Integrantes

Guilherme Mateus. Ra: 20779723.

Reginaldo Campos. Ra: 21008225.

Rafael Carmignotto Ra: 20991254.

Luiz Cerqueira Ra: 20843482.

Mauricio Sorato Ra: 20900180.

# Objetivo

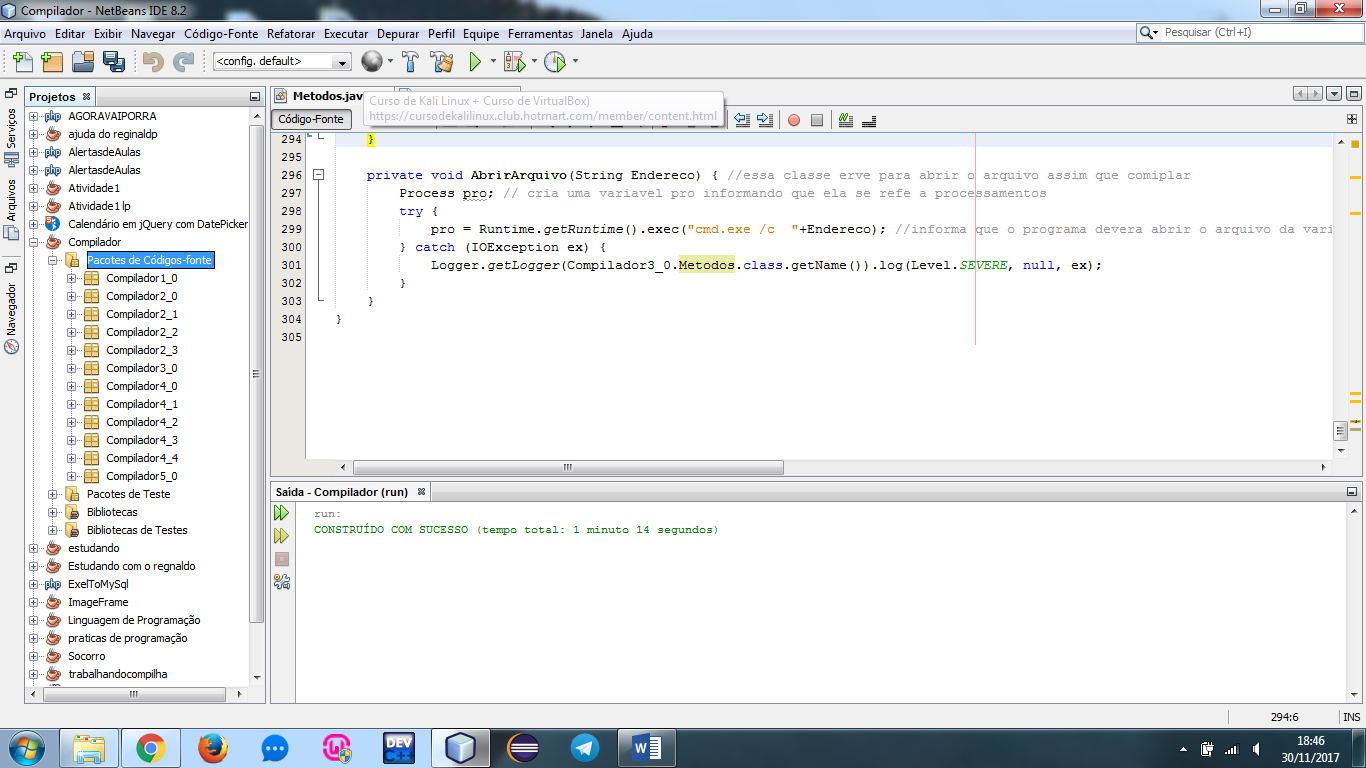
Nosso trabalho teve como objetivo o desenvolvimento de um compilador de linguagem assemble, utilizamos a linguagem de programação Java para isso, onde ele receberá uma expressão matemática e essas expressões seguem as seguintes regras:

Somente letras e essas letras não podem ser “negativas” Ex: (-A) dentre outras regras que estarão na parte final desse arquivo com o título de Alertas e Instruções para uso, onde ele teria que respeitar as regras da matemática, de sinais, resolver essas expressões e desenvolver o código assemble.

# Desenvolvimento do projeto

O desenvolvimento do projeto não foi nada fácil, tivemos muitas dificuldades por conta de ser um projeto que exige uma complexidade maior, porém alcançamos resultados animadores, o projeto foi desenvolvido totalmente em POO (Programação Orientada a Objetos) pelo motivo de ser mais seguro, reaproveitamento de código e também de fácil leitura, ainda mais que o programa foi feito totalmente comentado para que até as pessoas que está iniciando agora consigam entender o funcionamento do mesmo.

Chegar ao resultado final desse programa não foi nada fácil, desenvolvemos e otimizamos ele muito, como mostra na figura abaixo até chegarmos no programa que foi finalizado.



O compilador foi projetado para suportar expressões com até cinco grupos de parênteses, além disso conta com um excelente sistema de reaproveitamento de variáveis por conta que ele só utiliza seis variáveis para a resolução da expressão e assim que acontece uma operação com uma das variáveis ele automaticamente zera o seu conteúdo par que ela seja utilizada novamente.

Além de tudo nosso programa gera dois arquivos de texto (.txt) um arquivo que contém toda a resolução da expressão com as instruções em assembly corretas para inicialização e finalização da aplicação e também gera um arquivo de “log” onde mostra passo a passo a expressão como foi resolvida e também mostra as variáveis que estão sendo trabalhadas juntamente com o AC.

# Funcionamento e Métodos

Atenção todas as classes que possuem: throws IoException tem escrita de informações em arquivos, por isso o throws para fazer o tratamento de erros na própria classe ao invés de usar vários try catch’s para cada instrução escrita.

## **Processamento(String expressao):**

O método processamento é utilizado para a resolução da expressão, assim que ele recebe a mesma do txtExpressão é chamado o medo processamento(expressao) que recebe como parâmetro a expressao digitada pelo usuário.

Assim que recebida ele retira os espaços em brancos do começo e final da string utilizando o .trim(), após isso ele utiliza o .toUpperCase() para deixar a expressão com todas as letras em maiúsculas.

Cria três variáveis para a validação da expressao int parenF = 0, parenA = 0; boolean verifica = false; parenF significa parênteses fechado e paranA significa parênteses abertos, nessa parte do código elas servem analisar quantos parênteses tem abertos e fechados na expressão enquanto a variável verifica inicia em false para verificação se existem dígitos na mesma.

A partir daí ele entra em um laço com três if’s um para verificação de dígitos usando: Character.isDigit(expressao.charAt(i)) == true, se for igual a true significa que tem números e a variável verifica = true, depois ele verifica se tem parênteses abertos: expressao.charAt(i) == '(' se for verdadeiro parenA++; e o terceiro if verifica os parênteses fechados usando: expressao.charAt(i) == ')' se verdadeiro: parenF++.

Após isso ele faz a verificação da expressão com um if assim: if (parenA != parenF || verifica == true) se as variáveis parênteses ou então a variável verifica for igual a true então eles não seguem as regras básicas para a expressão, que são a mesma quantidade de parênteses abertos tem que ser igual a de parênteses fechado e também não pode conter números na expressão, se alguma das duas forem verdadeiras ele mostra uma mensagem de erro de sintaxe da expressão e fecha o programa.

Se a expressão estiver dentro dos padrões ele entra no else e começa a criação do desenvolvimento da expressão, após isso ele começa chamando o método Criartxt() que cria os arquivos .txt, logo após isso ele escreve no arquivo de log a expressão para o controle da resolução naquele arquivo e chama o método CarregaInicio() onde ele carrega o inicio de qualquer programa em assemble.

