PSA7: Pictures and Sounds

Read all instructions in this document before starting any coding!
START EARLY, START OFTEN!

Due: Sunday, November 26th, 11:59pm

Overview: In this assignment you will be able to complete an updated version of one method from PSA4, and also start to generate digital sounds using Java!

As described in class, **you may complete at most 4 of the quarter's programming assignment using pair programming.** If you have decided to work in pairs, please review the guidelines for pair programming. In addition, note the following details about working with a partner:

- For this assignment, if you decided to work in pairs, find a partner before you start on the assignment. Or else, you have violated the policy on pair programming.
- You will submit only ONE version between the two of you though your group can submit as many times as you want to.

Helpful Information:

Online Communication: Using Piazza, Opening Regrade Requests

Getting Help from Tutor, TA, and Professor's Hours

Lab and Office Hours (Always refer to this calendar before posting.)

Academic Integrity: What You Can and Can't Do in CSE 8A

Remote Lab Access and File Transfer Guide or **Yingjun's SSH and VNC post**

Picture Documentation **and** **Pixel Documentation**

Table of Contents:

How to Get Started:

These instructions assume you are working in the lab machines in **B230**. We STRONGLY recommend that you work there (**or in ANY OTHER LAB in the same building, they all run CentOS and have the software you need**), but if you want to get started on your own machine, there are some additional steps you can check on Piazza.
We also assume that you have already set up the programming environment (Dr. Java). If you have not, see PSA0 for more directions.

1) Open a terminal
2) Copy the following lines, paste them on the terminal and then press enter

```
cd ~
mkdir psa7
cp ~/../public/psa7/* ~/psa7
cd psa7
ls
```

3) Verify that the '**PSA7PictureTester.java**', '**Sound.java**', '**GuitarHero.java**', '**noise440.wav**', and '**sound440.wav**' files names are shown as the result of the last command. If these commands don't work for whatever reason, the required files can be found in the PSA7.zip.

Following these procedures, your file will be named correctly and be located in the correct place. For every assignment you need to follow correct naming assignments in the correct locations to get full credit for your work.

4) You should also copy your Picture.java from PSA4 into the psa7 folder. A sample command to do so is the following. This command might not work if your psa4 folder doesn't exist anymore. Ask a tutor for help if you can't copy your Picture.java from PSA4 into your PSA7 folder.

```
cp ~/psa4/Picture.java ~/psa7/Picture.java
```

5) Proceed to start your programming assignment.

Problem: Pictures and Sounds (20 points)

Overview:

In this assignment you will be writing a program using your flipHorizontalRectangle method from PSA4 to handle edge cases where the square is out of bounds, then you will be generating cool guitar sounds using an algorithm, Karplus-Strong algorithm from Physics!

This PSA is worth 20 points total, allocated as follows :

- Part A: Picture flipping, within bounds (**5 points**).
- Part B: Guitar Hero (**11 points**).
- Style: Comments and Style (**2 points**)
- Program Description (**2 points**).

Part A: Picture Flipping, again! (5 points):

In first part of this assignment, you will revisit picture flipping that you did in PSA4. Your task in part A is to write a flipping method similar to the one you wrote in PSA4, except it has to handle edge cases, and the (x,y) **location is changed from center of a region to the bottom left corner of a region**. You will use the main method of the PSA7PictureTester class (starter code provided, including code that reads input from the user) to test your method by doing the following:

1. Prompts the user to select a picture and create a Picture object. It then shows the original picture.
2. Prompts the user to enter an (x, y) coordinate and a size. See example output
    a. After each x, y coordinate and size entered, perform a horizontal flip on that part of the image and return a new Picture.
    b. If the region to be flipped is out of bounds, only flip the region that is within bounds of the picture, ignoring the regions that go out of bounds.
        i. Hint: you can use some algebra and checks in your looping to ignore pixels that are out of bounds, or learn the dimensions that should be flipped, the **continue** statement in Java might help.
        ii. Remember to keep track of the actual size, so that coordinate (0,0) → (0,5) is actually size 6. For example, if you want to flip a region and the edge pixel of that region is 10, and your start pixel is 5, the size of the region is 10 - 5 + 1.
        iii. If the initial point implies a square that is completely out of bounds of the picture, the method should just return the picture with nothing flipped.
    c. The x and y coordinate are the **bottom left corner** of the region to be flipped
    d. **DO NOT HARDCODE ANYTHING** that will only make your code working for a specific image. It will not pass our tester and you will lose points!
    e. Method Signatures MUST USE THESE ONES:

