**Objetivos:**

* El objetivo principal del proyecto es crear una “Calculadora” funcional, que pueda realizar las operaciones de sumas y productos, así como la operación de la potencia. El proyecto debe ser realizado en tiempo y forma y entregado el miércoles 21 de Septiembre de 2022. Se deberá entregar una interfaz gráfica que simule una calculadora y tenga el correcto funcionamiento de una calculadora, como tal, deberá poder evaluar número negativos, números con punto decimal, números racionales e irracionales. El proyecto se realizará en el IDE de “Apache NetBeans”.

**Solución Diseñada:**

* Los principales problemas que nos encontramos para el correcto funcionamiento de la Calculadora son 3:

1. Revisar la sintaxis de la expresión, para anticiparnos a cualquier error de escritura y la calculadora pueda evaluar expresiones verdaderas y reales.
2. Convertir la expresión ya revisada con anterioridad a su forma de Postfija para que la Calculadora pueda evaluar correcta y adecuadamente, teniendo en cuenta el orden de jerarquía de los operadores, paréntesis, corchetes, llaves.
3. Dada la expresión en su forma Postfija, poder evaluar la expresión de una manera adecuada y arrojando el resultado esperado.

* Ahora, ¿Por qué es preferible convertir de notación Infija a notación Postfija?
  + “Las expresiones postfijas son, generalmente, más fáciles de evaluar que las expresiones infijas, porque no es necesario tener en cuenta ni reglas de precedencia, ni paréntesis. El orden de los valores y de los operadores en la expresión es suficiente para determinar el resultado. A menudo, los compiladores de los lenguajes de programación en los entornos de ejecución utilizan expresiones postfijas en sus cálculos internos, por esa misma razón.

El proceso de evaluación de una expresión postfija puede enunciarse mediante una única regla simple: analizando la expresión de izquierda a derecha, hay que aplicar cada operación a los dos operandos inmediatamente precedentes y sustituir el operador por el resultado. Al final, lo que nos quedará será el resultado total de la expresión.” – John Lewis.

* Para resolver los problemas anteriormente mencionados, necesitamos emplear tres códigos que solucionen estos problemas, y hacer uso de la estructura abstracta “Pilas” en ellos para que sea un código eficiente y bien estructurado.
* Por esto hemos implementado estos tres códigos y a continuación se presentan sus UML de las clases:
  + Comenzamos con las clases UML de las estructuras abstractas “Pila” que vamos a utilizar.

|  |
| --- |
| <<interface>>  PilaADT |
| +pop(): T  +push (T dato): void  +peek(): T  +isEmpty(): boolean |

|  |
| --- |
| PilaA |
| -pila[]: T  -tope: int |
| +pop(): T  +push (T dato): void  +peek(): T  +isEmpty(): boolean |

* Ahora mostraremos las clases UML de los tres métodos principales para lograr el correcto funcionamiento de nuestra Calculadora:

|  |
| --- |
| EvaluacionSintaxis |
| +EvaluacionSintaxis()  +parentesisBalanceados(String expresion): boolean  +esOperador(Character c): boolean  +noDobleOperador(String expresion):boolean  +noDobleDecimal(String expresion): boolean  +noParentesisSinOperador(String expresion): boolean  +expresionValida(String expresion): boolean |

|  |
| --- |
| Postfija |
| +Postfija()  +conviertePostfija(String expresion): String |

|  |
| --- |
| EvaluacionPostfija |
| +EvaluacionPostfija()  +evaluaPost(String expresion): double  +esOperador(Character c):boolean  +esParentesis(Character c): boolean |

* Ya que tenemos las clases en UML de nuestros métodos, es preciso también incluir las clases en UML de nuestra interfaz gráfica:

|  |
| --- |
| CalculadoraDemo |
|  |
| +main() |

|  |
| --- |
| CalculadoraGUI |
| +keyPanel  +textPanel  +equalPanel  +0-9button  +inputTF  +CalculadoraGUI()  +display()  +ClearListener()  +KeyListener()  +EqualListener() |

**Pruebas:**

* Se realizaron diversas pruebas para confirmar el correcto funcionamiento de nuestros códigos y nuestra GUI.
* Algunas de las pruebas que se realizaron fueron:
* Expresión con caracteres que no son dígitos
* Expresión con errores de sintaxis (balanceo de paréntesis, letras y números, expresiones erróneas.)
* Expresión con operaciones matemáticas inválidas (dividir entre cero, elevar el 0 con 0)
* Expresión con positivos y negativos
* Expresión con todos los operadores
* Expresión con diversas jerarquías de operaciones