A partir daí começa o laço Do While que a condição de saída é (expressao.length() != 1) ou seja ele só sai do laço quando a expressão estiver resolvida com apenas um caractere, nesta parte do código ele reutiliza as variáveis parenA e parenF, parenA começa em 0 e parenF ele inicia no primeiro parênteses fechado com o parenF = expressao.indexOf(")") e cria a variavel quebra onde futuramente ele vai fazer as quebras dentro dos parênteses e colocar nessa variável, após isso ele inicia em for assim:

for (int i = expressao.indexOf("("); i < parenF; i++) {

if (expressao.charAt(i) == '(')

parenA = i;

}

O int I inicia a partir do primeiro parêntese aberto e i tem que ser menos que a posição do primeiro parêntese fechado, dentro dele tem um if para verificar se tem outro parêntese aberto entre esses dois parênteses, caso tenha outro parênteses aberto: parenA = a posição desse parêntese Ex: ((A+B)-C) parenA = 1 e parenF = 5, caso não tenha parenA continua com 0.

Depois vem um if que verifica se parenF == 0, parenA ele recebe o primeiro parêntese aberto com o indexOf, e por último if ele verifica se parenF < 0, se for menos significa que não existe parênteses, então ele chama a variável expressao e fala que ela é igual ela entre parênteses com essa linha de código: expressao = "(" + expressao + ")"; e faz a mesma coisa com a variável quebra: quebra = "(" + expressao + ")"; pelo seguinte motivo no final da resolução da expressão após ter resolvido todos os parentes chega uma hora que a expressão fica sem nenhum parênteses porém existem regras de sinais para a sua resolução, quando a expressão fica sem parênteses entra nessa condição do if e coloca parênteses entre ela.

Se ele não entrar nesse If a variável quebra ela recebe: quebra = expressao.substring(parenA, parenF + 1); ou seja ela cria uma nova subString somente com os parênteses mais internos para serem resolvidos primeiro, por isso a importância de pegar sempre o parenA e parenF ou seja o primeiro parêntese aberto e o primeiro parêntese fechado.

Assim que ele já estiver a quebra dos parênteses mais internos n variável quebra ele passa essa mini expressão a ser resolvida para o método Resolve(); e pega o seu retorno na variável aux.  
Exemplos: Se quebra for: (A+B/C) o retorno em aux será: A+U, ou então se quebra for igual a: (A+B) o retorno em aux seria a letra U que ele resolveria A+B e colocaria na variável , mostraria a operação para o usuário mostrando qual letra ele carregou, a operação e aonde ele armazenou.

Nesta parte acontece o principal motivo para a resolução da expressão, ele pega a variável expressao, chama expressao.repleace e pede para ele substituir o que tem em quebra para o retorno de aux, porém dentro da expressão.

Exemplo: expressao = ((A+B)-C), quebra = (A+B) e aux = U, faríamos a substituição com a seguinte linha de código: expressao = expressao.replace(quebra, aux); a nova expressão seria (U-C) e escreve no arquivo log a nova expressão, o programa fica no laço com todos esses comandos ate a expressao = 1, quando for assim a expressão estará resolvida.

Após isso ele chama o método CarregarFinal() onde ele mostra e escreve no txt arquivo o final que todo o programa em assembler deve conter para finalizar e ele chama o método FecharArquivo() onde ele fecha os arquivos log e arquivo.

## **Resolve(String aux)**

O método Resolve ele recebe como parâmetro a variavel quebra para fazer a resolução dessa quebra e retorna a expressão ou então uma parte dela resolvida.

Ele já inicia o método criando uma variável boolean divMult = false; serve para verificar se existe multiplicação ou divisão na quebra ou melhor agora a quebra seria aux, depois da criação ele retira os parênteses da expressão com a seguinte linha de código: aux = aux.substring(1, aux.length() - 1); e ainda cria duas String quebras e novaExp que se iniciam vazias.

Após isso ele entra dentro de um for para a verificação se existe \* ou / usando um for seguido de um if assim:

for (int i = 1; i < aux.length(); i++) {

if (aux.charAt(i) == '\*' || aux.charAt(i) == '/') {

String op1 = "" + aux.charAt(i - 1), op2 = aux.charAt(i + 1) + "";

quebras = op1 + aux.charAt(i) + op2;

quebras = quebras.trim();

Load(aux.charAt(i - 1));

divMult = true;

i = aux.length();

}

}

Se ele achar um \* ou / ele cria duas strings: String op1 = "" + aux.charAt(i - 1), op2 = aux.charAt(i + 1) + ""; que pega um caractere antes e um caractere depois do sinal, adiciona em quebras: op1+ aux.charAt(i) (o sinal da operaçao)+op2, após ele retira os espaços do começo e final da string com o .trim(), chama o método Load() passando como parâmetro para a expressão a primeira letra do quebra, após isso ele muda o estado de divMult para true e informa que i=aux.length() para sair do for assim que ele encontrar o primeiro sinal de \* ou /.

O próximo if ele verifica se quebras esta vazio, se quebra estiver vazio significa que ele não possui divisão ou multiplicação, sendo assim ele chama o Load() e passa como parâmetro a primeira letra da variável aux.

Após isso ele entra em um if só pra ter certeza que não estamos tratando uma expressão assim por exemplo (U), ai como na entrada desse método retira os parênteses ficaria só U, ou seja a expressão estaria resolvida, esse if: if (aux.length() != 1) resolve esse problema.

Em seguida ele vai verificar as duas outras possibilidades para a quebra de expressão:

if (aux.length() > 3 && divMult != true)

quebras = aux.substring(0, 3);

Caso aux seja maior que três e não possua divisão ou multiplicação, Ex: A+B-C. Quebras recebe uma substring da posição 0 até 3 da expressão aux, ou seja, quebras = A+B.

Se não:

else if (aux.length() <= 3 && divMult != true) {

quebras = aux;

Caso o tamanho de aux seja menor ou igual a 3 e não possuir divisão ou multiplicação quebras = aux, Ex: A+B.

Depois de ter feito a quebra ele faz uma das partes mais importantes do programa, a resolução da expressão quebra, é usado um switch case para fazer a escolha:

switch (quebras.charAt(1)) {

case '+':

Add(quebras);

break;

case '-':

Sub(quebras);

break;

case '\*':

Mpy(quebras);

break;

case '/':

Div(quebras);

break;

}

O switch recebe quebras.charAt(1) que sempre será a posição do sinal daquela operação e mediante a esse sinal que ele entra no case e chama os métodos de resolução que são: Add(quebras), Sub(quebras), Div(quebras) e Mpy(quebras), esss funções imprimem na tela a operação que foi feita.

Depois de “resolvida” a quebra ela precisa ser substituída pela variável corresponde para ser retornada e ser substituída na expressão do processamento, sendo assim, agora aux recebe o retorno do método Story(quebras) que retorna a letra que aquela resolução de quebra foi inserida na memória.

Após esse tratamento e resolução da variável quebras acontece algo bem semelhante ao do processamento(), a String novaExp recebe ela mesma substituindo o conteúdo de quebras pelo conteúdo aux assim: novaExp = novaExp.replace(quebras, aux), após isso ocorre um if para o tratamento da expressão, caso o tamanho de novaExp seja diferente de 1 ele vai colocar parênteses entre a string novaExp assim: novaExp = "(" + novaExp + ")”, para que o próximo parênteses a ser resolvido seja esse aqui que sobrou de quebra.

Ex: (A+B/C) após o tratamento e resolução ele fica A+U, após isso entra no ultimo isso e sai assim (A+U) o método Resolve() devolvera essa expressão e como ela está entre parênteses será a próxima a ser tratada e a retorna par a substituição na variável expressao.

## **Load(char op1)**

Um método bem simples, porém, muito importante é ele que mostra ao usuário qual a letra que está sendo movida para memória ou melhor para o AC.

Após ele receber esse char ele escreve essa informação tanto no arquivo como no log, a diferença no log, é que essa informação tem mais complexibilidade e assim o usuário pode acompanhar todo o processamento do programa, Exemplo de como apareceria no .txt do log: LOAD X AC=X.

**Story(String aux) E return aux.**

Esse método que faz o gerenciamento de onde será armazenado essas informações e quais variáveis estarão sendo utilizadas ou livres, além de avisar para o usuário qual a variável que está sendo armazenado a resolução.

