COP 3502 Spring 2016

Project 3

**Interplanetary Music Festival**

**UML Diagram with Rationale Writeup Due Date: Friday March 25, 2016 at 3:00pm**

**Program jar file due date (with all three of your java classes): Monday April 4 at 3:00pm**

Your brave work thus far has translated interplanetary messages, and helped create software tools to help rescue humans from Mars. You have ushered in an era of intergalactic peace and creativity. It’s a new Renaissance. At the center of this multi-planet Renaissance is a musical awakening, spurred by vastly different cultures, musical instruments, and combinations of frequency tones never before heard on Earth.

It is time for the first annual interplanetary music festival! Each planet is invited to give two performances: one that reflects that planet’s musical history, and a second that embraces the new, emerging interplanetary sound. There was a lottery to determine which country would send performers this year, and the US won. Then, in a stroke of unprecedented luck, your Citizen Identification Number was randomly selected in a lottery for ordinary citizens who would be performing. In other words, you have been nominated to create and perform Earth’s “new sound”! You’ll travel to the planet of Beaumonde for the music festival which is held in April, a time of year when Beaumonde’s leaves are just falling from the trees. It should be a beautiful trip, and as a bonus, you’ll be traveling with Beyonce and Madonna who will be singing a duet at the music festival to represent a more Earthly musical sound.

This “new sound” you will create is generated fully by computer and it explores the artistic notion of what music could be if we shed all typical structures of scales, along with the combinations of frequencies that are traditionally considered to constitute Earth’s musical notes. After all, Western music (like Beyonce’s, as well as Bach, Mozart, Beethoven, and all the other old guys) follows a regimented series of frequencies (expressed in Hz) which determine the notes. Eastern music does too, but it has many more frequencies that are closer together, which to the Western ear can seem to “bend” sound in interesting ways!

Get ready to bend some sound. You are going to write a song using a computer-generated “guitar” to play chords (with any number of strings you can imagine!) and a “voice” to sing melody (which isn’t constrained to the usual human vocal range).

**Warm-Up Act.**

You must create a Java program that uses three classes to accomplish its objectives. Create a UML diagram of the structure of your program. In an explanation that accompanies your UML diagram, explain each class and what you plan for the job of each method to be.

The Songwriter class contains the main method. This is the class you will run. It takes three command line parameters: two ints followed by a double. Their purpose will be described shortly.

The Guitar class is a template for objects that will store the song itself (much like a guitarist carries music with her so that she is ready to play at a given moment). The guitar object is also where a song will get composed, and it has behaviors you might expect – like “playing” the song.

The third class is Vocalist. A Vocalist object will hold a melody (a sequence of frequencies). A vocalist object also has behaviors you might expect – like singing the song. Ultimately your song will be performed accappella (vocal only, no accompanying instrument). But the guitar is crucial in your songwriting process.

Your code will also use a fourth class that will be provided to you: the StdAudio.java class. Do not attempt to use (or even think about) this class until you have completed Act 1 as described below.

**Act 1.**

You should create the Guitar class. Here are some specifications:

The song must be stored as a two-dimensional array. The rows represent each guitar string. The columns therefore represent each chord (a chord is just one or more notes played at the same time, like when you strum a guitar all of the strings play some frequency). There is an additional row at the bottom (the highest index row) which holds the number of beats each chord gets held. This two-dimensional array captures all the notes that the guitar will play and for how long it should play each one of them.

You should construct a guitar by taking in a number of strings and the song length in number of chords. These will fully determine the size of your 2D song array. Your main program (Songwriter class) will take in these values through command line parameters and can then use them to construct a guitar object.

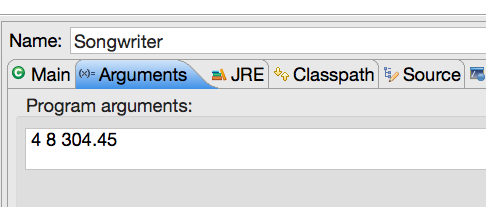
Your guitar object will need a generateSong() method which populates the 2D song array with frequencies for all the notes, along with the number of beats for which each note should be held. You aren’t constrained to normal physical rules of guitars: the frequencies you use to generate the notes are random ranging between 27.5 Hz (inclusive) and 4186 Hz (exclusive). Each frequency is displayed with two decimal places accuracy but stored with its full accuracy.

Then, each chord will be played for a number of beats between 1 second (inclusive) and 3 seconds (exclusive). Fractional seconds are permitted. You randomly generate the duration of each chord.

