



INSTITUTO TECNOLÓGICO DE NUEVO LEÓN

Ingeniería en sistemas computacionales

Lenguaje y Autómatas 2

Proyecto Final

Catedrático

Juan Pablo Rosas Baldazo

Alumno

Axel Johnary Liñan Estrada

Mario Humberto Uriegas De León

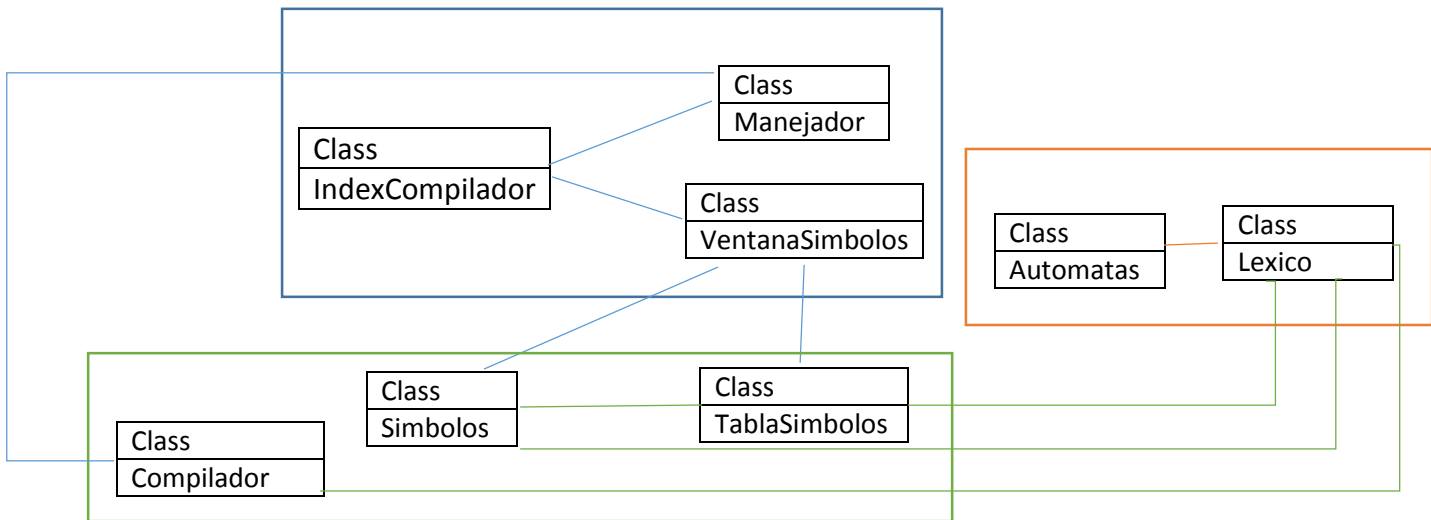
Rafael Salazar Rodríguez

01/06/2018

Introducción

Este proyecto consiste en tomar un archivo .txt con un código en java y que sea capaz de analizarlo línea por línea incluso leer cada palabra por medio de tokens e imprime como resultado un .txt de la salida, además de su comparación con la tabla de símbolos.

Diagrama de clases.



