TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de La Laguna

TECNOLÓGICO NACIONAL DE MÉXICO®

Ingeniería en Sistemas Computacionales

Taller de Sistemas Operativos

Manual Técnico del Compilador 7Tokens

#21130598 Viramontes Gutiérrez Edmundo

#22130628 Maldonado Paz Neo Brandon Mictlantecuhtli

#21130596 Hernández Aguilar Diego Jesús

Docente - Pablo Saucedo Martinez

Torreón, Coahuila, México

30 de Mayo del 2024

Control de Versiones

Versión	Fecha	Descripción	Autores
2.0	Mayo 30 de 2024	Versión Inicial del Documento	- Edmundo Viramontes Gutiérrez - Neo Brandon Maldonado Paz - Diego Jesús Hernández Aguilar

Índice:

Introducción

Objetivo

Requerimientos de Software

Requerimientos Mínimos de Hardware

Explicacion del codigo

Modificaciones

1. Introducción

En este manual se presentan las modificaciones y avances hechos para el compilador 7Tokens, el compilador utiliza varias clases con el motivo de ser capaz de crear símbolos gramaticales los cuales pueden ser analizados según automatas definidos para un analisis lexico y popular pilas las cuales serán expuestas a producciones gramaticales con el motivo de un analisi sintactico.

2. Objetivo

El objetivo de este manual es el entender y explicar de una manera comprensiva el funcionamiento del compilador 7tokens el cual fue proporcionado por el docente de la materia.

Específicos:

- La agregación al análisis léxico de ser capaz de reconocer operadores dobles (++, –, **).
- La reparación del análisis sintáctico en función de establecer los mensajes de errores descriptivos y en qué línea se encuentra.
- Explicación del código del programa.

3. Requerimientos de Software

En esta sección se detalla los requisitos mínimos del sistema para poder ejecutar los aplicativos usados para modificar el software compilación de código Java:

Java: 21.0.1; Java HotSpot(TM) 64-Bit Server VM 21.0.1+12-LTS-29

Esta es la versión de Java utilizada para el desarrollo del compilador y por ende la versión recomendada para su utilización.

Apache NetBeans IDE 19

Si el usuario administrador / programador desea realizar alguna modificación sobre la base del código entonces deberá instalar la versión 19 de Apache NetBeans para comenzar con el reconocimiento del código por parte de un IDE

confiable, este IDE puede ser encontrado en el enlace https://netbeans.apache.org/front/main/index.html la cual es la página oficial del IDE de desarrollo.

4. Requerimientos Mínimos de Hardware

De los requisitos de hardware para la ejecución del compilador de software se necesitará de un equipo con al menos estos requisitos mínimos

Como parte de sistema operativo probado se requiere al menos:

- Sistema Operativo: Windows 7, Windows 8.1

Un procesador de al menos:

- Procesador: Intel Core Celeron , Intel Core Duo

Memoria RAM:

- Memoria RAM: 4 GB

Almacenamiento mínimo:

- Disco Duro: 20 Mb

Resolución de pantalla depende mucho de su gusto, pero mínimo correspondiente es:

- Resolución de pantalla: 1280 x 720 píxeles

Periféricos para poder interactuar con el software:

- Periféricos: Teclado, ratón

5. Explicacion del codigo

Clase SimbGram.java

```
public class SimbGram {

   String _elem;

   public SimbGram(String sValor) {
      _elem = sValor;
   }
```

```
    public String Elem() {
        return _elem;
    }
}
```

Esta clase es utilizada para poder crear símbolos gramaticales los cuales van a ser analizados posteriormente por las otras clases.

Clase Pila.java

```
public class Pila {

    final int MAX = 5000;
    SimbGram[] _elems;
    int _tope;

    public Pila() {
        _elems = new SimbGram[MAX];
        for (int i = 0; i < _elems.length; i++) {
            _elems[i] = new SimbGram("");
        }
        _tope = 0;
    }

    public boolean Empty() {
        return _tope == 0 ? true : false;
    }

    public boolean Full() {
        return _tope == _elems.length ? true : false;
    }

    public void Push(SimbGram oElem) {
        _elems[_tope++] = oElem;
    }

    public int Length() {
        return _tope;
    }

    public SimbGram Pop() {
        return _elems[--_tope];
    }

    public void Inicia() {
        _tope = 0;
    }
```

```
    public SimbGram Tope() {
        return _elems[_tope - 1];
    }
}
```

Esta clase tiene como función la generación de una pila [], la cual se va a utilizar para poder comprobar la existencia de las producciones de gramática. La clase cuenta con los métodos:

- Empty() = Este método lo que hace es vaciar el tope de la pila.
- Full() = Este método nos indica si la pila se encuentra llena.
- Push(SimbGram oElem) = Este método nos permite insertar a la pila un elemento SimboloGramatical.
- Length() = Este método nos retorna la longitud de la pila.
- Pop() = Este método saca de la pila un elemento.
- Inicia() = Este método establece el tope como 0.

