         }
128
129         /* (non-Javadoc)
130          * @see ATMusicSymbol#getAttributes()
131          */
132         public Map getAttributes() {
133                 TreeMap m = new TreeMap();
134                 m.put(ATDrawConstant.STR_TYPE, new Integer(type));
135                 m.put(ATDrawConstant.STR_LENGTH, new Integer(length));
136                 m.put(ATDrawConstant.STR_PITCH, new Integer(pitch));
137                 m.put(ATDrawConstant.STR_DOTTED, new Integer(dotted));
138                 m.put(ATDrawConstant.STR_TOTALLENGTH, new Integer(getTotalLength()));
139
140                 return m;
141         }
142
```

```
143         private Graphics2D drawDot(Graphics2D g, double x, double y) {
144                 AffineTransform oldAT = g.getTransform();
145                 g.translate(x + 9, y + 2.5);
146                 //g.scale(1.4,1.4);
147
148                 Arc2D.Double d = new Arc2D.Double(Arc2D.CHORD);
149                 d.setArcByCenter(0, 0, 1.5 ,0 ,360 ,Arc2D.CHORD);
150                 g.fill(d);
151                 g.setTransform(oldAT);
152                 return g;
153         }
154
155         protected int getTotalLength() {
156                 int retval = 0;
157                 retval += length;
158                 int tdot = dotted;
159                 int tlength = length;
160                 while (tdot > 0) {
161                         tlength /= 2;
162                         retval += tlength;
163                         tdot--;
164                 }
165                 if (tie != null) {
166                         retval += tie.getTotalLength();
167                 }
168
169                 return retval;
170         }
171
172
173 }
```

```
 1 /*
 2  * Created on Feb 5, 2004
 3  *
 4  * To change the template for this generated file go to
 5  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 6  */
 7 package com.aaron.StaffGraphic;
 8
 9 /**
10  * @author SoundBooth
11  *
12  * To change the template for this generated type comment go to
13  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
14  */
15 public class ATDrawNoteAdjust {
16
17         double lSpace = 0;
18         double middleC;
19         int accidental = 0;
20
21         public ATDrawNoteAdjust(double lineSpace) {
22                 lSpace = lineSpace;
23                 //middleCcoord = middleC;
24         }
25
26         public double adjustYToNearestNote(double y){
27
28                 double noteScale = lSpace / 2;
29
30                 long mult = Math.round(y / noteScale);
31                 return noteScale * mult;
32         }
33
34         public int getNearestNote(double y){
35
36                 return 0;
37         }
38
39         public int getAccidental() {
40                 return accidental;
41         }
42         public ATDrawPVertInfo getYfromPitch(double MiddleC, int pitch) {
43                 ATDrawPVertInfo retval = new ATDrawPVertInfo();
44                 final int MIDDLECPITCH = 60;
45                 accidental = 0;
46
47                 //look up array to get ofset based on pitch.  Enharmonics are predetermined
48                 //.3 = flat, .7 = sharp
49                 final double [] pitchOffset = {0, 1.3, 1, 2.3, 2, 3, 3.7, 4, 5.3, 5, 6.3, 6, 7};
50
51                 int octave = (int) Math.floor((double) pitch / 12) - 5;
52
53                 int firstOctave = pitch % 12;
54                 double iOffset = pitchOffset[firstOctave];
55                 int offset = (int) Math.floor(iOffset);
56                 double fractPart = iOffset - offset;
57                 if(fractPart == .3) {
58                         accidental = ATDrawConstant.FLAT;
59                 }
60                 if(fractPart == .7){
61                         accidental = ATDrawConstant.SHARP;
62                 }
63
64
65                 retval.yCoord = MiddleC  + (offset * lSpace/2) + (octave * 7 * lSpace/2);
66
67                 double topSpace = MiddleC + (11 * lSpace/2);
68                 double bottomSpace = MiddleC + lSpace/2;
69                 while(retval.yCoord < bottomSpace) {
70                         retval.ledgerLines--;
71                         bottomSpace -= lSpace;
```

```
72                  }
73
74                  while(retval.yCoord > topSpace) {
75                          retval.ledgerLines++;
76                          topSpace += lSpace;
77                  }
78
79                  return retval;
80
81
82          }
83
84          public int getPitchFromY(double MiddleC, double y) {
85                  final int MIDDLECPITCH = 60;
86                  final int [] pitchAdjust = {0, 2, 4, 5, 7, 9, 11};
87                  int octave = 0;
88
89                  double normY = adjustYToNearestNote(y);
90                  double diff = normY - MiddleC;
91                  int spaceDiff = (int)Math.floor(diff / (lSpace / 2));
92                  if(spaceDiff > 0) {
93                          while(spaceDiff >= 7) {
94                                  spaceDiff -= 7;
95                                  octave++;
96                          }
97                  }
98                  else {
99                          while ( spaceDiff <= -7) {
100                                 spaceDiff += 7;
101                                 octave--;
102                         }
103                 }
104
105                 int sign = spaceDiff > 0 ? 1 : -1;
106
107                 return MIDDLECPITCH + 12*octave + sign * pitchAdjust[Math.abs(spaceDiff)];
108
109
110         }
111 }
```

```
 1 /*
 2  * Created on Mar 30, 2004
 3  *
 4  * To change the template for this generated file go to
 5  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 6  */
 7 package com.aaron.StaffGraphic;
 8
 9 /**
10  * @author SoundBooth
11  *
12  * To change the template for this generated type comment go to
13  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
14  */
15 public class ATDrawPVertInfo {
16
17         public double yCoord = 0;
18         /** negative ledgerLines for below staff, positive for above */
19         public int ledgerLines = 0;
20         public int accidental = ATDrawConstant.NATURAL;
21
22
23
24 }
```

```
 1 /*
 2  * Created on Jan 17, 2004
 3  *
 4  * To change the template for this generated file go to
 5  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 6  */
 7 package com.aaron.StaffGraphic;
 8
 9 import java.awt.Graphics2D;
10 import java.awt.geom.AffineTransform;
11 import java.awt.geom.Point2D;
12
13
14
15 import java.awt.Rectangle;
16 import java.util.*;
17
18 import com.aaron.MidiReader.*;
19
20 /**
21  * @author aaron
22  *
23  * This class will just get objects added to and draw a staff with all the objects
24  * The objects will be added to staff by giving there X-Y coords.  The staff class will
25  * use the X-Y coords to add the object, but will also convert these into midiReader form.
26  * The class must be able to return a list of events in MidiReader package format.
27  *
28  * To determine where to place notes, the class will use an underlying grid.  Notes
29  * will be placed on the staff based on the closest grid points.
30  *
31  */
32 public class ATDrawStaff {
33
34         private final int SPACE_SIZE = 10;
35         private double MIDDLEC_YCOORD = 50;
36         private double STAFFBOTTOM_COORD = 59;
37         private double TOPA_YCOORD;
38
39         private class DrawListItem {
40                 public double x;
41                 public double y;
42                 public ATMusicSymbol mSymbol;
43                 public ATBeatStruct beat;
44         }
45
46         protected boolean isVisible;
47
48         protected Vector nList;
49         protected double topCoord;
50         protected double leftCoord;
51         protected double scaleX;
52         protected double scaleY;
53
54         protected ATDrawNoteAdjust Yadjust;
55         protected ATDrawTimeAdjust Xadjust;
56
57         protected int indexOfTimeSig = -1;
58
59         public ATDrawStaff(double left, double top) {
60                 isVisible = true;
61
62                 Yadjust = new ATDrawNoteAdjust(SPACE_SIZE);
63                 Xadjust = new ATDrawTimeAdjust(60);
64
65                 scaleX = 1;
66                 scaleY = 1;
67                 topCoord = top;
68                 leftCoord = left;
69                 MIDDLEC_YCOORD = -SPACE_SIZE * 5;
70                 TOPA_YCOORD = SPACE_SIZE;
71
```

