TECNOLÓGICO DE ESTUDIOS SUPERIORES DE JOCOTITLÁN

Sentencia While

Alumnos: Bernal Urbina Leydi Laura (2022150481147)
Estrada Martínez Aarón (2022150480299)
López Piña Andrés (2022150480607)

Materia: Lenguajes y Autómatas I

Docente: M. en T.C**.** Erika López González

Grupo: IC-603

Jocotitlán, Estado de México; 23 abril del 2025

Contenido

Introducción

La construcción de autómatas es un proceso esencial para comprender cómo funcionan los lenguajes regulares. A través de esta técnica, se diseñan modelos llamados Autómatas Finitos Deterministas (AFD) y No Deterministas (AFN) que simulan el comportamiento de una máquina de estado finito.

Estos autómatas se utilizan para representar lenguajes formales, verificar patrones de entrada, validar expresiones regulares y facilitar procesos como el análisis léxico en compiladores.

En esta etapa de aprendizaje, construir autómatas permite al estudiante desarrollar habilidades lógicas y matemáticas, al mismo tiempo que entiende cómo se relacionan los lenguajes con los algoritmos que los procesan.

Esta base es fundamental para avanzar hacia modelos más complejos en cursos posteriores, como los autómatas con pila o las máquinas de Turing.

Marco teórico

Lenguaje Formal:
Es un conjunto de cadenas finitas formadas por símbolos pertenecientes a un alfabeto. Los lenguajes formales se utilizan para definir reglas sintácticas en programación, compiladores y lógica computacional.

Alfabeto (Σ):
Es un conjunto finito y no vacío de símbolos. Por ejemplo, Σ = {0,1} es un alfabeto binario. A partir de este alfabeto se forman las cadenas del lenguaje.

Cadena:
Es una secuencia finita de símbolos del alfabeto. Por ejemplo, si Σ = {a, b}, entonces "ababa" es una cadena válida. La cadena vacía se representa con ε.

Lenguaje Regular:
Es un tipo de lenguaje formal que puede ser descrito mediante una expresión regular, una gramática regular o un autómata finito. Los lenguajes regulares se caracterizan por su sencillez y eficiencia en su análisis.

Autómata:
Es un modelo matemático que simula el comportamiento de una máquina que procesa cadenas de símbolos y decide si pertenecen a un lenguaje determinado. Está compuesto por estados, transiciones, un estado inicial y uno o varios estados de aceptación [1].

Autómata Finito Determinista (AFD):

- Es un autómata donde, para cada combinación de estado y símbolo de entrada, existe una única transición. Se define formalmente como una 5-tupla $(Q, \Sigma, \delta, q_0, F)$, donde:
- Q: conjunto finito de estados
- Σ: alfabeto
- δ: función de transición
- q_0: estado inicial
- F: conjunto de estados finales

Autómata Finito No Determinista (AFN):
Es similar al AFD, pero permite múltiples transiciones desde un mismo estado con un mismo símbolo e incluso transiciones sin leer símbolos (ε-transiciones). Aunque parece más flexible, cualquier AFN puede convertirse en un AFD equivalente.

Función de Transición (δ):
Es la regla que indica a qué estado se debe mover el autómata cuando recibe un símbolo de entrada desde un estado determinado. En el AFD es única; en el AFN puede ser múltiple.

Estado Inicial (q_0):
Es el estado desde el cual el autómata comienza a leer la cadena de entrada.

Estado(s) de Aceptación (F):
Son los estados que, si el autómata termina en ellos después de leer toda la cadena, indican que la cadena ha sido aceptada.

Construcción de Autómatas:
Es el proceso de diseñar un autómata (AFD o AFN) que reconozca un lenguaje específico. Puede realizarse desde una descripción informal del lenguaje, una expresión regular o una gramática regular.

Diagrama de Estados:
Es una representación gráfica de un autómata, donde los círculos representan estados y las flechas las transiciones entre ellos.

Expresión Regular:
Es una notación matemática que describe un lenguaje regular. Cualquier lenguaje representado por una expresión regular puede ser reconocido por un autómata finito [2].

