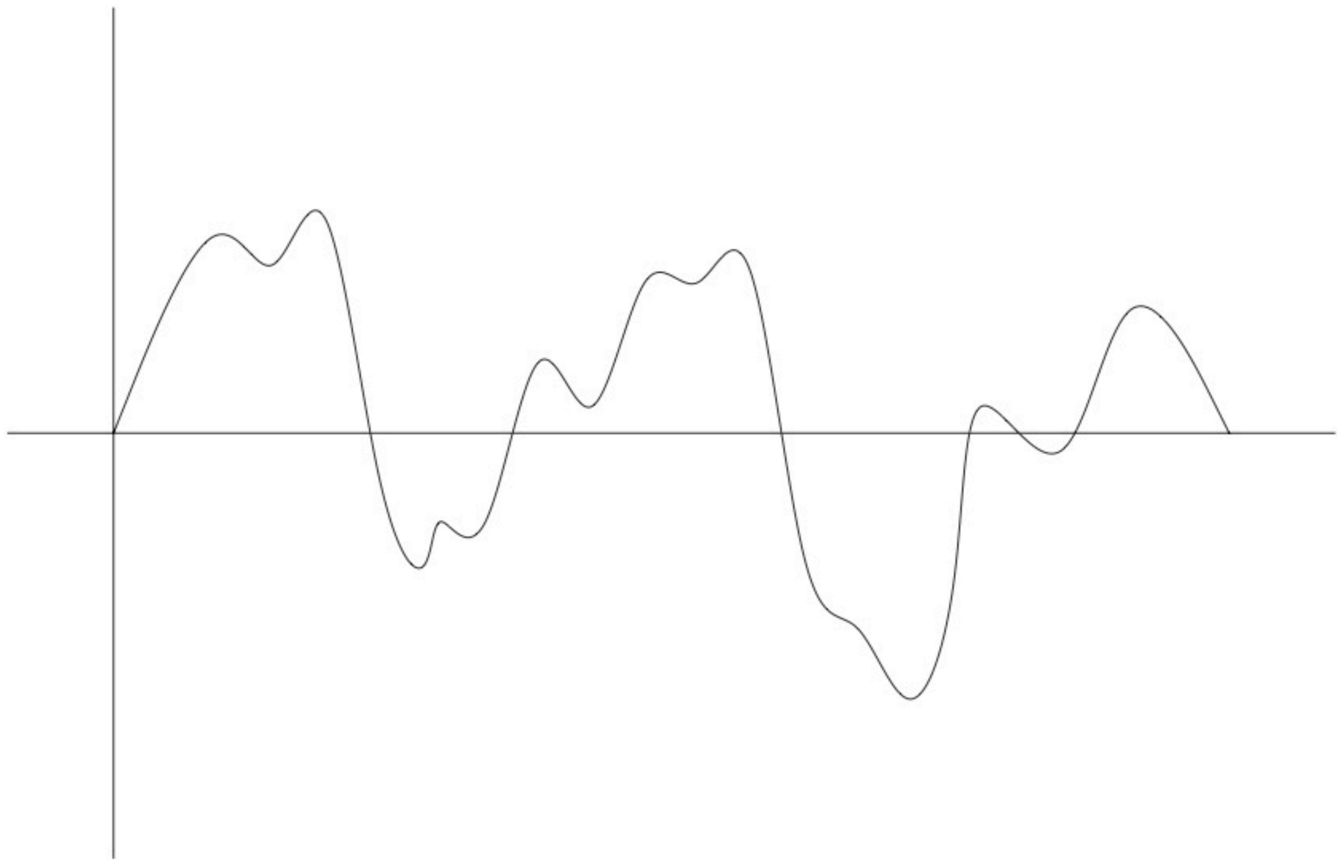
David Galles
Computer Science
Univeristy of San Francisco

