SUMMARY
In this assignment you will develop a class to work with solos in jMusic.

DEADLINE
This assignment is due on Sunday, May 11 at 11:00 pm.

DESCRIPTION
A LITTLE MUSIC MATH
Music is very mathematical when you break it down. Don’t worry if you don’t have a music background. This section should bring you up to speed on what you need to know for the next part.

The key of a song describes which pitches fit into a song, and which do not. A key can be described by a root and a scale. The root is an integer value pitch that determines the start point of the scale. The scale is an array of ints that determines which offset pitches are “in-key.”

Let’s start with the major scale. The int array for the major scale is \{0, 2, 4, 5, 7, 9, 11\}.
This is also given by JMC.MAJOR_SCALE.

Now, given any root \(N\), figuring out which pitches are in the key is just a simple math problem. The following pitches are in the key:
\(N, N+2, N+4, N+5, N+7, N+9, N+11\)

Which means, of course that the following pitches are not in the key:
\(N+1, N+3, N+6, N+8, N+10\)

Take the root of C for example. Using \(C_4 = 60\) as an example, the following pitches are in the key of C Major:
60, 62, 64, 65, 67, 69, 71

Also, \(C_5 = 72\), so the following pitches are also in the key of C Major:
72, 74, 76, 77, 79, 81, 83

In fact, adding or subtracting 12 to any root gives the same note letter in a different octave, so we can use this formula to determine if any pitch \(P (0 \leq P \leq 127)\) is in the key or not.

Furthermore, this can be done with any root \(R (0 \leq R \leq 127)\), and any scale given as an array of ints \(S = \{S_1, S_2, \ldots, S_n\}\) where \(0 \leq S_i \leq 11\) for \(0 \leq i \leq n\).
**SOLO**

Create a new class called **Solo**, in a package called **solo**. This class will allow you to create and manipulate a solo. It will use a linked list of **SoloNode**s (class provided) to store the note data. In particular your class should have the following:

- **Fields:**
  - `private SoloNode head`

- **Constructors:**
  - `public Solo(SoloNode head)`
    - Create a Solo object with the list referenced by head.
  - `public Solo(double rhythmTotal, int pitchMin, int pitchMax)`
    - Create a random solo:
      - The rhythmValue of each note should be a multiple of 0.25, no greater than 1.
      - The sum of the rhythmValues of all the notes in the solo should be equal to the parameter `rhythmTotal`.
      - The pitch of each note should be between `pitchMin` and `pitchMax`, inclusive.

- **Methods:**
  - `public void fitKey(int root, int[] scale)`
    - Traverse the list and check each note to see if it is in the given key. If not, correct it by setting it to the nearest pitch greater than the original pitch.
    - Only correct pitches by moving up, not down.
  - `public void reverse()`
    - Reverse the solo.
    - For full credit, do not create any new nodes while reversing.
      - i.e., do not use the `new` keyword.
  - `public Phrase getPhrase()`
    - Collect and return the phrase for this solo.
  - `public String getRMA()`
    - Get the relative melody abstraction string for the solo. (See below) The string should be returned with no spaces or additional characters.
  - You may add other methods if needed. Any additional methods should be private.
RELATIVE MELODY ABSTRACTION

The relative melody abstraction of a phrase is a sequence of characters where each character is one of \{U, D, E\}. Given a phrase of \(n\) notes, the RMA has \(n-1\) characters. For each pair of adjacent notes, the RMA string indicates whether the pitch increases (U), decreases, (D), or remains the same (E) from the first note to the second.

EXAMPLE

The following is one example of how to test your code. You should create many other tests as well.

```java
SoloNode head = new SoloNode(new Note(62, 0.5),
    new SoloNode(new Note(65, 0.5),
        new SoloNode(new Note(69, 0.5),
            new SoloNode(new Note(67, 0.5),
                new SoloNode(new Note(69, 0.5),
                    new SoloNode(new Note(69, 0.5),
                        new SoloNode(new Note(72, 0.5),
                            new SoloNode(new Note(74, 0.5),
                                ))))))));
Solo solo1 = new Solo(head);
System.out.println(solo1.getRMA());
View.notate(new Phrase(solo1.getPhrase().copy().getNoteArray(),
    "Manual Solo Original"), 20, 50);
solo1.fitKey(74, MAJOR_SCALE);
System.out.println(solo1.getRMA());
View.notate(new Phrase(solo1.getPhrase().copy().getNoteArray(),
    "Manual Solo In D Major"), 20, 250);
solo1.reverse();
System.out.println(solo1.getRMA());
View.notate(new Phrase(solo1.getPhrase().copy().getNoteArray(),
    "Manual Solo in D Major Reversed"), 20, 450);

Solo solo2 = new Solo(16.0, E2, E6);
View.notate(new Phrase(solo2.getPhrase().copy().getNoteArray(),
    "Random Solo Original"), 600, 50);
solo2.fitKey(C4, MAJOR_SCALE);
View.notate(new Phrase(solo2.getPhrase().copy().getNoteArray(),
    "Random Solo in C Major"), 600, 250);
solo2.fitKey(C4, PENTATONIC_SCALE);
View.notate(new Phrase(solo2.getPhrase().copy().getNoteArray(),
    "Random Solo in C Pentatonic"), 600, 450);
```
**EXAMPLE OUT**

Below are the notations and RMA strings for the manual solo. The random solo will be different each time.

<table>
<thead>
<tr>
<th>NOTATION</th>
<th>RMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPN: Manual Solo Original</td>
<td>UUDUEUU</td>
</tr>
<tr>
<td>CPN: Manual Solo In D Major</td>
<td>UUDUEUU</td>
</tr>
<tr>
<td>CPN: Manual Solo In D Major Reversed</td>
<td>DDEDDUDD</td>
</tr>
</tbody>
</table>
JMusic Documentation

To complete this assignment you need to use many of the features of the Note and Phrase classes, as well as be familiar with the JMC constants. You can find all this information in the jMusic Documentation at:

http://www.cs.vassar.edu/~carle/jmusic/doc/

Reminder

Remember the following points:

- You must use the package solo.
- You must use the SoloNode type for your linked list.
- You cannot modify the SoloNode class. The version provided with the assignment will be the version used to test your code.
- Submit only Solo.java. Do not submit any other files.

Submission

Submit the file

Solo.java

with your method added by the deadline given above. Place all files for submission in a directory in your cs account. Submit the entire directory with the following command:

submit102 assign4 MY_DIRECTORY

For example, if you named your directory assignment-04, and your current working directory contained the assignment-04 directory, you would submit with the following command:

submit102 assign4 assignment-04