MANUAL TÉCNICO

ANALIZADOR LÉXICO
PRACTICA 1 LENGUAJES FORMALES
APLICACIÓN DE ESCRITORIO

CRISTIAN ALEJANDRO ROLDÁN LÓPEZ
202147280

Especificaciones

Versión de Java utilizado: Java 21.
Versión de JDK: 21
Sistema Operativo Utilizado: Ubuntu 24.04 LTS
IDE utilizado: Apache NetBeans IDE 21.

Metodos para identificar un token identificador.

```
public class generadorIdentificadores {

    private char numero[] = {'0','1','2','3','4','5','6','7','8','9'};
    private char subra = '_';
    private char letras [] = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
            'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',
            'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
            'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'};



    public String validarCadena(String cadena) {
        if (cadena == null || cadena.isEmpty()) {
            return "Cadena inválida: La cadena está vacía.";
        }

        char primerCaracter = cadena.charAt(0);
        if (!esLetra(primerCaracter)) {
            return "Cadena inválida: La cadena debe comenzar con una
letra.";
        }

        for (int i = 1; i < cadena.length(); i++) {
            char c = cadena.charAt(i);
            if (!esLetra(c) && !esNumero(c) && c != subra) {
                return "Cadena inválida: La cadena contiene caracteres
inválidos.";
            }
        }

        return "Cadena válida: " + cadena;
```

```
    }

    public Color obtenerColorParaCadena(String cadena) {
        // Validar la cadena
        String resultado = validarCadena(cadena);

        // Si la cadena es válida, devolvemos el color amarillo, sino el
color blanco
        if (resultado.equals("Cadena válida: " + cadena)) {
            return Color.YELLOW;
        } else {
            return Color.WHITE;
        }
    }

    public boolean esLetra(char c) {
        for (char letra : letras) {
            if (c == letra) {
                return true;
            }
        }
        return false;
    }

    private boolean esNumero(char c) {
        for (char num : numero) {
            if (c == num) {
                return true;
            }
        }
        return false;
    }
```

```
    // Nuevo método para verificar identificadores alfanuméricos válidos
    public boolean esIdentificadorValido(String cadena) {
        if (cadena == null || cadena.isEmpty()) {
            return false;
        }

        char primerCaracter = cadena.charAt(0);
        if (!esLetra(primerCaracter)) {
            return false;
        }

        for (int i = 1; i < cadena.length(); i++) {
            char c = cadena.charAt(i);
            if (!esLetra(c) && !esNumero(c) && c != subra) {
                return false;
            }
        }

        return true;
    }


}
```

Token de signos

```
public class generadorSignos {

    private Color coloParen = new Color(0x9AD8D8);
    private Color coloLlaves = new Color(0xDBD29A);
```

```
    private Color coloCorche = new Color(0xDBA49A);
    private Color coloComa = new Color(0xB79ADB);
    private Color coloPunto = new Color(0x9ADBA6);

    private String paren [] = {"(", ")"};
    private String llaves [] = {"{}"};
    private String corche [] = {"[]"};
    private char coma [] = {','};
    private char punto [] = {'.'};

    public String validarSignos (String cadena){
        if (cadena == null || cadena.isEmpty()) {
            return "Cadena inválida: La cadena está vacía.";
        }

        cadena = cadena.trim();

        if (esParentesis(cadena) || esLlave(cadena) ||
esCorchete(cadena) || esComa(cadena.charAt(0)) ||
esPunto(cadena.charAt(0))) {
            return "Cadena válida: " + cadena;
        }

        return "Cadena inválida: La cadena no es un operador de
comparación válido.";

    }


    public Color colorSigno (String cadena){
        String resultado = validarSignos(cadena);
```

```
    if (resultado.equals("Cadena válida: " + cadena)) {
        cadena = cadena.trim();
        if (esParentesis(cadena)) {
            return coloParen;
        }else if (esLlave(cadena)) {
            return coloLlaves;
        }else if (esCorchete(cadena)) {
            return coloCorche;
        }else if (esComa(cadena.charAt(0))) {
            return coloComa;
        }else if (esPunto(cadena.charAt(0))) {
            return coloPunto;
        }
    }
    return Color.WHITE;
}