Desarrollo de la practica

1.- Importaciones
Se importan las bibliotecas necesarias para crear la interfaz gráfica (Swing), manipular colores y diseño (AWT), trabajar con expresiones regulares (regex) y aplicar estilos a texto (text).

```
import javax.swing.*;
import java.awt.*;
import java.util.regex.*;
import javax.swing.text.*;
```

Ilustración 1 Importaciones

2.- Definición de clase y atributos

La clase AnalizadorWhile hereda de JFrame, lo que permite crear una ventana de interfaz. Se declaran los elementos de la GUI: un área de texto para escribir código, un botón de análisis, un panel para mostrar resultados, y etiquetas de estado. También se definen los colores que se usarán para resaltar los distintos tipos de tokens del código.

```
public class AnalizadorWhile extends JFrame {
    private JTextPane inputArea;
    private JButton analizarBtn;
    private JPanel resultadoPanel;
    private JLabel statusLabel;
```

Ilustración 2 Definición de clases y atributos

3.- Constructor – panel

Se configura la ventana principal con un título, tamaño fijo, acción al cerrar, y distribución general de los componentes.

```
public AnalizadorWhile() {
    setTitle(title:"Analizador de Sentencia while (Java)");
    setSize(width:800, height:600);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setLayout(new BorderLayout());
```

Ilustración 3 Constructor - panel

4.- Panel superior título e instrucciones

Se crea un panel con una etiqueta de título centrado y una instrucción al usuario para que escriba una sentencia while.

```
// Panel superior con título e instrucciones
JPanel titlePanel = new JPanel(new BorderLayout());
JLabel titleLabel = new JLabel(text:"Analizador de Sentencia while en Java", JLabel.CENTER);
titleLabel.setFont(new Font(name:"Arial", Font.BOLD, size:16));
JLabel instruccionesLabel = new JLabel(text:"Introduce una sentencia while de Java:", JLabel.LEFT);
instruccionesLabel.setBorder(BorderFactory.createEmptyBorder(top:5, left:10, bottom:5, right:10));
titlePanel.add(titleLabel, BorderLayout.NORTH);
titlePanel.add(instruccionesLabel, BorderLayout.SOUTH);
add(titlePanel, BorderLayout.NORTH);
```

Ilustración 4 Panel superior título e instrucciones

5.- Área central del área de código

Se crea un área donde el usuario puede escribir código. Se usa JTextPane porque permite aplicar estilos. Se le añade un DocumentListener que activa el resaltado de sintaxis automáticamente cada vez que el usuario escribe o borra texto.

```
// Área de texto para código
inputArea = new JTextPane();
inputArea.setFont(new Font(name:"Monospaced", Font.PLAIN, size:14));
doc = inputArea.getStyledDocument();

// Listener para colorear sintaxis
inputArea.getDocument().addDocumentListener(new javax.swing.event.DocumentListener() {
    public void insertUpdate(javax.swing.event.DocumentEvent e) { colorearSintaxis(); }
    public void removeUpdate(javax.swing.event.DocumentEvent e) { colorearSintaxis(); }
    public void changedUpdate(javax.swing.event.DocumentEvent e) {}
});
add(new JScrollPane(inputArea), BorderLayout.CENTER);
```

Ilustración 5 Área central del área de código

6.- Panel inferior (estado y botón)

Aquí se agrega un botón que el usuario presiona para iniciar el análisis. La etiqueta statusLabel muestra mensajes como "válido" o "error".

```
// Panel inferior con estado y botón
JPanel bottomPanel = new JPanel(new BorderLayout());
statusLabel = new JLabel(text:" ");
statusLabel.setBorder(BorderFactory.createEmptyBorder(top:5, left:10, bottom:5, right:10));
analizarBtn = new JButton(text:"Analizar");
analizarBtn.setPreferredSize(new Dimension(width:100, height:30));
JPanel buttonPanel = new JPanel();
buttonPanel.add(analizarBtn);
bottomPanel.add(statusLabel, BorderLayout.CENTER);
bottomPanel.add(buttonPanel, BorderLayout.EAST);
add(bottomPanel, BorderLayout.SOUTH);
```