**Limitaciones:**

* La Calculadora no está desarrollada cómo para revolver expresiones matemáticas complejas por ejemplo:
  + Calculo de funciones
  + Calculo de derivadas o integrales
  + Calculo de raíz cuadrada
  + Calculo de raíz cubica
  + Calculo de funciones trigonométricas
  + Generar estadísticas
  + Exhibir gráficos

**Mejoras y Conclusiones:**

* Sin duda alguna hemos logrado un excelente trabajo, la calculadora funciona perfectamente y posee todas las capacidades que teníamos previstas para ella, realiza las operaciones eficazmente y arroja el resultado esperado
* ¿Qué podemos mejorar en la calculadora?
  + A pesar de que logramos el correcto funcionamiento de una calculadora con expresiones matemáticas básicas, se puede mejorar el grado de complejidad que la calculadora puede adoptar, como sacar raíces, resolver ecuaciones, resolver funciones trigonométricas, sin embargo, este tipo de operaciones planeamos implementarlas en un futuro, con el adecuado conocimiento para implementarlo.
* Nuestra experiencia en general fue buena, como cualquier proyecto individual o en equipo tiene su grado de complejidad y de trabajo, logramos identificar las habilidades de cada uno de nosotros, y establecimos roles de trabajo para reunir con más rapidez y eficiencia el trabajo de cada uno.

Siempre es un reto colaborar con personas nuevas, cada una tiene ideas, conocimientos y habilidades muy valiosos que al fusionarlos se logra completar el objetivo dispuesto. En cada línea de código se ve demostrada el conocimiento y sobretodo el conocimiento aplicado de las diversas estructuras de datos, incluida la “Pila”. Es preciso afirmar que nuestro proyecto quedó concluido de la mejor manera y establecimos nuevas relaciones de amistad y laborales para colaborar en el futuro.

**REFERENCIAS:**

* [Guardati Buemo Silvia del Carmen](javascript:open_window(%22https://hammurabi.itam.mx:443/F/S77FIJ46188NV57YVS18JNLHCVB5L5JDQGU4P43FNTNQEIE9Q3-94115?func=service&doc_number=000264283&line_number=0009&service_type=TAG%22);), 2015, *Estructuras de datos básicas: programación orientada a objetos con Java*, 1ª ed., México, D. F., Alfaomega Grupo Editor.
* John Lewis, Joseph Chase, 2006, *Estructuras de datos con Java: diseño de estructuras y algoritmos,* 2a ed., Madrid, Pearson Addison Wesley.

