CS 251 Intermediate Programming
Lab 5: Piano Simulator*

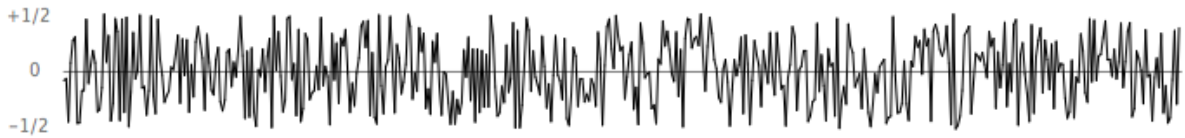
Brooke Chenoweth

Spring 2024

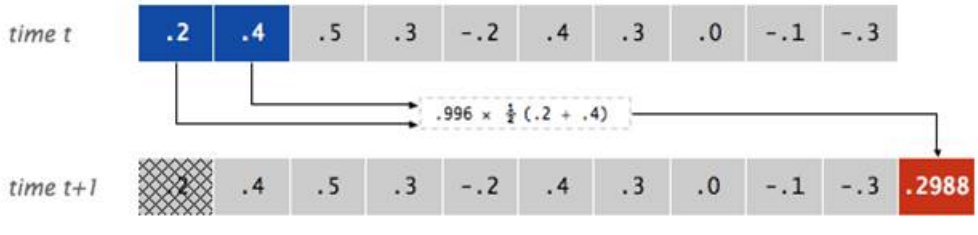
In this assignment, you will use Java collections to simulate the sound of a piano.

Karplus-Strong algorithm

When a piano wire is struck, the wire vibrates and creates sound. The vibration can be measured by sampling the displacement of the wire at equally spaced points in time. These displacement measurements can be stored digitally, say in a list or queue structure, then used to recreate the sound wave over a speaker.



For this assignment, you will store the displacement values for a piano wire in a queue structure. When excited, the wire can contain energy at any frequency. This is modeled by assigning the queue to contain random real numbers between -1/2 and +1/2. After the wire is struck, it vibrates, causing a displacement that spreads wave-like over time. The Karplus-Strong algorithm simulates this vibration using a fairly simple update process: it repeatedly deletes the first sample from the queue and adds to the end of the queue the average of the first two samples, scaled by an energy decay factor (0.996 in this example).



the Karplus-Strong update

*Based on http://nifty.stanford.edu/2018/reed-nifty-remixes/Guitar/Piano.html

This simple algorithm provides an effective model of the wire vibration due to two features: the queue feedback mechanism and the averaging operation.

- *The queue feedback mechanism.*

  The queue models the medium (a wire fixed at both ends) in which the energy travels back and forth. The length of the queue determines the fundamental frequency of the resulting sound. Sonically, the feedback mechanism reinforces only the fundamental frequency and its harmonics (frequencies at integer multiples of the fundamental). The energy decay factor (.996 in this case) models the slight dissipation in energy as the wave makes a roundtrip through the wire.

- *The averaging operation.*

  The averaging operation serves as a gentle low pass filter (which removes higher frequencies while allowing lower frequencies to pass). Because it is in the path of the feedback, this has the effect of gradually attenuating the higher harmonics while keeping the lower ones, which corresponds closely with how a struck wire actually sounds.

What do you need to do?

I have provided you with the following interfaces and classes.

- `SimpleSound.java` – Class encapulating some of the Java sound library
- `MusicString.java` – Interface representing a single vibrating instrument string
- `PianoTest.java` – Testing class that will play some music on your piano.

Implement a Piano

You will write a `Piano` class. Inside this class, you will have an inner `PianoString` class.

`Piano` class

- You should have two private member variables in this class.
  - A `double` value for the decay factor.
  - A map of notes (represented as text strings) to piano string objects. (Something like
    `private Map<String, PianoString> stringMap;`
- The constructor should set the delay factor and initialize an empty string map. (We'll add the strings with a separate method in our tests.)
- The `addStringForNote` method takes a String for the note and an int for the piano string queue size, creates a new piano string object, and puts the note string key and piano string value into the map. This method returns nothing.

- The `strikeNote` method takes a String for the note, looks up the associated piano string object in the map, and calls its `strike` method. This method returns nothing.
- The `sampleAll` method calls the `sample` method on all of the piano strings in this piano and returns the sum of the values returned.

`PianoString` class

The `PianoString` class should implement the provided `MusicString` interface.

- This class contains a queue of double values representing the displacement of the vibrating string.
- The constructor takes a queue size and creates a queue of that size filled with zeros.
- The `strike` method fills the queue with random values between -0.5 and 0.5, without changing the length of the queue.
- The `sample` method performs one update step of the Karplus-Strong algorithm. It will compute the average of the first two values in the queue multiplied by the decay factor (from the outer `Piano` object), add that new value to the end of the queue, and return the value removed from the front of the queue. The length of the queue should remain the same after this method is complete.

Turning in your assignment

Once you are done with your assignment, use Canvas to turn in `Piano.java`. You should not have changed any of the files I provided you, so do not submit them.