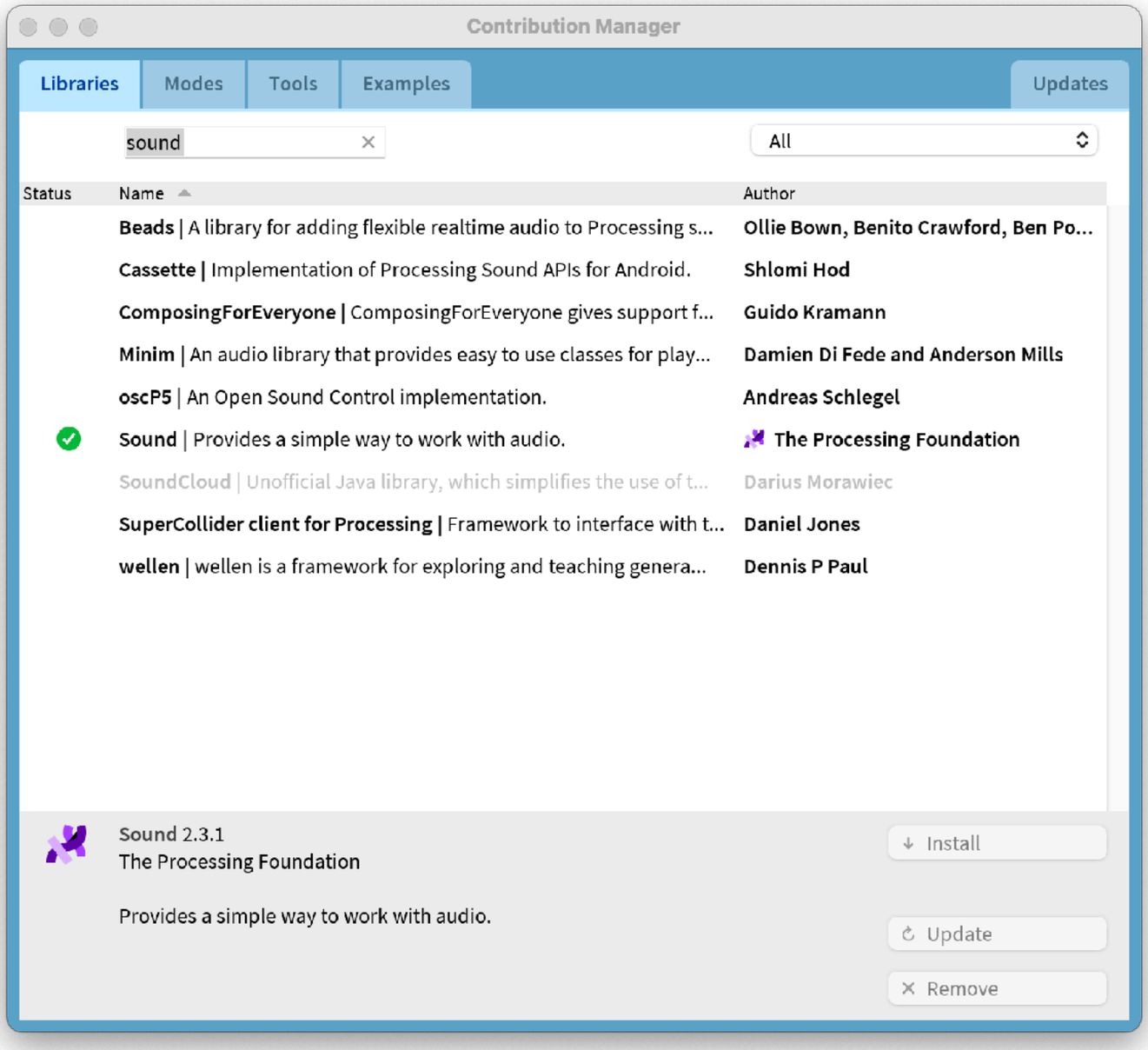
WEEK 9

AUDIO PROCESSING

Processing Sound Library

- Plays, analyzes and synthesizes sound
- Provides
  - a collection of oscillators for basic wave forms
  - a variety of noise generators
  - effects and filters to play and alter sound files and other generated sounds

To install through Library Manager:



https://github.com/processing/processing-sound

Libraries

Extend Processing beyond graphics and images into audio, video, and communication with other devices.

Core

DXF Export	Create DXF files to save geometry for loading into other programs. It works with triangle-based graphics including polygons, boxes, and spheres.
Hardware I/O	Access peripherals on the Raspberry Pi and other Linux-based computers
Network	Send and receive data over the Internet through simple clients and servers.
PDF Export	Create PDF files. These vector graphics files can be scaled to any size and printed at high resolutions.
Serial	Send data between Processing and external hardware through serial communication (RS-232).
Sound	Playback audio files, audio input, synthesize sound, and effects.
SVG Export	Create SVG files.
Video	Read images from a camera, play movie files, and create movies.

Documentation →

Package processing.sound

Full JavaDoc documentation for advanced users:
https://processing.github.io/processing-sound/processing/sound/package-summary.html

This Javadoc is only provided as reference for advanced users. For the basic documentation of the Processing Sound library with examples, please see https://processing.org/reference/libraries/sound/.

Class Summary

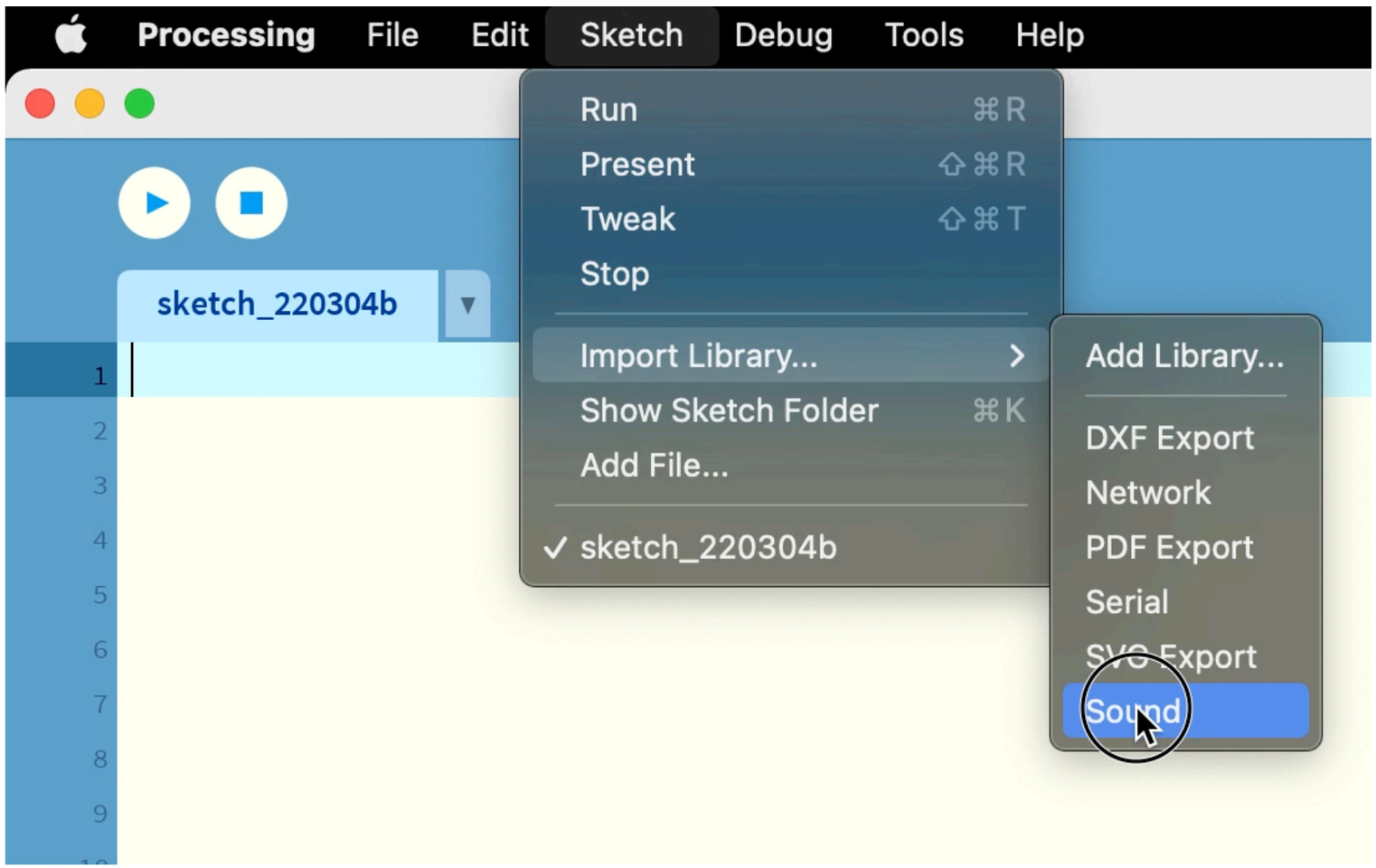
Class	Description
Amplitude	This is a volume analyzer.
AudioDevice	Deprecated.
AudioIn	AudioIn lets you grab the audio input from your sound card.
AudioSample	This class allows you low-level access to an audio buffer to create, access, manipulate and play back sound samples.
BandPass	This is a band pass filter.
BeatDetector	The BeatDetector analyzer looks for spikes in the energy of an audio signal which are often associated with rhythmic musical beats and can be used to trigger a response whenever the incoming audio signal pulses.
BrownNoise	This is a brown noise generator.
Delay	This is a simple delay effect.
Effect<EffectType extends com.jsyn.unitgen.UnitFilter>	For advanced users: common superclass of all effect types
Env	This is an ASR (Attack Sustain Release) Envelope Generator
FFT	This is a Fast Fourier Transform (FFT) analyzer.
HighPass	This is a high pass filter
LowPass	This is a low pass filter
Noise<JSynNoise extends com.jsyn.unitgen.UnitGenerator>	For advanced users: common superclass of all noise generators
Oscillator<JSynOscillator extends com.jsyn.unitgen.UnitOscillator>	For advanced users: common superclass of all oscillator sound sources
PinkNoise	This is a pink noise generator.
Pulse	This is a simple Pulse oscillator.
Reverb	This is a simple reverb effect.
SawOsc	This is a simple Saw Wave Oscillator
SinOsc	This is a simple Sine Wave Oscillator
Sound	This class can be used for configuring the Processing Sound library.
SoundFile	This is a Soundfile player which allows to play back and manipulate sound files.
SoundObject	For advanced users: common superclass of all sound sources (oscillators, noise, audio samples and even AudioIn).
SqrOsc	This is a simple Square Wave Oscillator
TriOsc	This is a simple triangle (or "saw") wave oscillator
Waveform	This is a Waveform analyzer.
WhiteNoise	This is a White Noise Generator.

Import Sound Library

- The import statement must be added in order to use the Sound Library:

```
import processing.sound.*;
```

- The keyword import is used to load a library into a Processing sketch.



