Instituto Tecnológico Autónomo de México

Primer Proyecto
Calculadora Aritmética

Estructura de Datos

Maria Meraz Inostrosa
Martin Alejandro Pozos Arreola
Mauricio Yañez Barrera

16 Marzo 2023

Índice

Descripción del problema:

Objetivo:

El objetivo de este proyecto es crear una calculadora empleando el uso de pilas y de interfaz gráfica para crear el diseño de la calculadora. Esta calculadora será capaz de realizar operaciones básicas como: suma, resta, multiplicación, división, obtener el inverso de un número, etc.; además la calculadora usará el punto decimal, podrá borrar elemento por elemento o incluso borrará toda la operación ingresada al oprimir un botón.

La calculadora evaluará operaciones aritméticas usando paréntesis “()”, con el fin de checar si la operación esta equilibrada (es decir, debe tener la misma cantidad de paréntesis que abren y paréntesis que cierran), en caso de que la operación ingresada no este equilibrada, mandará un mensaje de error ya que no se ingresó de manera correcta la operación.

Requisitos:

La calculadora debe ser capaz de manejar números negativos y positivos, debe realizar las operaciones aritméticas básicas (suma, resta, multiplicación, división), debe leer de forma correcta el punto decimal y poder trabajar sin ningún inconveniente, realizando todas las operaciones ingresadas por él usuario.

Restricciones:

Las restricciones que se presentan para este proyecto son:

- Se debe de usar la aplicación Java NetBeans.
- Se deben usar los métodos de pilas.
- Se debe crear una interfaz gráfica “agradable” para el usuario.
- La calculadora debe ser capaz de obtener los resultados a partir de las operaciones ingresadas.
- La calculadora deberá mostrar los resultados correctos en la pantalla de la interfaz gráfica del programa realizado.

Solución diseñada

(UML de clases, algoritmos principales)

Calcu
memoria1: String memoria2: String signo: String
+Calcu() +resta(pantallatxt) +suma(pantallatxt) +multiplicacion(pantallatxt) +division(pantallatxt) +borrar() +borrarTodo() +inversa() +calcula(String memoria1,String memoria2, String signo): String +existepunto(String cadena): boolean

PilaA
-colect: T -tope:int -MAX: int
+PilaA() +PilaA(int max) +push(T dato):void +pop():T +peek():T +isEmpty():Boolean +aumentaCapacidad():void

PilaADT
+pop():T +push(T dato):T +peek():T +isEmpty():boolean

Posfijo
-cadena:String

+Posfijo()
+Posfijo(String cadena)
+obtieneTokens():String[]
+conviertePostfija():String[]
-noEsOperador(String dato):boolean
-prioridad(String dato):int

Calculadora

-resultado:double

+Calculadora()
+getResultado():double
+calcula(String entrada):double
-revisa(String entrada):boolean
-obtieneTokens[](String entrada):String
-conviertePostfija[](String entrada):String
-noEsOperador(String dato):boolean
-prioridad(String dato):int
-evalúa(String postfija[]):double

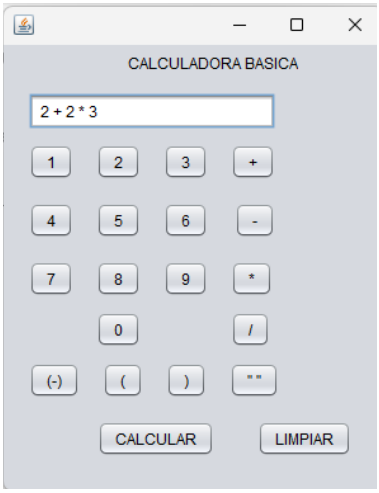
Pruebas (descripción de las pruebas realizadas, resultados obtenidos, análisis)

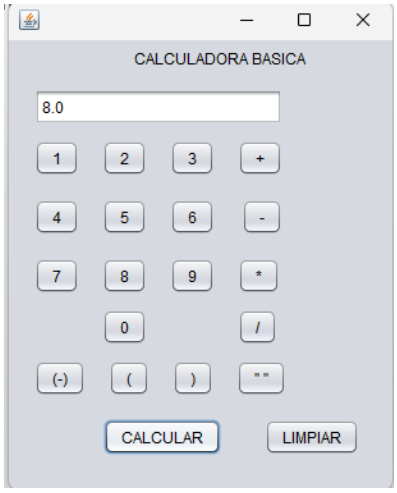
En esta sección se mostrarán los resultados obtenidos al realizar el programa de la interfaz gráfica de la calculadora.

