Ulises Bojórquez Ortiz

PROYECTO SEGUNDO PARCIAL: ANALIZADOR SINTÁCTICO DE ARITMÉTICA SIMPLE

Materia: Matemáticas Computacionales



A01114716

**Índice**

Introducción ……………………………………………………………………………….3

Propósito …………………………………………………………………………………..3

Desarrollo …………………………………………………………………………….……3

Diagrama de flujo …………………………………………………………………3

Diagrama de clases ………………………………………………………………4

Descripción de funciones ………………………………………………………..5

Algoritmos …………………………………………………………………………7

Código ……………………………………………………………………………..8

Pruebas y resultados ……………………………………………………………21

Conclusiones …………………………………………………………………………….23

Referencias ………………………………………………………………………………23

**Introducción**

Este es un programa desarrollado como parte del proyecto de segundo parcial de la materia “Matemáticas Computacionales”, el cual es un analizador sintáctico de cadenas aritméticas que dada una gramática que acepte todas las combinaciones validas de aritmética. En dado caso de ser aceptada la operación de entrada mostrará su árbol de derivación. Los símbolos aceptados son: dígitos, “+”, “-“, “\*”, “(“, “)”, “^”, “/”.

**Propósito**

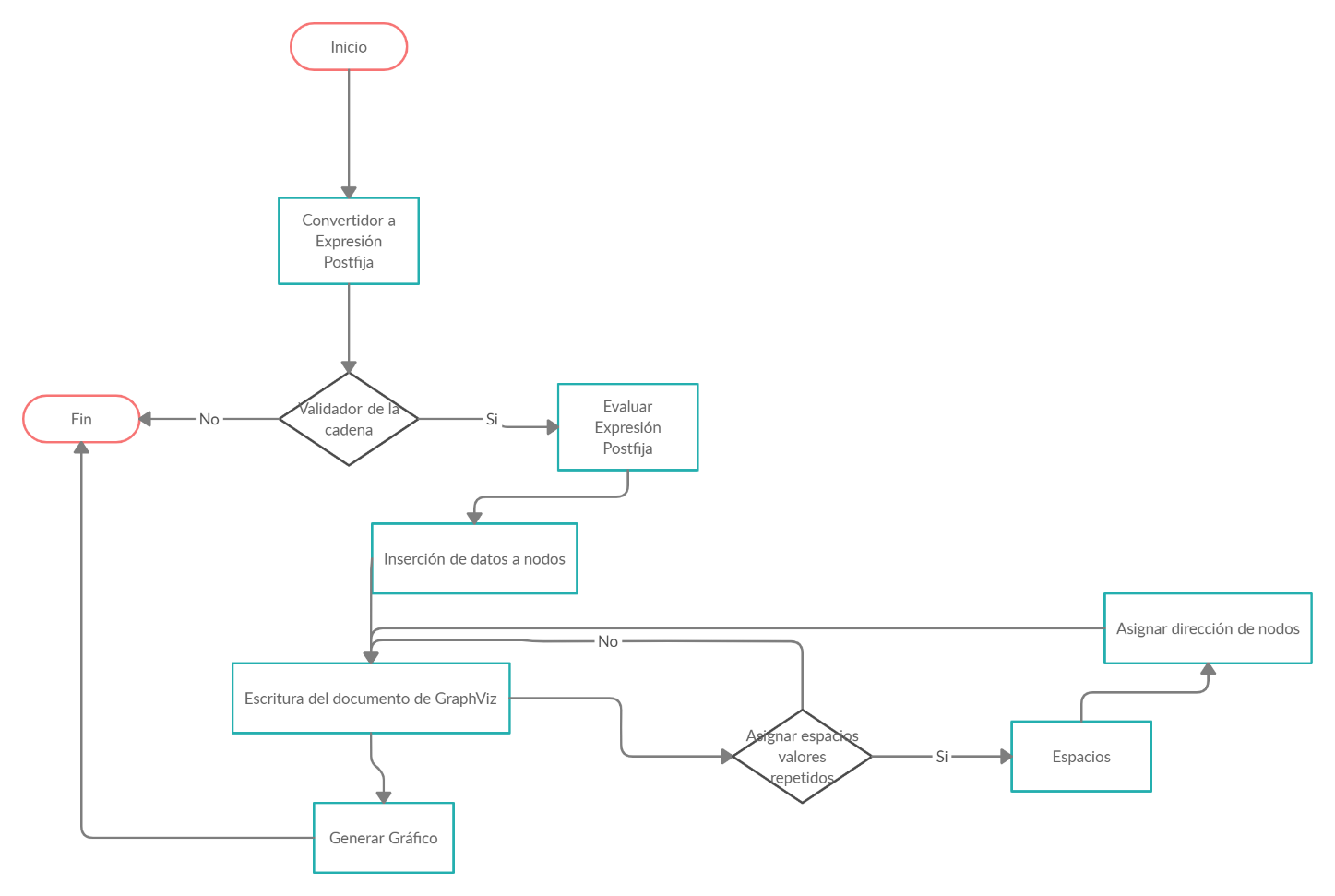
El propósito de este documento es mostrar de una forma más detallada todo el código, la lógica y aplicaciones del programa, se mostrarán los diagramas de flujo de programa, así como sus diagramas de clases para ver de una forma más sencilla y explicita el código, además que se va a describir detalladamente lo que hace cada una de sus funciones, para que se usa y todos sus respectivos detalles. Se adjuntarán imágenes y diagramas para ver cada proceso y su respectivo producto final en dado caso que así sea.

