Universidad Mariano Gálvez de Guatemala

Ingeniería en sistemas de la información

Manual Técnico – Proyecto Final

Curso: Compiladores

Catedrático: Ing. Miguel Alejandro Catalan Lopez

José Benjamin Méndez Salguero

No. De Carné: 7590-20-812

09 de junio del 2023

CONTENIDO

OBJETIVO

El objetivo de este manual de usuario es proporcionar instrucciones claras y detalladas sobre cómo utilizar la herramienta web de traducción de código de Java a Python. El manual guiará a los usuarios a través de los pasos necesarios para cargar un programa en lenguaje Java, traducirlo automáticamente a Python y descargar el programa resultante en un archivo con extensión .py.

ALCANCE Y REQUERIMIENTOS

- Descripción de la herramienta web y su funcionalidad principal.
- Requisitos del sistema y navegadores compatibles.
- Instrucciones sobre cómo acceder a la herramienta web y cómo cargar un programa en lenguaje Java.
- Explicación detallada del proceso de traducción de Java a Python utilizando JFlex y CUP.
- Cómo visualizar el programa traducido en Python en la interfaz de la herramienta web.
- Cómo descargar el programa traducido en un archivo con extensión .py.
- Solución de problemas comunes y preguntas frecuentes.

HERRAMIENTAS UTILIZADAS

IntelliJ IDEA

es un entorno de desarrollo integrado (IDE) para el desarrollo de programas informáticos. Es desarrollado por JetBrains para Windows, Linux y MacOS. Está disponible en dos ediciones: edición para la comunidad y edición comercial.

https://www.jetbrains.com/es-es/idea/download/#section=windows

JFLEX

Es una herramienta de generación de analizadores léxicos (scanners) escrita en Java. Se utiliza para construir analizadores léxicos eficientes que reconocen y descomponen el texto en tokens individuales. En el contexto de la traducción de código de Java a Python, JFlex se puede utilizar para realizar el análisis léxico del código fuente escrito en lenguaje de programación Java.

CUP

Es una herramienta de generación de analizadores sintácticos escrita en Java. Se utiliza para construir analizadores sintácticos (parsers) a partir de especificaciones de gramáticas formales. En el contexto de la traducción de código de Java a Python, CUP se puede utilizar para realizar el análisis sintáctico y semántico del código fuente escrito en lenguaje de programación Java.

ANALIZADOR LÉXICO

Se creo el archivo .jflex, donde se genera el paquete umg.compiladores y el import java_cup.runtime.* que nos sirve para que al generar el archivo Lexer tenga las librerías apropiadas para aceptar la herramienta cup.

Tambien se genero public class Lexer que es el nombre principal de nuestra clase de Lexer, se indico que se usara char, líneas y columnas para la lectura léxica y la herramienta cup.

Seguido se crea un método que haga un return de nuestros símbolos o lexemas que pasan a ser tokens en cup.

Se generaron cuatro expresiones regulares las cuales son lectura de letras, lectura de números, lectura de palabras o combinaciones de letras y números y una que identifica los espacios en blanco, tabulaciones y enters.

```
package umg.compiladores;

import java_cup.runtime.*;

%%

%public
%class Lexer
%char
%line
%column
%cup

%{
    private Symbol symbol(int tipo, Object valor) {
        return new Symbol(tipo, yyline, yycolumn, valor);
    }
%}

palabra = [a-zA-Z]+
digito = [0-9]+
identificador = [a-zA-Z][a-zA-Z0-9]*
espacios_blanco = [\r|\n|\r\n| |\t]
```

Seguido de eso se encuentran las reglas léxicas

Se mostrarán unos ejemplos

“public” nos indica que cuando se encuentre la palabra reservada de java public se indicara su valor, columna y fila, y se retonar el token PUBLICO a cup, junto con su valor.

```
%%

//Reglas léxicas
"public"     { System.out.println("Lexema: "
              + yytext()
              + " columna: "
              + yychar
              + " fila: "
              + yyline ); return symbol(sym.PUBLICO, yytext());}
```