Para demostrar su funcionamiento se ingresarán las siguientes operaciones y se mostrarán en pantalla el resultado.

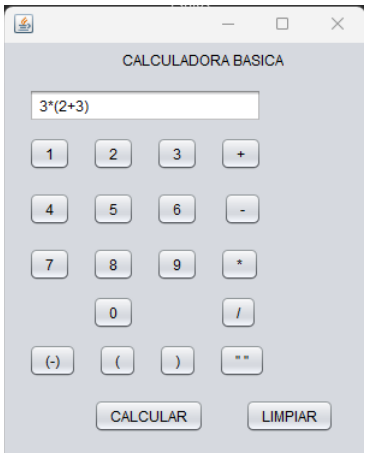
Ejemplos:

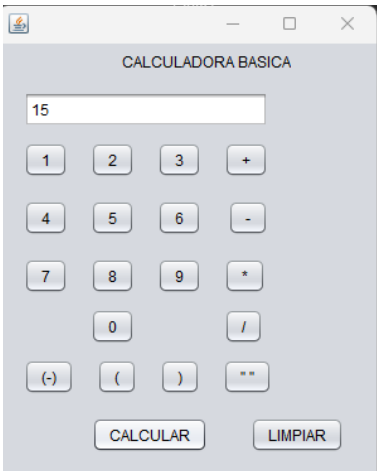
1) 2+2*3= 8



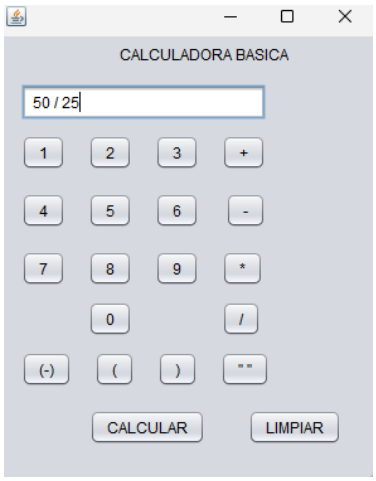


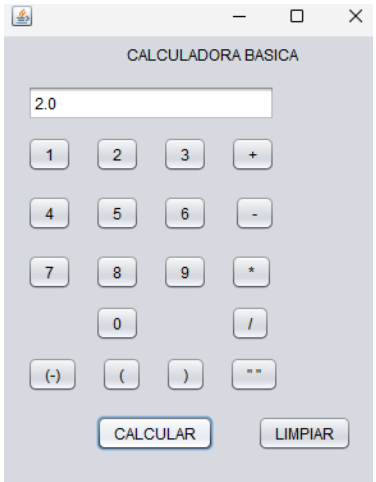
2) 3*(2+3) =15





3) 50/25





Estos resultados se verificaron con el uso calculadora solamente para ver si el resultado estaba correcto.

Viendo los resultados que se realizaron anteriormente, los resultados están correctos, por lo que podemos llegar a que la calculadora puede realizar operaciones aritméticas básicas de manera correcta.

Limitaciones de la solución

Al momento de realizar este proyecto se presentaron limitaciones en el programa de la interfaz gráfica de la calculadora.

Solamente se presento una limitación en el programa; la limitación es que al momento de ingresar los valores numéricos es que es necesario ingresar espacios después de cada número, paréntesis u operador para que de esta forma el programa funcione correctamente.

