```
public class Alocacao {
    StackInt valores;
    StackChar operadores;

    public Alocacao() {
        this.valores = new StackInt();
        this.operadores = new StackChar();
    }

    public int alocador(String infixa) {
        char[] digitos = infixa.toCharArray();
        for (int x = 0; x < digitos.length; x++) {

            if(digitos[x] == ' ') continue; // ignorar espaços brancos.
            if(digitos[x] >= '0' && digitos[x] <= '9') {

                StringBuffer auxiliar = new StringBuffer();
                //  no momento que o if começa ele acaba pegando todos os
 caracteres de 0 a 9 dentro de digito

                while(x < digitos.length && digitos[x] >= '0' && digitos[
x] <='9') {
                   auxiliar.append(digitos[x++]);
                }
                valores.push(Integer.parseInt(auxiliar.toString()));

                x--;
            } else if (digitos[x] == '(') {
                operadores.push(digitos[x]);
            } else if (digitos[x] == ')') {
                while (operadores.peek() != '(') {
                    valores.push(aplicarOperadores(operadores.pop(), valo
res.pop(),valores.pop()));
                }
                operadores.pop();
            }
             // Se for operador.
            else if (digitos[x] == '+' ||
                     digitos[x] == '-' ||
                     digitos[x] == '*' ||
                     digitos[x] == '/') {
                while (!operadores.isEmpty() && temPrecedencia(digitos[x]
, operadores.peek())) {
                    valores.push(aplicarOperadores(operadores.pop(), valo
res.pop(), valores.pop()));
                }
                //entrou grandão, pop; entrou prqueno push push.
```

```
                // Se o valor de digitos[x] tiver uma precedência maior q
ue o valor no topo da pilha operadores, vai dar false. Caso contrário, tr
ue e vai entrar ali naquele laço de repetição. Lá eles vão colocar dentro
 da stacl de operadores o valor da soma de valor + operador + valor. GENI
AL. FUCKIN GENIAL.
                operadores.push(digitos[x]);
            }
        }
        while (!operadores.isEmpty()) {
            valores.push(aplicarOperadores(operadores.pop(), valores.pop(
), valores.pop()));
        }
        return valores.pop();
    }

    public int aplicarOperadores(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 dividir por 0");
            return a / b;
        }
        return 0;
    }
    public static boolean temPrecedencia(char op1, char op2)
    {
        //Se aquele que tem uma precedência maior estiver entrando(op1),
eu retorno false.
        //Se aquele que está entrando(op1), tem uma precedência maior que
 aquele que já está dentro(op2), false.
        // Agora, se a precedência de op1 é menor que op2, true.
        if (op2 == '(' || op2 == ')')
            return false;
        if ((op1 == '*' || op1 == '/') && (op2 == '+' || op2 == '-'))
            return false;
        else
            return true;
    }
}
```