Your guitar constructor should take care of initializing all of the member variables for itself, including generating the song. In other words your Guitar(int, int) constructor method calls the generateSong() method. The generateSong() method does not require any parameters because the values it needs are member variables in its same class, so it can just access their values as it needs to.

Here is an example.

Under Run -> Run Configurations -> Arguments tab, I entered:



The guitar therefore has 4 strings and the song will have 8 chords. Here is the output to the console generated by the Guitar() constructor (which is called from the Songwriter class containing the main method). The constructor writes this output to the console directly. The values are generated randomly so they will be different every time.

Guitar(): Generated new guitar with 4 strings. Song length is 8 chords.

1546.59 1986.61 3604.10 3469.56 3589.76 538.16 3664.72 2622.66

838.69 1204.43 3051.07 3941.62 1339.81 2385.39 984.57 2925.13

157.18 977.21 3471.75 1887.60 2943.96 182.63 1217.72 3485.39

3844.56 3162.71 3399.16 3411.66 1712.71 447.49 3773.18 2324.46

1.4 1.2 2.0 2.0 1.0 2.2 1.8 1.5

The guitar class must also simulate the playing of a song by printing the chords (combinations of frequencies) to the console along with a visualization of how many beats each chord is played. The printing actually waits that many beats before displaying the next chord. For every beat that it waits, it prints a period. In the simulateSong method, each chord is shown on a single line of output (in contrast to the Guitar() output where each chord is shown vertically). This is for ease of watching the song “play.”

Here is example output of watching the above song simulate playing. It does not all print at once. Each line of numbers prints at once, then each period prints at one second intervals.

Guitar.simulateSong()

3844.56 157.18 838.69 1546.59..

3162.71 977.21 1204.43 1986.61..

3399.16 3471.75 3051.07 3604.10...

3411.66 1887.60 3941.62 3469.56...

1712.71 2943.96 1339.81 3589.76.

447.49 182.63 2385.39 538.16...

3773.18 1217.72 984.57 3664.72..

2324.46 3485.39 2925.13 2622.66..

Finally, your guitar class must return a chord (a column of its 2D array) as a 1D array through the getChordAsArray() method. It returns a double[] and takes a parameter int which tells it which column (chord) you want it to return. It checks if the column is valid and if it is not it prints to the console “No such chord!” and returns null. This method will be needed by your vocalist.

Before proceeding to Act 2, you should now create code in Songwriter’s main method which creates a Guitar object and tests its functionality thoroughly.

**Act 2.**

Second, you should create the Vocalist class.

A vocalist has a mid-range frequency which constitutes the “center” of his or her vocal range. This frequency must be stored as a double. Since the vocalist will always sing a frequency from the guitar chord, the mid-range frequency will be used to determine which frequency from the guitar chord is closest.

A vocalist has a melody variable which is a 2D array. The first row stores the frequencies the vocalist will sing, and the second row stores the number of beats each note should be sung.

Implement a toString() method which returns the vocalist’s melody and number of beats as a string on two lines, formatted like this:

157.18 977.21 3051.07 1887.60 1339.81 182.63 984.57 2324.46

1.4 1.2 2.0 2.0 1.0 2.2 1.8 1.5

Before you proceed, stop and thoroughly test your methods which extract melody notes and then display them. Make sure they are correctly selecting the closest frequency from each guitar chord. Use lots of different tests with different sizes of arrays and different mid-range frequencies. (In other words, vary the command line parameters.)

**Act 3. (The Finale!)**  
Implement a sing() method which actually produces the sound of each melody note for the specified number of beats. You will be able to hear these notes through your computer’s audio. This is the song you have composed! In order to do this, you will use the StdAudio.java class which is being provided to you by calling StdAudio.playTone(double,double). The first double is the frequency of tone that should be played. The second double is (actually an int in our case) how many seconds to play the tone.

**Example output of entire program running with command line parameters 4 8 304.45**

Guitar(): Generated new guitar with 4 strings. Song length is 8 chords.

1546.59 1986.61 3604.10 3469.56 3589.76 538.16 3664.72 2622.66

838.69 1204.43 3051.07 3941.62 1339.81 2385.39 984.57 2925.13

157.18 977.21 3471.75 1887.60 2943.96 182.63 1217.72 3485.39

3844.56 3162.71 3399.16 3411.66 1712.71 447.49 3773.18 2324.46

1.4 1.2 2.0 2.0 1.0 2.2 1.8 1.5

Guitar.simulateSong()

3844.56 157.18 838.69 1546.59..

3162.71 977.21 1204.43 1986.61..

3399.16 3471.75 3051.07 3604.10...

3411.66 1887.60 3941.62 3469.56...

1712.71 2943.96 1339.81 3589.76.

