Chapter 6: Structuring Music Using Linked Lists
Chapter Objectives

The computer science goals for this chapter are:
- To create linked lists of various kinds.
- To understand and use operations on linked lists, including traversals, insertion, deletion, repetition, and weaving.
- To understand the trade-offs between linked lists and arrays.
- To demonstrate a need for tree structures.

The media learning goals for this chapter are:
- To manipulate data flexibly.
- To develop different structures for composing MIDI creatively.
- To use rudimentary forms of automated composition of music.
JMusic: Java Music library

- JMusic knows about WAV files and many other formats, too (e.g., QuickTime)
  - [http://sourceforge.net/projects/jmusic/](http://sourceforge.net/projects/jmusic/)
- We’ll use it for manipulating MIDI
  - Musical Instrument Digital Interface, an industry-standard interface used on electronic musical keyboards and PCs for computer control of musical instruments and devices.
- MIDI is about recording *music*, not *sound*. 
JMusic pieces

- import jm.music.data.*;
  - // To get Note and Phrase classes
  - // .* means “All the classes in the package.”

- import jm.util.*;
  - // To get the View class that knows how to draw music notation from MIDI.

- import jm.JMC;
  - // The class containing constants like JMC.FLUTE and JMC.C4
Welcome to DrJava.

```java
import jm.music.data.*
import jm.JMC;
import jm.util.*;
Note n = new Note(JMC.C4,JMC.QUARTER_NOTE);
n
jMusic NOTE: [Pitch = 60][RhythmValue = 1.0][Dynamic = 85][Pan = 0.5][Duration = 0.9]
JMC.C4
60
JMC.QUARTER_NOTE
1.0
JMC.QN
1.0
Note n2 = new Note(64,2.0);
n2
jMusic NOTE: [Pitch = 64][RhythmValue = 2.0][Dynamic = 85][Pan = 0.5][Duration = 1.8]
```

JMC=JMusic Constants
Makes code easier to read from a music perspective
Creating Phrases

> Phrase phr = new Phrase();
> phr.addNote(n);
> phr.addNote(n2);
> double [] notes1 = {67, 0.25, 64, 0.5, 60, 1.0}
> phr.addNoteList(notes1)
> double [] notes2 = {JMC.G4, JMC.QN, JMC.E4, JMC.EN, JMC.C4, JMC.WN}
> phr.addNoteList(notes2)

Using notes, or an array of note pieces.
Viewing Phrases

> View.notate(phr)
From Viewer: Manipulate and MIDI

- Can save or open MIDI files
- Can change key or time signature.
- Other tools allow changing other characteristics, like tempo.

![CPN: Untitled Phrase menu](image)
# MIDI notes

<table>
<thead>
<tr>
<th>Octave #:</th>
<th>C</th>
<th>C#</th>
<th>D</th>
<th>D#</th>
<th>E</th>
<th>F</th>
<th>F#</th>
<th>G</th>
<th>G#</th>
<th>A</th>
<th>A#</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>0</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
</tr>
<tr>
<td>1</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
</tr>
<tr>
<td>2</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
</tr>
<tr>
<td>3</td>
<td>48</td>
<td>49</td>
<td>50</td>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>59</td>
</tr>
<tr>
<td>4</td>
<td>60</td>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>71</td>
</tr>
<tr>
<td>5</td>
<td>72</td>
<td>73</td>
<td>74</td>
<td>75</td>
<td>76</td>
<td>77</td>
<td>78</td>
<td>79</td>
<td>80</td>
<td>81</td>
<td>82</td>
<td>83</td>
</tr>
<tr>
<td>6</td>
<td>84</td>
<td>85</td>
<td>86</td>
<td>87</td>
<td>88</td>
<td>89</td>
<td>90</td>
<td>91</td>
<td>92</td>
<td>93</td>
<td>94</td>
<td>95</td>
</tr>
<tr>
<td>7</td>
<td>96</td>
<td>97</td>
<td>98</td>
<td>99</td>
<td>100</td>
<td>101</td>
<td>102</td>
<td>103</td>
<td>104</td>
<td>105</td>
<td>106</td>
<td>107</td>
</tr>
<tr>
<td>8</td>
<td>108</td>
<td>109</td>
<td>110</td>
<td>111</td>
<td>112</td>
<td>113</td>
<td>114</td>
<td>115</td>
<td>116</td>
<td>117</td>
<td>118</td>
<td>119</td>
</tr>
<tr>
<td>9</td>
<td>120</td>
<td>121</td>
<td>122</td>
<td>123</td>
<td>124</td>
<td>125</td>
<td>126</td>
<td>127</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TABLE 2.1: MIDI notes**
Relationship between Music and MIDI