Programming Assignment 1: Playing with Sound

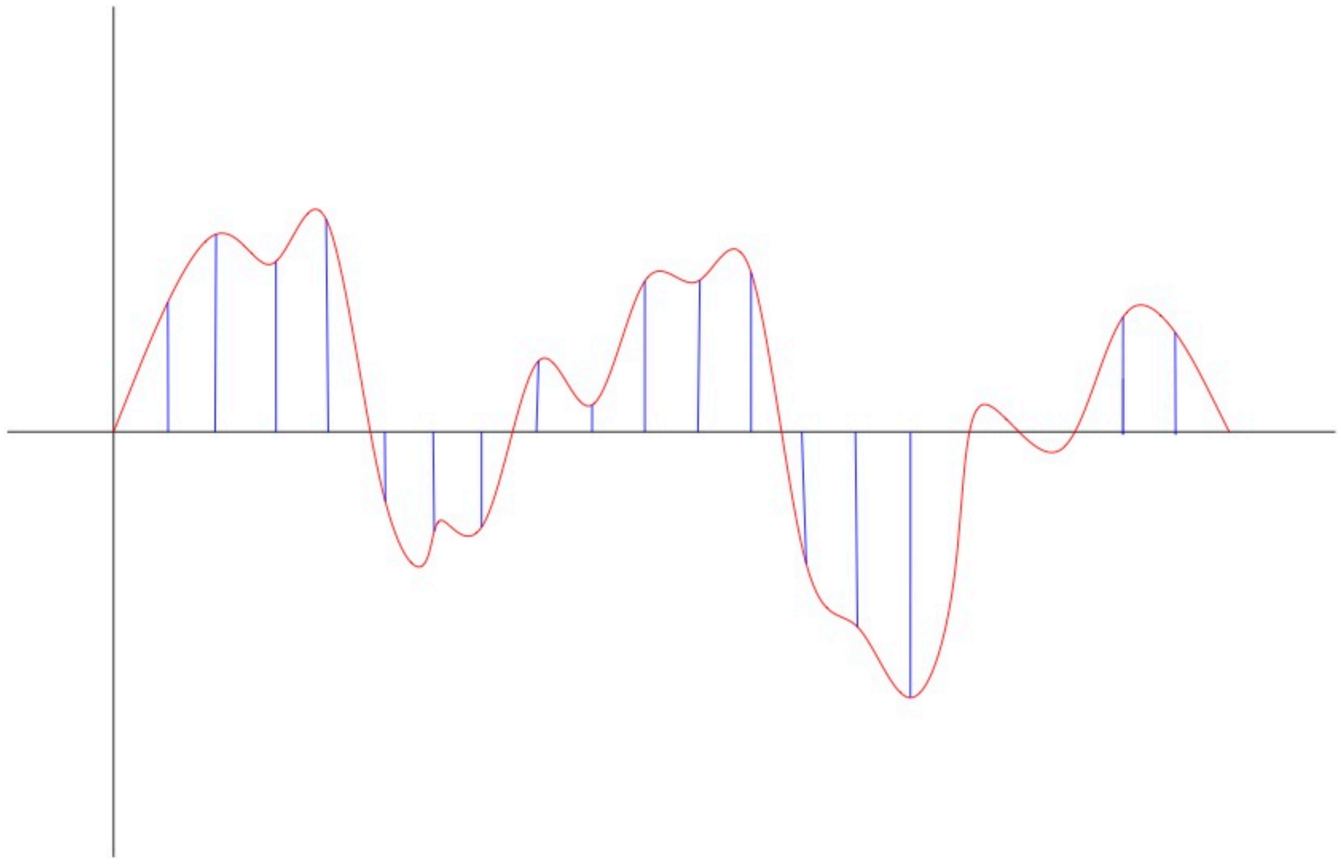
- Due Date: 2/22/2016

Sound

Sound waves are compressions of air.



