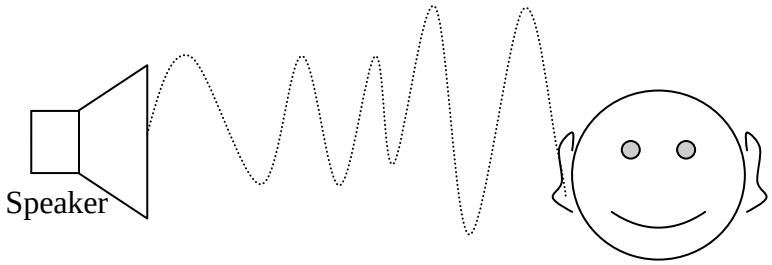
CSC 110 Assignment 6 :
Sound Mixer

Due:
- July 20th @ 11:55pm

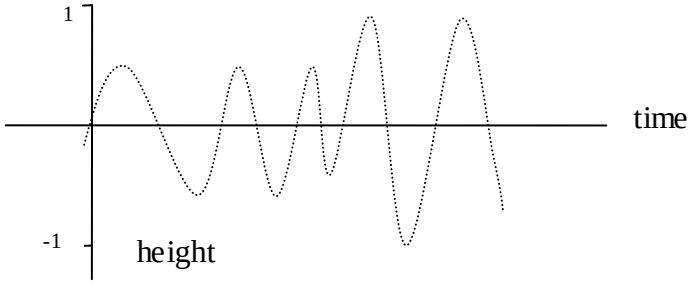
Learning Outcomes:
- How to create a 1 dimensional array
- How to read from and write to array elements, using both explicit and computed index values.

Motivation:
The music industry today heavily relies on the use of software order to create, edit and mix digital audio recordings. There is a growing trend among musicians to make use of computers and technology for both composition and performance. In this assignment you will develop software for manipulating digital audio recordings.

Background:
Our ears hear sounds because of movement (or waves) of air molecules hit our ear drums and are translated by our brains to sound. Speakers that play recorded sound are moving such that they create these waives of air.



The movement of air molecules is called the *Sound* Wave. A computer stores information about sound waves by keeping a list of numbers that represents the height of the wave at many points in time.



Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	etc
Height	.48	.38	-.1	-.5	-.37	.49	-.2	-.5	.4	-.38	.6	.59	-1	-.37	.7	.6	-.6	. . . .

The more points in time that the height is stored, the better the computer is able to make the speaker recreate a simulation of the real sound. This storage of the height of the sound wave at various points in time is called *sampling*. So, when computers store sound, what they store are a list of numbers, called **samples**, that are used to represent the sound at a particular point in time. **Each one of these samples is simply a real number between -1 and +1 !!** When you listen, for example: to an MP3 through a computer's speakers, the computer's hardware is taking the stream of numbers that make up the sound and is converting them into an electric current which your speakers then convert into sound waves.

Loud sounds are actually just larger numbers (ie, closer to +1 or -1) while quiet sounds are smaller numbers (ie, nearer to 0). High pitched sounds are actually sounds in which the wave has a high frequency (ie, the numbers fluctuate quickly between positive and negative), while low pitched sounds created from low frequency waves.

In this assignment you will be asked to write code that:

1. increases or decreases the volume of a sound,
2. mixes two sounds together,
3. reverses a sound (Hey, isn't that something John Lennon did?),
4. splices one sound directly after another
5. increases or decreases the pitch of a sound,

Activities - Part 1 (Making Methods):

1. For the purpose of initial testing your code, two example 'sounds' will be created. In the main method of your program create two arrays of double numbers that are in the range of [-1.0,1.0]. These two arrays should have at least 20 elements each. The initial values can be created however you wish: they could be read in from a file, you could use explicit initialization, etc. Print out the values of both arrays in a manner that makes it easy for you, the programmer, to review the contents of each of your example 'sounds'.

