1. Download PostfixEvaluator.java from ~dxu/handouts/labs. Study the code. Test it using TestPostfixEvaluator.java. Write a Java program to convert a postfix expression to a parenthesized infix expression. Read a postfix expression from the user. Convert the postfix expression to infix and display it to the user. The operators to be considered are +, -, *, /, %. This program should be very similar to PostfixEvaluator.java.

Sample Input 1
5 6 + 9 *

Output
( ( 5 + 6 ) * 9 )

Sample Input 2
8 9 10 + *

Output
( 8 * ( 9 + 10 ) )

2. Implement a DoubleStack class such that

- a single underlying array stores two different stacks (stack 1 and stack 2), one grows from index 0 upward, one grows from the end of the array down. So these two stacks grow towards each other. The top indexes are denoted by top1 and top2 for stack 1 and stack 2, respectively. Thus, there are three instance variables: E[] theArray, int top1, int top2
- theArray locations 0 to top1 contain elements in stack 1 and theArray locations theArray.length-1 downto top2 stores the elements in stack 2.
- Write methods
  - push(int stackId, E e): push e onto stack stackId (1 or 2). In other words, it will push onto stack 1 if stackId==1 and onto stack 2 if stackId==2. Throw an IllegalStateException if stack is full - for now.
  - E pop(int stackId): pop from stackId, return null if empty.
  - E top(int stackId): top elementn from stackId, return null if empty.
  - int size(int stackId): return size of stack stackId
  - boolean isEmpty(int stackId)
  - printStack(int stackId)
that will implement the push, pop, top, size, isEmpty, and printStack operations for the stack given by parameter stackId (1 for stack 1, 2 for stack 2)

3. Change push so that if the array gets full, instead of throwing an exception, resize the array to double size.