En caso de que no se ingresen los espacios como se mencionó con anterioridad, al ingresar los valores, mandará error de código y no funcionará el programa.

Es la única limitación que se presenta en el programa, que a futuro se puede mejorar para evitar estos percances que se presentan en el programa.

Posibles mejoras y conclusiones

Algunas posibles mejorar que se pueden realizar al programa podría ser que se pueda mejorar es la parte de la presentación de la interfaz gráfica, se podría añadir más funciones matemáticas a la calculadora para que de una cierta forma sea lo más parecido a las funciones de una calculadora científica, otra mejora que se puede realizar es mejorar el aspecto que al ingresar los valores en la calculadora para que no necesite añadir espacios entre cada valor ingresado, que sea posible ingresar los valores juntos y de esta forma, también se pueda ver en pantalla el resultado correcto

Cómo conclusión al elaborar este proyecto nos dio una buena y clara idea de como es el diseño y el programa que utilizan las calculadoras que hoy en día usamos.

Posiblemente la mayoría de las personas han pensado que programar una calculadora es fácil, en teoría si es algo fácil, pero tiene su chiste realizar cada paso de la calculadora; como por ejemplo la parte de la interfaz gráfica de la calculadora tiene que ser “agradable” a la vista del usuario (es decir, que a simple vista se entienda los comandos que puede realizar la calculadora), otro ejemplo es la programación de la calculadora, porque primero se realiza la parte de la interfaz gráfica y a partir de aquí, se empieza a programar y darle la función a cada parte que conformará la calculadora.

En este proyecto el reto que enfrentamos fue juntar el programa con los métodos de pilas que se tenían que emplear, porque en esta sección del programa tiene que

evaluar expresiones aritméticas y mostrar el resultado correcto. Un ejemplo para explicar esto sería como esta siguiente expresión:

1) 2+3*2= 8 → Primero se realiza la multiplicación por jerarquía de operaciones, y al final la suma.
2) 2*(2+2) =8→En este caso se realiza primero lo que esta dentro del paréntesis, después se realizará la multiplicación
3) (3*(8/2) =? → En este caso cómo no se ingresaron la cantidad de paréntesis en la expresión, mandará un error porque no está completa la expresión matemática para realizar la operación, porque el programa recorre la expresión matemática ingresada y checa los paréntesis; observa que hay un paréntesis que abre "(", después checa que hay otro paréntesis que abre "(", luego hay un paréntesis que cierra ")" por lo que al anterior paréntesis que abre ya está completa una pequeña parte de la operación, pero al final no encuentra otro paréntesis que cierre por lo que el paréntesis que abre del principio no cierra en ningún momento, así que la expresión matemática que se ingresó está incompleta y no se realizará la operación hasta que se complete la expresión.
   Este fue el reto más interesante del proyecto, pero se logró conseguir el objetivo, por lo que nos deja una enseñanza a cada miembro del equipo que a futuro se podrá utilizar este aprendizaje para proyectos.