<table>
<thead>
<tr>
<th>MIDI</th>
<th>61</th>
<th>63</th>
<th>66</th>
<th>68</th>
<th>70</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notes</td>
<td>C4</td>
<td>D4</td>
<td>E4</td>
<td>F4</td>
<td>G4</td>
</tr>
<tr>
<td>MIDI</td>
<td>60</td>
<td>62</td>
<td>64</td>
<td>65</td>
<td>67</td>
</tr>
<tr>
<td>Hz</td>
<td>291.6</td>
<td>293.67</td>
<td>329.63</td>
<td>349.23</td>
<td>392.00</td>
</tr>
</tbody>
</table>
## MIDI Instruments

<table>
<thead>
<tr>
<th>Piano</th>
<th>Bass</th>
<th>Reed</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 — Acoustic Grand Piano</td>
<td>32 — Acoustic Bass</td>
<td>64 — Soprano Sax</td>
</tr>
<tr>
<td>1 — Bright Acoustic Piano</td>
<td>33 — Electric Bass (finger)</td>
<td>65 — Alto Sax</td>
</tr>
<tr>
<td>2 — Electric Grand Piano</td>
<td>34 — Electric Bass (pick)</td>
<td>66 — Tenor Sax</td>
</tr>
<tr>
<td>3 — Honky-tonk Piano</td>
<td>35 — Fretless Bass</td>
<td>67 — Baritone Sax</td>
</tr>
<tr>
<td>4 — Rhodes Piano</td>
<td>36 — Slap Bass 1</td>
<td>68 — Oboe</td>
</tr>
<tr>
<td>5 — Chorused Piano</td>
<td>37 — Slap Bass 2</td>
<td>69 — English Horn</td>
</tr>
<tr>
<td>6 — Harpsichord</td>
<td>38 — Synth Bass 1</td>
<td>70 — Bassoon</td>
</tr>
<tr>
<td>7 — Clavinet</td>
<td>39 — Synth Bass 2</td>
<td>71 — Clarinet</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chromatic Percussion</th>
<th>Strings</th>
<th>Pipe</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 — Celesta</td>
<td>40 — Violin</td>
<td>72 — Piccolo</td>
</tr>
<tr>
<td>9 — Glockenspiel</td>
<td>41 — Viola</td>
<td>73 — Flute</td>
</tr>
<tr>
<td>10 — Music box</td>
<td>42 — Cello</td>
<td>74 — Recorder</td>
</tr>
<tr>
<td>11 — Vibraphone</td>
<td>43 — Contrabass</td>
<td>75 — Pan Flute</td>
</tr>
<tr>
<td>12 — Marimba</td>
<td>44 — Tremolo Strings</td>
<td>76 — Bottle Blow</td>
</tr>
<tr>
<td>13 — Xylophone</td>
<td>45 — Pizzicato Strings</td>
<td>77 — Shakuhachi</td>
</tr>
<tr>
<td>14 — Tubimba</td>
<td>46 — Orchestral Harp</td>
<td>78 — Whistle</td>
</tr>
<tr>
<td>15 — Tubular Bells</td>
<td>47 — Timpani</td>
<td>79 — Ocarina</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Organ</th>
<th>Ensemble</th>
<th>Ethnic</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 — Hammond Organ</td>
<td>48 — String Ensemble 1</td>
<td>104 — Sitar</td>
</tr>
<tr>
<td>17 — Percusive Organ</td>
<td>49 — String Ensemble 2</td>
<td>105 — Banjo</td>
</tr>
<tr>
<td>18 — Rock Organ</td>
<td>50 — Synth Strings 1</td>
<td>106 — Shamisen</td>
</tr>
<tr>
<td>19 — Church Organ</td>
<td>51 — Synth Strings 2</td>
<td>107 — Koto</td>
</tr>
<tr>
<td>20 — Reed Organ</td>
<td>52 — Choir Aahs</td>
<td>108 — Kalimba</td>
</tr>
<tr>
<td>21 — Accordion</td>
<td>53 — Voice Oohs</td>
<td>109 — Bagpipe</td>
</tr>
<tr>
<td>22 — Harmonica</td>
<td>54 — Synth Voice</td>
<td>110 — Fiddle</td>
</tr>
<tr>
<td>23 — Tango Accordian</td>
<td>55 — Orchestra Hit</td>
<td>111 — Shanai</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Guitar</th>
<th>Brass</th>
<th>Percussive</th>
</tr>
</thead>
<tbody>
<tr>
<td>24 — Acoustic Guitar (nylon)</td>
<td>56 — Trumpet</td>
<td>112 — Tinkle Bell</td>
</tr>
<tr>
<td>25 — Acoustic Guitar (steel)</td>
<td>57 — Trombone</td>
<td>113 — Agogo</td>
</tr>
<tr>
<td>26 — Electric Guitar (jazz)</td>
<td>58 — Tuba</td>
<td>114 — Steel Drums</td>
</tr>
<tr>
<td>27 — Electric Guitar (clean)</td>
<td>59 — Muted Trumpet</td>
<td>115 — Woodblock</td>
</tr>
<tr>
<td>28 — Electric Guitar (muted)</td>
<td>60 — French Horn</td>
<td>116 — Taiko Drum</td>
</tr>
<tr>
<td>29 — Overdriven Guitar</td>
<td>61 — Brass Section</td>
<td>117 — Melodic Tom</td>
</tr>
<tr>
<td>30 — Distortion Guitar</td>
<td>62 — Synth Brass 1</td>
<td>118 — Synth Drum</td>
</tr>
<tr>
<td>31 — Guitar Harmonics</td>
<td>63 — Synth Brass 2</td>
<td>119 — Reverse Cymbal</td>
</tr>
</tbody>
</table>
Different ways of creating Phrases

