Laboratorio de Organización de Lenguajes y Compiladores 1, Sección C.

PROYECTO 1 PSEUDO PARSER

MANUAL TECNICO

FECHA: 19/09/2022

Juan Josue Zuleta Beb
Carné: 202006353

Introducción

El presente documento describe los aspectos técnicos informáticos de la aplicación, deseñada a través de la interfaz gráfica de Java. El documento familiariza al personal técnico especializado encargado de las actividades de mantenimiento, revisión, solución de problemas, instalación y configuración del sistema.

Objetivos

Instruir el uso adecuado del de la instalación y comprensión del código y de la implementación de métodos, para el acceso oportuno y adecuado en la inicialización de este, mostrando los pasos a seguir en el proceso de inicialización, así como la descripción de los archivos relevantes del sistema los cuales nos orienten en la configuración.

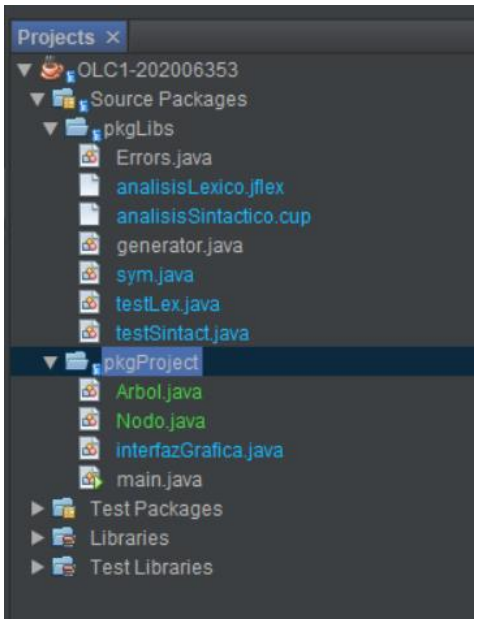
Requisitos Mínimos del Sistema

Sistema operativo 64 bits

- Microsoft Windows 10/8/7/Vista/2003/XP (incl.64-bit)
- macOS 10.5 o superior
- Linux GNOME o KDE desktop
- Procesador a 1.6 GHz o superior
- 1 GB (32 bits) o 2 GB (64 bits) de RAM (agregue 512 MB al host si se ejecuta en una máquina virtual)
- 3 GB de espacio disponible en el disco duro
- Disco duro de 5400 RPM
- Tarjeta de vídeo compatible con DirectX 9 con resolución de pantalla de 1024 x 768 o más.
- Navegador web (Recomendado: Google Chrome)

Estructura raíz:

El proyecto tiene la siguiente estructura de directorios:



Directorio de la App: Raiz

La clase main:

Contiene el método principal de nuestra aplicación además de todas las clases y controlara todos los métodos y la implementación de la interfaz gráfica.

DESCRIPSION DE LOS METODOS Y CLASES:

Los métodos y clases utilizados para este programa fueron pensados y diseñados específicamente para el desarrollo e implementación de este, siguiendo rigurosamente los requerimientos de diseño inicialmente propuestos en el manual de requerimientos.

La clase Generator:

La clase generator se encargar de compilar los archivos .jflex y .cup para la generación de las clases TestLex y la clase TestSintact.

```java
package pkgLibs;

public class generator {


    public static void generarCompilador() {
        try {
            String ruta = "src/pkgLibs/"; //ruta donde tenemos los archivos con extension .jflex y .cup
            String opcFlex[] = {ruta + "analisisLexico.jflex", "-d", ruta};
            jflex.Main.generate(opcFlex);
            String opcCUP[] = {"-destdir", ruta, "-parser", "testSintact", ruta + "analisisSintactico.cup"};
            java_cup.Main.main(opcCUP);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

La clase SYM:

Se encargar de manejar las expresiones del análisis sintáctico como tokens para poder utilizarlos en el análisis sintáctico.

```java
package pkgLibs;

/** CUP generated class containing symbol constants. */
public class sym {
  /* terminals */
  public static final int INGRESAR = 28;
  public static final int COMENTARIOS = 65;
  public static final int HASTAQUE = 46;
  public static final int EXP = 71;
  public static final int MAYOR = 17;
  public static final int ESIGUAL = 21;
  public static final int SI = 31;
  public static final int CONVALOR = 30;
  public static final int INICIO = 26;
  public static final int POT = 16;
  public static final int POR = 13;
  public static final int METODO = 48;
  public static final int ORCF = 76;
  public static final int DELOCONTRARIO = 33;
  public static final int INTABRE = 62;
```

La clase analisisLexico.jflex:

Es donde se definió las expresiones regulares y todo tipo de tokes que será capaz de reconocer nuestra gramática, es aquí donde definimos nuestro lenguaje.

```
S = [+/*<>=!"-"]
W = [a-zA-z]
D = [0-9]
C = ["¿""?""¡""!"]


id = "_"{W}|({W}|{D})*"_"

tipo = "numero"|"cadena"|"boolean"|"caracter"
bool = "verdadero"|"falso"

entero = {D}|{D}+
flotante = {D}+"."{D}+

cadena = (\"[^\"]*\")|(\"[^\"]*\"(\"[^\"]*\")*)

character = "'"("$"{"{D}+"}")"'"
caracter = "'"{W}+"'"

comentario = "//"[^\n]*
comentarios = "/*"[^\*]*"*/"

simbolo = [ \t\r\n\f]

exp_case = {C}({entero}|{flotante}|{cadena}|{caracter})+{C}

exp_cc = ("("){entero}|{flotante}|{S})+(")")
exp_sc = ({entero}|{flotante}|{S})+

exp_pot = "["({entero}|{flotante}|{S}|{id})+"]"
exp = ({exp_cc}|{exp_sc})+
```

La clase analisisSintactico.cup:

Aquí se definió la gramática de tipo 2 la cual se utilizó para validar el orden de entrada por parte del seudocódigo y así poder interpretar y extraer la información que el usuario está intentado ingresar.

```
pseudo::= pseudo opciones
        | opciones
;

//GRAMATICA PRINCIPAL
opciones::= declaration:a
    {: pseudoG += String.valueOf(a); :}
    | asignation:a
    {: pseudoG += String.valueOf(a); :}
    | coments:a
    {: pseudoG += String.valueOf(a); :}
    | prints:a
    {: pseudoG += String.valueOf(a); :}
    | retorn:a
    {: pseudoG += String.valueOf(a); :}
    | sentencia:a
    {: pseudoG += String.valueOf(a); :}
    | ciclos:a
    {: pseudoG += String.valueOf(a); :}
;
```

Repositorio del proyecto: https://github.com/BeKingGO/OLC1-202006353