
Processing Sound Ranges

Barb Ericson
Georgia Institute of Technology
July 2005

Creating a Sound Clip

- To clip the “This” out of “This is a test”.
 - Determine where it starts and stops
 - Using the sound explorer:
 - String file =
FileChooser.getMediaPath(“thisisatest.wav”);
 - Sound s = new Sound(file);
 - s.explore();

Finding the End of the This

Play before bar to check

The screenshot shows a Java application window titled 'c:/intro-prog-java/mediasources/thisisatest.wav'. The interface includes several control buttons: 'Play Entire Sound', 'Play Before', 'Play After', 'Stop', 'Play Selection', and 'Clear Selection'. Below the buttons, there are 'Start Index: N/A' and 'Stop Index: N/A' labels. The main area is a black rectangle containing a white audio waveform. A vertical cyan line is positioned at the beginning of the first significant audio pulse. An arrow points to this line with the text 'Position the bar'. Below the waveform, there are playback controls: a stop button, a previous button, a 'Current Index' text label, a text input field containing '8500', a 'Sample Value' text label, a text input field containing '-204', a next button, and a play button. An arrow points from the text 'Get the index' to the 'Current Index' label. Below these controls is a text label 'The number of samples between pixels:' followed by a text input field containing '100'. At the bottom center is a 'Zoom In' button.

Position the bar

Current Index: 8500 Sample Value: -204

The number of samples between pixels: 100

Get the index

Zoom In

To Create a Sound Clip

- Create a new Sound object
 - Of the appropriate size
 - Ending value – starting value + 1
- Loop from start to end (inclusive)
 - for (int x = start; x <= end; x++)
- Use `getSampleValueAt(index)`
 - And `setSampleValueAt(index,value);`
- Return the new sound object

Clip Method

```
public Sound clip(int
    start, int end)
{
    // calculate the number
    // of samples
    int lengthInSamples =
        end - start + 1;
    Sound target = new
        Sound(lengthInSamples);
    int value = 0;
    int targetIndex = 0;
    int i = start;

    // copy from start to end
    while (i <= end)
    {
        value =
            this.getSampleValueAt(i);

        target.setSampleValueAt(
            targetIndex, value);
        i = i + 1;
    }

    return target;
}
```

Challenge

- Create a clip of “is” from thisisatest.wav
- Determine where to start and end the clip
- Create the clip
- Write it to a file

Returning a Value from a Method

- To return a value from a method
 - Include a return statement
 - The type of the thing being returned must match the declared return type
 - The clip method declared that it returned a Sound object
 - The return statement returned the target Sound object
 - If the types don't match you will get a compile error

Splicing Sounds Together

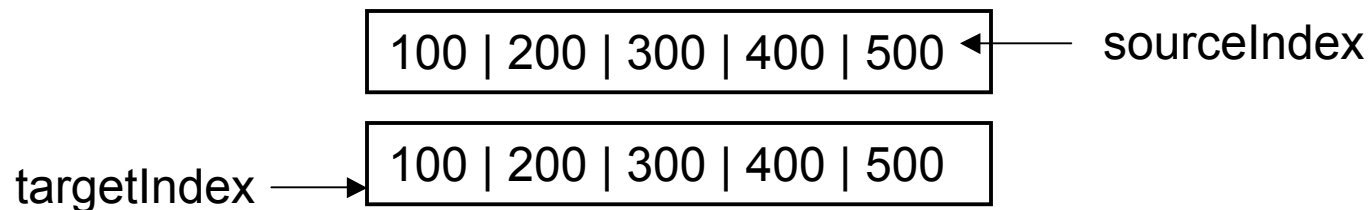
- Originally cut the sound tape into segments and then assembled them into the right order
- Easy to do digitally
- Copy more than one sound into a target sound
 - Track the source index and target index

