Manual Técnico

Christian Alessander Blanco González
202000173
Facultad de Ingeniería de la Universidad de San Carlos de Guatemala

Contenido

INTRODUCCIÓN

El presente documento describe los aspectos técnicos informáticos del sistema de información. El documento familiariza al personal técnico especializado encargado de las actividades de mantenimiento, revisión, solución de problemas, instalación y configuración del sistema.

OBJETIVOS

- Entender el funcionamiento de un compilador en sus dos primeras fases.
- Entender los conceptos de análisis.
- Realizar el uso correcto de herramientas de análisis.
- Comprender los tipos de análisis sintáctico.

REQUERIMIENTOS

- Tener Java (versión 8 o superiores) instalado.
- Se recomienda tener un editor de código (NetBeans 8.2 u otro) para ver y editar el código de este programa.

EXPRESIONES REGULARES

Las siguientes expresiones regulares son las encargadas de identificar el tipo de texto especificado por el programador.

```
28  ESPACIOS = [ \t\r\n]+
29  COMENTARIO_L = ("//".*\r\n)|("//".*\n)|("//".*\r)
30  COMENTARIO_M = "/*""/"*([^*/]|[^*]"/"|"*"[^/])*"*"*"*/"
31  CARACTER = ([\'][^\t\'\"\n]|(\\\")|(\\n)|(\\\')|(\\t))?[\']|\'\$\{\d+\}\'
32  BOOLEAN = ("verdadero" | "falso")
33  IDENTIFICADOR = [_][a-zA-Z0-9]+[_]
34  CARACTER_E = [\"][\\][\"\'n][\"];
35  CADENA = [\"][^\"]*[\"]
36  DIGITOS = [0-9]+("." [0-9]+)?
```

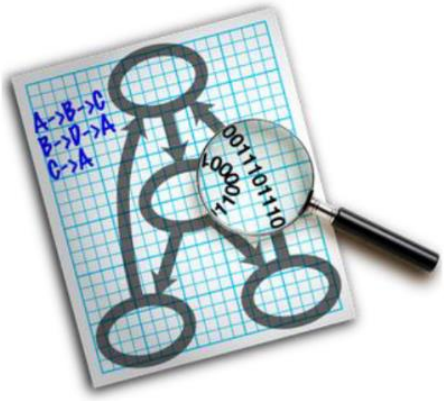
DESCRIPCION DEL SISTEMA

1. Descripcion del contenido del sitema

Este es un sistema en el cual el usuario interactúa directamente con una interfaz gráfica, este programa permite la lectura de un código escrito por el usuario, lo analiza y traduce el código a Python y Golang y guardarlos cada uno con su extensión respectiva.

2. Lenguajes Utilizados

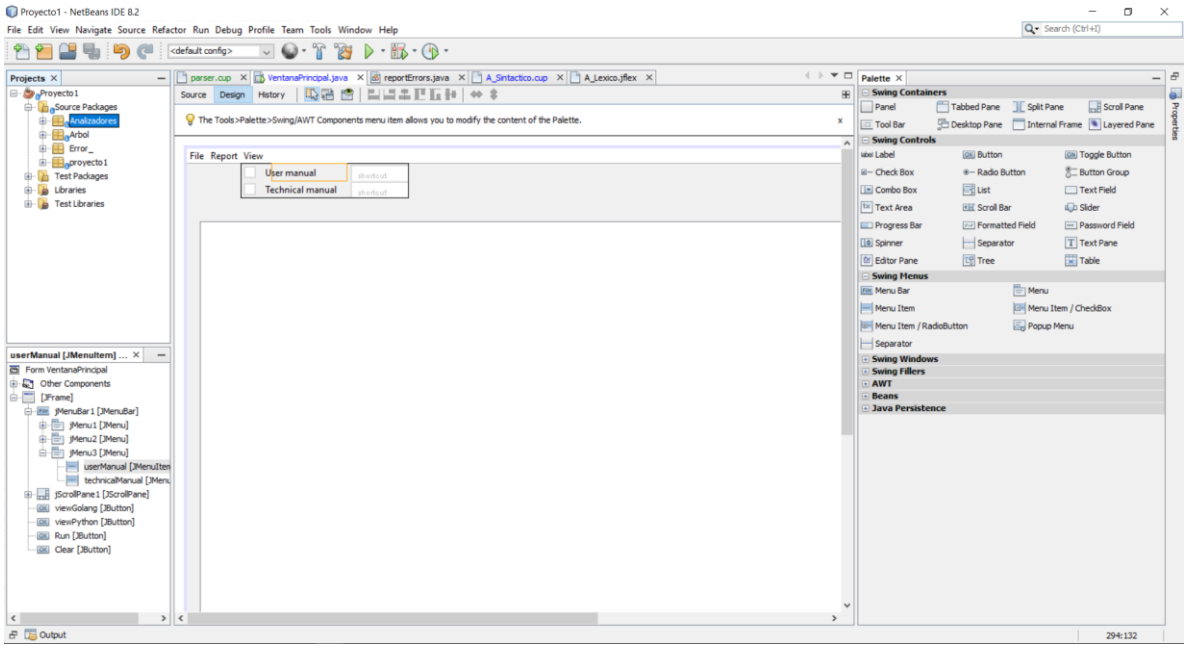
Para la creación de este sistema todas las funcionalidades se realizaron en lenguaje de java y también se utilizó Graphviz para la creación del AST. Utilizando un enfoque al paradigma de programación POO (programación Orientada a Objetos). Creando las diferentes estructuras de datos para almacenar la información.



