Homework 2

Problem 1 - MyStack - 15 points

Recall from class the underlying implementation of Java's `ArrayList`. It uses a regular Java array to store data and resizes to a larger array once it is full. Implement a stack in a class called `MyStack` that uses an array to store data and resizes the array when necessary. All methods should run in constant time, except when the array must be resized (this is still amortized constant time).
NOTE: Your class must implement `MyStackInterface` for full credit. You are not allowed to use any `List` object as your instance variable. You are only allowed to use arrays as your data retaining instance variable. Your `MyStack` must be generic as well.

Problem 2 - Symbol Balance - 30 points

Define a class called *SymbolBalance* in the provided empty SymbolBalance.java file.
Your `SymbolBalance` class will read through a Java file and check for simple syntatical errors. You should write two methods, as specified by the `SymbolBalanceInterface` which you must implement for full credit.
The first method, `setFile`, should take in a String representing the path to the file that should be checked.
The second method, `checkFile`, should read in the file character by character and check to make sure that all { }'s, ( )'s, [ ]'s, " "'s, and /* */'s are properly balanced. Make sure to ignore characters within literal strings (" ") and comment blocks (/* */). Process the file by iterating through it one character at a time. During iteration, the symbol currently pointed to in the loop will be referred to as `<Current Symbol>` henceforth.

You **do not need** to handle single line comments (those that start with //), literal characters (things in single quotes), or the diamond operator(<>).

There are three types of errors that can be encountered:

- The file ends with one or more opening symbols missing their corresponding closing symbols.
- There is a closing symbol without an opening symbol.
- There is a mismatch between closing and opening symbols (for example: { [}]).
  Once you encounter an error, return a `BalanceError` object containing error information. Each error type has its own class that descends from `BalanceError` and each has its own required parameters:

- Symbol mismatch after popping stack: return `MismatchError(int lineNumber, char currentSymbol, char symbolPopped)`
- Empty stack popped: `EmptyStackError(int lineNumber)`
- Non-empty stack after parsing entire file: `NonEmptyStackError(char topElement, int sizeOfStack)`
  If no error is found, return `null`.
  Only push and pop the `*` character to the stack when handling multi-line comments. Do **not** push the `/` character or the string `\*`.
  You must use your `MyStack` from Problem 1 in this problem.

We have provided you with a number of test inputs in the sub-folder TestFiles. We will use our own test files to grade your performance on all conditions - those files will be released **after** the assignment is due.

Problem 3 - Two Stack Queue - 15 points

Build a queue out of two completely separate stacks, `S1` and `S2`. Enqueue operations happen by pushing the data on to stack 1.
Dequeue operations are completed with a pop from stack 2.

Obviously you will have to find some way to get the input stack information over to the output stack. Your job is to figure out how and when to do that, using only push and pop operations.
Write a class TwoStackQueue that provides the Queue ADT (by implementing the `MyQueueInterface`) using two stacks. Your class should explicitly implement the interface provided above. Since the interface is generic, your class should be as well. Provide all methods specified in the interface. Your class should not use any additional memory to store the data in the queue except for the two stacks.

For your stacks, use the MyStack class that you implemented in the first problem.

All methods must run in constant time, except for one, which is allowed to run in linear time occasionally.