Homework Turnin

Name:	Ameya Singh
Account:	ameyas (ameyas@uw.edu)
Student ID:	1868457
Section:	AQ
Course:	CSE 143 18au
Assignment:	a2
Receipt ID:	df469f49c9f96b2ded5a1871bc45f700

Warning: Your turnin is 1 day late. Assignment a2 was due Thursday, October 11, 2018, 11:30 PM.

Turnin Successful!

The following file(s) were received:

Guitar37.java (2938 bytes)

```
1. /**
2.  * This class allows for creating a multi string Guitar using the
3.  * GuitarString class.
4.  *
5.  * @author Ameya Singh, CSE143 A, TA: Soham P.
6.  */
7. public class Guitar37 implements Guitar {
8.     /**
9.      * Holds the String of acceptable inputs
10.     */
11.    public static final String KEYBOARD =
12.            "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' ";  // keyboard layout
13.    /**
14.     * Holds the total number of GuitarStrings in the class
15.     */
16.    public static final int NUM_STRINGS = 37;
17.
18.    /**
19.     * Array of all GuitarStrings
20.     */
21.    private GuitarString[] guitarStrings = new GuitarString[NUM_STRINGS];
22.    /**
23.     * Holds the count of number of tic() preformed
24.     */
25.    private int tics;
26.
27.    /**
28.     * Constructs a new Guitar37 with all 37 GuitarStrings
29.     */
30.    public Guitar37() {
31.        for (int i = 0; i < NUM_STRINGS; i++) {
32.            double frequency = 440.0 * Math.pow(2, (i - 24.0) / 12.0);
33.            guitarStrings[i] = new GuitarString(frequency);
34.        }
35.        tics = 0;
36.    }
37.
38.    /**
39.     * Plucks the GuitarString of specified pitch
40.     *
41.     * @param pitch Pitch value of the GuitarString to play
42.     *              Valid values are between -24 and 12, inclusive
```

```
     */
    public void playNote(int pitch) {
        if (pitch >= -24 && pitch <= 12) {
            guitarStrings[pitch + 24].pluck();
        }
    }

    /**
     * Returns if the specified char key is valid and corresponds to an
     * existing GuitarString
     *
     * @param key Char to be compared against valid input keys
     * @return Returns true if the char is valid
     * Returns false if the char is not valid
     */
    public boolean hasString(char key) {
        return KEYBOARD.indexOf(key) != -1;
    }

    /**
     * Plucks the GuitarString that corresponds to the specified character
     *
     * @param key char representing GuitarString to be played
     * @throws IllegalArgumentException thrown if the specified key to be
     *                                  played is not a valid input
     */
    public void pluck(char key) {
        int index = KEYBOARD.indexOf(key);

        if (index == -1) {
            throw new IllegalArgumentException();
        }

        guitarStrings[index].pluck();
    }

    /**
     * Samples all the GuitarStrings present
     *
     * @return Sum of sample() of all contained GuitarStrings
     */
    public double sample() {
        double sample = 0;
        for (GuitarString string : guitarStrings) {
            sample += string.sample();
        }
        return sample;
    }

    /**
     * Calls tic() on all contained GuitarStrings
     */
    public void tic() {
        for (GuitarString string : guitarStrings) {
            string.tic();
        }
        tics++;
    }

    /**
     * Returns the current time
     *
     * @return Returns the number of times tic() has been called
     */
    public int time() {
        return tics;
    }
}
```

GuitarString.java (2225 bytes)

```
import java.util.*;

/**
 * GuitarString models the vibration of a guitar string at a given frequency
 *
 * @author Ameya Singh, CSE143 A, TA: Soham P.
 */
```

```java
public class GuitarString {
    /**
     * Represents the current ring buffer of the string
     */
    private Queue<Double> ringBuffer = new LinkedList<>();

    /**
     * Holds the constant sampling rate
     */
    public static final int SAMPLE_RATE = StdAudio.SAMPLE_RATE;
    /**
     * Holds the constant energy decay factor
     */
    public static final double ENERGY_DECAY_FACTOR = 0.996;

    /**
     * Creates a new GuitarString whose ring buffer size is based the given
     * frequency
     *
     * @param frequency Frequency to create GuitarString of
     */
    public GuitarString(double frequency) {
        int capacity = (int) Math.round(SAMPLE_RATE / frequency);

        if (capacity < 2 || frequency <= 0) {
            throw new IllegalArgumentException();
        }

        for (int i = 0; i < capacity; i++) {
            ringBuffer.add((double) 0);
        }
    }

    /**
     * This constructor is used for testing
     * Creates a new GuitarString whose ring buffer represents the contents of
     * the passed array
     *
     * @param init Values to initialize ring buffer to
     */
    public GuitarString(double[] init) {
        if (init.length < 2) {
            throw new IllegalArgumentException();
        }

        for (int i = 0; i < init.length; i++) {
            ringBuffer.add(init[i]);
        }
    }

    /**
     * Fills the ring buffer with white noise
     */
    public void pluck() {
        Random random = new Random();
        for (int i = 0; i < ringBuffer.size(); i++) {
            ringBuffer.remove();
            ringBuffer.add(random.nextDouble() - 0.5);
        }
    }

    /**
     * Applies one tic of the Karplus-Strong algorithm to the ring buffer
     */
    public void tic() {
        double avg = (ringBuffer.remove() + ringBuffer.peek()) / 2.0;
        ringBuffer.add(avg * ENERGY_DECAY_FACTOR);
    }

    /**
     * Samples the ring buffer
     *
     * @return Returns the first value in the ring buffer
     */
    public double sample() {
        return ringBuffer.peek();
    }
}
```