public boolean esParentesis (String c){
    c = c.trim();
    for (String sigParen : paren) {
        if (c.equals(sigParen)) {
            return true;
        }
    }
    return false;
}

public boolean esLlave (String c){
    c = c.trim();
    for (String sigLlave : llaves) {
        if (c.equals(sigLlave)) {
```

```
            return true;
        }
    }
    return false;
}

public boolean esCorchete (String c){
    c = c.trim();
    for (String sigCorche : corche) {
        if (c.equals(sigCorche)) {
            return true;
        }
    }
    return false;
}


public boolean esComa (char c){
    for (char sigComa : coma) {
        if (c == sigComa) {
            return true;
        }
    }
    return false;
}

public boolean esPunto (char c){
    for (char sigPunto : punto) {
        if (c == sigPunto) {
            return true;
        }
    }
```

```
        return false;
    }


}
```

Token Tipo de dato

```
package Identificadores;

import java.awt.Color;

public class generadorTiposDato {

    private Color enteroColor = new Color(0x1BA1E2);
    private Color deciColor = new Color(0xFFFF88);
    private Color cadeColor = new Color(0xE51400);
    private Color booColor = new Color(0xFA6800);
    private Color caraColor = new Color(0x0050EF);

    private char[] entero = {'0','1','2','3','4','5','6','7','8','9'};
    private String [] decimal = {"."};
    private String[] booleano = {"True", "False"};

    public String validarTipos(String cadena){
        if (cadena == null || cadena.isEmpty()) {
            return "Cadena inválida: La cadena está vacía.";
        }

        if (!esEntero(cadena.charAt(0)) && !esDecimal(cadena) && !
esCadena(cadena) && !esBooleano(cadena) && !
esCaracter(cadena)) {
```

```
            return "Cadena inválida: La cadena no corresponde a ningún tipo de dato válido.";
        }

        return "Cadena válida: " + cadena;
    }

    public Color obtenerColorTipo(String cadena) {
        String resultado = validarTipos(cadena);

        if (resultado.equals("Cadena válida: " + cadena)) {
            if (esDecimal(cadena)) {
                return deciColor;
            } else if (esEntero(cadena.charAt(0))) {
                return enteroColor;
            } else if (esCadena(cadena)) {
                return cadeColor;
            } else if (esBooleano(cadena)) {
                return booColor;
            } else if (esCaracter(cadena)) {
                return caraColor;
            }
        }
        return Color.WHITE;
    }

    public boolean esEntero(char c){
        for (char sigEntero : entero) {
            if (c == sigEntero) {
                return true;
            }
        }
```

```
        return false;
    }



    public boolean esDecimal(String cadena) {
        // Verificar que la cadena contiene exactamente un punto
        int puntoIndex = cadena.indexOf('.');
        if (puntoIndex == -1 || cadena.indexOf('.', puntoIndex + 1) != -
1) {
            return false; // No tiene un punto o tiene más de un punto
        }

        // Verificar que haya al menos un dígito antes y después del
punto
        if (puntoIndex == 0 || puntoIndex == cadena.length() - 1) {
            return false; // No hay dígitos antes o después del punto
        }

        // Verificar que los caracteres antes y después del punto sean
dígitos
        for (int i = 0; i < puntoIndex; i++) {
            if (!Character.isDigit(cadena.charAt(i))) {
                return false; // Caracter antes del punto no es dígito
            }
        }
        for (int i = puntoIndex + 1; i < cadena.length(); i++) {
            if (!Character.isDigit(cadena.charAt(i))) {
                return false; // Caracter después del punto no es dígito
            }
        }
```

```
        return true; // La cadena cumple con todos los requisitos para ser un decimal
    }


    public boolean esCadena(String cadena) {
        return cadena.length() >= 2 && cadena.charAt(0) == "" && cadena.charAt(cadena.length() - 1) == "";
    }

    public boolean esBooleano(String cadena) {
        for (String sigBool : booleano) {
            if (cadena.equalsIgnoreCase(sigBool)) {
                return true;
            }
        }
        return false;
    }

    public boolean esCaracter(String cadena) {

        return cadena.length() == 3 && cadena.charAt(0) == '\'' && cadena.charAt(2) == '\'';
    }
}
```

Token lógico.

