INSTITUTO FEDERAL
Rio Grande do Sul | Campus Farroupilha

ATIVIDADE PRÁTICA 4 - APLICAÇÃO DAS ESTRUTURAS DE DADOS PILHA E FILA

CURSO: TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
DISCIPLINA: ESTRUTURAS DE DADOS
PERÍODO LETIVO: 2024-01
PROFESSOR: FELIPE MARTIN SAMPAIO

INSTRUÇÕES:

- A atividade pode ser desenvolvida **INDIVIDUALMENTE ou EM DUPLAS**;
- Esta atividade será contabilizada como parte do instrumentos "Atividades práticas" da etapa N1;
- Devem ser enviados na tarefa disponível no Moodle:
  - Códigos .java com a implementação das soluções;
  - Pode ser enviado o projeto inteiro do IntelliJ compactado como .zip.
- Prazo para entrega: **22 de abril de 2024**.

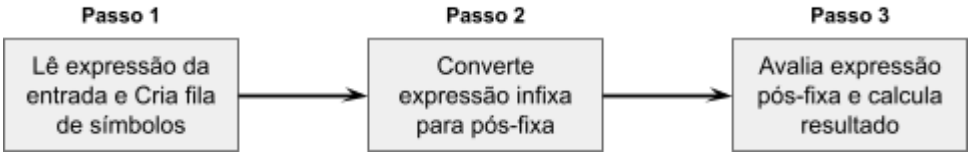
OBJETIVOS:

- Desenvolver as atividades práticas relacionadas à programação, utilizando a linguagem Java, a partir da resolução de um problema prático.
- Aplicar a lógica de funcionamento das estruturas de dados do tipo Pilha e Fila para a resolução de um problema real.

ESPECIFICAÇÃO DA APLICAÇÃO:

- Desenvolver um programa, na linguagem de programação Java, que **calcule o resultado de expressões matemáticas**. O usuário deverá informar, como entrada, uma expressão matemática na notação infixa (formato tradicional). Como resultado, o programa deverá processar a expressão e devolver ao usuário o resultado final.

- O programa deverá ser desenvolvido em três etapas, conforme o fluxograma abaixo:



- Considere os seguintes casos de teste para compreensão e teste das implementações:

Entrada (infixa)	Após conversão do Passo 2 (Pós-fixa)	Saída
4 * 3 + 5	4 3 * 5 +	17
4 * (3 + 5)	4 3 5 + *	32
(20 - 8) / (5 + 7)		1
((40 - 9 * 7) + 5) - (36 / 9 + 8)	40 9 7 * - 5 + 36 9 / 8 + -	-30
(10 * 2) - ((10 - 5 + 1) / 2)	10 2 * 10 5 - 1 + 2 / -	17

- Para a implementação das estruturas de dados, poderão ser utilizadas as seguintes implementações da biblioteca Java Collections:
  - Classe Stack<E> para a estrutura de dados Pilha:
    - Exemplo: `Stack<Integer> pilha = new Stack();`
  - Interface Queue<E> e classe LinkedList<E> para a estrutura de dados Fila:
    - Exemplo: `Queue<Integer> fila = new LinkedList();`

Passo 1:

- Ler a expressão matemática (forma infixa) do teclado e construir uma fila contendo os símbolos na ordem correta.

- De forma a facilitar a manipulação da expressão matemática de entrada, considere que a expressão será digitada em uma linha e que os símbolos serão sempre separados pelo caractere de espaço. O trecho de código a seguir exemplifica uma maneira de dividir uma String em múltiplas Strings menores, utilizando como delimitador o caractere espaço.

```
Scanner scan = new Scanner(System.in);
String exp = scan.nextLine();
String[] simbolos = exp.split(" ");

for(String simb : simbolos) {
     (...)
}
```

Passo 2:

- Converter a expressão de entrada na forma infixa para uma expressão na forma pós-fixa. A entrada será a fila criada com os símbolos no Passo 1. Como saída, será gerado um novo objeto do tipo fila, agora com os símbolos na ordem pós-fixa.

- O algoritmo a seguir descreve a sequência de passos para esta conversão:

Algoritmo: Conversão de notações (infixa para pós-fixa)

```
1.  pilhaConv ← Inicia uma pilha vazia
2.  Enquanto a filaInfixa não é vazia:
3.      simbFila ← Desenfileira símbolo da filaInfixa
4.      Se o simbFila é "operando":
5.          Enfileira o simbFila na filaPosFixa
6.      Senão Se simbFila é "abre parênteses":
7.          Empilha o simbFila na pilhaConv
8.      Senão Se simbFila é "operador":
9.          Enquanto (pilhaConv não é vazia) E (símbolo do topo da pilhaConv
    tiver prioridade maior ou igual ao simbFila):
10.             simbPilha ← Desempilha símbolo da pilhaConv
11.             Enfileira simbPilha na filaPosFixa
12.         Fim Enquanto
13.         Empilha o simbFila na pilhaConv
14.     Senão Se simbFila é "fecha parênteses":
15.         Enquanto o símbolo do topo da pilhaConv não é "abre parênteses":
16.             simbPilha ← Desempilha símbolo da pilhaConv
17.             Enfileira simbPilha na filaPosFixa
18.         Fim Enquanto
19.         Desempilha símbolo da pilhaConv
20.     Fim Se
21. Fim Enquanto
22. Enquanto pilhaConv não é vazia:
23.     simbPilha ← Desempilha símbolo da pilhaConv
24.     Enfileira simbPilha na filaPosFixa
25. Fim Enquanto
26. O resultado do algoritmo é a filaPosFixa
```

Passo 3:

- Executar o cálculo da expressão na forma pós-fixa.
- Como entrada, o algoritmo recebe o objeto da estrutura fila gerado no Passo 2, contendo os símbolos na notação pós-fixa. Como saída, o resultado da avaliação da expressão é calculado.
- O algoritmo a seguir apresenta a lógica de avaliação de expressões pós-fixas.

Algoritmo: Avaliação de expressões pós-fixas

```
1.  pilhaCalc ← Inicia uma pilha vazia
2.  Enquanto a filaPosFixa não é vazia:
3.      simbFila ← Desenfileira símbolo da filaPosFixa
4.      Se o simbFila é "operando":
5.          Empilha o simbFila na pilhaCalc
6.      Senão Se simbFila é "operador":
7.          operandoA ← Desempilha símbolo da pilhaCalc
8.          operandoB ← Desempilha símbolo da pilhaCalc
9.          resultado ← operandoB operador operandoA
10.         Empilha resultado na pilhaCalc
11.     Fim Se
12. Fim Enquanto
13. O resultado do algoritmo está no topo da pilhaCalc
```

Observações importantes:

- A calculadora irá suportar a seguintes operações:
  - Abre e fecha parênteses (símbolos "(" e ")") - prioridade 0 (*)
  - Adição (símbolo "+") e Subtração (símbolo "-") - prioridade 1 (*)
  - Multiplicação (símbolo "*" e Divisão (símbolo "/") - prioridade 2 (*)

  **** utilizar estas prioridades para a comparação entre os operadores exigida na linha 9 do algoritmo de conversão de notações***

- Considere a criação de uma classe `Simbolo` para armazenar objetos String de cada símbolo. Além disso, é aconselhável a criação de métodos para facilitar a manipulação: `isOperando()`, `isOperador()`, `isAbreParenteses()`, `isFechaParenteses()`, `verificaPrioridade()`, ...