Assignment 3 Stacks and Queues

Due Date: Monday Feb 22, before midnight
Weight: 5%
This assignment should be completed individually.

Description

This assignment is a little different. Instead of creating one large program you will create three small programs, each that accomplishes a particular task.

1. Palindromes: write a program that reads words and uses stacks to determine whether or not the word is a palindrome.
2. Long Addition: write a program that reads two arbitrarily long integers (100s of digits) and uses stacks to add them.
3. Capital Gains: write a program that computes the capital gains or losses on a series of stock purchases and sales.

Like the first two assignments, these programs will all be run from the command line. There is a slight hitch with this. When creating a jar file, you can only specify one entry point, or main method but we will have three! You will be creating three main programs: `A3a.java`, `A3b.java` and `A3c.java`. You can create your jar file in the usual way including all `.java` and `.class` files.

```
jar cvf A3.jar *.class *.java
```

The finished programs will be run like this:

```
cat words.txt | java -cp A3.jar A3a > a3a_out.txt
cat numbers.txt | java -cp A3.jar A3b > a3b_out.txt
cat stocks.txt | java -cp A3.jar A3c > a3c_out.txt
```

This version of the `java` command says to add `A3.jar` to the `CLASS_PATH`, the list of places to look for Java class files and jar files. If a jar file is in the `CLASS_PATH` you can run any class that has a main method by simply specifying the class name. In this case, `A3a`, `A3c` and `A3c`.

I will provide sample input and output files for each portion of the program. The output from your program must be *Exactly* as shown in the examples.

Stacks and Queues

For `A3a` and `A3b` you will need a class that implements a stack and for `A3c` you will need a class that implements a queue. You should use your `SLL` class from assignment 2 as the basis for both the `Stack` and `Queue` classes.

In the lab you created a `Stack` that extended the `SLL`

```
pulic class Stack<T extends Comparable<T>> extends SLL<T>
```

and similarily for the `Queue` class. This will work, but may not be the best way. Why? What is another way?

A stack needs the following methods:

- `public T pop() //  Pop the top element from the stack.`
- `public void push( T e) // Add an element to the stack.`
- `public T peek()  // Return the top element, without popping it.`
- `public boolean isEmpty() // Return true if this stack is empty, false otherwise.`
- `public void empty() // Empty the stack.`

Remember that a stack is LIFO — Last In, First Out.

A queue needs the following methods:

- `public T dequeue() // Remove and return the first item added to the queue.`
- `public void enqueue() // Add an item to the stack queue, at the end of the queue.`
- `public T peek() // Return the first item in the queue, without removing it.`
- `public boolean isEmpty() // Return true if this queue is empty, false otherwise.`
- `public void empty() // Empty the queue.`

Remember that a queue is FIFO — First In, First Out.

You should make as much use as possible of the `SLL` class so that your `Stack` and `Queue` classes will be short.

Palindromes

A palindrome is any word that is the same forwards as backwards. In this case words need not be English and may consist of any non-whitespace characters. Case matters.

```
abba : Yes
ABba : No
123#$_$#321 : Yes
KAyAK : Yes
aabbbccccbbaa: Yes
```

- Create a class called `A3a`. It will have the `main()` method and do the bulk of the processing. Be sure to set it up to instantiate an `A3a` object.
- This program will read a series of words from the input, separated by *whitespace*. See notes below. For each word it will print it to the output file, append a colon, then the word "Yes" or "No" to indicate if this is a palindrome or not.
- Reading the words must be down character by character without putting it into a string. The only String in your program should be the `": Yes"` and `": No"` you use to print the results.
- All processing must be done via stacks. You will need more than one!

  You will need to push characters onto your stack. `char` is a primitive type and cannot be used in place of an object so you need to create a `Character` object and use that. See the Java documentation for constructors and methods of `Character`.

Long Addition

A Java `long int` is a 64 bit signed number with a maximum value of 9,223,372,036,854,775,807. What if you need to deal with numbers larger than that?

It is easy to represent huge numbers as strings of digits. Adding them together requires a bit of cleverness. Turns out stacks work very nicely for this problem.

