Guidelines:

- **Make sure to submit your files to Google Classroom before the deadline,** otherwise, your work won't be considered for grading.
- **Submit only the java files as separate files (i.e. not as zipped files).**
- **Create a class for every program named based on the question name then write all the necessary methods and/or the main method**

Stack Implementation:

For the following questions, refer to the class files posted on the course classroom: MUArrayStack<E> and MULinkedStack<E>.

1. Override the **toString** method for the class MUArrayStack<E>. The stack should be printed as follows where 3 is at the top of the stack:
   3
   4
   5
   -9
   10
2. Override the **toString** method for the class MULinkedStack<E>. The stack should be printed just like part 1.
3. Write a **constructor** for the class MULinkedStack<E> that loads the stack from an array parameter. The last array element should be at the top of the stack.
   Constructor prototype: public MULinkedStack (E[] data);
4. Test your methods in a main method.

Stack Application Exercises:

Program 1:

1. Write a main function that creates three stacks of Integer objects. Store the numbers –1, 15, 23, 44, 4, 99 in the first two stacks. The top of each stack should store 99.
2. Write a loop to get each number from the first stack and store it into the third stack.
3. Write a second loop to remove a value from the second and third stacks and display each pair of values on a separate output line. Continue until the stacks are empty. Show the output.

Program 2:

Write a program that reads a line and reverses the words in the line (not the characters) using a stack. For example, given the following input:

The quick brown fox jumps over the lazy dog

you should get the following output:

dog lazy the over jumps fox brown quick The

Hints:

- Use split method of String class
- Use StringBuilder to create the new reversed string

Program 3: PalindromeFinder:

In a class PalindromeFinder, implement the static method isPalindromeLettersOnly(String input) that decides if the string is palindrome based only on the letters in a string (ignoring spaces, digits, and other characters that are not letters). Refer to the below example:

isPalindromeLettersOnly("I1saw 2 wasI") returns **true**

isPalindromeLettersOnly("I1saw2wasX") returns **false**

Write a main method to test your code.

Hint: Use the method isLetter in Character class

Program 4: Balanced Parenthesis:

A balanced parenthesis expression is an expression that has a matching closing parenthesis for every opening parenthesis. The allowed parentheses are: {,},(,),[,]. Refer to the below examples:

- [[{()}]] is balanced
- {[()]} is balanced
- (){}[] is balanced
- ()[{}] is balanced
- {}} is not balanced
- {}(is not balanced
- ({)} is not balanced
- [{()]} is not balanced

In a class BalancedParanthesis, implement the static method isBalanced(String exp) which returns **true** if the expression is balanced, **false** otherwise. All symbols other than the allowed parentheses are ignored.

Write a main method to test your code.

Hint: Use a stack to check if the expression is balanced.

Bonus Question: Sort a Stack

Write a method that gets a stack of integers and returns a sorted stack. The returned stack should have the top element as the maximum.

Hint: Use a temporary stack to complete the task.

```
Enter numbers, stop by inserting -1
1
-5
7
8
0
9
-1
Printing the original stack ....
9       0       8       7       -5      1
Sorting and printing the stack
9       8       7       1       0       -5
```