**Desarrollo**

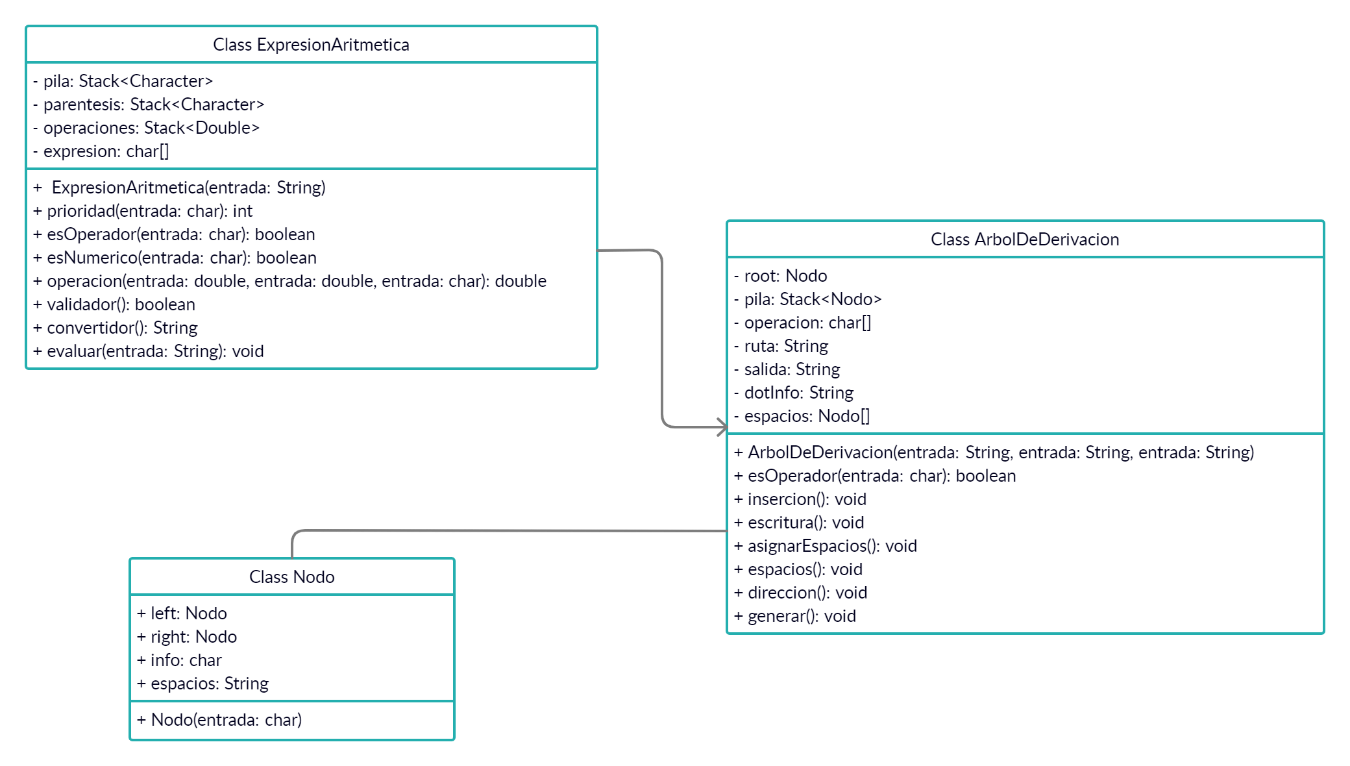
*Diagrama de flujo*

El diagrama de flujo muestra un poco cual es el proceso que se sigue dentro del código realizado, que hace si cumple con una de las condiciones o no, de forma más detalla este diagrama de flujo quiere decir lo siguiente:

Se recibe una entrada de texto la cual se va a convertir de infija a postfija, después se validará si es parte de la gramática, si no es entonces el programa acaba, por otro lado, si es aceptado evaluará esa operación aritmética dada y dará su resultado. A continuación, se inserta cada carácter en nodos y se construye el árbol de derivación usando un algoritmo, después se empieza a realizar la escritura del documento para generar el GraphViz, si es que hay valores repetidos dentro del árbol se asignan espacios dentro de la información de los nodos, ya que se hayan asignado los espacios se generara el árbol de derivación de forma gráfica.



*Diagrama de clases*



*Descripción de funciones*

**Class ExpresionAritmetica**

* **ExpresionAritmetica(entrada: String)**

Este es el constructor de la función en el que recibe un String que es la expresión aritmética, dentro de se inicializa las 3 pilas y separamos la cadena en un arreglo.

* **prioridad(entrada: char): int**

Se le una entrada de tipo char y te regresa la prioridad del operador de tipo int.

* **esOperador(entrada: char): boolean**

Recibe de entrada un tipo char y si la entrada es un operador regresa true, en caso contrario regresa false.

* **esNumerico(entrada: char) boolean**

Recibe de entrada un tipo char y si la entrada es un numero regresa true, en caso contrario regresa false.

* **operacion(entrada: double, entrada: double, entrada: char): double**

Recibe tres entradas, las dos primeras entradas son de números tipo double y la última es un operador de tipo char, este función realiza la operación y regresa el resultado de tipo double.

* **validador(): boolean**

Este es un validador que verifica si la expresión aritmética está en la forma de la gramática correcta, en dado caso que la entrada si forme parte de la gramática regresa true, en caso contrario regresa false.

* **convertidor(): String**

Este es la función que evalúa la expresión aritmética y la convierte de infijo a postfijo una vez que se haya validado que la gramática esta correcta y regresa un String con la expresión aritmética en postfijo.

* **evaluar(entrada: String): void**

Esta función recibe una entrada de tipo String que es la expresión aritmética en postfijo, realiza las operaciones para obtener el resultado de esa operación e imprime el resultado.

**Class ArbolDeDerivacion**

* **ArbolDeDerivacion(entrada: String, entrada: String, entrada: String)**

Este es el constructor de la clase, la cual recibe de entrada 3 Strings, el primero es la expresión postfija, el segundo es la ruta del archivo de texto, nombre de salida del grafo. Se asgina el root a NULL, se crea la pila, se agrega la expresión postfija a un arreglo, se inicializa un String que es una variable que guarda información y se crea un arreglo de Nodos.

