CSE 8A Programming Assignment 5

Fall 2019

Due Date: Mon. Nov 4, 11:59PM (PDT)

If you are having trouble getting started, understanding the assignment, or running into an error that you don't understand, please see the PA5 FAQ on Piazza.

Learning Goals:

- Write Java methods using arrays
- Manipulate sounds via the integer samples that represent them
- Run code using an autograder

Submission:

You will submit *two* components for this assignment.

- Submission 1: You will submit your `Sound.java` file to gradescope in the **PA5-Code** assignment. The autograder will run and give you feedback about what you got right and wrong. You may correct your code and resubmit based on this feedback.
- Submission 2: You will submit a PDF document created using the submission template found here.

Overview

In this assignment, you will be creating a program that creates new sounds by digitally manipulating existing or synthesized sounds.

In order to do this, you will write several functions to create and return new integer arrays based on arrays passed in as parameters. Then in the main method, you will use these functions to create interesting sounds.

Getting Started

Unlike previous assignments, we are providing you with starter code and libraries for this assignment. Follow these instructions for getting set up.

- Download the code from here. Click on "Download All" in the top right corner. If you don't see a "Download All" button, download the files individually but **be sure** to place all files in a folder named `PA5 Starter`.

- After unzipping the file you'll find a folder named `PA5 Starter` (or after downloading the individual files and then placing them into a folder named `PA5 Starter`). All the code you write should be in this folder.
- Inside the folder, you'll find:
    - A folder named `sounds`. It will have two .wav sounds in it already. You'll use this in Step 2.
    - A file named `Sounds.java`.
        - **This is the only file you'll make changes in**. In the file, you will find the 4 unimplemented methods you are responsible for writing. Please **do not change** the function headers or the other code that is already in the file for you.
        - You'll also see a line of code `CSE8ALib lib = new CSE8ALib();`. `CSE8ALib` is a library that provides the helper functions you will need to complete this assignment. You can use the functions provided in the library in the following manner: `lib.exampleFunc();` Look at the documentation to determine which functions you want to use.
        - At the bottom of the file, you'll find a function `changeVolume(int[], double)`. Use this function as an example of how you can create and manipulate sounds. Also, if your sounds end up being too loud or too soft, you can use this change the volume of the sound.

How to compile and run your code:

Because we are providing you with various libraries that your code will use, compiling and running your code will be slightly more challenging. During the compilation process, we must tell Java where to find the resources (libraries and packages) it needs to run. We tell this to Java by including a `classpath` flag (`-cp`) in the following manner. Start at the project's root directory (be in the folder you downloaded):

For MacOS/Unix:

```
javac -cp lib/*:. Sounds.java
```

For Windows:

```
javac -cp lib/*;. Sounds.java
```

This will compile your code and you should now see a file named `Sounds.class`. This means that your code has been compiled successfully and now you can run your code using:

For MacOS/Unix:

```
java -cp lib/*:. Sounds
```

For Windows:

```
java -cp lib/*;. Sounds
```

This will run your code. Remember that **every time** you change your code you **must** recompile your code as in the steps described above.

Part 1: Implementing methods to manipulate sounds.

In this section you will implement four methods to generate and manipulate sounds, which are represented as arrays of integers, as described in class.

Although these methods will ultimately be used on sound arrays, they should work for any array of integers, and you will test them on **very small** integer arrays and then show us the results of these tests.

Implement the following methods in the provided `Sounds.java` file. Function headers and empty function bodies (that return dummy values) are already provided in that file.

crop

`crop` takes three parameters: an integer array `sound` representing a sound, an integer `start_index`, and an integer `end_index`. It returns a new integer array containing the samples from `sound` starting at index `start_index` and ending at the sample immediately before `end_index`. (You can assume that `end_index > start_index` and that all indices greater or equal to `start_index` and less than `end_index` exist in the `sound` array).

For example, after the following code executes:

```
// Shorthand way to create a new array containing these 6 elements.
int[] inputSound = {3, 10, 5, 20, 9, 6};
int[] cropped = crop( inputSound, 2, 5 );
```

`inputSound` should contain its original array of ints and **`cropped`** should be an array of length 3 containing the values 5, 20, 9.

concatSounds

`concatSounds` takes two parameters: two int arrays representing two sounds. It returns a new sound (array of ints) that contains all of the samples from the first sound followed by all of the samples from the second.
For example, after the following code executes:

```
// Shorthand way to create two new arrays containing these 6
elements.
int[] inputSound1 = {3, 10, 5};
int[] inputSound2 = {7, -50, 13};
int[] concatenated = concatSounds( inputSound1, inputSound2 );
```

`concatenated` should be an array of length 6 containing the values 3, 10, 5, 7, -50, 13.

sineSound

`sineSound` takes three inputs: a length in `samples`, a `frequency` in oscillations per second, and an `amplitude`. It should return a sound (array of ints) that has a length of `samples`. Each entry in the array should have the value given by $sin(\frac{2\pi i}{samplerate} * frequency) * amplitude$, where i is the index of the entry.
For example, after the following code executes:

```
int[] sineWave = sineSound(10, 10000, 150);
```

`sineWave` should be an array of length 10 with values 0, 43, -82, 115, -138, 149, -147, 133, -108, 73.