Se llaman las expresiones regulares que antes se nombraron se llaman con llaves indicando que es una variable ya declarada.

```
{identificador} { System.out.println("Lexema: "
              + yytext()
              + " columna: "
              + yychar
              + " fila: "
              + yyline ); return symbol(sym.IDENTIFICADOR, yytext());}

{digito}    { System.out.println("Lexema: "
              + yytext()
              + " columna: "
              + yychar
              + " fila: "
              + yyline ); return symbol(sym.DIGITO, new Integer(yytext()));}
```

ANALIZAR SINTACTICO

En cup se declara el paquete donde estarán ubicados los archivos que se generen, tambien se hacen los imports de java_cup y de un arraylist

Se hizo un método parser code que indica la creación de una ArrayList llamada resultados.

```
package umg.compiladores;

import java_cup.runtime.*;

import java.util.ArrayList;


parser code {:
    public ArrayList<String> resultados = new ArrayList<>();
:};
```

Se declararon los terminales y no terminales

Los terminales son los tokens que devuelve el analizador léxico

Y los no terminales son los nombres de las gramáticas y producciones de nuestra gramática.

```
terminal SYSTEM, SALIDA, PUNTO, IMPRIMIR, IMPRIMIR_LINEA, PARENTESIS_ABRE, PARENTESIS_CIERRA, PUNTO_COMA, NOT, COMILLA_SIMPLE_CIERRA;
terminal MULTI, SUMA, COMA, RESTA, DIVIDIR, DOS_PUNTOS, MENOR, ASIGNAR, MAYOR, TEXTO, CORCHETE_ABRE, CORCHETE_CIERRA;
terminal POTENCIA, GUION_BAJO, LLAVE_ABRE, LLAVE_CIERRA, DIFERENTE, COMILLAS_DOBLES, AND, MENOR_IGUAL, IGUAL, MAYOR_IGUAL, HACER, SI;
terminal IN, OR, REPETIR, INTEGER, NUEVO, CASO, ADEMAS, LONG, SIGUIENTE, VERDAD, VACIO, ROMPER, CLASE, FALSO, FLOAT, SHORT, MIENTRAS;
terminal DOBLE, RETORNAR, CAMBIO, SCAN, BOOL, SIGUIENTE_INT, SIGUIENTE_LINEA, SIGUIENTE_LONG, SIGUIENTE_FLOAT, SIGUIENTE_SHORT;
terminal SIGUIENTE_DOBLE, SIGUIENTE_BOOL, PRIVADO, PUBLICO, ARGS, MAIN, STATIC, POR_DEFECTO;
terminal Integer DIGITO;
terminal String IDENTIFICADOR;

non terminal inicio, tipo_imprimir, tipo_metodo, imprimir, metodo, main, funcionalidad, operador, aritmetica, declarar, parametros, estado_bool,
```

Se inicializa la gramática indicando que inicio es el símbolo inicial, el cual contiene la lectura de la clase básica como seria public class Ejemplo {}

```
start with inicio;

inicio              ::= privacidad CLASE:clase_val IDENTIFICADOR:nom_clase_val LLAVE_ABRE:llave_abre_val main LLAVE_CIERRA:llave_cierra_val
                    |   privacidad CLASE:clase_val IDENTIFICADOR:nom_clase_val LLAVE_ABRE:llave_abre_val funcionalidad main LLAVE_CIERRA:llave_cierra_val
```

