Sound Exercise

Due April 8th (after break).

(30 pts)

Speech Synthesis:

Synthesized speech is a technology we are seeing crop up in more and more places today. We want our technology to be able to speak for a variety of reasons. As such, there are two basic schools of synthesized speech, depending on the needs of the user.

One school of synthesized speech is known **as "Database-Driven" synthetic speech**. This is the type of synthetic speech used by Siri and other technologies in which naturalness and "humanness" is considered important. Many people who use synthetic speech as their main form of communication (e.g., people with cerebral palsy, brain trauma, a stroke, autism, and a spinal cord injury (high level)) most likely use Database-driven synthetic speech. With this type of speech synthesis, some poor person records hours and hours of speech in a sound-proof room. Then the speech is analyzed and labeled (this used to be done by some other even poorer grad student, but now is done automatically) and stored in a database (which could be a multi-dimensional array of objects). Then, when speech is synthesized, the longest continuous pieces of recorded speech are appended to the best matching longest segments of recorded speech.

This type of synthesis has the advantage of sounding relatively natural and human, it incorporates accents and dialects, and allows different people to each have their own voice.

However, it has a few disadvantages. For one, it doesn't necessarily do well with novel utterences (e.g., unusual names or new words or new word combinations that weren't recorded and stored in the database). When you listen to Siri or perhaps your car's direction system and you hear a glitch in the speech, what you're hearing is two pieces of speech that are being put together to make a word or utterance, and those two pieces of speech don't match up particularly well. It would be impossible to record a database of speech large enough to cover all possible utterances and utterance combinations.

Another disadvantage is that the speed of the speech is pretty much stuck at the rate at which the individual who recorded the database spoke it. Why is this a disadvantage? Well, people who are blind who use synthetic speech to read through documents need their synthetic speech to go at ridiculously fast speeds. We'll talk about this more in a moment.

The other type of synthesized speech is known as "Rule based" synthetic speech, or Formant based synthetic speech. This was the original type of synthetic speech, and is the type of synthesizer Stephen Hawkins uses. It is also used by the parking garage ticket payment machines at various garages across campus. Does that sound particularly human to you? It is also the type of synthesis used by people who need documents read back to them at a fast pace (i.e., people who are blind and low vision, dyslexic, etc.). Instead of visually reading or scanning through a document, people who are blind use synthetic speech to read the document to them. However, we visually read through documents MUCH more quickly than we would read a document aloud. So database-driven synthetic speech won't allow for people who are blind to read through a document in anything close to the time it takes a visual reader to get through a document. This is a serious disadvantage at work and in school. As a result, devices that read documents aloud often use rule-based synthetic speech. People often set the rate of these devices to be between 400 and 500 words per minute (our normal speaking rate is about 180 words per minute). I'm including a link on my web page to a synthesizer that is reading at 400 words per minute. Clear, huh?

Rule based synthesis works by using rules for how to create particular sounds (e.g., a vowel sound), and then including rules for how to transition from one sound to another. Every single person does this a bit differently, and we can't make rules for every single person, so we lose the naturalness with this type of speech. However, rules can be sped up, so we gain the ability to have the synthesizer speak at quite fast rates.

But back to vowel sounds. A basic vowel sound consists of a basic (fundamental) frequency, representing your pitch, and then a combination of resonating frequencies based on how you shape your vocal track, including your mouth, your tongue, your teeth, and your lips.   Each of these resonating frequencies combines with the fundamental frequency to make a vowel (it's not quite this simple, as you'll hear, but this is where we started and what rule-based synthesis is based on). So, for instance, the "ah" vowel in words like "father" and "top" is created in most men with a fundamental frequency of on average 123, a first resonating frequency of 768, a second resonating frequency of 1333, a third resonating frequency of 2522, and a fourth resonating frequency of 3687. The first two resonating frequencies are the most important in creating a vowel sound.

Your job:

You're going to create a class for making sounds (for this exercise, we'll be creating vowel sounds). I'm including a file called Speech.java that includes all the fields and methods needed to play out sound. The file has a field called data, which is an array of bytes. The bytes are what are played. An array of doubles is converted to an array of bytes using type conversion, before it is played out.

You will make a class for creating vowel sounds. (For this class, you will need to import the java.io.File library at the top of your class definition).

Fields:

The class will have as a field a Speech object from the Speech class file you are downloading from my web page. It will also need a field for your SoundDevice, e.g.,

SoundDevice device = new SoundDevice();

This is a class that gives you information about your sound device and must be downloaded from my web page. Among other things, it has as a field Sampling_Rate, which is the rate at which it is able to play back information. In other words, how many numbers in an array can be played back per second. So, for instance, if you want a sound to play for 2 seconds, and your machine is able to play back 8000 samples per second, you'd need to create an array that is 16000 bytes long. (And, just for the record, most quality recorded sound is closer to 40,000 samples per second. Your array gets pretty long pretty fast, huh.)

