Spring, 2016 CIT 590

Programming Languages and Techniques

Homework 10

All deadlines as per canvas

This homework deals with the following topics

* Designing a user interface.

* Working with some starter code (not doing everything from scratch).

* Making something cool to finish off the course.

Please send us information about your partner in the usual form

1. send an email to me
2. write a comment in your canvas submission saying who your partner is
3. only one person should submit. If both of you submit, you will be penalized.

Overall idea

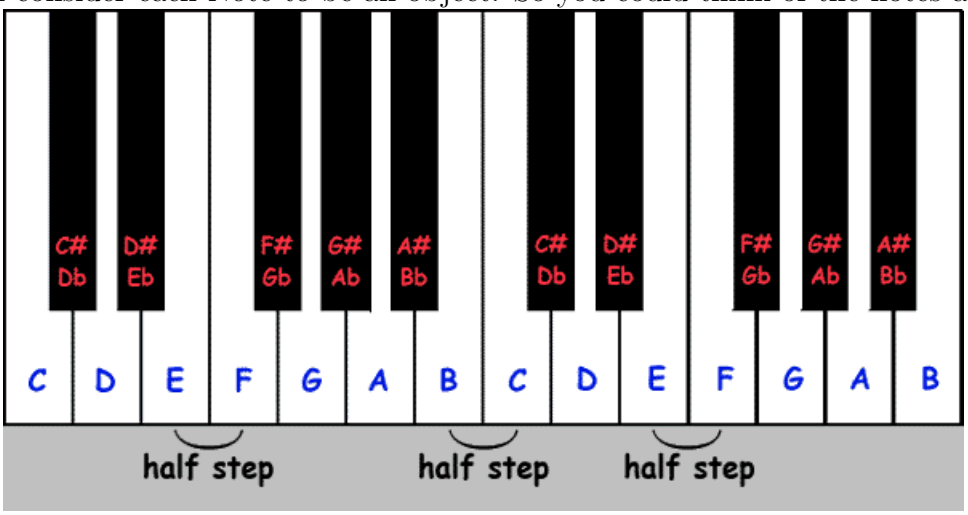
The programming assignment this time concerns writing software that can play some form of music. A song consists of notes, each of which has a duration and pitch. The pitch of a note is described with a letter ranging from A to G. Each set of 7 notes is considered an octave. Since 7 notes alone would not be enough to play very interesting music, after we reach note G, we start another octave from A to G. The note A in one octave will have properties in common to the note A in the next octave. But the second will sound higher since every octave is higher than the last.

Notes may also be accidentals, meaning that they are altered slightly from their usual state. The usual state of notes A through G is called "natural". Making a note "sharp" means it will be a bit higher, at a tone halfway between this note and the next. Making a note "flat" makes it a little lower, halfway between this note and the one before. Finally, if we want to create a few seconds of silence that is called a rest.

For this program we will be representing notes using scientific pitch notation. This style of notation represents each note as a letter and a number specifying the octave it belongs to.

You do not need to understand much about scientific pitch notation, but you can read more about it here: http://en.wikipedia.org/wiki/Scientific_pitch_notation.

It also might help to see the the following figure because while programming in Java you will consider each Note to be an object. So you could think of the notes as piano keys.



Note that while the assignment will eventually produce sound, the implementation is not in line with the way that real music works.

The goal of the assignment is to design a user interface which has buttons and sliders for playing music and adjusting certain properties of the music. Since this is Java, we will be doing this in an object oriented manner. We provide some classes to you and you have to build other classes using them.

There are 2 main classes that we want you to write. One is **Song**. This class uses an array to represent a song which just consists of a series of notes.

The other class we want you to write is the user interface class that actually has interactive UI elements that you use to play the song. This class is called **MusicPlayer**.

A song could have repeated sections. We would like to save the least amount of information so for any section of notes that is going to be repeated, we will save just one set of notes.

The Song class will read files in a very specific format.

Note about specs for this assignment

This is your final assignment of the course. You have done most of your assignments by following instruction after instruction. In this final assignment we would like to ensure that you are able to take a problem without too many detailed instructions and still be able to solve it.

Therefore, please spend time thinking about what you want to write. Remember the main concepts of the course like modularity, unit testing etc. Also, please do not ask us how many unit tests to write, what to unit test etc.

If you feel like something can be unit tested and it is worth unit testing, you should be writing a unit test.

Reading the file

Since it would be difficult for us to read input that is in the form of standard sheet music, we will read input from a standard text file in a specific format.

```
TTLS
Jane Taylor
8
1 C 4 NATURAL false
.5 G 4 NATURAL true
.5 G 4 NATURAL false
.5 F 4 NATURAL false
.5 F 4 NATURAL false
.5 E 4 NATURAL false
.5 E 4 NATURAL false
1 D 4 NATURAL true
```

An example input file is shown above. You can see more examples in the HW10 folder.

The first line is the song title. The second line is the artist.

The third line contains the number of notes in the song. This is equal to the number of subsequent lines.

Each subsequent line represents a single note with 5 arguments that can be passed into the constructor of the note class.