Apéndice del código

Clase Calculadora

```
1. /*
2.  * Click
   nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-
   default.txt to change this license
3.  * Click
   nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java
   to edit this template
4.  */
5. package calculadora;
6.
7. /**
8.  *
9.  * @author mariameraz
10.       */
11.     public class Calculadora {
12.       // Es una expresión dada en notación infija
13.         private double resultado;
14.
15.         public Calculadora() {
16.         }
17.
18.         // Se construye un objeto tipo calculadora y se le
   asigna un valor a la
19.         //expresión infija
20.
21.         // Método para cambiar el valor de la expresión que
   se evaluará por
22.         //medio de la calculadora
23.
24.         // Método para obtener el resultado
25.         public double getResultado(){
26.             return resultado;
27.         }
28.
29.         /* Método que tiene el control de las operaciones
   que lleva a
30.          * cabo la calculadora para evaluar la expresión. Si
   la expresión
31.          * se puede evaluar deja el resultado en
32.          * el atributo resultado y regresa true. En caso
   contrario regresa false.
33.          */
34.         public static double calcula(String entrada){
35.             double resultado=-1;
36.             boolean resp;
```

```
37.
38.             resp = revisa(entrada); // Revisa que los
  paréntesis estén bien balanceados
39.             if (resp){
40.                 String[] elementos = obtieneTokens(entrada);
   // Obtiene los elementos de la expresión
41.                 elementos = conviertePostfija(elementos); //
  Convierte a postfija
42.                 resultado = evalúa(elementos); // Evalúa la
  expresión ya en notación postfija
43.             }
44.             return resultado;
45.         }
46.
47.         /* Método auxiliar que revisa si la expresión dada
  en notación infija tiene los
48.          * paréntesis bien balanceados. Es decir, si el
  número de paréntesis izquierdos
49.          * concuerda con el número de paréntesis derechos.
50.          * Utiliza un objeto tipo PilaE para almacenar los
  paréntesis izquierdos temporalmente.
51.          */
52.         private static boolean revisa(String entrada){
53.             PilaADT<Character> pila = new PilaA();
54.             boolean resp;
55.             int i, n;
56.
57.             n = entrada.length();
58.             i= 0;
59.             resp = true;
60.             while (i < n && resp){
61.                 if (entrada.charAt(i) == '(')
62.                     pila.push(entrada.charAt(i)); // Guarda
  el paréntesis izquierdo
63.                 else
64.                     if (entrada.charAt(i) == ')') // Si es
  paréntesis derecho
65.                         if (pila.isEmpty()) // intenta sacar
  su correspondiente izquierdo
66.                             resp = false;   // de la pila.
  Si no hay, altera la variable
67.                         else                // para no
  seguir analizando la expresión.
68.                             pila.pop();
69.                 i++;
70.             }
71.             return resp && pila.isEmpty();
72.         }
73.
74.         /* Método auxiliar que permite separar -por medio
  del método split() de la clase String
```

```
75.          * de Java- la cadena dada y guarda cada uno de sus
   elementos (operadores, operandos
76.          * y paréntesis) en un arreglo de cadenas.
77.          */
78.         private static String[] obtieneTokens(String
   entrada){
79.           return entrada.split(" ");
80.         }
81.
82.         /* Método auxiliar que recibe un arreglo de cadenas
   que representa a una cadena en
83.          * notación infija y regresa otro arreglo con la
   misma expresión pero ahora en
84.          * notación postfija. Usa un objeto tipo PilaE para
   almacenar temporalemente algunos
85.          * elementos de la expresión.
86.          */
87.         private static String[] conviertePostfija(String
   elementos[]){
88.             String
   postfija[] = new String[elementos.length];
89.             PilaADT <String> pila = new PilaA();
90.             int e, p, n;
91.
92.             e = 0;
93.             p = 0;
94.             n = elementos.length;
95.             while (e < n){