SoundFile

- SoundFile player supports play back and manipulate sound files
- Supported formats: WAV, MP3 or AIF/AIFF (Audio Interchange File Format)
- playback using .play() or .loop() method

```
import processing.sound.*;
SoundFile song;
void setup() {
  song = new SoundFile(this, "file.mp3");
  song.play(); // or song.loop();
}
void draw() { // cannot skip draw()
}
```

constructor

SoundFile

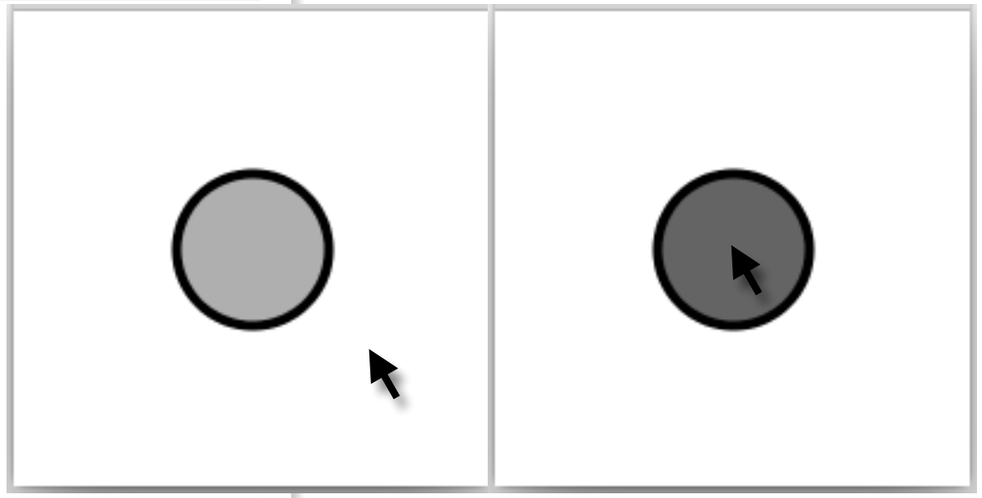
Methods

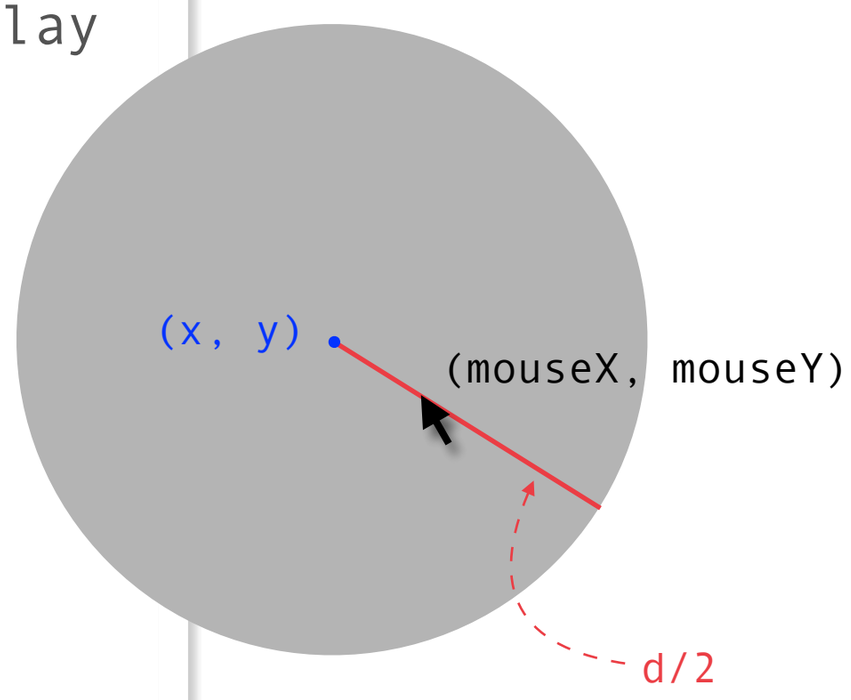
`removeFromCache()`	Remove this SoundFile's decoded audio sample from the cache, allowing it to be garbage collected once there are no more references to this SoundFile.
`channels()`	Returns the number of channels of the soundfile as an int (1 for mono, 2 for stereo).
`cue()`	Cues the playhead to a fixed position in the soundfile.
`duration()`	Returns the duration of the soundfile in seconds.
`frames()`	Returns the number of frames of this soundfile.
`play()`	Starts the playback of the soundfile.
`jump()`	Jump to a specific position in the soundfile while continuing to play (or starting to play if it wasn't playing already).
`pause()`	Stop the playback of the file, but cue it to the current position.
`isPlaying()`	Check whether this soundfile is currently playing.
`loop()`	Starts playback which will loop at the end of the soundfile.
`amp()`	Changes the amplitude/volume of the player.
`pan()`	Move the sound in a stereo panorama.
`rate()`	Set the playback rate of the soundfile.
`stop()`	Stops the playback.

Example 1

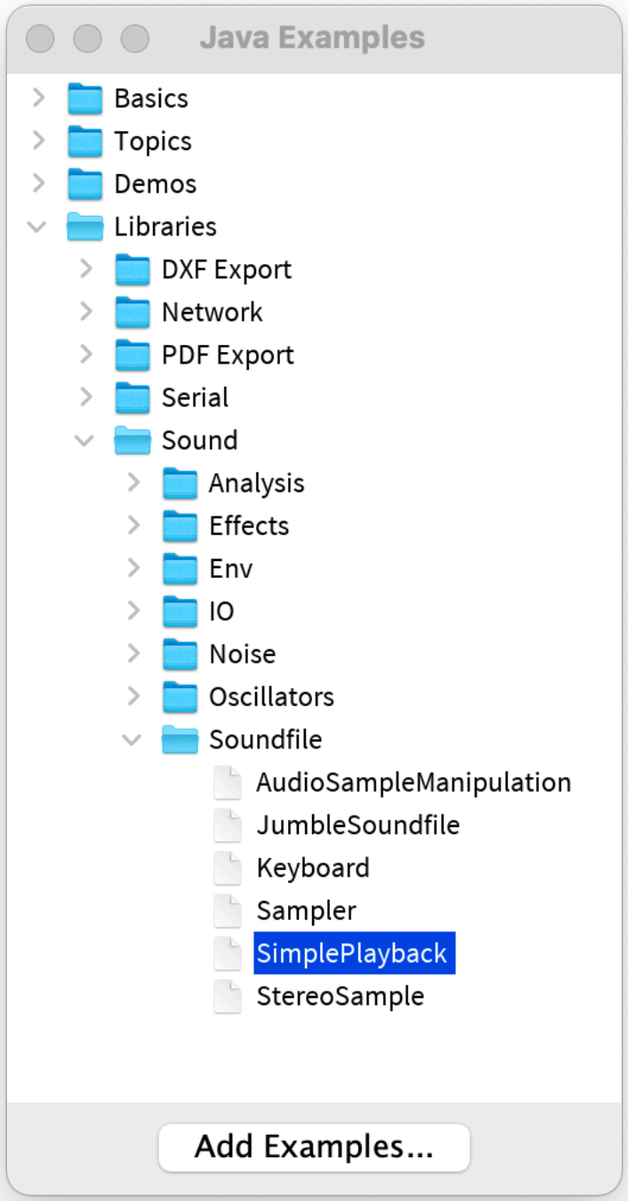
```
void draw() {
  background(255);
  if (checkInCircle())
    fill(100);
  else
    fill(175);
  ellipse(x, y, d, d);
}
void mousePressed() {
  // If the user clicks on the doorbell, play
the sound!
  if (checkInCircle()) {
    dingdong.play();
  }
}
boolean checkInCircle() {
  if (dist(mouseX, mouseY, x, y) <= d/2)
    return true;
  else
    return false;
}
```

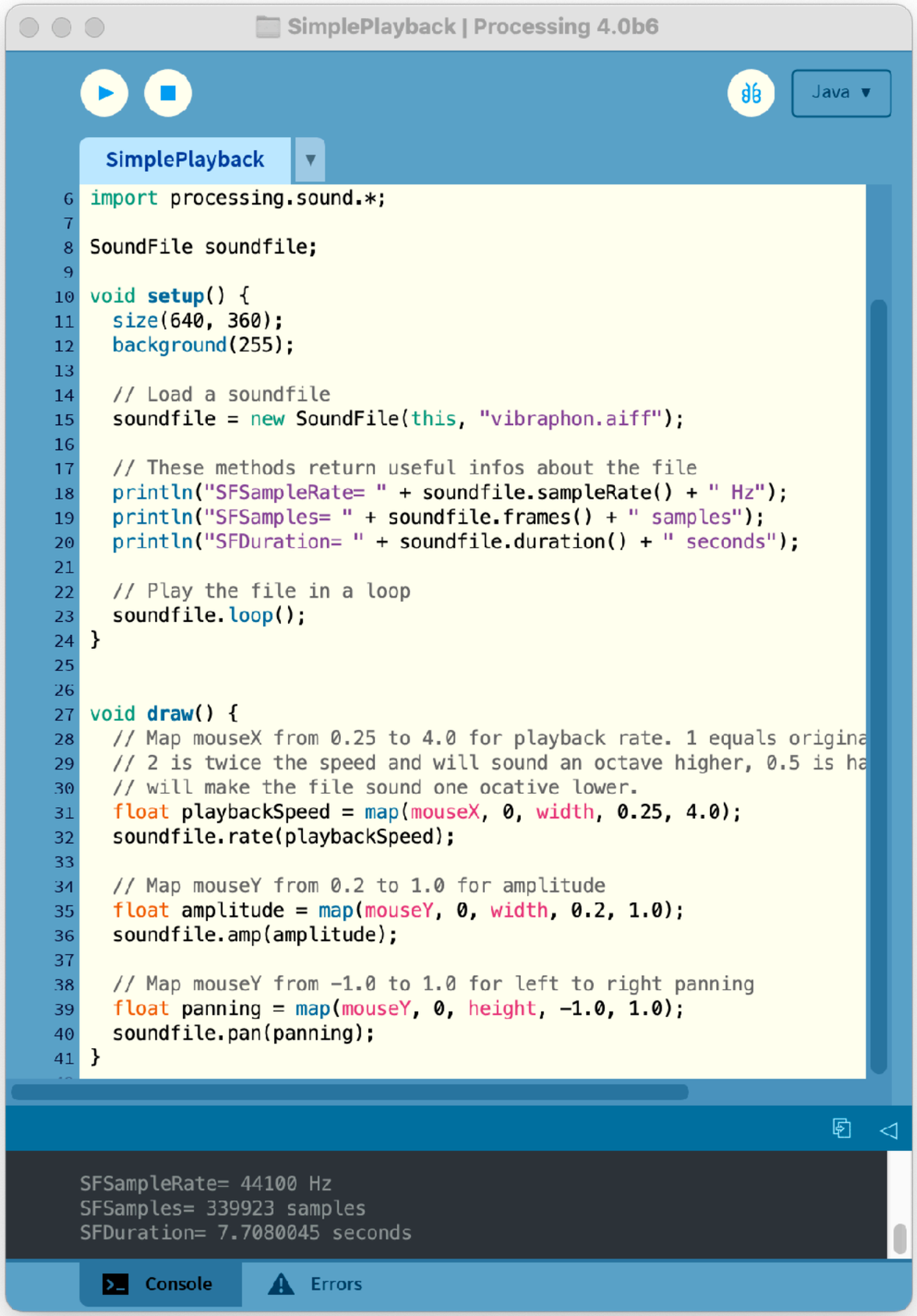
return type is boolean!