```
72                nList = new Vector();
73
74                Hashtable h = new Hashtable();
75                h.put(ATDrawConstant.STR_TYPE, new Integer(ATDrawConstant.NOTE));
76                h.put(ATDrawConstant.STR_LENGTH, new Integer(ATDrawConstant.QUARTER));
77                h.put(ATDrawConstant.STR_PITCH, new Integer(60));
78                //nList.add(new ATDrawNote(h));
79
80                h.put(ATDrawConstant.STR_LENGTH, new Integer(ATDrawConstant.HALF));
81                //nList.add(new ATDrawNote(h));
82
83
84                h.put(ATDrawConstant.STR_LENGTH, new Integer(ATDrawConstant.WHOLE));
85                //nList.add(new ATDrawNote(h));
86
87                addStaffHeaders();
88
89        }
90
91        private void addStaffHeaders() {
92                //this.add(new ATDrawCleff(ATDrawConstant.CLEF_TREBLE), leftCoord + 15, -topCoord + 38
93
94                // The cleff must always be first and the time signature second
95
96                this.addRelative(new ATDrawCleff(ATDrawConstant.CLEF_TREBLE), null, 25, STAFFBOTTOM_CO
97                //this.add(new ATDrawCleff(ATDrawConstant.CLEF_TREBLE), 10, 50);
98
99                this.addRelative(new ATDrawTimeSig(4,4), null, 50, STAFFBOTTOM_COORD);
100               indexOfTimeSig = nList.size() - 1;
101
102       }
103
104       public double getLeft() {return leftCoord; }
105       public double getTop() {return topCoord;}
106       public void setLeft(double left) {leftCoord = left; }
107       public void setTop(double top) {topCoord = top; }
108       public void scale(double sx, double sy) {
109               scaleX = sx;
110               scaleY = sy;
111       }
112       public void isVisible(boolean b) {
113               isVisible = b;
114       }
115       public int getPitchFromY(double y) {
116               double t = toStaffY(y);
117               return Yadjust.getPitchFromY(MIDDLEC_YCOORD, t);
118       }
119
120       public ATBeatStruct getBeatFromX(double x) {
121               x = toStaffX(x);
122               DrawListItem d1 = null;
123               DrawListItem d2 = null;
124               boolean foundEnd = false;
125               boolean foundBegin = false;
126               int end = 0;
127               int begin = 0;
128               int i = 1;
129
130               String d1Type = null;
131               String d2Type = null;
132
133               //d1 = (DrawListItem) nList.get(i - 1);
134               //d1Type = d1.mSymbol.getType();
135
136
137           while(!foundEnd && i < nList.size() - 1) {
138                       d1 = (DrawListItem) nList.get(++i);
139                       d1Type = d1.mSymbol.getType();
140
141                       while(!(d1Type.equals("ATDrawNote") || (d1Type.equals("ATDrawExtra-BarLine")))
142                               d1 = (DrawListItem) nList.get(++i);
```

```
143                             d1Type = d1.mSymbol.getType();
144                     }
145                     if(d1.x > x) {
146                             foundEnd = true;
147                             end = i;
148                     }
149             }
150             i++;
151             while(!foundBegin && i > 0) {
152                     d2 = (DrawListItem) nList.get(--i);
153                     d2Type = d2.mSymbol.getType();
154                     while(!(d2Type.equals("ATDrawNote") || (d2Type.equals("ATDrawExtra-BarLine")))
155                             d2 = (DrawListItem) nList.get(--i);
156                             d2Type = d2.mSymbol.getType();
157                     }
158                     if(d2.x < x && i != 0) {
159                             foundBegin = true;
160                             begin = i;
161                     }
162             }
163
164
165             if(foundBegin && foundEnd) {
166                     if((d1Type.equals("ATDrawNote") && d2Type.equals("ATDrawNote")) ||
167                             (d1Type.equals("ATDrawExtra-BarLine") && d2Type.equals("ATDrawNote")))
168
169                             ATBeatStruct retval = new ATBeatStruct();
170                             retval.majorBeat = d2.beat.majorBeat;
171                             retval.minorBeat = d2.beat.minorBeat;
172                             retval.measure = d2.beat.measure;
173
174                             Map m = d2.mSymbol.getAttributes();
175                             retval.length = ((Integer)m.get(ATDrawConstant.STR_TOTALLENGTH)).intVa
176
177                             retval.doIAdd = true;
178                             if(d1Type.equals("ATDrawNote") && d2Type.equals("ATDrawNote")){
179                                     retval.doIAdd = false;
180                             }
181                             return retval;
182                     }
183                     if(d1Type.equals("ATDrawNote") && d2Type.equals("ATDrawExtra-BarLine")) {
184                             ATBeatStruct retval = new ATBeatStruct();
185                             retval.majorBeat = 0;
186                             retval.minorBeat = 0;
187                             retval.measure = d1.beat.measure;
188
189                             Map m = d1.mSymbol.getAttributes();
190                             retval.length = ((Integer)m.get(ATDrawConstant.STR_TOTALLENGTH)).intVa
191                             retval.doIAdd = false;
192                             return retval;
193                     }
194             }
195
196             if(foundBegin) {
197                     // too simple-- always starts first note of new measure as beat 1
198                     ATBeatStruct retval = new ATBeatStruct();
199                     retval.majorBeat = 1;
200                     retval.minorBeat = 0;
201                     Map m = d2.mSymbol.getAttributes();
202                     retval.measure = ((Integer)m.get(ATDrawConstant.STR_MEASURENUM)).intValue() +
203                     retval.length = 0;
204                     retval.doIAdd = true;
205                     return retval;
206
207             }
208
209             if(foundEnd) {
210                     //insert note
211                     ATBeatStruct retval = new ATBeatStruct();
212                     retval.majorBeat = 1;
213                     retval.minorBeat = 0;
```