**Apéndice con Código:**

1. Evalúa sintaxis

|  |
| --- |
| package calculadora; |
|  |  | |
|  | /\*\* | |
|  | \* | |
|  | \* @author jeronimo | |
|  | \*/ | |
|  | public class EvaluacionSintaxis { | |
|  | /\*\* | |
|  | \* Determina si una expresion matematica tiene los parametros balanceados | |
|  | \* | |
|  | \* @param formula String de la expresion que queremos evaluar | |
|  | \* @return <ul> | |
|  | \* <li>true: si los parentesis estan balanceados | |
|  | \* <li>false: si los parentesis no estan balanceados | |
|  | \* </ul> | |
|  | \*/ | |
|  | public static boolean parentesisBalanceados(String formula){ | |
|  | boolean res=true; | |
|  | Character c; | |
|  | PilaA<Character> pila= new PilaA(); | |
|  | int i=0; | |
|  | while(i<formula.length() && res){ | |
|  | c= formula.charAt(i); | |
|  | if(c=='('){ | |
|  | pila.push(c); | |
|  | } | |
|  | else{ | |
|  | if(c==')'){ | |
|  | if(pila.isEmpty()){ | |
|  | res=false; | |
|  | } | |
|  | else{ | |
|  | pila.pop(); | |
|  | } | |
|  | } | |
|  | } | |
|  | i++; | |
|  | } | |
|  | return res; | |
|  | } | |
|  | /\*\* | |
|  | \* Metodo que nos permite determinar si un character es un operador ('+','/','-','\*') | |
|  | \* | |
|  | \* @param c el caracter que queremos evaluar | |
|  | \* @return <ul> | |
|  | \*<li>true: si el character es '+','/','-','\*'; | |
|  | \*<li>false: si el character no es '+','/','-','\*'; | |
|  | \*</ul> | |
|  | \*/ | |
|  | public static boolean esOperador(char c){ | |
|  | boolean res= false; | |
|  | if (c=='\*' || c=='/' || c=='+' || c=='-'){ | |
|  | res= true; | |
|  | } | |
|  | return res; | |
|  | } | |
|  | /\*\* | |
|  | \* Metodo para revisar que no existan dos operados juntos en una expresion | |
|  | \* | |
|  | \* @param formula String que representa la expresion que queremos evaluar | |
|  | \* @return <ul> | |
|  | \* <li>true: si no hay dos operadores juntos en la expresion | |
|  | \* <li>flase: si hay dos operados juntos en la expresion | |
|  | \* </ul> | |
|  | \*/ | |
|  | public static boolean noDobleOperador(String formula){ | |
|  | boolean res= true; | |
|  | Character c; | |
|  | Character c2; | |
|  | //nunca debe llegar a .length ya que checamos dos valores uno mayor al contador | |
|  | int i=0; | |
|  | while(i<formula.length()-1 && res){ | |
|  | c= formula.charAt(i); | |
|  | c2= formula.charAt(i+1); | |
|  | if(esOperador(c) && esOperador(c2)){ | |
|  | res= false; | |
|  | } | |
|  | i++; | |
|  | } | |
|  | return res; | |
|  | } | |
|  | /\*\* | |
|  | \* Metodo para revisar que no existan dobles decimales | |
|  | \* | |
|  | \* @param formula String que representa la expresion que queremos evaluar | |
|  | \* @return <ul> | |
|  | \* <li>true: si no hay doble decimal | |
|  | \* <li>flase: si hay doble decimal | |
|  | \* </ul> | |
|  | \*/ | |
|  | public static boolean noDobleDecimal(String formula){ | |
|  | boolean res= true; | |
|  | Character c; | |
|  | PilaA<Character> pila= new PilaA(); | |
|  | int i=0; | |
|  | while(i<formula.length() && res){ | |
|  | c= formula.charAt(i); | |
|  | if(c=='.'){ | |
|  | if(pila.isEmpty()){ | |
|  | pila.push(c); | |
|  | } | |
|  | else{ | |
|  | res=false; | |
|  | } | |
|  | } | |
|  | else{ | |
|  | if(esOperador(c)&& !pila.isEmpty()){ | |
|  | pila.pop(); | |
|  | } | |
|  | } | |
|  | i++; | |
|  | } | |
|  | return res; | |
|  | } | |
|  | /\*\* | |
|  | \* Metodo para revisar que no existan parentesis sin operador | |
|  | \* | |
|  | \* @param formula String que representa la expresion que queremos evaluar | |
|  | \* @return <ul> | |
|  | \* <li>true: si no hay parentesis sin operador | |
|  | \* <li>flase: si hay parentesis sin operador | |
|  | \* </ul> | |
|  | \*/ | |
|  | public static boolean noOperadorAntesDeParentesis(String formula){ | |
|  | boolean res= true; | |
|  | Character c; | |
|  | Character c2; | |
|  | //empezamos en 1 ya que queremos revisar el del contador y el anteriro | |
|  | int i=1; | |
|  | while(i<formula.length() && res){ | |
|  | c= formula.charAt(i); | |
|  | c2= formula.charAt(i-1); | |
|  | if(c=='(' && !esOperador(c2)){ | |
|  | res= false; | |
|  | } | |
|  | i++; | |
|  | } | |
|  | return res; | |
|  | } | |
|  | /\*\* | |
|  | \* Metodo para evaluar que la expresion este bien escrita | |
|  | \* | |
|  | \* @param formula String que representa la expresion que queremos evaluar | |
|  | \* @return <ul> | |
|  | \* <li>true: si esta bien escrita | |
|  | \* <li>flase: si no esta bien escrita | |
|  | \* </ul> | |
|  | \*/ | |
|  | public static boolean expresionValida(String formula){ | |
|  | boolean res= false; | |
|  | if(parentesisBalanceados(formula)&&noDobleOperador(formula)&&noDobleDecimal(formula)&&noOperadorAntesDeParentesis(formula)){ | |
|  | res=true; | |
|  | } | |
|  | return res; | |
|  | }   1. Convierte Infija a Postfija | |
|  |  | |
| package calculadora; | |
|  | |  |
|  | | import java.util.HashMap; |
|  | | import pilas.PilaA; |
|  | | import pilas.PilaADT; |
|  | |  |
|  | | /\*\* |
|  | | \* Definición de clase Postfijo que convierte expresión en postfija |
|  | | \* @author m-gla |
|  | | \*/ |
|  | | public class Posfijo { |
|  | | /\*\* |
|  | | \* Método que regresa cadena de caracteres en postfija |
|  | | \* @param expresion |
|  | | \* @return String |
|  | | \*/ |
|  | | public String cadena(String expresion){ |
|  | | String posfija = ""; |
|  | | PilaADT <Character> pila = new PilaA(); |
|  | | HashMap <Character, Integer> operadores = new HashMap <>(); |
|  | | operadores.put('+',1); |
|  | | operadores.put('-', 1); |
|  | | operadores.put('\*',2); |
|  | | operadores.put('/', 2); |
|  | | operadores.put('^', 3); |
|  | |  |
|  | | int i = 0; |
|  | | Character c; |
|  | |  |
|  | | while(i < expresion.length()){ |
|  | | c = expresion.charAt(i); |
|  | | if((i == 0 && expresion.charAt(i) == '-') || Character.isDigit(c) || c == '.' || (c == '-' && !Character.isDigit(expresion.charAt(i-1)))){ |
|  | | posfija += '('; |
|  | | posfija += c; |
|  | | int j = i + 1; |
|  | | while(j < expresion.length() && (Character.isDigit(expresion.charAt(j)) || expresion.charAt(j) == '.')){ |
|  | | posfija += expresion.charAt(j); |
|  | | j++; |
|  | | }posfija += ')'; |
|  | | i = j - 1; |
|  | | } |
|  | | else if(!pila.isEmpty()){ |
|  | | if(c == '('){ |
|  | | pila.push(c); |
|  | | }else if(c == ')'){ |
|  | | while(pila.peek() != '('){ |
|  | | posfija += pila.pop(); |
|  | | } |
|  | | pila.pop(); |
|  | | }else if(pila.peek() == '(' || operadores.get(c) > operadores.get(pila.peek())){ |
|  | | pila.push(c); |
|  | | }else if(operadores.get(c) <= operadores.get(pila.peek())){ |
|  | | posfija+= pila.pop(); |
|  | | pila.push(c); |
|  | | } |
|  | | }else{ |
|  | | pila.push(c); |
|  | | }i++; |
|  | | }while(!pila.isEmpty()){ |
|  | | posfija += pila.pop(); |
|  | | } |
|  | | return posfija; |
|  | | } |
|  | |  |
|  | | } |

