Assignment 1

Course CS 3345 – 503
Course Title Data Structures & Algorithm Analysis
Term Spring 2023

Due by **02/27/2023 at 11:59PM CST**

General Instructions

- Assignment is due on or before **Sunday 02/27/2023 at 11:59PM CST**
- This assignment is meant to be solved completely by individual students and not in groups.
- Assignments **will not** be accepted late, unless eLearning has an accessibility issue and I'm informed about it. The submission URL will be disabled after the due-date, and hence no late submissions will be possible.
- Any form of dishonesty will result in a score of **0** on the entire assignment. Any students involved will, according to Departmental Policy, be referred to the Director of Community Standards & Conduct for adjudication.
- Do not use anything that would trivialize the assignment (e.g. don't import/use java.util.ArrayList for an Array List solution, etc)

Assignment Details

The goal of this project is to write your own Stack ADT implementations using array and linked list. This project uses Java `double` data type. The idea is to write a program that can reverse a sound clip, which is a famous process known as backmasking. Backmasking was used by famous musicians and movie producers. You can reach more about it in details here (https://en.wikipedia.org/wiki/Backmasking)

You are required to write a program that reads a sound file in .dat format, and writes another .dat sound file that contains the reverse of the input file. The client main file `BackMasking.java` is provided to you. The program is simple: 1) reads line-by-line of the .dat file, 2) pushes all the read values into a stack, and 3) pops them off and writes them into an output .dat file.
The Stack interface `BKStack.java` is also provided, and you are requested to write two implementations to that Stack interface provided to you

You will implement:

- `ArrayStack:` Provides an array-based implementation of the BackMasking interface
- `ListStack:` Provides a Linked List implementations of the BackMasking interface, which also implements the `Iterable` interface. It uses the enhanced `for-loop` to count the elements in the stack.

The `ArrayStack` should define an `INITIAL_CAPACITY` for the array of type `int`. You should try different values and observe the performance impact. Also you should add a method which would resize the array to be double the current size when the array gets full.

The `Stack` 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;
```

The only Java class that you should use to complete the implementations of your stacks is `java.util.EmptyStackException`.

Provided Material

- You are provided with the following files:
  - `BackMasking.java` which is the main program used to generate a backmasking audio file,
  - `BKStack.java` which is the stack interface you are required to build two implementations for.
  - `clip.wav` and `ciphered.wav` audio files used for testing your program.

Converting Media File to .dat

To run you program, you will need to convert the media files (.wav) into .dat format. An analog "real" found is converted to digital, so it can be stored and processed by the computer. This happens in a process called Analog to Digital Conversion (ADC) using sampling rate measured in kHz.
The media files which are read and played by the different media players will need to be converted into another .dat format for your program to be able to parse and process it. Similarly, the output .dat format your program produces will need to be converted into some media format to be able to play it using any of the media programs.
SOund eXchange (SOX) is one simple to use utility that you can use to convert the provided .wav files to .dat for your processing, and you can use it once again to convert the output .dat file into .wav to listen to. You can also use any conversion of your choice.

Submission Details

- Your submission consists of the following files in the following specified names:
  - `ArrayStack.java`
  - `ListStack.java`
  - `ListStackNode.java`, the linked-list node for use with your `ListStack` class. If preferred, you can use an inner class inside `ListStack`; hence, you will need only the ListStack.java file.
  - Do not use any Java Collection Classes, except Strings and arrays.
  - `README.pdf` file, which is the submission report described below.
  - Do not modify or submit the BackMasking.java or BMStack.java. Your code should work with these given files without any modifications to them.

README Report Details

- Your report should include the following
  - Your name and NetID, section #, etc
  - Two sections describing your `ArrayStack` and `LinkedStack` programs. Highlight your solution approach and describe key parts in detail.
  - Explain the upper-bound O(N) complexity of each implementation.
  - (Optional) Highlight any specific algorithm you came up with to enhance the performance.

Grading Policy

- 100 points for the correct and complete implementation, code style, descriptive comments and performance
- 50 points for the README report

GOOD LUCK!