Hoja de trabajo No. 3

Realizar: Programa que para evaluar expresiones **infix** (Deben ser convertidas a **postfix** por el programa y luego ser evaluadas).
Realizarse: De forma individual.

Objetivos:
a. Utilización de genéricos.
b. Diseño del ADT para pilas (stack), debe tener: 1. Interfaz, 2. Clase Abstracta, 3. Clases de implementación.
c. Diseño del ADT para listas, con: 1.Interfaz, 2. Clase Abstracta, 3. Clases de implementación.
d. Uso de patrones de diseño: Factory y Singlenton.
e. Control de versiones del programa.
f. Emplear JUnit para casos de prueba.

Programa a realizar:
Su programa debe realizar el cálculo de una expresión infix, para lograrlo de primero debe convertir la expresión a postfix y luego evaluarla tal como se realizó en la hoja de trabajo no. 2. Debe leer esa expresión de un archivo de texto llamado **datos.txt.**

Por ejemplo, el cálculo de la expresión postfix: (1+2)*9
La expresión es evaluada obtendrá como resultado 27

En los anexos de esta hoja de trabajo encontrará el algoritmo necesario para la conversión de una expresión infix a postfix, para simplificar asuma que los números serán únicamente de un solo dígito.

NOTA: esta hoja implementa varios patrones de diseño. Su programa debe solicitar al usuario que indique la implementación deseada para el stack (arrayList, vector, lista). Si se desea usar la implementación de stack basada en listas, entonces también debe solicitar al usuario que indique la implementación de listas a emplear (simplemente encadenada, doblemente encadenada).

Tareas:
1. Diagrama UML de clases. NO utilice ingeniería reversa. Muestre la relación entre las clases que usa para la pila y las utilizadas para la lista.
2. Construir la interfaz de la Pila, su clase abstracta y las clases de implementación con: 1. ArrayList, 2. Vector, 3. Lista. Debe utilizar genéricos.
3. Construir la interfaz de la Lista, su clase abstracta y las clases de implementación con: 1. Simplemente encadenadas, 2. Doblemente encadenadas. Debe utilizar genéricos. **NOTA: no es necesario que implemente todos los métodos de la lista**. Implemente solo aquellos que sean necesarios para soportar la operación de una pila.
4. Su programa principal (realizado en la hoja no. 2) debe usar:
    a. **Patron de diseño Factory**: para seleccionar la implementación de la pila a utilizar. Favor notar que si selecciona una implementación basada en listas, deberá utilizar nuevamente ese patrón para indicar la implementación de lista a utilizar.
    b. **Patron de diseño Singlenton**: para asegurarse que existe solo una instancia de su clase Calculadora. Esta es independiente de la GUI que usted utilice.
5. Debe dejar evidencia de todo el desarrollo en el repositorio de Git o sistema similar para control de versiones. Indicar como acceder a su repositorio y si es necesario, agregar a su catedrático y auxiliar para que tengan acceso al mismo.
6. Incluya pruebas unitarias con JUnit para las clases de implementación de pila y la clase lista.
7. Revise que ventajas / desventajas hay al utilizar el patrón Singlenton en general, ya que su comportamiento es muy similar a una variable global. ¿Cree que su uso es adecuado en este programa? Conteste en un documento PDF.

Debe subir a Canvas todos los productos elaborados y los enlaces a su repositorio de Git (o similar), y también el documento PDF con su respuesta al uso de Singlenton.

Calificación: deben existir los diagramas de Clases para que sea calificado su programa.

Aspecto	Puntos
Documentación generada con Javadoc, tiene precondiciones y postcondiciones en los métodos de los ADT empleados	5
Uso del repositorio: existen más de tres versiones guardadas, la última versión es igual a la colocada en el Canvas	5
Diagrama de clases: muestran la abstracción y encapsulación de las operaciones	10
Utilización de los patrones de diseño Factory y Singlenton	20
ADT pila, con uso de genéricos	10
ADT lista, con uso de genéricos	20
Funcionamiento del programa	10
Pruebas JUnit para las operaciones de la pila / lista / calculadora	15
Respuesta a pregunta sobre uso de SINGLENTON	5
TOTAL:	100

ANEXOS

Infix to Postfix algorithm using Stack

Inputs

1. The infix expresión
   a. 1+2*9
   b. (1+2)*9

Output

1. The postfix expresión
   a. 1 2 9 * +
   b. 1 2 + 9 *

Algorithm

```
1. Begin
2.    initially push some special character say # into the stack
3.    for each character ch from infix expression, do
4.       if ch is alphanumeric character, then
5.          add ch to postfix expresión (queue or list)
6.       else if ch = opening parenthesis (, then
7.          push ( into stack
8.       else if ch = ^, then //any operator of higher precedence
9.          push ^ into the stack
10.      else if ch = closing parenthesis ), then
11.         while stack is not empty and stack top ≠ (, do
12.            pop and add item from stack to postfix expression
13.         done
14.
15.          pop ( also from the stack
16.       else
17.               while stack is not empty AND precedence of ch <=
   precedence of stack top element, do
18.                  pop and add into postfix expression
19.               done
20.
21.               push the newly coming character.
22.         done
23.
24.         while the stack contains some remaining characters, do
25.            pop and add to the postfix expression
26.         done
27.         return postfix
28.      End
```

Source: https://www.tutorialspoint.com/Convert-Infix-to-Postfix-Expression#:~:text=To%20convert%20infix%20expression%20to,maintaining%20the%20precedence%20of%20them. Accessed on Feb. 14th. 2021.

INFIX TO POSTFIX ALGORITHM - EXAMPLE 1			
STEP	Input (String)	Stack	Output (String)
1	1+2*9		
2	+2*9		1
3	2*9	+	1
4	*9	+	1 2
5	9	* +	1 2
6		* +	1 2 9
7		+	1 2 9 *
8			1 2 9 * +

INFIX TO POSTFIX ALGORITHM - EXAMPLE 2			
STEP	Input (String)	Stack	Output (String)
1	(1+2)*9		
2	1+2)*9	(	
3	+2)*9	(	1
4	2)*9	+ (	1
5	)*9	+ (	1 2
6	*9		1 2 +
7	9	*	1 2 +
8		*	1 2 + 9
9			1 2 + 9 *