```
214                     retval.measure = 1;
215                     retval.doIAdd = false;
216                     retval.length = 0;
217                     return retval;
218
219
220             }
221
222             return null;
223             /*
224
225             while(!(d1Type.equals("ATDrawNote") && (d1Type.equals("ATDrawExtra-BarLine"))) && i <
226                     d1 = (DrawListItem) nList.get(++i - 1);
227             }
228
229             while(!found && i < nList.size() - 1) {
230                     d2 = (DrawListItem) nList.get(++i);
231                     while(!d2.mSymbol.getType().equals("ATDrawNote") && i < nList.size() - 1) {
232                             d2 = (DrawListItem) nList.get(++i);
233                     }
234                     double list1X = d1.x;
235                     double list2X = d2.x;
236                     if(list1X < x && list2X > x) {
237                             found = true;
238                     }
239                     else {
240                             d1 = d2;
241                     }
242
243             }
244             if(found) {
245                     ATBeatStruct retval = new ATBeatStruct();
246                     retval.majorBeat = d1.beat.majorBeat;
247                     retval.minorBeat = d1.beat.minorBeat;
248                     retval.measure = d1.beat.measure;
249
250                     Map m = d1.mSymbol.getAttributes();
251                     retval.length = ((Integer)m.get(ATDrawConstant.STR_TOTALLENGTH)).intValue();
252             } else {
253
254             }
255
256             return retval;
257             */
258
259     }
260
261
262
263     //public Graphics2D drawBlankStaff(Graphics2D g, double x, double y) {
264     //      g = drawFiveLineStaff(g,x,y,100);
265     //      return g;
266     //}
267
268     public Graphics2D drawFiveLineStaff(Graphics2D g, double len) {
269             AffineTransform oldAT = g.getTransform();
270
271             for(int i = 0; i < 5; i++) {
272                     Rectangle r = new Rectangle();
273                     r.setRect(0,0, len, .01);
274                     g.fill(r);
275                     g.translate(0, -SPACE_SIZE);
276
277             }
278             g.setTransform(oldAT);
279             return g;
280
281     }
282
283     public Graphics2D drawBarLineStaff(Graphics2D g, double x) {
284             AffineTransform oldAT = g.getTransform();
```

```
285                 double top = toStaffYNoShift(topCoord);
286                 Rectangle r = new Rectangle();
287                 r.setRect(x,top, .01, -SPACE_SIZE * 5);
288
289                 g.setTransform(oldAT);
290                 return g;
291
292
293         }
294
295         public Graphics2D draw(Graphics2D g) {
296                 if(isVisible == false) return g;
297
298                 AffineTransform oldAT = g.getTransform();
299                 g.translate(leftCoord, topCoord);
300                 g.scale(scaleX, scaleY);
301
302                 ATDrawNote n = new ATDrawNote();
303                 g = drawFiveLineStaff(g, 2000);
304
305                 Enumeration e = nList.elements();
306
307
308
309                 //int shift = 10;
310                 while(e.hasMoreElements()) {
311                         DrawListItem d;
312                         d = (DrawListItem) e.nextElement();
313                         ATMusicSymbol s = d.mSymbol;
314                         s.draw(g,(d.x), (d.y));
315                         //shift += 10;
316                 }
317                 g.setTransform(oldAT);
318                 return g;
319         }
320
321         public void clear() {
322                 nList.clear();
323                 addStaffHeaders();
324         }
325
326         public void add(ATMusicSymbol m, ATBeatStruct b, double currentX, double currentY) {
327                 DrawListItem d = new DrawListItem();
328                 d.beat = null;
329                 d.x = toStaffX(currentX);
330                 d.y = toStaffY(currentY);
331                 d.y = Yadjust.adjustYToNearestNote(d.y);
332                 d.mSymbol = m;
333                 if(b != null) {
334                         d.beat = new ATBeatStruct();
335                         d.beat.majorBeat = b.majorBeat;
336                         d.beat.minorBeat = b.minorBeat;
337                         d.beat.measure = b.measure;
338                 }
339                 nList.add(d);
340
341         }
342
343         protected void addRelative(ATMusicSymbol m, ATBeatStruct b, double currentX, double currentY)
344                 DrawListItem d = new DrawListItem();
345                 d.beat = null;
346                 d.x = toStaffXNoShift(currentX);
347                 d.y = toStaffYNoShift(currentY);
348                 //d.y = Yadjust.adjustYToNearestNote(d.y);
349                 d.mSymbol = m;
350                 if(b != null) {
351                         d.beat = new ATBeatStruct();
352                         d.beat.majorBeat = b.majorBeat;
353                         d.beat.minorBeat = b.minorBeat;
354                         d.beat.measure = b.measure;
355                 }
```

```
356                 nList.add(d);
357
358         }
359
360         public void add(ATStaff staff) {
361
362                 final int NOTE_SPACE = 25;
363                 int tcount = 60;
364                 int t = -50;
365                 int meas = -1;
366
367                 //Time Signature
368                 ATTimeSignature time = staff.getTimeSignature();
369                 changeTimeSig(time.getNumerator(), ATNoteDuration.convertToTimeSigDenominator(time.get
370
371
372                 Iterator measures = staff.iterator();
373                 double currentX = leftCoord + 60;
374                 while(measures.hasNext()) {
375                         ATMeasure m = (ATMeasure) measures.next();
376                         meas++;
377                         Iterator notes = m.iterator();
378
379                         //No bar line the first time
380                         if(notes.hasNext() == false) continue;
381
382                         while(notes.hasNext()) {
383                                 //add notes
384                                 ATNoteBeat nb = (ATNoteBeat) notes.next();
385                                 ATNote n = nb.getNote();
386                                 ATMusicSymbol ms = new ATDrawNote(n.getLength(), n.getType(), n.getDot
387                                 DrawListItem d = new DrawListItem();
388                                 d.beat = new ATBeatStruct();
389                                 tcount += NOTE_SPACE;
390                                 ATDrawPVertInfo yInfo = Yadjust.getYfromPitch(MIDDLEC_YCOORD, n.getPit
391                                 d.mSymbol = ms;
392                                 //add(ms, currentX, currentY);
393                                 d.x = currentX;
394                                 currentX += Xadjust.computeDurationSpacing(n.getLength()) + 5;
395
396                                 d.y = Yadjust.adjustYToNearestNote(yInfo.yCoord );
397                                 d.beat.majorBeat = nb.getMajorBeat();
398                                 d.beat.minorBeat = nb.getMinorBeat();
399                                 d.beat.measure = meas;
400                                 //d.y = t;
401                                 //t += 10;
402                                 nList.add(d);
403
404                                 //add ledger lines
405                                 if(Math.abs(yInfo.ledgerLines) > 0) {
406                                         if(yInfo.ledgerLines < 0){
407                                                 int lines = Math.abs(yInfo.ledgerLines);
408                                                 double yLineCoord = MIDDLEC_YCOORD;
409                                                 ATMusicSymbol ms1 = new ATDrawExtra(ATDrawExtra.TYPE_L
410                                                 while(lines > 0) {
411                                                         DrawListItem d1 = new DrawListItem();
412                                                         d1.beat = null;
413                                                         d1.x = d.x;
414                                                         d1.y = yLineCoord;
415                                                         d1.mSymbol = ms1;
416                                                         nList.add(d1);
417                                                         yLineCoord = yLineCoord - SPACE_SIZE;
418                                                         lines--;
419                                                 }
420                                         }
421                                         if(yInfo.ledgerLines > 0) {
422                                                 int lines = Math.abs(yInfo.ledgerLines);
423                                                 double yLineCoord = TOPA_YCOORD;
424                                                 ATMusicSymbol ms1 = new ATDrawExtra(ATDrawExtra.TYPE_L
425                                                 while(lines > 0) {
426                                                         DrawListItem d1 = new DrawListItem();
```