Se crearon producciones llamadas main, declarar y parámetros los cuales hacen

Main: lee los atributos de la clase main de java

Declarar: muestra los tipos de datos a declarar

Parámetros: permite el ingreso de los parámetros en métodos de java

```
main                ::= privacidad STATIC:static_val VACIO:vacio_val MAIN:main_val PARENTESIS_ABRE:parentesis_abre_val declarar CORCH
                    |   privacidad STATIC:static_val VACIO:vacio_val MAIN:main_val PARENTESIS_ABRE:parentesis_abre_val declarar CORCH
                    |   privacidad STATIC:static_val VACIO:vacio_val MAIN:main_val PARENTESIS_ABRE:parentesis_abre_val declarar CORCH
                    |   privacidad STATIC:static_val VACIO:vacio_val MAIN:main_val PARENTESIS_ABRE:parentesis_abre_val declarar CORCH

declarar            ::= TEXTO:tipo_dato_val                                    {: RESULT = "str()"; resultados.add(RESULT); :}
                    |   INTEGER:tipo_dato_val                                  {: RESULT = "int()"; resultados.add(RESULT); :}
                    |   LONG:tipo_dato_val                                     {: RESULT = "int()"; resultados.add(RESULT); :}
                    |   FLOAT:tipo_dato_val                                    {: RESULT = "float()"; resultados.add(RESULT); :}
                    |   DOBLE:tipo_dato_val                                    {: RESULT = "float()"; resultados.add(RESULT); :}
                    |   SHORT:tipo_dato_val                                    {: RESULT = "int()"; resultados.add(RESULT); :}
                    |   BOOL:tipo_dato_val                                     {: RESULT = "bool()"; resultados.add(RESULT); :}
                    |   VACIO:tipo_dato_eval
                    |   declarar CORCHETE_ABRE CORCHETE_CIERRA:tipo_dato_val   {: RESULT = "str()"; resultados.add(RESULT); :};

parametros          ::= declarar IDENTIFICADOR:identificador_val
                    |   declarar IDENTIFICADOR:identificador_val COMA parametros
                    |   IDENTIFICADOR:identificador_val COMA parametros;
```

Método: permite el ingreso de un metodo

tipo_metodo: permite el ingreso de los atributos específicos de un método como su nombre, parámetros, etc.

```
metodo              ::= privacidad tipo_metodo;

tipo_metodo         ::= parametros PARENTESIS_ABRE:parentesis_abre_va
                    |   parametros PARENTESIS_ABRE:parentesis_abre_va
                    |   parametros PARENTESIS_ABRE:parentesis_abre_va
```

Funcionalidad: seria una de las producciones principales pues esta contiene las funcionalidades que puede tener un método como, declaración de variables, impresión de datos, generar sentencias de selección o cíclicas, etc.

```
funcionalidad       ::= imprimir funcionalidad
                    |   SCAN IDENTIFICADOR:identificador_val ASIGNAR NUEVO SCAN PARENTESIS_ABRE:parentesis_abre_val PARENTESIS_C
                    |   parametros funcionalidad
                    |   parametros ASIGNAR:asignar_val DIGITO:digito_val funcionalidad
                    |   parametros ASIGNAR:asignar_val RESTA:resta_val DIGITO:digito_val funcionalidad
                    |   parametros ASIGNAR:asignar_val IDENTIFICADOR:identificador_val funcionalidad
                    |   parametros ASIGNAR:asignar_val DIGITO:digito_val1 PUNTO DIGITO:digito_val2 funcionalidad
                    |   parametros ASIGNAR:asignar_val RESTA:resta_val DIGITO:digito_val1 PUNTO DIGITO:digito_val2 funcionalidad
                    |   parametros ASIGNAR:asignar_val aritmetica funcionalidad
                    |   IDENTIFICADOR:identificador_val ASIGNAR:asignar_val IDENTIFICADOR funcionalidad
                    |   IDENTIFICADOR:identificador_val ASIGNAR:asignar_val aritmetica funcionalidad
                    |   IDENTIFICADOR:identificador_val ASIGNAR:asignar_val estado_bool PUNTO_COMA:punto_coma
                    |   SI:if_val PARENTESIS_ABRE:parentesis_abre_val comparacion PARENTESIS_CIERRA:parentesis_cierra_val LLAVE_
                    |   MIENTRAS:while_val PARENTESIS_ABRE:parentesis_abre_val comparacion PARENTESIS_CIERRA:parentesis_cierra_va
                    |   HACER:do_valor LLAVE_ABRE:llave_abre_val funcionalidad LLAVE_CIERRA:llave_val MIENTRAS:while_val PARENTES
                    |   REPETIR:for_val PARENTESIS_ABRE:parentesis_abre_val INTEGER IDENTIFICADOR:identificador_val1 ASIGNAR DIG
                        LLAVE_ABRE:llave_abre_val funcionalidad LLAVE_CIERRA:llave_val
                    |   CAMBIO PARENTESIS_ABRE:parentesis_abre_val IDENTIFICADOR PARENTESIS_CIERRA:parentesis_cierra_val LLAVE_A
                    |   PUNTO_COMA:punto_coma funcionalidad
                    |   PUNTO_COMA:punto_coma;
```

