ITAM

Calculadora

María José Domenzain	C.U. 157144
Rodrigo González	C.U. 183873
Montserrat Olivares	C.U. 179905
Sebastián Zarate	C.U. 180087

Estructuras de Datos

21 · febrero · 2019

Índice

Descripción del problema

Introducción

El propósito del presente documento es dar un informe global sobre el proyecto realizado, los métodos diseñados, así como los resultados obtenidos.
Utilizando lo aprendido en clase se realizó una calculadora capaz de validar y, en caso necesario, corregir, los datos introducidos por el usuario para con éstos realizar las operaciones aritméticas básicas requeridas.

Objetivo

Colaborar en equipo para aplicar los conceptos aprendidos en clase, principalmente el de la estructura pila, y de esta forma diseñar una calculadora funcional con interfaz gráfica amigable que realice las operaciones aritméticas requeridas por el usuario.

Requisitos

Las condiciones necesarias para la correcta elaboración del proyecto son:

1. El programa debe ser capaz de realizar operaciones con números reales, ya sean enteros, decimales, o negativos.

2. Debe haber una interfaz gráfica fácil de utilizar, con los botones correspondientes a los números, signos de puntuación, y operadores aritméticos necesarios para escribir las expresiones que se quieran evaluar.

3. Una vez que el usuario valide la operación que desea realizar, el programa debe ser capaz de verificar la validez aritmética de dicha expresión en notación infija, convertir esta última a notación postfija, y finalmente evaluar la expresión.

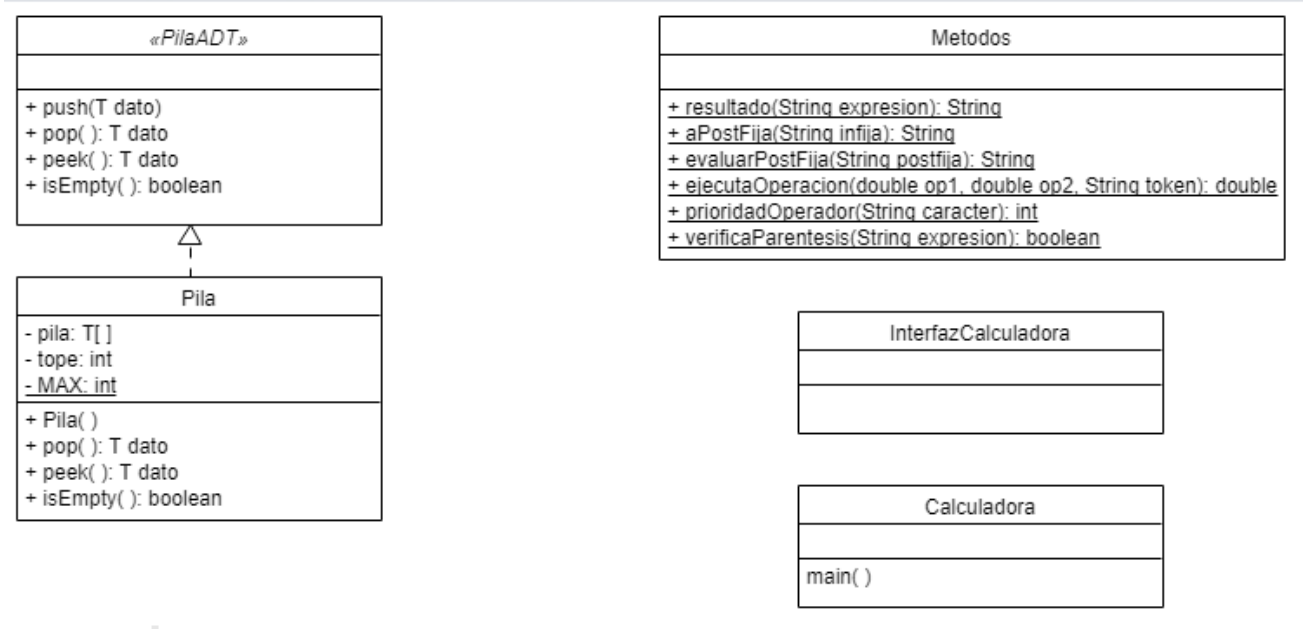
Restricciones

- El programa debe ser elaborado a través de la plataforma NetBeans utilizando el lenguaje de programación Java.

- La calculadora no puede realizar operaciones más complejas que las aritméticas básicas (suma, resta, multiplicación, y división).