Clase Automata.java

```
private boolean ReconoceReal1(String texto, int[] _i, int iniToken) {
    //Aquí ya llega numérico i alterada, o sea ya tenemos una parte del
    //lexema
    //Expresión regular para los que terminan con un .
    Pattern pa = Pattern.compile( input: "[illegible]");
    String tex = "";
    Matcher ma = pa.matcher( input: texto.substring( beginIndex: iniToken, _i[0]));
    //boolean b;
    if (ma.matches()) {
        //Si coincide con un punto, puede que sea un real2 [illegible]
        //Para eso llamamos a real2
        ReconoceReal2(texto, _i, iniToken);
    }
    return ma.matches();
}

private boolean ReconoceReal2(String texto, int[] _i, int iniToken) {
    Pattern pa = Pattern.compile( input: "[illegible]");
    String tex = "";
    Matcher ma = null;
    try {
        ma = pa.matcher( input: texto.substring( beginIndex: iniToken, ++_i[0]));
        tex = texto.substring( beginIndex: iniToken, _i[0]);
    } catch (Exception e) {
        --_i[0];
        return false;
    }
    while (ma.matches()) {
        try {
            tex = texto.substring( beginIndex: iniToken, ++_i[0]);
            ma = pa.matcher( input: tex);
        } catch (Exception e) {
            tex = texto.substring( beginIndex: iniToken, (_i[0]) - 1);
            ma = pa.matcher(tex + "[illegible]");
        }
    }
    if (!ma.matches() && !tex.equals( anObject: "")) {
        _i[0]--;
        return true;
    } else {
        _i[0] = iniToken;
        return false;
    }
}

private boolean ReconoceReal3(String texto, int[] _i, int iniToken) {
    //Aquí ya llega numérico i alterada.
    Pattern pa = Pattern.compile( input: "[illegible]");
    String tex = "";
    Matcher ma = null;
    try {
        //[illegible]
        ma = pa.matcher( input: texto.substring( beginIndex: iniToken, ++_i[0]));
    } catch (Exception e) {
        //[illegible]
        //[illegible]
        int i = _i[0] - 2;
        _i[0] = i;
        return false;
    }
    //[illegible]
    //[illegible]
    while (ma.matches()) {
        try {
            //[illegible]
            tex = texto.substring( beginIndex: iniToken, ++_i[0]);
            ma = pa.matcher( input: tex);
        } catch (Exception e) {
            ma = pa.matcher(tex + "[illegible]");
```

```
    }
    //[illegible]
    if (!ma.matches() && !tex.equals( anObject: "")) {
        _i[0]--;
        return true;
        //Recuperación de error [illegible]
    } else {
        _i[0] = iniToken;
        return false;
    }
}

//[illegible]
//[illegible]
//[illegible]
private void ReconoceExponencial(String texto, int[] _i, int iniToken) {
    //[illegible]
    String e = "[illegible]";
    String E = "[illegible]";
    String es = "[illegible]";
    String Es = "[illegible]";
    //[illegible]
    //[illegible]
    Pattern pa = Pattern.compile(e + "|" + E + "|" + es + "|" + Es + "[illegible]" + "[illegible]");
    Matcher ma;
    try {
        //[illegible]
        if (texto.charAt(_i[0]) == 'e' || texto.charAt(_i[0]) == 'E') {
            //[illegible]
            //[illegible]
            int i = _i[0];
            ++_i[0];

            ma = pa.matcher( input: texto.substring( beginIndex: i, ++_i[0]));
            //[illegible]
            String tex = "";
            //[illegible]
            while (ma.matches()) {
                try {
                    tex = texto.substring( beginIndex: i, ++_i[0]);
                    ma = pa.matcher( input: tex);
                    //[illegible]
                } catch (Exception exc) {
                    tex = texto.substring( beginIndex: i, (_i[0]) - 1);
                    ma = pa.matcher(tex + "[illegible]");
                }
            }
            boolean j = ma.matches();
            if (!ma.matches() && !tex.equals( anObject: "")) {
                --_i[0];
            } else {
                _i[0] = iniToken;
            }
        } else {
        }
        //[illegible]
    } catch (Exception ex) {
        int e = _i[0] - 2;
        _i[0] = e;
    }
}

private boolean ReconoceSep(String texto, int[] _i, int iniToken) {
    //[illegible]
    Pattern pa = Pattern.compile( input: "[illegible]");
    //[illegible]
    Matcher ma = pa.matcher( input: texto.substring( beginIndex: iniToken, ++_i[0]));
    if (ma.matches()) {
        //Si coincide, es un separador
```

```
private boolean ReconoceCad(String texto, int[] _i, int iniToken) {
    //Reconocer cualquier número de caracteres (1 o más) que sean parte
    //del ASCII y estén entre comillas.
    Pattern pa = Pattern.compile( regex: "[illegible]");
    Matcher ma = pa.matcher( input: texto.substring( beginIndex: iniToken, ++_i[0]));
    //Mientras no coincida.
    //¿Porqué esto? Porque al inicio nunca coincidirá, ya que no tendremos
    //las comillas de cierre hasta que lleguemos al final de la cadena.
    while (!ma.matches()) {
        //Intentamos obtener todos los lexemas y vamos al compararlos
        try {
            ma = pa.matcher( input: texto.substring( beginIndex: iniToken, ++_i[0]));
        } catch (Exception e) {
            //Excepción por si se nos acaba el espacio
            ma = pa.matcher( input: texto.substring( beginIndex: iniToken, --_i[0]));
            break;
        }
    }
    //Si coincide, es un caracter
    if (ma.matches()) {
        return true;
    } else {
        //Recuperación de error, siguiente autómata
        _i[0] = iniToken;
        return false;
    }

}

private boolean ReconoceCar(String texto, int[] _i, int iniToken) {
    //Cualquier ASCII entre apostrofes. Esto incluye hexadecimal y octal.
    Pattern pa = Pattern.compile( regex: "[illegible]");
    //Si tiene un backslash al inicio
    if (texto.charAt( index: iniToken) == '[illegible]') {
        Matcher ma = pa.matcher( input: texto.substring( beginIndex: iniToken, ++_i[0]));
        String tex;
        //Mientras no coincida, leemos el siguiente caracter
        while (!ma.matches()) {
            try {
                tex = texto.substring( beginIndex: iniToken, ++_i[0]);
                ma = pa.matcher( input: tex);
                //Si es una 'x' coincidirá luego luego. Si no, irá tomando de 1 por 1
                //hasta que se complete el hexadecimal o el octal
            } catch (Exception e) {
                //Si entramos aquí es porque nos quedamos sin espacio.
                //¡Salgamos de aquí!
                ma = pa.matcher( input: texto.substring( beginIndex: iniToken, --_i[0]));
                break;
            }
        }
        //Regresa si coincide. Si no entramos en la excepción, todo ha ido bien.
        //True si no entramos en la excepción, false si entramos.
        return ma.matches();
    }

    return false;

}

private boolean ReconoceTer(String texto, int[] _i, int iniToken) {
    //Expresión regular para reconocer puntos y comas
    Pattern pa = Pattern.compile( regex: "[illegible]");
    Matcher ma = pa.matcher( input: Character.toString( c: texto.charAt( index: iniToken)));
    if (ma.matches()) {
        _i[0]++;
        return true;
    } else {
        return false;
    }
```

