Editado em 25/04/2024: A tabela que consta na etapa C foi corrigida. Ver destaque em vermelho no texto.

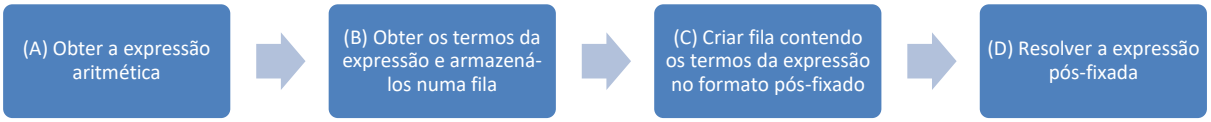
Trabalho 1

O objetivo deste trabalho é construir um programa capaz de resolver expressões aritméticas simples. O programa deve permitir que o usuário informe uma expressão aritmética e a partir desta, calcular e apresentar o resultado da expressão.

Uma expressão aritmética é uma combinação de operadores e operandos. Os operadores definem a operação aritmética. O programa deve reconhecer os operadores `+`, `-`, `*` e `/`. Os operandos são os valores que serão computados na operação. Por exemplo, na expressão: `15 / 3`, o operador é o `/` e os operandos são `15` e `3`.

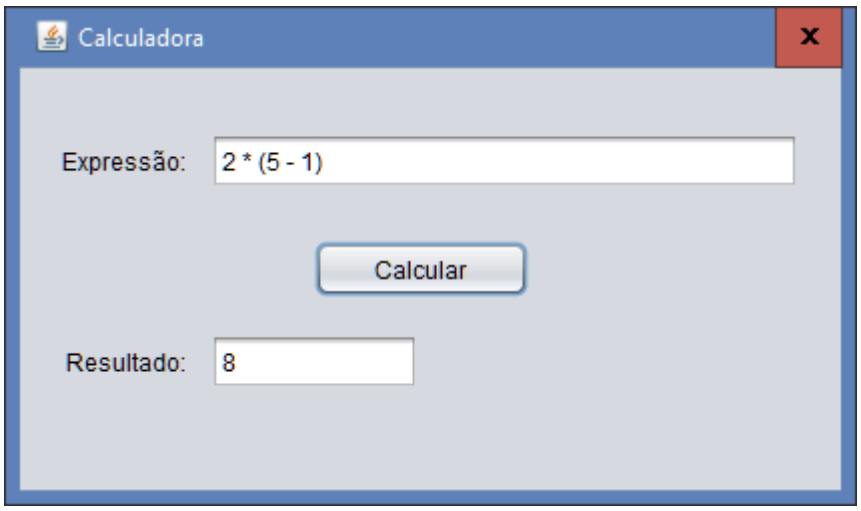
Para resolver a expressão aritmética, deve-se considerar a precedência natural entre os operadores. O programa deve aceitar expressões com parênteses para indicar uma ordem diferente de precedência, inclusive o uso de parênteses aninhados (parênteses dentro de parênteses).

O procedimento computacional para resolver este problema está dividido nas seguintes etapas:



ETAPA (A)

Esta etapa consiste em possibilitar ao usuário informar uma expressão aritmética. O programa deve ter interface gráfica semelhante à vista abaixo:



O campo "Expressão" deve possibilitar que o usuário digite uma expressão aritmética. Quando o botão "Calcular" for pressionado, o cálculo deverá ser realizado (através das etapas B, C e D, descritas adiante) e o resultado deverá ser apresentado no campo "Resultado".