Solución diseñada

Diagrama UML



Nota: Declaramos los métodos como públicos para poder implementar el JavaDoc, inicialmente habíamos considerado todos los métodos, excepto el de **resultado**, como privados pues no queremos que puedan ser modificados por el usuario.

resultado(String expresion): String

Éste método recibe como parámetro la expresión a evaluar en forma de cadena. A continuación, declara una variable del mismo tipo, *res* y llama al método **verificaParentesis(String)** para validar la expresión. Si el balance de paréntesis es correcto, llama al método **evaluarPostFija(String)** que recibe la expresión en forma postfija, transformada de su forma infija por el método **aPostFija(String)**; y devuelve el resultado de las operaciones. En caso de que los paréntesis no estén balanceados, devuelve la frase 'Paréntesis incorrectos'.

aPostFija(String infija): String

El parámetro que recibe este método es la expresión de tipo cadena que el usuario ingresa en notación infija. Se declara una pila, en la que se valida el balance los paréntesis, así como una variable de tipo cadena, *token*, la cual lee caracter por caracter la expresión recibida. Posteriormente, se llama al método **prioridadOperador(String)** ingresando como parámetro el token leído para poder asignarle una prioridad, y de acuerdo a esta validar si se trata de un paréntesis, de un operador, o de un número, para de esta forma acomodarlo en la pila o en el *StringBuilder postfija* en el que se guarda la expresión en notación postfija.
Finalmente, el método devuelve la expresión que el usuario ingrese, en notación postfija.

evaluarPostFija(String postfija): String

El método recibe como parámetro la expresión del usuario en notación postfija, declara una pila de tipo doble, un *scanner*, dos variables de tipo cadena: **token** y **resultadoStr**, ambas inicializadas en null, y tres variables de tipo doble: *op1*, *op2* y *res*. Con un ciclo *while* oobtiene el caracter siguiente a través del *scanner* y lo asigna a *token*. Posteriormente se verifica la prioridad, y si esta es igual a 0, es decir, es un número, y se agrega su valor en doble a la pila. En caso contrario, se le asigna los dos últimos valores almacenados en la pila a *op1* y *op2*, y se manda llamar al método **ejecutaOperacion(double, double, String)** con *op1*, *op2* y *token*. Se introduce a la pila el resultado de la operación y a *res* se le asigna el último valor de la pila. A continuación, se verifica si la pila está vacía, en cuyo caso se le asigna a *resultadoStr* el valor en cadena de *res*. En caso contrario, se devuelve la expresión "faltan operadores".
Si no se pueden obtener los valores a través del *scanner*, se lanza la excepción correspondiente y se le asigna a *resultadoStr* un comentario acorde, ya sea por falta de operadores o por operaciones indeterminadas, como dividir entre cero. Finalmente, el método devuelve *resultadoStr*.

ejecutaOperacion(double op2, double op1, String token): double

Lo que este método recibe como parámetros son dos números de tipo doble, *op2* y *op1*, y una cadena, *token*. Declara e inicializa una variable de tipo doble donde se almacenará el resultado de la operación. A continuación, con el primer caracter del token ingresado, se realiza la operación que este último indique a través de un *switch*.[1] Como respuesta, el método devuelve el resultado de la operación realizada.

[1] En la división está contemplado el caso en el que usuario intente dividir entre cero

prioridadOperador(String caracter): int

Aquí, el método recibe como parámetro un caracter. A continuación, declara la variable prioridad, que es de tipo entero, y que será asignada al caracter tras compararlo con los diferentes operadores o símbolos posibles. Si el caracter es un + o un −, le asigna 4 como prioridad, a los operadores ∗ y /, les asigna el valor de 3, al paréntesis izquierdo le da como prioridad 1, y al derecho, 2. Finalmente, a cualquier otro símbolo le da como valor 0. El método regresa ese valor entero de prioridad.