|  |
| --- |
| 1. Evalúa la expresión Postfija   package calculadora; |
|  |  |
|  | import pilas.PilaA; |
|  | import pilas.PilaADT; |
|  |  |
|  | /\*\* |
|  | \* |
|  | \* @author Ernesto Palma |
|  | \* \* Definiciòn de la clase Evaluación de Postfija que usa pilas |
|  | \* @param <T> |
|  | \*/ |
|  | public class EvaluacionPostfija <T>{ |
|  |  |
|  | /\*\* |
|  | \* Constructor por omisión |
|  | \*/ |
|  | public EvaluacionPostfija(){ |
|  |  |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Método que regresa el resultado de une expresión en postfija usando pilas |
|  | \* @param cadena |
|  | \* @return double |
|  | \*/ |
|  | public double evaluaPost(String cadena){ |
|  | PilaADT<Double> pila=new PilaA(); |
|  | int i=0; |
|  | double resul, numer; |
|  | while(i<cadena.length()){ |
|  | System.out.println("Valor de i en inicio: "+i); |
|  | if(this.operador(cadena.charAt(i))){ |
|  | numer=pila.pop(); |
|  | switch(cadena.charAt(i)){ // es más fácil usar casos que condicional if |
|  | case'+': resul=pila.pop() + numer; |
|  | pila.push(resul); |
|  | break; |
|  | case'-': resul=pila.pop() - numer; |
|  | pila.push(resul); |
|  | break; |
|  | case'\*': resul=pila.pop() \* numer; |
|  | pila.push(resul); |
|  | break; |
|  | case'/': resul=pila.pop() / numer; |
|  | pila.push(resul); |
|  | break; |
|  | case'^': resul=Math.pow(pila.pop(), numer); |
|  | pila.push(resul); |
|  | break; |
|  | } |
|  | } |
|  | else if(this.parentesis(cadena.charAt(i))){ |
|  | StringBuilder sb=new StringBuilder(); |
|  | int cont=0; |
|  | while(!this.parentesis(cadena.charAt(i+1))){ |
|  | sb.append(cadena.charAt(i+1)); |
|  | i++; |
|  | cont++; |
|  | } |
|  | pila.push(Double.parseDouble(sb.toString())); |
|  | i++; |
|  | System.out.println("valor de i"+i); |
|  | } |
|  | else |
|  | pila.push(Double.parseDouble(cadena.substring(i, i+1))); |
|  | i++; |
|  | } |
|  | return pila.peek(); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Método que indica si un caracter de una cadena de texto es un operador |
|  | \* @param cadena |
|  | \* @return boolean |
|  | \*/ |
|  | public boolean operador(char cadena){ |
|  | boolean resp=false; |
|  | if(cadena == '+' | cadena == '-' | cadena == '\*' | cadena == '/' | cadena == '^') |
|  | resp=true; |
|  | return resp; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Método que indica si un caracter de una cadena de texto es un paréntesis |
|  | \* @param cadena |
|  | \* @return |
|  | \*/ |
|  | public boolean parentesis(char cadena){ |
|  | boolean resp=false; |
|  | if(cadena == '(' | cadena== ')') |
|  | resp = true; |
|  | return resp; |
|  | } |
|  |  |
|  | } |