> Phrase phr2 = new Phrase("Phrase 2", 4.0, JMC.FLUTE);
> phr2.addNoteList(notes2)
A Phrase that starts later

> Phrase phr2 = new Phrase("Phrase 2",4.0,JMC.FLUTE);
> phr2.addNoteList(notes2)
> View.notate(phr2)
Adding parts into phrases (Wrong way first)

```
> Part part1 = new Part();
> part1.addPhrase(phr);
> part1.addPhrase(phr2);
> View.notate(part1);
```

Kinda lost the phrase distinctions.
Building Parts and Scores

> Part partA = new Part("Part A", JMC.PIANO, 1)
> partA.addPhrase(phr);
> Part partB = new Part("Part B", JMC.SAX, 2)
> partB.addPhrase(phr2);
> Score score1 = new Score("My Two Part Score");
> score1.addPart(partA);
> score1.addPart(partB);
Viewing the Score

> View.notate(score1);
Amazing Grace

```java
import jm.music.data.*;
import jm.JMC;
import jm.util.*;
import jm.music.tools.*;

/**
 * Class that represents the song Amazing Grace
 * @author Mark Guzdial
 * @author Barb Ericson
 */

public class AmazingGraceSong {

    /**
     * Constructor to set up the song
     */
    public AmazingGraceSong() {
        myScore.setTimeSignature(3,4);
    }

    private Score myScore = new Score("Amazing Grace");
}

> AmazingGraceSong song1 = new AmazingGraceSong();
> song1.showMe();
```
double[] phrase1Data = {
    JMC.G4, JMC.QN,
    JMC.C5, JMC.HN, JMC.E5, JMC.EN, JMC.C5, JMC.EN,
    JMC.E5, JMC.HN, JMC.D5, JMC.QN, JMC.C5, JMC.HN, JMC.A4, JMC.QN,
    JMC.G4, JMC.HN, JMC.G4, JMC.EN, JMC.A4, JMC.EN,
    JMC.C5, JMC.HN, JMC.E5, JMC.EN, JMC.C5, JMC.EN,
    JMC.E5, JMC.HN, JMC.D5, JMC.QN, JMC.E5, JMC.EN,
    JMC.G5, JMC.DHN
};

double[] phrase2Data = {
    JMC.G5, JMC.HN, JMC.E5, JMC.EN, JMC.G5, JMC.EN,
    JMC.G5, JMC.HN, JMC.E5, JMC.EN, JMC.C5, JMC.EN,
    JMC.E5, JMC.HN, JMC.D5, JMC.QN,
    JMC.C5, JMC.HN, JMC.A4, JMC.QN,
    JMC.G4, JMC.HN, JMC.G4, JMC.EN, JMC.A4, JMC.EN,
    JMC.C5, JMC.HN, JMC.E5, JMC.EN, JMC.C5, JMC.EN,
    JMC.E5, JMC.HN, JMC.D5, JMC.QN,
    JMC.C5, JMC.DHN
};
Phrase myPhrase = new Phrase();
myPhrase.addNoteList(phrase1Data);
myPhrase.addNoteList(phrase2Data);

// create a new part and add the phrase to it
Part aPart = new Part("Parts",
    JMC.FLUTE, 1);
aPart.addPhrase(myPhrase);
// add the part to the score
myScore.addPart(aPart);

/**
 * Method to show the song
 */
public void showMe() {
    View.notate(myScore);
}
Imports and some *private* data

```java
import jm.music.data.*;
import jm.JMC;
import jm.util.*;
import jm.music.tools.*;

public class AmazingGraceSong {
    private Score myScore = new Score("Amazing Grace");

    // myScore is private instance data
```
Filling the Score

This is a constructor that will create the data for the object when it is created.

