UNIVERSIDAD MARIANO GÁLVEZ DE GUATEMALA
FACULTAD DE INGENIERÍA EN SISTEMAS DE INFORMACIÓN
AUTOMATAS Y LENGUAJES FORMALES
ING. JOSSEPH EMMANUEL TURNIL MURGA

PROYECTO 1

Analizador Léxico

Cristopher Augusto Arana Charuc 7690-19-13982
Esdras Wilfredo Pérez Coloma 7690-14-6737

GUATEMALA 14 DE OCTUBRE, 2023

CODIGO FUENTE

```
import java.util.ArrayList;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import javax.swing.*;

import javax.swing.table.DefaultTableModel;


public class AnalizadorLexico {

    public static void main(String[] args) {

        SwingUtilities.invokeLater(() -> {

            entornoGrafico();

        });

    }


    private static void entornoGrafico() {

        JFrame frame = new JFrame("Analizador Léxico de una clase de C#");

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.setSize(1500, 500);


        JPanel panel = new JPanel();

        JLabel label = new JLabel("Definición de Clase:");

        JTextArea inputArea = new JTextArea(15, 40);

        JButton botonAnalizar = new JButton("Analizar");

        JTable tokenTable = new JTable(new DefaultTableModel(new Object[]{"LEXEMA",
"PATRÓN"}, 0));

        JLabel classNameLabel = new JLabel("Nombre de la Clase:");

        JTextField classNameField = new JTextField(15);

        JTable attributeTable = new JTable(new DefaultTableModel(new
Object[]{"ACCESIBILIDAD", "NOMBRE ATRIBUTO", "TIPO DATO"}, 0));
```

```
    panel.add(label);
    panel.add(new JScrollPane(inputArea));
    panel.add(botonAnalizar);
    panel.add(tokenTable);
    panel.add(classNameLabel);
    panel.add(classNameField);
    panel.add(attributeTable);

    frame.add(panel);
    frame.setVisible(true);

    botonAnalizar.addActionListener(e -> {
        String inputText = inputArea.getText();
        ClaseDefinicion classDefinition = parseClassDefinition(inputText);
        mostrarTokens(tokenTable, classDefinition.tokens);
        classNameField.setText(classDefinition.className);
        mostrarAtributos(attributeTable, classDefinition.attributes);
    });
}

private static ClaseDefinicion parseClassDefinition(String input) {
    ClaseDefinicion classDefinition = new ClaseDefinicion();
    classDefinition.tokens = new ArrayList<>();

    // Definicion de  expresiones regulares para patrones.
    String classPattern = "\\bclass\\b";
    String identifierPattern = "[a-zA-Z][a-zA-Z0-9_]*";
```

```
        String openBracePattern = "\\{";

        // Compilar expresiones regulares
        Pattern classRegex = Pattern.compile(classPattern);
        Pattern identifierRegex = Pattern.compile(identifierPattern);
        Pattern openBraceRegex = Pattern.compile(openBracePattern);

        // Token entrada
        String[] lines = input.split("\n");
        for (String line : lines) {
            Matcher classMatcher = classRegex.matcher(line);
            Matcher identifierMatcher = identifierRegex.matcher(line);
            Matcher openBraceMatcher = openBraceRegex.matcher(line);

            while (classMatcher.find()) {
                classDefinition.tokens.add(new Token(classMatcher.group(), "Palabra reservada class"));
            }

            while (identifierMatcher.find()) {
                classDefinition.tokens.add(new Token(identifierMatcher.group(), "Identificador"));
                classDefinition.className = identifierMatcher.group();
            }

            while (openBraceMatcher.find()) {
                classDefinition.tokens.add(new Token(openBraceMatcher.group(), "Símbolo llave abre"));
            }
        }
```

```
        // Extraer atributos (suponiendo que los atributos sigan patrones específicos)
        String[] attributeLines = input.split(";");
        classDefinition.attributes = new ArrayList<>();
        for (String attributeLine : attributeLines) {
            String[] parts = attributeLine.trim().split("\\s+");
            if (parts.length >= 3) {
                String accessibility = parts[3];
                String attributeName = parts[1];
                String dataType = parts[2];
                classDefinition.attributes.add(new Atributos(accessibility, attributeName,
dataType));
            }
        }

        return classDefinition;
    }

    private static void mostrarTokens(JTable table, ArrayList<Token> tokens) {
        DefaultTableModel model = (DefaultTableModel) table.getModel();
        model.setRowCount(0);
        for (Token token : tokens) {
            model.addRow(new Object[]{token.lexeme, token.pattern});
        }
    }

    private static void mostrarAtributos(JTable table, ArrayList<Atributos> attributes) {
        DefaultTableModel model = (DefaultTableModel) table.getModel();
```

```
        model.setRowCount(0);
        for (Atributos attribute : attributes) {
            model.addRow(new Object[]{attribute.name, attribute.accessibility,
attribute.dataType});
        }
    }

    private static class Token {
        String lexeme;
        String pattern;

        Token(String lexeme, String pattern) {
            this.lexeme = lexeme;
            this.pattern = pattern;
        }
    }

    private static class Atributos {
        String accessibility;
        String name;
        String dataType;

        Atributos(String accessibility, String name, String dataType) {
            this.accessibility = accessibility;
            this.name = name;
            this.dataType = dataType;
        }
    }
```

```
    private static class ClaseDefinicion {
        ArrayList<Token> tokens;
        String className;
        ArrayList<Atributos> attributes;
    }
}
```