CS 102: Data Structures

Project Three Point One - Sound Blaster

Student Name	Mohammad El-Abid
Student ID	8905652
Professor	Subrina Thompson
Date	Sunday, March 25, 2012

Table of Contents

Abstract

Digital sound creates an array of frequencies and volumes, these can be pushed to a last-in-first-out stack to reverse or normalize a reversed audio clip.

Introduction

I had already made my `Stack` class' `push` method allocate more space if needed, so for this application I just needed to google how to run a command line utility from Java. The application was straight forward, but was a more practical use of the `Stack` class then checking strings. It was also more fun the checking parenthesis.

Screenshots

Not applicable, application just outputs current position in program to `STDIN`.

Questions:

1. The scent of bitter almonds always reminded him of the fate of unrequited love.
2. It creates a new elements array and copies over the data, then proceeds as normal.
3. Practical use with more satisfaction then checking parenthesis, palindromes, or reversing a string.
4. I wasn't a fan of having to install an external dependency. Also, Gabriel García Márquez.

Code

File: Application.java

```
package edu.bridgeport.mohammad;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;


public class Application {
    // MacOS with homebrew:
    //   brew install sox
    // Uninstall sox with deps:
    //   brew uninstall sox pkg-config libogg libvorbis lame flac
libao mad

    /**
     * location of sox binary, can be found on NIX by `which sox`
     */
    static final String SOX_BINARY = "/usr/local/bin/sox";

    /**
     * @param args
     * @throws IOException
     */
    public static void main(String[] args) throws IOException {

          // Convert to .dat
          System.out.println("Calling sox to convert to dat");
          Runtime.getRuntime().exec(SOX_BINARY + " secret.wav
secret.dat");

          // Start pushing to stack
          System.out.println("Loading .dat");
          FileReader data = new FileReader("secret.dat");
          Scanner input = new Scanner(data);
          Stack<String> stack = new Stack<String>();
          // header should be lines that start with ;, though in this
case it's the first two
          StringBuilder header = new StringBuilder("");
          while(input.hasNext("/^;/")){
               header.append(input.nextLine() + "\n");
          }
```

```
            while(input.hasNextLine()) {
                  stack.push(input.nextLine());
            }

            input.close();
            data.close();

            // Create reversed .dat file
            System.out.println("Reversing .dat");
            FileWriter revealedData = new FileWriter("secret-
revealed.dat");
            BufferedWriter output = new BufferedWriter(revealedData);

            output.write(header.toString());
            while(!stack.isEmpty()) {
                  output.write(stack.pop() + "\n");
            }
            output.close();
            revealedData.close();
            stack = null;

            // Convert to .wav
            System.out.println("Calling sox to convert back to wav");
            Runtime.getRuntime().exec(SOX_BINARY + " secret-
revealed.dat secret-revealed.wav");

            System.out.println("Done!");
            System.out.println("It was inevitable: the scent of bitter
almonds always reminded him of the fate of unrequited love. - Gabriel
García Márquez");
      }

}
```

File: Stack.java

```
package edu.bridgeport.mohammad;
public class Stack <T> {
      private T[] elements;
      int insert_index = 0;

      public Stack(){
            constructElementArray(10);
      }

      public Stack(T[] objs) {
            this.elements = objs;
```

```
            // find first null index
            for(int i = 0; i < objs.length; i++) {
                  if(objs[i] == null) {
                        break;
                  } else {
                        insert_index++;
                  }
            }
      }

      public Stack(int size) {
            constructElementArray(size);
      }

      @SuppressWarnings("unchecked")
      private void constructElementArray(int size) {
            elements = (T[]) new Object[size];
            insert_index = 0;
      }

      @SuppressWarnings("unchecked")
      private void addElementLength(int addLength) {
            T[] new_elements = (T[]) new Object[elements.length +
addLength];
            System.arraycopy(elements, 0, new_elements, 0,
elements.length);
            elements = new_elements;
      }

      public boolean push(T obj) {
            if(insert_index >= elements.length){
                  addElementLength(15);
            }
            elements[insert_index++] = obj;
            return true;
      }

      public T pop() {
            if(insert_index > 0) {
                  insert_index--;
                  T obj = elements[insert_index];
                  elements[insert_index] = null;
                  return obj;
            } else {
                  return null;
            }
      }

      public T top() {
```

```
            if(insert_index > 0) {
                  return elements[insert_index - 1];
            } else {
                  return null;
            }
      }

      public void reset() {
            constructElementArray(elements.length);
      }

      public boolean isEmpty() {
            return insert_index == 0;
      }

      public boolean isFull() {
            return insert_index >= elements.length;
      }

      public int size() {
            return insert_index;
      }
}
```

Conclusion

It was nice to know how to work with external applications and have a more practical and fun usage for the stack class we've built.

Works Used

How to Run Command Line or Execute External Application From Java - http://www.linglom.com/2007/06/06/how-to-run-command-line-or-execute-external-application-from-java/