```
427                                                     d1.beat = null;
428                                                     d1.x = d.x;
429                                                     d1.y = yLineCoord;
430                                                     d1.mSymbol = ms1;
431                                                     nList.add(d1);
432                                                     yLineCoord = yLineCoord + SPACE_SIZE;
433                                                     lines--;
434                                             }
435
436                                     }
437
438                             }
439
440
441
442
443                     }
444                     //barline
445                     ATMusicSymbol msbl = new ATDrawExtra(ATDrawExtra.TYPE_BARLINE);
446                     TreeMap attrib = new TreeMap();
447                     attrib.put("STAFFSIZE", new Double((double)SPACE_SIZE));
448                     attrib.put(ATDrawConstant.STR_MEASURENUM, new Integer((meas)));
449                     msbl.setAttributes(attrib);
450                     DrawListItem pd = new DrawListItem();
451                     pd.beat = null;
452
453                     pd.mSymbol = msbl;
454                     pd.x = currentX;
455                     pd.y = toStaffYNoShift(STAFFBOTTOM_COORD + 15);
456                     nList.add(pd);
457                     currentX += 15;
458
459             }
460     }
461
462     protected double toStaffX(double currentX) {
463             AffineTransform at = new AffineTransform(scaleX, 0, 0, -scaleY, leftCoord, -topCoord);
464             try {
465                     at = at.createInverse();
466             } catch(Exception e) {}
467
468         Point2D p = at.transform(new Point2D.Double(currentX, 0), null);
469         return p.getX();
470
471             //return (currentX - (leftCoord / scaleX)) / scaleX;
472     }
473     protected double toStaffY(double currentY) {
474             AffineTransform at = new AffineTransform(scaleX, 0, 0, -scaleY, leftCoord, -topCoord);
475             try {
476                     at = at.createInverse();
477             } catch(Exception e) {}
478
479             Point2D p = at.transform(new Point2D.Double(0, currentY), null);
480             return p.getY();
481
482             //return (-currentY - (topCoord / scaleY)) / scaleY;
483     }
484
485     protected double toStaffXNoShift(double currentX) {
486             AffineTransform at = new AffineTransform(scaleX, 0, 0, -scaleY, 0, 0);
487             try {
488                     at = at.createInverse();
489             } catch(Exception e) {}
490
491             Point2D p = at.transform(new Point2D.Double(currentX, 0), null);
492             return p.getX();
493
494             //return (currentX - (leftCoord / scaleX)) / scaleX;
495     }
496     protected double toStaffYNoShift(double currentY) {
497             AffineTransform at = new AffineTransform(scaleX, 0, 0, -scaleY, 0, 0);
```

```
498                 try {
499                         at = at.createInverse();
500                 } catch(Exception e) {}
501
502                 Point2D p = at.transform(new Point2D.Double(0, currentY), null);
503                 return p.getY();
504
505                 //return (-currentY - (topCoord / scaleY)) / scaleY;
506         }
507
508         protected double toCurrentX(double staffX) {
509                 return (staffX + leftCoord);// * (1/scaleX);
510         }
511         protected double toCurrentY(double staffY) {
512                 return (staffY - topCoord);// * (1/scaleY)) ;
513         }
514
515         private void changeTimeSig(int num, int den) {
516                 nList.remove(indexOfTimeSig);
517                 DrawListItem d = new DrawListItem();
518                 d.beat = new ATBeatStruct();
519                 d.x = toStaffXNoShift(50);
520                 d.y = toStaffYNoShift(STAFFBOTTOM_COORD);
521                 //d.y = Yadjust.adjustYToNearestNote(d.y);
522                 d.mSymbol = new ATDrawTimeSig(num, den);
523                 nList.add(indexOfTimeSig, d);
524         }
525
526 }
```

```
 1 /*
 2  * Created on Feb 11, 2004
 3  *
 4  * To change the template for this generated file go to
 5  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 6  */
 7 package com.aaron.StaffGraphic;
 8
 9 import com.aaron.MidiReader.*;
10
11 /**
12  * @author SoundBooth
13  *
14  * To change the template for this generated type comment go to
15  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
16  */
17 public class ATDrawTimeAdjust {
18         int scale = 1;
19
20         public ATDrawTimeAdjust(int scale) {
21                 this.scale = scale;
22         }
23
24         public double adjustXtoGrid(double x, double grid) {
25
26                 long mult = Math.round(x / grid);
27                 return grid * mult;
28
29         }
30
31         public double computeDurationSpacing(int noteLen) {
32                 double normLen = (double) noteLen / ATNoteDuration.WHOLE;
33                 return scale * Math.pow(.7,Math.log(1 / normLen));
34
35         }
36 }
```

```
 1 /*
 2  * Created on Mar 31, 2004
 3  *
 4  * To change the template for this generated file go to
 5  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 6  */
 7 package com.aaron.StaffGraphic;
 8
 9 import java.awt.Graphics2D;
10 import java.util.Map;
11 import java.awt.geom.AffineTransform;
12 import java.awt.Font;
13
14
15 /**
16  * @author SoundBooth
17  *
18  * To change the template for this generated type comment go to
19  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
20  */
21 public class ATDrawTimeSig implements ATMusicSymbol {
22         int numerator;
23         int denominator;
24
25         public ATDrawTimeSig(int num, int den){
26                 numerator = num;
27                 denominator = den;
28         }
29         /* (non-Javadoc)
30          * @see com.aaron.StaffGraphic.ATMusicSymbol#getType()
31          */
32         public String getType() {
33                 // TODO Auto-generated method stub
34                 return new String("ATDrawTimeSig");
35         }
36
37         /* (non-Javadoc)
38          * @see com.aaron.StaffGraphic.ATMusicSymbol#draw(java.awt.Graphics2D, double, double)
39          */
40         public Graphics2D draw(Graphics2D g, double x, double y) {
41                 // TODO Auto-generated method stub
42                 return drawTime(g,x,y);
43         }
44
45         /* (non-Javadoc)
46          * @see com.aaron.StaffGraphic.ATMusicSymbol#setAttributes(java.util.Hashtable)
47          */
48         public void setAttributes(Map h) {
49                 // TODO Auto-generated method stub
50
51         }
52
53         /* (non-Javadoc)
54          * @see com.aaron.StaffGraphic.ATMusicSymbol#getAttributes()
55          */
56         public Map getAttributes() {
57                 // TODO Auto-generated method stub
58                 return null;
59         }
60
61         private Graphics2D drawTime(Graphics2D g, double x, double y) {
62                 AffineTransform oldAT = g.getTransform();
63                 g.translate(x, y );
64                 //g.fill(MusicalSymbols.noteHead());
65
66                 g.transform(new AffineTransform(1,0,0,-1,0,0));
67                 g.setFont(new Font("Serif", Font.BOLD, 29));
68                 g.drawString(String.valueOf(denominator),0,0);
69                 g.translate(0,-20);
70                 g.drawString(String.valueOf(numerator),0,0);
71                 g.setTransform(oldAT);
```