447.49 182.63 2385.39 538.16...

3773.18 1217.72 984.57 3664.72..

2324.46 3485.39 2925.13 2622.66..

Vocalist(): midRangeFreq: 304.45

Vocalist(): songlength: 8

157.18 977.21 3051.07 1887.60 1339.81 182.63 984.57 2324.46

1.4 1.2 2.0 2.0 1.0 2.2 1.8 1.5

**Example output of entire program running with command line parameters 12 20 150**

Guitar(): Generated new guitar with 12 strings. Song length is 20 chords.

3988.64 1460.63 667.75 2892.66 1658.76 1236.02 376.23 548.37 1921.20 4154.55 4109.52 1842.33 650.04 1374.61 3004.42 3558.89 2133.43 1828.53 2602.80 3170.94

2474.13 2042.49 4064.62 243.61 4137.54 415.54 4124.25 2687.05 1492.20 3532.85 3806.21 3972.99 1317.25 2493.78 3844.88 3588.72 609.83 2010.01 3972.68 4012.89

1162.62 3257.21 55.63 160.39 3762.85 3716.05 2512.47 2134.63 2471.96 1393.54 3355.49 1992.86 454.04 3296.77 2329.73 3226.05 2115.37 1548.45 3996.07 1743.90

3041.90 3321.26 946.45 778.60 3174.73 744.51 3600.92 145.64 1749.45 1780.39 127.27 2565.86 3410.07 456.52 2132.00 1298.78 3186.19 2354.31 739.09 4132.93

3816.78 1494.28 460.46 4043.93 1928.08 3799.10 468.82 1871.37 4050.30 313.29 1936.70 175.49 973.99 3378.15 129.60 3960.42 3344.68 3667.98 3694.20 3149.78

658.75 3259.98 3684.98 97.64 454.43 679.29 3767.52 2710.10 2759.40 3750.04 922.06 3178.40 1805.22 2747.12 954.34 3104.26 2132.05 3722.50 4139.78 2708.34

366.35 3003.72 2815.10 3423.42 2261.32 1535.19 499.34 4078.95 1909.43 3203.97 1240.16 4006.37 3242.02 2660.11 2150.48 2811.11 2883.68 467.52 2680.48 1341.98

493.30 3884.25 941.72 1329.35 2972.86 1138.44 3428.35 2445.33 2096.51 3434.82 2501.56 1323.56 1735.36 3785.31 692.93 1687.93 3479.68 2560.52 3358.52 262.34

293.16 4060.72 1839.89 1686.13 3276.74 79.04 759.00 1468.07 2968.08 2697.69 3862.22 1154.44 2564.58 3007.44 3817.03 801.93 3135.29 3346.14 2869.98 1548.28

1792.96 1167.45 3228.82 3260.30 2262.19 1523.36 3575.17 4053.72 775.89 4016.11 2868.76 3311.58 3350.67 3844.91 3264.44 937.15 3258.34 2341.39 3148.23 3129.79

3913.05 2194.12 982.98 2916.46 1083.89 3166.25 1187.52 2655.19 3174.18 3714.08 531.42 1684.26 2143.77 3193.90 47.58 1492.20 2123.98 3678.37 753.20 306.33

3314.18 1642.30 2806.98 408.80 1863.26 4029.78 2562.40 1818.00 404.46 2896.47 971.18 2868.79 2586.22 2132.46 1417.77 629.89 374.93 1446.38 1160.22 374.15

1.9 2.2 1.5 2.0 0.9 1.1 0.7 0.4 0.6 0.7 0.9 2.4 2.3 1.5 1.6 1.2 0.6 1.3 1.4 1.3

Guitar.simulateSong()

3314.18 3913.05 1792.96 293.16 493.30 366.35 658.75 3816.78 3041.90 1162.62 2474.13 3988.64..

1642.30 2194.12 1167.45 4060.72 3884.25 3003.72 3259.98 1494.28 3321.26 3257.21 2042.49 1460.63...

2806.98 982.98 3228.82 1839.89 941.72 2815.10 3684.98 460.46 946.45 55.63 4064.62 667.75..

408.80 2916.46 3260.30 1686.13 1329.35 3423.42 97.64 4043.93 778.60 160.39 243.61 2892.66...

1863.26 1083.89 2262.19 3276.74 2972.86 2261.32 454.43 1928.08 3174.73 3762.85 4137.54 1658.76.

4029.78 3166.25 1523.36 79.04 1138.44 1535.19 679.29 3799.10 744.51 3716.05 415.54 1236.02..

2562.40 1187.52 3575.17 759.00 3428.35 499.34 3767.52 468.82 3600.92 2512.47 4124.25 376.23.