verificaParentesis(String expresion): boolean

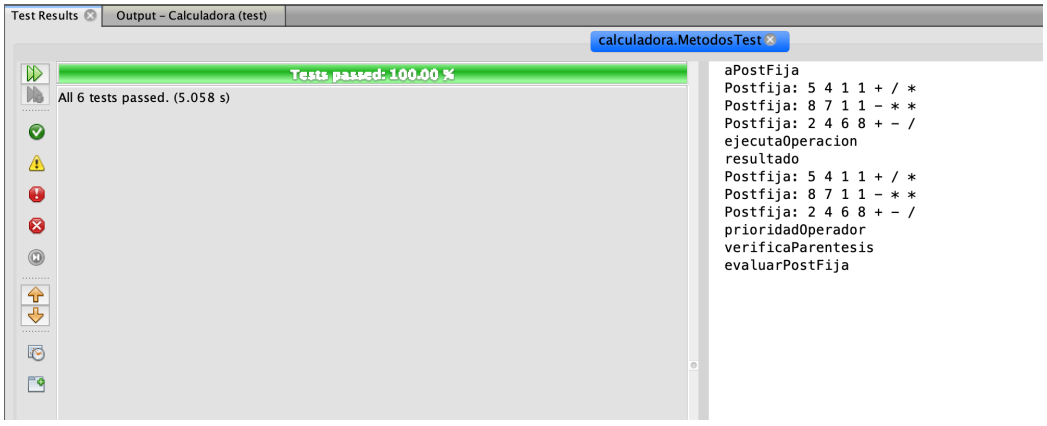
El método recibe como parámetro la expresión en forma de cadena, genera una pila de tipo caracter, y declara dos variables i, y j de tipo entero inicializadas en cero y en la longitud de la cadena, respectivamente.
A través de un ciclo *while,* valida si el primer carácter de la cadena es un paréntesis izquierdo; en caso afirmativo, lo mete en la pila, aumentando la variable i en 1. De lo contrario, verifica si el caracter es un paréntesis derecho; en este caso saca un elemento de la pila (de no ser posible sacar un elemento se lanza la excepción correspondiente), se le asigna el valor del tamaño de la cadena a i y se introduce el paréntesis derecho a la pila.
Cuando $i = j$ el método sale del ciclo y regresa una variable *res* de tipo booleano que verifica si la pila está vacía; si esto se cumple, y *res* es verdadera, entonces los paréntesis están balanceados, lo opuesto ocurre en el caso contrario.

Pantalla de la Calculadora

Pruebas

Para revisar la funcionalidad del código, se realizaron diferentes pruebas utilizando JUnit para verificar la operatividad de los métodos implementados en la clase *Métodos*. En total, se realizaron tres pruebas por cada método.

Resultados de JUnit



Limitaciones

El programa tiene ciertas limitaciones entre las que se encuentran:

- La calculadora no interpreta la sintaxis ()() como multiplicación.
- La calculadora únicamente realiza operaciones aritméticas básicas, no realiza operaciones como el cálculo de potencias y raíces.
- En caso de que aparezca un error en la calculadora, hay que presionar el botón AC para borrar el mensaje de error antes de escribir una nueva expresión.
- La calculadora no tiene memoria, es decir, no se pueden guardar resultados obtenidos para ser utilizados posteriormente.

Posibles mejoras

- Modificar los métodos para que la calculadora permita realizar operaciones más allá de las aritméticas básicas, imitando una calculadora científica.

- Agregar la funcionalidad de ingresar datos directamente desde el teclado, en lugar de utilizar la interfaz proporcionada.

Conclusiones

La realización de este proyecto fue sumamente provechoso para reforzar los conocimientos y habilidades al trabajar con la estructura de Pilas.
Además, la realización de este proyecto representó un reto significativo en la obtención de un programa que cumpliera con los requisitos de manera eficiente y que fuera fácil de utilizar por el usuario promedio.
Adicionalmente, este proyecto nos permitió conocer otras formas de trabajar en equipo, así como adaptarnos a la personalidad de cada uno de los miembros, y a la forma de trabajar de los mismos. Conocer de antemano el rol que cada uno tiende a asumir en los trabajos en equipo nos permitió hacer la tarea de una forma más organizada y delegar de mejor forma las tareas.
Algo muy importante de lo que nos dimos cuenta al realizar este proyecto es la cantidad de esfuerzo y dedicación que implica realizar una calculadora, esto nos dio una visión más amplia de todo lo que conlleva la creación de dispositivos.

Apéndice

Manual de usuario

Para utilizar la calculadora es necesario:

1. Utilizar la plataforma NetBeansIDE, abrir la carpeta *Calculadora,* en donde se encuentran todas las clases.

2. La clase a ejecutar es InterfazCalculadora, la cual despliega la calculadora con los siguientes botones:

   - Botones individuales para cada número del 0 al 9.
   - Botón de $(-)$ para escribir números negativos [2].
   - Botón de punto decimal.
   - Botón $+/-$ para cambiarle el signo a un número tras haberlo escrito.
   - Botones correspondientes a las operaciones aritméticas $(+, -, \times, /)$ y a los paréntesis.

[2]Es necesario dar oprimir el botón del signo antes del número que se quiere poner como negativo.

