Manual de Usuario

Requerimientos

Sistema Operativo (Windows o Linux)

Java 1.8.0_201 o compatibles.

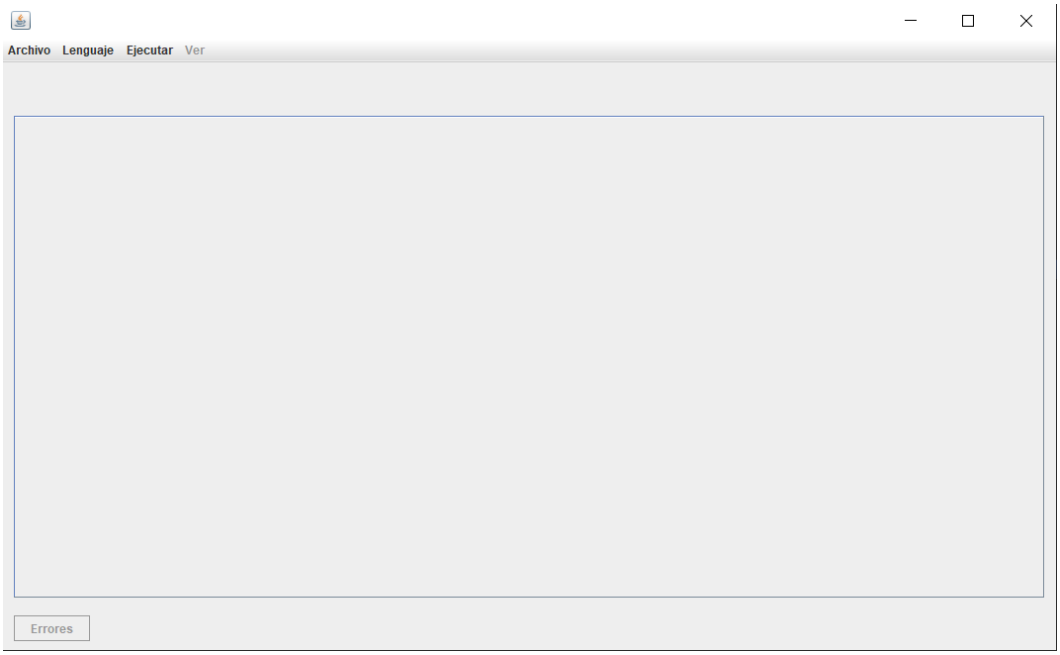
Jar ejecutable o proyecto completo.

Los archivos deben de tener la extensión del lenguaje que este en uso.

Descripción

El programa consiste en un editor de cadenas de entradas y también genera lenguajes a partir de un archivo de entrada. En este archivo se estructura el lenguaje que el usuario desea utilizar. Posteriormente ingresa una cadena y es compilada.

Ventana Inicio



En esta ventana es donde se ingresa el texto para el funcionamiento del lenguaje.

Pestaña Archivo

Nuevo: Despliega una nueva pestaña, en la cual el usuario podrá editar un archivo.

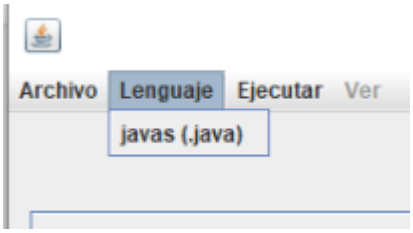
Abrir: Permite abrir un archivo para que este pueda ser editado.

Guarda: Permite guardar un archivo en la ubicación que el usuario desea, el archivo deberá de guardarse con la extensión del lenguaje seleccionado.

Guardar como: Esta opción permite guardar un archivo previamente almacenado con diferente nombre, el archivo deberá de guardarse con la extensión del lenguaje seleccionado.

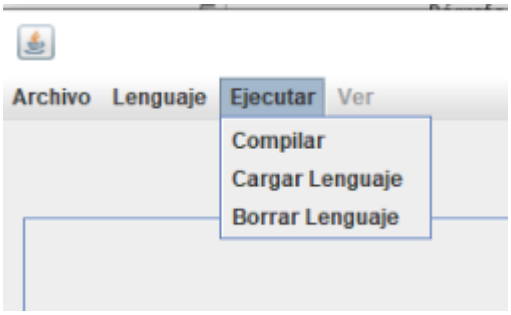
Salir: Con esta opción se cierra la aplicación.

Pestaña Lenguaje



Lenguaje: En este submenú se deben mostrar todos los lenguajes que la aplicación soporta y permite seleccionar el lenguaje con el que se va a trabajar.

Pestaña Ejecutar



Compilar: A través de esta opción se podrán compilar la o las cadenas de entrada ingresadas por el usuario. Se deberá tomar en cuenta que previamente el usuario deberá seleccionar un lenguaje para compilar dichas cadenas. El contenido que se tomará en cuenta para realizar este proceso será el que esté en la pestaña activa.