* **esOperador(entrada: char): boolean**
* Recibe de entrada un tipo char y si la entrada es un operador regresa true, en caso contrario regresa false.
* **insercion(): void**

Lo que hace esta función es crear e insertan cada carácter de la expresión postfija un nodo y guardarlo en el arreglo de Nodos para posteriormente usarlo en otro función. Además, los agrega a la pila y empieza a construir su árbol de derivación. Al final hace un pop y asigna el root al nodo que se le hizo ese pop. Dentro de esta función llamada a las funciones escritura() y generar().

* **escritura(): void**

Lo que hace esta función es que empieza a escribir el documento que se le pasara al GraphViz para crearnos una imagen del árbol de derivación, si en dado caso que el archivo no exista, se crea, y se llama a la función asignarEspacio() para que genere los espacios en el documente txt.

* **asignarEspacios(): void**

Dado que con GraphViz hay un problema si se encuentran valores repetidos, para hacerlos únicos y no apunten a ese mismo nodo se les asigna al Nodo información de su espacio cada vez que se encuentre un valor igual dentro de la entrada postfija. Después, llama a la función espacios() y a dirección().

* **espacios(): void**

Esta función asigna la información de los espacios de los Nodos al árbol principal que se mostrará en el GraphViz.

* **direccion(): void**

Lo que hace esta función es construir los apuntadores de los nodos, los guarda en la variable dotInfo y la agrega en el documento que se le pasara a GraphViz.

* **generar(): void**

Esta función genera el comando desde Java para poder llamar a la aplicación de GraphViz y que genere el árbol en forma de imagen con el formato jpg.

**Class Nodo**

* **Nodo(entrada: char)**

Este es el constructor de la clase la cual recibe una entrada tipo char que es la información que tiene dentro el Nodo, se asigna su Nodo derecho e izquierdo a NULL y su espacio a ninguno (“”).

*Algoritmos*

Se uso el algoritmo de este video <https://www.youtube.com/watch?v=5zhXm3J9xdE> para convertir de una expresión infija a postfija. Su funcionamiento es el siguiente:

Prioridad o precedencia de operadores:

1. (, )
2. ^
3. \*, /
4. +, -

Reglas:

* Operador = precedencia -> Se cambia
* Operador > precedencia -> Se agrega a la pila
* Operador < precedencia -> Saca operadores
* Operador = Paréntesis derecho “)” -> vaciar pila

Pasos:

* Si se encuentra un “(“ se agrega a la pila.
* Si se encuentra un “)” se saca todo de la pila hasta encontrar un “(” y se agrega a la expresión postfija final.
* Si se encuentra un número se agrega a la expresión postfija final.
* Si se encuentra un operador y el tope de la pila es un “(” o está vacía, se agrega a la pila.
* Si se encuentra un operador y el tope de la pila es otro operador, se aplican reglas.
* Si el operador es de igual orden al que esta al tope de la pila, saca el operador dentro de la pila y lo agrega a la expresión postfija final, mientras que el operador que se leyó se agrega a la pila.
* Si el operador es de mayor orden al que esta al tope de la pila, se agrega a la pila.
* Si el operador es de menor orden al que esta al tope de la pila, saca todos los operadores de la pila y las agrega a la expresión postfija final, y mete ese operador a la pila.
* Cuando se haya leído toda la expresión aritmética de entrada, se vacía la pila y se agrega a la expresión postfija final.

Se uso el algoritmo de este video <https://www.youtube.com/watch?v=WHs-wSo33MM&t=313s> para convertir de una expresión postfija a su árbol de derivación. Su funcionamiento es el siguiente:

* Se recibe una expresión postfija de entrada de texto.
* (Se repite) Se empieza a recorrer la entrada y si el carácter que leyó NO es un operador, crea un nodo y lo mete a una pila.
* (Se repite) Si el carácter que lee es un operador, crea un nodo, saca dos elementos (Nodos) al tope de la pila y asigna el primer elemento que saco de la pila como si hijo derecho y el segundo como su hijo izquierdo, y mete ese nodo a la pila.
* Cuando se haya leído la expresión completa, solamente se saca el elemento al tope de la pila que es el nodo root del árbol de derivación que se generó.

*Código*

**Class ExpresionArtimetica**

/\*

Nombre: Ulises Bojorquez Ortiz

Matricula: A01114716

Materia: Matematicas Computacionales

PROYECTO SEGUNDO PARCIAL: ANALIZADOR SINTACTICO DE ARITMETICA SIMPLE

\*/

**import** java.util.Stack;

**public** **class** ExpresionAritmetica {

**private** Stack<Character> pila, parentesis;

**private** Stack<Double> operaciones;

**private** **char**[] expresion;

**public** ExpresionAritmetica(String ea) {

**this**.expresion=ea.toCharArray();

**this**.pila=**new** Stack<>();

**this**.operaciones=**new** Stack<>();

**this**.parentesis=**new** Stack<>();

}

//Maneja la prioridad de operadores y regresa un entero que representa su prioridad de mayor a menor

**public** **int** prioridad(**char** operador) {

**if**(operador=='(') {

**return** 4;

}**else** **if**(operador=='^') {

**return** 3;

}**else** **if**(operador=='\*' || operador=='/') {

**return** 2;

}**else** **if**(operador=='+' || operador=='-') {

**return** 1;

}**else** {

**return** 0;

}

}

//Verifica si el caracter es operador

**public** **boolean** esOperador(**char** caracter) {

**if**(caracter=='(' || caracter=='\*' || caracter=='+' || caracter=='-' || caracter==')' || caracter=='/' || caracter=='^') {

**return** **true**;

}**else** {

**return** **false**;

}

}

//Verifica si el caracter es numero o no

**public** **boolean** esNumerico(**char** caracter) {

String strNum=String.*valueOf*(caracter);