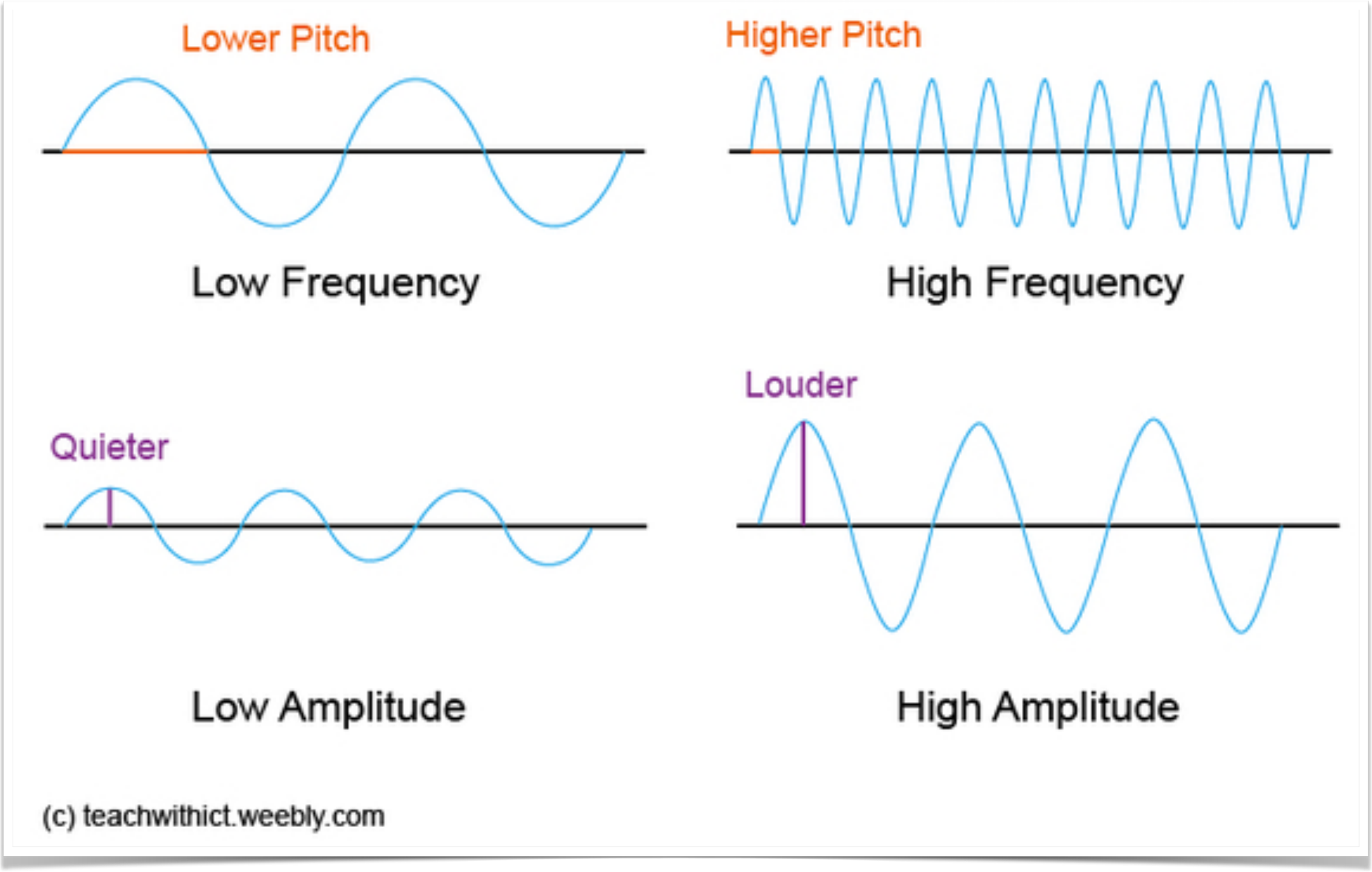


Example/libraries/Sound/Soundfile/SimplePlayback





Sound Analysis



Amplitude

which means the average value of a continuously varying function

- Volume analyser that returns the root mean square (RMS) of the amplitude of each audio block
- Declare and create an amplitude analyzer

```
Amplitude amp;
amp = new Amplitude(this);
```

constructor

- Defines the audio input of the Amplitude analyzer

```
amp.input(song); //song is an instant of SoundFile;
```

or

```
amp.input(in); //in is an instant of AudioIn;
```

- Queries a value from the analyzer and returns a float between 0 and 1

```
float amplitude = amp.analyze();
```

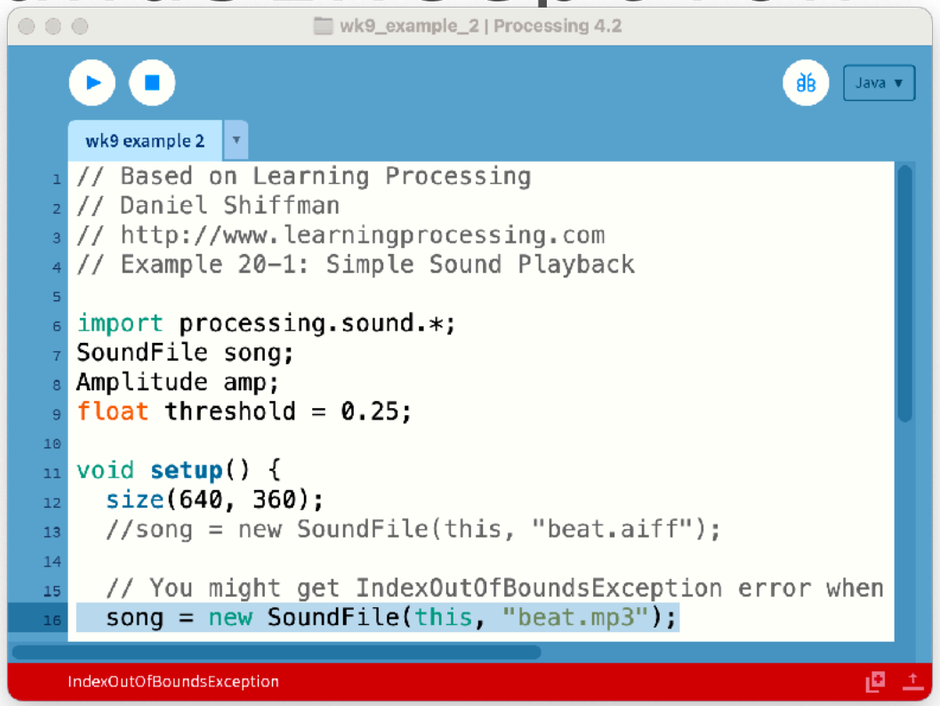
Example 2

```
import processing.sound.*;
SoundFile song;
Amplitude amp;
float threshold = 0.25;

void setup() {
  size(640, 360);
  song = new SoundFile(this, “beat.aiff");
  song.loop();
  amp = new Amplitude(this);
  amp.input(song);
}

void draw() {
  background(amp.analyze()*255);
  //if (amp.analyze()>threshold)
  //  background(255);
  //else
  //  background(0);
}
```

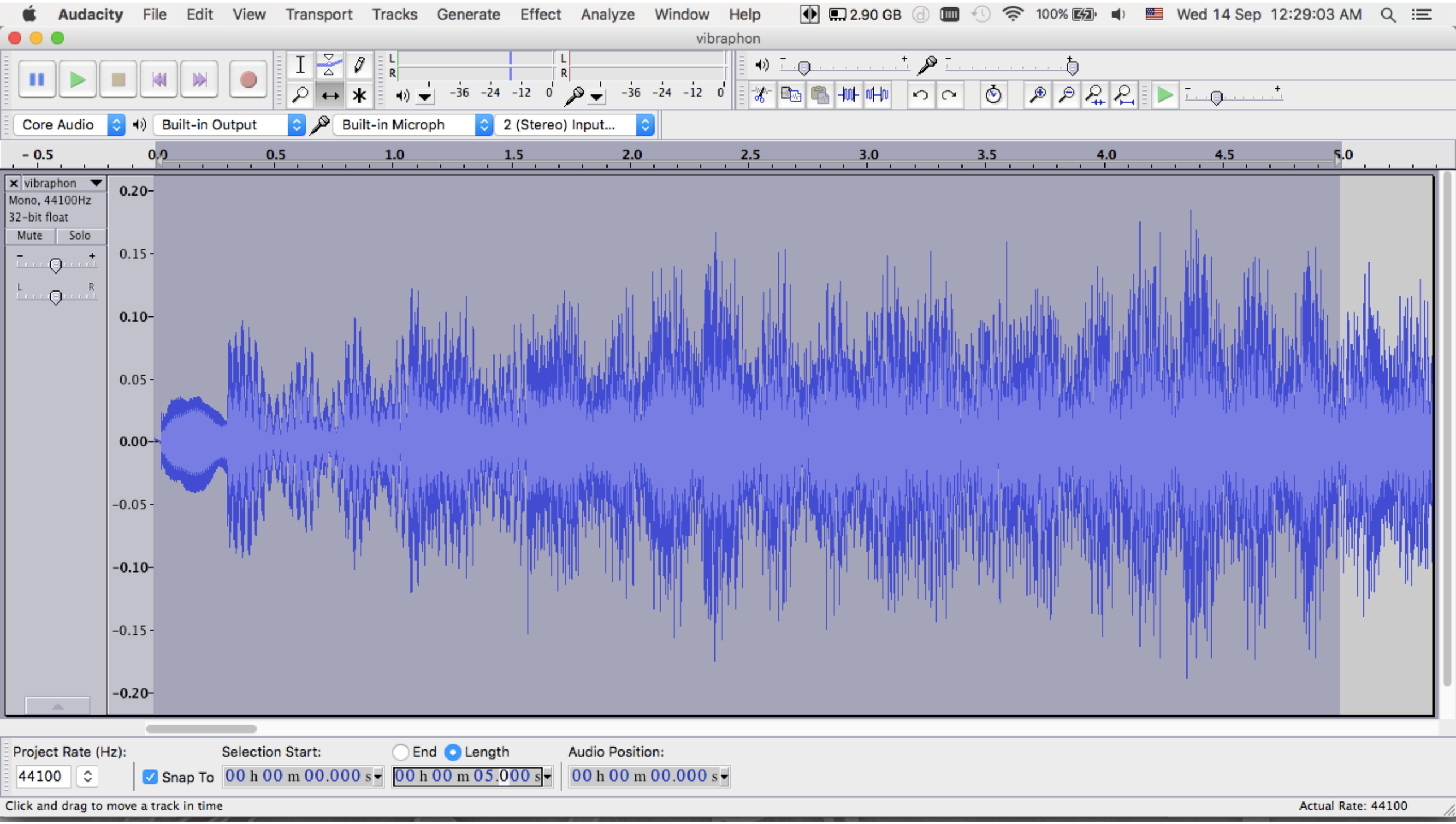
Fixing the IndexOutOfBoundsException error when loading mp3s

- Processing Sound library cannot load mono (1 channel) mp3s and will give an “IndexOutOfBoundsException” error

- Change the mono mp3 to stereo with any software such as Audacity or Adobe Audition CC will fix the issue