Todo fue realizado con el IDE NetBeans 8.2:

3. Interfaz Gráfica

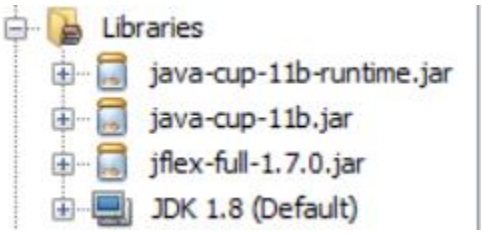
Para la realización de la interfaz grafica se utilizo el entorno de NetBeans, creando un nuevo jFrame con extensión Form y con drag and drop de los componentes que se utilizaron:



4. Analizadores

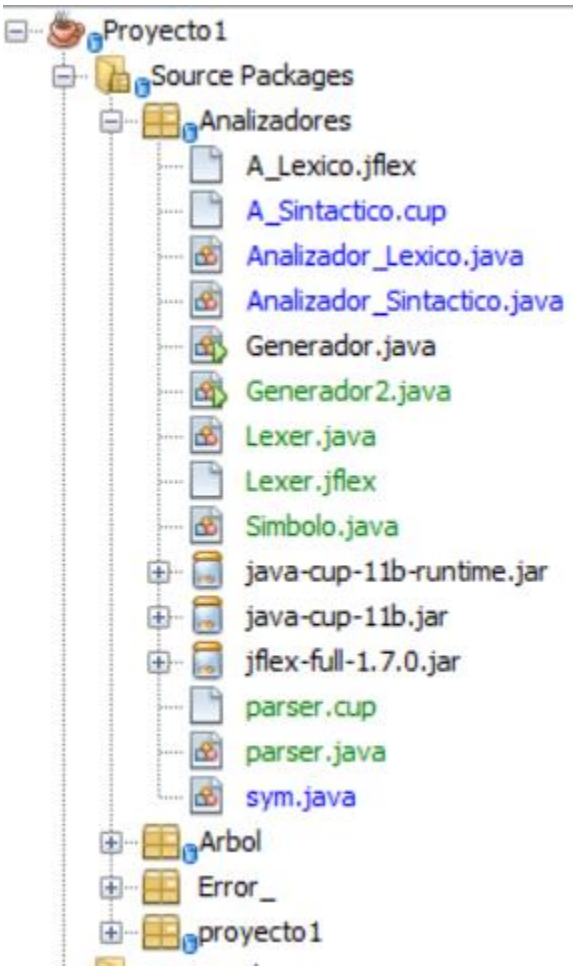
4.1 Analizador léxico: para el análisis léxico se utilizo la librería jflex-1.8.2 en el cual se describen los caracteres y cadenas de texto que el lenguaje puede reconocer.