Each array is note, duration, note, duration, note, duration, etc.

Broke it roughly into halves, just to make it easier to manipulate.

```java
public AmazingGraceSong() {
    myScore.setTimeSignature(3, 4);

double[] phrase1Data = {
    JMC.G4, JMC.QN,
    JMC.C5, JMC.HN, JMC.E5, JMC.EN, JMC.C5, JMC.EN,
    JMC.E5, JMC.HN, JMC.D5, JMC.QN,
    JMC.C5, JMC.HN, JMC.A4, JMC.QN,
    JMC.G4, JMC.HN, JMC.G4, JMC.EN, JMC.A4, JMC.EN,
    JMC.C5, JMC.HN, JMC.E5, JMC.EN, JMC.C5, JMC.EN,
    JMC.E5, JMC.HN, JMC.D5, JMC.EN, JMC.E5, JMC.EN,
    JMC.G5, JMC.DHN
};
double[] phrase2Data = {
    JMC.G5, JMC.HN, JMC.E5, JMC.EN, JMC.G5, JMC.EN,
    JMC.G5, JMC.HN, JMC.E5, JMC.EN, JMC.C5, JMC.EN,
    JMC.E5, JMC.HN, JMC.D5, JMC.QN,
    JMC.C5, JMC.HN, JMC.A4, JMC.QN,
    JMC.G4, JMC.HN, JMC.G4, JMC.EN, JMC.A4, JMC.EN,
    JMC.C5, JMC.HN, JMC.E5, JMC.EN, JMC.C5, JMC.EN,
    JMC.E5, JMC.HN, JMC.D5, JMC.EN, JMC.C5, JMC.EN,
    JMC.C5, JMC.DHN
};
Phrase myPhrase = new Phrase();
myPhrase.addNoteList(phrase1Data);
myPhrase.addNoteList(phrase2Data);

// create a new part and add the phrase to it
Part aPart = new Part("Parts",
    JMC.FLUTE, 1);
aPart.addPhrase(myPhrase);
// add the part to the score
myScore.addPart(aPart);
}
```
public void showMe(){
    View.notate(myScore);
};
The Organization of JMusic Objects

Score: timeSignature, tempo, &

Part: Instrument &

Phrase: startingTime &

Note (pitch,duration)  Note (pitch,duration)

Phrase: startingTime &

Note (pitch,duration)  Note (pitch,duration)  Note (pitch,duration)

Phrase: startingTime &

Note (pitch,duration)  Note (pitch,duration)

Phrase: startingTime &

Note (pitch,duration)  Note (pitch,duration)
Thought Experiment

- How are they doing that?
- How can there be *any number* of Notes in a Phrase, Phrases in a Part, and Parts in a Score?
  - (Hint: They ain’t usin’ arrays!)
How do we explore composition here?

- We want to quickly and easily throw together notes in different groupings and see how they sound.
- The current JMusic structure models music.
  - Let’s try to create a structure that models thinking about music as bunches of riffs/SongElements that we want to combine in different ways.
Introducing the **Linked List**

- A linked list is information broken into smaller pieces, where each piece knows the next piece, but none other.
Another example of a linked list

- Non-linear video editing (like in iMovie)
  - You have a collection of video clips (information)
  - You drag them into a timeline.
  - Each clip still doesn’t know all clips, but it knows the next one.
Why use linked lists versus arrays?

- Just two reasons now, more later:
  1. Can grow to *any* size (well, as long as memory permits)
     - Just create a new element and poke it into the list.
  2. *MUCH* easier to insert!
SongNode and SongPhrase

- SongNode instances will hold pieces (phrases) from SongPhrase.
- SongNode instances will be the nodes in the linked list
  - Each one will know its next.
- Ordering will encode the order in the Part.
  - Each one will get appended after the last.
Using SongNode and SongPhrase

Welcome to DrJava.
> import jm.JMC;
> SongNode first = new SongNode();
> first.setPhrase(SongPhrase.riff1());
> first.showFromMeOn(JMC.FLUTE);
> SongNode second = new SongNode();
> second.setPhrase(SongPhrase.riff2());
> second.showFromMeOn(JMC.PIANO);
> first.setNext(second);
> first.showFromMeOn(JMC.SAX);
Inserting a third node