i. `public Picture flipHorizontalSquare(int x, int y, int size)`
ii. Note that it returns a picture, this should work like your Chromakey methods. It shouldn't change the calling object. If you don't remember what this means, refer to PSA5 writeup
iii. You can reuse your flipHorizontalRectangle code, but there will be some modifications, including a uniform size instead of width and height, the meaning of (x,y) in the parameters, and returning a new Picture that is a modified copy of the original Picture.
iv. This is the only method we will look for in PIcture.java, you can get rid of the other ones if you wish

You will use your PSA4 Picture.java code to write this program. Copy Picture.java from PSA4 to your PSA7 folder. If there were any errors in your flipping methods for that PSA you should fix them now.

Example Output:

More clarification

Here is an example of this program in action. Black text is what the program displays, the blue text is entered by the user [The program opens a dialog box for the user to select a Picture file, then displays the selected picture. Suppose the image to be flipped is of size 250 by 225]

```
Picture loaded with width=250 and height=225
Please enter the x, y coordinates of lower left corner of the box to flip horizontally, x
first:
100
100
Enter the size of the box to flip:
50
```

[The program returns a new picture showing the horizontally flipped with lower left corner 100, 100 and size 50]

```
Please enter the x, y coordinates of lower left corner of the box to flip horizontally, x
first:
225
100
Enter the size of the box to flip:
50
```

[The program returns a new picture showing the horizontally flipped with lower left corner 225, 100 and size **25x50**, flipped horizontally]

```
Please enter the x, y coordinates of lower left corner of the box to flip horizontally, x
first:
200
25
Enter the size of the box to flip:
50
```

[The program returns a new picture showing the horizontally flipped with lower left corner 200, 25 and size **50x25**, flipped horizontally as if you called flipHorizontalRectangle(200, 25, 50, 25)]

```
Please enter the x, y coordinates of lower left corner of the box to flip horizontally, x
first:
200
25
Enter the size of the box to flip:
25
```

[The program returns a new picture showing the horizontally flipped with lower left corner 200, 25 and size 25.]

```
Please enter the x, y coordinates of lower left corner of the box to flip horizontally, x
first:
-25
-15
Enter the size of the box to flip:
10
```

[The program returns the original picture.]

Part B: Guitar Hero (11 points)

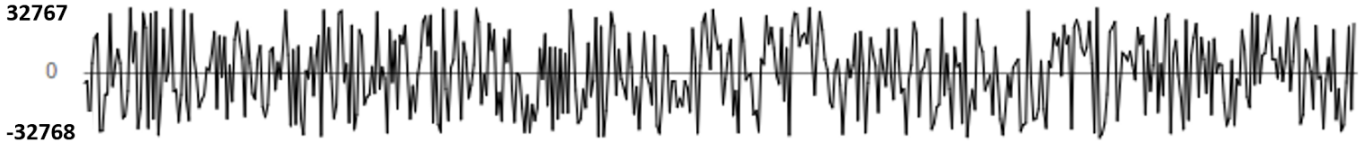
In this program, we will implement a couple methods in Sound.java to simulate plucking a guitar string using the Karplus-Strong Algorithm. This algorithm played a seminal role in the emergence of physically modeled sound synthesis (where a physical description of a musical instrument is used to synthesize sound electronically). *This part of the PSA is adopted from original work by Andrew Appel, Jeff Bernstein, Maia Ginsburg, Ken Steiglitz, Ge Wang, and Kevin Wayne from Princeton University.*

1. Physics to generate sound

First, let's talk about some physics on how to **simulate the plucking a guitar string.** When a guitar string is plucked, the string vibrates and creates sound. The length of the guitar string determines its *fundamental frequency* of vibration. We model a guitar string by sampling its *displacement* (i.e. the amplitude of sound, varying between -32,768 and 32,767) at n equally spaced points in time. The integer n equals the *sampling rate* (22,050 Hz in our case) divided by the desired fundamental frequency, rounded ***up*** to the nearest integer.