**if** (strNum == **null**) {

**return** **false**;

}

**try** {

**double** d = Double.*parseDouble*(strNum);

} **catch** (NumberFormatException nfe) {

**return** **false**;

}

**return** **true**;

}

//Realiza las operaciones

**public** **double** operacion(**double** num1, **double** num2, **char** operador) {

**if**(operador=='\*') {

**return** num2\*num1;

}**else** **if**(operador=='/') {

**return** num2/num1;

}**else** **if**(operador=='^') {

**return** Math.*pow*(num2, num1);

}**else** **if**(operador=='+') {

**return** num2+num1;

}**else**{

**return** num2-num1;

}

}

//Valida toda la cadena es parte de la gramatica o no

**public** **boolean** validador() {

//Validar si tiene o no tiene ningun elemento

**boolean** resNoVacio=**true**;

**try** {

**this**.expresion[0]=**this**.expresion[0];

}**catch**(ArrayIndexOutOfBoundsException ex) {

resNoVacio=**false**;

}

**if**(resNoVacio) {

System.***out***.println("Lo cadena NO esta vacia");

}**else** {

System.***out***.println("Lo cadena esta vacia");

}

//Validador primer elemento

**boolean** resPrimer=**true**;

**try** {

**if**(**this**.expresion[0]=='^' || **this**.expresion[0]=='\*' || **this**.expresion[0]=='/' || **this**.expresion[0]=='-' || **this**.expresion[0]=='+') {

resPrimer=**false**;

}

}**catch**(ArrayIndexOutOfBoundsException ex) {

}

**if**(resPrimer) {

System.***out***.println("No inicio con un operador");

}**else** {

System.***out***.println("Inicio con un operador");

}

//Validar operadores y parentesis

**boolean** resOperadores=**true**;

**for**(**int** i=0;i<**this**.expresion.length;i++) {

//+,-,\*/,^ y si es solo un operador

**if**(**this**.expresion.length==1 && esOperador(**this**.expresion[i])) {

resOperadores=**false**;

**break**;

// \*),+), /), -), ^)

}**else** **if**((**this**.expresion[i]=='\*' || **this**.expresion[i]=='-' || **this**.expresion[i]=='+' || **this**.expresion[i]=='/' || **this**.expresion[i]=='^')

&& **this**.expresion[i+1]==')') {

resOperadores=**false**;

**break**;

//()() o () si tiene este tipo de formato no es valido

}**else** **if**((i+1<**this**.expresion.length && **this**.expresion[i]==')' && **this**.expresion[i+1]=='(') ||

(**this**.expresion[i]=='(' && **this**.expresion[i+1]==')')) {

resOperadores=**false**;

System.***out***.println("parentesis mal colocados");

**break**;

//\*-, +/, -\*, etc Operadores seguidos menos parentesis

}**else** **if**((**this**.expresion[i]=='\*' || **this**.expresion[i]=='-' || **this**.expresion[i]=='+' || **this**.expresion[i]=='/' || **this**.expresion[i]=='^')

&& (**this**.expresion[i+1]=='\*' || **this**.expresion[i+1]=='-' || **this**.expresion[i+1]=='+' || **this**.expresion[i+1]=='/' || **this**.expresion[i+1]=='^')) {

resOperadores=**false**;

System.***out***.println("ente operadores seguidos");

**break**;

}

}

**if**(resOperadores) {

System.***out***.println("Tienes los operadores correctamente colocados");

}**else** {

System.***out***.println("NO Tienes los operadores correctamente colocados");

}

//Validador de aritmetica correcta

**boolean** resAritmetica=**true**;

**for**(**int** i=0;i<**this**.expresion.length;i++) {

**if**(esOperador(**this**.expresion[i]) || esNumerico(**this**.expresion[i])) {

resAritmetica=**true**;

}**else** {

resAritmetica=**false**;

**break**;

}

}

**if**(resAritmetica) {

System.***out***.println("Tiene aritmetica valida");

}**else** {

System.***out***.println("NO Tiene aritmetica valida");

}

//Validador de parentesis bien ordenados

**boolean** resParentesis, flag=**true**;

**for**(**int** i=0;i<**this**.expresion.length;i++) {

**if**(**this**.expresion[i]=='(' || **this**.expresion[i]==')') {

**if**(**this**.expresion[i]=='(') {

**this**.parentesis.push('(');

}**else** **if**(**this**.expresion[i]==')' && !**this**.parentesis.isEmpty()) {

**this**.parentesis.pop();

}**else** {

flag=**false**;

**break**;

}

}

}

**if**(**this**.parentesis.isEmpty() && flag) {

resParentesis=**true**;

System.***out***.println("Los parentesis estan balanceados");

}**else** {

resParentesis=**false**;

System.***out***.println("Los parentesis NO estan balanceados");

}

**this**.parentesis.clear(); //limpiar la pila

//VERIFICAR SI PROCEDE A REALIZARCE O NO

System.***out***.println("------------------------------------------------");

**if**(resNoVacio && resPrimer && resOperadores && resAritmetica && resParentesis) {

System.***out***.println("La operacion gramatica esta correctamente validada");

System.***out***.println("------------------------------------------------");

**return** **true**;

}**else** {

System.***out***.println("La operacion gramatica NO esta correctamente validada");

}

System.***out***.println("------------------------------------------------");

**return** **false**;

}

//Funcion principal

//Regresa una expresion postfija si es valida la cadena

**public** String convertidor() {

String expPostfija="";

**if**(validador()) {

**for**(**int** i=0;i<**this**.expresion.length;i++) {

**if**(esOperador(**this**.expresion[i])) {

**if**(**this**.expresion[i]=='(') {

pila.push('(');

}**else** **if**(**this**.expresion[i]==')') {

**while**(!pila.isEmpty()) {

**if**(pila.peek()!='(') {

expPostfija+=pila.pop();

}**else** {

pila.pop();

**break**;

}

}

}**else** **if**(**this**.expresion[i]=='\*' || **this**.expresion[i]=='+' || **this**.expresion[i]=='-' || **this**.expresion[i]=='/' || **this**.expresion[i]=='^') {

**if**(!pila.isEmpty() && pila.peek()!='(') {

**if**(prioridad(**this**.expresion[i])==prioridad(pila.peek())) {

expPostfija+=pila.pop();

pila.push(**this**.expresion[i]);