2. Write a method to increase or decrease the volume of one sound. The amount by which the sound is increased or decreased is called the factor. The method should have the following signature and specifications:

```
public static void adjustVolume(double [] samples, int startIndex,
int endIndex, double factor)
```

   - `samples` is a reference to the array containing the sound.
   - Only part of the sound is going to be adjusted. In particular, the samples between indices `startIndex` (inclusive, i.e. including) to `endIndex` (exclusive, i.e. not including) only should be adjusted.
   - `factor` represents the amount of adjustment and needs to be a positive number. In particular, multiply each element of `samples` by the `factor`.
     - Place a comment in your method's code explaining what happens to the volume if `factor > 1` and if `factor < 1`.
   - Test your method with an `endIndex` value that is greater than the length of the `samples` array: A runtime error should occur. Adjust your method such that it will not fail, but will produce an appropriate result. (As the program designer you can solve this error however you choose: there are several possible techniques.)

Print out the values of the sound array both before and after the call to the method and check to ensure that the method functioned correctly.

3. Write a method to mix two sounds together. Mixing requires that the sample values at corresponding indices of the two sounds are added together. The method should have the following signature and specifications:

   `public static void add(double [] samples1, double [] samples2)`
   - The result of adding the two sets of samples together is placed into `samples1`**.** In particular, the `samples1` array is modified by this method, but the `samples2` array is not.
   - Ensure the method handles the following error: If `samples1` contains more samples than `samples2` (or vice versa) the method should not fail, but should produce an appropriate result.
   - Ensure the method handles the following error (called *clipping*): If results of the addition of a pair samples give a magnitude larger than 1, then adjust the volume (i.e., call the `adjustVolume( ... )` method on the `samples1` array in order to *decrease* the magnitude of those sample values.

   Print out the values of the arrays both before and after the call to the method and check to ensure that the method functioned correctly.

4. Write a method to reverse a sound. The method should have the following signature and specifications:

   `public static void reverse( double [] samples )`
   - Reverse the samples. For example, if the original sound contained `{0.1 0.2 0.3 0.4 0.5}`, the result would be `{0.5 0.4 0.3 0.2 0.1}`.

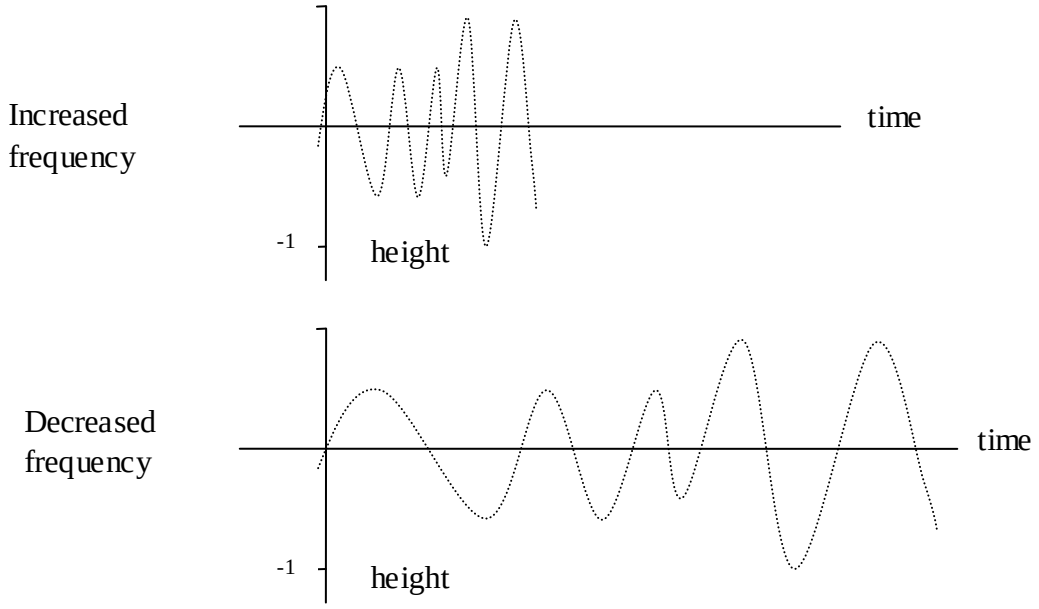
   Print out the values of the sound array both before and after the call to the method and check to ensure that the method functioned correctly.