Aritmética: se encarga de realizar suma, resta, multiplicación, división y potencia.

Imprimir: se encarga de imprimir en consola algo deseado

Tipo_imprimir: selecciona si se desea imprimir en la misma línea o hacer un salto de línea

Retorno_metodo: nos indica el retorno de un método de un tipo de variable.

If_anidado: nos permite generar un if anidado como if,else if, else, etc.

Switch_case: nos permite hacer un switch caso completo.

```
aritmetica              ::= DIGITO:digito_val1 operador DIGITO:digito_val2
                        |   DIGITO:digito_val1 SUMA:suma DIGITO:digito_val2
                        |   IDENTIFICADOR:identificador_val1 SUMA:suma IDENTIFICADOR:identificador_
                        |   IDENTIFICADOR:identificador_val1 operador IDENTIFICADOR:identificador_v

imprimir                ::= SYSTEM:sys_val PUNTO SALIDA PUNTO tipo_imprimir PARENTESIS_ABRE:parente

tipo_imprimir           ::= IMPRIMIR:imprimir_val
                        |   IMPRIMIR_LINEA:imprimir_linea_val

retorno_metodo          ::= RETORNAR:retornar_val IDENTIFICADOR:identificador_val PUNTO_COMA;

if_anidado              ::= ADEMAS LLAVE_ABRE:llave_abre_val funcionalidad LLAVE_CIERRA:llave_val
                        |   ADEMAS SI PARENTESIS_ABRE:parentesis_abre_val comparacion PARENTESIS_CI
                        |   /* ε */;

switch_case             ::= CASO DIGITO DOS_PUNTOS funcionalidad ROMPER PUNTO_COMA switch_case
                        |   POR_DEFECTO DOS_PUNTOS funcionalidad ROMPER PUNTO_COMA
                        |   /* ε */;
```

Comparación: nos permite comparar tipos de datos para las sentencias selectivas o cíclicas como 1 < 2 o nombre == nombre1.

Operador_comparacion: tiene el listado de los operadores comparativos de llama como <, >, ==.

Operador_logico: nos indica la lista de los operadores lógicos como and (&&), y or (||) para poder evaluar mas de un tipo de dato en un sentencia.

Estado_bool: indica los estados de un tipo de datos boolean.