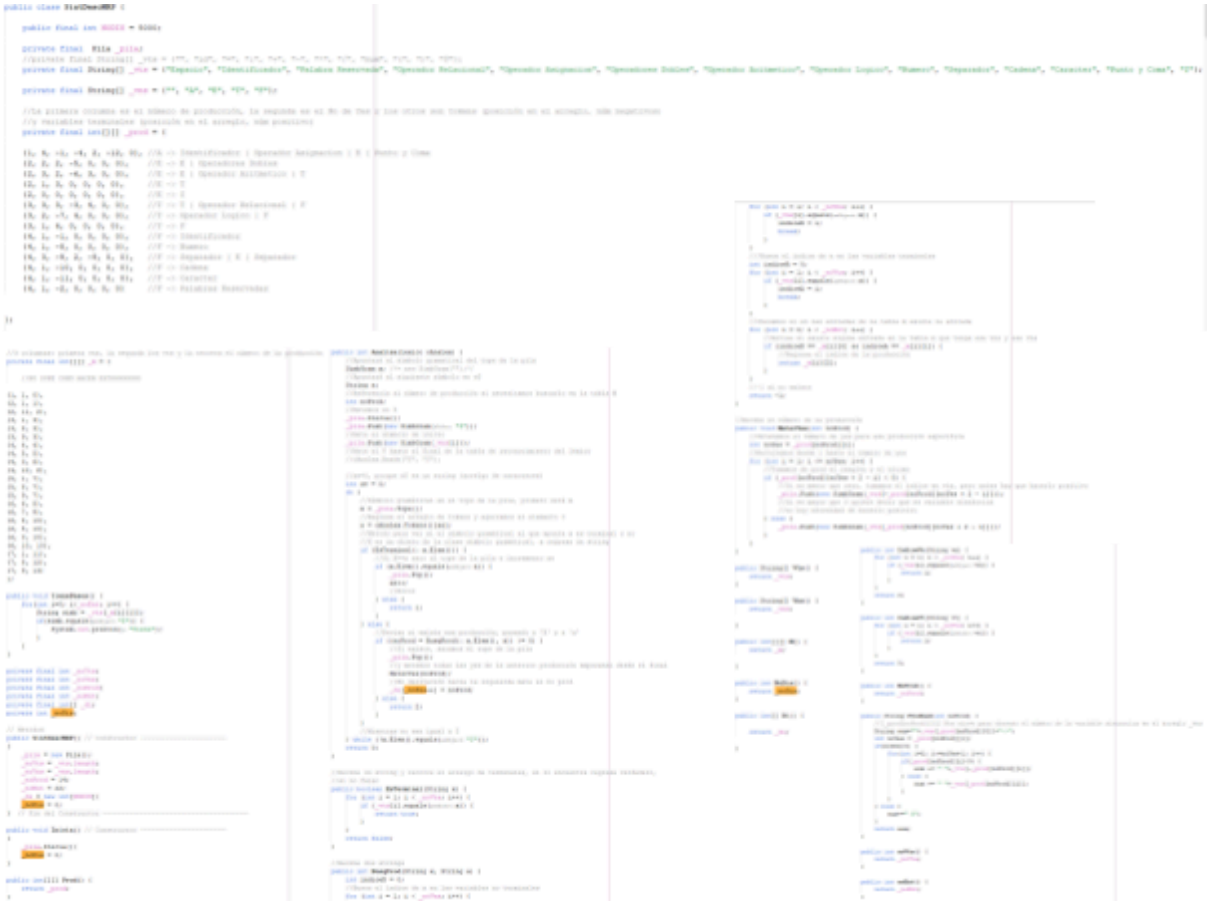
Esta clase tiene la función de identificar el tipo de autónoma que es el texto, existen bastantes métodos en este clase (ver imágenes) pero estos se manejan de una manera muy similar, mediante el uso de java.util.regex.Matcher y java.util.regex.Pattern establecemos parámetros para poder identificar patrones específicos con el fin de identificar el tipo, los métodos identifican doce tipos de autómatas: Espacios, Identificador, Operador Relacional, Operador Asignacion, Operadores Dobles, Operador Aritmetico, Operador Logico, Numero, Separador, Cadena, Caracter y Punto y Coma.