ETAPA (B)

O objetivo desta etapa é criar uma fila com os termos da expressão aritmética, onde cada termo corresponde a um operador, operando ou parêntese. Suponha que tenha sido submetida a expressão `(25 + 10) / 3,5`. Partindo desta expressão, a etapa (B) consiste em criar uma fila com todos os termos da expressão, na ordem em que eles aparecem. Para este exemplo, a fila conteria 7 elementos, como visto abaixo:

(	25	+	10	)	/	3,5

↑
início da fila

ETAPA (C)

Uma das formas de resolver o problema proposto neste trabalho é representar a expressão aritmética no formato pós-fixado (também conhecida como "notação polonesa reversa"). Nesta notação, ao invés do operador ser escrito entre os operandos (que é o formato "infixado"), o operador é colocado após os operandos. Exemplo: a expressão `147 + 12` está escrita no formato infixada. No formato pós-fixada, a expressão equivalente é `147 12 +`. Resolver uma expressão aritmética pós-fixada é mais conveniente do que resolver a expressão infixada. Sendo assim, esta etapa consiste em criar outra fila de termos que representam a expressão no formato pós-fixado. Esta etapa deverá ser executada a partir da fila de termos em formato infixado (obtido na etapa B).

Exemplo, se for submetida para este processo a fila abaixo, que contém a expressão infixada:

(	15	+	5	)	/	3

O resultado da etapa (C) seria criar uma fila cujo conteúdo seria:

15	5	+	3	/

↑
início da fila

No formato pós-fixado não existem parênteses, mas eles são utilizados para determinar precedência. Veja outros exemplos de expressões no formato infixado e seu correspondente no formato pós-fixado:

Formato infixado	Formato pós-fixado
`15 + 5 / 3`	`15 5 3 / +`
`(20 + 5) / (8 - 2) * 3`	`20 5 + 8 2 - / 3 *`
`2 * (1 + (10 / 2))`	`2 1 10 2 / + *`

O algoritmo que produz a expressão aritmética no formato pós-fixada precisará trabalhar com três estruturas de dados: a fila com termos no formato infixado (aqui denominada de fila "A" – é a fila originada da etapa anterior), uma pilha que será usada para armazenar temporariamente os operadores e parênteses (pilha "B") e uma fila que conterá os termos no formato pós-fixado (fila "C"). A forma como estas estruturas serão utilizadas consta a seguir.

O algoritmo é um procedimento repetitivo, que deve retirar um termo da fila "A" e para cada termo retirado, executar uma das regras da tabela abaixo em que o termo se enquadrar.

Termo retirado da fila "A"	Ação a ser tomada
Operando	Insira o operando na fila "C".
Parêntese de abertura	Empilhe o parêntese de abertura na pilha "B"
Parêntese de fechamento	Desempilhe os elementos da pilha "B" até remover o "parêntese de abertura" correspondente. Adicione cada operador desempilhado à fila "C".
Operador	Empilhe-o na pilha "B". Porém, antes disso, remova os operadores que estejam na pilha "B" e que tenham procedência igual ou maior ao operador que foi lido da fila "A" ~~fila "C"~~. Ao encontrar um parêntese de abertura ou um operador com precedência menor, interrompa o desempilhamento. Adicione estes operadores removidos da pilha à fila "C".

Depois de processado todos os termos da fila "A", pode ser que a pilha "B" ainda contenha operadores. Caso acontecer, desempilhe todos os operadores e adicione-os na fila "C".

ETAPA (D)

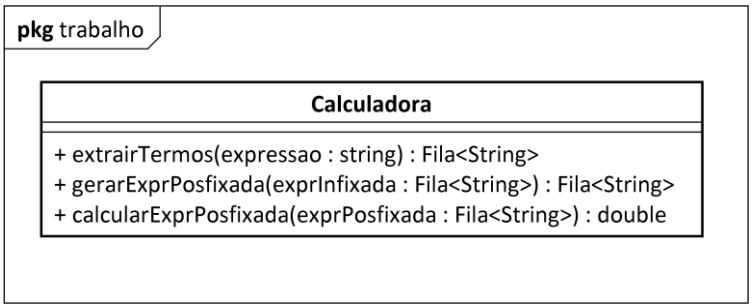
Esta etapa deverá resolver a expressão aritmética, partindo da fila que foi criada na etapa anterior.

Para resolver esta etapa, será necessário utilizar outra pilha auxiliar. O procedimento deve retirar os dados da fila que foi produzida pela etapa (C) e a cada termo retirado, aplicar uma das seguintes regras:

Termo retirado da fila	Ação a ser tomada
Operando	Empilhe-o na pilha auxiliar
Operador	Desempilhe dois operandos da pilha e aplique o operador lido da fila com os operandos que foram desempilhados. Empilhe o resultado na pilha auxiliar.