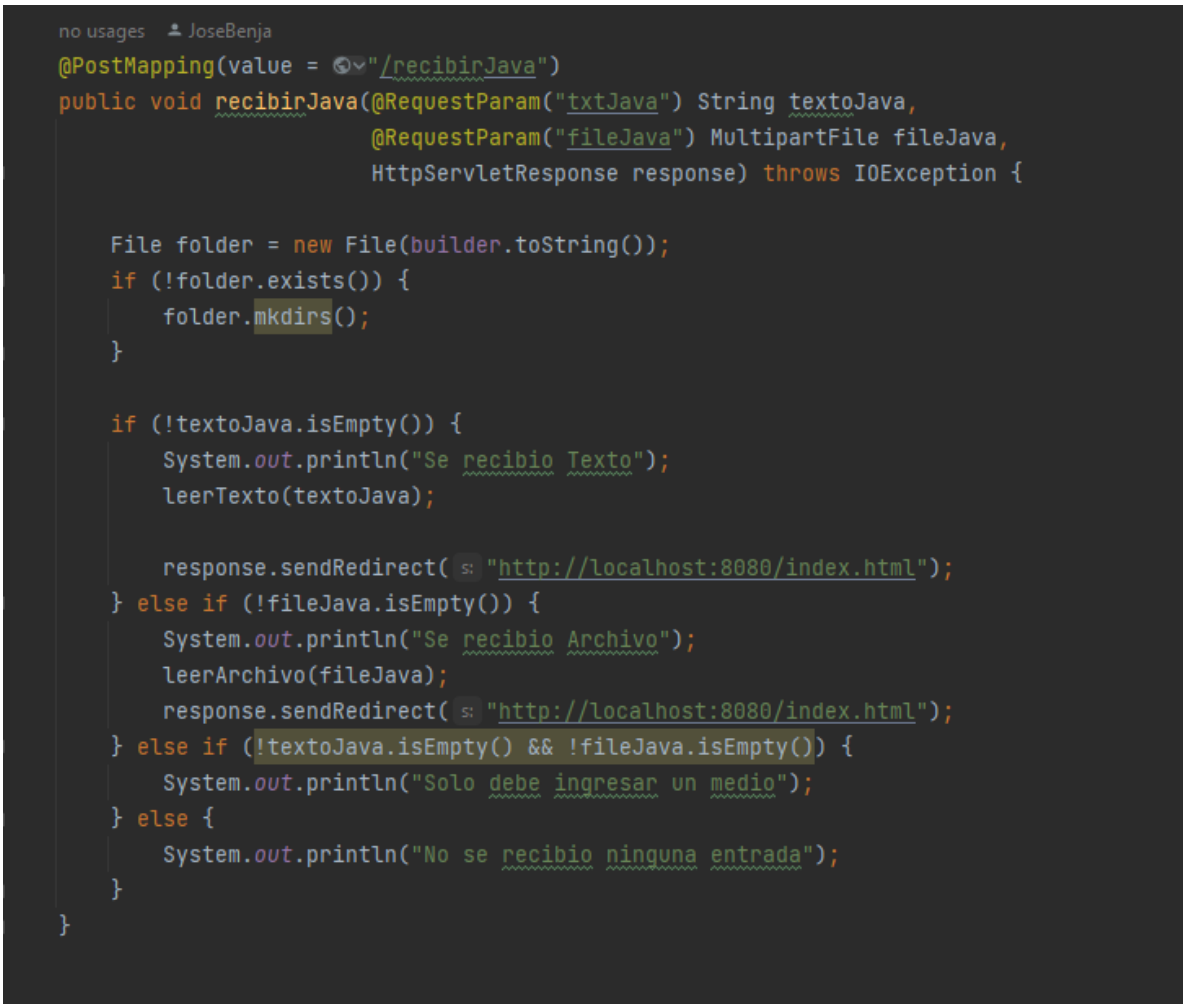
Operador: Nos lista los operadores como, +, -, * y /.

Privacidad: nos indica el tipo de privacidad de una clase , método o variable.

```
comparacion             ::= IDENTIFICADOR operador_comparacion IDENTIFICADOR operador_logico
                        |   IDENTIFICADOR operador_comparacion DIGITO operador_logico
                        |   DIGITO operador_comparacion DIGITO operador_logico
                        |   DIGITO operador_comparacion IDENTIFICADOR operador_logico;

operador_comparacion    ::= MAYOR:operador_comparacion_val
                        |   MAYOR_IGUAL:operador_comparacion_val
                        |   MENOR:operador_comparacion_val
                        |   MENOR_IGUAL:operador_comparacion_val
                        |   DIFERENTE:operador_comparacion_val
                        |   IGUAL:operador_comparacion_val;

operador_logico         ::= AND:operador_logico_val comparacion
                        |   OR:operador_logico_val comparacion
                        |   /* ε */;

estado_bool             ::= VERDAD:estado_val
                        |   FALSO:estado_val;

operador                ::= RESTA:operador_val
                        |   MULTI:operador_val
                        |   DIVIDIR:operador_val
                        |   POTENCIA:operador_val;

privacidad              ::= PUBLICO:privacidad_val
                        |   PRIVADO:privacidad_val;
```

