Universidad San Carlos De Guatemala

Centro Universitario De Occidente

División De Ciencias De La Ingeniería

Lenguajes Formales y de Programación

Auxiliar Julio Fernando

Manual Técnico

202132299, Gérman Alex Ramírez Limatuj

Quetzaltenango, Octubre de 2023

Manual técnico
Analizador Sintáctico
Parse Py

Parse-py (parte del Analizador Sintactico)

Este manual técnico proporciona una guía detallada sobre el diseño y la funcionalidad del Analizador Léxico y Sintactico Interactivo en Java. El objetivo de este proyecto es permitir a los usuarios analizar el código fuente en un lenguaje similar a Python, identificar y visualizar los tokens y errores léxicos, y generar gráficos para representar mejor la estructura del código, asi como todo lo referido al analisis sintactico de dichos tokens generados en el analizador lexico.

Entorno de Desarrollo

Para el desarrollo de esta aplicación se utilizó lo siguiente:

- JDK 17

- IDE Apache Netbeans 19

- Sistema Operativo Debian GNU/Linux 12 (bookworm) x86_64

- diagrams.net para la realización de diagrama de clases

Estructura del Proyecto

Paquete BackEnd.Herramientas

En este paquete se encuentran las clases que brindan funcionalidades auxiliares para el análisis léxico.

Clase Token

La clase Token almacena información sobre los tokens identificados en el código fuente. Cada token tiene un tipo, un lexema, un valor y coordenadas de fila y columna.

```
public class Token {
    // Atributos y constructor
    // Métodos de acceso
}
```

Clase Lexer

La clase Lexer se encarga del análisis léxico del código fuente y genera una lista de tokens. Utiliza expresiones regulares para identificar patrones en el texto.

```
public class Lexer {
    public void analyze(String sourceCode) {
        // Implementación del análisis léxico
    }

    public List<Token> getTokens() {
```

```
        // Obtener lista de tokens
    }
}
```

Clase OpenFile

La clase OpenFile permite cargar un archivo de texto en un componente de edición.

```
public class OpenFile {
    public static void openFileAndSetText(JTextPane textPane) {
        // Implementación para abrir y cargar archivo en el textPane
    }

    public static void loadTxtFile(File file, JTextPane textPane) {
        // Cargar contenido del archivo en el textPane
    }
}
```

Paquete BackEnd.Herramientas.TokenType

Clase Token(tipo)

```
public enum TypePalabraReservada {
//atributos de cada enum}
```

Lo mismo con cada uno de los tipos de tokens requeridos en el lenguaje

Paquete FrontEnd

En este paquete se encuentran las clases relacionadas con la interfaz gráfica y la ejecución del programa.

Clase EntornoVisual

La clase EntornoVisual representa la interfaz gráfica principal de la aplicación.

```
public class EntornoVisual extends JFrame {
    // Atributos y componentes
    public EntornoVisual() {
        // Configuración de la interfaz gráfica
    }

    private void analyzeCode() {
        // Realizar análisis léxico y mostrar resultados
```

}

// Otros métodos y manejo de eventos

}

Clase AnalisisLexico

Descripción:

La clase AnalisisLexico representa una parte fundamental del análisis léxico en una aplicación. Esta clase se utiliza para analizar un texto de entrada en busca de tokens, identificar su tipo y generar tablas de información. La clase forma parte del paquete FrontEnd.

Atributos:

mapTokens: Un mapa que almacena los tipos de tokens encontrados y sus valores asociados.

infoTabla: Una lista de listas que almacena información detallada de los tokens analizados, incluyendo el tipo, patrón, valor, fila y columna de ocurrencia.

tablaToken: Una lista de listas que almacena información sobre los tokens analizados, incluyendo el tipo, patrón, valor, fila y columna de ocurrencia.

tablaErroresLexicos: Una lista de listas que almacena información sobre los errores léxicos encontrados, incluyendo el tipo, patrón, valor, fila y columna de ocurrencia.

text: El texto de entrada que se va a analizar léxicamente.

Constructor AnalisisLexico(String inputText)

Descripción: El constructor crea una instancia de AnalisisLexico y toma el texto de entrada como parámetro.

Configuración inicial: Se realiza una configuración inicial, inicializando las estructuras de datos utilizadas para el análisis léxico.

Método analyze()

Descripción: Este método realiza el análisis léxico del texto de entrada.

Lógica de análisis léxico: El método utiliza un analizador léxico (clase Lexer) para escanear el texto y generar tokens. Se procesa cada token, identificando su tipo y registrando información detallada en las tablas de información y errores léxicos.