```
package Identificadores;

import java.awt.Color;
```

```
/**
 *
 * @author crisa
 */
public class generadorLogico {

    Color yColor = new Color(0x414ED9);
    Color oColor = new Color(0x41D95D);
    Color negaColor = new Color(0xA741D9);



    String Y [] = {"And"};
    String O [] = {"Or"};
    String negacion [] = {"Not"};

    public String validarLogico(String cadena) {
        if (cadena == null || cadena.isEmpty()) {
            return "Cadena inválida: La cadena está vacía.";
        }

        // Validar si la cadena corresponde a un operador de
comparación válido
        if (!esAnd(cadena) && !esOr(cadena) && !esNega(cadena)) {
            return "Cadena inválida: La cadena no es un operador de
comparación válido.";
        }

        return "Cadena válida: " + cadena;
    }

    public Color colorLogico(String cadena) {
```

```
        // Validar la cadena
        String resultado = validarLogico(cadena);

        // Si la cadena es válida, devolver el color correspondiente
        if (resultado.equals("Cadena válida: " + cadena)) {
            if (esAnd(cadena)) {
                return yColor;
            } else if (esOr(cadena)) {
                return oColor;
            } else if (esNega(cadena)) {
                return negaColor;
            }
        }

        return Color.WHITE; // Color por defecto para cadenas
inválidas
    }

    public boolean esAnd (String c){
        for (String sigY : Y) {
            if (c.equals(sigY)) {
                return true;
            }
        }

        return false;
    }

    public boolean esOr (String c){
        for (String sigO : O) {
            if (c.equals(sigO)) {
                return true;
```

```
            }
        }

        return false;
    }

    public boolean esNega (String c){
        for (String sigNega : negacion) {
            if (c.equals(sigNega)) {
                return true;
            }
        }

        return false;
    }
}
```

Token Palabras Reservadas

```
/*
 * Click
nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-
default.txt to change this license
 * Click
nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to
edit this template
 */
package Identificadores;

import java.awt.Color;

/**
```

```
 *
 * @author crisa
 */
public class generadorPalabrasReservadas {

    Color reservadaColor = new Color(0x60A917);

    String palaReservada [] = {"Module", "End", "Sub", "Main",
"Dim", "As",
                        "Integer", "String", "Boolean", "Double", "Char",
                        "Console.WriteLine", "Console.ReadLine", "If",
"ElseIf",
                        "Else", "Then", "While", "Do", "Loop", "For",
"To", "Next",
                        "Function", "Return", "Const"};


    public String validarReservada(String cadena){
        if (cadena == null || cadena.isEmpty()) {
            return "Cadena inválida: La cadena está vacía.";
        }

        if (!esReservada(cadena)) {
            return "Cadena inválida: La cadena no es un operador de
comparación válido.";
        }

        return "Cadena válida: " + cadena;
    }

    public Color colorReservada(String cadena){
        String resultado = validarReservada(cadena);
```

```
        if (resultado.equals("Cadena válida: " + cadena)) {
            if (esReservada(cadena)) {
                return reservadaColor;
            }
        }

        return Color.WHITE;
    }

    public boolean esReservada(String c){
        for (String resePalabra : palaReservada) {
            if (c.equals(resePalabra)) {
                return true;
            }
        }
        return false;
    }
}
```

Generador de automata grafico dependiendo del token que reconozca

```
/*
 * Click
nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-
default.txt to change this license
 * Click
nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to
edit this template
 */
package Automata;
```

```
import java.awt.BorderLayout;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

/**
 *
 * @author alejandro
 */

public class Automata {
    private celdaToken token;

    public Automata(celdaToken token) {
        this.token = token;
    }

    // Método para graficar el autómata usando Graphviz
    public void graficarAutomata() {
        try {
```

```
        System.out.println("Iniciando la generación del autómata para el token: " + token.getToken());

        // Crear el archivo .dot que se usará para generar el gráfico con Graphviz
        FileWriter fileWriter = new FileWriter("automata.dot");