> SongNode third = new SongNode();
> third.setPhrase(SongPhrase.riff3());
> third.showFromMeOn(JMC.PIANO);
> first.setNext(third);
> third.setNext(second);
> first.showFromMeOn(JMC.FLUTE);
How to think about it

- To start:

  first
  myPhrase: riff1
  next: second

  second
  myPhrase: riff2
  next: null
How to think about it

first

myPhrase: riff1
next: third

second

myPhrase: riff2
next: null

third

myPhrase: riff3
next: second
import jm.music.data.*;
import jm.JMC;
import jm.util.*;
import jm.music.tools.*;

public class SongNode {
    /**
     * the next SongNode in the list
     */
    private SongNode next;
    /**
     * the Phrase containing the notes and durations associated with this node
     */
    private Phrase myPhrase;

    SongNode’s know their Phrase and the next node in the list
Constructor for SongNode

/**
* When we make a new element, the next part is empty, and ours is a blank new part
*/

public SongNode(){
    this.next = null;
    this.myPhrase = new Phrase();
}
Setting the phrase

/**
 * setPhrase takes a Phrase and makes it the one for this node
 * @param thisPhrase the phrase for this node
 */

public void setPhrase(Phrase thisPhrase){
    this.myPhrase = thisPhrase;
}

** Linked list methods

```java
/**
   * Creates a link between the current node and the input node
   * @param nextOne the node to link to
   */
public void setNext(SongNode nextOne){
    this.next = nextOne;
}

/**
   * Provides public access to the next node.
   * @return a SongNode instance (or null)
   */
public SongNode getNext(){
    return this.next;
}
```
/**
 * Insert the input SongNode AFTER this node,
 * and make whatever node comes NEXT become the next of the input node.
 * @param nextOne SongNode to insert after this one
 */

class SongNode {
    public void insertAfter(SongNode nextOne) {
        SongNode oldNext = this.next(); // Save its next
        this.setNext(nextOne); // Insert the copy
        nextOne.setNext(oldNext); // Make the copy point on to the rest
    }
}
Using and tracing `insertAfter()`

```java
SongNode nodeA = new SongNode();
SongNode nodeB = new SongNode();
nodeA.setNext(nodeB);
SongNode nodeC = new SongNode()
nodeA.insertAfter(nodeC);
```

```java
public void insertAfter(SongNode nextOne)
{
    SongNode oldNext = this.next(); // Save its next
    this.setNext(nextOne); // Insert the copy
    nextOne.setNext(oldNext); // Make the copy point on to the rest
}
```
Traversing the list

```java
/**
 * Collect all the notes from this node on
 * in an part (then a score) and open it up for viewing.
 * @param instrument MIDI instrument (program) to be used in playing this list
 */
public void showFromMeOn(int instrument)
{
    // Make the Score that we'll assemble the elements into
    // We'll set it up with a default time signature and tempo we like
    // (Should probably make it possible to change these -- maybe with inputs?)
    Score myScore = new Score("My Song");
    myScore.setTimeSignature(3,4);
    myScore.setTempo(120.0);

    // Make the Part that we'll assemble things into
    Part myPart = new Part(instrument);

    // Make a new Phrase that will contain the notes from all the phrases
    Phrase collector = new Phrase();

    // Start from this element (this)
    SongNode current = this;
    // While we're not through...
    while (current != null)
    {
        collector.addNoteList(current.getNotes());

        // Now, move on to the next element
        current = current.next();
    }

    // Now, construct the part and the score.
    myPart.addPhrase(collector);
    myScore.addPart(myPart);

    // At the end, let's see it!
    View.notate(myScore);
}
```
The Core of the Traversal

```
// Make a new Phrase that will contain the notes from all the phrases
Phrase collector = new Phrase();

// Start from this element (this)
SongNode current = this;
// While we're not through...
while (current != null)
{
    collector.addNoteList(current.getNotes());

    // Now, move on to the next element
    current = current.next();
}
```
Step 1:
// Start from this element (this)
SongNode current = this;