4.2 Analizador sintactico: para el análisis sintactico se utilizo la librería java-cup-11b en el cual se describe la gramática correspondiente al lenguaje que se desea leer, utilizando los tokens leídos por el analizador léxico.

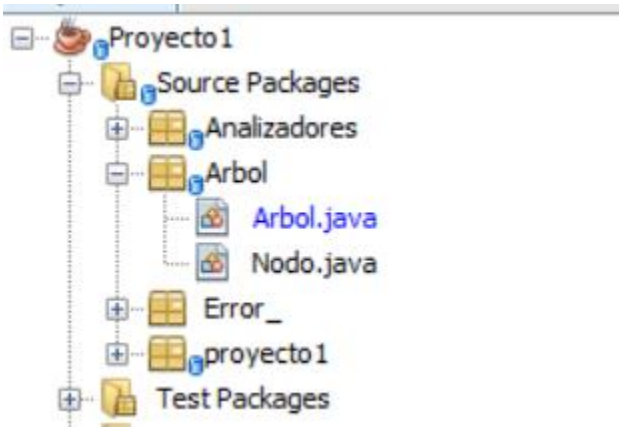


5. Desarrollo de package

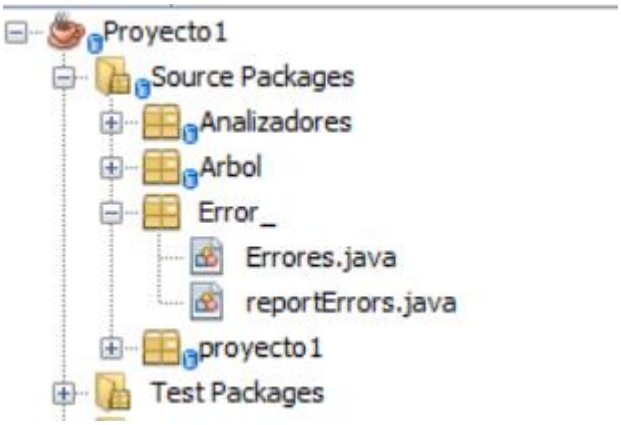
En el package Analizadores fue realizado todo lo del análisis del programa, donde se creó un archivo llamado A_Lexico con extensión jflex que es el encargado de poder reconocer cada uno de los tokens que sean necesarios del peseudo-parser. Luego fue creado un archivo llamado A_Sintactico con extensión cup para poder realizar la gramática libre de contexto y así poder establecer la sintexis que se necesita para leer el pseudocodigo. Luego fue generado una java class llamada Generador que es la encargada de poder crearnos nuestros archivos Analizador_Lexico y Analizador_Sintactico creados desde nuestro archivo jflex y cup.



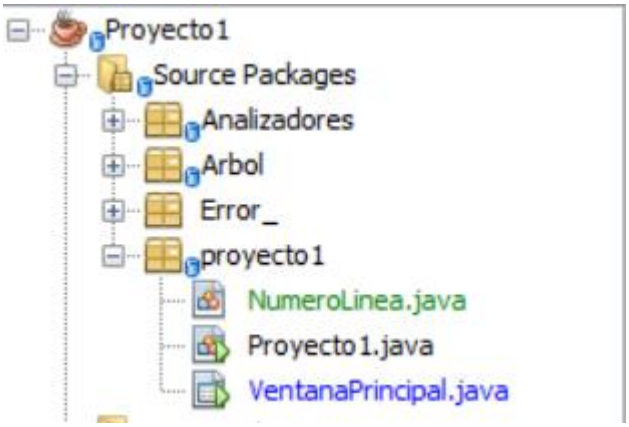
El package Árbol es el encargado de generar nuestro árbol sintactico. Esta conformada por una java class llamada Árbol y otra java class llamada Nodo. Nuestra java class Nodo es la encargada de poder obtener los valores desde nuestro archivo cup y poder almacenarlos en nuestra clase llamada Árbol. Nuestra clase llamada es la encargada de poder realizar la recursividad para poder posicionar cada nodo en su posición correcta y luego poder graficarlo y guardarlo.



