

|  |
| --- |
| 14/12/21 |

LENGUAJES AUTÓTAS II

***Profesor: Olga Gabriela Delgado Cansino***



Compilador Léxico y Sintáctico



Integrantes del Equipo:

**Andrea Jacqueline Bautista Salas**

**Guadalupe Nataly Ruiz Torres**

**Victor Emanuel Marquez Echevarria**

**Carlos Alejandro Villareal Lopez**

**Hugo Alejandro Quezada Rodríguez**

**Jesús Uriel Teran Vazquez**

Índice

[Desarrollo de un Compilador 1](#_Toc90384121)

[Objetivo Principal. 1](#_Toc90384122)

[Tipos de Analizadores 2](#_Toc90384123)

[Identificadores / Alfabeto: 2](#_Toc90384124)

[Operadores: 3](#_Toc90384125)

[o Introducir Comentarios: 3](#_Toc90384126)

[Delimitadores: 3](#_Toc90384127)

[Palabras Reservadas: Comunes del lenguaje Java. 4](#_Toc90384128)

[Palabras Reservadas: Proyecto compilador léxico. 4](#_Toc90384129)

[Error Léxico: 5](#_Toc90384130)

[Proyecto: Descripción. 5](#_Toc90384131)

[Concepto: Analizador Sintáctico 6](#_Toc90384132)

[Funciones: Analizador Sintáctico 6](#_Toc90384133)

[Gramática utilizada por un analizador sintáctico 7](#_Toc90384134)

[Derivaciones 8](#_Toc90384135)

[Combinaciones 9](#_Toc90384136)

[Notación BNF 10](#_Toc90384137)

[Manual Técnico del Compilador 12](#_Toc90384138)

[IDE para desarrollo del programa 12](#_Toc90384139)

[Librería JFlex.jar 12](#_Toc90384140)

[Estructura del Compilador: Elementos. 12](#_Toc90384141)

[Creación y contenido: 13](#_Toc90384142)

[Lexer.flex 13](#_Toc90384143)

[Contenido: Lexer.flex y Tokens.java 14](#_Toc90384144)

[Descripción: Lexer.flex 15](#_Toc90384145)

[Descripción: Tokens.java 15](#_Toc90384146)

[Nombres asignados de los operadores, palabras reservadas y delimitadores: 15](#_Toc90384147)

[Descripción: generadora.java 16](#_Toc90384148)

[Generar Lexer: 17](#_Toc90384149)

[Precauciones: 17](#_Toc90384150)

[Descripción: Lexer.java 18](#_Toc90384151)

[LexerCup.flex 19](#_Toc90384152)

[Syntax.cup 20](#_Toc90384153)

[Botón Analizador Sintáctico: 21](#_Toc90384154)

[Diseño de interfaz gráfica 22](#_Toc90384155)

[Color de paneles y botones: 23](#_Toc90384156)

[Áreas de texto: 24](#_Toc90384157)

[Texto de etiquetas: 24](#_Toc90384158)

[Función de los botones: 25](#_Toc90384159)

[Error Léxico: 25](#_Toc90384160)

[Ejemplo de funcionamiento de interfaz gráfica: 26](#_Toc90384161)

[Manual de Usuario 27](#_Toc90384162)

[Ejecutar programa: 27](#_Toc90384163)

[Uso de Programa: 28](#_Toc90384164)

[Nombres asignados de los operadores, palabras reservadas y delimitadores: 28](#_Toc90384165)

[Elementos de la interfaz: 29](#_Toc90384166)

# Desarrollo de un Compilador

## Objetivo Principal.

##### El objetivo de este proyecto es realizar un compilador diseñando las herramientas necesarias para así poder elaborar un programa analizador de los datos ingresados por el usuario; en otras palabras: diseñaremos un programa con la capacidad de construir otros programas, de manera que, interprete un lenguaje de alto nivel (el del usuario), traduciéndolo a otro de bajo nivel (lenguaje máquina).