Actualización de las estructuras de datos: Se actualizan las estructuras de datos mapTokens, infoTabla, tablaToken y tablaErroresLexicos con la información de los tokens y errores encontrados.

Método private void clearData()

Descripción: Este método se utiliza para limpiar y restablecer las estructuras de datos utilizadas para el análisis léxico, preparándolas para un nuevo análisis.

Método private void processToken(Token token, Lexer lexico, int fila)

Descripción: Este método procesa un token identificado por el analizador léxico.

Determinación del tipo de token: El método determina el tipo del token y su patrón, así como otros detalles como el valor, la fila y la columna de ocurrencia.

Registro de información: La información del token se registra en las estructuras de datos correspondientes, incluyendo las tablas de tokens y errores léxicos.

Método private boolean isTypeSpace(Object type)

Descripción: Este método verifica si un objeto dado es de tipo TypeSpace, que indica un espacio o separador en el análisis léxico.

Métodos de Obtención de Datos:

getMapTokens(): Permite obtener el mapa de tipos de tokens y sus valores asociados.

getInfoTabla(): Permite obtener la lista de listas con información detallada de los tokens analizados.

getTablaToken(): Permite obtener la lista de listas con información de los tokens analizados.

getTablaErroresLexicos(): Permite obtener la lista de listas con información de errores léxicos.

Clase ColoreoTokens

Descripción:

La clase ColoreoTokens se encarga de aplicar colores a diferentes tipos de tokens en un documento de texto, lo que facilita la visualización y comprensión del código. Los colores se basan en el tipo de token y se aplican mediante estilos en un StyledDocument.

Métodos:

colorPalabras(StyledDocument doc, String text, List<List<Object>> tablaToken)

Descripción: Este método aplica estilos y colores a los tokens en el texto.

Parámetros:

doc: El StyledDocument en el que se aplicarán los estilos.

text: El texto completo en el que se buscarán los tokens.

tablaToken: Una lista de listas que contiene información sobre los tokens (tipo, patrón y valor).

getColorForTokenType(Object tokenType)

Descripción: Este método determina el color que se aplicará a un token según su tipo.

Parámetro:

tokenType: El tipo de token a evaluar.

Devuelve: Un objeto Color que representa el color a aplicar al token.

Comentarios Adicionales:

La clase utiliza StyledDocument para aplicar estilos al texto.

Los colores se seleccionan en función del tipo de token, como identificadores, palabras reservadas, operadores aritméticos, operadores lógicos, etc.

Si el tipo de token es desconocido, se usa un color por defecto (gris oscuro).

Los colores son seleccionados de acuerdo con su significado semántico para mejorar la legibilidad del código.

Clase Reportes

Descripción:

La clase Reportes representa una interfaz gráfica que proporciona una serie de botones para acceder a informes o reportes relacionados con el análisis de código fuente. Estos informes se muestran en ventanas emergentes separadas. La clase se utiliza en la interfaz de usuario de la aplicación.

Métodos:

initComponents()

Descripción: Este método inicializa la interfaz gráfica de la clase Reportes. Crea botones para diferentes informes y establece los manejadores de eventos para cada botón.

Métodos jButtonXActionPerformed()

Descripción: Estos métodos manejan las acciones cuando se hace clic en los botones. Cada uno muestra un informe específico en una ventana emergente.

Comentarios Adicionales:

La clase utiliza una disposición de cuadrícula (GridLayout) para organizar los botones verticalmente.

Los informes se muestran en ventanas emergentes separadas utilizando la clase JFrame.

Cada informe es generado por una clase relacionada (como TablaDeSG, TablaDeSBC, ListaDeInstruccionesBC, etc.), que se obtiene a través de métodos de la clase EntornoVisual.

Clase Syntactic

Descripción:

La clase Syntactic se encarga de realizar un análisis sintáctico de un conjunto de tokens y generar información relevante acerca de la estructura del código fuente. Esta clase se utiliza para identificar y procesar elementos específicos del código y generar impresiones detalladas para cada uno de ellos.