```
72                      return g;
73              }
74
75
76  }
```

```
 1 /*
 2  * Created on Jan 20, 2004
 3  *
 4  * To change the template for this generated file go to
 5  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 6  */
 7 package com.aaron.StaffGraphic;
 8 import java.awt.Graphics2D;
 9 import java.util.Map;
10 /**
11  * @author aaron
12  *
13  * To change the template for this generated type comment go to
14  * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
15  */
16 public interface ATMusicSymbol {
17
18         public abstract String getType();
19         public abstract Graphics2D draw(Graphics2D g, double x, double y);
20         public abstract void setAttributes(Map m);
21         public abstract Map getAttributes();
22
23
24 }
```

```
 1 /*
 2  * MusicalSymbols.java
 3  *
 4  * Created on April 29, 2003, 2:13 AM
 5  */
 6 package com.aaron.StaffGraphic;
 7 import java.awt.geom.GeneralPath;
 8 import java.awt.geom.AffineTransform;
 9 import java.awt.Graphics2D;
10 import java.awt.Rectangle;
11 /**
12  *
13  * @author  aaron
14  */
15 public class MusicalSymbols {
16
17     /** Creates a new instance of MusicalSymbols */
18     protected MusicalSymbols() {
19     }
20
21
22     public static Graphics2D noteQuarter(Graphics2D g, double x, double y) {
23         return noteQuarter(g, x, y, 20, true);
24     }
25
26     public static Graphics2D noteQuarter(Graphics2D g, double x, double y, double len, boolean stemUp)
27                 AffineTransform oldAT = g.getTransform();
28                 double scaleX = oldAT.getScaleX();
29                 double scaleY = oldAT.getScaleY();
30
31                 double mat[] = new double[6];
32                 //g.translate(x, y + (.5 * Math.abs(scaleY)));
33
34                 g.translate(x, y + .5); //center at scale(1,1)
35
36                 //g.translate(x - 4, y + 5); center at scale(2,2) and scale(3,3)
37                 //g.translate(x + 6 , y - 6); center at scale(.8,.8)
38                 //g.translate(x + 40 , y - 40);  center at scale(.5,.5)
39                 oldAT.getMatrix(mat);
40                 g.scale(1.4,1.4);
41                 AffineTransform at = g.getTransform();
42                 if (stemUp == false) {
43                         at.rotate(Math.PI);
44                         g.setTransform(at);
45                 }
46                 g.fill(MusicalSymbols.noteHead());
47                 Rectangle r = new Rectangle();
48                 r.setRect(3.5, 1, .5, len);
49                 g.fill(r);
50                 g.setTransform(oldAT);
51                 return g;
52     }
53
54         public static Graphics2D noteEighth(Graphics2D g, double x, double y) {
55                 return noteEighth(g, x, y, 20, true);
56         }
57
58     public static Graphics2D noteEighth(Graphics2D g, double x, double y, double len, boolean stemUp)
59         g = noteQuarter(g, x, y, len, stemUp);
60         AffineTransform oldAT = g.getTransform();
61                 if (stemUp == true) {
62                         g.translate(x + 3.5 * 1.4, y + len * 1.4);
63                         g.scale(1.4,1.4);
64                 g.fill(MusicalSymbols.noteFlagUp());
65                 }
66             else {
67                 g.translate(x - 3.5 * 1.4, y - len * 1.4);
68                         g.scale(1.4,1.4);
69                 g.fill(MusicalSymbols.noteFlagDown());
70             }
71         g.setTransform(oldAT);
```

```
72          return g;
73      }
74
75          public static Graphics2D noteSixteenth(Graphics2D g, double x, double y) {
76                  return noteSixteenth(g, x, y, 20, true);
77          }
78
79      public static Graphics2D noteSixteenth(Graphics2D g, double x, double y, double len, boolean stemU
80          g = noteQuarter(g, x, y, len, stemUp);
81                  AffineTransform oldAT = g.getTransform();
82
83                  if (stemUp == true) {
84                          g.translate(x + 3.5 * 1.4, y + len * 1.4);
85                          g.scale(1.4,1.4);
86                          g.fill(MusicalSymbols.noteFlagUpDouble());
87                  }
88                  else {
89                          g.translate(x - 3.5 * 1.4, y - len * 1.4);
90                          g.scale(1.4,1.4);
91                          g.fill(MusicalSymbols.noteFlagDownDouble());
92                  }
93                  g.setTransform(oldAT);
94                  return g;
95
96      }
97          public static Graphics2D noteThirtysecond(Graphics2D g, double x, double y) {
98                  return noteThirtysecond(g, x, y, 20, true);
99          }
100
101     public static Graphics2D noteThirtysecond(Graphics2D g, double x, double y, double len, boolean st
102
103                 g = noteQuarter(g, x, y, len, stemUp);
104                 AffineTransform oldAT = g.getTransform();
105                 if (stemUp == true) {
106                         g.translate(x + 3.5 * 1.4, y + len * 1.4);
107                         g.scale(1.4,1.4);
108                         g.fill(MusicalSymbols.noteFlagUpDouble());
109                         g.translate(0, -9.5);
110                         g.fill(MusicalSymbols.extraFlagUp());
111
112                 }
113                 else {
114                         g.translate(x - 3.5 * 1.4, y - len * 1.4);
115                         g.scale(1.4,1.4);
116                         g.fill(MusicalSymbols.noteFlagDownDouble());
117                         g.translate(0, 9.5);
118                         g.fill(MusicalSymbols.extraFlagDown());
119                 }
120                 g.setTransform(oldAT);
121                 return g;
122
123     }
124         public static Graphics2D noteHalf(Graphics2D g, double x, double y) {
125                 return noteHalf(g, x, y, 20, true);
126         }
127
128     public static Graphics2D noteHalf(Graphics2D g, double x, double y, double len, boolean stemUp) {
129                 AffineTransform oldAT = g.getTransform();
130                 g.translate(x, y);
131                 g.scale(1.4,1.4);
132                 AffineTransform at = g.getTransform();
133                 if (stemUp == false) {
134                         at.rotate(Math.PI);
135                         g.setTransform(at);
136                 }
137                 g.fill(MusicalSymbols.noteHeadOpen());
138                 Rectangle r = new Rectangle();
139                 r.setRect(3.5, 1, .5, len);
140                 g.fill(r);
141                 g.setTransform(oldAT);
142                 return g;
```