Ilustración 6 Panel inferior estado y botón

7.- Panel lateral para resultados

Este panel muestra los resultados del análisis en forma de "tarjetas". Por ejemplo, se puede mostrar "token: while – palabra reservada".

```
// Panel de resultados
resultadoPanel = new JPanel();
resultadoPanel.setLayout(new BoxLayout(resultadoPanel, BoxLayout.Y_AXIS));
resultadoPanel.setBorder(BorderFactory.createEmptyBorder(top:10, left:10, bottom:10, right:10));
add(new JScrollPane(resultadoPanel), BorderLayout.EAST);
```

Ilustración 7 Panel lateral para resultados

8.- Análisis al presionar el botón

Este método se ejecuta al presionar el botón "Analizar". Primero limpia el panel de resultados. Luego valida si la sentencia while es correcta. Si lo es, llama a desglosarJava() para analizar tokens. Si no, llama a resaltarErrores() para indicar los problemas.

```
private void analizar() {
    resultadoPanel.removeAll();
    String codigo = inputArea.getText().trim();

    boolean esValido = esCodigoJavaValido(codigo);
    statusLabel.setText(esValido ? "Análisis completado exitosamente" : "Error: No es una sentencia while válida");
    statusLabel.setForeground(esValido ? COLOR_ESTADO_OK : COLOR_ESTADO_ERROR);

    if (esValido) {
        desglosarJava(codigo);
    } else {
        resaltarErrores(codigo);
    }

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

Ilustración 8 Analizar al presionar el botón

9.- Verificación básica de sintaxis

Comprueba si hay una estructura básica válida para un while, incluyendo paréntesis y llaves. También descarta errores típicos de otros lenguajes como Python (uso de : o elif).

```
private boolean esCodigoJavaValido(String codigo) {
    if (!codigo.contains(s:"while")) return false;

    // Verificación básica de sintaxis while
    Pattern whilePattern = Pattern.compile(regex:"while\\s*\\(.+\\)\\s*\\{");
    if (!whilePattern.matcher(codigo).find()) return false;

    // Detectar elementos de sintaxis no-Java (pero permitiendo and, or, not)
    return !(codigo.contains(s:":") && !codigo.contains(s:"\"") && !codigo.contains(s:"'"))
        && !codigo.contains(s:"elif");
}
```

Ilustración 9 Verificación básica de sintaxis

10.- Mostrar errores comunes

Agrega tarjetas rojas al panel de resultados si faltan elementos clave como while, (), {} o si se usa sintaxis no válida.

```
private void resaltarErrores(String codigo) {
    if (!codigo.contains(s:"while"))
        agregarTarjeta(titulo:"No se encontró 'while'", descripcion:"Toda sentencia while debe comenzar con la palabra clave 'while'", esValido:false);
    if (codigo.contains(s:":"))
        agregarTarjeta(titulo:"Sintaxis incorrecta: ':'", descripcion:"En Java no se usan los dos puntos ':' para delimitar bloques", esValido:false);
    // Se eliminan las validaciones de 'and', 'or' y 'not' para que sean aceptados
    if (!codigo.contains(s:"(") || !codigo.contains(s:")"))
        agregarTarjeta(titulo:"Faltan paréntesis", descripcion:"En Java, la condición del while debe estar entre paréntesis", esValido:false);
    if (!codigo.contains(s:"{") || !codigo.contains(s:"}"))
        agregarTarjeta(titulo:"Faltan llaves", descripcion:"En Java, el bloque de código del while debe estar entre llaves {}", esValido:false);
}
```

Ilustración 10 Mostrar errores comunes

11.- Creación de tarjetas explicativas

Crea visualmente una “tarjeta” con un título (token detectado) y una descripción. Se usa para explicar tanto tokens válidos como errores.

