Infix to postfix Algorithm

1. Print the operand as they arrive.
2. If the stack is empty or contains a left parenthesis on top, push the incoming operator on to the stack.
3. If the incoming symbol is '(', push it on to the stack.
4. If the incoming symbol is ')', pop the stack and print the operators until the left parenthesis is found.
5. If the incoming symbol has higher precedence than the top of the stack, push it on the stack.
6. If the incoming symbol has lower precedence than the top of the stack, pop and print the top of the stack. Then test the incoming operator against the new top of the stack.
7. If the incoming operator has the same precedence with the top of the stack then use the associativity rules. If the associativity is from left to right then pop and print the top of the stack then push the incoming operator. If the associativity is from right to left then push the incoming operator.
8. At the end of the expression, pop and print all the operators of the stack.

infix exp ⇒ A*B-C(C+D)+E

Input character	operation on stack	stack	postfix exp
A		empty	A
*	Push	*	A
B		*	AB
-	check and push	-	AB*
((open parenthesis)	Push	-(	AB*
C		-(	AB*C
+	check and push	-(+	AB*C
D		-(+	AB*CD
)	Pop and append to postfix till '('	-	AB*CD+
+	check and push	+	AB*CD+-
E		+	AB*CD+-E
end of input	pop till empty		AB*CD+-E+

AB*CD+-E+

A*B-(C+D)+E
Step1 ⇒ AB* - CD+ +E
Step2 ⇒ AB*CD+- +E
Step3 ⇒ AB*CD+-E+ → postfix

infix exp ⇒ ((A+B)-C*(D/E))+F → postfix
L → R
AB+CDE/*-F+

input character	operation on the stack	stack	postfix exp
(	Push	(	
(	Push	((	
A		((	A
+	Push	((+	A
B		((+	AB
)	Pop and append to postfix till '('	(	AB+
-	Push	(-	AB+
C		(-	AB+C
*	Push	(-*	AB+C
(	Push	(-*(	AB+C
D		(-*(	AB+CD
/	Push	(-*(/	AB+CD
E		(-*(/	AB+CDE
)	Pop and append to postfix till '('	(-*	AB+CDE/
)	Pop and append to postfix till '('		AB+CDE/*-
+	Push	+	AB+CDE/*-
F		+	AB+CDE/*-F
end of input	pop till empty		AB+CDE/*-F+

postfix exp

AB+CDE/*-F+

infix to postfix conversion by program

```
public class InfixToPostfix {
    public static int checkOperatorPrecedence(char ch) {
        switch (ch) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
        }
        return -1;
    }

    public static String infixToPostfix(String exp) {
        //String exp = "A*B-(C+D)+E";
        String postfixExpression = "";
        Stack<Character> stack = new Stack<>();
        for (int i = 0; i < exp.length(); i++) {
            char ch = exp.charAt(i);
            if (Character.isLetterOrDigit(ch)) {
                postfixExpression = postfixExpression + ch;
            } else if (ch == '(') {
                stack.push(ch);
            } else if (ch == ')') {
                while (!stack.isEmpty() && stack.peek() != '(') {
                    postfixExpression = postfixExpression + stack.peek();
                    stack.pop();
                }
                stack.pop();
            } else {
                while (!stack.isEmpty() && checkOperatorPrecedence(ch) <= checkOperatorPrecedence(stack.peek())) {
                    postfixExpression = postfixExpression + stack.peek();
                    stack.pop();
                }
                stack.push(ch);
            }
        }
        while (!stack.isEmpty()) {
            postfixExpression = postfixExpression + stack.peek();
            stack.pop();
        }
        return postfixExpression;

    }

    public static void main(String[] args) {
        //String exp = "A*B-(C+D)+E";
        //String exp = "((A+B)-C*(D/E))+F";

        //String exp = "(A+B)*C-(D-E)*(F+G)";
        //String exp = "(((A+B)*C)-((D-E)*(F+G)))";
        String exp = "A+B*C/D-F+A*E";
        System.out.println("infix expression : "+exp);
        System.out.println("postfix expression : "+infixToPostfix(exp));
    }

}
```