## Topics

1. Create Stack Interface
2. Create Stack Using Array
3. Create Stack Using Linked Lists
4. Implement Basic Methods of Stack

* isEmpty()
* size()
* top()
* push(E e)
* pop()

## Homework

1. Implement a method with signature transfer(S, T) that transfers all elements from stack S onto stack T, so that the element that starts at the top of S is the first to be inserted onto T, and the element at the bottom of S ends up at the top of T.

public static <E> void transfer(Stack<E> S, Stack<E> T) {  
    while (![S.isEmpty](http://S.isEmpty)()) {  
        [T.push](http://T.push)([S.pop](http://S.pop)());  
    }  
}

1. Give a recursive method for removing all the elements from a stack.

public static <E> void clearStack(Stack<E> stack) {  
    if (![stack.isEmpty](http://stack.isEmpty)()) {  
        [stack.pop](http://stack.pop)();  
        clearStack(stack); // استدعاء ذاتي لإزالة العنصر التالي  
    }  
}

1. Postfix notation is an unambiguous way of writing an arithmetic expression without parentheses. It is defined so that if “(exp1)op(exp2)” is a normal fully parenthesized expression whose operation is op, the postfix version of this is “pexp1 pexp2 op”, where pexp1 is the postfix version of exp1 and pexp2 is the postfix version of exp2. The postfix version of a single number or variable is just that number or variable. So, for example, the postfix version of “((5 + 2) ∗ (8 − 3))/4” is “5 2 + 8 3 − ∗ 4 /”. Describe a nonrecursive way of evaluating an expression in postfix notation.

لتقييم تعبير بوستفيكس غير تكراري:  
  
1. استخدم مكدس لتخزين النتائج الوسيطة.  
  
2. قم بمعالجة التعبير من اليسار إلى اليمين:  
  
إذا صادفت رقمًا، ادفعه إلى المكدس.  
  
إذا صادفت عملية (مثل +, -, \*, /):  
  
أخرج عنصرين من المكدس.  
  
نفّذ العملية.  
  
أعد دفع النتيجة إلى المكدس.  
  
3. في النهاية، القيمة المتبقية في المكدس هي الناتج النهائي.

1. Implement the clone( ) method for the ArrayStack class.

public ArrayStack<E> clone() {  
    ArrayStack<E> clonedStack = new ArrayStack<>([this.data.length](http://this.data.length));  
    [clonedStack.size](http://clonedStack.size) = [this.size](http://this.size);  
    for (int i = 0; i < [this.size](http://this.size); i++) {  
        [clonedStack.data](http://clonedStack.data)[i] = [this.data](http://this.data)[i];  
    }  
    return clonedStack;  
}

1. Implement a program that can input an expression in postfix notation (see Exercise C-6.19) and output its value

import [java.util.Stack](http://java.util.Stack);  
  
public class PostfixEvaluator {  
    public static int evaluatePostfix(String expression) {  
        Stack<Integer> stack = new Stack<>();  
  
        for (String token : [expression.split](http://expression.split)(" ")) {  
            if (isOperator(token)) {  
                int b = [stack.pop](http://stack.pop)();  
                int a = [stack.pop](http://stack.pop)();  
                [stack.push](http://stack.push)(applyOperator(token, a, b));  
            } else {  
                [stack.push](http://stack.push)([Integer.parseInt](http://Integer.parseInt)(token));  
            }  
        }  
  
        return [stack.pop](http://stack.pop)();  
    }  
  
    private static boolean isOperator(String token) {  
        return "+-\*/".contains(token);  
    }  
  
    private static int applyOperator(String operator, int a, int b) {  
        switch (operator) {  
            case "+": return a + b;  
            case "-": return a - b;  
            case "\*": return a \* b;  
            case "/": return a / b;  
            default: throw new IllegalArgumentException("Invalid operator");  
        }  
    }  
  
    public static void main(String[] args) {  
        String expression = "[5 2](tel:5%202) + [8 3](tel:8%203) - \* 4 /";  
        [System.out.println](http://System.out.println)("Result: " + evaluatePostfix(expression));  
    }  
}