5. Write a method to replace a segment of one array with a segment of another array. The method should have the following signature and specifications:

   `public static void splice( double [] source, int sourceStart, int sourceStop, double [] destination, int destStart )`
   - The method will copy a segment of one array (called the `source`) from index `sourceStart` to `sourceStop` , to another array (called the `destination`), beginning at index `destStart`
   - Ensure the method handles the following error: If `sourceStop` is beyond the last element in `source`, OR if the `destination` array does not contain enough samples to store all the samples copied from `sourceStart` to `sourceStop`**.**

   Print out the values of the arrays both before and after the call to the method and check to ensure that the method functioned correctly.

6. Write a method to adjust the frequency of the sound represented by an array by a specified factor (positive number). Increasing the frequency of a sound is equivalent to squeezing the sound wave closer together while decreasing the frequency stretches it out.



The algorithm provided below increases the frequency by copying only some of the sample values in the original array to the final array. It decreases the frequency by copying multiple copies of the sample values in the original array to the final array.

The method should have the following signature and specifications:

```
public static double[] adjustFrequency(double[] samples, double
                                                          factor)
```

- `factor` is a positive number and **must** be a double.
  - The value of `factor` determines whether the frequency will increase or decrease.
  - Place a comment in your method's code explaining what happens to the volume if `factor > 1` and if `factor < 1` (but `> 0`).
- The following pseudocode defines how the method *must* be implemented. The `newSamplesIndex` *must* be an `int` variable and the `samplesIndex` *must* be a `double` variable. You will need to work through the pseudo-code, to understand how it adjusts frequency.

```
CREATE newSamples: an array whose length is samples.length / factor
SET samplesIndex = 0

FOR EACH newSamplesIndex FROM 0 to newSamples.length-1

   newSamples[newSamplesIndex ] = samples[samplesIndex ]
   samplesIndex = samplesIndex + factor

ENDFOR
```

A reference to the `newSamples` array is returned.

When calling this method, send the array of sample numbers that represent the sound and capture the returned array reference in same variable (for example,

```
testSample = adjustFrequency(testSample, 0.7);
```

where `testSample` is the name of the original array of numbers in the calling method). Print out the values of the array both before and after the call to the method and check to ensure that the method functioned correctly.

What to Hand In:

An electronic copy of the final *.java program: This is handed in using the assignments link on the course Connex site.

More to Discover:

Some musicians create music solely through interaction with computers and technology. For example, PLOrk, the Princeton University Laptop Orchestra, is composed of computer musicians who use laptops and computer programming to produce music. Other musicians, such as those at the Music Technology Group in Barcelona, Spain, have been able to create innovating technology-driven instruments such as the Reactable, a sound-synthesizing multi-touch-sensitive table.

In this assignment, you use programming to create tools for audio manipulation. Some computer musicians create music by sampling pre-recorded tracks in this way. Other computer musicians synthesize digital (discrete) sound waves from scratch, which can then be converted into analog (continuous) sound and played through a set of speakers. If you would like to learn more about computer music, check out the following resources : [2], [3]. In addition, you may be interested in exploring one of several free software tools for computer music composition and performance, such as CSound [4, 5], jMusic [6], and Pure Data (PD) [7].

References:

[1] Mark Guzdial , Barbara Ericson, *Introduction to Computing and Programming with Java: A Multimedia Approach*, Prentice-Hall, Inc., Upper Saddle River, NJ, 2006.

[2] Charles Dodge, Thomas A. Jerse, *Computer music: Synthesis, composition, and performance*, Schirmer Books: London, 1997.

[3] Curtis Roads, *The computer music tutorial,* MIT Press, Cambridge, MA, 1996.

[4] Richard Boulanger (ed), *The csound book: Perspectives in synthesis, sound design, signal processing, and programming,* MIT Press, Cambridge, MA, 2000.

[5] CSound : http://www.csounds.com/

[6] jMusic : http://jmusic.ci.qut.edu.au

[7] Pure Data : http://puredata.info