Be careful that some notes can be rest notes (See the section on the note class).

The first number on each line describes the duration of the note in seconds. The next letter describes the pitch of the note, using the standard letters A-G or R for a rest. The third piece of information is the octave that the note is in. The fourth is the note's accidental value of sharp, flat, or natural. (For a rest, the octave and accidental are omitted.) The final piece of information on a line indicates whether the note is the start or stop of a repeated section: true if so, and false otherwise.

In the example above, notes 2-8 represent two repeated sections. The meaning of the file above is that the song should play notes 1, 2, 3, 4, 5, 6, 7, 8, 2, 3, 4, 5, 6, 7, 8.

Our format does not allow nested repetition, nor sections that repeat more than twice.

Implementation details

The implementation of this project involves the following steps. As noted before, we have deliberately provided less detail.

Downloading all the starter files

There are certain aspects of this program that we definitely do not expect you to write yourself. These include things like the actual production of a note from the speakers of your computer. Parts like that are being handled by us. So begin by going to the HW12 folder and downloading each one of the java files. Create a project and put all those files under a single package. Call the project and the package something reasonable.

When you download the classes you will notice that while some of them are completely written out, some are almost entirely empty.

Understanding the Note class

The Note class is completely implemented for you. A Note object represents a single musical note that will form part of a song.

It has as instance variables the length of the note in seconds as a double, the note's pitch (the letters A through G and R for a rest), an int to represent the octave, and the accidental (sharp, natural or flat). Each Note object also uses a boolean to say whether it is beginning or the end of a repeated section.

You pass all this information to the Note's constructor when you create a Note object.

The Note also uses constants named Pitch and Accidental. Pitch.A through Pitch.G or Pitch.R, gives the frequency of the note. Accidental.SHARP, Accidental.FLAT, and Accidental.NATURAL respectively represent the constants associated with sharp, flat and natural.

For the complete documentation of the Note class we want you to use the generated documentation at this web address.

http://www.seas.upenn.edu/ bhusnur4/cit590_spring2015/doc/

You should not have to change a single line of the Note class. You are also not expected to write unit tests for the note class

The Song class

Open up the Song class and you will notice that it is entirely empty. We want you to write the following methods. As you probably suspect, the Song class will contain a collection of notes. We want you to implement the collection of notes using an array. Please note **array** and not **ArrayList**.

You are free to make whatever instance variables you want.

Here are the list of methods to implement. Please copy paste the method signatures down as a first step for making this file.

- public Song(String filename)

  In this constructor you should populate your song's array of notes by reading note data from the specified file. The file format will be the same as described previously. You should use the split function to break up the contents of each note line.

To convert a string into a Pitch or Accidental constant value, like turning the string "SHARP" into the equivalent constant of Accidental.SHARP, use the valueOf method, as shown in the following example:

```
String sharp = input.next();                    // "SHARP"
Accidental acc = Accidental.valueOf(sharp);     // Accidental.SHARP
```

The constructor is the only part of your code that should read data from the input file. All other methods should refer only to your internal array of notes.

Assume valid input. You may assume that the file exists, is readable, and that its contents exactly follow the format described on the previous page.

- public String getTitle()

  Returns the title of the song

- public String getArtist()

  Returns the artist.

- public double getTotalDuration()

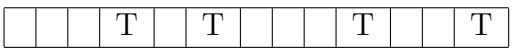
  In this method you should return the total duration (length) of the song, in seconds. In general this will be equal to the sum of the durations of the song's notes, but if some sections of the song are repeated, those parts count twice toward the total.

  For example, a song whose notes' durations add up to 8 seconds that has a 1.0-second repeated section and a 1.5-second repeated section has a total duration of (8.0 + 1.5 + 1.0) = 10.5 seconds.

- public void play()

  In this method you should play your song so that it can be heard on the computer's speakers. Essentially this consists of calling the play method on each Note in your array.

  The notes should be played from the beginning of the list to the end, unless there are notes that are marked as being part of a repeated section. If a series of notes represents a repeated section, that sequence is played twice. For example, in the diagram below, suppose the notes at indexes 3, 5, 9, and 12 all indicate that they are start/end points of repeated sections (their isRepeat method returns true). In this case, the correct sequence of note indexes to play is 0, 1, 2, 3, 4, 5, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 9, 10, 11, 12.



  This method should not modify the state of your array. Also, it should be possible to call play multiple times and get the same result each time.

This method is one of trickier parts of this assignment. Please be careful about how your write it.

- public boolean octaveDown()

  In this method you should modify the state of the notes in your internal array so that they are all exactly 1 octave lower in pitch than their current state. For example, a C note in octave 4 would become a C note in octave 3. Any note that is a rest is not affected by this method, and the notes' state is otherwise unchanged other than the octaves.

  There is a special case. Octave 1 is the lowest possible octave allowed. If any note(s) in your song are already down at octave 1, then the **entire** octaveDown call should do nothing. In such a case, no notes are changed

  Return true if this method lowered the octave, and false if you hit the special case.