define:
  first
  myPhrase: riff1
  next: third

second
  myPhrase: riff2
  next: null

third
  myPhrase: riff3
  next: second
// While we're not through...
while (current != null) {
    collector.addNoteList(current.getNotes());
}
// Now, move on to the next element
current = current.next();

```java
class Node {
    String myPhrase;
    Node next;
}

// Example
Node first = new Node();
first.myPhrase = "riff1";
first.next = second;

Node second = new Node();
second.myPhrase = "riff2";
second.next = null;

Node third = new Node();
third.myPhrase = "riff3";
third.next = second;
...
// While we're not through...
while (current != null)
{
    collector.addNoteList(current.getNotes());
}
// Now, move on to the next element
current = current.next();
};
// While we're not through...
while (current != null) {
    collector.addNoteList(current.getNotes());
}
// Now, move on to the next element
current = current.next();
}
// While we're not through...
while (current != null)
{
    first
    myPhrase: riff1
    next: node2

    second
    myPhrase: riff2
    next: null

    third
    myPhrase: riff2
    next: second

    current
    null

    Collector:

Then return what you collected

// Now, construct the part and the score.
myPart.addPhrase(collector);
myScore.addPart(myPart);

// At the end, let's see it!
View.notate(myScore);
SongPhrase

- SongPhrase is a collection of static methods.
- We don’t ever need an instance of SongPhrase.
- Instead, we use it to store methods that return phrases.
  - It’s not very object-oriented, but it’s useful here.
import jm.music.data.*;
import jm.JMC;
import jm.util.*;
import jm.music.tools.*;

public class SongPhrase {
    //Little Riff1
    static public Phrase riff1() {
        double[] phrasedata =
        {JMC.G3, JMC.EN, JMC.B3, JMC.EN, JMC.C4, JMC.EN, JMC.D4, JMC.EN};

        Phrase myPhrase = new Phrase();
        myPhrase.addNoteList(phrasedata);
        return myPhrase;
    }
}
//Little Riff2
static public Phrase riff2() {
    double[] phrasedata =
    {JMC.D4, JMC.EN, JMC.C4, JMC.EN, JMC.E4, JMC.EN, JMC.G4, JMC.EN};

    Phrase myPhrase = new Phrase();
    myPhrase.addNoteList(phrasedata);
    return myPhrase;
}
Computing a phrase

static public Phrase pattern1() {
    double[] riff1data = 
    {JMC.G3, JMC.EN, JMC.B3, JMC.EN, JMC.C4, JMC.EN, JMC.D4, JMC.EN};
    double[] riff2data =
    {JMC.D4, JMC.EN, JMC.C4, JMC.EN, JMC.E4, JMC.EN, JMC.G4, JMC.EN};
    Phrase myPhrase = new Phrase();
    // 3 of riff1, 1 of riff2, and repeat all of it 3 times
    for (int counter1 = 1; counter1 <= 3; counter1++)
    {for (int counter2 = 1; counter2 <= 3; counter2++)
        myPhrase.addNoteList(riff1data); 
        myPhrase.addNoteList(riff2data);
    };
    return myPhrase;
}
As long as it’s a phrase...

- The way that we use SongNote and SongPhrase, any method that returns a phrase is perfectly valid SongPhrase method.
10 Random Notes
(Could be less random...)

```java
/**
 * 10 random notes
 **/
 static public Phrase random() {
    Phrase ranPhrase = new Phrase();
    Note n = null;

    for (int i=0; i < 10; i++) {
        n = new Note((int) (128*Math.random()),0.1);
        ranPhrase.addNote(n);
    }
    return ranPhrase;
}
```
10 Slightly Less Random Notes

```java
/*
 * 10 random notes above middle C
 ***/
static public Phrase randomAboveC() {
    Phrase ranPhrase = new Phrase();
    Note n = null;

    for (int i=0; i < 10; i++) {
        n = new Note((int) (60+(5*Math.random())),0.25);
        ranPhrase.addNote(n);
    }
    return ranPhrase;
}
```
Going beyond connecting nodes

- So far, we’ve just created nodes and connected them up.
- What else can we do?
- Well, music is about repetition and interleaving of themes.
  - Let’s create those abilities for SongNodes.
Welcome to DrJava.
> SongNode node = new SongNode();
> node.setPhrase(SongPhrase.randomAboveC());
> SongNode node1 = new SongNode();
> node1.setPhrase(SongPhrase.riff1());
> node.repeatNext(node1, 10);
> import jm.JMC;
> node.showFromMeOn(JMC.PIANO);
### What it looks like

<table>
<thead>
<tr>
<th>node</th>
<th>node1</th>
<th>node1</th>
<th>node1</th>
<th>...</th>
</tr>
</thead>
</table>

![Screenshot of CPN: My Song with musical notes and nodes labeled node, node1, node1, node1, ...]

/**
 * Repeat the input phrase for the number of times specified.
 * It always appends to the current node, NOT insert.
 * @param nextOne node to be copied in to list
 * @param count number of times to copy it in.
 */

public void repeatNext(SongNode nextOne, int count) {
    SongNode current = this; // Start from here
    SongNode copy; // Where we keep the current copy

    for (int i=1; i <= count; i++) {
        copy = nextOne.copyNode(); // Make a copy
        current.setNext(copy); // Set as next
        current = copy; // Now append to copy
    }
}
Here’s making a copy