Clase Lexico.java

```
            }
            if (noAuto != 0) {
                _lexemas[_noTokens++] = _lexema;
            }
        } else {
            return false; //La parte de recuperación del error léxico se reemplaza por
        }       //este return false para indicar que existe un error léxico

        //Movemos iniToken, estamos listos para leer el siguiente token
        _iniToken = _i[0];
    }
    return true; //El análisis léxico ha sido exitoso cuando acaba el while
```

La función de esta clase es reconocer el autómata y mediante el número del autómata establecer el lexema correspondiente al token que se está leyendo.

Clase SintDescNRP.java

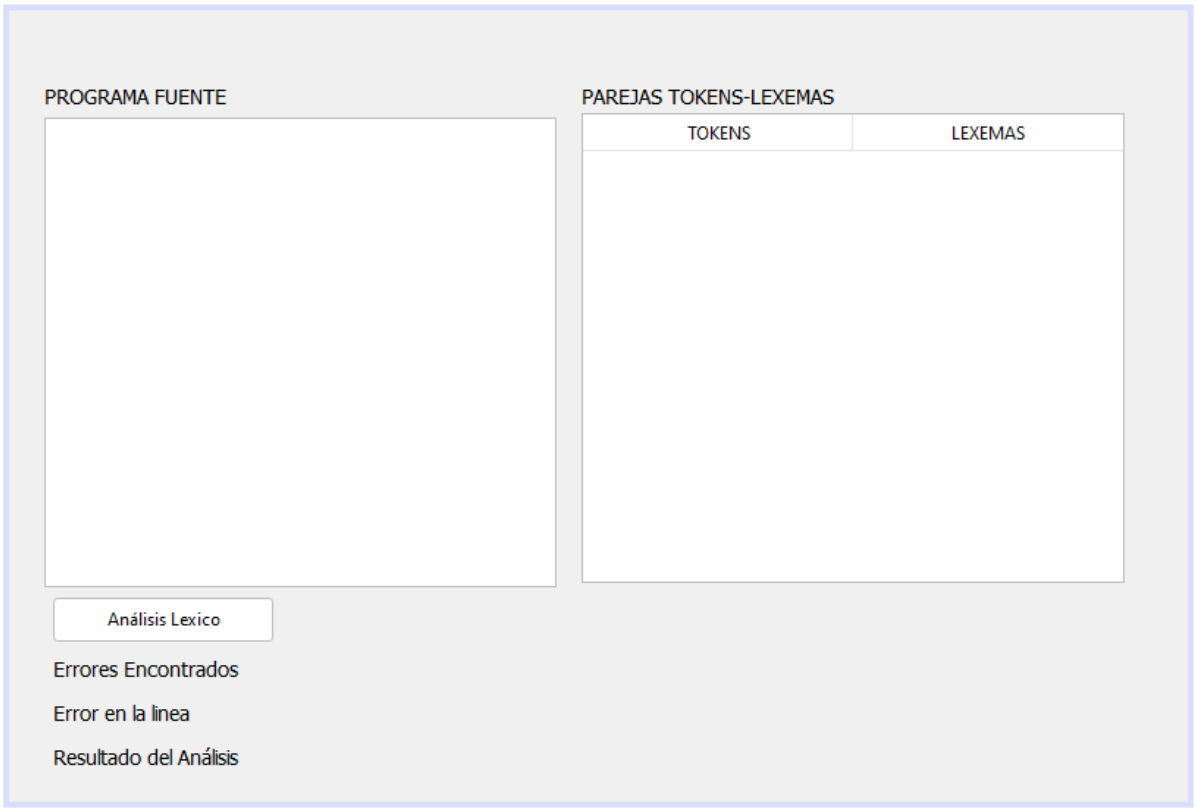


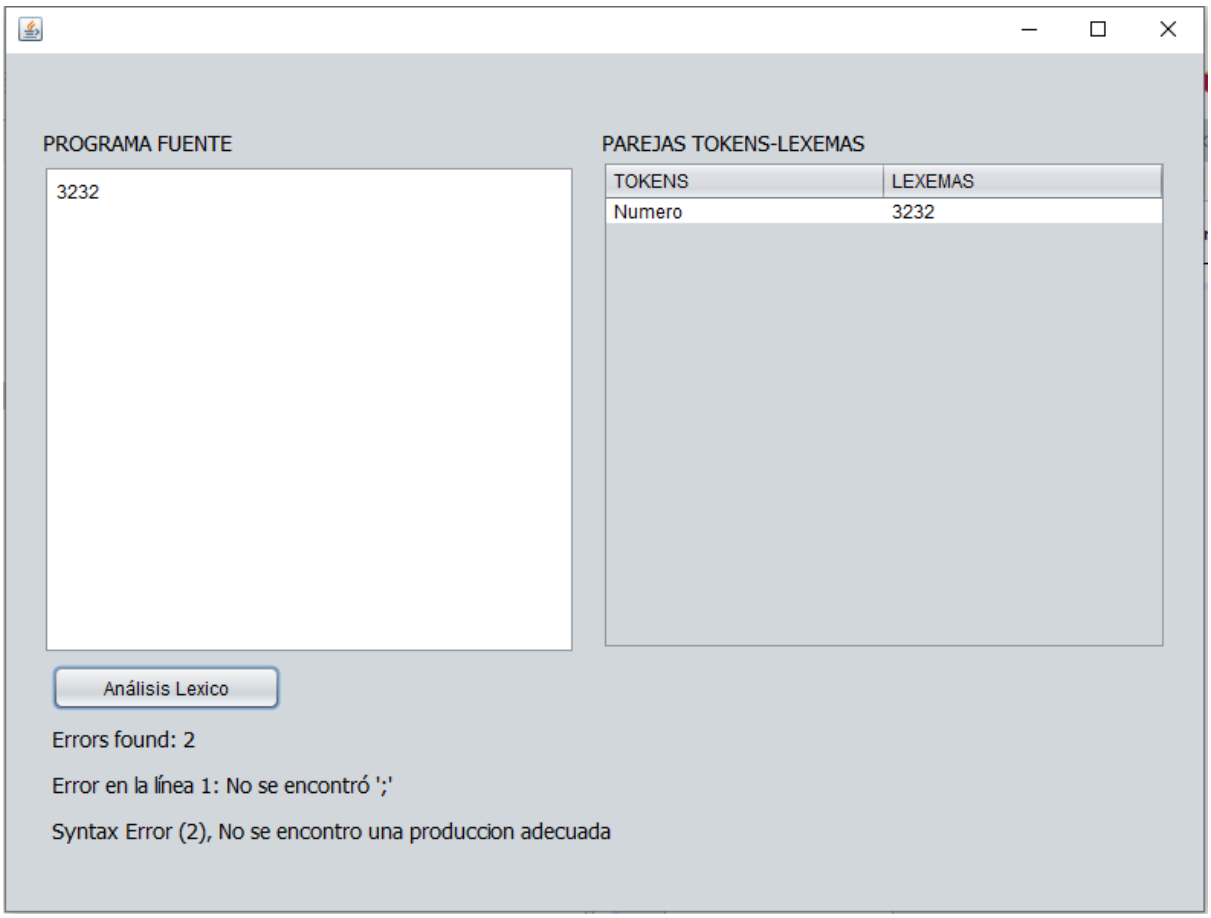
Esta clase tiene como objetivo identificar error en la sintaxis según la gramática establecida, siendo los errores que es capaz de analizar:

- Error (1) hace referencia a que no se reconoce el token en las variables terminales establecidas.
- Error (2) hace referencia a que no es capaz de asignar una producción,

y si no hay error, manda una instrucción la cual cual identifica que se analizó correctamente sin errores

AnaLex7TokensApp.frame





Frame en el cual nos permite insertar secuencias de texto y nos muestra su análisis léxico en el cual nos identifica la pareja token-lexema, además de si nuestro texto tiene algún error de sintaxis o léxico.

5. Modificaciones:

Cambios realizados por #21130598

AnaLex7TokensApp.java:

1. los imports:
   ```
   import java.util.regex.Matcher;
   import java.util.regex.Pattern;
   import java.util.HashMap;
   import java.util.Map;

   //Declaración
   private Map<String, Number> numericVariables;

   //Inicialización
   numericVariables = new HashMap<>();
   ```

   Con el motivo de mandarlos llamar en los nuevos métodos que agregue

2. modificaciones a métodos ya establecidos:

   en: private void btnAnaSinActionPerformed(java.awt.event.ActionEvent evt)

   ```
   findVariables(lineas[i]);
         jLabel_FormGen.setText("Variables numericas y sus valores: " +
   numericVariables);
   ```

   Con el motivo de buscar las variables numéricas (int, double, etc, checar el matcher del método findVariables)

3. nuevos métodos:

   Estos buttons utilizan métodos nuevos (Ver más adelante que establecen estos nuevos métodos)

   ```
   private void formGeneral_btnActionPerformed(java.awt.event.ActionEvent evt) {
         // TODO add your handling code here:
         String[] lineas = txaProgFuente.getText().split("\\n");
         for (int i = 0; i < lineas.length; i++) {
            try {
               evaluateFormulaGeneral(lineas[i]);
   ```

```
            } catch (Exception ex) {


Logger.getLogger(AnaLex7TokensApp.class.getName()).log(Level.SEVERE, null,
ex);
            }
        }

    }
```

Este btn le das el click al botón y busca en las líneas del JTextArea “formulaGeneral” mediante el método evaluateFormulaGeneral().

—----------

```
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        jLabel_FormGen.setText("");
        DresultadosFG_lbl.setText("");
        resultadosFG_lbl1.setText("");
        numericVariables.clear();
        errorFG_lbl.setText("");
    }
```

Este btn limpia parámetros y labels

—----------

```
    private void jButton_EJEMPLOBIENActionPerformed(java.awt.event.ActionEvent
evt) {
        // TODO add your handling code here:
        txaProgFuente.setText("int a = 2;\nint b = 3;\nint c = 1;\nformulaGeneral(a, b,
c)");
    }
```

Pobla el JTextField con un ejemplo de fórmula general correcto.

—----------

```
    private void jButton_EJEMPLOMALActionPerformed(java.awt.event.ActionEvent
evt) {
        // TODO add your handling code here:
        txaProgFuente.setText("int a = 2;\nint b = 3;\nint c = 99;\nformulaGeneral(a, b,
c)");
    }
```

Pobla el JTextField con un ejemplo de fórmula general incorrecto.

—----------

```
private void findVariables(String text) {
        //PATRON DE IDENTIFICACION DE VARIABLES

        String variablePattern =
"\\b(int|double|float|long|short)\\s+(\\w+)\\s*=\\s*([-+]?\\d*\\.?\\d+);";
        Pattern pattern = Pattern.compile(variablePattern);
        Matcher matcher = pattern.matcher(text);

        while (matcher.find()) {
            String type = matcher.group(1);
            String name = matcher.group(2);
            String value = matcher.group(3);

            switch (type) {
                case "int":
                    //LO AGREGAMOS AL AL HASHMAP SI ES NUMERICO PA
                    numericVariables.put(name, Integer.parseInt(value));
                    break;
                case "double":
                    numericVariables.put(name, Double.parseDouble(value));
                    break;
                case "float":
                    numericVariables.put(name, Float.parseFloat(value));
                    break;
                case "long":
                    numericVariables.put(name, Long.parseLong(value));
                    break;
                case "short":
                    numericVariables.put(name, Short.parseShort(value));
                    break;
                default:

                    break;
            }
        }
    }
```

Este metodo busca el patron (variables numericas declaradas e inicializadas de manera correcta) dentro del JTextArea y si matchea las clasifica y las agrega a nuestro HashMap(numericVariables) con a su tipo y valor.