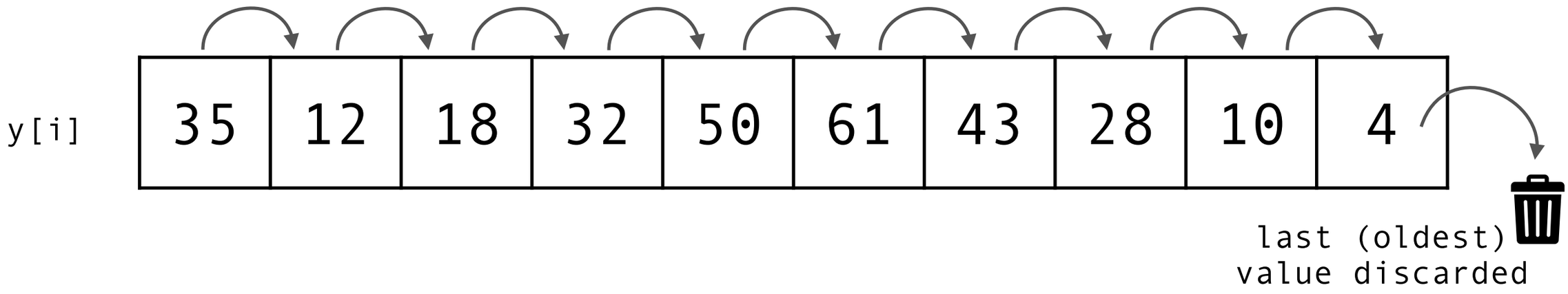


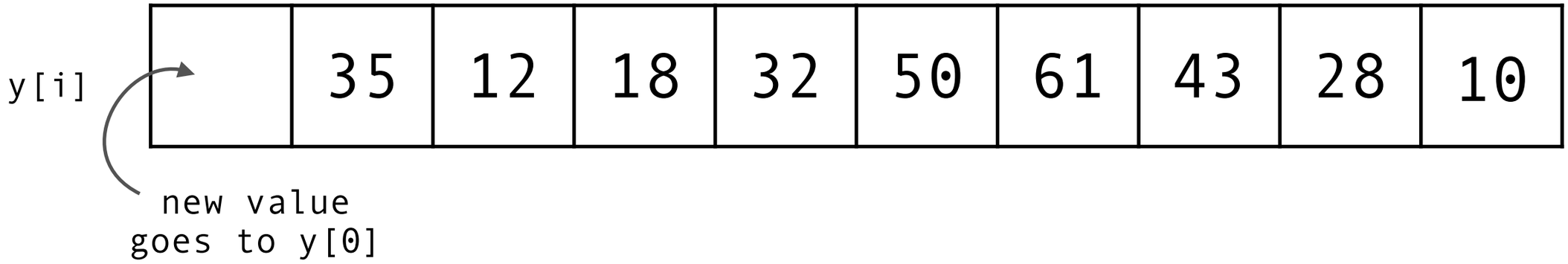
Displaying waveform



Storing Data with Arrays

Goal: Shifting values to the right






```
int[] y = {35, 12, 18, 32, 50, 61, 43, 28, 10, 4};
for (int i = y.length - 1; i > 0; i--)
  y[i] = y[i-1];
}
```

Descending order

Read the array from the end to the beginning to avoid overwriting the data

Example 3A

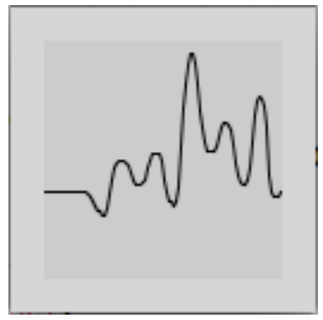
PROCESSING HANDBOOK, 2ND ED
P.421 CHAPTER 28 EXAMPLE 15

```
int[] y;
void setup() {
  size(800, 600);
  y = new int[width];
}
void draw() {
  background(204);
  // Read the array from the end to the
  // beginning to avoid overwriting the data
  for (int i = y.length-1; i > 0; i--) {
    y[i] = y[i-1];
  }
  // Add new values to the beginning
  y[0] = mouseY;
  // Display each pair of values as a line
  for (int i = 1; i < y.length; i++) {
    line(i, y[i], i-1, y[i-1]);
  }
}
```

A

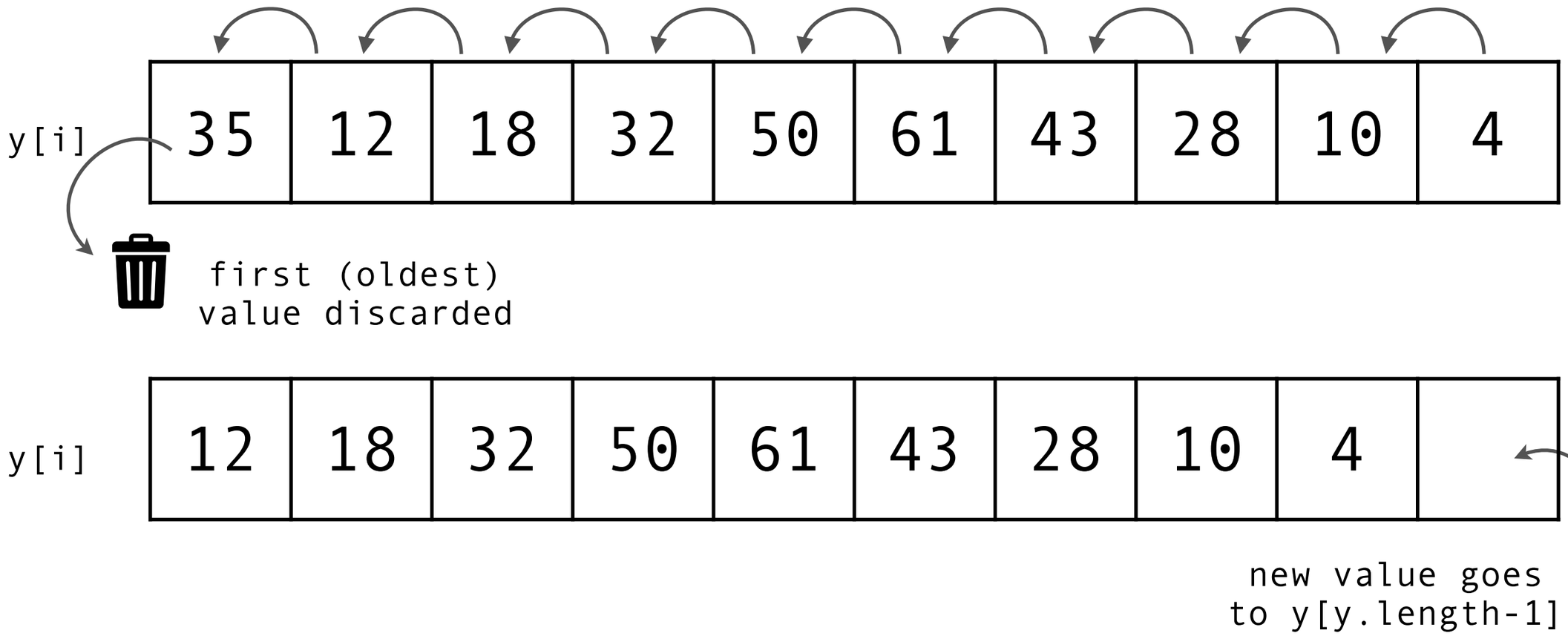
B

C



Shifting values to the RIGHT

Shifting Values to the LEFT

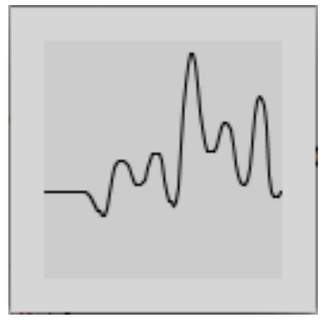



```
int[] y = {35, 12, 18, 32, 50, 61, 43, 28, 10, 4};
for (int i = 0; i < y.length - 1; i++)
  y[i] = y[i+1];
}
```

ascending order

Example 3B

```
int[] y;
void setup() {
  size(800, 600);
  y = new int[width];
}
void draw() {
  background(204);
  // Shift the values to the left
  for (int i = 0; i < y.length - 1; i++)
    y[i] = y[i+1];
  // Add new values to the end
  y[y.length-1] = mouseY;
  // Display each pair of values as a line
  for (int i = 1; i < y.length; i++)
    line(i, y[i], i - 1, y[i-1]);
}
```



A

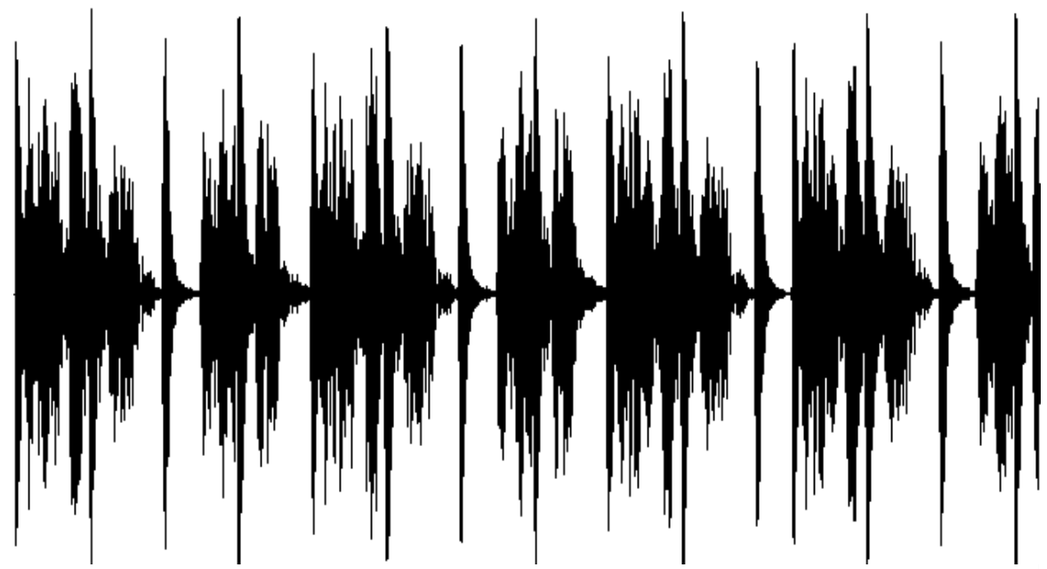
Shifting values to the LEFT

B

C

Example 3C

```
void setup() {
  size(640, 360);
  song = new SoundFile(this, "beat.mp3");
  song.loop();
  analyzer = new Amplitude(this);
  analyzer.input(song);
  // create an empty array of size equals width
  history = new float[width];
}
```

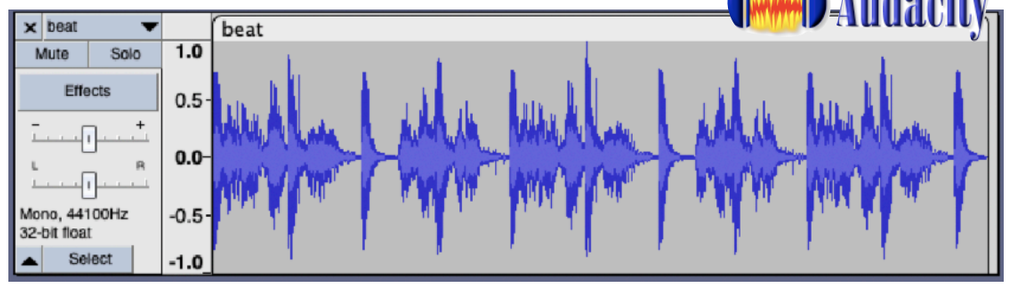


```
void draw() {
  background(255);
  // update (shift array to the left)
  for (int i=0; i<width-1; i++)
    history[i] = history[i+1];
  // save current amplitude at the last position
  history[width-1] = analyzer.analyze();
  // draw all recorded amplitude values
  for (int i=0; i<width; i++)
    line(i, height/2-history[i]*height*scale, i, height/2+history[i]*height*scale);
}
```