Donde ■ Representa el paquete gui

Donde ■ Representa el paquete compilador

Donde ■ Representa el paquete analizador.lexico

//Clase encargada de generar la interfaz gráfica y mostrar la ventana de símbolos por medio de un botón, además de la entrada y salida de archivos .txt

```
public class IndexCompilador extends JFrame{  
    private static final long serialVersionUID = 3345263108329418543L;
```

```
    private JLabel labelTextoEntrada;  
    private JLabel labelTextoAnalizar;  
    private JTextArea textArchivoEntrada;  
    private JScrollPane scrollEntrada;  
    private JTextArea textArchivoSalida;  
    private JScrollPane scrollSalida;  
    private JButton botonCargarArchivo;  
    private JButton botonAnalizarArchivo;
```

```

//Botón ventana tabla símbolos

private JButton botonTablaSimbolos;

public IndexCompilador(){
    iniciarComponentes();
    asignarDimensiones();
    adicionarObjetos(labelTextoEntrada);
    adicionarObjetos(scrollEntrada);
    adicionarObjetos(botonCargarArchivo);
    adicionarObjetos(labelTextoAnalizar);
    adicionarObjetos(scrollSalida);
    adicionarObjetos(botonAnalizarArchivo);
    adicionarObjetos(botonTablaSimbolos);
    Manejador manejador = new Manejador(this);
    botonCargarArchivo.addActionListener(manejador);
    botonAnalizarArchivo.addActionListener(manejador);
}

public void iniciarComponentes(){
    labelTextoEntrada = new JLabel("Texto de Entrada");
    labelTextoAnalizar = new JLabel("Texto de Salida");
    textArchivoEntrada = new JTextArea();
    textArchivoSalida = new JTextArea();
    botonCargarArchivo = new JButton("Cargar Archivo");
    botonAnalizarArchivo = new JButton("Analizar");
    //Asignar botón Tabla de Símbolos
    botonTablaSimbolos = new JButton("Tabla de Símbolos");
    botonTablaSimbolos.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            showSymbolDataTableActionPerformed(e);
        }
    }
}

```

```

    });

    scrollEntrada = new JScrollPane(textArchivoEntrada);

    scrollSalida = new JScrollPane(textArchivoSalida);

}

public void asignarDimensiones(){

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

    setTitle("Analizador Léxico");

    setResizable(false);

    setSize(650,500);

    setLocation(100,200);

    setDefaultLookAndFeelDecorated(true);

    getContentPane().setLayout( null );

    labelTextoEntrada.setBounds(10,10,100,10);

    scrollEntrada.setBounds(10,25,300,400);

    textArchivoEntrada.setEditable(false);

    botonCargarArchivo.setBounds(100,430,120,30);

    labelTextoAnalizar.setBounds(340,10,100,10);

    scrollSalida.setBounds(340,25,300,400);

    textArchivoSalida.setEditable(false);

    botonAnalizarArchivo.setBounds(420,430,120,30);

    botonTablaSimbolos.setBounds(250,430,140,30);

}

public void showSymbolDataTableActionPerformed(ActionEvent ev) {

    new VentanaSimbolos(this, true).setVisible(true);

}

public void adicionarObjetos(Component component){

    getContentPane().add(component);

}

```

```

    public JTextArea getTextArchivoEntrada() {
        return textArchivoEntrada;
    }

    public JTextArea getTextArchivoSalida() {
        return textArchivoSalida;
    }

    public static void main(String args[]){
        new IndexCompilador().setVisible(true);
    }
}

```

//Clase encargada del manejo de archivos por medio de su relación con la clase IndexCompilador y compilador, es aquella que permite cargar los archivos .txt y la salida del archivo .txt.

```

public class Manejador implements ActionListener{
    private IndexCompilador indexCompilador;

    private JTextArea textArchivoEntrada;

    private JTextArea textArchivoSalida;

    private Compilador compilador;

    private boolean readyFile = false;

    public Manejador(IndexCompilador indexCompilador) {
        this.indexCompilador = indexCompilador;

        textArchivoEntrada = indexCompilador.getTextArchivoEntrada();

        textArchivoSalida = indexCompilador.getTextArchivoSalida();
    }

    public void actionPerformed(ActionEvent event) {
        if(((JButton)event.getSource()).getText().equalsIgnoreCase("Cargar Archivo"))
            cargarArchivo();

        if(((JButton)event.getSource()).getText().equalsIgnoreCase("Analizar"))
            analizar();
    }
}

```

```

private void cargarArchivo(){
    File file = new File("");
    JFileChooser chooser = new JFileChooser();
    chooser.showOpenDialog(indexCompilador);
    file = chooser.getSelectedFile();
    if(file != null){
        compilador = new Compilador(file, this);
        this.readyFile = true;
    }
}

private void analizar(){
    if(readyFile)
        compilador.analizar();
}

public void setTextEntrada(String string){
    this.textArchivoEntrada.setText(string);
    this.textArchivoSalida.setText("");
}

public void setTextSalida(String string){
    this.textArchivoSalida.setText(string);
}
}

