Overview:
This project entails two different implementations of array and list stack. The program is designed to reverse engineer and restore a reversed .dat sound clip using custom ArrayStack and ListStack data structures. I develop a program capable of reversing a sound file in .dat format, generating another .dat file containing the reversed sound. The main involves reading the .dat file line-by-line, pushing the read values into a stack, and subsequently popping them off to create an output .dat file.

ArrayStack Implementation:
Utilizes an array-based approach to implement the BackMasking interface. It incorporates an INITIAL_CAPACITY for the array and should include a resizing method to double the size of the array when it reaches full capacity.

ListStack Implentation:
Implements the BackMasking interface using a Linked List structure. This implementation also adheres to the Iterator interface, employing the enhanced for-loop to count elements in the stack. It is crucial that the Iterator method throws java.util.ConcurrentModificationException if modifications to the list structure occur outside the iterator after the iterator methods have been invoked.

Outputs:

- I have provide the the old .wav clip which I converted into .dat format using Sound Exchange(SOX) which converts sound from analog to digital which then it can be compiled through the array and list stack program.

Complexity for ArrayStack:
Moreover, with the complexities of the program. The first method included the double size which has O(n) due to having to go through each element in the old array and transfer them to the new size array. When it comes to pop it has a complexity of O(1). Push also has a big o of O(1), however, in the worst case of having to double its complexity is now O(n).
Also peek is O(1). Therefore, overall the arrayStack has a complexity of O(n) in worst cases however majority of is O(1).

Complexity for ListStack:

The complexity for three major parts of listStack include pop, peek, and push. Pop had a complexity of O(n). Similarly push and peek also had a complexity of O(1). Due to Linked List not having the resize issue compared to arraystack. The overall complexity of ListStack was O(1). The iterator had a complexity of O(n) due to iterating thought the list. However the methods next and has next are both O(1).Therefore, linked list is more of an efficient data structure for Stack implement on.