Assim que o método inicia ele criar a variável zerarVar que recebe aux, essa variável vai servir para zerar a variável caso ela tenha sido utilizada na expressão anterior para que possa ser utilizada novamente.

Após isso ele verifica qual das variáveis está vazia para retornar a letra dela, as variáveis vão de U até Z, caso a variável esteja vazia ele informa que aux recebera aquela letra da variável e essa variável recebe um, indicando que já está em uso, sendo assim o próximo passo foi criar um for para percorrer a variável zerarVar, chamamos o método ZerarVar(char letra) passando como parâmetro zerarVar.charAt(i) e todas as letras que estiverem naquela expressão que são variáveis receberiam um 0 dentro do método ZerarVar() para serem utilizadas novamente.

O processo seguinte é bem parecido com o Load() ele grava essas informações nos dois arquivos de texto que temos, lembrando que sempre o log será mais detalhado enquanto arquivo só recebe as resoluções de expressão retornado para o método resolver a variável onde foi adicionado a resolução.

## **ZerarVar(char letra)**

O método mais simples desse programa serve somente para receber um char e verificar se ele é variável, se o conteúdo conter uma variável ela recebera zero para poder ser reutilizada novamente.

Composta por um Switch case que recebe um achar e verifica se aquele char é igual algumas das variáveis, caso seja, essa variável recebe zero para ser utilizada novamente no código.

Ex: switch (letra) {

case 'U':

u = 0;

break;

}

## **Add(String aux), Sub(String aux), Mpy(String aux) e Div(String aux).**

São os principais métodos para a resolução do programa, são eles que informa ao usuário e gravam nos arquivos o tipo de operação que está sendo tratada e logo após informam a letra que está tratando.

Todos eles recebem a string aux e assim que o método inicia já criam a varial op2 = aux.charAt(2) pegando o último caractere do aux informando que ele esta sob aquela operação, logo após ele salva em ambos os arquivos essas informações.

Logicamente o que diferencia os métodos são as informações segundos as operações, caso seja adição mostrará: ADD+op2, caso seja subtração mostrará: SUB+op2, caso seja multiplicação mostrará: MPY+op2 e caso seja divisão mostrará: DIV+op2;

## **CarregaInicio()**

Carrega todas as informações que um programa compilado em assembly precisa ter no seu começo ou “cabeçalho” para funcionar.

Somente grava essas instruções no txt arquivo que vai ser o arquivo para a resolução da expressão.

## **CarregaFinal()**

Carrega todas as informações que um programa compilado em assembly precisa ter no seu final para funcionar.

Somente grava essas instruções no txt arquivo que vai ser o arquivo para a resolução da expressão.

## **CrarTxt()**

Esse método é um pouco complexo e foi um pouco difícil de desenvolver pois tivemos que utilizar JFileChoosser, basicamente ele faz o gerenciamento de diretórios onde vamos salvar os nossos arquivos.

Assim que inicia o método é criado um JFileChooser, logo após as ariáveis com o endereço e com o nome do arquivo digitado pelo usuário, ele criar um filtro e cria a função accept(file f) que nos retorna o diretório ou então o nome do arquivo em minúsculo com o .txt no final e a função getDescription() caso o usuário queria só verificar na janelas extenções como (\*.txt).

Criamos uma variável int para pegar o estado do botão e saber se ele foi clicado e qual botão foi clicado a partir daí ele é verificado por três if o primeiro pra saber se o clique foi no salvar, se sim ele utiliza: endereco = fc.getSelectedFile().getAbsolutePath(); par pegar o caminho do diretório já com o nome do arquivo que o usuário digitou e logo após pega o nome do arquivo: nome = fc.getSelectedFile().getName(). Os outros dois if’s são somente pra saber se o usuário clicou em cancelar ou em fechar a janela.

Feito isso ele criar o txt arquivo que recebe a resolução da expressão passando variável endereço como parâmetro: arquivo = new FileWriter(endereco), após isso utilizando o repleace ele substitui a variável nome por “log”+nome e coloca em uma nova variável chamada enderecoLog e criar o arquivo de log: enderecoLog = endereco.replace(nome, "log" + nome) e logo depois log = new FileWriter(enderecoLog) e retorna o endereço para depois ser passado para o método AbrirArquivo().

## **AbriArquivo( String endereco)**

Esse método ele serve para assim que a compilação do programa for feita aconteça a abertura do arquivo.txt, recebe comoparametro o endereço do arquivo criado.

Criar uma variável com o nome pro informando o tipo como process, logo após pro recebe o comando para abrir o arquivo via cmd e ao invés de passarmos o caminho, passamos a variável endereco pois la se encontra o caminho do último arquivo criado e sendo assim abre o mesmo para o usuário ter a visualização da resolução da expressão.