A sound wave

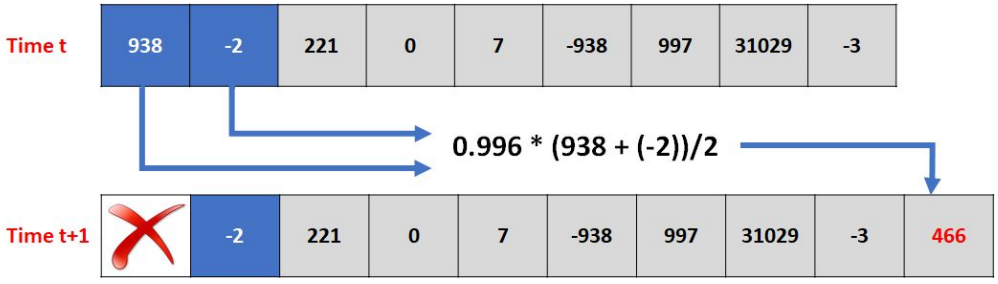
Plucking the string. The excitation of the string can contain energy at any frequency. We simulate the excitation with *white noise*: set each of the n SoundSamples to a random integer between –32,768 and +32,767.



A generated sound wave

The resulting vibrations. After the string is plucked, the string vibrates. The pluck causes a displacement that spreads wave-like over time. The *Karplus–Strong* algorithm simulates this vibration by maintaining a *ring buffer* of the n SoundSamples**: the algorithm repeatedly deletes the first sample from the ring buffer and adds to the end of the ring buffer the average of the deleted sample and the first sample, scaled by an *energy decay factor* of 0.996.**

For example:



The rotation of the SoundSample array is called a ring buffer as it has this circular behavior from the left side to the right side. So the best way to implement this rotation is to use a single array and operate on it without having to re-create new arrays. In other words, we should implement the rotation method **in place**. This is what we need to do in codes.

Why it works? The two primary components that make the Karplus–Strong algorithm work are the ring buffer feedback mechanism and the averaging operation.

The ring buffer feedback mechanism. The ring buffer models the medium (a string tied down at both ends) in which the energy travels back and forth. The length of the ring buffer 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 (0.996 in this case) models the slight dissipation in energy as the wave makes a round trip through the string.

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 to the sound that a guitar string makes when plucked.

From a mathematical physics viewpoint, the Karplus–Strong algorithm approximately solves the 1D wave equation, which describes the transverse motion of the string as a function of time.

2. Codes to implement the Karplus-Strong algorithm

The way we generate the sound is to first generate a white noise sound based on a frequency. Then use the white noise sound to generate the actual plucking sound of a guitar string.

1. Your first task is to write a method in Sound.java called whitenoise to generate a whitenoise sound. (3 points)

```
public static Sound whitenoise(int frequency)
```

This method's parameter is the frequency of the sound you want to generate, and it returns the Sound containing the white noise with the correct length. The length of the SoundSample array should be calculated as 22050/frequency (round up) where 22050 is the sampling frequency we use in our Sound class. The value of each cell in the returned SoundSample array should be randomly generated in the range of -32768 and 32767 with both boundaries included. You should use the Random class to generate random numbers. Your method should first create a sound with the correct length, modify its SoundSample array, and then return the sound.

2. The next method we will write is to generate the sound with the help of the whitenoise sound from the whitenoise method. (5 points)

```
public Sound pluck (int soundLength)
```

The parameter of the method, soundLength, is the number of SoundSamples in the actual pluck sound. It isn't related to the length of the whitenoise sound. Based on our experience, a good value to pass in is in the range of 10000 to 30000. You can experiment to see which soundLength gives you the best sound sonically. Make sure the sound neither stops prematurally nor has a long silence in the end.

The calling object of this method should be the white noise sound generated by the whitenoise method. The pluck method will generate the sound with soundLength SoundSamples by following the steps below.

1. Create a Sound, `result`, with the correct length (i.e. soundLength) and obtain its SoundSample array. Now we will have two Sounds, the one from the calling object for rotating using Karplus-Strong algorithm, and the one we just created for storing the final resulting sound.
2. for each element s in the SoundSample array of result
    a. Copy the 0^{th} element from the noise calling object's SoundSample array into s.
    b. Rotate the noise calling object's SoundSample array one position to the left and fill in the correct value on the last index (i.e. ring buffer and averaging operations, see Karplus-Strong algorithm above).
3. Return sound .