```

//Clase que permite crear la ventana con los símbolos utilizados, además de identificar estos símbolos pues esta cuenta con una relación con las clases Simbolos y TablaSimbolosBase que son las que marcan la pauta para la separación de las palabras

```

public class VentanaSimbolos extends JDialog{
    private static final long serialVersionUID = 1L;

    private JScrollPane jcScrollPane;

```

```
private JTable symbolDataTable;

private DefaultTableModel model;


public VentanaSimbolos(JFrame parent, boolean modal) {
    super(parent, modal);
    super.setTitle("Tabla de Símbolos");
    initComponents();
    loadSymbolDataTable();
}


public void initComponents() {
    setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
    setResizable(false);
    setSize(400, 500);
    setLocation(100, 200);
    setDefaultCloseOperationLookAndFeelDecorated(true);
    getContentPane().setLayout(null);
    symbolDataTable = new JTable();
    symbolDataTable.setEnabled(false);
    jcScrollPane = new JScrollPane(symbolDataTable);
    jcScrollPane.setViewportViewView(symbolDataTable);
    jcScrollPane.setBounds(10, 25, 375, 430);
    adicionarObjetos(jcScrollPane);
}


public void adicionarObjetos(Component component) {
    getContentPane().add(component);
}
```

```

public void loadSymbolDataTable() {
    String titles[] = { "Token", "Lexema", "Palabra Reservada" };
    String data[][] = new String[0][3];
    model = new DefaultTableModel(data, titles);
    symbolDataTable.setModel(model);

    ArrayList<Simbolo> listaInicial =
TablaSimboloBase.getInstance().getSimbolosInicial();

    //Permite analizar todas las palabras if (!listaInicial.isEmpty()) {
        for (Simbolo simbolo : listaInicial) {
            String row[] = { simbolo.getToken(), simbolo.getLexema(),
(simbolo.isPalabraReservada() ? "Yes" : "No" ) };
            model.addRow(row);
        }
    }
}

//Clase que establece el código para el manejador de archivos en la clase Manejador y la clase
Lexico para determinar mediante la tabla que es la palabra leyéndolas mediante ciclos

public class Compilador {
    private Manejador manejador;

    private String pathSalida = "";
    private ArrayList<String> lineas = new ArrayList<String>();

    public Compilador(File archivoEntrada, Manejador manejador) {
        this.manejador = manejador;
        this.pathSalida = archivoEntrada.getParent() + "/salida.txt";
        this.cargarArchivo(archivoEntrada);
    }

    private void cargarArchivo(File archivoEntrada) {

```



```

FileReader reader = null;

BufferedReader bufferedReader = null;

try {

    reader = new FileReader(archivoEntrada);

    bufferedReader = new BufferedReader(reader);

    while (bufferedReader.ready()) {

        this.lineas.add(bufferedReader.readLine());

    }

} catch (Exception e) {

    e.printStackTrace();

} finally {

    if (bufferedReader != null)

        try {

            bufferedReader.close();

        } catch (IOException e) {

            e.printStackTrace();

        }

    if (null != reader)

        try {

            reader.close();

        } catch (IOException e) {

            e.printStackTrace();

        }

}

String textEntrada = "";

int index = 0;

for (String linea : this.lineas) {

    textEntrada += index + " " + linea + "\n";

    index++;

```

```

    }

    manejador.setTextEntrada(textEntrada);
}

public void analizar() {
    lexico();
}

private void lexico() {
    Lexico lexico = new Lexico();
    String logSalida = "";
    int index = 0;
    for (String linea : this.lineas) {
        lexico.analizarLinea(linea);
        logSalida += index + " " + linea + "\n";
        logSalida += lexico.getLogSalida();
        index++;
    }
    manejador.setTextSalida(logSalida);
    FileWriter fileWriter = null;
    try {
        fileWriter = new FileWriter(this.pathSalida);
        fileWriter.write(logSalida);
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (null != fileWriter) {
            try {
                fileWriter.close();
            }