- public boolean octaveUp()

  Similar to octaveDown, this raises the octave by 1. This method also has a special case. We do not allow octaves above 10. 10 is the max value that is allowed.

- public void changeTempo(double ratio)

  In this method you should multiply the duration of each note in your song by the given ratio. Passing a ratio of 1 will do nothing. A ratio of 3 will make each note's duration three times as long (slow down the song), or a ratio of 0.25 will make the song speed past at 4 times the speed.

- public void reverse()

  In this method you should reverse the order of the notes in your melody. Future calls to play would play the notes in the opposite of the order they were in before the call. For example, a song containing notes A, F, G, B would become B, G, F, A. This amounts to reversing the order of the elements of your internal array of notes. Do not make a complete copy of your internal array, and do not create any other data structures such as arrays, strings, or lists; just modify your array in-place.

  Please write the reversal code yourself, do not use an existing Java array reversal library.

- public String toString()

  You are writing a toString method in your Song class simply for debugging and for the purposes of also being able to write some kind of unit test. It is near impossible to unit test sound being produced from a speaker. Make your toString method print out enough information. It will be useful here to know that Arrays.toString returns a string representation of an array.

```
> int[] arr = new int[4];
> arr = new int[] {3,3,3,5};
> System.out.println(Arrays.toString(arr))
[3, 3, 3, 5]
```

The user interface - MusicPlayer

The simplest way to describe this class, is that we expect you to make a user interface that looks something like the image below.

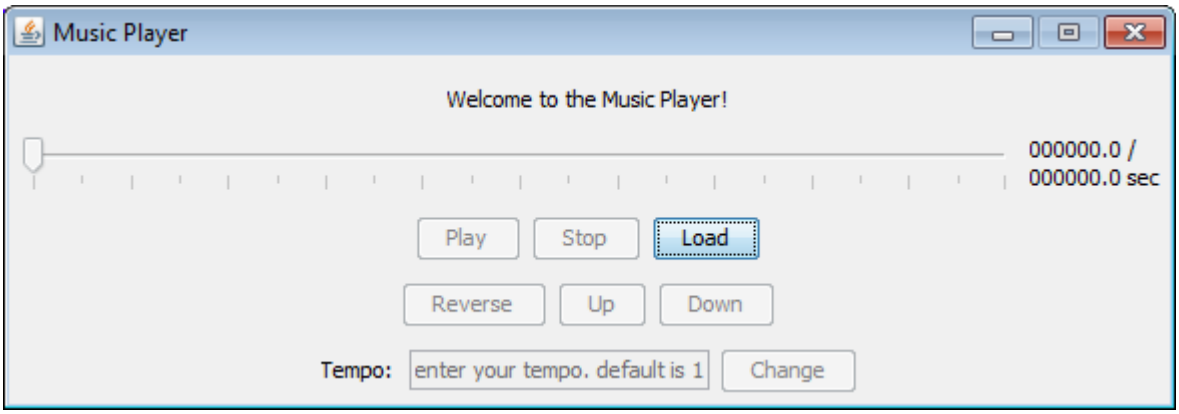
You are definitely free to make it fancy and experiment with different Swing component classes.

We do have the following 'bare-minimum' requirements though.

- There should be some UI component for Play, Stop, Load, Reverse, Up (Octave up), Down (Octave down).
- There should some way of controlling tempo. We are fine with a textbox but you could try something fancier as well.
- There should be a change button which basically reads the tempo entered by the user and changes tempo accordingly.
- There has to be a slider which represents the length of the song and as the song plays the slider gets activated.

While you are welcome to try and make exactly the same user interface as we tried to make, this is the part where I do want to encourage creativity. Come up with your own placement of UI elements.

Please note that some percentage of points in this assignments will be allocated to the user interface. This is definitely a subjective evaluation. Please do not ask the TAs or me whether your 'UI is good enough to get full points'.



When you open up the provided MusicPlayer.java file you will notice that a lot of places are marked with the word TODO. We need you to fill in some code in those places.

If you want to modify any of the provided code, that is totally fine. The provided code is meant to help you, but if you want to design your code differently, that is definitely ok with us.

What to run, how to test audio

At the end of writing all your code, you should be able to run Main.java and be able to play the songs that we have in the HW folder. A big thanks to Yao and Theresa for providing you with some interesting songs to try out.

Hopefully some of these songs are familiar enough that you can tell if your assignment is working or not. Again, remember that this simulator is not perfect. As long as it sounds close to what the file claims to be, you are most likely correct.

Submission

Submit the src folder of your eclipse project as a zip file. Please ensure to include all files needed for your program to run. This includes the code that we are supplying you.

If you are having fun with this assignment and want to submit a song of your own, by all means do that as well. But please note that it does not carry extra credit.

Evaluation

1. 10 pts for user interface
2. 3 pts for a working music player.
3. 10 pts for the non user interface part of your code. You have to ensure that the methods are correct and that they have good style
4. 5 pts for your own unit tests

The TAs will award upto 2 extra credit points for really amazing user interfaces. The emphasis in this HW is very much on the UI part.