}**else** **if**(prioridad(**this**.expresion[i])>prioridad(pila.peek())) {

pila.push(**this**.expresion[i]);

}**else** **if**(prioridad(**this**.expresion[i])<prioridad(pila.peek())) {

**while**(!pila.isEmpty()) {

expPostfija+=pila.pop();

}

pila.push(**this**.expresion[i]);

}

}**else** {

pila.push(**this**.expresion[i]);

}

}

}**else** {

expPostfija+=**this**.expresion[i];

}

}

**while**(!pila.isEmpty()) {

expPostfija+=pila.pop();

}

evaluar(expPostfija);

**return** expPostfija;

}**else** {

**return** "NO ES UNA OPERACION ARITMETICA VALIDA";

}

}

//Evalua la expresion postfija e imprime el resultado

**public** **void** evaluar(String expPostfija) {

**double** valor;

**for**(**int** i=0;i<expPostfija.length();i++) {

**if**(!esOperador(expPostfija.charAt(i))) {

**this**.operaciones.push((**double**) Character.*getNumericValue*(expPostfija.charAt(i)));

}**else** {

valor=operacion(**this**.operaciones.pop(), **this**.operaciones.pop(), expPostfija.charAt(i));

**this**.operaciones.push(valor);

}

}

System.***out***.println("El resultado es: "+**this**.operaciones.pop());

}

**public** **static** **void** main(String[] args) {

//ExpresionAritmetica ea= new ExpresionAritmetica("(6+2)\*3/2^2-4");

ExpresionAritmetica ea= **new** ExpresionAritmetica("2/(3\*2)");

//ExpresionAritmetica ea= new ExpresionAritmetica("");

String resultado=ea.convertidor();

**if**(resultado=="NO ES UNA OPERACION ARITMETICA VALIDA") {

System.***out***.println(resultado);

}**else** {

System.***out***.println("Expresión Aritmética en Postfijo: "+resultado);

ArbolDeDerivacion ad=**new** ArbolDeDerivacion(resultado, "ej2.txt", "grafo2.jpg");

ad.insercion();

}

}

}

**Class ArbolDeDerivacion**

/\*

Nombre: Ulises Bojorquez Ortiz

Matricula: A01114716

Materia: Matematicas Computacionales

PROYECTO SEGUNDO PARCIAL: ANALIZADOR SINTACTICO DE ARITMETICA SIMPLE

\*/

**import** java.io.BufferedWriter;

**import** java.io.File;

**import** java.io.FileWriter;

**import** java.util.Stack;

**public** **class** ArbolDeDerivacion {

**private** Nodo root;

**private** Stack<Nodo> pila;

**private** **char**[] operacion;

**private** String ruta, salida;

**private** String dotInfo;

**private** Nodo[] espacios;

**public** ArbolDeDerivacion(String expPostfija, String ruta, String salida) {

**this**.root=**null**;

**this**.pila=**new** Stack<>();

**this**.operacion=expPostfija.toCharArray();

**this**.ruta=ruta;

**this**.salida=salida;

**this**.dotInfo="";

**this**.espacios=**new** Nodo[**this**.operacion.length];

}

//Verifica si el caracter es Operador o no

**public** **boolean** esOperador(**char** caracter) {

**if**(caracter=='\*' || caracter=='+' || caracter=='-' || caracter=='/' || caracter=='^') {

**return** **true**;

}**else** {

**return** **false**;

}

}

//Inserta los Nodos en un arbol (Lo crea)

**public** **void** insercion() {

Nodo nuevo;

**for**(**int** i=0;i<**this**.operacion.length;i++) {

**if**(!esOperador(**this**.operacion[i])) {

nuevo=**new** Nodo(**this**.operacion[i]);

**this**.espacios[i]=nuevo;

pila.push(nuevo);

}**else** {

nuevo=**new** Nodo(**this**.operacion[i]);

nuevo.right=**this**.pila.pop();

nuevo.left=**this**.pila.pop();

**this**.espacios[i]=nuevo;

**this**.pila.push(nuevo);

}

}

**this**.root=**this**.pila.pop();

//Verificar que es igual a la entrada

escritura();

generar();

System.***out***.println("ARBOL GENERADO");

}

//Escritura para poder leer el doc de GraphViz

**public** **void** escritura() {

**try** {

File file = **new** File(**this**.ruta);

// Si el archivo no existe es creado

**if** (!file.exists()) {

file.createNewFile();

}

FileWriter fw = **new** FileWriter(file);

BufferedWriter bw = **new** BufferedWriter(fw);

bw.write("digraph G {\n");

asignarEspacios();

bw.write(**this**.dotInfo);

bw.write("}");

bw.close();

} **catch** (Exception e) {

e.printStackTrace();

}

}

//Asinga espacios a caracteres repetidos para que que el GraphViz genere correctamente el arbol

**public** **void** asignarEspacios() {

Nodo actual;

String espacio=" ";

**for**(**int** i=0;i<**this**.espacios.length;i++) {

actual=**this**.espacios[i];

**for**(**int** j=i+1;j<**this**.espacios.length;j++) {

**if**(actual.info==**this**.espacios[j].info) {

**this**.espacios[j].espacios+=espacio;

}

}

}

**boolean**[] condicion=**new** **boolean**[**this**.espacios.length];

**for**(**int** i=0;i<condicion.length;i++) {

condicion[i]=**true**;

}

espacios(**this**.root,condicion);

direccion(**this**.root);

}

//Asigna los espacios en el Arbol principal

**public** **void** espacios(Nodo current, **boolean**[] condicion) {

**if**(current!=**null**) {

espacios(current.left, condicion);

espacios(current.right, condicion);

**for**(**int** i=0;i<**this**.espacios.length;i++) {

**if**(current.info==**this**.espacios[i].info && condicion[i]==**true**) {

current.espacios=**this**.espacios[i].espacios;

condicion[i]=**false**;