```

```
public class Simbolo {  
    private String token;  
    private String lexema;  
    private boolean palabraReservada;  
  
    public Simbolo(String token, String lexema, boolean palabraReservada) {  
        this.token = token;  
        this.lexema = lexema;  
        this.palabraReservada = palabraReservada;  
    }  
    public String getToken() {  
        return token;  
    }  
  
    public void setToken(String token) {  
        this.token = token;  
    }  
    public String getLexema() {  
        return lexema;  
    }  
}
```

```

    public void setLexema(String lexema) {
        this.lexema = lexema;
    }

    public boolean isPalabraReservada() {
        return palabraReservada;
    }

    public void setPalabraReservada(boolean palabraReservada) {
        this.palabraReservada = palabraReservada;
    }
}

//Clase que permite crea el respectivo significado de la palabra en cuestión aquí se establece el
significado para que compilador y símbolo puedan trabajar

public class TablaSimboloBase {
    private static TablaSimboloBase INSTANCE;

    public static TablaSimboloBase getInstance() {
        if (null == INSTANCE) {
            INSTANCE = new TablaSimboloBase();
        }
        return INSTANCE;
    }

    private ArrayList<Simbolo> simbolosInicial;

    public TablaSimboloBase() {
        simbolosInicial = new ArrayList<Simbolo>(
            Arrays.asList(new Simbolo("pro", "programa", true), new
Simbolo("int", "int", true),

```

```

Simbolo("float", "float", true),
new Simbolo("char", "char", true), new
Simbolo("imp", "imprimir", true),
new Simbolo("leer", "leer", true), new
true), new Simbolo("*", "*", true),
new Simbolo("+", "+", true), new Simbolo("-", "-",
true), new Simbolo("/", "/", true), new Simbolo("=", "=",
true), new Simbolo("ter", "terminar", true),
new Simbolo("min", "mientras", true), new
Simbolo("si", "si", true),
new Simbolo("sino", "sino", true), new
Simbolo("\'", "\'", true), new Simbolo(";", ";", true),
new Simbolo(";", ";", true), new Simbolo("(", "(",
true), new Simbolo(")", ")", true),
new Simbolo("{", "{", true), new Simbolo("}", "}"),
true), new Simbolo("&", "&", true),
new Simbolo("&&", "&&", true), new Simbolo("|",
"|", true), new Simbolo("||", "||", true));
}

```

```

public ArrayList<Simbolo> getSimbolosInicial() {
    return simbolosInicial;
}

```

```

public void setSimbolosInicial(ArrayList<Simbolo> simbolosInicial) {
    this.simbolosInicial = simbolosInicial;
}

```

```

public Simbolo getSimboloByLexema(String lexema){
    for(Simbolo simbolo : this.simbolosInicial)
        if(simbolo.getLexema().equalsIgnoreCase(lexema))
            return simbolo;
}

```

```

        return null;
    }

    public boolean isLexemaSimbolo(String lexema){
        for(Simbolo simbolo : this.simbolosInicial)
            if(simbolo.getLexema().equalsIgnoreCase(lexema) &&
simbolo.isPalabraReservada())
                return true;
        return false;
    }

    public boolean existLexemaId(String lexema) {
        for (Simbolo simbolo : this.simbolosInicial) {
            if ((lexema.equalsIgnoreCase(simbolo.getLexema())) &&
(!simbolo.isPalabraReservada())) {
                return true;
            }
        }
        return false;
    }
}

//Clase que permite determinar si el string es una palabra, numero o numero real
public class Automatas {
    public static boolean isIdentificador(String lexema) {
        String letra = "[A-Za-z]";
        String digitoLetra = "[0-9A-Za-z]";
        String caracter = "";
        int estado = 1;
        for (int i = 0; i < lexema.length(); i++) {
            caracter = lexema.charAt(i) + "";

```

```

        switch (estado) {
        case 1:
            if (caracter.matches(letra))
                estado = 2;
            else
                estado = 3;
            break;
        case 2:
            if (caracter.matches(digitoLetra))
                estado = 2;
            else
                estado = 3;
            break;
        }
    }
    if (estado != 3)
        return true;
    return false;
}

```

```

public static boolean isNumero(String lexema) {
    String digito = "[0-9]";
    String caracter = "";
    int estado = 1;
    for (int i = 0; i < lexema.length(); i++) {
        caracter = lexema.charAt(i) + "";
        switch (estado) {
        case 1:
            if (caracter.matches(digito))