Métodos:

analizar()

Descripción: Este método inicia el análisis sintáctico de los tokens almacenados en la propiedad tablaToken. Procesa cada token secuencialmente y determina el tipo de estructura presente en el código fuente.

procesarToken(List<Object> tokenInfo, String lexema, int fila, int columna)

Descripción: Este método procesa un token en función de su tipo y contenido. Identifica elementos como declaraciones de variables, ciclos for, ciclos while, definiciones de funciones, comentarios y operadores ternarios.

mostrarBloqueDeCodigo(int inicioBloque, int finBloque)

Descripción: Este método muestra un bloque de código en función de su inicio y final. Muestra el código identificado, así como los comentarios presentes en el bloque.

getBloqueDeCodigoIdentificado()

Descripción: Devuelve el bloque de código identificado en formato de cadena.

detectarIndentacion(String lexema)

Descripción: Este método determina el nivel de indentación de una línea de código mediante la detección de espacios o tabulaciones al inicio de la línea.

procesarOperadorTernario(int fila, int columna)

Descripción: Procesa un operador ternario, identificando sus componentes y generando impresiones relacionadas con él.

procesarDeclaracionVariable(int fila, int columna)

Descripción: Procesa una declaración de variable, identificando el identificador, el operador de asignación y el valor de la variable. Puede manejar variables simples y arreglos.

obtenerFinArreglo(String valor)

Descripción: Este método determina el final de una definición de arreglo en caso de que exista.

procesarCicloFor(int fila, int columna)

Descripción: Procesa un ciclo for, identificando sus componentes y generando impresiones relacionadas con él.

procesarCicloWhile(int fila, int columna)

Descripción: Procesa un ciclo while, identificando sus componentes y generando impresiones relacionadas con él.

procesarDefinicionFuncion(int fila, int columna)

Descripción: Procesa una definición de función, identificando el nombre de la función, los parámetros y generando impresiones relacionadas con ella.

getImpresiones()

Descripción: Devuelve una lista de impresiones generadas durante el análisis sintáctico, que contienen detalles sobre las estructuras identificadas en el código fuente.

Comentarios Adicionales:

La clase utiliza una serie de contadores para hacer un seguimiento del número de elementos procesados, como declaraciones de variables, ciclos, definiciones de funciones y comentarios.

El método mostrarBloqueDeCodigo se utiliza para mostrar comentarios en línea presentes en el código.

Las impresiones generadas se almacenan en una lista de cadenas llamada impresiones, que contiene información detallada sobre las estructuras encontradas en el código fuente.

Clase TokensIdentificados

Descripción

Esta clase representa una interfaz gráfica que muestra una tabla de tokens identificados y proporciona la funcionalidad para filtrar los tokens por tipo.

Métodos

actualizarTabla(List<List<Object>> tokenList)

Descripción: Este método se utiliza para actualizar la tabla de tokens con una nueva lista de tokens. Borra las filas existentes y agrega los nuevos datos a la tabla.

filtrarTablaPorTipo(String tipo)

Descripción: Filtra la tabla de tokens para mostrar solo los tokens de un tipo específico. Oculta los tokens de otros tipos.

mostrarContenidoOriginal()

Descripción: Restaura el contenido original de la tabla mostrando todos los tokens sin ningún filtro.

Componentes de la Interfaz

jTable1: Un componente de tabla donde se muestran los tokens identificados.

Botones (por ejemplo, filtroIdentificador, filtroPalabraReservada): Cada botón permite filtrar la tabla para mostrar solo tokens de un tipo específico.

mostrarTodo: Un botón que restablece la tabla para mostrar todos los tokens sin ningún filtro.

Constructor

TokensIdentificados(): El constructor de la clase inicializa la interfaz y agrega los componentes necesarios, como la tabla y los botones de filtro.

Clase TablaDeSG (y las demas tablas dentro de la clase de reportes)

Descripción

Esta clase representa una interfaz gráfica que muestra una tabla de descripciones estructurales generadas a partir del análisis de código fuente. También permite filtrar las descripciones por tipo y mostrar el contenido original sin filtro.

Métodos

actualizarTabla(List<String> impresiones)

Descripción: Este método se utiliza para actualizar la tabla de descripciones con una lista de impresiones. Borra todas las filas existentes y agrega las nuevas descripciones a la tabla.

filtrarTablaPorTipo(String tipo)

Descripción: Filtra la tabla para mostrar solo las descripciones de un tipo específico. Oculta las descripciones de otros tipos.

mostrarContenidoOriginal()

Descripción: Restablece el contenido original de la tabla, mostrando todas las descripciones sin ningún filtro.

Componentes de la Interfaz

jTable1: Un componente de tabla donde se muestran las descripciones estructurales generadas a partir del análisis del código fuente.