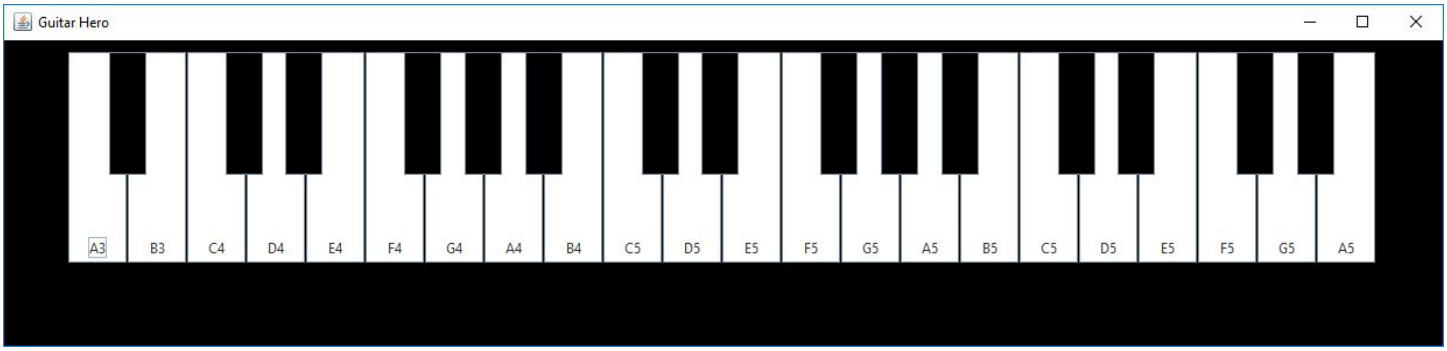
3. Testing your methods (3 points)

As what we did before for methods for Pictures, we need to test the two methods that we implemented above. Write a method that compares if two Sounds are the same, like what we have done before in PSA3 and PSA4. The method's header should be

```
public boolean sameSound(Sound s)
```

The method should return true if all `SoundSamples` from the calling object and the parameter sound are the same. Or else, it should return false. For your reference, we have provided you with a random noise sound for base frequency 440Hz, noise440.wav, and the pluck sound it generated, sound440.wav. sound440.wav has 10000 SoundSamples in it. You can use these two files to test if your pluck method generates the correct sound. You should create a main method inside your Sound.java and call appropriate methods to test out your codes. Note that sameSound method can only test if your pluck method produces the correct sound given a white noise calling object. You need to make sure that your whitenoise method is correct. You can create your own tester for the whitenoise method.

Once you are done with your code and test it properly, you can use the provided graphical user interface to play music! Open the given GuitarHero.java together with your Sound.java in Dr. Java. Compile GuitarHero.java and run it. It may take a few seconds before a piano keypad appears. You can use this keypad to play music.



You can play the different keys by tapping a key on your keyboard. The mapping is the following but it basically maps the four rows of keys on your keypad into corresponding white and black keys.