        // Generar el código .dot para representar el autómata del token
        fileWriter.write("digraph G {\n");
        fileWriter.write("rankdir=LR;\n");
        fileWriter.write("node [shape=circle];\n");

        // Crear un nodo por cada carácter del lexema
        String lexema = token.getToken();
        for (int i = 0; i < lexema.length(); i++) {
            String nodoActual = "q" + i;
            String nodoSiguiente = "q" + (i + 1);

            // Último nodo se convertirá en un nodo final
            fileWriter.write(nodoActual + " -> " + nodoSiguiente + " [label=\"" + lexema.charAt(i) + "\"];\n");
            if (i == lexema.length() - 1) {
                fileWriter.write(nodoSiguiente + " [shape=doublecircle];\n");
            }
        }

        fileWriter.write("}\n");
        fileWriter.close();
        System.out.println("Archivo .dot creado correctamente.");
```

```
            // Ejecutar el comando de Graphviz para generar el gráfico en
formato PNG
            Process proceso = Runtime.getRuntime().exec("dot -Tpng
automata.dot -o automata.png");

            // Esperar a que el proceso termine para asegurar que el
archivo esté listo
            int exitCode = proceso.waitFor();
            if (exitCode == 0) {
                System.out.println("Imagen del autómata generada como
automata.png");
            } else {
                System.out.println("Error al generar la imagen del
autómata. Código de salida: " + exitCode);
            }

        } catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    }

    // Mostrar la información del token en una ventana
    public void mostrarInformacion() {
        JFrame ventana = new JFrame("Información del Token");
        ventana.setSize(400, 300);

ventana.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        ventana.setLayout(new BorderLayout());

        JTextArea infoToken = new JTextArea();

        infoToken.setEditable(false);
```

```
        // Botón para graficar el autómata
        JButton botonGraficar = new JButton("Mostrar Autómata");
        botonGraficar.addActionListener(e -> {
            graficarAutomata();
            try {
                mostrarImagenAutomata();
            } catch (IOException ex) {

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

        ventana.add(new JScrollPane(infoToken),
BorderLayout.CENTER);
        ventana.add(botonGraficar, BorderLayout.SOUTH);
        ventana.setVisible(true);
    }


    // Método para mostrar el autómata generado
    private void mostrarImagenAutomata() throws IOException {
        File archivoImagen = new File("automata.png");

        if (archivoImagen.exists()) {
            // Cargar la imagen generada por Graphviz
            BufferedImage imagen = ImageIO.read(archivoImagen);
            ImageIcon iconoImagen = new ImageIcon(imagen);
            JFrame ventanaAutomata = new JFrame("Autómata");
            ventanaAutomata.setSize(500, 500);
```

```
ventanaAutomata.setDefaultCloseOperation(JFrame.DISPOSE_ON
_CLOSE);
            JLabel etiquetaImagen = new JLabel(iconoImagen);
            ventanaAutomata.add(etiquetaImagen);
            ventanaAutomata.setVisible(true);
                } else {
                    // Mostrar un mensaje de error si la imagen no fue
generada
                    JOptionPane.showMessageDialog(null, "Error: no se
pudo generar el autómata.");
                }
    }
}
```

Metodo para mostrar la cuadricula donde se ejecuta el programa

```
package Interfaz;

import Automata.automataIdentificador;
import Automata.celdaToken;
import Identificadores.generadorAritmeticos;
import Identificadores.generadorAsignacion;
import Identificadores.generadorComentario;
import Identificadores.generadorComparacion;
import Identificadores.generadorIdentificadores;
import Identificadores.generadorLogico;
import Identificadores.generadorPalabrasReservadas;
import Identificadores.generadorSignos;
import Identificadores.generadorTiposDato;
import Identificadores.identificadorEspecial;
```

```
import ListaTokens.ListaTokens;
import Reportes.reportePanel;
import Reportes.reporteTokens;
import static
com.sun.java.accessibility.util.AWTEventMonitor.addMouseListener
;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics2D;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.image.BufferedImage;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;
import javax.imageio.ImageIO;
import javax.swing.BorderFactory;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