```
private void agregarTarjeta(String titulo, String descripcion, boolean esValido) {
    JPanel tarjeta = new JPanel(new BorderLayout());
    tarjeta.setBorder(BorderFactory.createLineBorder(Color.GRAY));
    tarjeta.setBackground(esValido ? COLOR_FONDO_TARJETA_OK : COLOR_FONDO_TARJETA_ERROR);

    JLabel tituloLabel = new JLabel("  " + titulo);
    tituloLabel.setFont(new Font(name:"Arial", Font.BOLD, size:12));

    // Colorear según tipo de token
    if (esElementoPalabraReservada(titulo))
        tituloLabel.setForeground(COLOR_PALABRA_RESERVADA);
    else if (titulo.contains(s:"\"") || titulo.contains(s:"'"))
        tituloLabel.setForeground(COLOR_CADENA);
    else if (titulo.matches(regex:"[a-zA-Z_]\\w*"))
        tituloLabel.setForeground(COLOR_VARIABLE);
    else if (titulo.matches(regex:"[+-]?\\d+(\\.\\d+)?"))
        tituloLabel.setForeground(COLOR_NUMERO);
    else if (esOperador(titulo))
        tituloLabel.setForeground(COLOR_OPERADOR);
    else if (esSimbolo(titulo))
        tituloLabel.setForeground(COLOR_SIMBOLO);
    else if (titulo.contains(s:"System.out.println"))
        tituloLabel.setForeground(COLOR_METODO);

    tarjeta.add(tituloLabel, BorderLayout.NORTH);
    tarjeta.add(new JLabel("  " + descripcion), BorderLayout.CENTER);
    tarjeta.setMaximumSize(new Dimension(width:500, height:50));
    resultadoPanel.add(tarjeta);
    resultadoPanel.add(Box.createVerticalStrut(height:5));
}
```

Ilustración 11 Creación de tarjetas explicativas

12.- Análisis detallado del token

Separa el código en fragmentos y analiza uno por uno. Cada token es identificado como palabra reservada, variable, operador, número, etc. Luego se muestra una tarjeta con su significado.

```java
private void desglosarJava(String codigo) {
    // Mejora la expresión regular para capturar operadores lógicos como && y ||
    String[] tokens = codigo.split(regex:"\\s+|(?=[(){};])|(?<=[(){};])|(?=[+\\-/=<>!&|%^])|(?<=[+\\-/=<>!&|%^])");

    boolean tieneWhile = false;
    boolean tieneCondicion = false;
    boolean tieneLlaves = false;
    boolean sintaxisCorrecta = true;

    // Procesamiento especial para && y ||
    String codigoTemp = codigo;
    if (codigoTemp.contains(s:"&&")) {
        agregarTarjeta(titulo:"&&", descripcion:"Operador lógico AND", esValido:true);
    }
    if (codigoTemp.contains(s:"||")) {
        agregarTarjeta(titulo:"||", descripcion:"Operador lógico OR", esValido:true);
    }

    for (String token : tokens) {
        token = token.trim();
        if (token.isEmpty()) continue;