## Tipos de Analizadores

##### Si un compilador tuviera que procesar sólo programas correctos, su diseño e implementación se simplificarían mucho. Las primeras versiones de los programas suelen ser incorrectas, y un buen compilador debería ayudar al programador a identificar y localizar errores. Es más, considerar desde el principio el manejo de errores puede simplificar la estructura de un compilador y mejorar su respuesta a los errores. Los errores en la programación pueden ser de los siguientes tipos:

##### Léxicos, producidos al escribir mal un identificador, una palabra clave o un operador.

##### Sintácticos, por una expresión aritmética o paréntesis no equilibrados.

##### Semánticos, como un operador aplicado a un operando incompatible.

##### Lógicos, puede ser una llamada infinitamente recursiva.

##### De corrección, cuando el programa no hace lo que el programador realmente deseaba.

##### Los compiladores actuales se centran en el reconocimiento de los tres primeros tipos de errores; ya que resulta evidente que los errores de corrección no pueden ser detectados por un compilador, ya que en ellos interviene el concepto abstracto que el programador tiene sobre el programa que construye.

## Identificadores / Alfabeto:

##### Abarca todas las letras del abecedario, desde la A a la Z, tanto mayúsculas como y minúsculas, y caracteres alfanuméricos, dígitos que van del 0 al 9:

[A,..,Z],[a,..,z],[0,..,9]

## Operadores:

Incluiremos los caracteres del código ASCII, estos son:

Operador: Aquél que realiza un efecto de concatenación.

|  |
| --- |
| < , > , <<, >> = , >=, <=, !=, \* , &, && , % , | , / , +, -, ++, --, ==, ‘’, “”, \, : |

## Introducir Comentarios:

Sustituir “//” por “&&”:

## Delimitadores:

|  |
| --- |
| “,”, “;”, ( ) , [ ] , { } |

Símbolos utilizados como separadores de las distintas construcciones de un lenguaje de programación.

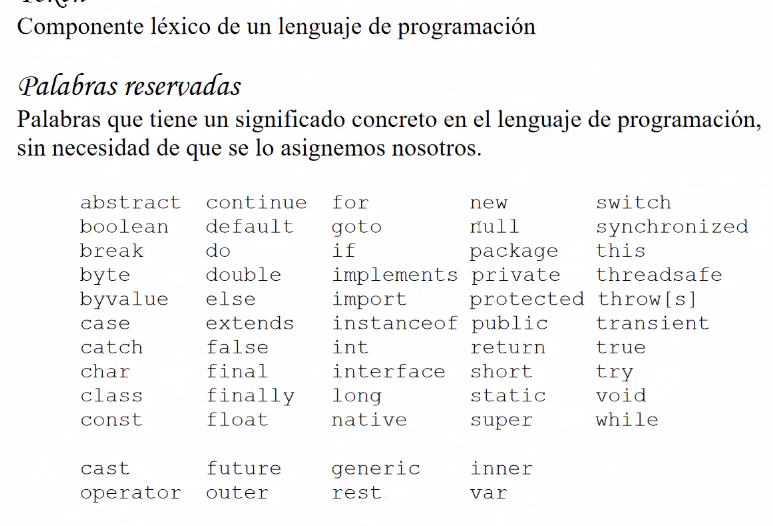
* () PARENTESIS: Lista de parámetros en la definición y llamada a métodos, precedencia en expresiones para control de flujo y conversiones de tipo.
* {} LLAVES: Inicialización de arrays, bloques de código, clases métodos y ámbitos locales.
* [] CORCHETES: Para uno de arreglos (arrays).
* (“;“) PUNTO Y COMA: Separador de sentencias.
* (“,”) COMA: Identificadores consecutivos en una declaración de variables y sentencias encadenadas dentro de una sentencia ***for.***
* (“.”) PUNTO: Separado de nombres de paquetes, subpaquetes y clases; separador entre variables y métodos/miembros.

## **Palabras Reservadas: Comunes del lenguaje Java**.

Emplearemos las palabras traducidas al español y convertidas al femenino.

Ejemplo: “**abstract**” cambiará a “**abstracta**”.

Switch: cambiara.



## Palabras Reservadas: Proyecto compilador léxico.

##### Hasta el momento tenemos considerados los siguientes posibles caracteres, dejando espacios en blando para nuevas consideraciones a emplear en nuestro proyecto.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Java*** | ***Nuevo*** | ***Java*** | ***Nuevo*** | ***Java*** | ***Nuevo*** |
| if | entonces | int | entera | boolean | boleana |
| else | ademas | char | caracter | break | rota |
| while | mientrasque | float | flotante | private | privada |
| for | para | double | doble |  |  |
| new | nueva |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## Error Léxico:

Al momento de no reconocer alguno de los símbolos introducidos al programa, éste emitirá un mensaje de alerta:

**“Error! Símbolo no definido”.**

## Proyecto: Descripción.

##### El objetivo principal es realizar un analizador léxico el cual pueda reconocer los elementos que se le introducen. El ejemplo que se muestra a continuación está realizado con el lenguaje java.