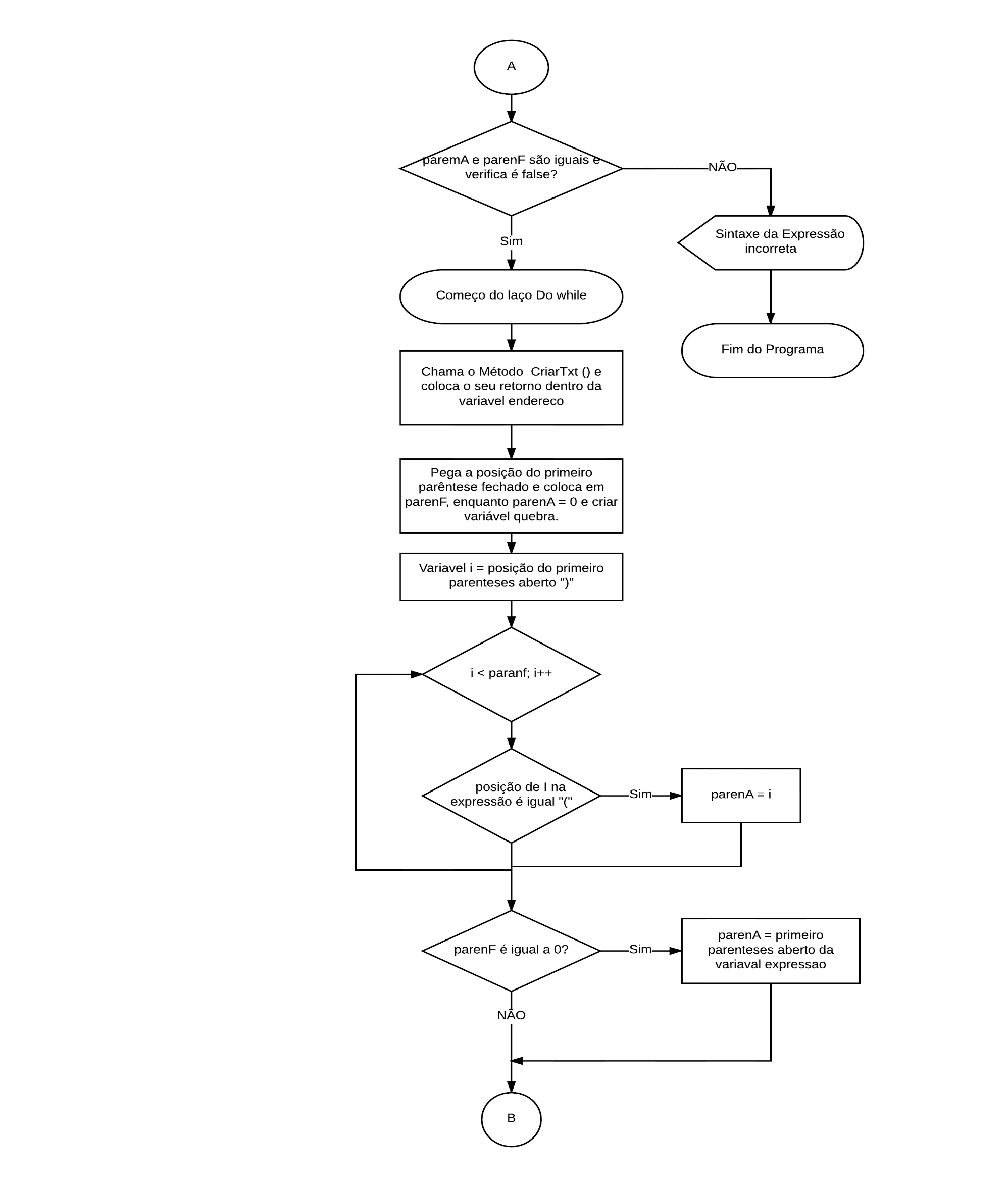
## **FecharArquivo()**

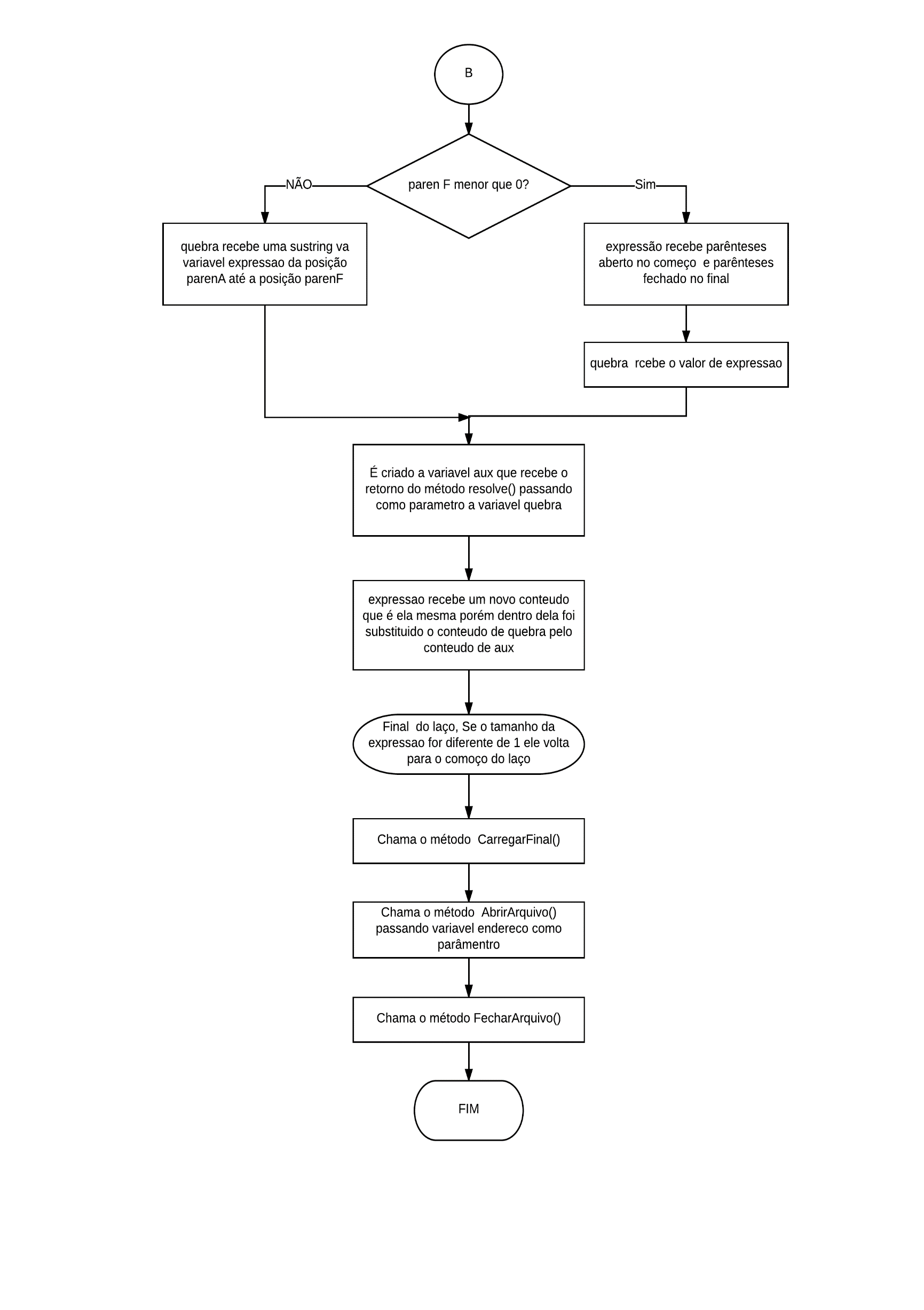
Simplesmente essa classe essa serve para fechar ambos os arquivos.

A classe possui um try catch para casso aconteça algum erro e logo após fechamos esses arquivos com .close() em ambos os txt que estavam sendo manipulados.