—----------

```
private void evaluateFormulaGeneral(String expression) throws Exception {
            // Busca formulaGeneral() y saca el valor de las variables
```

```
        Pattern pattern =
Pattern.compile("formulaGeneral\\((\\w+),\\s*(\\w+),\\s*(\\w+)\\)");
        Matcher matcher = pattern.matcher(expression);
           if (matcher.find()) {
              String aVar = matcher.group(1);
              String bVar = matcher.group(2);
              String cVar = matcher.group(3);

              // Get the values of the variables
              double a = numericVariables.get(aVar).doubleValue();
              double b = numericVariables.get(bVar).doubleValue();
              double c = numericVariables.get(cVar).doubleValue();

              // Evaluate the quadratic formula
              double temp1 = Math.pow(b, 2); // b^2
              double temp2 = 4 * a;          // 4a
              double temp3 = temp2 * c;      //4ac
              double temp4 = temp1 - temp3; //b^2 - 4ac

              double d = temp4;
              DresultadosFG_lbl.setText("<html>Valor de d: <b>" + d + "</b></html>");
              //System.out.println("Value of d: " + d);

              if (d < 0) {
                 errorFG_lbl.setText("<html>La solucion no es real. <b>Numeros
imaginarios</b></html>");
                 System.out.println("NO, FALLO!!! CHeCA el plantamiento del ejercicio
manco");
              } else {
                 double temp5 = -b;
                 double temp6 = 2 * a;
                 double temp7 = Math.sqrt(d);
                 double temp8 = temp5 + temp7;
                 double temp9 = temp5 - temp7; //alterno
                 double temp10 = temp8 / temp6;
                 double temp11 = temp9 / temp6; //alterno


                 double x1 = (-b + Math.sqrt(d)) / (2 * a);
                 double x2 = (-b - Math.sqrt(d)) / (2 * a);

                 numericVariables.put("x1", x1);
                 numericVariables.put("x2", x2);

                 resultadosFG_lbl1.setText("<html>x1: <b>" + x1 + "</b> x2: <b>" + x2
+ "</b></html>");
                 temps.setText("<html>temp1: "+temp1+ " -> <em>b^2</em><br>" +
                       "temp2: "+temp2 + " -> <em>4a</em><br>" +
```

```
                        "temp3: "+temp3 + " -> <em>4ac</em><br>" +
                        "temp4: "+temp4 + " -> <em>b^2 - 4ac</em><br>" +
                        "temp5: "+temp5 + " -> <em>- b</em><br>" +
                        "temp6: "+temp6 + "  -> <em>2 * a</em><br>" +
                        "temp7: "+temp7 + " -> <em>sqrt(b^2 - 4ac)</em><br>" +
                        "temp8: "+temp8 + " -> <em>- b <b>+</b> sqrt(b^2 -
4ac)</em><br>" +
                        "temp9: "+temp9 + " -> <em>- b <b>-</b> sqrt(b^2 -
4ac)</em><br>" +
                        "temp10: "+temp10 + " -> <em><b>+</b> /(2a)</em><br>" +
                        "temp11: "+temp11 +" -> <em><b>-</b> /(2a)</em></html>");
                    //System.out.println("x1: " + x1);
                    //System.out.println("x2: " + x2);
                }
            } else {
                errorFG_lbl.setText("No se encontro error en el calculo");
            }
        }
```