Splice Method

```
public void splice()
{
    Sound sound1 = new
        Sound(FileChooser.getMediaPath("guzdial.wav"));
    Sound sound2 = new
        Sound(FileChooser.getMediaPath("is.wav"));
    int targetIndex = 0; // the starting place on the target
    int value = 0;
    int i = 0;
    // copy all of sound 1 into the current sound (target)
    while (i < sound1.getLength())
    {
        value = sound1.getSampleValueAt(i);
        this.setSampleValueAt(targetIndex,value);
        i = i + 1;
    }
    // NOW WHAT?
```

Reversing a Sound

- To reverse a sound
 - Create a copy of the original sound
 - `Sound orig = new Sound(this.getFileName());`
 - Then loop starting the `sourceIndex` at the last index in the source and the `targetIndex` at the first index in the target
 - Decrement the `sourceIndex` each time
 - Increment the `targetIndex` each time

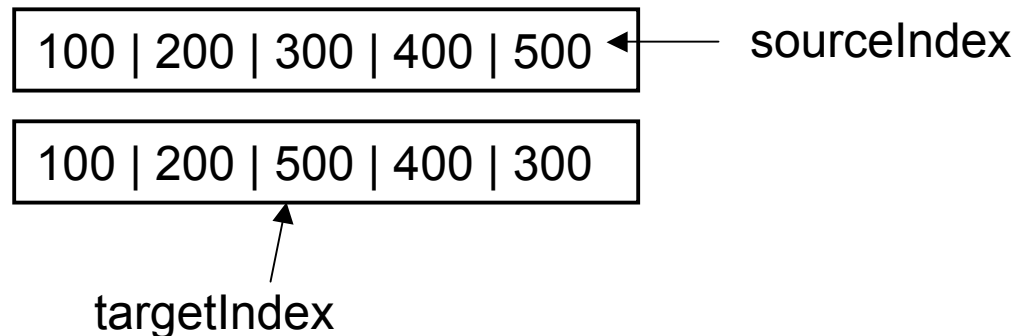


Reversing Method

```
public void reverse()  
{  
    Sound orig = new Sound(this.getFileName());  
    int length = this.getLength();  
  
    // loop through the samples  
  
}
```

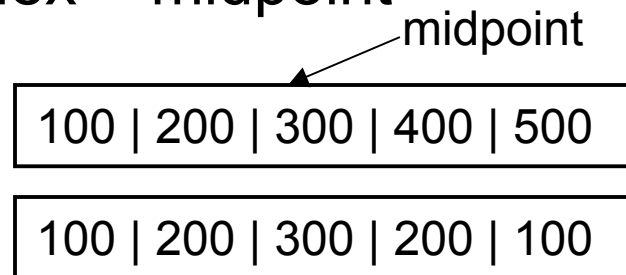
Reverse Part of a Sound Exercise

- Reverse just the second half of a sound
 - Start the `targetIndex` at the `length / 2`
 - Start the `sourceIndex` at the `length - 1`
 - Loop when the `targetIndex < length`



Mirror a Sound

- Copy the first half of the sound to the second half
 - And reverse the sounds in the second half
 - This is very similar to mirroring a picture
 - Calculate the midpoint ($\text{length} / 2$)
 - Start the index at 1 and copy from $\text{midpoint} - i$ to $\text{midpoint} + i$
 - While $\text{index} < \text{midpoint}$

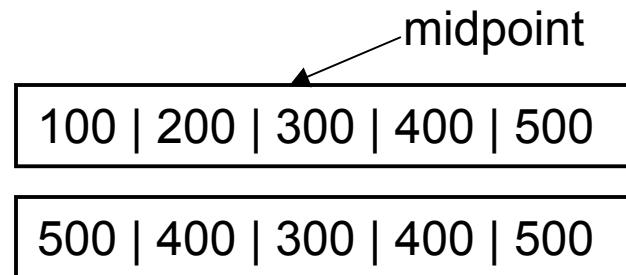


Mirror Sound Method

```
public void mirrorFrontToBack()
{
    int length = this.getLength(); // save the length
    int mirrorPoint = length / 2; // mirror around this
    int value = 0; // hold the current value
    int i = 1;
    // loop from 1 to mirrorPoint
    while (i < mirrorPoint)
    {
        value = this.getSampleValueAt(mirrorPoint-i);
        this.setSampleValueAt(mirrorPoint+i,value);
        i = i + 1;
    }
}
```