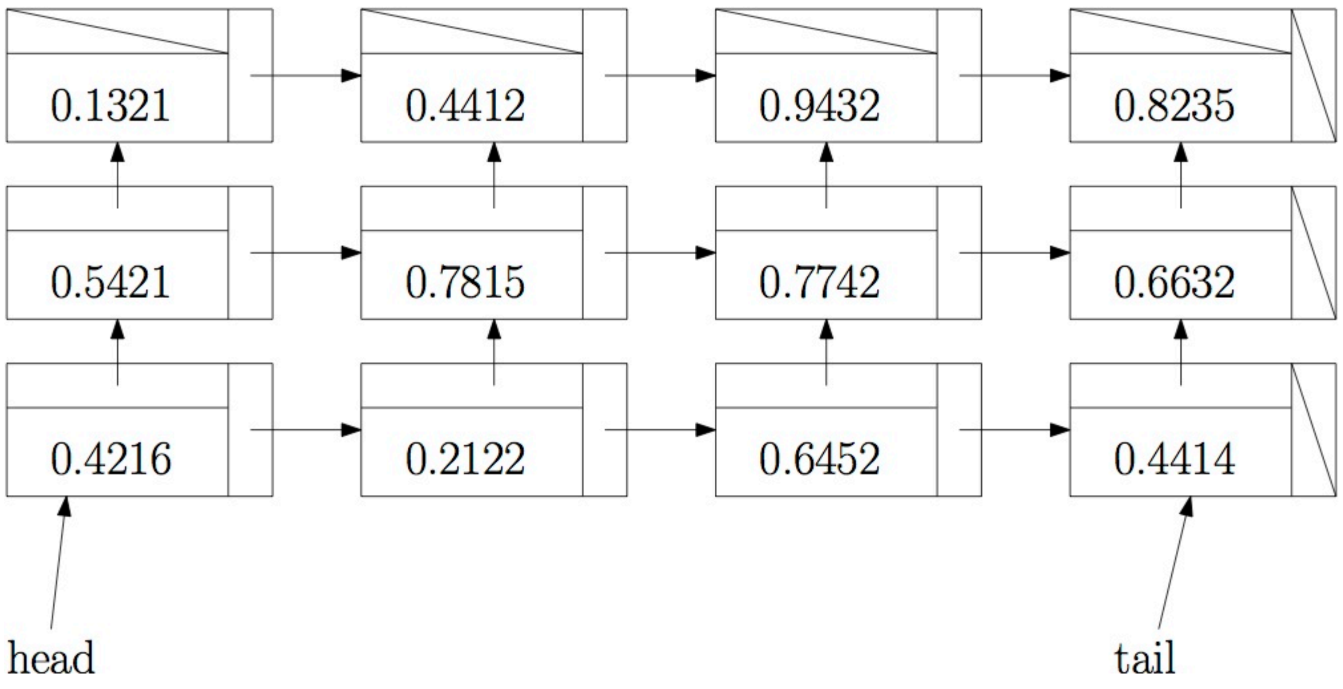
Sound waves are an inherently analog phenomena, to represent them in a digital domain, we need to *sample* them. We choose a *sampling rate* (say, 10000 samples a second) and then every 1/1000th of a second, we record the value of the wave at that point. We then save all of these samples, and that is our internal representaion of the sound.



Sometimes we want to record multiple channels of sound. For instance, stereo headphones have 2 separate channels of sound (one for the left ear, and one for the right). Surround sound systems can have even more channels -- 5.1 has 6 channels: Front-Left, center, Front-Right, back-left, back-right, subwoofer.

Linked Lists

For the first project, You will implement a data structure that stores and manipulates sound clips using a linked-list structure. Speficially, sounds will be stored as a linked list of samples, where each sample is a linked list of the sampled values for each channel. For example, a SoundList for a 3-channel sound containing 4 samples would look something like the following:



Note that your are NOT ALLOWED to use ArrayList, java.util.LinkedLists, or any other element from the Java collection framework to create you MusicList -- you need to create your own Linked List **ENTIRELY FROM SCRATCH**. Also, you should **NOT** use arrays of any kind to store your MusicList (though you will need to use arrays as parameters / return values of some of your methods

But ... But ...

I can already hear you saying:

- But arrays woould be more effiient!
- But using ArrayLists is much easiear!
- But in the *real world* we will use libraries!

All valid points, however:

- The point of this assignment is to get comfortable with linked lists. I tried to come up with a compelling application, but the point is to learn to manipulate linked structures, not to create the best possible implemenation of a sound library.
- In this instance, easier is not better -- remember that the point is to learn how to use somewhat complicated linked structures.
- Before you can use libraries effectively, you need to know exactly how they are implemented (that is sort of the point of this class!) Be glad I have not given you an assignmment that *cannot* easily use the Java libraries! (Yes, such problems exist! I have needed to solve them in the wild. They are also very. very difficult. Fun, but difficult!)