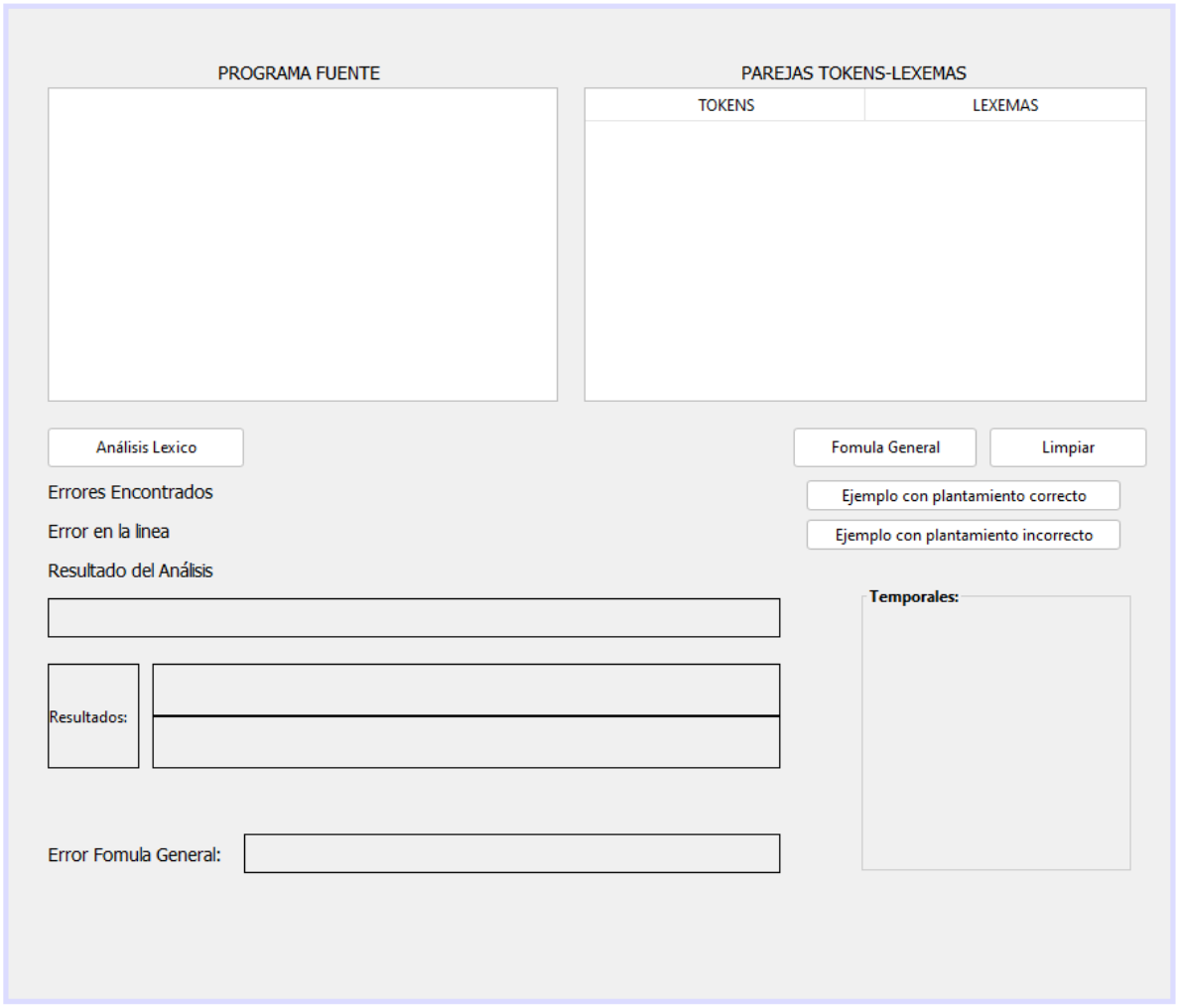
Este método al igual que el anterior busca dentro del JTextField un patrón (formulaGeneral(a, b, c)) y al encontrarlo se declaran las variables aVar, bVar, cVar con el motivo de poblarlas con las variables ya establecidas dentro de nuestro HashMap (numericVariables), se calcula los temporales y si d es menor que 0 esto significa que la solución NO es real, es decir seria imaginaria, en cambio si d es mayor que 0 se calcula los temporales de x1 y x2.

```
// ---------------------------------- Prueba variable numerica =
letra-----------------------------------------------------------------------------------------------
        String ultimaLinea = lineas[lineas.length - 1];

        String regex = "^(int|double|short|float|long)\\b.*";

        int prueba = 0;

        for (int io = 0; io < lineas.length; io++) {
            String linea = lineas[i].trim();
            boolean apertura2 = linea.matches(regex);

            if (apertura2) {
                prueba = 1;
            } else {
                prueba = 2;
            }

            // Para probar, podemos imprimir el valor de 'prueba' y la línea
correspondiente
            System.out.println("Linea: " + linea + " -> Prueba: " + prueba);
```

```
        }

        if (ultimaLinea.length() >= 2) {
        char penultimoCaracter = ultimaLinea.charAt(ultimaLinea.length() - 2);

        // Verificar si el penúltimo carácter es una letra o un dígito
        if (Character.isLetter(penultimoCaracter) && prueba == 1) {
            System.out.println("El penúltimo carácter es una letra.");
            lblResult1.setText("Error en la línea " + (i + 1) + ": No se puede asignar una
    letra a una variable numerica");
        } else if (Character.isDigit(penultimoCaracter)) {
            System.out.println("El penúltimo carácter es un dígito.");
        }
        }

    //-------------------------------------------------------------------------------------------------------------
    ------------------------------------------------------
```

Este cambio detecta un error si se le asigna a una variable numérica (int,double,short,float,long) una letra.

—----------



Se le dio una actualización al UI para un mayor tamaño y se le agregó el display de temporales.

6. Bibliografía

NetBeans IDE. (n.d.). Oracle. Retrieved Mayo 1, 2024, from https://www.oracle.com/mx/tools/technologies/netbeans-ide.html

¿Qué es Java y por qué lo necesito? (n.d.). Java. Retrieved Mayo 1, 2024, from https://www.java.com/es/download/help/whatis_java.html