Botones (por ejemplo, filtroOperadoresTernarios, filtroDeclaracionVariables): Cada botón permite filtrar la tabla para mostrar solo descripciones de un tipo específico.

mostrarTodo: Un botón que restablece la tabla para mostrar todas las descripciones sin ningún filtro.

Constructor

TablaDeSG(): El constructor de la clase inicializa la interfaz y agrega los componentes necesarios, como la tabla y los botones de filtro.

Uso de Swing para la Interfaz Gráfica

La biblioteca Swing se utiliza para crear la interfaz gráfica de la aplicación. Aquí se muestra cómo se utilizan algunos componentes de Swing en la clase EntornoVisual.

```
public class EntornoVisual extends JFrame {
    // ...

    private void initComponents() {
        JTextPane codeTextPane = new JTextPane();
        JScrollPane codeScrollPane = new JScrollPane(codeTextPane);

        JButton analyzeButton = new JButton("Analizar");
        analyzeButton.addActionListener(e -> analyzeCode());

        JButton generateGraphButton = new JButton("Generar Gráfico");
        generateGraphButton.addActionListener(e -> generateGraph());
```

```
        JTable tokenTable = new JTable();
        DefaultTableModel tableModel = new DefaultTableModel();
        tokenTable.setModel(tableModel);

        // Configuración del diseño de la interfaz gráfica
        // Agregar componentes a paneles y frames
    }

    // ...
}
```

Clase Main

La clase Main es el punto de entrada de la aplicación. Desde aquí se crea y muestra la interfaz gráfica principal EntornoVisual.

```
package BackEnd;

import FrontEnd.EntornoVisual;

/**
 * Punto de entrada de la aplicación.
 */
public class Main {
    public static void main(String[] args) {
        // Crear una instancia de EntornoVisual
        EntornoVisual entorno = new EntornoVisual();

        // Configurar la ubicación y visibilidad del entorno
        entorno.setLocationRelativeTo(null); // Centrar la ventana en la pantalla
        entorno.setVisible(true); // Mostrar la ventana
    }
}
```

La clase Main crea una instancia de EntornoVisual, que es la interfaz gráfica principal de la aplicación. Configura su ubicación para que se centre en la pantalla y la hace visible, lo que permite a los usuarios interactuar con la aplicación. La ejecución del método main inicia la aplicación y la hace funcional para el usuario.

Clase NumeracionFilas

La clase NumeracionFilas proporciona una numeración automática en las filas de un componente de texto, como JTextPane o JTextArea. Además, se encarga de actualizar la numeración en función del desplazamiento del cursor.

Esta clase es útil para mejorar la experiencia del usuario al mostrar números de línea, lo que facilita la referencia a una línea específica en un texto largo. El código se encarga de calcular y mostrar la numeración de las filas, centrándola verticalmente en cada línea del componente de texto.

Clase NumeracionColumnas

La clase NumeracionColumnas es responsable de generar y mostrar la numeración de columnas en un JTextPane, además de indicar la columna actual en la que se encuentra el cursor.

Esta clase mejora la interacción del usuario al proporcionar información sobre la posición horizontal exacta del cursor dentro del texto. Al detectar los cambios en el cursor, actualiza la numeración de columnas en función de la posición actual del cursor en la línea.

Gramatica:

-Letras, Digitos, Numeros y Cadenas

letra :: = [a-z]

| [A-Z]

digito :: = [0-9]

<numero> :: = digito+

<cadena> :: = “”caracter*””

-Declaracion de variables

<declaracion de variables> :: = <identificador> <operador_asignacion><expresion>

<identificador> :: = letra[letra|digito]*

<operador_asignacion> :: = “=”

<expresion> :: = <cadena>

| <numero>

| <lista>

| <diccionario>

<lista> :: = “[”<elemento_lista>”]”

elemento_lista :: = [letra|digito]*

-Condicionales

<condicional> ::= <if>< expresión> <[condicional expresión]*>< :>

<if> :: = “if”

<expresion> ::= <cadena>

| <numero>

| <lista>

| <diccionario>

<lista> :: = “[”<elemento_lista>”]”

elemento_lista :: = [letra|digito]*

<:> ::= “:”

-Ternario

<ternario> ::= <condition_if_true> <if_condition> <else> <condition_if_false>

-Ciclo For

<ciclo> ::=

-Ciclo While
<ciclo> ::=

-Funciones Metodos

<Funciones/Metodos> ::=

-Bloques de Codigo

<expresiones>

<instrucciones>

<condicionales>

<if>

<operador ternario>

<ciclos>

<for>

<for-else>

<while>