Implementation

You will implement the following interface:
MusicList.java

```
import java.util.Iterator;
public interface MusicList
{
    public int getNumChannels();

        public float getSampleRate();
        public int getNumSamples();
        public float getDuration();
        public void addEcho(float delay, float percent);
        public void reverse();
        public void changeSpeed(float percentChange);
        public void changeSampleRate(float newRate);
        public void addSample(float sample);
        public void addSample(float sample[]);
        public Iterator<float[]> iterator();
        public Iterator<Float> iterator(int channel);
        public void clip(float startTime, float duration);
        public void spliceIn(float startSpliceTime, MusicList clipToSplice);
        public void combine(MusicList clipToCombine, boolean allowClipping);
        public void makeMono(boolean allowClipping);
        public MusicList clone();
}
```

Where:

- **getSampleRate** Returns the sample rate of this sound clip. This one is easy, you just need to return the sampleRate that was used when the SoundList was created
- **getNumSamples** Returns the number of samples in the list. This should be stored (and not recalculated each time getNumSamples is called
- **getDuration** Returns the duration of the clip, in seconds. Relatively easy to compute given the number of samples and the sample rate
- **addEcho** Adds an echo effect to the soundList
- **reverse** Reverses the list of samples. Helpful for finding backmasked messages, like Paul is dead
- **changeSpeed(float percentChange)** Change the speed of the SoundList. A percentChange of 1.0 leaves the sound unchanged, while 2.0 makes the clip twice as fast (also twice as high!) Can be easily accomplished by just changing the sample rate, and leaving hte samples alone
- **changeSampleRate(float newRate)** Change the sample rate to to a new value. This is the most difficult of all of the methods
- **addSample(float sample)** Adds a sample to the end of the SoundList. If the SoundList is not single-channel, thiis method should throw and IllegalArgument exception
- **addSample(float sample[])** Adds a sample to the end of the SoundList. If the length of the sample array is not the same as the number of channels in the SampleList, throw an IllegalArgument exception
- **Iterator<float[]> iterator()** Returns an iterator to traverse the SoundList. Each call to next returns an array of samples, one for each channel
- **Iterator<Float> iterator(int channel)** Returns an iterator to traverse a single channel of the SoundList
- **clip(float startTime, float duration)** Clips the SoundList by throwing away all samples before the startTime (in secounds), and after the duration (in seconds). So, if the SoundList was 6 seconds long, and we called clip(4,2), the new SoundList would be 2 seconds long (and would consist of samples from second 4 to second 6 of the original SoundList)
- **spliceIn(float startSpliceTime, MusicList clipToSplice)** Splice clipToSplice into this clip, starting at startSpliceTime
- **combine(MusicList clipToCombine, boolean allowClipping)** Add the waveform of clipToCombine to this clip. If allowClipping is true, clip all samples in the range -1, 1. If allowClipping is false, rescale resuting waveform to be in the range -1, 1
- **public void makeMono(boolean allowClipping);** Combine all channels into one. If allowClipping is true, clip all samples in the range -1, 1. If allowClipping is false, rescale resuting waveform to be in the range -1, 1
- **MusicList clone()** Return a clone (deep copy) of the SoundList

Test Files

UPDATED We have provided the file TestMain.java to help you test your MusicList implementation. Now updated to test clip and iterator(channel) methods.

Assignment

For your first assignemnt, you will

- Implement a class named **MusicLinkedList** that implements the **MusicList** interface. The constructor for your NusicLinkedList should take as an input parameters the sample rate (float) and the number of channels (int)
- You are *Not Allowed* to use *any* Java Collections: No ArrayList, no LinkedList! You need to create your own linked structure for this project!
- Your MusicList should not store any arrays at all -- use linked structures only. (Yes, you could make an arguement that arrays would be a better fit for at least part of this assignment, but this is a linked list assignment)
- Your MusicList should be able to handle an arbitrarily large number of channels.

Due Date

Your MusicList class should be checked into subversion by Monday, Feb 22nd, 2016.

Submission

Submit your files using subversion. Your files should be stored in the subversion In fact, I recommmend that you don't wait until you program is done to get it into subversion, start right away to protect yourself. The subverion directory you should use for this proect is https://www.cs.usfca.edu/svn/*username*/cs245/project1/, where *username* is your cs username.

Collaboration

It is OK for you to discuss solutions to this program with your classmates. However, **no** collaboration should **ever** involve looking at one of your classmate's source programs! It is usually extremely easy to determine that someone has copied a program, even when the individual doing the copying has changed identifier names and comments.

Provided Files

The following files are provided. Note that you need to use the interfaces as they are given, so that you program will work correctly with the testing code that we will provide.

- MusicList.java Your Class MusicLinkedList needs to implement this interface
- JavaDoc Documentation of your required class and interface
- SoundUtil.java Some utility functions for reading .wav files into SoundLists, and playing SoundLists
- TestMain.java Some test code
- test2.wav Test file used in TestMain.java
- one_four.wav Test file used in TestMain.java
- two_three.wav Test file used in TestMain.java