96.                 if (elementos[e].equals("(")) // Es
   paréntesis izquierdo
97.                     pila.push(elementos[e]);
98.                 else
99.                     if (elementos[e].equals(")")){ // Es
   paréntesis derecho
100.                        while (!pila.peek().equals("(")){
101.                            postfija[p] = pila.pop();
102.                            p++;
103.                        }
104.                        pila.pop();
105.                    }
106.                    else
107.                        if (noEsOperador(elementos[e])){ //
   Es un operando
108.                            postfija[p] = elementos[e];
109.                            p++;
110.                        }
111.                        else {  // Es un operador
112.                            while (!pila.isEmpty() && prior
   idad(pila.peek())
113.                                    > prioridad(elementos[e]
   )){
```

```
114.                                 postfija[p] = pila.pop();
115.                                 p++;
116.                             }
117.                             pila.push(elementos[e]);
118.                         }
119.                 e++;
120.             }
121.              while (!pila.isEmpty()){
122.                 postfija[p] = pila.pop();
123.                 p++;
124.              }
125.              return postfija;
126.         }
127.
128.         // Método auxiliar que regresa true si la cadena
     recibida no es un operador
129.         private static boolean noEsOperador(String dato){
130.             return !dato.equals("+") && !dato.equals("-
     ") && !dato.equals("*") && !dato.equals("/");
131.         }
132.
133.         /* Método auxiliar para el manejo de las prioridades
     de los operadores. Regresa 0,
134.          * el valor más pequeño, cuando el dato dado es un
     "(". De esta manera
135.          * el "(" sólo se saca de la pila cuando se
     encuentre un ")".
136.          */
137.         private static int prioridad(String dato){
138.             int resultado = 0; // En caso de que el dato sea
     un paréntesis izquierdo
139.
140.             switch (dato.charAt(0)){
141.                 case '+':
142.                 case '-': resultado = 1;
143.                     break;
144.                 case '*':
145.                 case '/': resultado = 2;
146.             }
147.             return resultado;
148.         }
149.
150.         /* Método auxiliar para evaluar una expresión dada
     en notación postfija.
151.          * Usa un objeto tipo PilaE para almacenar
     temporalmente los operandos y los
152.          * resultados parciales que se van obteniendo.
153.          * Regresa un dato tipo double.
154.          */
155.         private static double evalúa(String postfija[]){
156.             PilaADT<Double> pila = new PilaA();
```

```
157.             double resul, op1, op2;
158.             int i;
159.
160.             resul = 0;
161.             i = 0;
162.             while (i < postfija.length && postfija[i] != nu
   ll){
163.                 if (noEsOperador(postfija[i])){ // Es
   operando
164.                     double
   n = Double.parseDouble(postfija[i]);
165.                     pila.push(n);
166.                 } else { // Es operador
167.                     op2 = pila.pop();
168.                     op1 = pila.pop();
169.                     switch (postfija[i].charAt(0)){
170.                         case '+': resul = op1 + op2;
171.                             break;
172.                         case '-': resul = op1 - op2;
173.                             break;
174.                         case '*': resul = op1 * op2;
175.                             break;
176.                         case '/': if (op2 == 0) // Si el
   denominador es 0 se lanza una excepción
177.                                                 throw new RuntimeEx
   ception();
178.                                     resul = op1 / op2;
179.                     }
180.                     pila.push(resul);

181.                     }
182.                 i++;
183.             }
184.               return pila.pop();
185.             }
186.         public static void main (String[] args){
187.             System.out.println(Calculadora.calcula("2 + ( 2
   * 3 ) "));
188.         }
189.     }
```