A

B

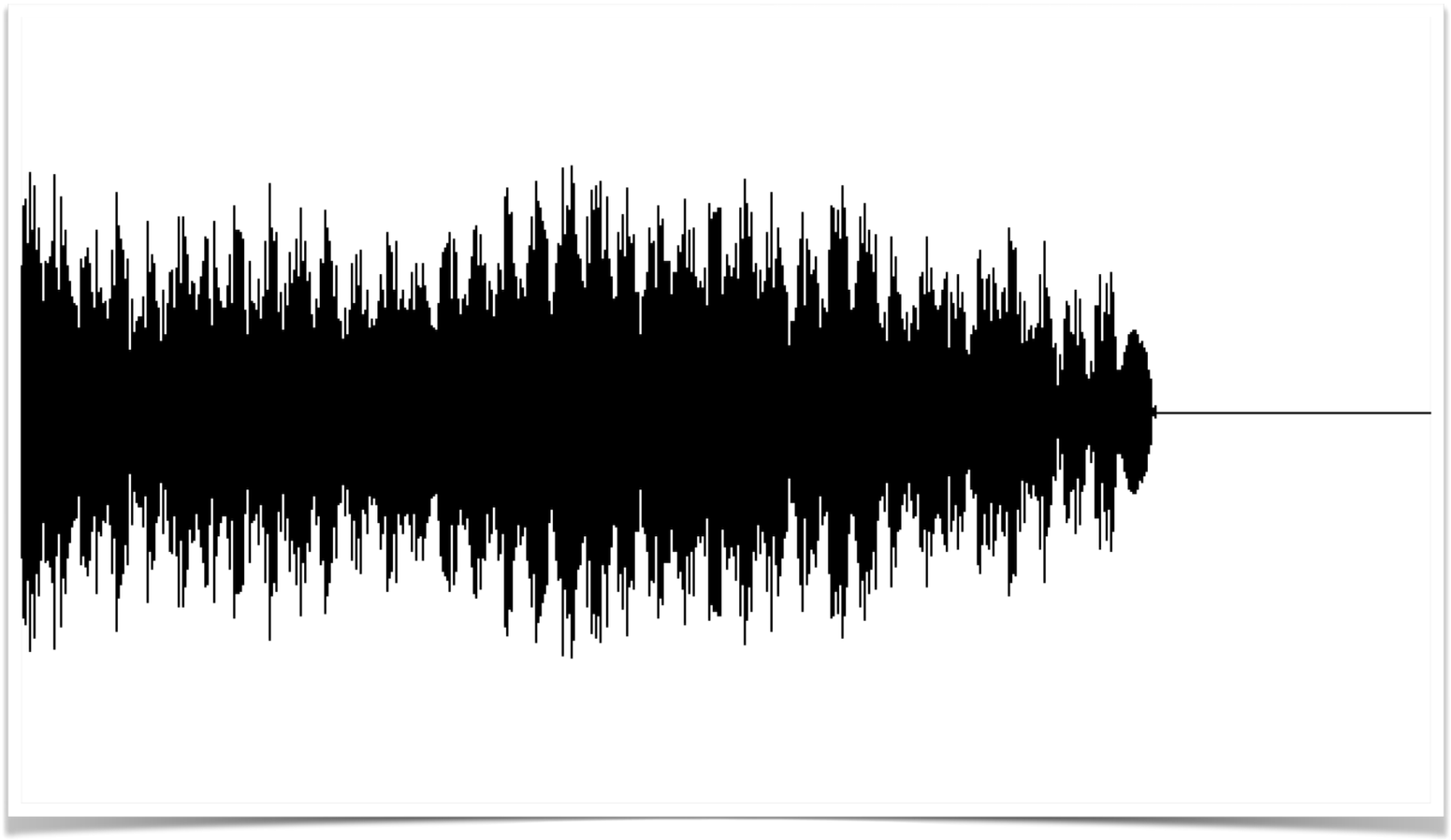
C



vertically symmetric

Exercise 1

- Based on Examples 3a-c, plot waveform of the sound file and move the wave rightwards



Waveform

- Waveform analyzer
- Returns the waveform of an audio stream the moment it is queried with the analyze() method
- Declare and create a Waveform analyzer

```
Waveform waveform;
waveform = new Waveform(this, samples);
```

constructor

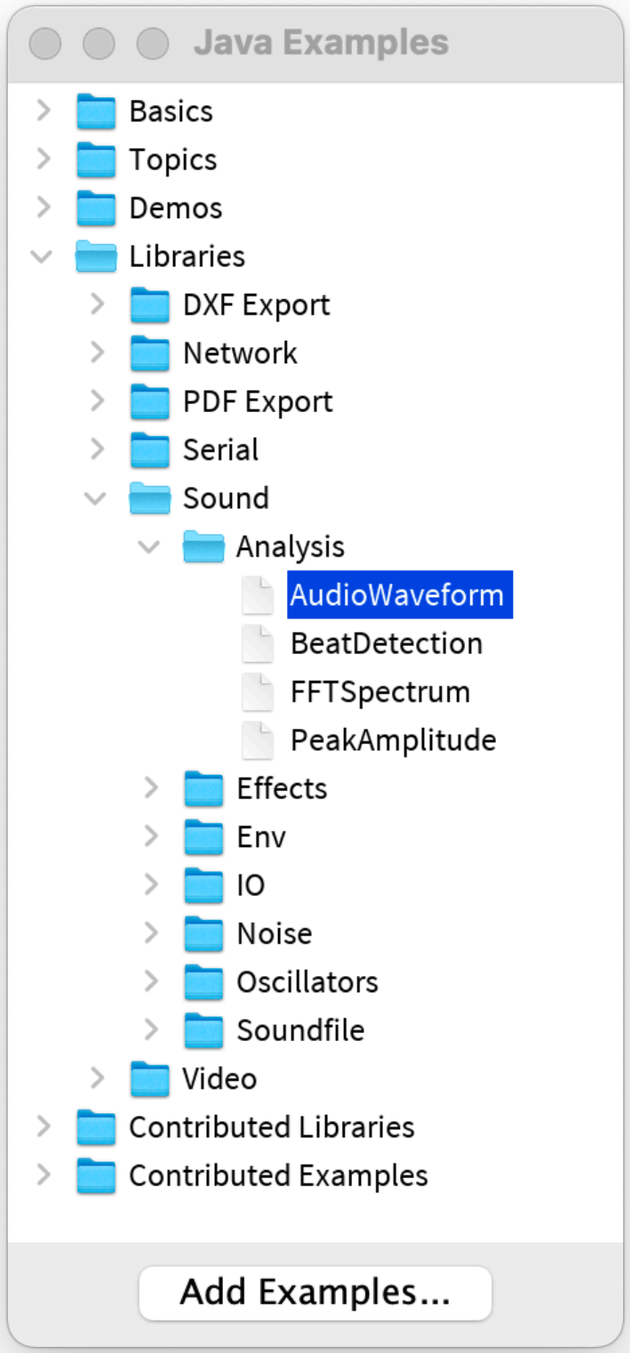
- Defines the audio input of the Waveform analyzer

```
waveform.input(sample);
```

- Gets the content of the current audiobuffer from the input source, writes it into the Waveform's data array (float[], each sample is between -1 and 1), and returns it

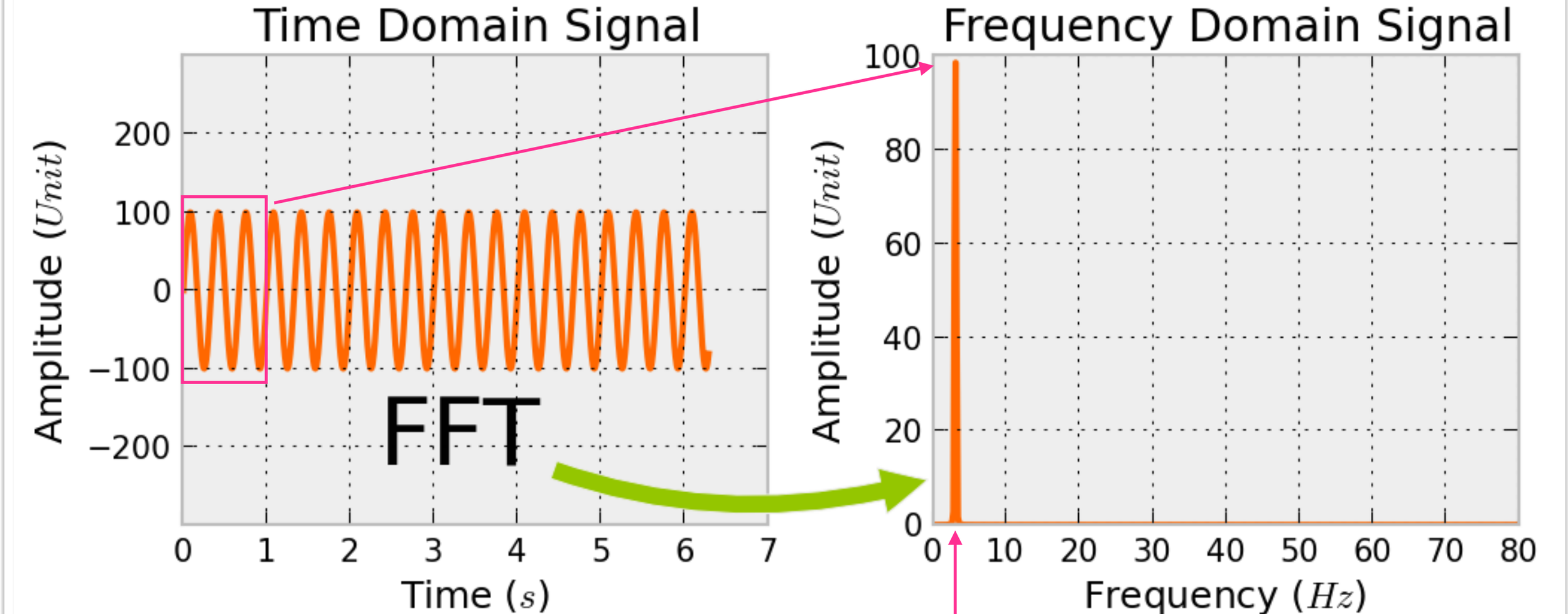
```
waveform.analyze();
waveform.data[i]
```

Example/libraries/Sound/Analysis/AudioWaveform




```
background(0);
stroke(255);
strokeWeight(2);
noFill();

// Perform the analysis
waveform.analyze();

beginShape();
for(int i = 0; i < samples; i++){
  // Draw current data of the waveform
  // Each sample in the data array is between -1 and +1
  vertex(
    map(i, 0, samples, 0, width),
    map(waveform.data[i], -1, 1, 0, height)
  );
}
endShape();
```

