ComSc 112 - Lab 6 - Prof. Rager

1. **Either get the lab files from github**:

   git clone https://github.com/jerager/lab6

2. **Or get them from remus/romulus. Make a new directory for this lab:**

   ```
   mkdir lab6
       and move into that directory:
   cd lab6
   ```

   And Copy some Java files:

   ```
   cp $cs112home/lab6/*.java .
   ```

   (Printouts of the two files are attached to this lab.)

3. Compile and run UseStack.java Look at UseStack to make sure you understand it.

4. You are going to program several applications of the stack. You will be using CS112Stack<E> as described in class, and as demoed in UseStack. Please make a **new class** with a main method for each application (with a new name - be sure to change the name of the class in the public class declaration). You can copy UseStack.java as a starting point. (e.g. cp UseStack.java Reverser.java)

   a. Write a program called Reverser.java to read a String from the keyboard, use a Stack of characters to help create a String that is the reverse of the original and then print that String. You will need to know that if s is a String variable, then:

   s.length() is the number of characters in s
   s.charAt(int i) is the ith character of the String s (starting at 0)

   The reversing should be done in a method with signature String reverse(String) This method should do neither input nor output.

   You must use a CS112Stack. The wrapper class for char is Character.

b. Write a program called Parentheses.java to read a String from the keyboard and check its parenthesis matching. Write a method with signature boolean balanced(String) that checks the matching of parenthesis characters in its String argument. This method should use a Stack, not recursion. You should ignore all characters except (,),{,},[, and]. (Hint: if you get an opening character, push it, if you get a closing character what do you do?)

Your program should handle all cases of matching parentheses, eg. the following sequence are OK:

([()])
([][[]))
()[](())

the following are not:

([)]
())
((()

c. Write a program called PostFixEval to read an expression from the keyboard and evaluate it as a postfix expression. Write a method with signature int value(Scanner) to do the evaluation. (You can pass the Scanner that is the keyboard to the method). The method must read the expression from the Scanner. A postfix expression looks like this:

1 2 + 3 * 14 - =

In evaluating it, when you encounter a number, you save it (where? on the stack!) when you encounter a binary operator (+,-,*,/), you use it to combine the last two saved numbers (which are then no longer saved) and save the result. When you encounter an "=" return the saved result (and end the process).

The expression above is evaluated as follows:

Stack contents:
1
1 2
3
3 3
9
9 14
-5 (note the order - the second item on the stack minus the top item)

prints: -5

To do this, you need to be able to recognize whether the Scanner contains an int before you read the next thing from the Scanner. The Scanner class has a method hasNextInt(), which returns true if the NEXT thing to be read is an int. So, if keyboard.hasNextInt() is true, you know the next token is an int, and you should do keyboard.nextInt(), otherwise, use next() to read a String

d. Create a class to store an Operator. The class should contain a char (*,-,/,+) and an int which is the precedence. The base values of the precedences are:

+, - 1
*, / 2

Use a stack to write a program called Convert to convert an infix expression to a postfix expression. Your working method should have signature String convert(Scanner) - read the information from a Scanner and put the result in a String.
e.g.

4 + 5 * 8 = becomes

4 5 8 * + =

4 * 5 + 8 = becomes

4 5 * 8 +

The algorithm is this:

1. If you get an number, put it into the output expression

2. If you get a = pop all the operators on the stack, writing the chars into the output expression

3. If you get a char which is an operator,
   - create an Operator, setting the precedence to the appropriate base precedence
   - as long as the precedence of the operator on the stack is >= the new one, pop the stack, writing the chars into the output expression
   - push the operator

4. If you get a "(" raise all the base precedences by 4 (question – where should the base precedences be stored?)

5. If you get a ")" lower all the base precedences by 4

(Try it by hand so you see how it works. **We will discuss this algorithm in class, so feel free to wait until after that discussion to work on this part of the lab**)

Once you have this working, use your method from Part c to evaluate the resulting postfix expressions. To do this you will need to create a Scanner to read a String, e.g.

String s = “12 a 123 b”;
Scanner readS = new Scanner(s);

int a = readS.nextInt(); // this will read 12

Submitting your work

As usual, when your work is complete, submit your programs to me online:

submit CS112Lab6 whatever files go here

(You can list as many files as you want. Please submit ALL the files that are needed for the lab, and please submit them all together.)

```
public class UseStack {

     public static void main(String[] args) {

    UseStack use;
    use = new UseStack();
    use.go();

     }

     public void go() {

    CS112Stack<Character> s = new CS112Stack<Character>();

    for (char i='a'; i<='z';i++)
         {
        s.push(i);
         }

    while (!s.isEmpty())
         {
        System.out.print(s.pop());
         }

    System.out.println();
     }

}
```

```
import java.util.ArrayList;

public class CS112Stack<E> {

    ArrayList<E> theStack = new ArrayList<E>();

    public void push(E toPush) {
        theStack.add(toPush);
    }

    public boolean isEmpty() {
        return theStack.size() == 0;
    }

    public E pop() {
        if (isEmpty())
            return null;
        else {
            E top = theStack.remove(theStack.size()-1);
            return top;
        }
    }

    public E top() {
        if (isEmpty())
            return null;
        else {
            E top = theStack.get(theStack.size()-1);
            return top;
        }
    }

    public void printStack() {
        for (E element : theStack)
            System.out.println(element);
    }




}
```