Cargar Lenguaje: Por medio de esta opción se podrán agregar nuevos lenguajes al repositorio de la aplicación, los cuales posteriormente deberán aparecer en el menú principal para que puedan ser seleccionados por el usuario para realizar el proceso de compilación. Se deberá tomar en cuenta que el lenguaje no podrá ser cargado a la aplicación sin estar libre de errores (léxicos, sintácticos y semánticos). El contenido que se tomará en cuenta para realizar este proceso será el que esté en la pestaña activa. Si ya existe una carga.

Borrar Lenguaje: Por medio de esta opción se podrá eliminar un lenguaje del repositorio de la aplicación el cual también deberá ser eliminado del menú principal.

Pestaña Ver

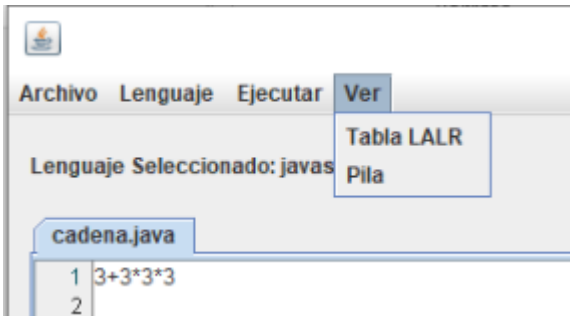
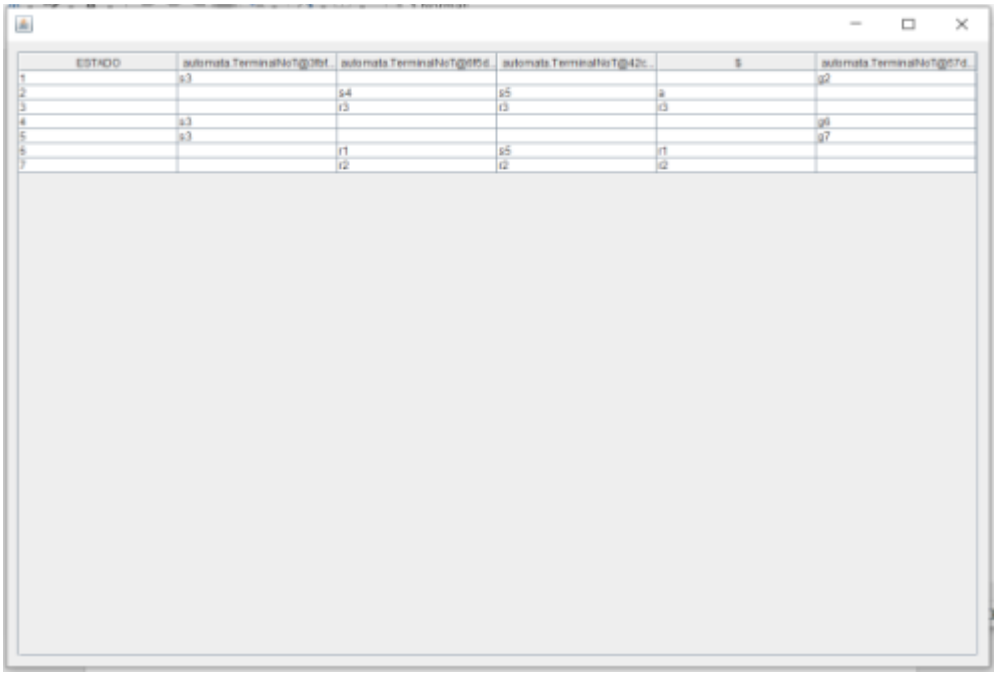
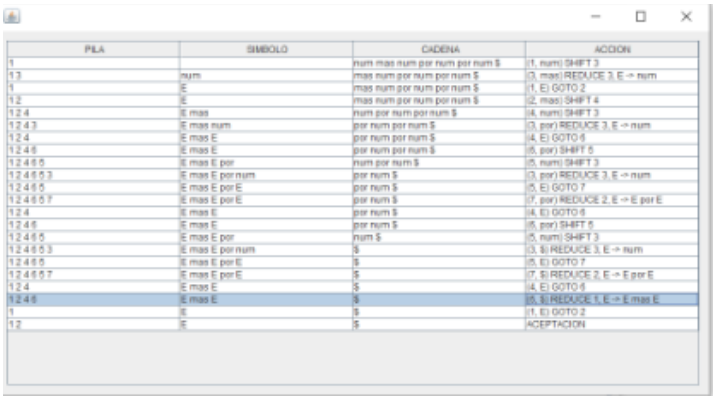


Tabla LALR: Esta opción despliega la tabla LALR, la cual fue generada en el momento que seleccionó un lenguaje para compilar una cadena de entrada.



Pila: Esta opción despliega pantalla con el contenido de la pila que se utilizó para reconocer una cadena de entrada previamente ingresada por el usuario.



Lenguaje

Estructura, Sintaxis

La estructura del archivo para agregar un nuevo lenguaje está compuesta por cinco secciones: información del lenguaje, código fuente, expresiones regulares, símbolos terminales y no terminales, finalmente se define la gramática.

Información Lenguaje

%%

Código fuente

%%

Expresiones Regulares

%%

Símbolos Terminales y No Terminales

%%

Gramática

Se puede realizar comentarios en cualquier sección del archivo usando de la siguiente

forma:

- Una línea: Esto permite hacer un comentario en una línea.