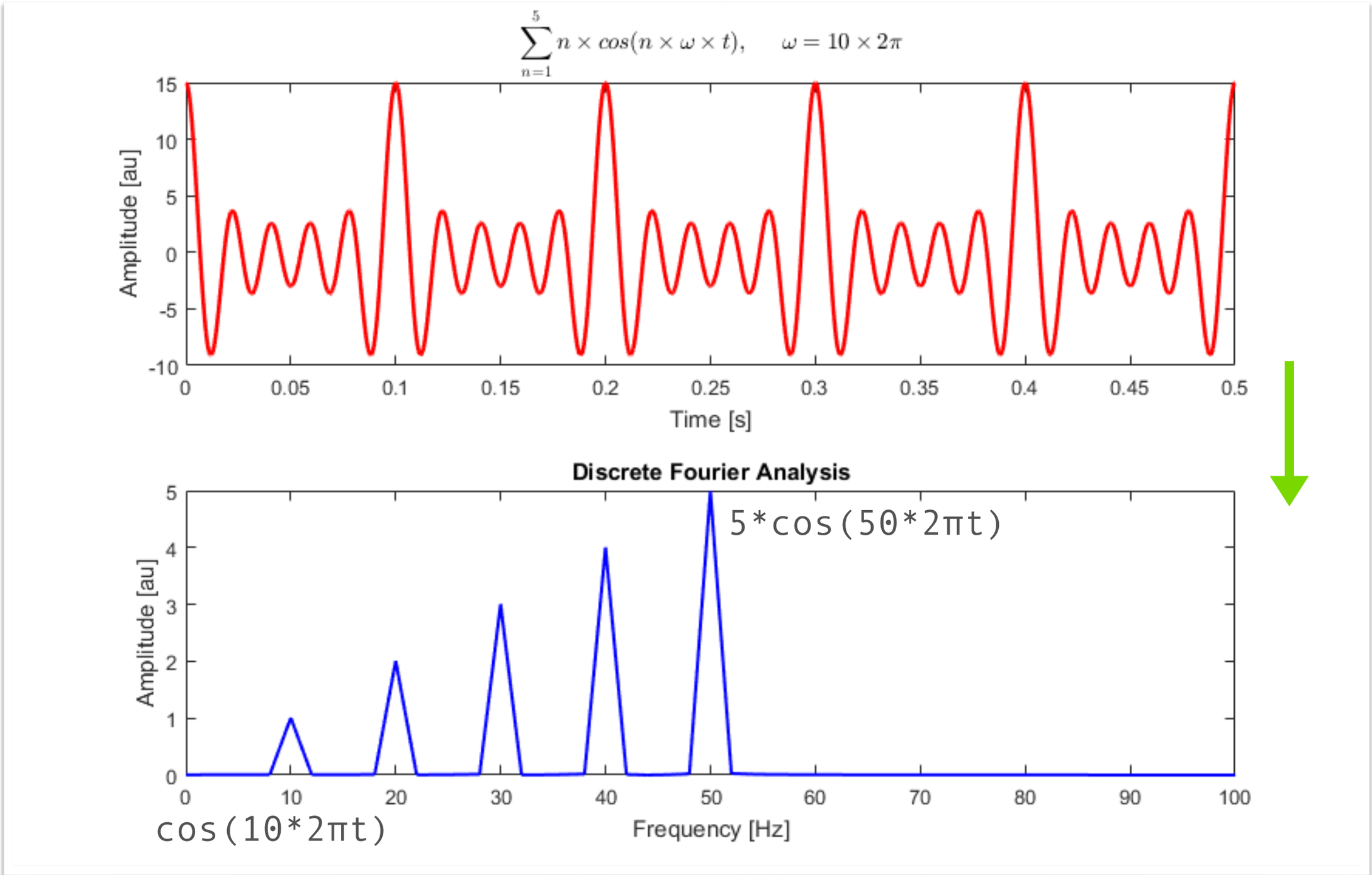
Fast Fourier Transform (FFT)



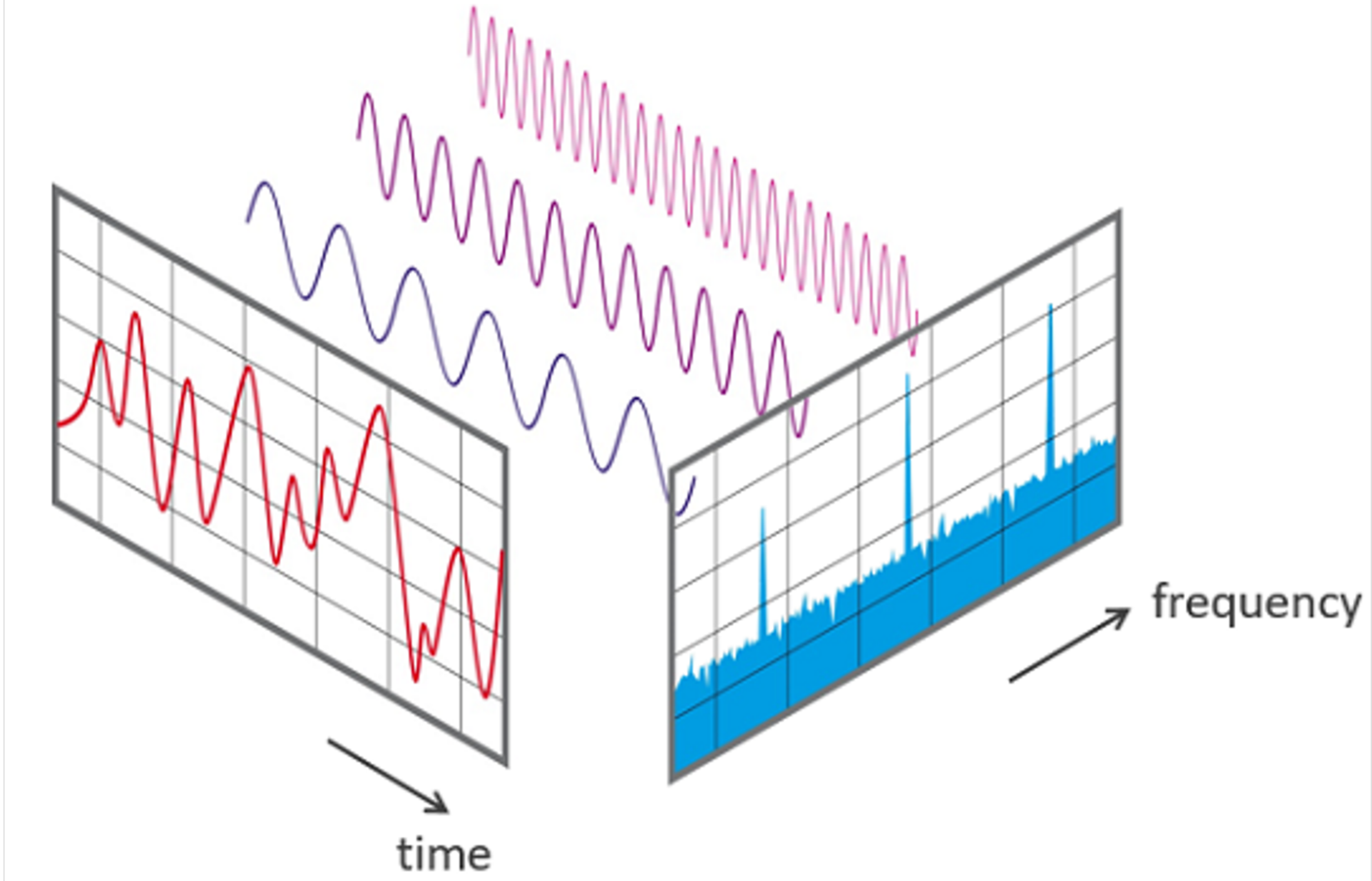
Frequency = 3Hz

Fast Fourier Transform (FFT)

- FFT of a sum of cosine waves at 10, 20, 30, 40 and 50Hz



Fast Fourier Transform (FFT)



Fast Fourier Transform (FFT)
Analyzer (Frequency)

- Calculate the normalized power spectrum of an audio stream the moment it is queried with the analyze() method
- Declare and create an FFT analyzer

```
FFT fft;
fft = new FFT(this, no_of_bands);
```

default 512, needs to be a power of 2 (e.g. 16, 32, 64, 128, ...)

- Define the audio input of the FFT analyzer

```
fft.input(song); //song is an instant of SoundFile;
```

or

```
fft.input(in); //in is an instant of AudioIn;
```

- Query a value from the analyzer and return a vector the size of the pre-defined number of bands

```
float[] spectrum = new float[no_of_bands];
fft.analyze(spectrum);
```

Example 4

```
import processing.sound.*;
SoundFile song;
FFT fft;
int bands = 512;
float[] spectrum = new float[bands];
int scale = 20;
void setup() {
  size(512, 360);
  background(255);
  song = new SoundFile(this, "vibraphon.aiff");
  song.loop();
  fft = new FFT(this, bands);
  fft.input(song);
}
void draw() {
  background(255);
  fft.analyze(spectrum);
  for(int i = 0; i < bands; i++){
  // The result of the FFT is normalized
  // draw the line for frequency band i scaling it up to get more amplitude
    line( i, height, i, height - spectrum[i]*height*scale );
  }
}
```

Music Makes You Travel by Makio135 @ OpenProcessing
https://openprocessing.org/sketch/138877
https://openprocessing.org/sketch/1141799 (replica)

AudioIn

- Streams data from audio input (microphone)

- Declare and create an input stream

```
AudioIn in;
in = new AudioIn(this, 0);
```