```
143
144     }
145
146     public static Graphics2D noteWhole(Graphics2D g, double x, double y) {
147                 AffineTransform oldAT = g.getTransform();
148                 g.translate(x, y);
149                 g.scale(1.4,1.4);
150                 g.fill(MusicalSymbols.noteHeadWhole());
151                 g.setTransform(oldAT);
152                 return g;
153
154     }
155
156     public static Graphics2D cleffTreble(Graphics2D g, double x, double y) {
157                 AffineTransform oldAT = g.getTransform();
158                 g.translate(x, y);
159                 g.scale(1.4,1.4);
160                 g.fill(MusicalSymbols.trebleCleff());
161                 g.setTransform(oldAT);
162                 return g;
163
164     }
165         public static Graphics2D cleffBass(Graphics2D g, double x, double y) {
166                 AffineTransform oldAT = g.getTransform();
167                 g.translate(x, y);
168                 g.scale(1.4,1.4);
169                 g.fill(MusicalSymbols.bassCleff());
170                 g.setTransform(oldAT);
171                 return g;
172
173         }
174
175
176
177
178
179
180     public static GeneralPath trebleCleff() {
181        GeneralPath p = new GeneralPath(GeneralPath.WIND_NON_ZERO);
182
183        p.moveTo(-2.22f, 5.91f);
184        p.lineTo(-2.96f, 4.8f);
185        p.curveTo(-5.18f, 7.02f,-3.7f,12.92f,0.73f,12.92f );
186        p.curveTo(2.95f, 13.66f,7.75f,11.07f,7.75f,6.27f);
187        p.curveTo(7.75f, 1.84f,3.69f,-0.38f,0f,-0.38f );
188        p.curveTo(-4.43f, -0.38f,-8.49f,1.84f,-8.49f,8.11f);
189        p.curveTo(-8.49f,10.69f,-8.12f,13.65f,-2.58f,18.08f);
190        p.curveTo(3.7f,22.51f,3.7f,25.1f,3.7f,26.94f );
191        p.curveTo(3.7f,28.79f,3.33f,30.26f,2.59f,30.26f);
192        p.curveTo(0.74f,28.78f,-0.73f,25.09f,-0.73f,22.88f );
193        p.curveTo(-0.73f,9.22f,3.7f,6.63f,4.07f,-2.97f);
194        p.curveTo(4.07f,-6.66f,1.85f,-8.51f,-1.47f,-8.51f );
195        p.curveTo(-3.69f,-8.51f,-5.53f,-6.66f,-5.53f,-4.82f);
196        p.curveTo(-5.53f,-2.97f,-4.42f,-1.5f,-2.58f,-1.5f );
197        p.curveTo(1.11f,-1.5f,1.11f,-7.41f,-2.21f,-6.3f);
198        p.curveTo(-1.84f,-8.52f,3.33f,-7.41f,3.33f,-3.35f );
199        p.curveTo(2.59f,8.83f,-1.84f,11.05f,-1.84f,24.71f );
200        p.curveTo(-1.84f,28.77f,-0.73f,32.09f,2.59f,33.57f);
201        p.curveTo(4.81f,32.09f,6.65f,29.51f,6.28f,26.92f);
202        p.curveTo(6.28f,22.86f,3.7f,19.9f,0.37f,16.58f );
203        p.curveTo(-1.85f,14f,-5.54f,12.15f,-5.54f,6.98f);
204        p.curveTo(-5.54f,2.55f,-2.96f,0.33f,0f,0.33f);
205        p.curveTo(2.95f,0.33f,5.54f,2.18f,5.54f,5.13f);
206        p.curveTo(5.54f,8.45f,2.59f,9.56f,0.74f,9.56f );
207        p.curveTo(-1.84f,9.56f,-3.32f,7.71f,-2.21f,5.87f );
208
209        return p;
210     }
211
212     public static GeneralPath bassCleff() {
213         GeneralPath p = new GeneralPath(GeneralPath.WIND_NON_ZERO);
```

```
214
215         p.moveTo( -9.8f, 2.5f);
216         p.curveTo( -1.5f, 6.5f, 3f, 11.5f, 3f, 15.5f);
217         p.curveTo( 3f, 20f, 1f, 23f, -1.3f, 22.8f);
218         p.curveTo( -2.3f, 23f, -5.5f, 23f, -7f, 19f);
219         p.curveTo( -5.5f, 19.5f, -3.5f, 19f, -3.5f, 17.5f);
220         p.curveTo( -3.5f, 16.5f, -4f, 15.5f, -5.5f, 15.5f);
221         p.curveTo( -6.5f, 15.5f, -7.5f, 16.4f, -7.5f, 18f);
222         p.curveTo( -7.5f, 20.5f, -5.4f, 23.5f, -1.5f, 23.5f);
223         p.curveTo( 2.5f, 23.5f, 6f, 21f, 6f, 16f);
224         p.curveTo( 6f, 10.5f, -0.5f, 5f, -9.5f, 2.3f);
225         p.curveTo( -1.2f, 6.3f, 3.3f, 11.3f, 3.3f, 15.3f);
226         p.curveTo( 3.3f, 19.8f, 1.3f, 22.8f, -1f, 22.6f);
227         p.curveTo( -2f, 22.8f, -5.2f, 22.8f, -6.7f, 18.8f);
228         p.curveTo( -5.2f, 19.3f, -3.2f, 18.8f, -3.2f, 17.3f);
229         p.curveTo( -3.2f, 16.3f, -3.7f, 15.3f, -5.2f, 15.3f);
230         p.curveTo( -6.2f, 15.3f, -7.2f, 16.2f, -7.2f, 17.8f);
231         p.curveTo( -7.2f, 20.3f, -5.1f, 23.3f, -1.2f, 23.3f);
232         p.curveTo( 2.8f, 23.3f, 6.3f, 20.8f, 6.3f, 15.8f);
233         p.curveTo( 6.3f, 10.3f, -0.2f, 4.8f, -9.2f, 2.1f);
234         p.moveTo( 7.7f, 20.3f );
235         p.curveTo( 7.7f, 21.8f, 9.7f, 21.8f, 9.7f, 20.3f);
236         p.curveTo( 9.7f, 18.8f, 7.7f, 18.8f, 7.7f, 20.3f);
237         p.moveTo( 7.7f, 14.8f);
238         p.curveTo( 7.7f, 16.3f, 9.7f, 16.3f, 9.7f, 14.8f);
239         p.curveTo( 9.7f, 13.3f, 7.7f, 13.3f, 7.7f, 14.8f);
240
241         return p;
242     }
243
244     public static GeneralPath tenorCleff() {
245
246         GeneralPath p = new GeneralPath(GeneralPath.WIND_NON_ZERO);
247
248         p.moveTo( 0f, 13f);
249         p.lineTo( 1.2f, 14.75f);
250         p.curveTo( 5.4f, 14.25f, 7.2f, 17f, 7.2f, 19.75f);
251         p.curveTo( 7.2f, 21.75f, 6.6f, 23.65f, 3.9f, 23.75f);
252         p.curveTo( 3.12f, 23.75f, 1.2f, 23.75f, 0.3f, 21.75f);
253         p.curveTo( 1.2f, 22f, 2.4f, 21.75f, 2.4f, 21f);
254         p.curveTo( 2.4f, 20.5f, 2.1f, 20f, 1.2f, 20f);
255         p.curveTo( 0.6f, 20f, 0f, 20.45f, 0f, 21.25f);
256         p.curveTo( 0f, 22.5f, 1.26f, 24f, 3.6f, 24f);
257         p.curveTo( 7.2f, 24f, 9f, 22.75f, 9f, 20.25f);
258         p.curveTo( 9f, 17f, 6f, 14.25f, 2.4f, 14.5f);
259         p.lineTo( 1.8f, 13f);
260         //p.closePath();
261
262         GeneralPath p2 = (GeneralPath) p.clone();
263
264         p.transform(new AffineTransform(1, 0, 0, -1, 0, 27));
265         p2.append(p,false);
266
267         return p2;
268     }
269
270     public static GeneralPath noteHead() {
271        GeneralPath p = new GeneralPath(GeneralPath.WIND_NON_ZERO);
272        p.moveTo(3.3f, 2.26f);
273        p.curveTo( 1.04f, 5.56f, -5.56f, 1.04f, -3.3f, -2.26f);
274        p.curveTo( -1.04f, -5.56f, 5.56f, -1.04f, 3.3f, 2.26f);
275        return p;
276     }
277
278     public static GeneralPath noteHeadOpen() {
279        GeneralPath p = new GeneralPath(GeneralPath.WIND_NON_ZERO);
280
281         p.moveTo(3.51f, 1.92f);
282         p.curveTo(1.47f, 5.65f, -5.55f, 1.82f, -3.51f, -1.91f);
283         p.curveTo(-1.47f, -5.64f, 5.55f, -1.81f, 3.51f, 1.92f);
284         p.moveTo(3.07f, 1.68f);
```