```

```
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextArea;
import javax.swing.SwingUtilities;
import javax.swing.event.CaretEvent;
import javax.swing.event.CaretListener;

public class cuadriculaInicio extends JFrame {

    private generadorIdentificadores geneIdentificadores = new
generadorIdentificadores();
    private generadorAritmeticos geneAritmeticos = new
generadorAritmeticos();
    private generadorComparacion geneComparacion = new
generadorComparacion();
    private generadorLogico geneLogico = new generadorLogico();
    private generadorAsignacion geneAsignacion = new
generadorAsignacion();
    private generadorPalabrasReservadas geneReservadas = new
generadorPalabrasReservadas();
    private generadorTiposDato geneDato = new
generadorTiposDato();
    private generadorSignos geneSignos = new generadorSignos();
    private generadorComentario geneComentario = new
generadorComentario();
    private ListaTokens listaTokens;
    private JTextArea texto;
    private JPanel panel;
    private int tamaGrid;
    private JLabel estadoCursor;
```

```
    private identificadorEspecial ideEspecial = new
identificadorEspecial();



    public cuadriculaInicio() {
        setTitle("Analizador Lexico");
        setSize(800, 600);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        texto = new JTextArea(10, 30);
        JScrollPane scrollPane = new JScrollPane(texto);
        add(scrollPane, BorderLayout.WEST);

        JPanel topPanel = new JPanel();
        JButton tamaButton = new JButton("Establecer Tamaño");
        tamaButton.addActionListener(new tamaPanel());
        topPanel.add(tamaButton);

        JButton cargarButton = new JButton("Cargar Archivo");
        cargarButton.addActionListener(new CargarArchivoPanel());
        topPanel.add(cargarButton);

        JButton mostrarTokensButton = new JButton("Mostrar
Tokens");
        mostrarTokensButton.addActionListener(new
MostrarTokensPanel());
        topPanel.add(mostrarTokensButton);

        JButton guardarImagen = new JButton("Guardar Imagen");
```

```
        guardarImagen.addActionListener(new
GuardarImagenPanel());
        topPanel.add(guardarImagen);

        add(topPanel, BorderLayout.NORTH);

        panel = new JPanel();
        add(panel, BorderLayout.CENTER);

        texto.addCaretListener(e -> actualizarCuadricula());

        estadoCursor = new JLabel("Línea: 1, Columna: 1");
        estadoCursor.setBorder(BorderFactory.createEtchedBorder());
        add(estadoCursor, BorderLayout.SOUTH);

        texto.addCaretListener(new CaretListener() {
            @Override
            public void caretUpdate(CaretEvent e) {
                try {
                    int pos = texto.getCaretPosition();
                    int row = texto.getLineOfOffset(pos);
                    int col = getColumnInText(texto.getText(), pos);
                    estadoCursor.setText(String.format("Línea: %d,
Columna: %d", row +1 , col ));
                } catch (Exception ex) {
                    ex.printStackTrace();
                }
            }
        });
    }

    private int getColumnInText(String text, int pos) {
```

```
        int col = 1;
        for (int i = 0; i < pos; i++) {
            if (text.charAt(i) == '\n') {
                col = 1;
            } else {
                col++;
            }
        }
        return col;
    }

    private class tamaPanel implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {

            String tamaCuadricula =
JOptionPane.showInputDialog("Ingrese el tamaño de la
cuadrícula:");
            try {
                tamaGrid = Integer.parseInt(tamaCuadricula);
                panel.setLayout(new GridLayout(tamaGrid, tamaGrid));
                panel.revalidate();
                actualizarCuadricula();
            } catch (NumberFormatException ex) {
                JOptionPane.showMessageDialog(null, "Por favor ingrese
un número válido.");
            }
        }
    }

    private class CargarArchivoPanel implements ActionListener {
        @Override
```

```
        public void actionPerformed(ActionEvent e) {
            JFileChooser fileChooser = new JFileChooser();
            int option =
fileChooser.showOpenDialog(cuadriculaInicio.this);

            if (option == JFileChooser.APPROVE_OPTION) {
                File archivo = fileChooser.getSelectedFile();
                try (BufferedReader br = new BufferedReader(new
FileReader(archivo))) {