- Botón AC para borrar toda la expresión.
- Botón C para borrar elemento por elemento.
- Botón de = para resolver la expresión.

3. En caso de que la sintaxis sea incorrecta, o que la operación sea inválida, la calculadora arroja como respuesta el error correspondiente.

Para conocer las limitaciones del programa, consultar el apartado correspondiente.

Código

Nota: Aunque a continuación no se muestra el JavaDoc en los códigos, este sí se encuentra adecuadamente documentado en el código de NetBeans.

Métodos

```
public class Metodos {
    public static String resultado (String expresion) {
        String res = "";
        if(verificaParentesis(expresion))
            res = evaluarPostFija(aPostFija(expresion));
        else
            res = " Syntax Error: Parentesis incorrectos ";
        return res;
    }

    public static String aPostFija (String infija) {
        StringBuilder postfija;
        PilaADT<String> pila;
        Scanner scan;
        String token;
        int prioridad;

        pila = new Pila<>();
        postfija = new StringBuilder();
        scan = new Scanner(infija);

        while(scan.hasNext()) {
```

```
            token = scan.next();
            prioridad = prioridadOperador(token);

            switch (prioridad) {
                case 0: postfija.append(token+" "); //es numero
                        break;

                case 1: pila.push(token); //parentesis abre
                        break;

                case 2: while(!pila.isEmpty() &&
                prioridadOperador((String) pila.peek()) != 1)
                //parentesis cierra
                            postfija.append(pila.pop()+" ");
                        pila.pop();
                        break;
                default:while(!pila.isEmpty() && prioridad <=
                prioridadOperador((String) pila.peek()))//es operador
                            postfija.append(pila.pop() +" ");
                        pila.push(token);

            }
        }
        while(!pila.isEmpty())
            postfija.append(pila.pop()+" ");
        System.out.println("Postfija: "+postfija.toString());
        return postfija.toString();
    }

    public static String evaluarPostFija (String postfija) {
        Pila<Double> pila;
        Scanner scan;
        String token = null, resultadoStr = null;
        double op1, op2, res;

        pila = new Pila<>();
        scan = new Scanner(postfija);
```

```
        try {
            while(scan.hasNext()) {
                token = scan.next();
                if(prioridadOperador(token) == 0)
                    pila.push(Double.parseDouble(token));
                else {
                    op1 = (Double) pila.pop();
                    op2 = (Double) pila.pop();
                    res = ejecutaOperacion(op1, op2, token);
                    pila.push(res);
                }
            }
            res = (Double) pila.pop();
            if(pila.isEmpty())
                resultadoStr = String.valueOf(res);
            else
                resultadoStr = " Faltan operadores! ";
        } catch(EmptyCollectionException ecx) {
            resultadoStr = " Faltan operadores! ";
        } catch(NumberFormatException nfe) {
            resultadoStr = " Syntax error! "+token;
        } catch(ArithmeticException ae) {
            resultadoStr = " Division entre 0! ";
        }

        return resultadoStr;
    }

    public static double ejecutaOperacion(double op2, double op1,
    String token) throws ArithmeticException {
        double res = 0;

        switch (token.charAt(0)) {
            case '+': res = op2 + op1;
                      break;
            case '-': res = op2-op1;
                      break;
            case '/': if(op2 == 0)
```

```
                    throw new ArithmeticException();
                else
                    res = op1/op2;
                break;
            case '*': res = op2*op1;
        }
        return res;
    }

    public static int prioridadOperador (String caracter) {
        int prioridad;

        if (caracter.equals("+") || caracter.equals("-"))
            prioridad = 4;
        else if (caracter.equals("*") || caracter.equals("/"))
            prioridad = 3;
        else if (caracter.equals("("))
            prioridad = 1;
        else if (caracter.equals(")"))
            prioridad = 2;
        else
            prioridad = 0;
        return prioridad;
    }

    public static boolean verificaParentesis (String expresion) {
        boolean res;
        Pila<Character> pila;
        int j, i;

        res = false;
        pila = new Pila<>();
        i = 0;
        j=expresion.length();
        while(i<j){
          if(expresion.charAt(i)=='(')
              pila.push('(');
          else
```

```
            if(expresion.charAt(i)==')'){
                try{
                    pila.pop();
                }catch(Exception e){
                    i=j;
                    pila.push(')');
                }
            }
          i++;
        }
        return pila.isEmpty();
    }
}
```

Main

```
public class Calculadora {

    public static void main(String[] args) {
        InterfazCalculadora.main(args);
    }
```