  // Este es un comentario

- Varias líneas: esto permite realizar comentarios en una o más líneas del archivo.

  /* Este es un comentario */

Información Lenguaje

En esta sección se deberá de insertar información acerca del lenguaje que se está definiendo en el archivo, los campos de esta sección son los siguientes:

Nombre: Nombre del lenguaje

Versión: la versión del lenguaje

Autor: el autor del lenguaje

Lanzamiento: el año del lanzamiento del lenguaje.

Extension: extensión de los archivos de cada lenguaje.

Ejemplo

nombre: java;
version: 11.0.7;
autor: sun Microsystems;
lanzamiento: 1996;
extension: java;

Sección de Código Fuente

En esta sección deberá ir el código fuente java de las funciones o procedimientos que serán utilizados en las reglas semántica.

```
List<String> listString=new ArrayList<>();

public void addString(String valor){
        listString.add(valor);
}

public Integer suma(Integer numero1,Integer numero2){
        return numero1+numero2;
}
```

Sección de Expresiones Regulares

En esta sección deberán ser declaradas las expresiones regulares para establecer la forma que tendrá cada token que es válido para un determinado lenguaje.

Expresión Regular	Significado
α?	α puede venir o no puede venir.
α*	α puede venir cero o más veces.
α+	α puede venir una o más veces.
αβ	Concatenación implícita entre αyβ .
α|β	Puede venir α o β.
[0-9]	Cualquier número entre 0 y 9.

[a-z]	Cualquier letra entre a y z.
\n	Nueva línea.
\t	Tabulador.
\b	Espacio en blanco.
“α”	Agrupa a uno o más caracteres.
() ó []	Agrupa expresiones regulares.

La sintaxis de esta sección está compuesta por un identificador y la expresión regular que

corresponde al identificador.

Identificador: el identificador será el nombre que tendrá cada token que cumpla con la expresión regular especificada.

Expresión Regular: Permite establecer la forma que tendrá cada token que es válido para el lenguaje que se está definiendo.

<identificador1> = <expresión regular1>;

<identificador2> = <expresión regular2>;

Ejemplo:

```
palabra = [a-z]+;

entero = [0-9]+;

real = [0-9]+((.)[0-9]+)?;

más = “+”;

menos = “-”;

& = [\n\t]; /* Significa que cuando se encuentre este token deberá ser ignorado */
```

Se puede emplear el identificador **“&”** para indicar que la expresión regular deberá ser

ignorada, cuando se analice un archivo.

Ejemplo:

& = [\n\t]; /* Significa que cuando se encuentre este token deberá ser ignorado */

Sección de símbolos terminales y no terminales

En esta sección se deberán declarar los símbolos terminales y no terminales del lenguaje

que se esté definiendo.

Sintaxis:

<no terminal | terminal> [entero|real|cadena] id1[,id2,……,idn];

Ejemplo:

terminal por, div;

terminal mas, menos;

terminal entero;

no terminal A, B;

no terminal S;

Sección de gramática

En esta sección se definirá la gramática del lenguaje y las reglas semánticas asociadas a cada producción. Las reglas semánticas van encerradas entre llaves al final de la producción y antes del punto y coma, y estas son opcionales para cada producción, este código se deberá de ejecutar al momento de reducir la producción. Cada símbolo No Terminal y Terminal puede tener una estructura o variable asociada del tipo declarado previamente, la cual se declarará con “:nombre” y existirá un atributo RESULT asociado a cada lado izquierdo de las producciones.

Sintaxis:

no terminal :: [<terminal | no terminal>[:nombre] …….] [{reglas semánticas}];

Ejemplos

S :: E:val {println(“Resultado = ”+val);};

E :: E:val menos E:val2 {RESULT=val - val2;};

E :: E:val mas E:val2 {RESULT=val + val2;};

E :: E:val por E:val2 {RESULT=val * val2;};

E :: E:val div E:val2 {RESULT=val / val2;};

E :: digito:val {RESULT=val;};

Ejemplo de todas las secciones:

```
//Informacion del lenguaje
nombre: java;
version: 11.0.7;
autor: sun Microsystems;
lanzamiento: 1996;
extension: java
%% //codigo fuente
List<String> listString=new ArrayList<>();
public void addString(String valor){ }
%% //Sección de expresiones regulares
palabra = [a-z]+;
entero = [0-9]+;
real = [0-9]+((.)[0-9]+)?;
más = “+”;
menos = “-”;
& = [\n\t]; /* Significa que cuando se encuentre este token deberá ser ignorado */
%% //Seccion de simbolos terminales y no terminales
terminal por, div;
terminal más, menos;
terminal entero;
no terminal A, B;
no terminal S;
S :: E:val {printf(“Resultado = %d”,val);};
E :: E:val menos E:val2 {RESULT=val - val2;};
E :: E:val mas E:val2 {RESULT=val + val2;};
E :: E:val por E:val2 {RESULT=val * val2;};
E :: entero:val {RESULT=val;};
```