Clase ExcepciónColecciónVacía

```
1. /*
2.  * Click
   nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-
   default.txt to change this license
3.  * Click
   nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java
   to edit this template
4.  */
5. package calculadora;
6.
7. /**
8.  *
9.  * @author mariameraz
10.       */
11.     public class ExcepciónColecciónVacía extends RuntimeExce
   ption{
12.
13.         public ExcepciónColecciónVacía(){
14.             super("Colección vacía");
15.         }
16.
17.         public ExcepciónColecciónVacía(String mensaje){
18.             super(mensaje);
19.         }
20.     }
```

Clase PilaA

```
1. package calculadora;
2. /**
3.  * @author Silvia Guardati
4.  * Programa 7.2
5.  * Clase que implementa una pila genérica usando un arreglo
   genérico.
6.  */
7. public class PilaA <T> implements PilaADT <T>{
8.     private T colec[];
9.     private int tope;
10.         private final int MAX = 10;
11.
12.         /* Se construye un arreglo de objetos y se lo
   convierte explícitamente a tipo T.
13.          * Inicialmente la pila está vacía, lo que se indica
   con tope igual a -1.
14.          */
15.         public PilaA() {
16.             colec = (T[]) (new Object[MAX]);
```

```
17.             tope = -1;
18.         }
19.
20.         /* Se construye un arreglo de objetos y se lo
    convierte explícitamente a tipo T.
21.          * El espacio máximo reservado queda determinado por
    el parámetro max.
22.          * Inicialmente la pila está vacía, lo que se indica
    con tope igual a -1.
23.          */
24.         public PilaA(int max) {
25.             colec = (T[]) (new Object[max]);
26.             tope = -1;
27.         }
28.
29.         /*
30.          * Agrega el dato en el tope, redefiniendo el valor
    de éste.
31.          * Si la pila está llena,se construye un arreglo de
    mayor capacidad
32.          * y se copian los elementos de la pila a éste.
33.          */
34.         public void push(T dato) {
35.             if (tope == colec.length - 1)
36.                 aumentaCapacidad();
37.             tope++;
38.             colec[tope] = dato;
39.         }
40.
41.         /*
42.          * Elimina y regresa el elemento que está en el tope
    de la pila,
43.          * redefiniendo el valor del tope. Si la pila está
    vacía lanza una excepción.
44.          */
45.         public T pop() {
46.             if (isEmpty())
47.                 throw new ExcepciónColecciónVacía("Pila
    vacía");
48.             else{
49.                 T dato = colec[tope];
50.                 tope--;
51.                 return dato;
52.             }
53.         }
54.
55.         /* Regresa el elemento que está en el tope.
56.          * Si la pila está vacía lanza una excepción.
57.          */
58.         public T peek() {
59.             if (isEmpty())
```

```
60.                 throw new ExcepciónColecciónVacía("Pila
   vacía");
61.             else
62.                 return colec[tope];
63.         }
64.
65.         // Regresa true si la pila está vacía.
66.         public boolean isEmpty() {
67.             return tope == -1;
68.         }
69.
70.         /* Método auxiliar que construye un arreglo de mayor
   tamaño y copia en él todos
71.          * los elementos de la pila, asignando al arreglo
   colec la referencia del nuevo
72.          * arreglo.
73.          */
74.         private void aumentaCapacidad(){
75.             T
   nuevo[] = (T[]) (new Object[colec.length * 2]);
76.             int i;
77.
78.             for (i= 0; i <= tope; i++){
79.                 nuevo[i] = colec[i];
80.             }
81.             colec = nuevo;
82.         }
83.     }
```

Clase PilaADT

```
1. /*
2.  * Click
   nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-
   default.txt to change this license
3.  * Click
   nbfs://nbhost/SystemFileSystem/Templates/Classes/Interface.ja
   va to edit this template
4.  */
5. package calculadora;
6.
7. /**
8.  *
9.  * @author mariameraz
10.       */
11.     public interface PilaADT <T>{
12.         public T pop(); // Debe quitar el elemento que está
   en el tope y regresarlo.
13.         public void push(T dato); // Agrega el dato en el
   tope de la pila.
```

```
14.         public T peek(); // Regresa el elemento que está en
    el tope, sin quitarlo.
15.         public boolean isEmpty(); // Regresa true si la pila
    no tiene elementos.
16.     }
```