El package de error esta conformado por dos java class, una llamada como Errores y la otra reportErrors. La clase Errores es la encargada de poder obtener cada uno de lo errores léxicos y sintácticos que contiene nuestro pseudocodigo y nuestra clase reportErrors es la encargada de generar una tabla con todos los errores en html.



El package proyecto 1 es nuestro package principal desde donde se ejecuta todo nuestro programa, esta conformada por una java class llamada Proyecto 1 que es la encargada de llamar nuestro jFrame que es llamado VentanaPrincipal y así podamos obtener una visualización de la interfaz grafica y poder hacer uso de cada una de las funciones dadas.



Para poder realizar el analizador se utilizo el siguiente código:

```
    private void analizadorLexico() throws Exception {
        String text2 = (String) jTextAreal.getText();
        Analizador_Lexico lexico = new Analizador_Lexico(new StringReader(text2));
        Analizador_Sintactico sintactico = new Analizador_Sintactico(lexico);
        sintactico.parse();
        //System.out.println(text2);
    }
```

Para guardar cualquier tipo de archivo, se utilizó el siguiente código:

```
private void saveHow() {
    JFileChooser save = new JFileChooser();
    save.showSaveDialog(null);
    save.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);

    File file = save.getSelectedFile();
    saveFile(jTextAreal.getText(), file);
}

private void saveFile(String cadena, File file) {
    FileWriter write;
    try {
        write = new FileWriter(file, true);
        write.write(cadena);
        write.close();
    } catch (FileNotFoundException ex) {
        JOptionPane.showMessageDialog(null, "Error al guardar, ingrese el nombre que le desea dar al archivo");
    } catch (IOException ex) {
        JOptionPane.showMessageDialog(null, "Error al guardar, en la salida");
    }
}
```

Para poder almacenar los errores léxicos y sintácticos se utilizó el siguiente código:

```
private void reportarErrors() {

    //Analizador_Lexico lexico;
    //Analizador_Sintactico sintactico;
    if (jTextAreal.getText() != null) {

        try {
            Boolean Errores = false;

            String text2 = (String) jTextAreal.getText();
            Analizador_Lexico lexico = new Analizador_Lexico(new StringReader(text2));
            Analizador_Sintactico sintactico = new Analizador_Sintactico(lexico);
            sintactico.parse();

            if (lexico.errores.size() > 0 || sintactico.errores.size() > 0) {
                if (lexico.errores.size() > 0) {
                    System.out.println("-------------ERRORES LEXICOS-----------");
                    for (Errores errore : lexico.errores) {
                        System.out.println(errore.toString());
                    }
                }
                if (sintactico.errores.size() > 0) {
                    System.out.println("-----------------ERRORES SINTACTICOS-----------------");
                    for (Errores errore : sintactico.errores) {
                        System.out.println(errore.toString());
                    }
                }
                Errores = true;
            } else {
                System.out.println("No se detectaron errores..");
                Errores = false;
            }
            System.out.println("Generando reporte de errores....");
            reporte.GenerarReporte(lexico.errores, sintactico.errores);
            // incluuir aca reporte de errores
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(null, "Ocurrio un error.");
        }
    } else {
        JOptionPane.showMessageDialog(null, "No se detecto ningun texto, cargue un archivo o escriba en el area de texto.");
    }
}
```

Para la creación del AST se utilizó el siguiente código:

```
private void ASTActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    Nodo raiz = null;

    try {
        Analizador_Sintactico sintactico = new Analizador_Sintactico(new Analizador_Lexico(new BufferedReader(new StringReader(jTextAreal.getText()))));
        sintactico.parse();
        raiz = sintactico.getRaiz();
        System.out.println("Generando arbol sintactico..!");


    } catch (Exception ex) {
        Logger.getLogger(Proyectol.class.getName()).log(Level.SEVERE, null, ex);
    }
    if (raiz != null) {
        Arbol arbolito = new Arbol(raiz);
        arbolito.GraficarSintactico();

    }
    JOptionPane.showMessageDialog(null, "Analizado con exito", "Informacion", JOptionPane.INFORMATION_MESSAGE);
    abrirArchivos("Arbol_Sintactico.svg");
}
```