Mirror Back to Front Exercise

- Write a method to mirror from the back to the front
 - Copy the back half of the sound reversed to the front



Blend Sounds

- How do we blend two sounds?
 - Copy the first 20,000 values of sound1
 - Copy from both by adding $.5 * \text{sound1 value}$ and $.5 * \text{sound2 value}$
 - Copy the next 20,000 values of sound 2

Blend Sounds Method

```
public void blendSounds()
{
    Sound sound1 =
        new Sound(FileChooser.getMediaPath("aah.wav"));
    Sound sound2 =
        new Sound(FileChooser.getMediaPath("bassoon-
c4.wav"));
    int value = 0;

    // copy the first 20,000 samples from sound1 into target
```

Testing Blend Sounds

- String fileName =
FileChooser.getMediaPath(
 "sec3silence.wav");
- Sound target = new Sound(fileName);
- target.explore();
- target.blendSounds()
- target.explore();

Modify Blend Sounds Exercise

- Create another blendSounds method
 - That takes the file name of the sounds to blend
 - And a value to start the blend at and another to stop the blend at
 - Modify the original blendSounds method to call this one

Overloading Methods

- You can have several methods with the same name
 - As long as the parameter list is different
 - In number of parameters
 - And/or types of parameters
 - `blendSounds()`
 - `blendSound(String name1, String name2, int startBlend, int endBlend)`

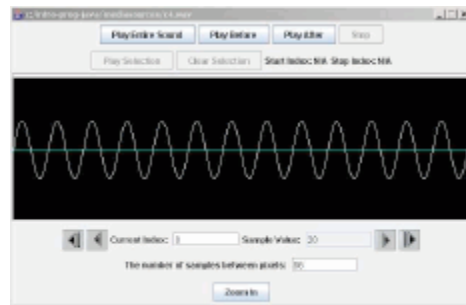
Changing the Sound Frequency

- The *frequency* of a wave is the number of cycles per second (cps), or *Hertz (Hz)*
 - (Complex sounds have more than one frequency in them.)
- Our perception of pitch is related (logarithmically) to changes in frequency
 - Higher frequencies are perceived as higher pitches
 - We can hear between 5 Hz and 20,000 Hz (20 kHz)
 - A above middle C is 440 Hz

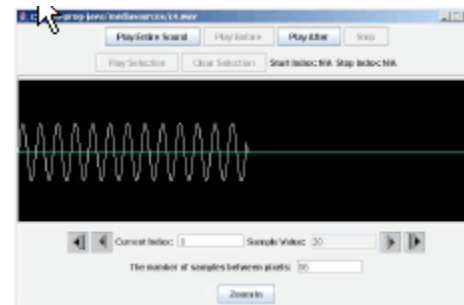


Double the Frequency

- If we take every other sample we double the frequency of the sound
 - Completes two cycles instead of one in the same time
 - It will sound higher



100 | 200 | 300 | 400 | 500



100 | 300 | 500 | 0 | 0

Double Frequency Method

```
public void doubleFreq()
{
    // make a copy of the original sound
    Sound s = new Sound(this.getFileName());

    /* loop and increment target index
    * by one but source index by 2,
    * and set target value
    * to the copy of the original sound
    */
```

Change Frequency Exercise

- Write a method that will copy each sound value 4 times to the target
 - Will the new sound be higher or lower?
- Can you make this more general?
 - By passing in the number of times to copy the source value
 - Try it with 3 times and check the index values to make sure that you are doing it right