**break**;

}

}

}

}

//Contenido del GraphViz, asignación de dirección de los nodos

**public** **void** direccion(Nodo current) {

**if**(current!=**null**) {

**if**(current.left!=**null**) {

**this**.dotInfo+=""+'"'+current.info+current.espacios+'"'+"->"+'"'+current.left.info+current.left.espacios+'"'+"\n";

}

**if**(current.right!=**null**) {

**this**.dotInfo+=""+'"'+current.info+current.espacios+'"'+"->"+'"'+current.right.info+current.right.espacios+'"'+"\n";

}

direccion(current.left);

direccion(current.right);

}

}

**public** **void** generar(){

**try** {

//path del dot.exe,por lo general es la misma, pero depende de donde hayas instalado el paquete de Graphviz

String dotPath = "C:\\Program Files (x86)\\Graphviz2.38\\bin\\dot.exe";

//path del archivo creado con el codigo del graphviz que queremos

String fileInputPath = **this**.ruta;

//path de salida del grafo, es decir el path de la imagen que vamos a crear con graphviz

//String fileOutputPath = "grafo1.jpg";

String fileOutputPath = **this**.salida;

//tipo de imagen de salida, en este caso es jpg

//String tParam = "-Tjpg";

String tParam = "-Tjpg";

String tOParam = "-o";

//concatenamos nuestras direcciones. Lo que hice es crear un vector, para poder editar las direcciones de entrada y salida, usando las variables antes inicializadas

//recordemos el comando en la consola de windows: C:\Archivos de programa\Graphviz 2.21\bin\dot.exe -Tjpg grafo1.txt -o grafo1.jpg Esto es lo que concatenamos en el vector siguiente:

String[] cmd = **new** String[5];

cmd[0] = dotPath;

cmd[1] = tParam;

cmd[2] = fileInputPath;

cmd[3] = tOParam;

cmd[4] = fileOutputPath;

//Invocamos nuestra clase

Runtime rt = Runtime.*getRuntime*();

//Ahora ejecutamos como lo hacemos en consola

rt.exec( cmd );

} **catch** (Exception ex) {

ex.printStackTrace();

} **finally** {

}

}

**public** **static** **void** main(String[] args) {

ArbolDeDerivacion ad=**new** ArbolDeDerivacion("62+3\*22^/4-", "ej1.txt","grafo.jpg");

ad.insercion();

}

}

**class** Nodo{

**public** Nodo left,

right;

