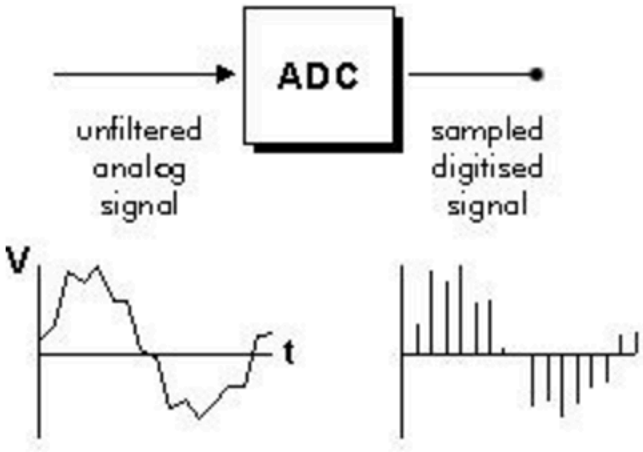
Project 1: Sound Blaster!

& Not a group project.....

This is a cool first project I did in my data structures class.

We're going to use the DoubleStack data structure to manipulate sound. We are going to use a stack to reverse a sound wave.

For this project, view sound as a continuous function of time from the positive real numbers (time) to the interval [-1.0, 1.0] (amplitude). Since a computer can't "hold" a function defined on the reals, we have to approximate the function. We do this by measuring (or "sampling") the sound several thousand times per second.



We're going to use stacks to manipulate sound now. You'll also gain a little practice with command-line – noteably, a utility called sox.

It will let you convert between different sound formats. We'll use them for .dat files.

The .dat File format

The .dat file format starts with one line describing the sample rate of the sound file. This line is required. The rest of the file is composed of two columns of numbers. The first column consists of the time (measured in seconds) when the sample was recorded, and the second column contains the value of the

sample, between -1.0 and 1.0. This is the beginning of a sample .dat file. Notice that the numbers in the first column increase by 1/44100 each step. This is because the sample rate is 44.1kHz.

```
; Sample Rate 44100 0 0 2.2675737e-05 0 4.5351474e-05 0
6.8027211e-05 0
9.0702948e-05 0
0.00011337868 0
0.00013605442 0
0.00015873016 0
0.00018140590 0
0.00020408163 0
```

General Strategy

The general strategy for using sox is as follows.

1. Take a .wav sound file of your choosing (e.g. secret.wav). This sound shouldn't be longer than a few seconds, or your program will run out of memory.
2. Convert it to a .dat file: **sox secret.wav secret.dat**
3. Manipulate it with the program you will write: j**ava Reverse list double secret.dat secret-revealed.dat**
Convert it back to a .wav file: **sox secret-revealed.dat secret-revealed.wav**
4. Listen to it with your favorite sound player.

Reversing a Sound

You'll take a .dat sound file, and output another .dat sound file. The output will reverse the numbers in the second column. This will make the sound "play backwards." I'll provide you with the implementation of DoubleStack class, and reverse class (which reads in a .dat sound file, put the sounds values on a stack, pops them off in reverse order, and puts the reversed values in a new .dat sound file.)

Your Project

You need to provide two stack implementations, one using an array and one using a linked list. They should be called ArrayStack and ListStack, respectively. They should implement the DStack interface given to you. Once you provide these implementations, Reverse

should work and create backwards sound files. It should take no more than a page or two of code to provide the implementations.

Notes:

Your array implementation should start with an array of size 10 elements and resize to use an array twice as large whenever the array becomes full, copying over the elements in the smaller array. When growing your array, do your copying "by hand" with a loop, do not use Arrays.copyOf or other similar methods. It is good to know that these methods exist, but for now we want to focus on understanding everything that is going on "under the covers" as we talk about efficiency. If you write the copy method yourself - then you really can see this. (You can either write a separate private helper method, or just put the code in directly.) Using the length property of an array is perfectly fine to do.

For your linked list implementation, you should implement your own linked list nodes and build a stack out of those, similar to the slides from lecture 1 that do the same thing for a queue. You should NOT be using any classes from Java collections. You may include your node class as a separate file (Don't forget to submit this file!) or as a nested/inner class. Either will be fine.
Both ArrayStack and ListStack should throw an EmptyStackException if pop() or peek() is called when the stack is empty. To use EmptyStackException, add the following line to your file:

```
import java.util.EmptyStackException;
```

Note that your solution does not require making changes to Reverse.java.

Running Your Program

The Reverse program takes 4 arguments (a.k.a. "command-line arguments"). The first is the word array or list, and specifies which implementation to use. The second is the word double or generic; the latter is for Phase B. The next two are the input and output .dat file names (you need to include the .dat extension). From the command-line, you can run the program with something like:

```
java Reverse list double in.dat out.dat
```

Write-Up Questions

1.Who and what did you find helpful for this project?

2.How did you test that your stack implementations were correct?

3.Other than java.util.EmptyStackException, did you use any classes from the Java framework or other class library? (You will get a low score on this project if you use a library to implement your stacks.)

4.Your array stacks start with a small array and double in size if they become full. For a .dat file with 1 million lines, how many times would this resizing occur? What about with 1 billion lines or 1 trillion lines (assuming the computer had enough memory)? Explain your answer.

5.Instead of a DStack interface, pretend you were given a fully-functional FIFO Queue class. How might you implement this project (i.e., simulate a Stack) with one or more instances of a FIFO Queue?

6.Write pseudocode for your push and pop operations. Assume your Queue class provides the operations enqueue, dequeue, isEmpty, and size.

7.Why would a stack implementation using a queue, as you described in the previous problem, be worse than your array and linked-list stack implementations?

Turn in

Submit zip file on canvas.

- Each file you turn in should have your name in the file at the beginning of the file. All text files should have your name on the first line; for Java files: your name should appear in the comments at the beginning of the file.

- **You must implement the list and array stacks by hand. You may not use any classes from the Java libraries to do the work. You should not use any import statements, except for java.util.EmptyStackException.**

  •Turn in the following files, named EXACTLY as follows:
  ◦ArrayStack.java

  ◦ListStack.java

  While it is acceptable to submit a separate file for your node class, try using an inner class.

Acknowledgements

Thanks to my undergrad CS Data Structures class UW's CSE 326 for this project! Special thanks to Ashish Sabharwal, Adrien Treuille, and Adrien's Data Structures professor, Timothy Snyder.