```java
/**
 * copyNode returns a copy of this node
 * @return another song node with the same notes
 */
public SongNode copyNode(){
    SongNode returnMe = new SongNode();
    returnMe.setPhrase(this.getPhrase());
    return returnMe;
}
```
Step 1:
public void repeatNext(SongNode nextOne, int count) {
    SongNode current = this; // Start from here
    SongNode copy; // Where we keep the current copy
}
Step 2:
copy = nextOne.copyNode(); // Make a copy
Step 3: current.setNext(copy); // Set as next

node

phrase: 10 random notes
next: 

node1

phrase: riff1()
next: null

current

copy

nextOne
Step 4:
`current = copy; // Now append to copy`

```
node
  phrase: 10 random notes
  next:

phrase: riff1()
next: null

node1
  phrase: riff1()
  next: null

current

copy

nextOne
```
Step 5 & 6:

```java
copy = nextOne.copyNode(); // Make a copy
current.setNext(copy); // Set as next
```
Step 7 (and so on):
current = copy; // Now append to copy

node

phrase: 10
random notes
next:

node1

phrase: riff1()
next: null

current

node

phrase: riff1()
next:

copy

nextOne

node

phrase: riff1()
next: null
What happens if the node already points to something?

- Consider `repeatNext` and how it inserts:
  It simply sets the next value.
- What if the node *already had a next*?
  `repeatNext` will *erase* whatever used to come next.
- How can we fix it?
/**
 * Repeat the input phrase for the number of times specified.
 * But do an insertion, to save the rest of the list.
 * @param nextOne node to be copied into the list
 * @param count number of times to copy it in.
 **/

public void repeatNextInserting(SongNode nextOne, int count){
    SongNode current = this; // Start from here
    SongNode copy; // Where we keep the current copy

    for (int i=1; i <= count; i++)
    {
        copy = nextOne.copyNode(); // Make a copy
        current.insertAfter(copy); // INSERT after current
        current = copy; // Now append to copy
    }
}
/**
 * Weave the input phrase count times every skipAmount nodes
 * @param nextOne node to be copied into the list
 * @param count how many times to copy
 * @param skipAmount how many nodes to skip per weave
 */

public void weave(SongNode nextOne, int count, int skipAmount)
{
    SongNode current = this; // Start from here
    SongNode copy; // Where we keep the one to be weaved in
    SongNode oldNext; // Need this to insert properly
    int skipped; // Number skipped currently

    for (int i=1; i <= count; i++)
    {
        copy = nextOne.copyNode(); // Make a copy

        //Skip skipAmount nodes
        skipped = 1;
        while ((current.next() != null) && (skipped < skipAmount))
        {
            current = current.next();
            skipped++;
        }

        oldNext = current.next(); // Save its next
        current.insertAfter(copy); // Insert the copy after this one
        current = oldNext; // Continue on with the rest
        if (current.next() == null) // Did we actually get to the end early?
            break; // Leave the loop
    }
Creating a node to weave

> SongNode node2 = new SongNode();
> node2.setPhrase(SongPhrase.riff2());
> node2.showFromMeOn(JMC.PIANO);
Doing a weave

> node.weave(node2,4,2);
> node.showFromMeOn(JMC.PIANO);
Weave Results

Before:

After
public void weave(SongNode nextOne, int count, int skipAmount)
{
    SongNode current = this; // Start from here
    SongNode copy; // Where we keep the one to be weaved in
    SongNode oldNext; // Need this to insert properly
    int skipped; // Number skipped currently
for (int i=1; i <= count; i++)
{
    copy = nextOne.copyNode(); // Make a copy

    //Skip skipAmount nodes
    skipped = 1;
    while ((current.next() != null) && (skipped < skipAmount))
    {
        current = current.next();
        skipped++;
    };
}
Then do an insert