```
285         p.curveTo(4.03f, -0.08f, -2.12f, -3.43f, -3.08f, -1.67f);
286         p.curveTo(-4.04f, 0.09f, 2.11f, 3.44f, 3.07f, 1.68f);
287         return p;
288     }
289
290     public static GeneralPath noteHeadWhole() {
291         GeneralPath p = new GeneralPath(GeneralPath.WIND_NON_ZERO);
292
293         p.moveTo(5.96f, 0f);
294         p.curveTo(5.96f, 1.08f, 3.25f, 3.52f, 0f, 3.52f);
295         p.curveTo(-3.25f, 3.52f, -5.96f, 1.08f, -5.96f, 0f);
296         p.curveTo(-5.96f, -1.08f, -3.25f, -3.52f, 0f, -3.52f);
297         p.curveTo(3.25f, -3.52f, 5.96f, -1.08f, 5.96f, 0f);
298         p.moveTo(-2.17f, 1.62f);
299         p.curveTo(-0.55f, 3.79f, 3.79f, 0.55f, 2.17f, -1.62f);
300         p.curveTo(0.55f, -3.79f, -3.79f, -0.55f, -2.17f, 1.62f);
301
302         return p;
303     }
304
305     public static GeneralPath noteFlagUp() {
306         GeneralPath p = new GeneralPath(GeneralPath.WIND_NON_ZERO);
307
308         p.moveTo(0f, 0f);
309         p.curveTo(1f, -2f, 0.67f, -1.67f, 2.67f, -4f);
310         p.curveTo(6f, -6.67f, 6f, -10.67f, 5.34f, -13.33f);
311         p.curveTo(4.67f, -16f, 2.67f, -17.33f, 4.34f, -14.33f);
312         p.curveTo(6.01f, -10f, 2.67f, -6f, 0.01f, -5.33f);
313
314         return p;
315     }
316     public static GeneralPath noteFlagDown() {
317         GeneralPath p = new GeneralPath(GeneralPath.WIND_NON_ZERO);
318
319         p.moveTo(0f, 0f);
320         p.curveTo(1.2f, 2f, 0.8f, 1.67f, 3.2f, 4f);
321         p.curveTo(7.2f, 6.67f, 7.2f, 10.67f, 6.4f, 13.33f);
322         p.curveTo(5.6f, 16f, 3.2f, 17.33f, 5.2f, 14.33f);
323         p.curveTo(7.2f, 10f, 3.2f, 6f, 0f, 5.33f);
324
325         return p;
326     }
327
328     public static GeneralPath noteFlagUpDouble() {
329         GeneralPath p = new GeneralPath(GeneralPath.WIND_NON_ZERO);
330
331         p.moveTo(0f, 0f);
332         p.curveTo(1.33f, -3.33f, 6f, -4f, 5f, -12f);
333         p.curveTo(5f, -6f, 1f, -4.33f, 0f, -4.33f);
334         p.curveTo(1.33f, -9.33f, 6f, -9.33f, 5f, -17.33f);
335         p.curveTo(5f, -11.33f, 1f, -9.66f, 0f, -9.33f);
336
337         return p;
338     }
339
340     public static GeneralPath noteFlagDownDouble() {
341         GeneralPath p = new GeneralPath(GeneralPath.WIND_NON_ZERO);
342
343         p.moveTo(0f, 0f);
344         p.curveTo(1.6f, 3.33f, 7.2f, 4f, 6f, 12f);
345         p.curveTo(6f, 6f, 1.2f, 4.33f, 0f, 4.33f);
346         p.curveTo(1.6f, 9.33f, 7.2f, 9.33f, 6f, 17.33f);
347         p.curveTo(6f, 11.33f, 1.2f, 9.66f, 0f, 9.33f);
348
349         return p;
350     }
351
352     public static GeneralPath extraFlagUp() {
353         GeneralPath p = new GeneralPath(GeneralPath.WIND_NON_ZERO);
354
355         p.moveTo(0f, 0f);
```