```

```

                estado = 1;
            else
                estado = 2;
            break;
        }
    }
    if (estado != 2)
        return true;
    return false;
}

```

```

public static boolean isReal(String lexema) {
    String character = "";
    int estado = 1;
    for (int i = 0; i < lexema.length(); i++) {
        character = lexema.charAt(i) + "";
        switch (estado) {
            case 1:
                if (Automatas.isNumero(character))
                    estado = 1;
                else {
                    estado = 2;
                    i--;
                }
                break;
            case 2:
                if (character.equalsIgnoreCase("."))
                    if (i + 1 != lexema.length())
                        estado = 3;

```



```

        else
            estado = 4;
        else
            estado = 4;
        break;
    case 3:
        if (Automatas.isNumero(caracter)) {
            estado = 3;
        } else
            estado = 4;
        break;
    }
}
if (estado != 4)
    return true;
return false;
}
}

```

//Clase relacionada con Simbolos y TablSimbolosBase que lee las líneas para así poder revisar que representan en nuestra tabla

```

public class Lexico {
    private ArrayList<Simbolo> simbolos = new ArrayList<Simbolo>();

    private String logSalida = "";

    public void analizarLinea(String linea) {
        simbolos.clear();
        logSalida = "";
        char[] caracteres = linea.toCharArray();
    }
}

```

```

String lexema = "";
boolean flagChar = false;
for (int i = 0; i < caracteres.length; i++) {
    String character = caracteres[i] + "";
    if (flagChar) {
        if (character.equalsIgnoreCase("\\")) {
            flagChar = false;
            Simbolo simbolo = new Simbolo("string", lexema, false);
            this.logSalida += "\tToken: " + simbolo.getToken() + "
Lexema: " + simbolo.getLexema() + "\n";
            this.simbolos.add(simbolo);
            analizarLexema(character);
            lexema = "";
            continue;
        }
        lexema += character;
        continue;
    }
    if (TablaSimboloBase.getInstance().isLexemaSimbolo(character)) {
        if (character.equalsIgnoreCase("\\")) {
            flagChar = true;
        }
        if (!lexema.isEmpty()) {
            analizarLexema(lexema);
        }
        analizarLexema(character);
        lexema = "";
    } else if (character.equalsIgnoreCase(" ")) {
        if (!lexema.isEmpty()) {

```

```

        analizarLexema(lexema);
    }
    lexema = "";
} else {
    lexema += caracter;
}
}
if (!lexema.isEmpty()) {
    analizarLexema(lexema);
}
}

```

```

private void analizarLexema(String lexema) {
    Simbolo simbolo = TablaSimboloBase.getInstance().getSimboloByLexema(lexema);
    if (null == simbolo) {
        if (Automatas.isIdentificador(lexema))
            simbolo = new Simbolo("id", lexema, false);
        else if (Automatas.isNumero(lexema))
            simbolo = new Simbolo("n_int", lexema, false);
        else if (Automatas.isReal(lexema))
            simbolo = new Simbolo("n_float", lexema, false);

        if (!TablaSimboloBase.getInstance().existLexemaId(lexema)) {
            TablaSimboloBase.getInstance().getSimbolosInicial().add(simbolo);
        }
    }
    if (null != simbolo) {
        this.logSalida += "\tToken: " + simbolo.getToken() + " Lexema: " +
simbolo.getLexema() + "\n";
    }
}

```

```

        this.simbolos.add(simbolo);
    } else {
        this.logSalida += "\tERROR: " + lexema + "\n";
    }
}

public ArrayList<Simbolo> getSimbolos() {
    return simbolos;
}

public void setSimbolos(ArrayList<Simbolo> simbolos) {
    this.simbolos = simbolos;
}

public String getLogSalida() {
    return logSalida;
}

public void setLogSalida(String logSalida) {
    this.logSalida = logSalida;
}
}

```

Conclusión

El proyecto si bien no traduce a Python este permite leer líneas y darles cierto significado con una tabla de símbolos, lo que era el objetivo principal de la actividad.