Methods:

Sin Wave Method:

The class will have a method that creates a sin wave of a particular duration and a particular frequency. Thus, it will take as input the frequency that you want the sin wave to be and the duration you want the sin wave to be (both doubles) and it will return an array of doubles, containing the sin wave. (To create your new array, you'll need the device.Sampling_Rate to get the sampling rate of your device).

To create a sin wave at a particular frequency for the array, you first need to figure out where in the wave's curve you should be at each particular index in the array. So you will first need to calculate a curve value, calculated as Math.PI multiplied by 2 and the frequency of the sin wav you are creating, then divided by the device's sampling rate. Now when you create the array holding the sin wave at the desired frequency, you'll use Math.sin(index * curve value).

This new array containing the sin wave is what is returned by this method.

Vowel Method:

This method takes as input 6 integers: the first is the frequency of the pitch, or fundamental frequency. The second, third, fourth, and fifth are the first, second, third, and fourth resonating frequencies, respectively. The 6th parameter is the duration (in seconds) of the vowel sound you want to create (I usually went with 2 seconds). The method then creates 1 sin wave for the pitch (fundamental frequency) using the method to follow, and then 4 more sin waves of the same duration for each of the other resonating frequencies. It then combines all of these frequencies into one sound wave (one array) as follows: at each location, you should create an average of the five different frequency wave arrays at that location, with the first and second resonating frequency having twice the weight of the other 3 waves. This is the array that should be returned from this method.

Pitch Method:

Finally, it turns out that our pitch (fundamental frequency) isn't a perfect sin wave. Our vocal cords open and then close relatively quickly, causing a lot of energy at the beginning of the cycle, and then a quick drop-off in energy. So we want to weight each sin wave cycle so that the beginning of the cycle has a lot of weight, and the end has very little weight. To do this we first create a new sin wave array of the fundamental frequency. We then need to figure out the duration of each cycle in the array. This is just the sampling rate divided by the fundamental frequency. Now, for each cycle, we multiply the array with a weight, starting at 1.0 and decreasing for each value by 1.0/cycle_length.

That's pretty much it for the vowel sound class. The only thing left is to create a main class with your main function in it, in which you create an object from your vowel sound class and then use the object to add a vowel sound to your soundwave, and then play it back and/or save it to a file, e.g.,:

```java
VowelClass st = new VowelClass();
st.speech.add(st.makevowel(123, 768, 1333, 2522,3687, 2));
st.speech.play(st.device);
st.speech.save(st.device.getFormat(), new File("vowel.wav"));
System.exit(0);
```

Now you have to test your speech synthesizer. The calculated average frequencies for vowels for men are as follows:

138, 342,2322,3000,3657 (see, heat)
135,427,2034,2684,3618 (hit, sitting)
129, 476, 2089, 2691,3649 (baby, gate)
127, 580,1799,2605, 3677 (met, bed)
123, 588,1952,2601,3624 (cat, fat)
123, 768, 1333, 2522,3687 (father, arm)
121, 652,997,2538,3486 (saw, bought)
129, 497, 910, 2459, 3384 (boat)
133, 469, 1122, 2434, 3400 (book)
143, 378, 997, 2343, 3357 (boot)
133,623, 1200, 2550, 3557 (cup, luck)
130, 474, 1379, 1710, 3334 (fur)

For females, the frequencies are as follows:

227, 437, 2761, 3372,4352(see, heat)
224, 483, 2365, 3053,4334 (hit, sitting)
219,536, 2530, 3047, 4319(baby, gate)
214, 731, 2058, 2979, 4294(met, bed)
215, 669, 2349, 2972, 4290 (cat, fat)
215, 936, 1551, 2815, 4299 (father, arm)
210,781, 1136,2824, 3923 (saw, bought)
217, 555, 1035, 2828, 3927 (boat)
230, 519, 1225, 2827, 4052 (book)
235, 459, 1105, 2735, 4115 (boot)
218, 753, 1426, 2933, 4092 (cup, luck)
217, 523, 1588, 1929, 3914 (fur)

These frequencies aren't set in stone- feel free to tweak them to see if you can get the sounds to sound more like the vowels you want. To prove that I'm not crazy and this really is how you synthesize vowels, look at both:

http://auditoryneuroscience.com/topics/two-formant-artificial-vowels

(Test the vowels here and see whether they sound like your vowels. Again, you can use this chart to tweak your first and second resonant (formant) frequencies to see if you can get the vowels to sound more like the vowels you want.)

And also: On this page you can listen to samples of rule-based synthesizers through the years:

http://www.cs.indiana.edu/rhythmsp/ASA/highlights.html

Extra Credit (5 – 15 pts, depending on effort): Create a synthetic song. As an alternative, synthesize an entire word. Make sure you figure out how best to transition from one sound to another.