- Create a class called `A3b`. It will have the `main()` method and do the bulk of the processing. Be sure to set it up to instantiate an `A3b` object.
- This program will read pairs of numbers and add them together and prints the result. The pair of numbers will appear on one line separated by whitespace. You can use a `Scanner` to read in the numbers as `String`s, or use your code from above.
- To do the addition you will use two stacks. It is up to you to figure out the algorithm to use.
- See the sample output for how to format the results. It must be exactly as shown.
- You can assume that the strings in the input are represent well formed integers.

Capital Gains

Imagine that you have a monthly standing order to buy some number of shares of Alphabet `https://abc.xyz/`, stock, currently the most valuable company in the world `http://www.vox.com/2016/2/4/10911364/google-apple-most-valuable`.

Each month the price of the share will different.

When you sell the stock you will realize a capital gain, or loss, but how much exactly? For example if you sell all your stocks at a given price, but you bought them at a variety of prices you will need to calculate the gain or loss based on those prices.

The rules say that for the purposes of calculating a capital gain you must sell the oldest stocks first and the gain is calculated by looking at the selling price minus the sum of the price that each stock was bought for.

- Create a class called `A3c`. It will have the `main()` method and do the bulk of the processing. Be sure to set it up to instantiate an `A3c` object.
- Buy and sell transactions will be read from a file, each transaction on one line: type of transaction, the number of share and the price.

  ```
  Buy 100 1
  Buy 10  2
  Buy 100 1
  Sell 50 3
  BUY 1 1
  Sell 100 2
  ```

  This will result in a capital gain of $190.
- The program should output a single line consisting of a signed integer representing the capital gain or loss.
- Use a queue to hold your stock transactions. Buy transactions arrive and are placed into the queue.

  Sell transactions area processed by taking as many shares as are needed from as many elements at the head of the queue as are needed.

Extras

If you wish to do a little extra here are two ways these programs could be improved.

1. In Long Addition add code to check that the input numbers are valid numbers. Error messages should be meaningful and printed to `System.err`. The program should recover from the error and go on to process the remaining entries in the file.
2. In the capital gains program add code to check for invalid operation, i.e. something other than BUY or SELL. Also check for invalid numbers. You cannot buy a negative number of stocks and stocks cannot have a negative

buying or selling price. Again, error messages should be meaningful and printed to `System.err`. The program should recover from the error and go on to process the remaining entries in the file.

Implementation Notes

Reading a file character by character

When we are reading tokens or lines a `Scanner` object works really well, but to read character to character it is better to use a slightly lower level object. A `BufferedReader` object has a `read()` method that reads one character at a time and returns the `int` code for that character. Set it up like:

```
private BufferedReader inp
       = new BufferedReader(new InputStreamReader( System.in));
```

`read()` returns -1 when end of file has been reached.

Whitespace

The program specification above states that words are separated by whitespace. What is whitespace? It is any character that occupies space but is otherwise blank. Whitespace characters are usually as shown in the table.

Character	Code	Description
HT	9	A horizontal tab.
LF	10	A line feed. Move to the next line below.
VT	11	A vertical tab.
FF	12	A form feed. A page break. Think about a pin-fold paper printer.
CR	13	Carriage return. From manual typewriter days, move the carriage (print head) back to the left side.
SPACE	32	A space.

Linux, Unix and Apple systems use a LF to indicate the end of a line. Windows systems use a combination of a CR and a LF to indicate the end of a line. Your program should work for both.

Testing

A will supply some test input and output files for you to use. These files do not constitute extensive testing. You will need to do more testing to ensure all

cases are covered.

I will be evaluating your program with test data that you have not yet seem and I will be trying to find flaws in your code!

What to Hand In

Hand in a single file, `A3.jar`.

Submit this to the Blackboard drop box provided. The jar should contain:

1. All of your `.class` and `.java` files.
2. Classes called `A3a`, `A3b` and `A3c`, each of which has a `main()` method.
3. Your `SLL` class, your `Stack` class and your `Queue` class.

Grading

A detailed grading rubric will be provided.

I will run your program with the command line given above on three text files and compare your output to the specifications.

Encapsulation is important! Good clean design is important! Pay close attention to keeping methods in the most appropriate classes.

Pay attention to the resulting big Oh of your code.

Outcomes

Once you have completed this assignment you will have:

- Implemented a Stack and a Queue;
- Used Stacks and Queue for problem solving;