Graphical user interface, text

Description automatically generated

# Concepto: Analizador Sintáctico

##### Como concepto: es la fase del analizador que se encarga de chequear la secuencia de tokens que representa al texto de entrada, en base a una gramática dada. En caso de que el programa de entrada sea válido, suministra el árbol sintáctico que lo reconoce en base a una representación computacional. Este árbol es el punto de partida de la fase posterior de la etapa de análisis: el analizador semántico.

## Funciones: Analizador Sintáctico

##### Incorpora acciones semánticas en las que colocar el resto de las fases del compilador (excepto el analizador léxico): desde el análisis semántico hasta la generación de código.

##### Informa de la naturaleza de los errores sintácticos que encuentra e intenta recuperarse de ellos para continuar la compilación.

##### Controla el flujo de tokens reconocidos por parte del analizador léxico.

##### En definitiva, realiza casi todas las operaciones de la compilación, dando lugar a un método de trabajo denominado compilación dirigida por sintaxis.

##### El manejo de errores de sintaxis es el más complicado desde el punto de vista de la creación de compiladores. Nos interesa que cuando el compilador encuentre un error, no cancele definitivamente la compilación, sino que se recupere y siga buscando errores. Recuperar un error no quiere decir corregirlo, sino ser capaz de seguir construyendo el árbol sintáctico a pesar de los errores encontrados. En vista de esto, el manejador de errores de un analizador sintáctico debe tener como objetivos:

##### Indicar los errores de forma clara y precisa. Debe informar mediante los correspondientes mensajes del tipo de error y su localización.

##### Recuperarse del error, para poder seguir examinando la entrada.

##### Distinguir entre errores y advertencias. Las advertencias se suelen utilizar para informar sobre sentencias válidas pero que, por ser poco frecuentes, pueden constituir una fuente de errores lógicos.

##### No ralentizar significativamente la compilación.

## Gramática utilizada por un analizador sintáctico

Gramática Propuesta:

|  |  |  |
| --- | --- | --- |
| ① A | → | A + C |
| ② | | | **C** |
| ③ C | → | **C** \* **R** |
| ④ | | | **R** |
| ⑤ R | → | No |
| ⑥ | | | numero |
| ⑦ | | | (**A**) |

Las derivaciones a izquierda quedarían: (primera manera de representarlo)

|  |
| --- |
| A **→ A + A → A\*A + A →** No **\* A + A →** No**1 \*** No **2 + A →** No **1 +** No **2 +** No **3** |

Sustituyendo a la primera “A” quedaría secuencialmente de la siguiente manera:

(Segunda manera de representarlo)

**A → A+A**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| A | → | A | | | + | A |
|  | | | | | | |
|  | → | **A** | **\*** | **A** | + | **A** |
|  | → | No | **\*** | A | + | A |
|  | → | No 1 | **\*** | No 2 | + | A |
|  | → | No 1 | **\*** | No 2 | + | No 3 |

## Derivaciones

(No**a**+ No**b**) \* No**a**+ No**b**

A**1**

A**2**

C**4**

+

C**3**

C**4**

R**5**

R**5**

(

)

A**1**

A**2**

C**4**

+

R**5**

No**b**

No**a**

C**4**

R**5**

R**5**

No**b**

No**a**

\*

## Combinaciones

Se pueden combinar todas las posibles sentencias reconocibles por la gramática propuesta que consiste en reemplazar con valores que un elemento representa.

* Gramática propuesta:

**No** \* C + A

**numero** \* C + A

**(A)** \* C + A

**R** \* C + A

|  |  |  |
| --- | --- | --- |
| ① A | → | C + A |
| ② | | | **C** |
| ③ C | → | **R \* C** |
| ④ | | | **R** |
| ⑤ R | → | No |
| ⑥ | | | numero |
| ⑦ | | | (**A**) |

**R** \* C

**No** \* C

**(A)** \* C

**No** + A

**(A)** + A

**numero** + A

**R** + A

C + A

**numero** \* C

C

R

(A)

numero

No

A

## Notación BNF

Para la descripción de un lenguaje de programación.