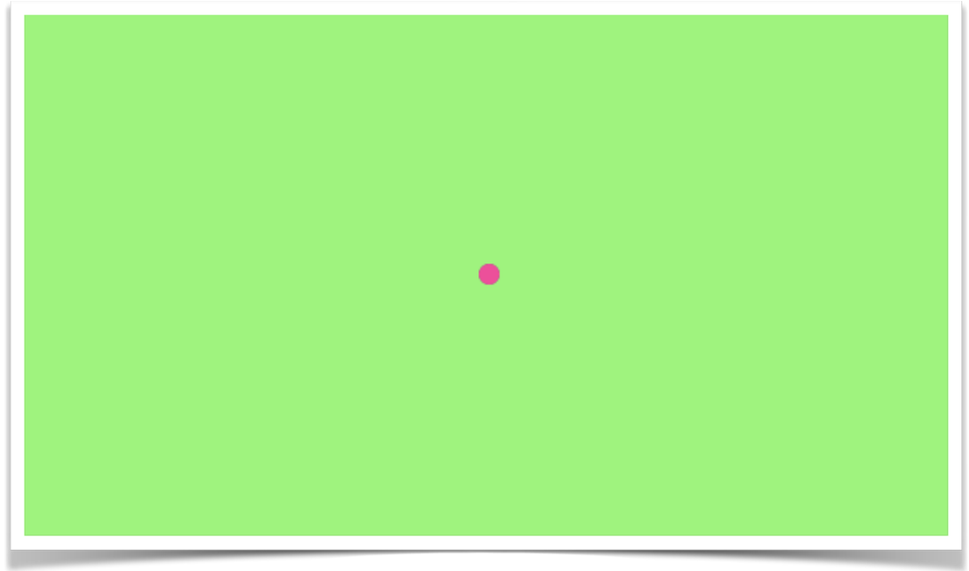
- Starts the input stream

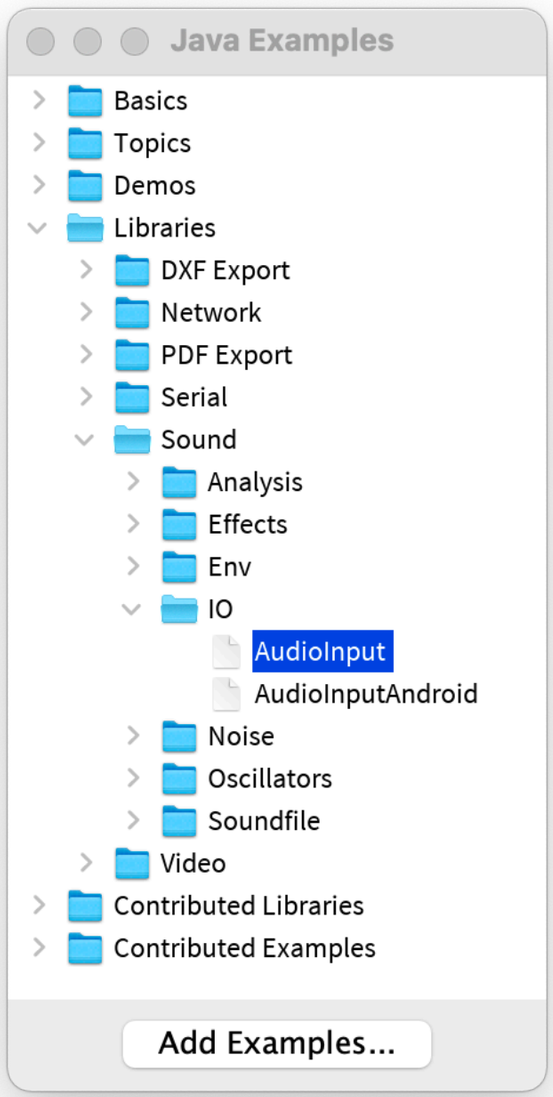
```
in.start(); // stream only
```

or

```
in.play(); // stream and route it to audio output
```

Example/libraries/ Sound/IO/AudioInput






```
void draw() {
  // Adjust the volume of the audio input based on mouse position
  float inputLevel = map(mouseY, 0, height, 1.0, 0.0);
  input.amp(inputLevel);

  // loudness.analyze() return a value between 0 and 1. To adjust
  // the scaling and mapping of an ellipse we scale from 0 to 0.5
  float volume = loudness.analyze();
  int size = int(map(volume, 0, 0.5, 1, 350));

  background(125, 255, 125);
  noStroke();
  fill(255, 0, 150);
  // We draw a circle whose size is coupled to the audio analysis
  ellipse(width/2, height/2, size, size);
}
```

Example 5A

```
void draw() {
    background(255);
    scale = int(map(amp.analyze(), 0, 0.2, 0, threshold));
    fill(c0);
    ellipse(width/2, height/2, scale, scale);
    if (scale>threshold)
      addCircle(scale);
    drawCircles();
}
void addCircle(int scale) {
  x = append(x, width/2);
  y = append(y, height/2);
  d = append(d, scale);
  sx = append(sx, (int)random(-10, 10));
  while (sx[sx.length-1]==0)
    sx[sx.length-1] = (int)random(-10, 10);
  sy = append(sy, (int)random(-10, 10));
  while (sy[sy.length-1]==0)
    sy[sy.length-1] = (int)random(-10, 10);
  c = append(c, c0);
  c0 = color(random(255),random(255),random(255), random(20,100));
  println("Num of circles: "+x.length);
}
```

to ensure non-zero speed

Example 5B (OOP Version of 5A)

```
class Circle {
  int x;    // x coordinates
  int y;    // y coordinates
  float sx; // speed in x direction
  float sy; // speed in y direction
  int d;    // diameters
  color c;  // colors
  Circle(int scale, color c) {
    x = width/2;
    y = height/2;
    d = scale;
    while (sx==0)
      sx = random(-10, 10);
    while (sy==0)
      sy = random(-10, 10);
    this.c = c;
  }
  void move() {
      // Update circle's position
      x += sx;
      y += sy;
      // Bounce back when touch the wall
      if ((sx<0 && x<d/2) || (sx>0 && x>width-d/2))
        sx *= -1;
      if ((sy<0 && y<d/2) || (sy>0 && y>height-d/2))
        sy *= -1;
    }
    void display() {
      fill(c);
      ellipse(x, y, d, d);
    }
}
```

Sound synthesis w/ Oscillators

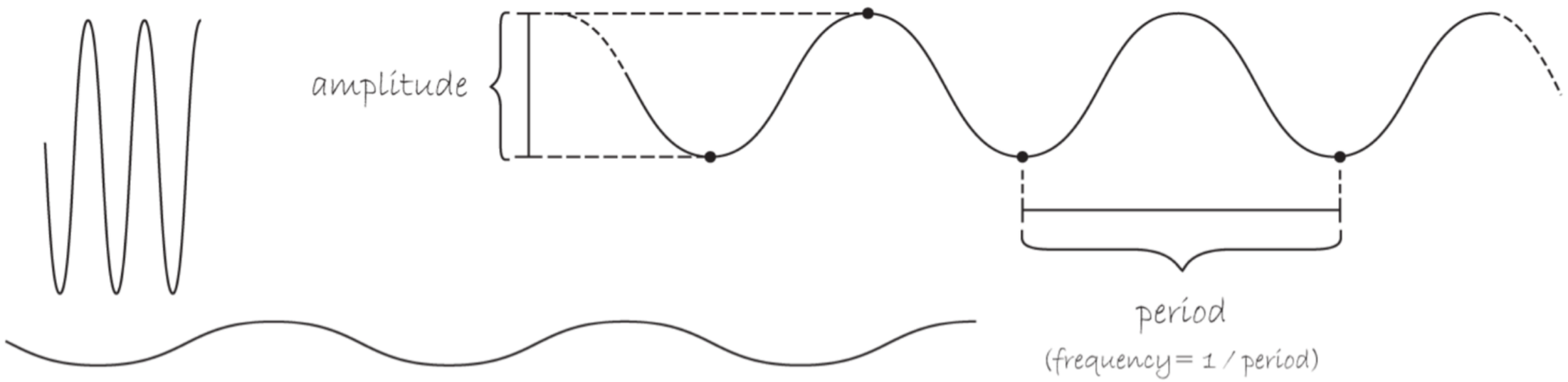


Figure 20-3

Oscillators

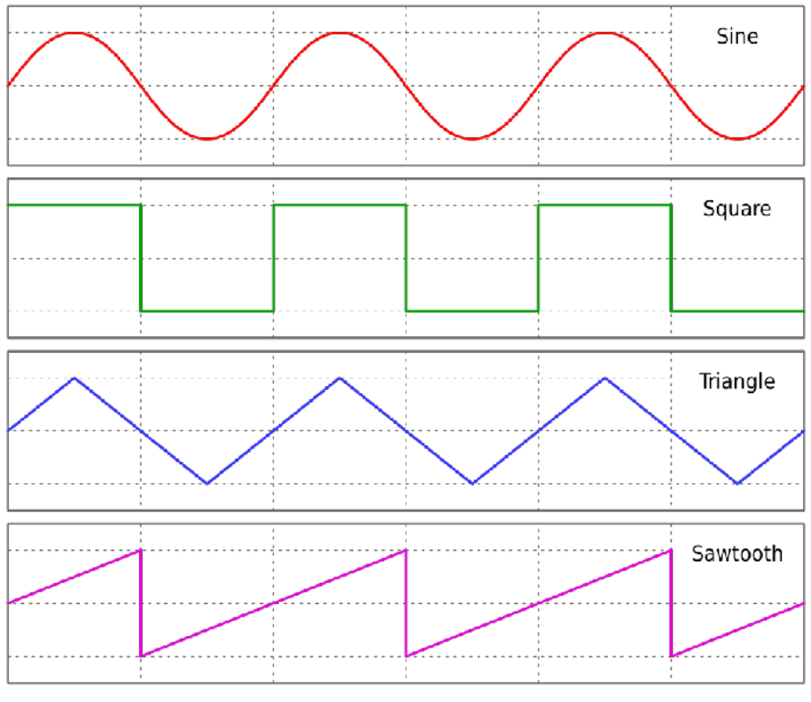
SinOsc

SawOsc

SqrOsc

TriOsc

Pulse

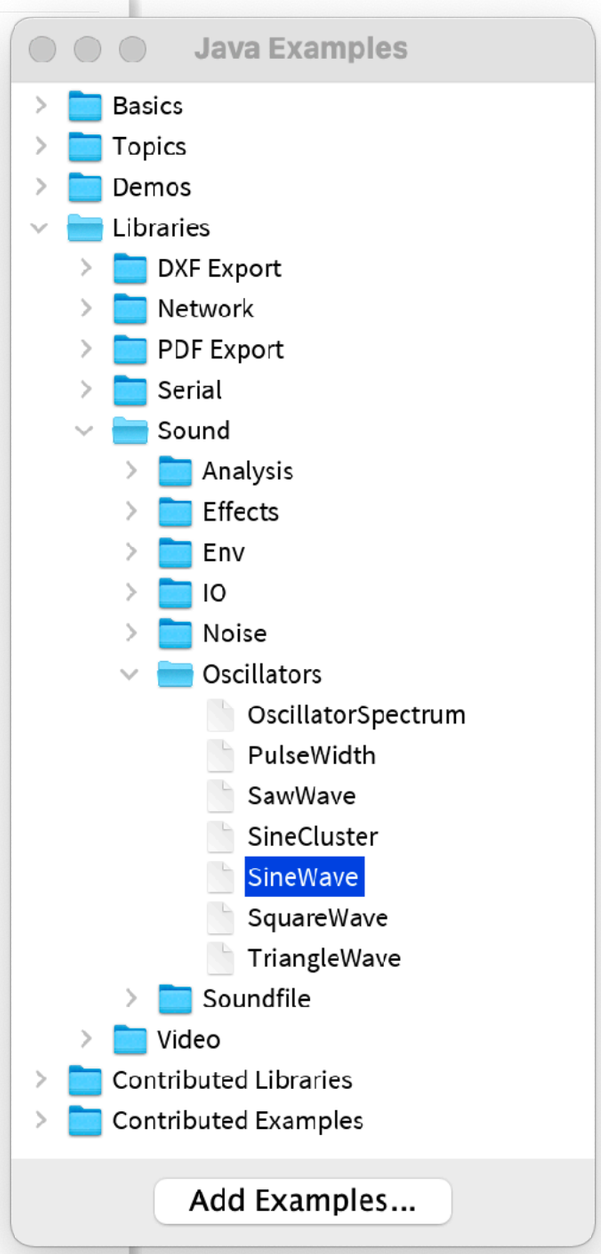


An oscillator is a repeating waveform with a fundamental frequency and peak amplitude and it forms the basis of most popular synthesis techniques.

https://en.wikibooks.org/wiki/Sound_Synthesis_Theory/Oscillators_and_Wavetables

SINE WAVE OSCILLATOR

```
import processing.sound.*;

SinOsc sine;

void setup() {
  size(640, 360);
  background(255);

  // create and start the sine oscillator.
  sine = new SinOsc(this);
  sine.play();
}

void draw() {

  // Map mouseY from 0.0 to 1.0 for amplitude
  float amplitude = map(mouseY, 0, height, 1.0, 0.0);
  sine.amp(amplitude);

  // Map mouseX from 20Hz to 1000Hz for frequency
  float frequency = map(mouseX, 0, width, 80.0, 1000.0);
  sine.freq(frequency);

  // Map mouseX from -1.0 to 1.0 for panning the audio to the left or right
  float panning = map(mouseX, 0, width, -1.0, 1.0);
  sine.pan(panning);
}
```