        // Resto del código de análisis de tokens...
        // (igual que antes)
        if (token.equals(anObject:"while")) {
            tieneWhile = true;
            agregarTarjeta(titulo:"while", descripcion:"Palabra reservada que inicia el ciclo.", esValido:true);
        } else if (token.matches(regex:"[a-zA-Z_]\\w*")) {
```

Ilustración 12 Análisis detallado del token

Diagrama de Flujo

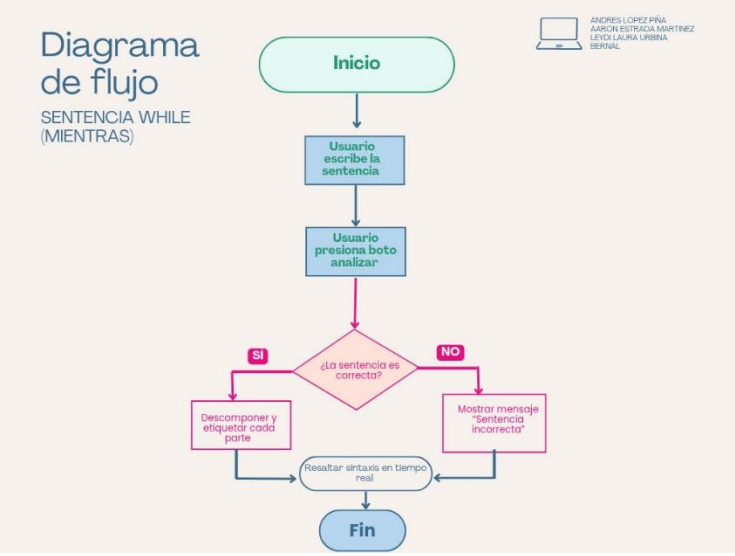

Ilustración 13 Diagrama de Flujo

Resultados

Ejemplo 1: while con una condición simple (normal)

Este bucle while se ejecuta mientras la variable contador sea menor que 5. En cada iteración, imprime el valor actual del contador y luego lo incrementa en uno. Es un ejemplo clásico de un bucle con una condición simple.

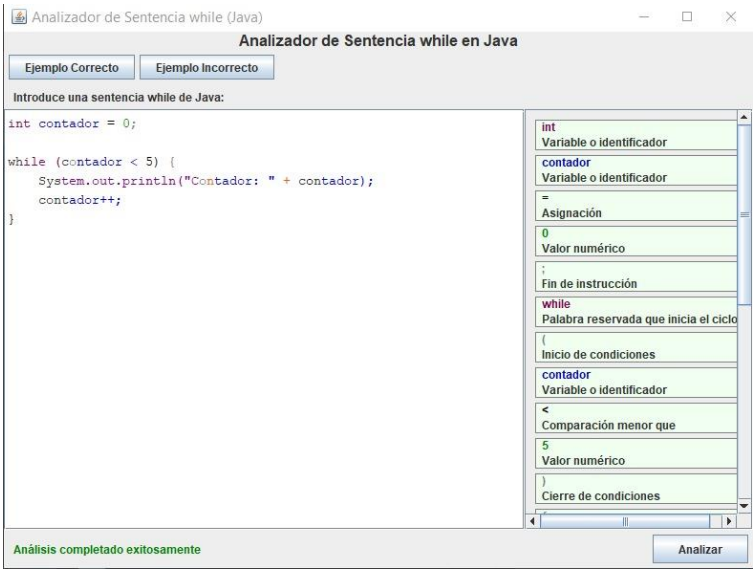

Ilustración 14 Ejemplo 1While con una condición simple

Ejemplo 2: while con and

Se utiliza un bucle while con un operador lógico AND (&&), lo que significa que el ciclo se ejecuta solo cuando ambas condiciones son verdaderas: la variable x debe ser menor que 5 y, al mismo tiempo, la variable y debe ser mayor que 0. En cada vuelta del ciclo, x aumenta en uno y y disminuye en dos. Esto hace que el ciclo se detenga tan pronto como una de las dos condiciones ya no se cumpla, es decir, cuando x ya no sea menor que 5 o y ya no sea mayor que 0.

Ilustración 15 Ejemplo 2: while con and

Ejemplo 3: while con or

Se usa un bucle while con el operador lógico OR (||), lo que significa que el ciclo continuará ejecutándose mientras al menos una de las condiciones sea verdadera. En este caso, se repite mientras a sea menor que 5 o b menor que 10. Dentro del ciclo, ambas variables aumentan en uno en cada iteración. A diferencia del ejemplo anterior, este ciclo puede continuar incluso si una de las condiciones se vuelve falsa, siempre que la otra siga cumpliéndose.