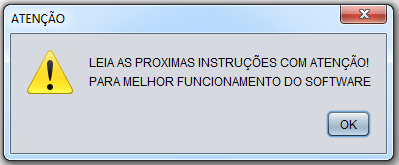
# Fluxograma

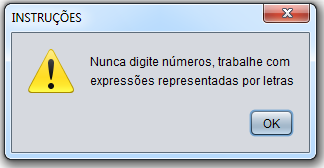
# 

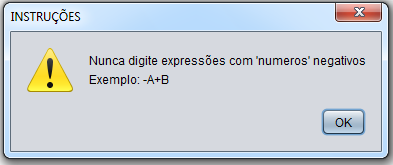


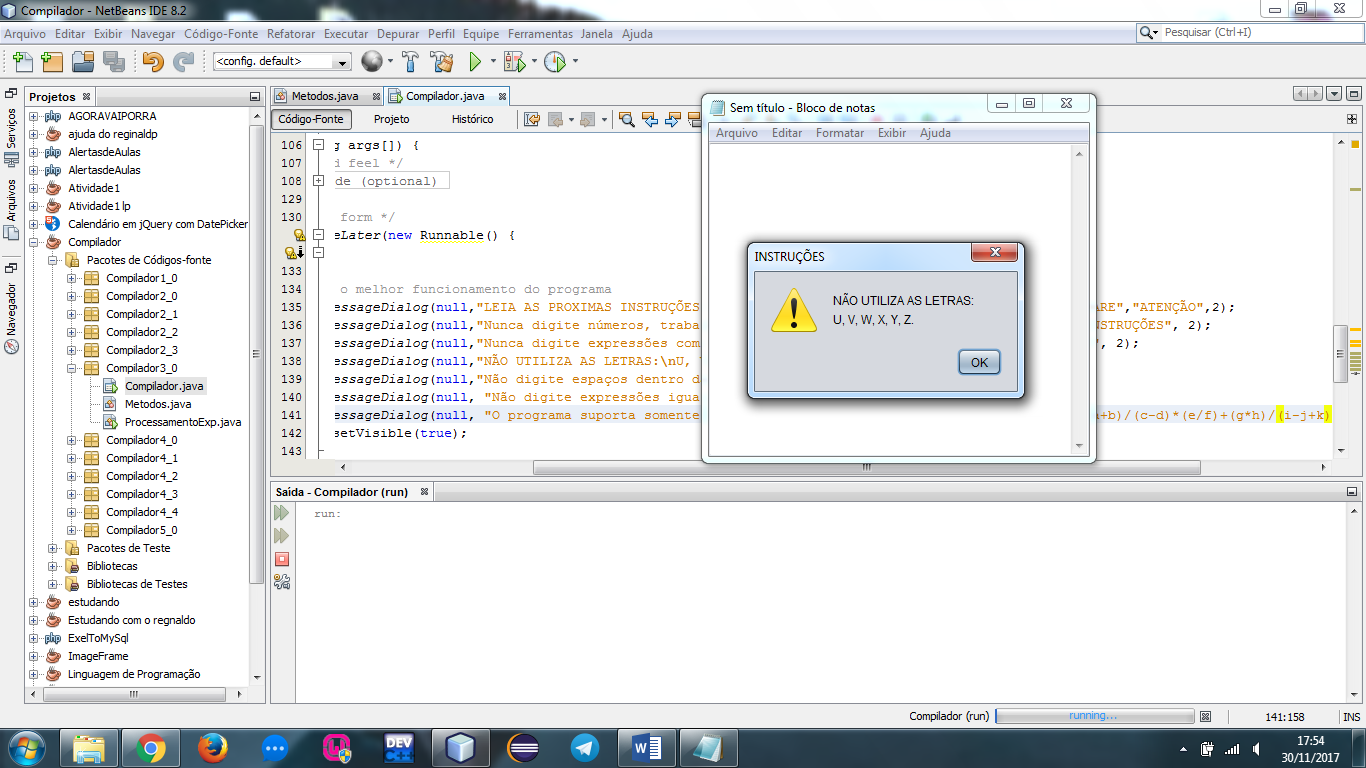


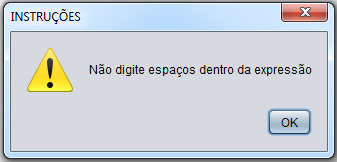
# Alertas e Instruções Para o Uso

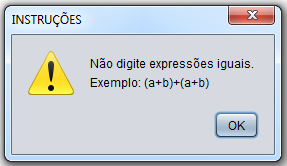


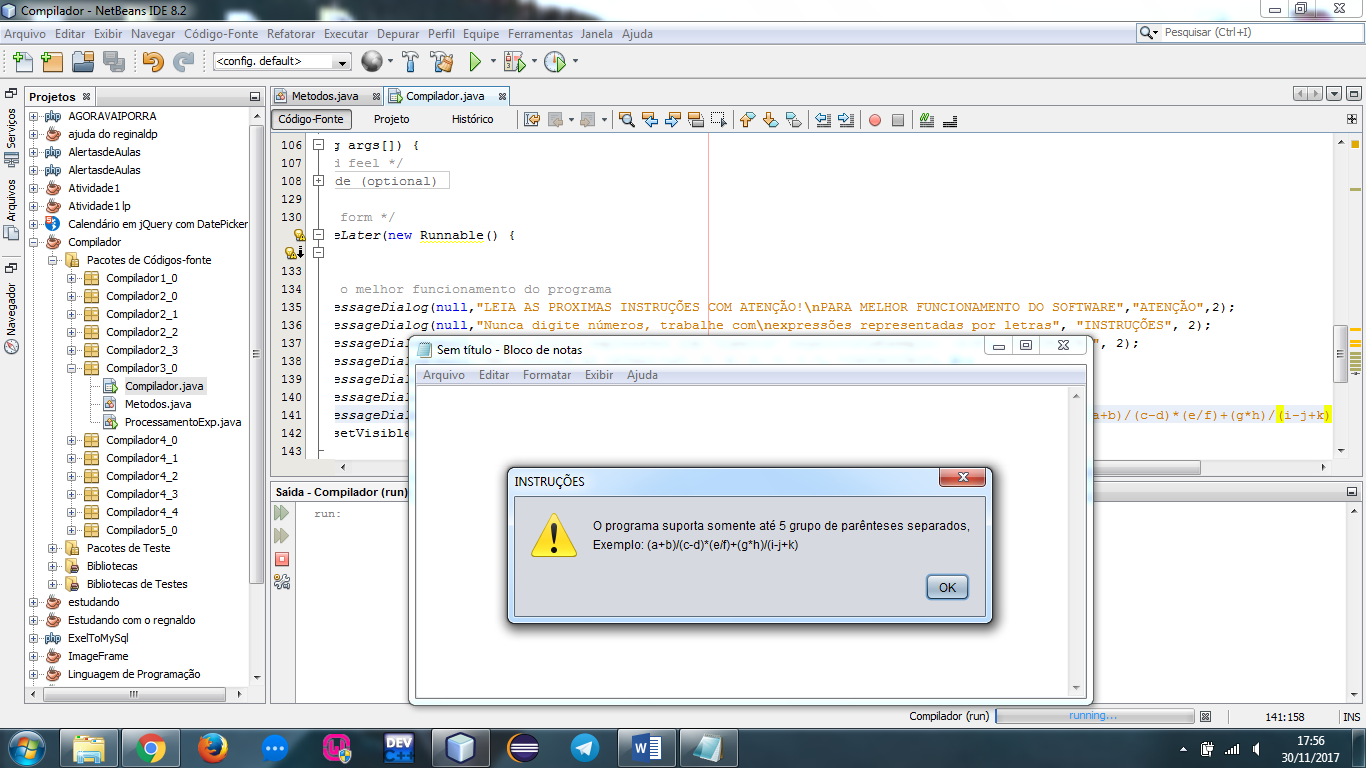






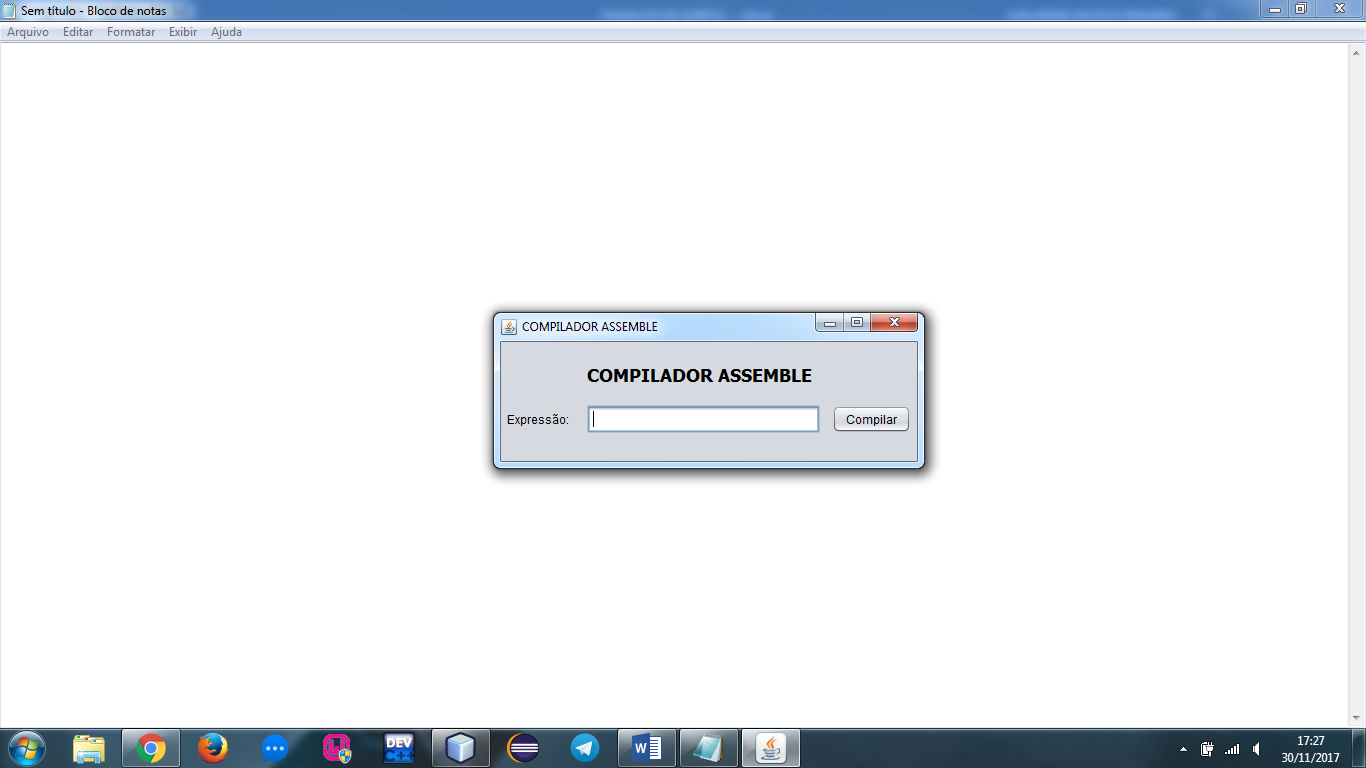




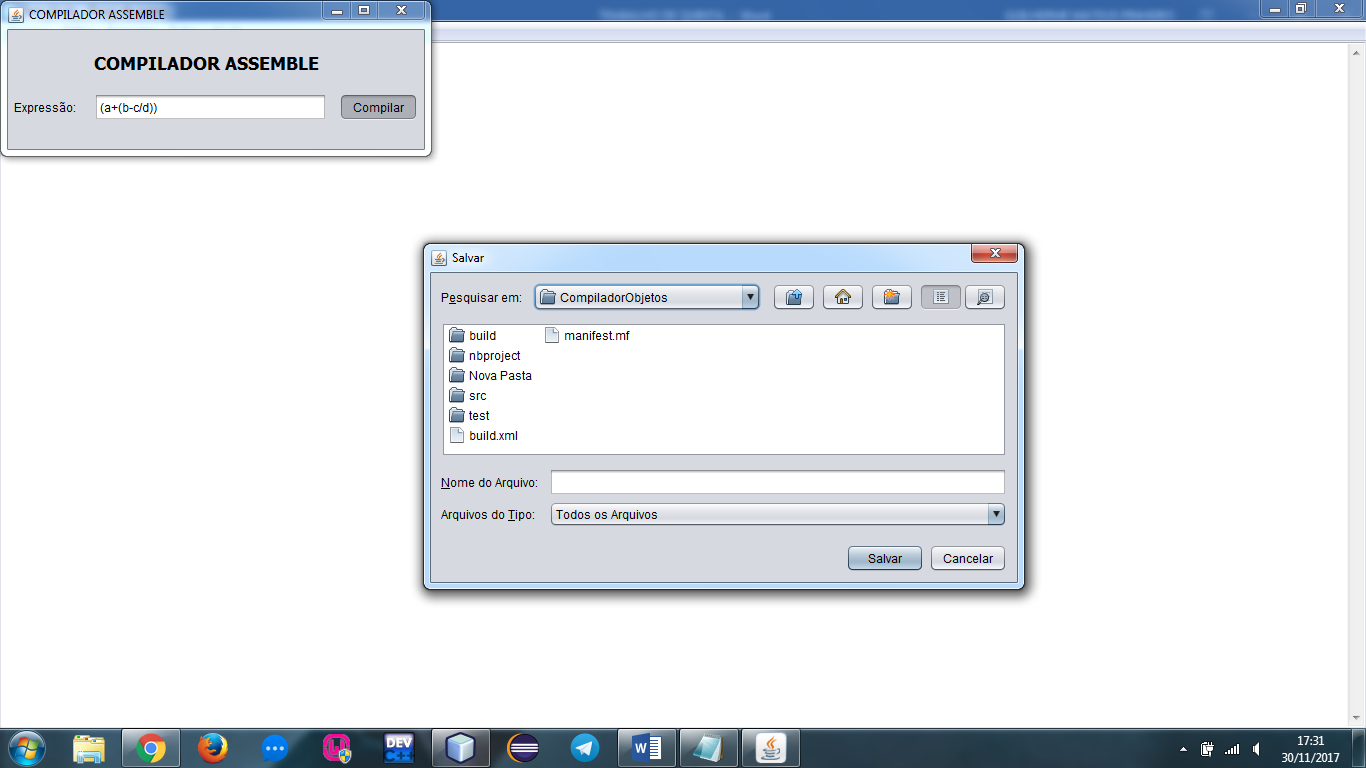


# Teste do Programa

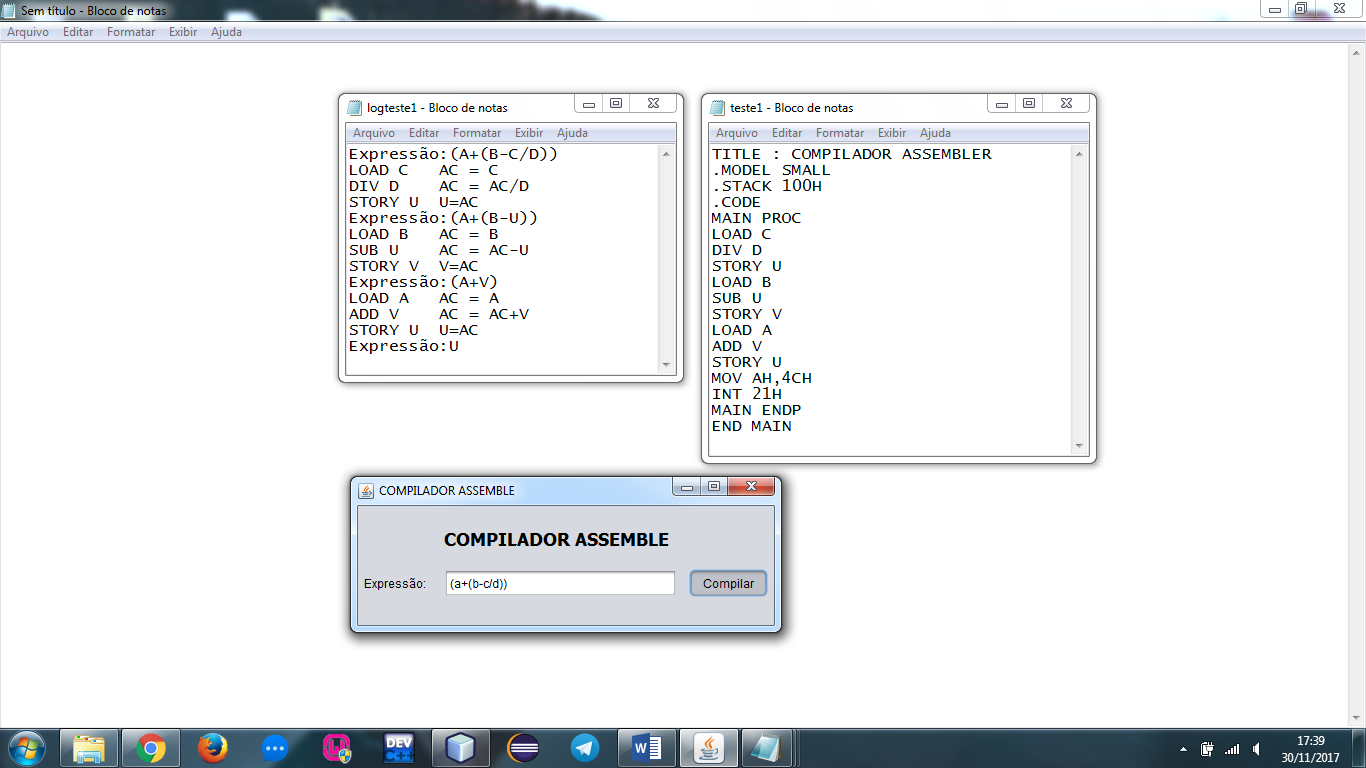
O programa sempre se inicia com essa interface:



Depois de digitar a expressão o programa abre uma janela para o usuário escolher o diretório onde deseja salvar o arquivo:



Após a escolha do diretório o sistema abrira somente o arquivo com a resolução da expressão, pois o arquivo de log nem sempre será necessário, caso o usuário queira abrir é só ir até o local que o arquivo estará lá.



# Código

## **Classe Métodos.**

Todos os métodos desta classe foram desenvolvidos por autoria própria e todo o conteúdo possui comentários para ao auxílio na logica deste programa para caso outra pessoa precise utiliza-lo para quaisquer aplicação.

package Compilador5\_0;

import java.io.File;

import java.io.FileWriter;

import java.io.IOException;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.swing.JFileChooser;

import javax.swing.JOptionPane;

import javax.swing.filechooser.FileFilter;

/\*

ATENÇÃO TODAS AS CLASSES QUE POSSUEM: throws IOException TEM ESCRITA DE INFORMAÇÕES EM ARQUIVOS, POR ISSO O

THROWS PARA FAZER O TRATAMENTO DE ERROS NA PROPRIA CLASSE AO INVES DE USAR VARIOS TRY CATCHS

EXISTEM DOIS ARQUIVOS:

\*LOG: que cria tecnicamente um arquivo de log, com todas as informações de como foi tratada as expressoes

mostrando os valores no acumulador AC e nas suas variaveis.

