```java
//David Schmidt
//CIS1154
//2-13-2017
//This program converts infix notation into postfix notation.

import java.util.Scanner; //To read the infix expression in

public class StackTester
{
  private static boolean isOperand(char c)
  {
    return (c >= 'a')&&(c <= 'z')||(c >= 'A')&&(c <='Z')||(c >= '0')&&(c <= '9');
  }//isOperand

  private static boolean isOperator(char c)
  {
    return (c == '+')||(c == '-')||(c == '*')||(c == '/');
  }//isOperator

  private static boolean higherPrec(char a, char b)
  {
    return ((a == '*')||(a == '/'))&&((b == '+')||(b == '-'));
  }//higherOrEqualPrec

  private static boolean isParen(char c)
  {
    return (c == '(')||(c == ')');
  }//isParen

  public static void main (String [] args)
  {
        //Read the expression
        CharStack s = new CharStack();
        Scanner scan = new Scanner(System.in);

        System.out.println("Enter infix expression: ");
        String inFix = scan.nextLine();
        String postFix = "";
        char current;

        //Conversion from infix to postfix
        for (int i = 0; i < inFix.length(); i++)
        {
                current = inFix.charAt(i);

                //Process the parentheses
                if (isParen(current))
                {
                   if (current == '(')                          //if left paren, push
                      s.push(current);
                   else if (current == ')')                     //if right paren, pop
                   until matching '(' is found
                   {
                     if (!s.isEmpty())
                       while(s.peek() != '(')
                       {
                         postFix = postFix+s.pop();
                       }//while                                 //discard '('
                       if (!s.isEmpty())
                         s.pop();                               //Error-if you want to
                         pop an empty stack too many right parens
                   }//else if
                }//if isParen

                //Process the operators
                if (isOperator(current))
                {
                   if ((s.isEmpty() || s.peek() == '('))       //if empty or '(', push
                   current
```

```
67                      s.push(current);
68                    else if
                      (higherPrec(current,s.peek()))                    //if
                      current is higher precedence push it
69
                        s.push(current);
                        //current must be lower or equal precedence so pop until
70
                      else
                      //top of stack is empty or current has higher precedence
71
                      {
                      //...and push current
72                      while((!s.isEmpty())&&(!higherPrec(current,s.peek())))
73                        postFix = postFix+s.pop();
74                      s.push(current);
75                    }//else
76                  }//if isOperator
77
78                  //Process the operands
79                  if (isOperand(current))
80                    postFix = postFix+current;
81
82          }//for
83
84          //All characters have been read so pop and write until empty
85          while (!s.isEmpty())
86            postFix = postFix+s.pop();
87
88
89          //Write the postfix expression out
90          System.out.println("Converted to postfix: "+postFix);
91
92      }//main
93
94  }//StackTester
95
```