**public** **char** info;

**public** String espacios;

**public** Nodo(**char** info) {

**this**.info=info;

**this**.right=**null**;

**this**.left=**null**;

**this**.espacios="";

}

}

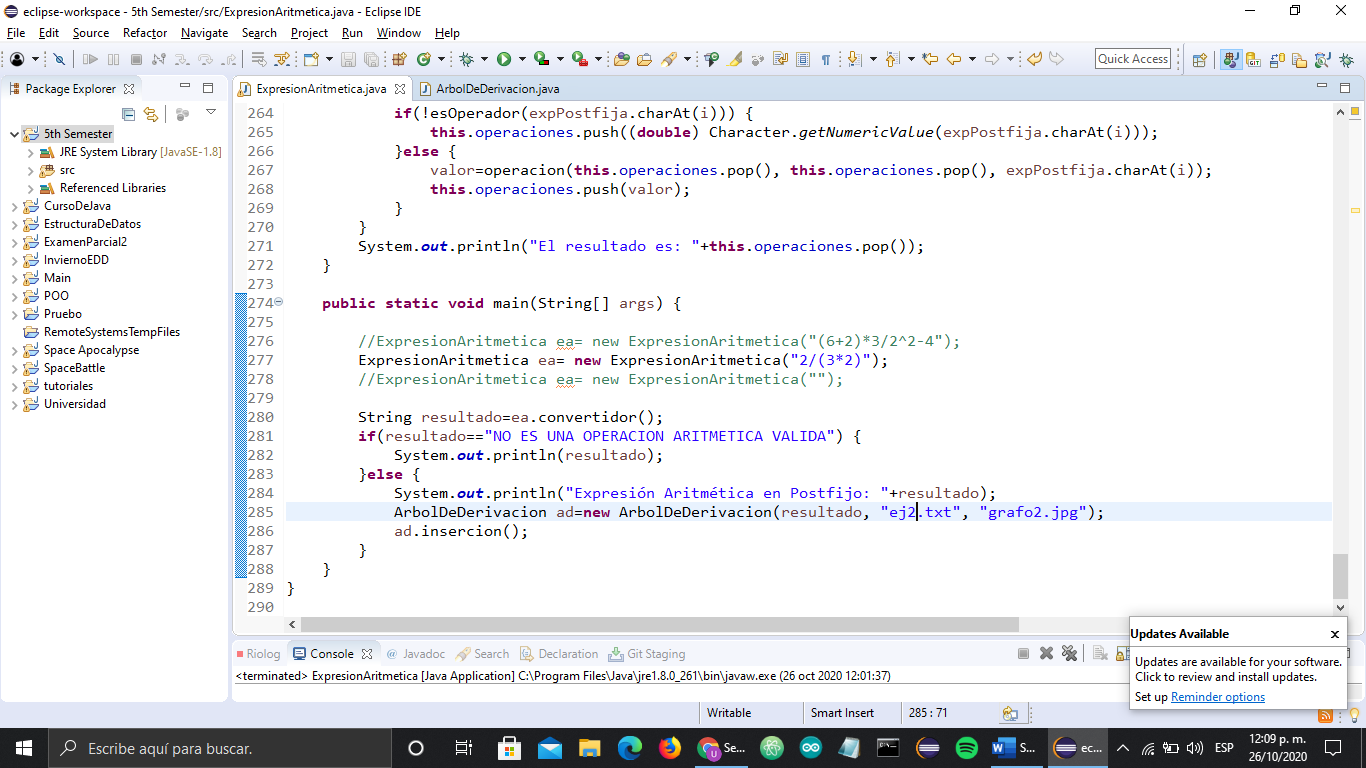
*Pruebas y resultados*

Expresión aritmética a validar es ***2/(3\*2)***

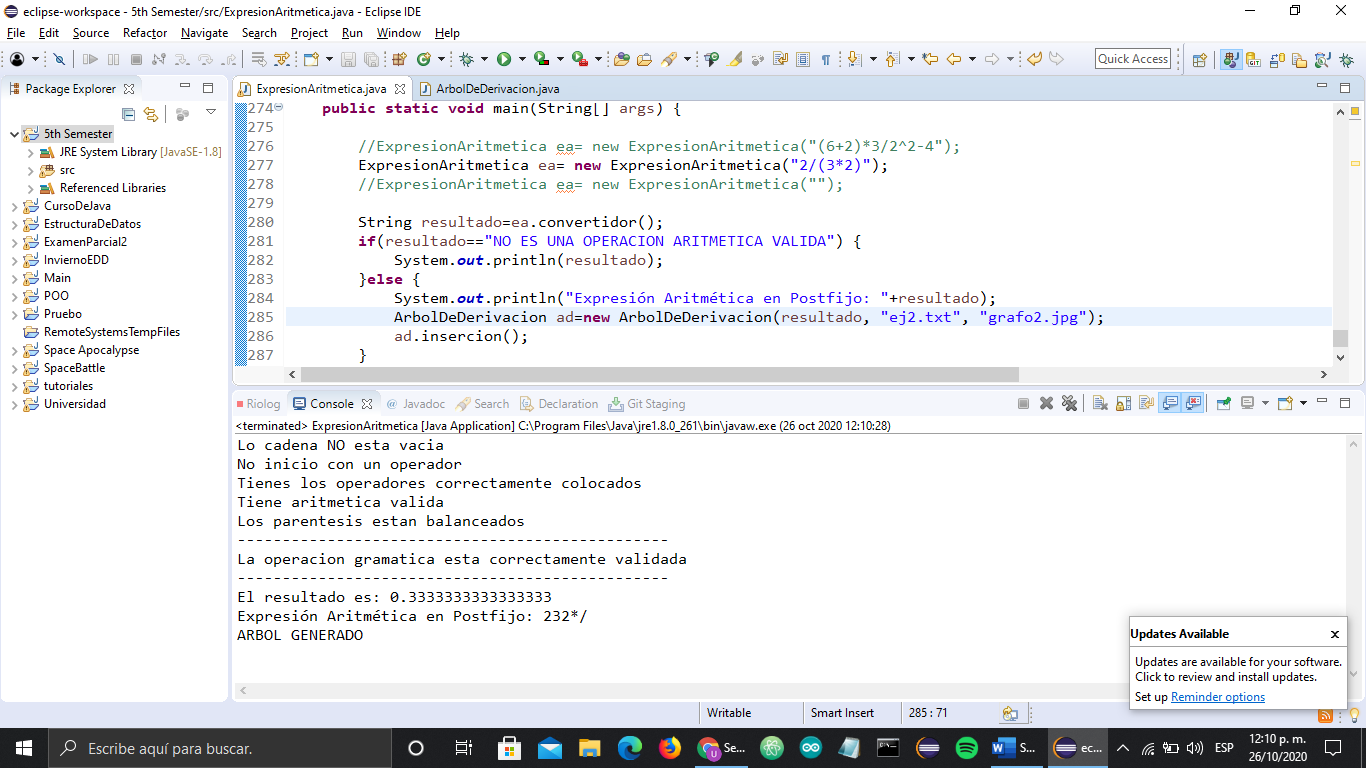
Dirección y nombre del documento de GraphViz ***ej2.txt***