\*ARQUIVO: cria um arquivo totalmente referente ao programa que fi traduzido para assembler desde suas primeiras

instruções, resolução da equação e a finalização do programa.

\*/

public class Metodos extends Compilador{

private FileWriter log, arquivo;

private int u = 0, v = 0, w = 0, x = 0, y = 0, z = 0;

private Compilador comp = new Compilador();

public void Processamento(String expressao) throws IOException{

/\* TRATAMENTO DO CONTEUDO RECEBIDO DENTRO DA VARIAVEL STRING \*/

expressao = expressao.trim(); //remove espaços em branco do inico e do final da String

expressao = expressao.toUpperCase(); //Deixar a expressao em letras maiusculas

int parenF = 0, parenA = 0; //cria duas variaveis para a verificação dos parenteses

boolean verifica = false; //essa variavel serve para saber se tem numeros na expressao

for (int i = 0; i < expressao.length(); i++) { // percorre a expresso

if (Character.isDigit(expressao.charAt(i)) == true) // valida se tem numeros

{

verifica = true;

}

if (expressao.charAt(i) == '(') //verifica quantos parenteses tem abertos

{

parenA++;

} else if (expressao.charAt(i) == ')') {//verifica quantos parenteses tem fecados

parenF++;

}

}

if (parenA != parenF || verifica == true) {

JOptionPane.showMessageDialog(null, "Sintaxe da expressão incorreta\nTente novamente", "Erro", 2);

System.exit(0);

} else {

String endereco = Criartxt();

log.write("Expressão:" + expressao + "\r\n");

CarregaInicio();

do {

parenA = 0; //zera parenteses aberto

parenF = expressao.indexOf(")"); //pega o primeiro parenteses fechado

String quebra = ""; //quebra igual vaziorr

for (int i = expressao.indexOf("("); i < parenF; i++) { //verifica se dentro dessa expressão ainda tem outro parenteses aberto

if (expressao.charAt(i) == '(') { // a variavel i = 0 primeiro parenteses aberto.

parenA = i;

}

}

if (parenF == 0) { // se a expressao não tiver parenteses dentro de parentes

parenA = expressao.indexOf("("); // então ele informa que o primeiro parenteses aberto é o certo

}

if (parenF < 0) { // se parenA for igual a 0 então a expressão não esta entre parenteses

/\* adiciona '(' no começo e o ')' no final da String NA VARIAVEL QUEBRA porque quando acontece a quebra

ele procura juntamente com o grupo de parenteses, quando essa quebra ele entra na função resolve ele tira esses parentes,

se não tive essa inclussão nas duas variavei o programa entra em loop infinito\*/

//quebra = "(" + expressao + ")";

expressao = "(" + expressao + ")";

quebra = expressao;

} else {

quebra = expressao.substring(parenA, parenF + 1);//quebra a parte do parenteses da string

}

String aux = Resolve(quebra); //passa a expressão

expressao = expressao.replace(quebra, aux); //retira da expressao o que foi quebrado e substitui pelo retorno resolvida

log.write("Expressão:" + expressao + "\r\n");

} while (expressao.length() != 1);

CarregarFinal();

AbrirArquivo(endereco);

FecharAquivo();

}

}

private String Resolve(String aux) throws IOException {

boolean divMult = false;

//RETIRA O PARENTESESDO INICIO

aux = aux.substring(1, aux.length() - 1); //retira o parentses do começo e final

String quebras = "";

String novaExp = aux;

for (int i = 1; i < aux.length(); i++) {

if (aux.charAt(i) == '\*' || aux.charAt(i) == '/') {

String op1 = "" + aux.charAt(i - 1), op2 = aux.charAt(i + 1) + ""; //pega as letras antes e depois do sinal

quebras = op1 + aux.charAt(i) + op2; //concatena elas juntamente com o sinal e adciona em quebras

quebras = quebras.trim(); //retira os espaços do começo e final da expressão

Load(aux.charAt(i - 1));//chama o metodo loade que move o primeiro para a memória

divMult = true; //coloca true na varial divMulti informando que existe divisão sim

i = aux.length(); //coloca o tamanho de aux no i para sair do laço na primeira vez que acha um desses sinais

}

}

if (quebras.equalsIgnoreCase("")) {

Load(aux.charAt(0));//chama o metodo loade que move o primeiro para a memória

}

//System.out.println("Tamanho:" + aux.length());

if (aux.length() != 1) { // acontece casos que é passa para o metodo string um conteudo por exemplo assim (U) nesse caso quando é retirado os parenteses o conteudo fica so U, esse if serve para retornar esse conteudo Já que não é necessário efetuar nenhuma resolução.

if (aux.length() > 3 && divMult != true) {

quebras = aux.substring(0, 3);

} else if (aux.length() <= 3 && divMult != true) {

quebras = aux;

//System.out.println("Quebra aux<=3 sem \*/:" + quebras);

}

switch (quebras.charAt(1)) {

case '+': //caso seja adição entra aqui

Add(quebras);

break;

case '-'://caso seja subtração entra aqui

Sub(quebras);

break;

case '\*'://caso seja multilicação entra aqui

Mpy(quebras);

break;

case '/'://caso seja divisão entra aqui

Div(quebras);

break;

}

}

aux = Story(quebras); //chama o metodo Story que retorna a letra da varial que sera armazenado a resolução

novaExp = novaExp.replace(quebras, aux); // faz a retirada do que tem na variavel quebras e coloca o retorno que está em aux

if (novaExp.length() != 1) { //se expressão for diferente de 1 significa ela sera a proxima expressão que tem que ser resolvida

novaExp = "(" + novaExp + ")"; //para que isso acontece nos colocamos um grupo de parenteses entre ela e retorna ela com os parentes para ser proxima a enterar nesse metodo

}

return novaExp;

}

private void Load(char op1) throws IOException {

log.write("LOAD " + op1 + "\r\t AC = " + op1 + "\r\n");

arquivo.write("LOAD " + op1 + "\r\n");

}

private String Story(String aux) throws IOException {

//variaveis para colocar o que tem dentro da memoria

String zerarVar = aux;

if (u == 0) {

aux = "U";

u = 1;

} else if (v == 0) {

aux = "V";

v = 1;

} else if (w == 0) {

aux = "W";

w = 1;

} else if (x == 0) {

aux = "X";

x = 1;

} else if (y == 0) {

aux = "Y";

y = 1;

} else if (z == 0) {

aux = "Z";

z = 1;

}

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

ZerarVar(zerarVar.charAt(i));

}

log.write("STORY " + aux + " " + aux + "=AC" + "\r\n");

arquivo.write("STORY " + aux + "\r\n");

return aux;

}

private void ZerarVar(char letra) {

switch (letra) {

case 'U':

u = 0;

break;

case 'V':

v = 0;

break;

case 'W':

w = 0;

break;

case 'X':

x = 0;

break;

case 'Y':

y = 0;

break;

case 'Z':

z = 0;

break;

}

}

private void Add(String aux) throws IOException {

char op2 = aux.charAt(2);

log.write("ADD " + op2 + "\r\t AC = AC+" + op2 + "\r\n");

arquivo.write("ADD " + op2 + "\r\n");

}

private void Sub(String aux) throws IOException {

char op2 = aux.charAt(2);

log.write("SUB " + op2 + "\r\t AC = AC-" + op2 + "\r\n");

arquivo.write("SUB " + op2 + "\r\n");

}

private void Mpy(String aux) throws IOException {

char op2 = aux.charAt(2);

log.write("MPY " + op2 + "\r\t AC = AC\*" + op2 + "\r\n");

arquivo.write("MPY " + op2 + "\r\n");