Notes:

- You can find the value of `PI` in the `Math` library. This library has been imported for you at the top of `Sounds.java`. To get the value of `PI`, use `Math.PI`.
- If your array contains mainly `0`s, increase the value of your amplitude.
- You can find the `sin` function in the CSE8ALib, use it as such:
  `lib.sin(` $\frac{2\pi i}{samplerate} * frequency$ `).`
- This function will return values of type `double`, so you'll need to convert them to values of type `int` by using CSE8ALib as such:
  `lib.doubleToInt(lib.sin(` $\frac{2\pi i}{samplerate} * frequency$ `) *` $amplitude$ `)).`

addSounds

`addSounds` takes two arrays of ints representing sounds that are assumed to be of the same length and returns a new sound (array of ints) where the value at each index of the new sound is the sum of the values of both input sounds at that index.
For example, after the following code executes:

```
// Shorthand way to create two new arrays containing these 6
elements.
int[] inputSound1 = {3, 10, 5};
Int[] inputSound2 = {7, -50, 13};
int[] added = addSounds( inputSound1, inputSound2 );
```

`added` should be an array of length 3 containing the values 10, -40, 18.

Testing

Demonstrate your methods working on **three examples for each method (including the tests provided for you).** You should fill in the test function (which is called from main) with these tests. We have provided code that runs each of the tests given above. Notice that in each case the input arrays are created, the method is run, then the expected values of the arrays are printed along with the actual values of the arrays so we can visually compare the actual output to the expected output. You should use this same approach in the tests you add.

In the Google doc template, copy and paste your complete test method as well as the output from running the test method.

Then explain why you selected each additional test and how you know your code is working correctly.

Part 2: Exploration

At any time, you can use the function `lib.explore(int[])` to see and interact with your sound.

After you are sure the functions above are working, comment the call to test in your main method. You will now use main to implement your exploration of sounds.

In your main method, **write the code that creates an interesting sound**. Here are the requirements:

- You must call at least *two* of the methods you implemented in part 1.
- Your created sound must be at least 1 second long.
- Your created sound must be some kind of combination of at least two other sounds. These sounds can be created from the sineSound method or they can be loaded from a file, or some combination.

Important notes:

- The legal range of a sound sample value is between -32768 and 32767. If your values go outside that range, the sound may sound distorted. You can use our provided `changeVolume` function to reduce the values so they stay inside the range (or you can write your own).
- There are a number of places online where you can find sound files that are free and that you are allowed to re-use with permissive licensing:
    - The BBC sound effects library (free for educational use): http://bbcsfx.acropolis.org.uk/. A tip: you can sort these by duration to pick some short sounds.
    - The media for the Media Computation book this part of the course is based on (http://coweb.cc.gatech.edu/mediaComp-teach#Java), see the link for mediasources.zip
    - freesound.org (downloading files requires creating an account): https://freesound.org/

  Please add a comment in your code giving credit for each source you use.
- You can (and should) use the `lib.play` method to hear the sound you created!

Here are some ideas for what you might do in this part:

- Create a chord by generating sounds at different frequencies and adding them together.
- Crop portions of sounds and splice them together.

- Overlay music on top of a conversation.
- Whatever else you find interesting.

When you are happy with your sound, copy your complete main method into the Google doc template. Then describe what sound you created and what you liked and didn't like about the process (see template for the specific prompt).

Part 3: Reflection

Once you have finished and submitted your assignment, fill out the reflection form here. Don't forget that EACH STUDENT must fill out their own reflection to get credit.

Star points

Star points will be given for explorations that are particularly creative or ambitious. Feel free to implement additional functions, or do creative things in your main method.

Submission Instructions

The submission for this assignment will be a bit more complicated than those for previous assignments. Please read all the instructions. You will submit **two** separate portions: your code and a PDF write up.

Uploading code:

1) Go to Gradescope
2) Select **PA5-Code** and upload your `Sounds.java` file

Submitting your PDF Write up:

1) Download the template here.
2) Submit this PDF as you have in previous assignments