EXERCISE 2

- Create a Processing sketch that uses sine wave oscillator to play a melody
- The notes of a major scale correspond to the following frequencies:

Note	Frequency
C	261.63
D	293.66
E	329.63
F	349.23
G	392.00
A	440.00
B	493.88
C	523.25

- Map each of the notes to a key or mouse region

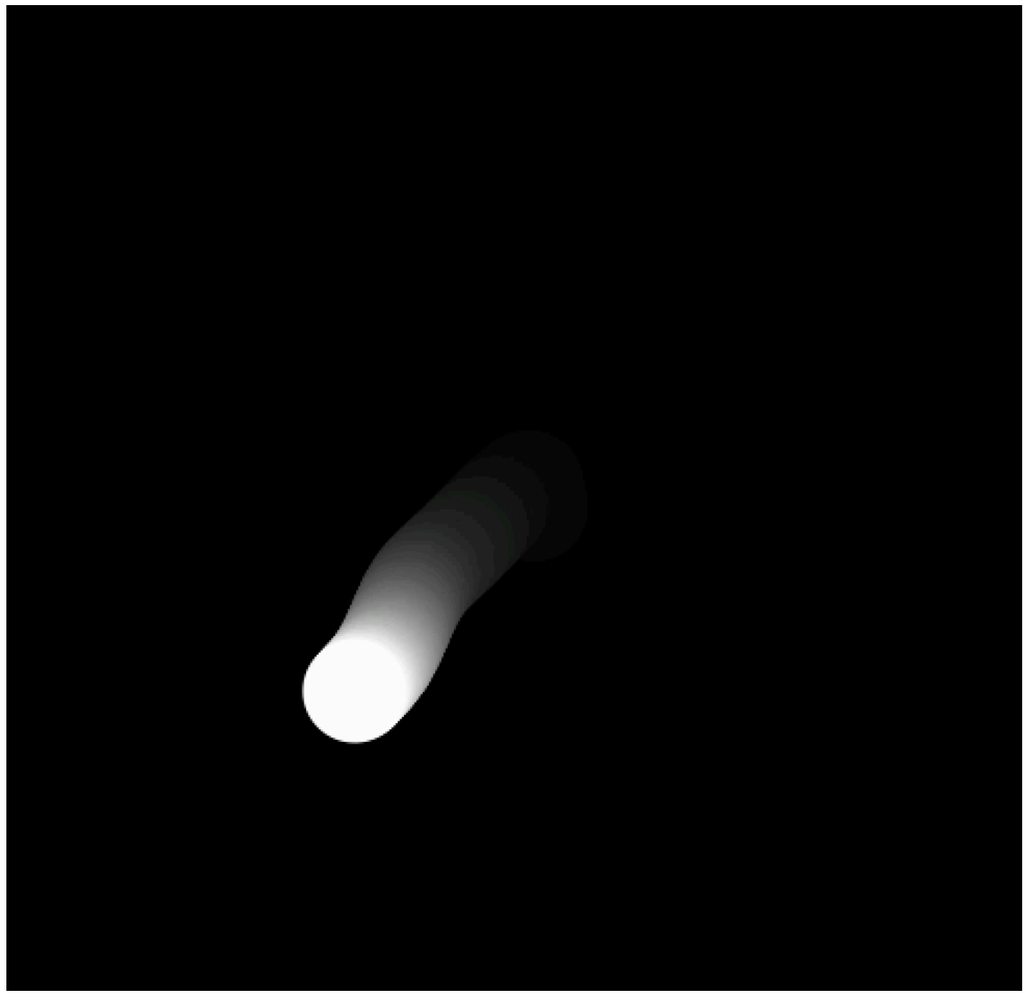
Example 6

```
import processing.sound.*;

SinOsc sine;
int index = 0;
float t = 0;

void setup() {
  size(640, 360);
  background(255);
  sine = new SinOsc(this);
  sine.play();
}

void draw() {
  float n = noise(t);
  sine.freq(map(n, 0, 1, 100, 1000));
  t += 0.005;
}
```

Update the frequency based on noise value

Example 7

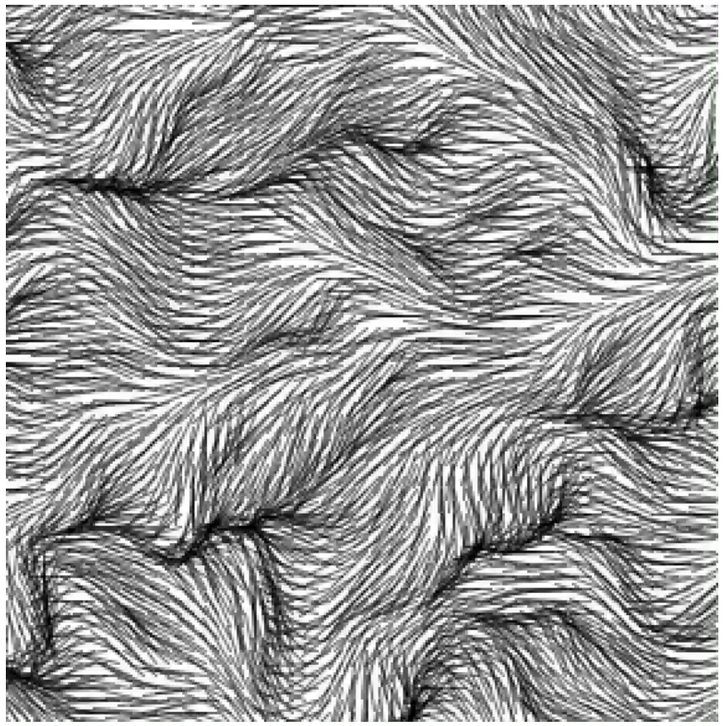
```
// Generate noise values
float nx = noise(tx);
float ny = noise(ty);

// Map noise values to screen coordinates
float x = nx * width;
float y = ny * height;


// Draw a circle at the mapped coordinates
fill(255);
ellipse(x, y, 50, 50);

// Map noise values to sound parameters
float rate = map(nx, 0, 1, 0.1, 1);
float amp = map(ny, 0, 1, 0.5, 1);

// Set sound parameters based on noise values
soundFile.rate(rate);
soundFile.amp(amp);

// Increment noise offsets
tx += 0.003;
ty += 0.003;
```



Update soundFile
playback rate and volume
based on noise values

Example 8 - control the step size/ increment of noise() function based on the amplitude of AudioIn (microphone)

```
// Detect the volume level of the microphone
float volume = loudness.analyze();

ynoise = ystart;

// Map the microphone volume level to the
// time-related noise parameter
tnoise += map(volume, 0, 1, 0.001, 0.1);
.
.
.
drawLine(x, y, noise(xnoise, ynoise,
tnoise));
```



Extended from Week 8 - Exercise 2

Noise generators

Noise

BrownNoise

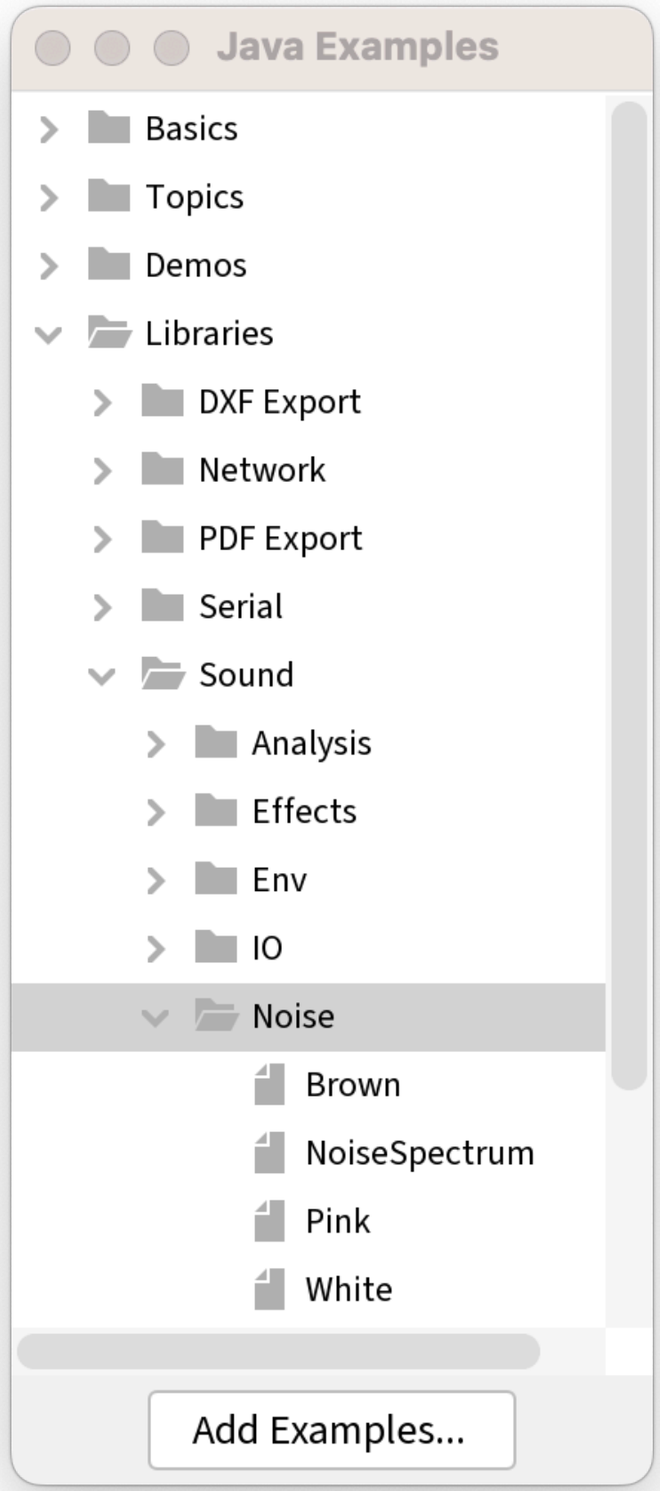
`BrownNoise`	This is a brown noise generator.
`add()`	Offset the output of this generator by a fixed value.
`amp()`	Changes the amplitude/volume of the noise generator.
`pan()`	Pan the generator in a stereo panorama.
`play()`	Start the generator.
`set()`	Sets amplitude, add and pan position with one method.
`stop()`	Stops the Brown Noise generator.

PinkNoise

`PinkNoise`	This is a pink noise generator.
`add()`	Offset the output of this generator by a fixed value.
`amp()`	Change the amplitude/volume of this sound.
`pan()`	Pan the generator in a stereo panorama.
`play()`	Start the generator.
`set()`	Sets amplitude, add and pan position with one method.
`stop()`	Stops the Pink Noise generator.

WhiteNoise

`add()`	Offset the output of this generator by a fixed value.
`amp()`	Change the amplitude/volume of this sound.
`WhiteNoise`	This is a White Noise Generator.
`pan()`	Move the sound in a stereo panorama.
`play()`	Start the generator.
`set()`	Set multiple parameters at once.
`stop()`	Stop the generator.



For Assignment 2, you're required to use the PERLIN noise() function but not white/pink/brown noise!

https://yogasleep.com/blogs/give-sleep-a-chance-blog/pink-noise-the-full-sound-spectrum

READINGS

- Sound Tutorial at Processing.org
- Learning Processing Chapter 20 Sound