}

private void Div(String aux) throws IOException {

char op2 = aux.charAt(2);

log.write("DIV " + op2 + "\r\t AC = AC/" + op2 + "\r\n");

arquivo.write("DIV " + op2 + "\r\n");

}

private void CarregaInicio() throws IOException {

arquivo.write("TITLE : COMPILADOR ASSEMBLER\r\n");

arquivo.write(".MODEL SMALL\r\n");

arquivo.write(".STACK 100H\r\n");

arquivo.write(".CODE\r\n");

arquivo.write("MAIN PROC\r\n");

}

private void CarregarFinal() throws IOException {

arquivo.write("MOV AH,4CH\r\n");

arquivo.write("INT 21H\r\n");

arquivo.write("MAIN ENDP\r\n");

arquivo.write("END MAIN\r\n");

}

private String Criartxt() throws IOException {

JFileChooser fc = new JFileChooser(System.getProperty("user.dir")); //criar um JFileChooser para escolha de arquivo e informa ao sistema que vai trabalhar com diretorio

String endereco = "", nome = ""; // cria as variavel nome e endereco

FileFilter filter = new javax.swing.filechooser.FileFilter() { //cria um filtro

public boolean accept(File f) {

return f.isDirectory() || f.getName().toLowerCase().endsWith(".txt"); // retorna o diretorio ou o nome em minusculo com o .txt no final

}

public String getDescription() {

return "(\*.txt)"; //caso o usario queira filtrar pelo txt

}

};

fc.addChoosableFileFilter(filter);

int returnVal = fc.showDialog(null, "Salvar"); //pega o valor do botão pra saber qual botão foi clicado

if (returnVal == JFileChooser.APPROVE\_OPTION) {

//como ele clicou em salvar acontecera isso:

endereco = fc.getSelectedFile().getAbsolutePath(); // pega o endereço do arquivo que vai ser contido o resultado da expressao

nome = fc.getSelectedFile().getName(); //pega o nome do arquivo para dps colocar a palavra log na frente e criar o arquivo de log naquela expressao

} else if (returnVal == JFileChooser.CANCEL\_OPTION) {

//caso o usuario clique em cancelar acontece isso

} else if (returnVal == JFileChooser.UNDEFINED\_CONDITION) {

}

arquivo = new FileWriter(endereco); //cria um aquivo txt

String enderecoLog = endereco.replace(nome, "log" + nome); // o nome do arquivo rece a palavra log na frente

log = new FileWriter(enderecoLog); //cria um aquivo txt

return endereco;

}

private void FecharAquivo() {

try {

log.close(); //fecha o aquivo txt

arquivo.close();

} catch (Exception e) {

System.out.println("Erro: " + e);

}

}

private void AbrirArquivo(String Endereco) { //essa classe erve para abrir o arquivo assim que comiplar

Process pro; // cria uma variavel pro informando que ela se refe a processamentos

try {

pro = Runtime.getRuntime().exec("cmd.exe /c "+Endereco); //informa que o programa devera abrir o arquivo da variavel endereço

} catch (IOException ex) {

Logger.getLogger(Metodos.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

## **Classe Compilador.**

A classe compilador a maioria de suas linhas de comando foram feitas pela própria IDE na criação de um arquivo JFrame, a parte que codamos foi somente o método private void BtnCompilarActionPerformed(java.awt.event.ActionEvent evt) que seria o evento de clique do nosso botão.

Também incluímos alguns alertas com o JOptionPane na classe public void run() dentro da classe main.

/\*

\* To change this license header, choose License Headers in Project Properties.

\* To change this template file, choose Tools | Templates

\* and open the template in the editor.

\*/

package Compilador5\_0;

import java.io.IOException;

import java.util.logging.Level;

import java.util.logging.Logger;

import javax.swing.JOptionPane;

/\*\*

\*

\* @author Guilherme

\*/

public class Compilador extends javax.swing.JFrame {

/\*\*

\* Creates new form Compilador

\*/

public Compilador() {

initComponents();

}

/\*\*

\* This method is called from within the constructor to initialize the form.

\* WARNING: Do NOT modify this code. The content of this method is always

\* regenerated by the Form Editor.

\*/

@SuppressWarnings("unchecked")

// <editor-fold defaultstate="collapsed" desc="Generated Code">

private void initComponents() {

txtExpressao = new javax.swing.JTextField();

jToggleButton1 = new javax.swing.JToggleButton();

jLabel1 = new javax.swing.JLabel();

jLabel2 = new javax.swing.JLabel();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

setTitle("COMPILADOR ASSEMBLE");

jToggleButton1.setText("Compilar");

jToggleButton1.addActionListener(new java.awt.event.ActionListener() {

public void actionPerformed(java.awt.event.ActionEvent evt) {

jToggleButton1ActionPerformed(evt);

}

});

jLabel1.setText("Expressão: ");

jLabel2.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N

jLabel2.setText("COMPILADOR ASSEMBLE");

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());

getContentPane().setLayout(layout);

layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addContainerGap()

.addComponent(jLabel1)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(layout.createSequentialGroup()

.addComponent(txtExpressao, javax.swing.GroupLayout.PREFERRED\_SIZE, 233, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addComponent(jToggleButton1))

.addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED\_SIZE, 243, javax.swing.GroupLayout.PREFERRED\_SIZE))

.addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))

);

layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

.addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()

.addGap(21, 21, 21)

.addComponent(jLabel2, javax.swing.GroupLayout.DEFAULT\_SIZE, 24, Short.MAX\_VALUE)

.addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

.addComponent(txtExpressao, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

.addComponent(jToggleButton1)

.addComponent(jLabel1))

.addGap(28, 28, 28))

);

pack();

}// </editor-fold>

private void jToggleButton1ActionPerformed(java.awt.event.ActionEvent evt) {

// TODO add your handling code here:

Metodos met = new Metodos();

String expressao = txtExpressao.getText();

try {

met.Processamento(expressao);

} catch (IOException ex) {

Logger.getLogger(Compilador.class.getName()).log(Level.SEVERE, null, ex);

}

}

/\*\*

\* @param args the command line arguments

\*/

public static void main(String args[]) {

/\* Set the Nimbus look and feel \*/

//<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">

/\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.

\* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html

\*/

try {

for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {

if ("Nimbus".equals(info.getName())) {

javax.swing.UIManager.setLookAndFeel(info.getClassName());

break;

}

}

} catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(Compilador.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(Compilador.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(Compilador.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

} catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(Compilador.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);

}

//</editor-fold>

/\* Create and display the form \*/

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

//instruções para o melhor funcionamento do programa

JOptionPane.showMessageDialog(null,"LEIA AS PROXIMAS INSTRUÇÕES COM ATENÇÃO!\nPARA MELHOR FUNCIONAMENTO DO SOFTWARE","ATENÇÃO",2);

JOptionPane.showMessageDialog(null,"Nunca digite números, trabalhe com\nexpressões representadas por letras", "INSTRUÇÕES", 2);

JOptionPane.showMessageDialog(null,"Nunca digite expressões com 'numeros' negativos\nExemplo: -A+B", "INSTRUÇÕES", 2);

JOptionPane.showMessageDialog(null,"NÃO UTILIZA AS LETRAS:\nU, V, W, X, Y, Z.", "INSTRUÇÕES", 2);

JOptionPane.showMessageDialog(null,"Não digite espaços dentro da expressão", "INSTRUÇÕES", 2);

JOptionPane.showMessageDialog(null, "Não digite expressões iguais.\nExemplo: (a+b)+(a+b)", "INSTRUÇÕES", 2);

JOptionPane.showMessageDialog(null, "O programa suporta somente até 5 grupo de parênteses separados,\nExemplo: (a+b)/(c-d)\*(e/f)+(g\*h)/(i-j+k)", "INSTRUÇÕES", 2);

new Compilador().setVisible(true);

}

});

}

// Variables declaration - do not modify

private javax.swing.JLabel jLabel1;

private javax.swing.JLabel jLabel2;

private javax.swing.JToggleButton jToggleButton1;

private javax.swing.JTextField txtExpressao;

// End of variables declaration

}