COMP 1600 Fall 2025

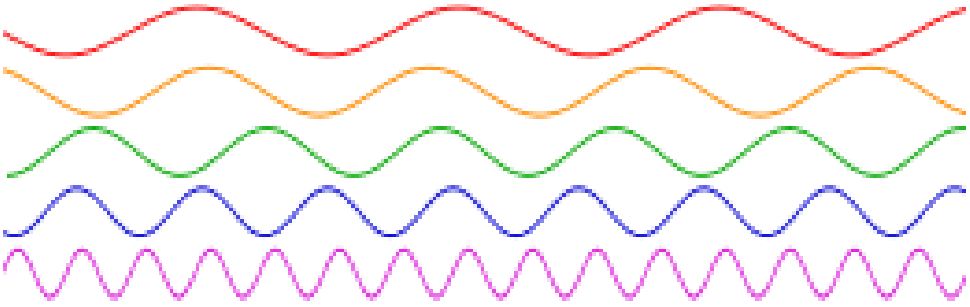
Project 3: Ride the Wave

Due by: Friday, October 24, 2025 at 11:59 p.m.

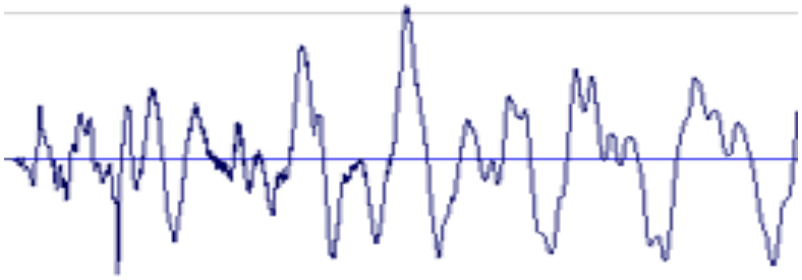
For this project you will write a program that processes audio files based on user input. You might be familiar with audio editing tools such as SoundForge or Cool Edit Pro. The program you will write will have some of the same functionality, only using text input commands instead of a GUI.

Background

This project is based around the manipulation of audio data inside a program. As you probably known from physics classes, sound travels as a wave. We can visualize this wave as something like a sine function.



Here is an image of several waves with different frequencies. Each is what is called a pure tone. At the top, the frequency is lower, making the equivalent note lower in pitch. The pitch of the notes increases as the waves go down the page. Of course, the representation of real sounds are more complex. Real sounds have chaotic looking wave forms because much more is going on than just a single pure tone, as shown below.



Of course, computers can't deal with nice, smooth waves. To record or manipulate a sound in a computer, you need to turn the wave into a series of values. This is done with a process called sampling. Sampling means that the wave is chopped up into lots of equal sized time divisions. For each division, we pick one value to represent the wave at that point. In this way, we can approximate the wave with a series of numbers. Below is a picture showing a tiny section of a wave form. Each small blue box represents the sample value of that wave.

With CD quality sound, there are 44,100 samples per second. That's why CD audio takes up so much space. Every single second has 44,100 numbers giving the height of various parts of a wave.

The `StdAudio` library developed by two Princeton professors lets us easily manipulate CD quality audio, but not MP3's. MP3's use a number of mathematical and psycho-acoustic techniques to use a lot fewer numbers to represent the waves. So, we're going to stick with CD quality audio, stored in the form of `.wav` files on your computer.

The `StdAudio` library provides a very easy way to load and save `.wav` files. When they are loaded, they will be represented as an array of `double` values. Each value represents one sample. So, for a song that is 3:56 long, the array will contain 10,407,600 `double` values. For this reason, we are going to stick to small `.wav` files. In the format used by `StdAudio`, each value in the array is between -1.0 and 1.0. Thus, a maximum volume sound has peaks at 1.0 and -1.0. A completely silent sound represented by samples at 0.