White keys from left to right

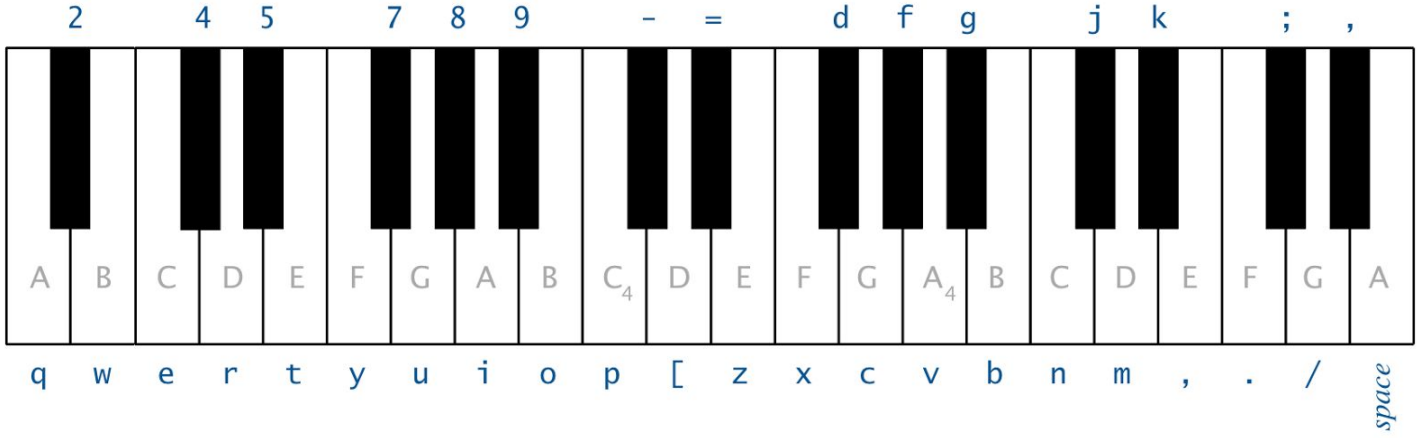
q w e r t y u i o p [→ maps to A3 to D5

z x c v b n m , . / (space) → maps to E5 to A5

Black keys from left to right

2 4 5 7 8 9 - = → maps to black keys above A3 to D5

d f g j k ; ‘ → maps to black keys above E5 to A5



Please note that the GUI only works after you have completed your Sound.java. Have fun with it and maybe you can create some thanksgiving jingles for your family!

Program Description (2 points)

Describe what your program does as if it was intended for a 5 year old or your grandmother. Do not assume your reader is a computer science major. The programs you should comment on in this segment include Sound.java and PSA7PictureTester.java. **Write this as comments at top of Sound.java file.**

Comments and Style (2 points)

Remember to include file header comments with your and your partner's name, login ID, and date in *all* your files.

We will be looking for the following additional comment/style points in your code:

- Meaningful variable names (i.e. don't name a pixel "n", "i", or "bob" or "hello". Name them something pertinent to their role, such as "sourcePixel" or "targetPixel")
- Proper code indentation. Every subsequent coding/loop block must be further indented properly. Every single code example in your textbook follows this convention, so see your book for examples
    - In Dr. Java, you can highlight everything and press tab -- it will automatically indent everything for you! Remember for future projects (cse8b, etc.), you won't have Dr. Java as your editor, and will have to do this indenting manually!
- Method header comments at beginning of methods indicating what they do at a high level.
    - Example: "*//method flips the image upside down*" or "*//method mirrors the image along the center vertical line*".
    - Include a description of the parameters and return values in the header comments (google javadoc styling if you're not sure).
- **Each code line should be no longer than ~80 characters.**
- **Do not copy-paste formatted text into your code! Apostrophes and certain characters being inside of your code will break the code! We will take points off if your code does not compile because of this!**

Star points (Optional)

In this star points, you have two choices to generate music, either through the GUI that we have provided or write a code to generate a piece of music. If you use the GUI to generate the music, please record your screen (**make sure you also open a file(.txt is fine) that has your NAME and PID on it right next to the GUI during screen record so that we know it is you otherwise you will not receive the star point**), upload a video on a video sharing service such as youtube, and put the link to your video in the very top of your Sound.java file.
If you choose to write a program to generate the music, you can use write your music as string of characters, analyze your string of characters, and generate the appropriate sound based on that character. If you write code to generate music automatically, you should include a Psa7_starpoint.java and submit it. **It is a bad idea to call the pluck method a lot of times in brute force (something we did in PSA0 when we write our name using the Turtle).**

To receive a star point, the music generated using the GUI has to fairly complicated. If you write a code to generate the music, the music can be slightly less complicated. To give you a yardstick for how complicated the music should be, here are some pieces that our head tutor (Alan Kuo) made. We would expect you to have more fun making audibly pleasing music. The goal is to have fun and enjoy the beauty of programming!

How to Turn in Your Homework

1. Place your psa7 folder in the home directory of your student account (cs8afxx).

2. Ensure that the psa7 folder contains at least all the following files: **PSA7PictureTester.java, Picture.java and Sound.java.**

3. Run the command: `cse8aturnin psa7`

4. Follow the prompts.

5. Verify that your homework was turned in with the command: `cse8averify psa7`

Refer back to psa0 turnin instructions for additional details.