                    StringBuilder sb = new StringBuilder();
                    String linea;
                    while ((linea = br.readLine()) != null) {
                        sb.append(linea).append("\n");
                    }
                    texto.setText(sb.toString());
                    actualizarCuadricula();
                } catch (IOException ex) {
                    JOptionPane.showMessageDialog(cuadriculaInicio.this,
"Error al leer el archivo", "Error",
JOptionPane.ERROR_MESSAGE);
                }
            }
        }
    }


    private class MostrarTokensPanel implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            String textoEscrito = texto.getText();
            String[] tokens = separarTokens(textoEscrito);
            List<reporteTokens> tokenReports = new ArrayList<>();
```

```
            int line = obtenerFilaTexto(textoEscrito);
            int column = obtenerColumnaTexto(textoEscrito);

            for (String token : tokens) {
                String tipo = identificarTipoToken(token);

                int gridFila = obtenerFilaTexto(token);
                int gridColumn = obtenerColumnaTexto(token);
                Color color = obtenerColorParaToken(tipo);


                reporteTokens report = new reporteTokens(tipo, token,
line, column, gridFila, gridColumn, color);
                tokenReports.add(report);


                line++;
                column++;
            }


            reportePanel.showReport(tokenReports, ((JFrame)
SwingUtilities.getWindowAncestor(texto)));
        }

        private String identificarTipoToken(String token) {
            if(especialToken(token)){
                return "Token_Especial";
            } else if (esComparacion(token)) {
                return "Comparación";
            } else if (esAsignacion(token)) {
```

```
            return "Asignación";
        } else if (esLogico(token)) {
            return "Lógico";
        } else if (esAritmetico(token)) {
            return "Aritmético";
        } else if (esReservada(token)) {
            return "Palabra_Reservada";
        } else if (esTipoDato(token)) {
            return "Tipo de Dato";
        } else if (esSigno(token)) {
            return "Signo";
        } else if (esComentario(token)) {
            return "Comentario";
        } else {
            return "Identificador";
        }

    }



}

private void actualizarCuadricula() {
    panel.removeAll();

    String textoEscrito = texto.getText();
    String[] tokens = separarTokens(textoEscrito);

    int cuadrosTotales = tamaGrid * tamaGrid;
    int tokenIndex = 0;
```

```
        List<String> tokensPintados = new ArrayList<>(); // Lista para almacenar tokens ya pintados

        for (int i = 0; i < cuadrosTotales; i++) {
            String token = tokenIndex < tokens.length ? tokens[tokenIndex] : null;
            Color color = (token != null) ? obtenerColorParaToken(token) : Color.WHITE;

            // Ajustar los cálculos para obtener la fila y columna en la cuadrícula
            int filaTexto = obtenerFilaTexto(token);
            int columnaTexto = obtenerColumnaTexto(token);

            // Convertir la fila y columna del texto a las coordenadas de la cuadrícula
            int gridFila = (tokenIndex / tamaGrid);
            int gridColumna = (tokenIndex % tamaGrid);

            celdaToken celda = new celdaToken(token, color, filaTexto, columnaTexto, gridFila, gridColumna);

            panel.add(celda);
            tokenIndex++;
        }

        panel.revalidate();
        panel.repaint();
    }
```

```
    // Implementar el método obtenerColorParaToken() para
determinar el color del token
    private Color obtenerColorParaToken(String token) {
        if (especialToken(token)) {
            return Color.decode(getColorHexFromToken(token)); // Usar
el color extraído del token especial
        }
        if (esComparacion(token)) return
geneComparacion.colorComparacion(token);
        if (esAsignacion(token)) return
geneAsignacion.colorAsignacion(token);
        if (esLogico(token)) return geneLogico.colorLogico(token);
        if (esAritmetico(token)) return
geneAritmeticos.colorAritmetico(token);
        if (esReservada(token)) return
geneReservadas.colorReservada(token);
        if (esTipoDato(token)) return
geneDato.obtenerColorTipo(token);
        if (esSigno(token)) return geneSignos.colorSigno(token);
        if (esComentario(token)) return
geneComentario.comentarioColor(token);
        else{