1818.00 2655.19 4053.72 1468.07 2445.33 4078.95 2710.10 1871.37 145.64 2134.63 2687.05 548.37.

404.46 3174.18 775.89 2968.08 2096.51 1909.43 2759.40 4050.30 1749.45 2471.96 1492.20 1921.20.

2896.47 3714.08 4016.11 2697.69 3434.82 3203.97 3750.04 313.29 1780.39 1393.54 3532.85 4154.55.

971.18 531.42 2868.76 3862.22 2501.56 1240.16 922.06 1936.70 127.27 3355.49 3806.21 4109.52.

2868.79 1684.26 3311.58 1154.44 1323.56 4006.37 3178.40 175.49 2565.86 1992.86 3972.99 1842.33...

2586.22 2143.77 3350.67 2564.58 1735.36 3242.02 1805.22 973.99 3410.07 454.04 1317.25 650.04...

2132.46 3193.90 3844.91 3007.44 3785.31 2660.11 2747.12 3378.15 456.52 3296.77 2493.78 1374.61..

1417.77 47.58 3264.44 3817.03 692.93 2150.48 954.34 129.60 2132.00 2329.73 3844.88 3004.42..

629.89 1492.20 937.15 801.93 1687.93 2811.11 3104.26 3960.42 1298.78 3226.05 3588.72 3558.89..

374.93 2123.98 3258.34 3135.29 3479.68 2883.68 2132.05 3344.68 3186.19 2115.37 609.83 2133.43.

1446.38 3678.37 2341.39 3346.14 2560.52 467.52 3722.50 3667.98 2354.31 1548.45 2010.01 1828.53..

1160.22 753.20 3148.23 2869.98 3358.52 2680.48 4139.78 3694.20 739.09 3996.07 3972.68 2602.80..

374.15 306.33 3129.79 1548.28 262.34 1341.98 2708.34 3149.78 4132.93 1743.90 4012.89 3170.94..

Vocalist(): midRangeFreq: 150.0

Vocalist(): songlength: 20

293.16 1167.45 55.63 160.39 454.43 79.04 376.23 145.64 404.46 313.29 127.27 175.49 454.04 456.52 129.60 629.89 374.93 467.52 739.09 262.34

1.9 2.2 1.5 2.0 0.9 1.1 0.7 0.4 0.6 0.7 0.9 2.4 2.3 1.5 1.6 1.2 0.6 1.3 1.4 1.3

Happy Listening!

Some general rules and hints:

* No starter code is being provided for download, but substantial starting points and tips were provided in class on Monday 3/21. If you missed class you must get notes from a classmate. If you ask questions on Piazza about information that was thoroughly covered in that class period, your question may be deleted by the teaching staff.
* If you are not on piazza, email the teaching staff to get yourself added!
* All member variables of classes must be private and use getters/setters as needed.
* All constants may be public.
* You may not use any Array helper methods other than length.
* You must use a regular Java array and access it using standard square bracket ([]) syntax.
* You must turn in your java files, along with your UML diagram as a PDF, in a jar file.
* Grading:
  + 10 points for UML Diagram
  + 65 points for automated tests (approximately 10 automated tests driven by varying command line parameters; checking expected output)
  + 25 points for manual grading (proper commenting of all classes and methods=10; correct audio playing=10; overall style, using concise and clear code=5)
* In the console output, things such as Guitar(), Vocalist(), and Guitar.simulateSong() indicate the method from which those lines of output are written to the console.
* Your main method must only do the following tasks:
  + Instantiate the Guitar and Vocalist objects
  + Tell the guitar object to simulate its song (a method call)
  + Call the toString method of Vocalist implicitly – by placing the name of the object within a System.out.println statement
  + Tell the vocalist object to sing (a method call)
* Hint: In eclipse, if a Java program is still running you will see a little red box on the top right of the Console tab. When you are using pauses it is common for java programs to continue running if you have a bug. You may need to manually terminate your program by clicking the red box.

Policies on submission:

* If you wish to use a slip day (and you have a slip day remaining) you must notify your TA prior to the due date/time of the project. If you do not notify of a slip day, and if you do not submit prior to the due date, your grade for Project 3 code will be a zero.
* You may use slip day(s) for the UML portion of this project or for the code portion. Your UML diagram must be accompanied by an explanation of each class and what each method will do. Basically, the rationale for your design.
* You may upload to Canvas as many times as you wish. We will grade the LATEST submission version. There is no reason to wait until the last minute to submit your code.
* We will ONLY grade files that were uploaded to Canvas. Verify that you uploaded the correct file before you walk away.