```java
if (current.next() == null) // Did we actually get to the end early?
    break; // Leave the loop

oldNext = current.next(); // Save its next
current.insertAfter(copy); // Insert the copy after this one
current = oldNext; // Continue on with the rest
```
More on Arrays vs. Lists

- **Arrays**
  - Much easier to traverse
  - Very fast to access a specific (n\textsuperscript{th}) element
  - But really a pain to insert and delete.
    - Hard to write the code
    - Can take a long time if it’s a big array

- **Lists**
  - More complex to traverse
  - Slower to access a specific element
  - Very easy to insert (and later we’ll see, delete)
    - Simple code
    - Takes no time at all to run
How could someone use our work?

- To help someone else use the classes we create, we use JavaDoc.
  - Create the comments in the format that we’ve been using, and the Web pages can be generated.
Another Version:
Creating a tree of song parts, each with its own instrument

- SongNode and SongPhrase offer us enormous flexibility in exploring musical patterns.
- But it’s only one part!
- We’ve lost the ability of having different parts starting at different times!
- Let’s get that back.
The Structure We’re Creating

Starting to look like a tree…
import jm.music.data.*;
import jm.JMC;
import jm.util.*;
import jm.JMC;

public class MyFirstSong {
    public static void main(String [] args) {
        Song songroot = new Song();

        SongNode node1 = new SongNode();
        SongNode riff3 = new SongNode();
        riff3.setPhrase(SongPhrase.riff3());
        node1.repeatNext(riff3,16);
        SongNode riff1 = new SongNode();
        riff1.setPhrase(SongPhrase.riff1());
        node1.weave(riff1,7,1);
        SongPart part1 = new SongPart(JMC.PIANO, node1);

        songroot.setFirst(part1);

        SongNode node2 = new SongNode();
        SongNode riff4 = new SongNode();
        riff4.setPhrase(SongPhrase.riff4());
        node2.repeatNext(riff4,20);
        node2.weave(riff1,4,5);
        SongPart part2 = new SongPart(JMC.STEEL_DRUMS, node2);

        songroot.setSecond(part2);
        songroot.show();
    }
}
import jm.music.data.*; import jm.JMC; import jm.util.*; import jm.music.tools.*;

/**
 * Class that represents a song
 * @author Mark Guzdial
 * @author Barb Ericson
 */

public class Song {

    /** first Channel */
    private SongPart first;

    /** second Channel */
    private SongPart second;

    Our Song will have two branches (parts)
/**
 * Take in a SongPart to make the first channel in the song
 * @param channel1 the first channel in the song
 */

public void setFirst(SongPart channel1) {
    first = channel1;
    first.getMyPart().setChannel(1);
}

/**
 * Take in a SongPart to make the second channel in the song
 * @param channel2 the second channel in the song
 */

public void setSecond(SongPart channel2) {
    second = channel2;
    first.getMyPart().setChannel(2);
}
/**
 * Make the score and show it
 */

public void show() {
    // Make the Score that we'll assemble the parts into
    // We'll set it up with a default time signature and tempo we like
    Score myScore = new Score("My Song");
    myScore.setTimeSignature(3,4);
    myScore.setTempo(120.0);

    // Now, construct the part and the score.
    first.getMyPart().addPhrase(first.collect());
    second.getMyPart().addPhrase(second.collect());
    myScore.addPart(first.getMyPart());
    myScore.addPart(second.getMyPart());

    // At the end, let's see it!
    View.notate(myScore);
}
import jm.music.data.*; import jm.JMC; import jm.util.*; import jm.music.tools.*;

/**
 * Class that represents a part of a song
 * @author Mark Guzdial
 * @author Barb Ericson
 */

public class SongPart {
    /** SongPart has a Part */
    private Part myPart;
    /** the first node in the linked list */
    private SongNode myList;
/**
 * Construct a SongPart
 * @param instrument MIDI instrument (program)
 * @param startNode where the song list starts from
 */

public SongPart(int instrument, SongNode startNode) {
    myPart = new Part(instrument);
    myList = startNode;
}

/**
 * Method to get the part
 * @return the part
 */

public Part getMyPart() {
    return myPart;
}

A SongPart has a Jmusic part, and a list of SongNodes
Let SongNode do the work here

/**
 * Collect parts of this SongPart
 * @return all the notes in a phrase
 */

public Phrase collect() {
    // delegate to SongNode's collect
    return this.myList.collect();
}
public void show() {
    // Make the Score that we'll assemble the part into
    // We'll set it up with a default time signature and tempo we like
    // (Should probably make it possible to change these --
    // maybe with inputs?)
    Score myScore = new Score("My Song");
    myScore.setTimeSignature(3,4);
    myScore.setTempo(120.0);

    // Now, construct the part and the score.
    this.myPart.addPhrase(this.collect());
    myScore.addPart(this.myPart);

    // At the end, let's see it!
    View.notate(myScore);
}
Linked lists (and a simple tree) are powerful representations for music.

Next, we’ll explore the use of this representation for manipulating images.