Overview

In this project, you will use your knowledge of loops, arrays, and the `StdAudio` library to make a program that takes an audio file as input and then applies a number of audio processing tasks and effects to the file. Each task is indicated by a command such as `p` or `n`. Some commands require

additional information to be read. All of these commands are given as input typed by the user. There is no limit to the number of commands that can be provided, and they should be processed **in order**.

There are 7 tasks in total, but each one is relatively simple. These tasks are:

Task	Command	Description
Play	`p`	Play the audio
Reverse	`r`	Reverse the audio
Increase speed	`s`	Speed up the audio by a specific factor
Add noise	`n`	Add a specified amount of white noise to the audio
Change volume	`v`	Scale the volume of the audio up or down
Output	`o`	Output the current version of the audio to a specified file
Quit	`q`	Quit the program

You must download `StdAudio.java` from **here**. You can either save it in your project `src` folder once you have created your project or you can add a `StdAudio` class to your project and paste the text in.

Sound Processing

Reversing Sounds

To reverse a sound, take the array of `double` values representing samples and reverse the order of its elements. The new array will effectively contain the audio backwards.

Speeding Up Sounds

First, create a new `double` array whose length is `length/factor`, where `length` is the original length and `factor` is the amount by which you are speeding up the audio. Then, loop through the new array filling it with values from the original array. Element `i` in the new array will correspond to element `i*factor` from the old array. That is, if `factor` is 2, you will use every other value from the original. Obviously, the index in the old array that you calculate will be a `double`. You will need to cast it to an `int` to use it. Anything other than very small values of `factor` will produce audio that is so fast as to be unlistenable. If you write your code correctly, it should be able to speed

up **and** slow down sounds. A sound will be sped up with a factor greater than 1 and slowed down with a factor less than 1.

Adding Noise

Noise is essentially just random values added to the audio. For each value in the array of audio samples, add a random number between `-amount` and `amount`, where `amount` is the value specified by the user. Make sure that no value is greater than 1.0 or less than -1.0 after noise is added.

Changing the Volume of Sounds

Multiply each value in the array of audio samples by the scale supplied by the user. If the scale is larger than 1, the noise will get louder. If the scale is smaller than 1, the noise will get quieter. A value of 0 would make the sound completely inaudible. Make sure that no sample in the array is greater than 1.0 or less than -1.0 after scaling.

Samples

Here is a list of 24 sample sounds we have included for your use.

- **amazing.wav**
- **breakbeato.wav**
- **cuckoo.wav**
- **coffee.wav**
- **compute.wav**
- **crazy.wav**
- **defeat.wav**
- **doomed.wav**
- **doorbell.wav**
- **eol.wav**
- **eva.wav**
- **hal.wav**
- **insane.wav**
- **milkshake.wav**
- **nature.wav**
- **obtuse.wav**
- **pacman.wav**
- **pain.wav**
- **prunes.wav**
- **saber.wav**
- **secret.wav**
- **shiny.wav**
- **whoa.wav**
- **world.wav**

Note: You will need to put these sounds in your project directory of your project for your program to access them. They should be in the top level of the project directory, **not** in the `out` or `src` directories.

Your program should prompt the user to for the name of a wav file. After that, you should repeatedly prompt the user for commands until he or she chooses the quit (`q`) option.

For example, the following sample program execution loads `whoa.wav`, plays the file without changes, reverses the sound, speeds up the sound by a factor of 1.7, adds a 0.1 amount of noise, scale the volume by a factor of 1.4, play the song another time to hear how all the changes have gone, saves the current audio to the file name `whoaChanged.wav`, and then quits.

```
Enter wav file name: whoa.wav
Select command (p, r, s, n, v, o, q): p
       Playing sound
Select command (p, r, s, n, v, o, q): r
       Reversing sound
Select command (p, r, s, n, v, o, q): s
       Speed up by how much? 1.7
       Speeding up sound
Select command (p, r, s, n, v, o, q): n
       Add how much noise? 0.1
       Adding noise
Select command (p, r, s, n, v, o, q): v
       Scale volume by how much? 1.4
       Scaling volume
Select command (p, r, s, n, v, o, q): p
       Playing sound
Select command (p, r, s, n, v, o, q): o
       Save to what file name? whoaChanged.wav
       Saving file
Select command (p, r, s, n, v, o, q): q
```

Here you can download the final version of **whoaChanged.wav** to give an example of what the sample from above would sound like.