Dirección y nombre del gráfico a generar ***grafo2.jpg***

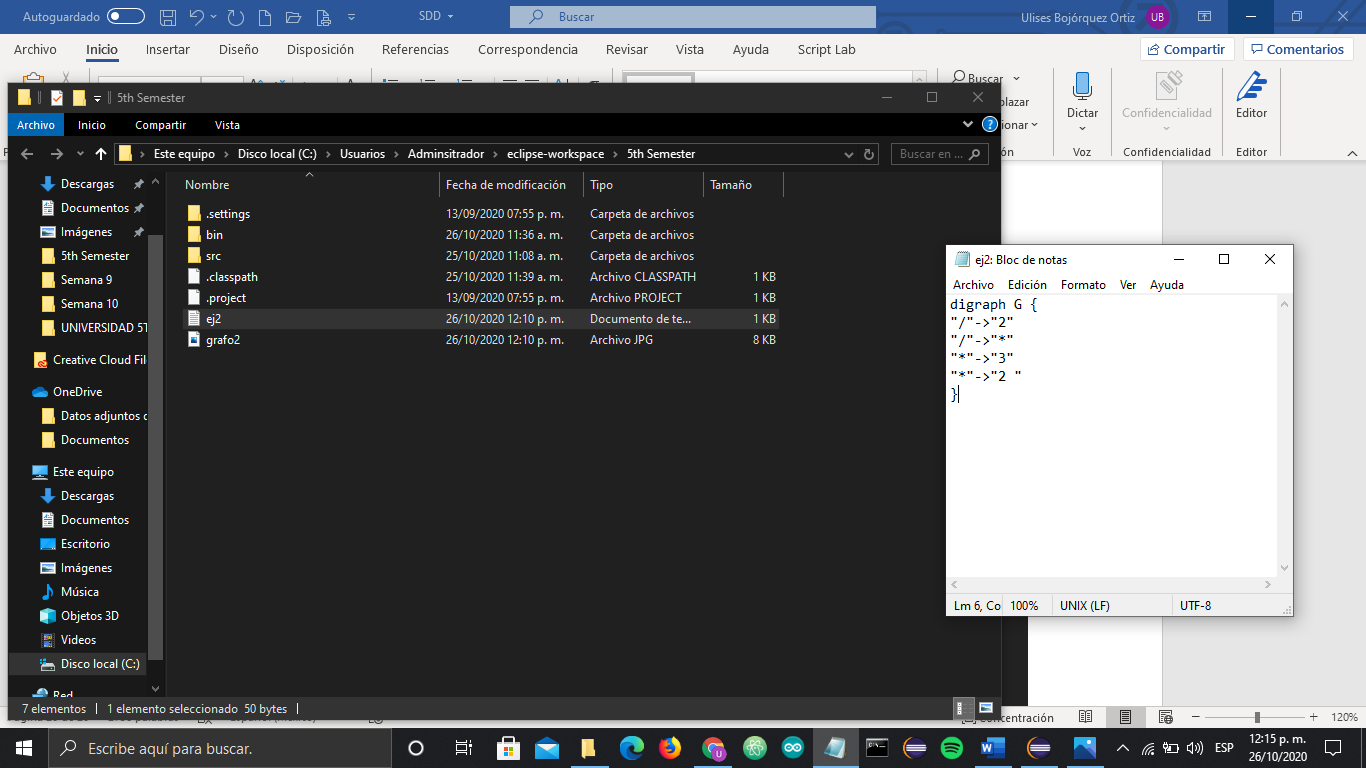
1. Se ingresan los datos



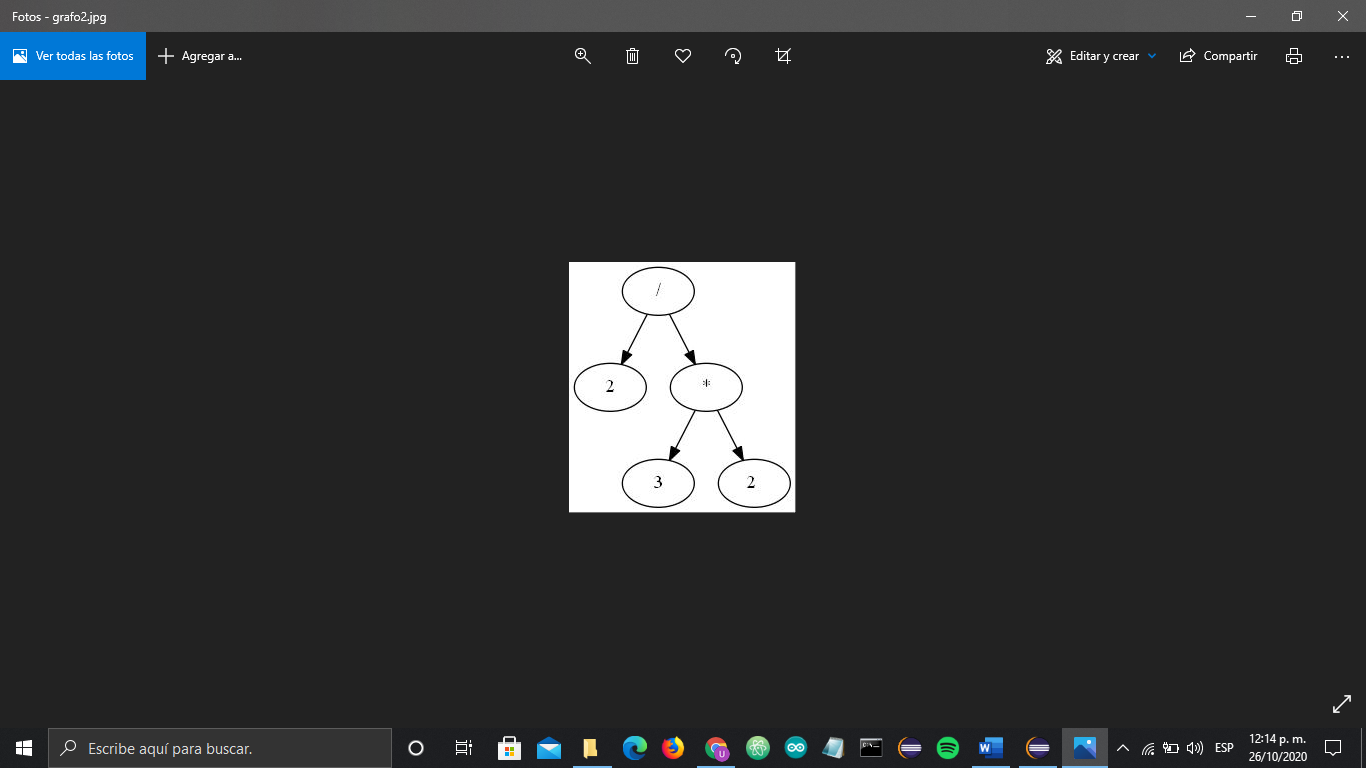
1. Se imprimen si la expresión aritmética es aceptada por la gramática, el resultado de dicha operación, al cadena de entrada en postfijo y si el arbol se genero



1. Documento generado



1. Árbol generado



**Conclusiones**

En conclusión, con este proyecto pude expandir mi conocimiento en varias áreas de la computación, retomé el uso de varios conceptos básicos que tenia tiempo sin aplicar y en base a todos esos conocimientos puede desarrollar un programa que resolviera la problemática, dando aún más de los requerimientos establecidos dentro del proyecto. Además, reforcé un tema de la clase que son las gramáticas de las cuales tenía varias complicaciones de comprensión.

**Referencias**

*Algoritmos*

<https://www.youtube.com/watch?v=5zhXm3J9xdE>

<https://www.youtube.com/watch?v=WHs-wSo33MM&t=313s>

*GraphViz*

Tutorial de instalación de GraphViz 1: <https://www.youtube.com/watch?v=WkLhdBbf-3E&t=157s>

Tutorial de instalación de GraphViz 2: <https://www.youtube.com/watch?v=JYAHwQ_tMG0> Implementación del GraphViz desde Java: <https://irisus90.wordpress.com/2011/06/25/uso-de-graphviz-desde-java/>

Proceso de instalación de forma escrita de GraphViz: <https://bobswift.atlassian.net/wiki/spaces/GVIZ/pages/20971549/How+to+install+Graphviz+software>