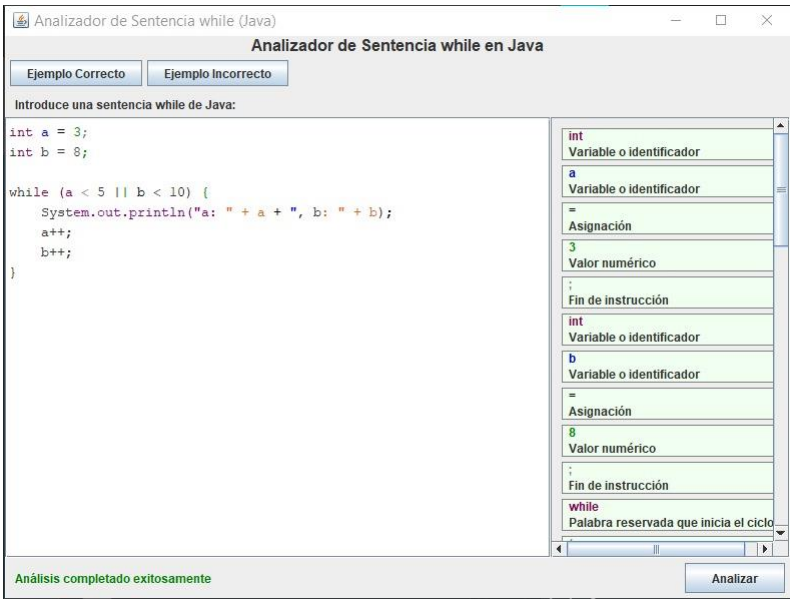

Ilustración 16 Ejemplo 3: while con or

Ejemplo 4: while con 3 condiciones (combinadas)

Muestra un bucle while con tres condiciones combinadas con AND (&&). El ciclo se ejecuta solamente cuando las tres condiciones son verdaderas a la vez: i debe ser menor que 3, j menor que 5 y k menor que 6. En cada iteración, las tres variables se incrementan en uno. El ciclo termina cuando una sola de las tres condiciones deja de cumplirse, lo cual corta la ejecución del bucle.

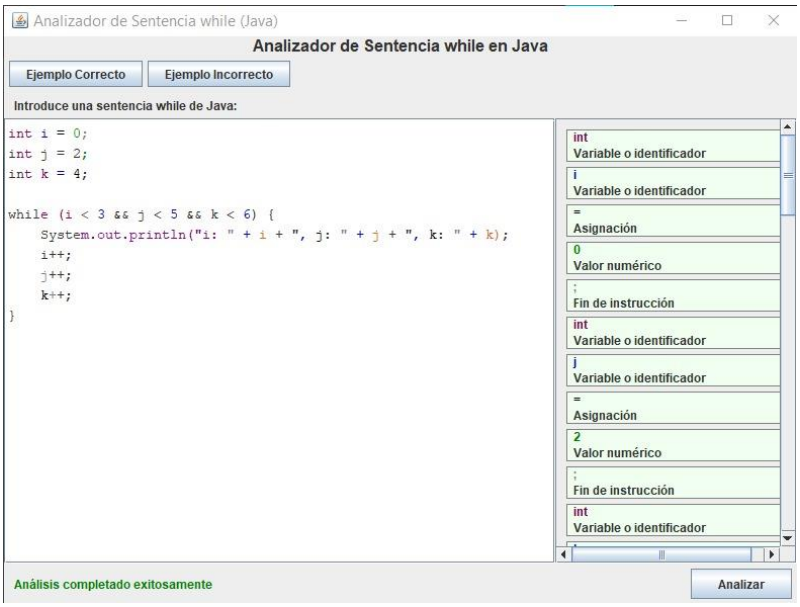

Ilustración 17 Ejemplo 4: while con 3 condiciones (combinadas)

Conclusiones

Este proyecto nos permitió crear una herramienta sencilla pero útil para analizar sentencias while escritas en Java. Al usar una interfaz gráfica, fue posible facilitar la escritura del código, resaltar sus partes más importantes con colores, y detectar errores comunes que los estudiantes suelen cometer.

Con esta práctica, se comprendió mejor cómo está formada una sentencia while, qué elementos necesita para funcionar correctamente y cómo se puede dividir en partes como palabras clave, operadores, paréntesis o llaves. Además, fue una buena manera de aplicar conocimientos de programación en Java, especialmente en la parte visual con botones, paneles y áreas de texto.

Referencias

[1] J. M. C. Lovelle, “CUADERNO DIDÁCTICO No”, Uniovi.es. [En línea]. Disponible en: https://reflection.uniovi.es/ortin/publications/automata.pdf. [Consultado: 22-abr-2025].

[2] L. Altamirano, M. Arias, J. Gonzalez, E. Morales, y G. Rodriguez, “Teor´ıa de Aut´omatas y Lenguajes Formales”, Inaoep.mx. [En línea]. Disponible en: https://ccc.inaoep.mx/~emorales/Cursos/Automatas/principal.pdf. [Consultado: 22-abr-2025].