            return geneIdentificadores.obtenerColorParaCadena(token);
        }
        /*
        else{


        }
        */

    }
```

```
private int obtenerFilaTexto(String token) {
    if (token == null) return -1;

    String textoEscrito = texto.getText();
    int index = textoEscrito.indexOf(token);

    if (index == -1) {
        return -1; // Token no encontrado
    }

    // Contar el número de saltos de línea antes del índice del token
    int fila = 1;
    for (int i = 0; i < index; i++) {
        if (textoEscrito.charAt(i) == ' ') {
            fila++;
        }
    }
    return fila;
}

private int obtenerColumnaTexto(String token) {
    if (token == null) return -1;

    String textoEscrito = texto.getText();
    int index = textoEscrito.indexOf(token);

    if (index == -1) {
        return -1; // Token no encontrado
    }
```

```
    // Buscar el último salto de línea antes del índice del token
    int lastNewLineIndex = textoEscrito.lastIndexOf(' ', index);
    int columna = index - lastNewLineIndex;

    return columna + 1; // La columna es 1-based
}




private boolean esAritmetico(String token) {
    return geneAritmeticos.esSuma(token.charAt(0)) ||
            geneAritmeticos.esResta(token.charAt(0)) ||
            geneAritmeticos.esExpo(token.charAt(0)) ||
            geneAritmeticos.esDivision(token.charAt(0)) ||
            geneAritmeticos.esMod(token) ||
            geneAritmeticos.esMulti(token.charAt(0));
}

private boolean esComparacion(String token) {
    return geneComparacion.esIgual(token) ||
            geneComparacion.esDife(token) ||
            geneComparacion.esMayor(token.charAt(0)) ||
            geneComparacion.esMenor(token.charAt(0)) ||
            geneComparacion.esMayIgua(token) ||
            geneComparacion.esMenIgua(token);
}

private boolean esLogico(String token) {
    return geneLogico.esAnd(token) ||
```

```
            geneLogico.esOr(token) ||
            geneLogico.esNega(token);
    }

    private boolean esAsignacion(String token){
        return geneAsignacion.esSimple(token.charAt(0)) ||
            geneAsignacion.esCompuesta(token);
    }

    public boolean esReservada(String token){
        return geneReservadas.esReservada(token);
    }

    private boolean esTipoDato(String token){
        return geneDato.esEntero(token.charAt(0)) ||
            geneDato.esDecimal(token) ||
            geneDato.esCadena(token) ||
            geneDato.esBooleano(token) ||
            geneDato.esCaracter(token);
    }

    private boolean esSigno(String token){
        return geneSignos.esParentesis(token) ||
            geneSignos.esLlave(token) ||
            geneSignos.esCorchete(token) ||
            geneSignos.esComa(token.charAt(0)) ||
            geneSignos.esPunto(token.charAt(0));
    }

    private boolean esComentario(String token){
        return geneComentario.esComentario(token);
    }
```

```
    private boolean especialToken(String token){
        return ideEspecial.esTokenEspecial(token);
    }

    private String getColorHexFromToken(String token) {
        String contenido = token.substring("Square.Color(".length(),
token.length() - 1);
        return contenido;
    }


    public String[] separarTokens(String texto) {
        int tamaCadena = texto.length();
        List<String> tokens = new ArrayList<>();
        StringBuilder tokenBuilder = new StringBuilder();
        boolean dentroDeNumero = false;  // Bandera para controlar
cuando estamos dentro de un número

        for (int i = 0; i < tamaCadena; i++) {
            char c = texto.charAt(i);

            if (Character.isWhitespace(c)) {
                if (tokenBuilder.length() > 0) {
                    tokens.add(tokenBuilder.toString());
                    tokenBuilder.setLength(0);
                    dentroDeNumero = false;
                }
            } else if (esDelimitador(c)) {
                if (tokenBuilder.length() > 0) {
                    tokens.add(tokenBuilder.toString());
                    tokenBuilder.setLength(0);
```

```
                dentroDeNumero = false;
            }
            if (c == '.' && dentroDeNumero && i + 1 < tamaCadena
&& Character.isDigit(texto.charAt(i + 1))) {
                tokenBuilder.append(c);
            } else {
                tokens.add(String.valueOf(c));
            }
        } else {
            // Detectar inicio de token especial
            if (esInicioTokenEspecial(texto, i)) {
                int finTokenEspecial =
encontrarFinTokenEspecial(texto, i);