Here is an example of each command. We have included an audio example using only a single effect flag to make a helpful example, but your program should handle any number of commands.

Task	Choice	Description	Example Usage	Sample Audio

Play	`p`	Play the current audio as it is, with the accumulated chosen effects. This option uses a `play()` method from `StdAudio`.	`Enter wav file name: coffee.wav`
			`Select command (p, r, s, n, v, o, q): p`
			`Playing sound`
				No change in audio

Reverse	`r`	Reverse the audio so that it plays backwards.	`Enter wav file name: shiny.wav`
			`Select command (p, r, s, n, v, o, q): r`
			`Reversing sound`
			`Select command (p, r, s, n, v, o, q): o`
			`Save to what file name?`
			`shinyReverse.wav`
				shinyReverse.wav

Task	Choice	Description	Example Usage	Sample Audio

Increase speed	`s`	Speed up the audio by a specific factor. Note that you will have to prompt the user for a `double` as the speed up factor.	`Enter wav file name: pain.wav`
			`Select command (p, r, s, n, v, o, q): s`
			`Speed up by how much? 2.5`
			`Speeding up sound`
			`Select command (p, r, s, n, v, o, q): o`
			`Save to what file name?`
			`painSpeed.wav`
				painSpeed.wav

Add noise	`n`	Add a specified amount of white noise to the audio. Note that you will have to prompt the user for a `double` as the amount of noise.	`Enter wav file name: pacman.wav`
			`Select command (p, r, s, n, v, o, q): n`
			`Add how much noise? 0.5`
			`Adding noise`
			`Select command (p, r, s, n, v, o, q): o`
			`Save to what file name?`
			`pacmanNoise.wav`
				pacmanNoise.wav

Change volume	`v`	Scale the volume of the audio up or down. Note that you will have to prompt the user for a `double` to know how to scale the volume.	`Enter wav file name: breakbeato.wav`
			`Select command (p, r, s, n, v, o, q): v`
			`Scale volume by how much? 0.5`
			`Scaling volume`
			`Select command (p, r, s, n, v, o, q): o`
			`Save to what file name?`
			`breakbeatoVolume.wav`
				breakbeatoVolume.wav

Output	`o`	Output the current version of the audio to a specified file. Note that you will have to prompt the user for a `String` to know the new file name. This option uses the `save()` method from `StdAudio`.	`Enter wav file name: milkshake.wav`
			`Select command (p, r, s, n, v, o, q): o`
			`Save to what file name?`
			`rename.wav`
				No change in audio

Quit	`q`	Quit the program.		

Turn In

Your IntelliJ project should be called `Project3`, but the class you create inside should be called `AudioProcessor`. Upload the `AudioProcessor.java` file from the `Project3\src` folder wherever you created your project to **Brightspace**.

All work must be submitted before Friday, October 24, 2025 at 11:59 p.m. unless you are going to use a grace day.

All work must be done individually. You may discuss general concepts with your classmates, but it is never acceptable for you to look at someone else's code. Please refer to the course policies if you have any questions about academic integrity. If you have trouble with the assignment, I am always available for assistance.

Grading

Your grade will be determined by the following categories:

Category	Weight
Play	10%
Reverse	20%
Increase speed	20%
Add noise	15%
Change volume	15%
File output	10%
Style and comments	10%

Under no circumstances should any student look at the code written by another student. Tools will be used to detect code similarity automatically.

Code that does not compile will automatically score zero points.