JUnit

```
public class MetodosTest {

    public MetodosTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }
```

```
@After
public void tearDown() {
}

/**
 * Test of resultado method, of class Metodos.
 */
@Test
public void testResultado() {
    System.out.println("resultado");
    //Primer test
    String infija = "5 * ( 4 / ( 1 + 1 ) )";
    String expResult = "10.0";
    String result = Metodos.resultado(infija);
    assertEquals(expResult, result);

    //Segundo test
    infija = "8 * ( 7 * ( 1 - 1 ) )";
    expResult = "0.0";
    result = Metodos.resultado(infija);
    assertEquals(expResult, result);

    //Tercer test
    infija = "2 / ( 4 - ( 6 + 8 ) )";
    expResult = "-0.2";
    result = Metodos.resultado(infija);
    assertEquals(expResult, result);
}

/**
 * Test of aPostFija method, of class Metodos.
 */
@Test
public void testAPostFija() {
    System.out.println("aPostFija");
    //Primer test
    String infija = "5 * ( 4 / ( 1 + 1 ) )";
```

```
        String expResult = "5 4 1 1 + / * ";
        String result = Metodos.aPostFija(infija);
        assertEquals(expResult, result);

        //Segundo test
        infija = "8 * ( 7 * ( 1 - 1 ) )";
        expResult = "8 7 1 1 - * * ";
        result = Metodos.aPostFija(infija);
        assertEquals(expResult, result);

        //Tercer test
        infija = "2 / ( 4 - ( 6 + 8 ) )";
        expResult = "2 4 6 8 + - / ";
        result = Metodos.aPostFija(infija);
        assertEquals(expResult, result);
    }

    /**
     * Test of evaluarPostFija method, of class Metodos.
     */
    @Test
    public void testEvaluarPostFija() {
        System.out.println("evaluarPostFija");
        //Primer test
        String postfija = "5 4 1 1 + / * ";
        String expResult = "10.0";
        String result = Metodos.evaluarPostFija(postfija);
        assertEquals(expResult, result);

        //Segundo test
        postfija = "8 7 1 1 - * * ";
        expResult = "0.0";
        result = Metodos.evaluarPostFija(postfija);
        assertEquals(expResult, result);

        //Tercer test
        postfija = "2 4 6 8 + - / ";
        expResult = "-0.2";
```

```
        result = Metodos.evaluarPostFija(postfija);
        assertEquals(expResult, result);
    }

    /**
     * Test of ejecutaOperacion method, of class Metodos.
     */
    @Test
    public void testEjecutaOperacion() {
        System.out.println("ejecutaOperacion");
        //Primer test
        double op1 = 5.0;
        double op2 = 13.0;
        String token = "-";
        double expResult = 8.0;
        double result = Metodos.ejecutaOperacion(op1, op2, token);
        assertEquals(expResult, result, 0.0);

        //Segundo test
        op1 = 5.0;
        op2 = 10.0;
        token = "/";
        expResult = 2.0;
        result = Metodos.ejecutaOperacion(op1, op2, token);
        assertEquals(expResult, result, 0.0);

        //Tercer test
        op1 = 6.0;
        op2 = 7.0;
        token = "*";
        expResult = 42.0;
        result = Metodos.ejecutaOperacion(op1, op2, token);
        assertEquals(expResult, result, 0.0);
    }

    /**
     * Test of prioridadOperador method, of class Metodos.
     */
```

```
@Test
public void testPrioridadOperador() {
    System.out.println("prioridadOperador");
    //Primer test
    String token = "(";
    int expResult = 1;
    int result = Metodos.prioridadOperador(token);
    assertEquals(expResult, result);

    //Segundo test
    token = "58.96";
    expResult = 0;
    result = Metodos.prioridadOperador(token);
    assertEquals(expResult, result);

    //Tercer test
    token = "*";
    expResult = 3;
    result = Metodos.prioridadOperador(token);
    assertEquals(expResult, result);
}

/**
 * Test of verificaParentesis method, of class Metodos.
 */
@Test
public void testVerificaParentesis() {
    System.out.println("verificaParentesis");
    //Primer test
    String expresion = ")85(";
    boolean expResult = false;
    boolean result = Metodos.verificaParentesis(expresion);
    assertEquals(expResult, result);

    //Segundo test
    expresion = "(8*(5+3))";
    expResult = true;
    result = Metodos.verificaParentesis(expresion);
```

```
        assertEquals(expResult, result);

        //Tercer test
        expresion = "(85*(-2)";
        expResult = false;
        result = Metodos.verificaParentesis(expresion);
        assertEquals(expResult, result);
    }

}

}
```