CONTROLADOR DE LA APLICACIÓN WEB

El método recibirJava se encarga de obtener los datos de nuestra vista web, en este caso el campo texto o archivo según corresponda, reconoce cual es el tipo de entrada que tiene y evalua a que método enviarlo.



```
no usages  JoseBenja
@PostMapping(value = "/recibirJava")
public void recibirJava(@RequestParam("txtJava") String textoJava,
                        @RequestParam("fileJava") MultipartFile fileJava,
                        HttpServletResponse response) throws IOException {

    File folder = new File(builder.toString());
    if (!folder.exists()) {
        folder.mkdirs();
    }

    if (!textoJava.isEmpty()) {
        System.out.println("Se recibio Texto");
        leerTexto(textoJava);

        response.sendRedirect( s: "http://localhost:8080/index.html");
    } else if (!fileJava.isEmpty()) {
        System.out.println("Se recibio Archivo");
        leerArchivo(fileJava);
        response.sendRedirect( s: "http://localhost:8080/index.html");
    } else if (!textoJava.isEmpty() && !fileJava.isEmpty()) {
        System.out.println("Solo debe ingresar un medio");
    } else {
        System.out.println("No se recibio ninguna entrada");
    }
}
```

Método leerArchivo()

Recibe el archivo .java y lo guarda en una ruta especifica.

```
1 usage   JoseBenja
public ArrayList<String> leerArchivo(MultipartFile file) {
    byte[] bytes;
    try {
        bytes = file.getBytes();
    } catch (IOException e) {
        throw new RuntimeException(e);
    }

    Path path = Paths.get(nomArchivo);
    try {
        Files.write(path, bytes);
        return compilarCup();
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Método leerTexto()

Recibe el texto ingresado de java y lo convierte en un archivo .java para analizarlo

```
public ArrayList<String> leerTexto(String txtJava) {
    try (BufferedWriter escritor = new BufferedWriter(new FileWriter( fileName: "C:/Users/VICTUS//IntelIJIDEA_proyectos/Proyecto-Compiladores/codigoCompilar.java"))) {
        // Escribe la entrada de texto en el archivo
        escritor.write(txtJava);
        System.out.printf("Escritor  " + escritor);
        return compilarCup();
    } catch (IOException e) {
        System.out.println("Ocurrió un error al crear el archivo: " + e.getMessage());
    }
    return null;
}
```

Método compilarCup()

Se encarga de llamar el archivo codigoCompilar.java y lo analiza en la clase Lexer que contiene los lexemas a evaluar de nuestro analizar léxico, luego al obtener los valores del archivo cup los guarda en un ArrayList de tipo String y los retorna.

```
2 usages  JoseBenja
public ArrayList<String> compilarCup() {
    try {
        System.out.printf(nomArchivo);
        Reader reader = new FileReader( fileName: "C:/Users/VICTUS//IntelIJIDEA_proyectos/Proyecto-Compiladores/codigoCompilar.java");
        parser p = new parser(new Lexer(reader));
        Object result = p.parse().value;

        ArrayList<String> inverso = new ArrayList<String>(p.resultados);
        Collections.reverse(inverso);
        return inverso;

    } catch (FileNotFoundException ex) {
        Logger.getLogger(CompiladorController.class.getName()).log(Level.SEVERE,  msg: "Error al generar en cup ", ex);
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
    return null;
}
```