```
356         p.curveTo(1.33f, -5f, 6f, -5f, 5f, -13f);
357         p.curveTo(5f, -7f, 1f, -5.33f, 0f, -5f);
358
359         return p;
360     }
361
362     public static GeneralPath extraFlagDown() {
363         GeneralPath p = new GeneralPath(GeneralPath.WIND_NON_ZERO);
364
365         p.moveTo(0f, 0f);
366         p.curveTo(1.6f, 5f, 7.2f, 5f, 6f, 13f);
367         p.curveTo(6f, 7f, 1.2f, 5.33f, 0f, 5f);
368
369         return p;
370     }
371     public static GeneralPath flat() {
372         GeneralPath p = new GeneralPath(GeneralPath.WIND_NON_ZERO);
373
374         p.moveTo(-1.6f, 2.67f);
375         p.curveTo(4.8f, 5.78f, 4.8f, -0.89f, -1.6f, -4f);
376         p.curveTo(3.2f, 0f, 3.2f, 3.56f, -1.6f, 1.78f);
377         p.moveTo(-1.6f, 8.89f);
378         p.lineTo(-1.6f, -4f);
379         p.lineTo(-2.8f,-4f);
380         p.lineTo(-2.8f, 8.89f);
381         p.lineTo(-1.6f, 8.89f);
382
383
384         return p;
385     }
386     public static GeneralPath sharp() {
387         GeneralPath p = new GeneralPath(GeneralPath.WIND_NON_ZERO);
388
389         p.moveTo(2.6f, 2.89f);
390         p.lineTo(2.6f, 5.06f );
391         p.lineTo(1.6f, 4.78f);
392         p.lineTo(1.6f, 8.31f); //top
393         p.lineTo(1.0f, 8.31f);
394         p.lineTo(1.0f, 4.62f);
395         p.lineTo(-1.0f, 4.06f);
396         p.lineTo(-1.0f,8.31f);
397         p.lineTo(-1.6f,8.31f);
398         p.lineTo(-1.6f, 3.90f);
399         p.lineTo(-2.6f, 3.62f );
400         p.lineTo(-2.6f, 1.45f );
401         p.lineTo(1f, 2.45f); // inside
402         p.lineTo(1f, -1.88f);
403         p.lineTo(-1f, -2.43f);
404         p.lineTo(-1f, 1.89f); // gap
405         p.lineTo(-1.6f, 1.72f);
406         p.lineTo(-1.6f, -2.60f);
407         p.lineTo(-2.6f, -2.88f);
408         p.lineTo(-2.6f, -5.05f);
409         p.lineTo(-1.6f, -4.71f);
410         p.lineTo(-1.6f, -7.22f); // bottom left
411         p.lineTo(-1f, -7.22f);
412         p.lineTo(-1f, -4.61f);
413         p.lineTo(1f, -4.05f);
414         p.lineTo(1f, -7.22f);
415         p.lineTo(1.6f, -7.22f);
416         p.lineTo(1.6f, -3.89f);
417         p.lineTo(2.6f, -3.61f);
418         p.lineTo(2.6f, -1.44f);
419         p.lineTo(1.6f, -1.72f);
420         p.lineTo(1.6f, 2.61f);
421         p.lineTo(2.6f, 2.89f);
422
423         return p;
424     }
425
426     public static GeneralPath natural() {
```

```
427         GeneralPath p = new GeneralPath(GeneralPath.WIND_NON_ZERO);
428
429         p.moveTo(-1.6f,8.31f);
430         p.lineTo(-1.6f, 1.72f );
431         p.lineTo(1f, 2.45f); // inside
432         p.lineTo(1f, -1.88f);
433         p.lineTo(-1f, -2.43f);
434         p.lineTo(-1f, 1.89f); // gap
435         p.lineTo(-1.6f, 1.72f);
436         p.lineTo(-1.6f, -4.71f);
437         p.lineTo(1f, -4.05f);
438         p.lineTo(1f, -7.22f);
439         p.lineTo(1.6f, -7.22f);
440         p.lineTo(1.6f, -3.89f);
441         p.lineTo(1.6f,4.78f);
442         p.lineTo(-1f,4.06f);
443         p.lineTo(-1f,8.31f);
444         p.lineTo(-1.6f,8.31f);
445
446         return p;
447     }
448
449     public static GeneralPath doubleSharp() {
450         GeneralPath p = new GeneralPath(GeneralPath.WIND_NON_ZERO);
451
452         p.moveTo( 0.39f, 0f);
453         p.lineTo( 2.17f, 1.67f);
454         p.lineTo( 3.34f, 1.67f);
455         p.lineTo( 3.45f, 3.45f);
456         p.lineTo( 1.67f, 3.34f);
457         p.lineTo( 1.67f, 2.17f);
458         p.lineTo( 0f, 0.39f);
459         p.lineTo( -1.67f, 2.17f);
460         p.lineTo( -1.67f, 3.34f);
461         p.lineTo( -3.45f, 3.45f);
462         p.lineTo( -3.34f, 1.67f);
463         p.lineTo( -2.17f, 1.67f);
464         p.lineTo( -0.39f, 0f);
465         p.lineTo( -2.17f, -1.67f);
466         p.lineTo( -3.34f, -1.67f);
467         p.lineTo( -3.45f, -3.45f);
468         p.lineTo( -1.67f, -3.34f);
469         p.lineTo( -1.67f, -2.17f);
470         p.lineTo( 0f, -0.39f);
471         p.lineTo( 1.67f, -2.17f);
472         p.lineTo( 1.67f, -3.34f);
473         p.lineTo( 3.45f, -3.45f);
474         p.lineTo( 3.34f, -1.67f);
475         p.lineTo( 2.17f, -1.67f);
476         p.lineTo( 0.39f, 0f);
477
478         return p;
479     }
480
481     public static GeneralPath restQuarter() {
482         GeneralPath p = new GeneralPath(GeneralPath.WIND_NON_ZERO);
483
484         p.moveTo(-0.52f, 8.87f);
485         p.curveTo(7.83f, 2.09f, -3.13f, 4.17f, 3.39f, -2.61f);
486         p.curveTo(-1.04f, -1.04f, -2.61f, -5.74f, 0.52f, -7.83f);
487         p.curveTo(-4.7f, -5.74f, -3.13f, 0f, 0.52f, -0.53f);
488         p.curveTo(-4.7f, 3.64f, 3.65f, 2.6f, -0.52f, 8.86f);
489
490         return p;
491     }
492
493     public static GeneralPath restEighth() {
494         GeneralPath p = new GeneralPath(GeneralPath.WIND_NON_ZERO);
495
496         p.moveTo(3.11f, 5.44f);
497         p.curveTo(1.94f, 3.5f, 1.17f, 1.94f, -0.58f, 1.55f);
```

```
498         p.curveTo(1.56f, 4.27f, -3.5f, 5.44f, -3.5f, 2.72f);
499         p.curveTo(-3.5f, 1.55f, -2.33f, 0.78f, -1.17f, 0.78f);
500         p.curveTo(1.55f, 0.78f, 1.94f, 2.72f, 2.72f, 4.28f);
501         p.lineTo(-0.7f, -7.78f);
502         p.lineTo(0.19f, -7.78f);
503         p.lineTo(3.71f, 5.44f);
504
505         return p;
506      }
507      public static GeneralPath restSixteenth() {
508         GeneralPath p = new GeneralPath(GeneralPath.WIND_NON_ZERO);
509
510         p.moveTo(3.11f, 5.44f);
511         p.curveTo(1.94f, 3.5f, 1.17f, 1.94f, -0.58f, 1.55f);
512         p.curveTo(1.56f, 4.27f, -3.5f, 5.44f, -3.5f, 2.72f);
513         p.curveTo(-3.5f, 1.55f, -2.33f, 0.78f, -1.17f, 0.78f);
514         p.curveTo(1.55f, 0.78f, 1.94f, 2.72f, 2.72f, 4.28f);
515
516         p.lineTo(1.48f, 0f);
517         p.curveTo(0.31f, -1.94f, -0.46f, -3.5f, -2.21f, -3.89f);
518         p.curveTo(-0.07f, -1.17f, -5.13f, 0f, -5.13f, -2.72f);
519         p.curveTo(-5.13f, -3.89f, -3.96f, -4.66f, -2.8f, -4.66f);
520         p.curveTo(-0.08f, -4.66f, 0.31f, -2.72f, 1.21f, -1.16f);
521        p.lineTo(-0.7f, -8.16f);
522         p.lineTo(0.19f, -8.16f);
523         p.lineTo(3.71f, 5.45f);
524
525         return p;
526      }
527
528
529
530
531
532
533
534
535
536
537 }
```