                tokens.add(texto.substring(i, finTokenEspecial + 1));
                i = finTokenEspecial;
            } else {
                tokenBuilder.append(c);

                if (Character.isDigit(c) || (c == '.' &&
dentroDeNumero)) {
                    dentroDeNumero = true;
                }

                if (i + 1 >= tamaCadena || esDelimitador(texto.charAt(i
+ 1)) || Character.isWhitespace(texto.charAt(i + 1))) {
                    tokens.add(tokenBuilder.toString());
                    tokenBuilder.setLength(0);
                    dentroDeNumero = false;
                }
            }
        }
    }
```

```
        if (tokenBuilder.length() > 0) {
            tokens.add(tokenBuilder.toString());
        }

        return tokens.toArray(new String[0]);
    }

    private boolean esInicioTokenEspecial(String texto, int index) {
        String tokenEspecial = "Square.Color(";
        int longitud = tokenEspecial.length();
        if (index + longitud <= texto.length()) {
            return texto.substring(index, index +
longitud).equals(tokenEspecial);
        }
        return false;
    }

    private int encontrarFinTokenEspecial(String texto, int index) {
        // Buscar el final del token especial, suponiendo que siempre
termina con ')'
        for (int i = index; i < texto.length(); i++) {
            if (texto.charAt(i) == ')') {
                return i;
            }
        }
        return texto.length() - 1;  // Si no se encuentra, regresar el final
del texto
    }

    private boolean esDelimitador(char c) {
        return !(Character.isLetter(c) || Character.isDigit(c) || c == '_')
```

```
            && (esSigno(String.valueOf(c)) ||
esAritmetico(String.valueOf(c))
            || esComparacion(String.valueOf(c)) ||
esLogico(String.valueOf(c)) || esAsignacion(String.valueOf(c)))
            && c != '.';
    }




    /*
    public String[] separarTokens(String texto) {
        int tamaCadena = texto.length();
        List<String> tokens = new ArrayList<>();
        StringBuilder tokenBuilder = new StringBuilder();

        boolean enToken = false;

        for (int i = 0; i < tamaCadena; i++) {
            char c = texto.charAt(i);

            if (Character.isWhitespace(c)) {
                if (enToken) {
                    tokens.add(tokenBuilder.toString());
                    tokenBuilder.setLength(0);
                    enToken = false;
                }
            } else {
                tokenBuilder.append(c);
                enToken = true;
            }
```

```
        }

        if (enToken) {
            tokens.add(tokenBuilder.toString());
        }

        return tokens.toArray(new String[0]);
    }
    */
    private class GuardarImagenPanel implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            BufferedImage image = new
BufferedImage(panel.getWidth(), panel.getHeight(),
BufferedImage.TYPE_INT_RGB);
            Graphics2D g2d = image.createGraphics();
            panel.paint(g2d);
            g2d.dispose();

            JFileChooser fileChooser = new JFileChooser();
            fileChooser.setDialogTitle("Guardar Imagen");
            fileChooser.setFileFilter(new
javax.swing.filechooser.FileNameExtensionFilter("Imagenes JPG y
PNG", "jpg", "png"));

            int sleccion =
fileChooser.showSaveDialog(cuadriculaInicio.this);
            if (sleccion == JFileChooser.APPROVE_OPTION) {
                File fileToSave = fileChooser.getSelectedFile();
                String filePath = fileToSave.getPath();
```

```
            String extension = extensionImagen(filePath);
            String formatName = "jpg";
            if (extension.equals("png")) {
                formatName = "png";
            }

            try {
                ImageIO.write(image, formatName, fileToSave);
                JOptionPane.showMessageDialog(cuadriculaInicio.this,
"Imagen guardada con éxito", "Éxito",
JOptionPane.INFORMATION_MESSAGE);
            } catch (IOException ex) {
                JOptionPane.showMessageDialog(cuadriculaInicio.this,
"Error al guardar la imagen", "Error",
JOptionPane.ERROR_MESSAGE);
            }
        }
    }

    private String extensionImagen(String filePath) {
        String fileName = new File(filePath).getName();
        int dotIndex = fileName.lastIndexOf('.');
        if (dotIndex > 0 && dotIndex < fileName.length() - 1) {
            return fileName.substring(dotIndex + 1).toLowerCase();
        }
        return "";
    }
}

}
```