Depois de ter processado todos os termos da lista, a pilha conterá apenas um dado. Este dado é o resultado do cálculo da expressão.

A solução que for adotada deve criar a classe chamada **`Calculadora`** no pacote **`trabalho`** e deverá conter os seguintes métodos:



Sendo que:

- O método **`extrarTermos()`** deve implementar o algoritmo descrito na Etapa (B). Isto é, partindo de uma **`string`** cujo conteúdo é uma expressão aritmética no formato infixada, deverá retornar uma fila de **`strings`**, onde cada elemento corresponde a um termo da expressão;
- O método **`gerarExprPosFixada()`** deve implementar o algoritmo descrito na Etapa (C), isto é, a partir de uma fila de **`strings`** contendo os termos de uma expressão infixada, deverá retornar uma nova fila com os termos da mesma expressão, porém no formato pós-fixada;
- O método **`calcularExprPosFixada()`** deve implementar o algoritmo descrito pelo Etapa (D), isto é, a partir de uma fila de **`strings`** contendo os termos de uma expressão pós-fixada, deverá calcular o valor da expressão.

Outros métodos privados poderão ser adicionados nesta classe. Outras classes poderão ser criadas para resolver o problema.

Exemplo (parcial) de como poderá ser implementado o código que tratará o evento do botão "Calcular" da tela:

```
Calculadora c = new Calculadora();
Fila<String> termosInfixada = c.extrairTermos( tfExpressao.getText() );
Fila<String> termosPosfixada = c.gerarExprPosfixada(termosInfixada);
double resultado = c.calcularExprPosfixada(termosPosfixada);
```

Obs: `tfExpressao` é um componente `JTextField`

Isto é, o código acionará as etapas (B), (C) e (D).

O trabalho pode ser feito em equipe de até três pessoas. O trabalho deve ser submetido no AVA até o dia **10/05/2024**.

O programa deve ser armazenado na pasta "Trabalho 1" do AVA. Crie um arquivo readme e informe os membros da equipe. O arquivo deve ser enviado somente por um dos membros da equipe.

Observações diversas:

- Todas as operações de divisão devem ser interpretadas como divisão de decimais. Exemplo: 5 / 2 deve resultar em "2,5" e não apenas "2";
- O programa deve possuir uma única classe com o método `main()`;
- Não podem ser usadas as estruturas de coleções da API Java (`ArrayList`, `List`, `HashMap`, etc). Utilize as implementações de estruturas de dados realizadas nesta disciplina. Não haverá necessidade de alterar as implementações das estruturas de dados que serão utilizadas;
- Não devem ser utilizados componentes de terceiros, isto é, componentes que não sejam nativos da biblioteca Java;
- Considere que a expressão poderá ou não conter caracteres em branco para separar os elementos. Exemplo: poderá ser submetido "`15/3`", "`15 / 3`", "`15 /3`", " `15 /3`", etc;
- O programa deve reconhecer que a expressão pode conter números decimais. O separador decimal que deve ser considerado é a vírgula;
- O programa deve reconhecer números negativos. Os números negativos serão precedidos pelo símbolo – (sem espaço entre o número e o símbolo).

Informações adicionais sobre a notação pós-fixada podem ser encontradas em:

https://pt.wikipedia.org/wiki/Notação_polonesa_inversa

https://panda.ime.usp.br/panda/static/pythonds_pt/02-EDBasicos/InfixPrefixandPostfixExpressions.html

Pode ser útil utilizar o conversor on-line de expressão infixada para pós-fixada, para validar seu algoritmo (Etapa C):

http://scanftree.com/Data_Structure/prefix-postfix-infix-online-converter

Bibliografia

☐ LAFORE, Robert. **Estruturas de Dados & Algoritmos em Java**. Rio de Janeiro: Editora Ciência Moderna. 2004.