```
public class Conversor {
    public int converter(String expressao) {
        char[] caracteres = expressao.toCharArray();

        PilhaInt operandos = new PilhaInt();

        Pilha operador = new Pilha();

        for (int i = 0; i < caracteres.length; i++) {

            // se for espaço, pula.
            if (caracteres[i] == ' ') continue;

            // se for um número, coloca ele dentro da pilha de números.
            if (caracteres[i] >= '0' &&
                caracteres[i] <= '9') {
                //uma string que pode ser modificada a qualquer momento a
través de métodos especiais.
                StringBuffer sbuf = new StringBuffer();

                // pode haver mais de um dígito num número.
                while (i < caracteres.length &&
                           caracteres[i] >= '0' &&
                           caracteres[i] <= '9')
                    sbuf.append(caracteres[i++]);
                    operandos.push(Integer.parseInt(sbuf.toString()));

                // agora o i aponta para o caractere proximo ao digito, j
á que o for também aumenta i, nós iríamos pular uma position. Por isso nó
s precisamos diminiur o valor de i por um para corrigir o valor de i.
                  i--;
            }

            //Se o caractere atual for um (, empurra ele pra dentro da pi
lha de operadores.
            else if (caracteres[i] == '(') operador.push(caracteres[i]);
            // se o ) for encontrado,
            // resolva toda a pilha.
            else if (caracteres[i] == ')') {
                while (operador.peek() != '('
                       operandos.push(aplicarOperador(operador.pop(),
                                                      operandos.pop(),
                                                      operandos.pop()));
                operador.pop();
            }

            // se o caractere atual for um operador.
            else if (caracteres[i] == '+' ||
                     caracteres[i] == '-' ||
```

```
                    caracteres[i] == '*' ||
                       caracteres[i] == '/') {
                // Enquanto o topo de operador tiver uma precedenciaigual
 ou maior que o caractere atual, que é um operador: Aplique o operador no
 topo de operador para amontoar dois elementos na pilha dos operandos.
                while (!operador.isEmpty() &&
                       temPrecedencia(caracteres[i], operador.peek()))

                       operandos.push(aplicarOperador(operador.pop(),
                       operandos.pop(),
                       operandos.pop()));

                // empurrar o caracterer atualpara dentro de operador.
                operador.push(caracteres[i]);
            }
        }

        // Toda a expressão já foi convertida até esse ponto, aplique o o
perador restante para os operandos que restaram;
        while (!operador.isEmpty())
            operandos.push(aplicarOperador(operador.pop(),
                                           operandos.pop(),
                                           operandos.pop()));

        //O topo de operandos contém o resultado, retorne-o.
        return operandos.pop();
    }

    // retorna true se op2 tem uma precedência maior ou igual a op2, caso
 contrário ele retorna false.
     public boolean temPrecedencia(char op1, char op2) {
        if (op2 == '(' || op2 == ')')
            return false;
        if ((op1 == '*' || op1 == '/') &&
            (op2 == '+' || op2 == '-'))
            return false;
        else
            return true;
    }



     public int aplicarOperador(char op, int b, int a) {
        switch (op) {
            case '+':
                return a + b;
            case '-':
                return a - b;
            case '*':
```

```
                return a * b;
            case '/':
                if (b == 0)
                    throw new UnsupportedOperationException("Não dá pra d
ividir por zero.");
                return a / b;
        }
        return 0;
    }

}
```