Clase PostFijo

```
1. /*
2.  * Click
    nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-
    default.txt to change this license
3.  * Click
    nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java
    to edit this template
4.  */
5. package calculadora;
6.
7. /**
8.  *
9.  * @author mariameraz
10.      */
11.     public class PostFijo {
12.         //Atributo
13.         private String cadena;
14.         //Constructores
15.         public PostFijo(){
16.
17.         }
18.         public PostFijo(String cadena){
19.             this.cadena = cadena;
20.         }
21.         //Metodos
22.         public static String[] obtenTokens(String cadena){
23.             return cadena.split(" ");
24.             /*
25.              * Devuelve la cadena dentro de celdas con
    separacion en cada espacio
26.             */
27.         }
28.         private static boolean noEsOperador(String dato){
29.             /*
30.             Evalua si los caracteres dentro de la cadena es
    o no un operador
31.             */
32.             return !dato.equals("+") && !dato.equals("-
    ") && !dato.equals("*")
33.                     && !dato.equals("/");
```

```
34.         }
35.         private static int prioridad(String dato){
36.             int resultado = 0; // En caso de que el dato sea
  un paréntesis izquierdo
37.             /*
38.             Evalua la prioridad del operador segun la
  jerarquía de operaciones
39.             */
40.             switch (dato.charAt(0)){
41.                 case '+':
42.                 case '-': resultado = 1;
43.                     break;
44.                 case '*':
45.                 case '/': resultado = 2;
46.             }
47.             return resultado;
48.         }
49.         public static String[] conviertePostFija(String
  cadena){
50.             String[] elementos = obtenTokens(cadena);
51.             String[] postFija = new String[elementos.length]
  ;
52.             PilaADT <String> pila = new PilaA();
53.             int e, p, n;
54.             n= elementos.length;
55.             e=0;
56.             p=0;
57.             while(e<n){
58.                 if(elementos[e].equals("(")){
59.                     /*
60.                     Si el elemento de la cadena es un
  parentesis de entrada,
61.                     se introduce el caracter a la pila
62.                     */
63.                      pila.push(elementos[e]);
64.                 }else if(elementos[e].equals(")")){
65.                     /*
66.                     En caso de no ser un parentesis de
  entrada, sino que uno
67.                     de salida, mientras no se halle en la
  pila un parentesis
68.                     de entrada, se introduce en la casilla
  correspondiente el ultimo
69.                     caracter de la pila y se aumenta el
  numero de casillas ocupadas
70.                     dentro del arreglo postFija
71.                     */
72.                     while (!pila.peek().equals("(")){
73.                             postFija[p] = pila.pop();
74.                             p++;
75.                         }
```

```
76.                         pila.pop();
77.                 }else if(noEsOperador(elementos[e])){
78.                     postFija[p] = elementos[e];
79.                     p++;
80.                 }else{
81.                     while(!pila.isEmpty() && prioridad(pila
   .peek()) > prioridad(elementos[e])){
82.                         postFija[p] = pila.pop();
83.                         p++;
84.                     }
85.                     pila.push(elementos[e]);
86.                 }
87.               e++;
88.             }
89.             while (!pila.isEmpty()){
90.                 postFija[p] = pila.pop();
91.                 p++;
92.             }
93.             return postFija;
94.         }
95.         public static double evalua(String postfija[]){
96.             PilaADT<Double> pila = new PilaA();
97.             double resul, op1, op2;
98.             int i;
99.
100.            resul = 0;
101.            i = 0;
102.            while (i < postfija.length && postfija[i] != nu
   ll){
103.                if (noEsOperador(postfija[i])) // Es
   operando
104.                    pila.push(Double.valueOf(postfija[i]));
105.                else { // Es operador
106.                    op2 = pila.pop();
107.                    op1 = pila.pop();
108.                    switch (postfija[i].charAt(0)){
109.                        case '+' -> resul = op1 + op2;
110.                        case '-' -> resul = op1 - op2;
111.                        case '*' -> resul = op1 * op2;
112.                        case '^' -> {
113.                            double pot = op1;
114.                            for(int j=0;j<=op2;j++){
115.                                pot = (pot*op1);
116.                            }
117.                            resul = pot;
118.                        }
119.                        case '/' -> {
120.                            if (op2 == 0) // Si el
   denominador es 0 se lanza una excepción
```

```
121.                                 throw new RuntimeException(
    );
122.                                 resul = op1 / op2;
123.                             }
124.                         }
125.                         pila.push(resul);

126.                         }
127.                     i++;
128.                 }
129.                   return pila.pop();
130.                 }
131.     }
```

Link GitHub

https://github.com/Mau-2911/Calculadora

Bibliografías y referencias

Guardati Buemo Silvia, (2015), ***"Estructura de datos básicas. Programación orientada a objetos con Java"***, Primera edición, Alfaomega Grupo Editor, México, págs.333-347, Recuperado de: ESTRUCTURAS DE DATOS BÁSICAS - Programación Orientada a Objetos con Java (elogim.com).