|  |  |  |
| --- | --- | --- |
| <programa> | **=** | <listaDeFunciones> |
| <listaDeFunciones> | **=** | <función> | <listaDeFunciones> <funcion> |
| <funcion> | **=** | FUNC<variable>(<ListaDeParametros>)<Sentencia> |
| <listadeDeParametros> | **=** | <ListaDeVariable> | **Ɛ** |
| <ListaDeVariable> | **=** | <variable> | <ListaDeVariables> **,** <variable> |
| <alfanumerico> | **=** | <letra> | <digito> |
| <letra> | **=** | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| <digito> | **=** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| <sentencia> | **=** | <sentenciaDeAsignacion>|<sentenciaDeRetorno> |<sentenciaDeImpresion>|<sentenciaNula>|<sentenciaCondicional> | sentenciaWhile> | <bloque> |
| <sentenciaDeAsignacion> | **=** | <variable> **=** <expresion> |
| <expresión> | **=** | <expresión><operadorBinario><expresión> |<operadorUnario><expresion>|(<expresion>)|<entero>|<variable> (<listaDeArgumentos>) |
| <operadorBinario> | **=** | + | - | \* | / | ~ |
| <operadorUnario> | **=** | - |
| <operadorIgualA> | **=** | == |
| <operadorMenorQue> | **=** | < |
| <operadorMayorQue> | **=** | > |
| <operadorMayorOigualA> | **=** | >= |
| <operadorMenorOigualA> | **=** | <= |
| <operadorDesplazamientoIzquierda> | **=** | << |
| <operadorDesplazamientoDerecha> | **=** | >> |
| <operadorDiferenteA> | **=** | != |
| <operadorTambien> | **=** | & |
| <operadorComentario> | **=** | && |
| <operadorModula> | **=** | % |
| <operadorComentada> | **=** | # |
| <operadorObien> | **=** | | |
| <operadorIncremento> | **=** | ++ |
| <operadorDecremento> | **=** | -- |
| <operadorBinarioABinario> | **=** | ~ |
| <operadorComilladoSimple> | **=** | ‘ |
| <operadorComilladoDoble> | **=** | “ |
| <operadorContinuaY> | **=** | ^ |
| <operadorComa> | **=** | , |
| <operadorPuntoYcoma> | **=** | ; |
| <operadorParentesisA> | **=** | ( |
| <operadorParentesisB> | **=** | ) |
| <operadorCorcheteA> | **=** | [ |
| <operadorCorcheteB> | **=** | ] |
| <operadorLlaveA> | **=** | { |
| <operadorLlaveB> | **=** | } |
| <entero> | **=** | <digito>|<entero><digito> |
| <listaDeArgumentos> | **=** | <listaDeExpresiones>  | <expresiones> , <listaDeExpresiones> |
| <sentenciaDeImpresion> | **=** | PRINT<listaDeImpresion> |<listaDeImpresion>,<elementoDeImpresion> |
| <elementoDeImpresion> | **=** | <Expresion> | “<Texto>” |
| <texto> | **=** | <caracter> | <caracter><texto> |
| <carácter> | **=** | <caracterImprimible> | <caracterEscapado> |
| <caracterImprimible | **=** | “Cualquier carácter ASCII imprimible” |
| <caracterEscapado> | **=** | \n |
| <sentenciaDeRetorno> | **=** | RETURN <expresion> |
| <sentenciaCondicional> | **=** | **If**<expresion>**then**<sentencia>**fi**  **|if**<expreison>**then**<sentencia>**else**<sentencia>**entonces**<sentencia>fi **|if**<expreison>**then**<sentencia>**else**<sentencia>**entonces**<sentencia> **ademas**<sentencia>**fi** **|for**<expresión>**then**<sentencia>**fi** **|for**<expresión>**then**<sentencia>**para**<sentencia>**fi** |
| <sentenciaWhile> | **=** | **while**<expresion>**do**<sentencia>**done** |
| <bloque> | **=** | {<listaDeDeclaraciones><listaDeSentencias>} |
| <listaDeDeclaraciones> | **=** | <declaración><listaDeDeclaraciones>| **Ɛ** |
| <declaracion> | **=** | VAR<listaDeVariables> |
| <listaDeSentencias> | **=** | <sentencia>|<listaDeSentencias><sentencia> |

# Manual Técnico del Compilador

## IDE para desarrollo del programa

##### Descargamos e instalamos la versión de Java NetBeans 8.2.

##### (**Nota**: En caso de fallos, tener cuidado con la versión de jdk que se está ejecutando desde java NetBeans 8.2. Procurar que esta versión no sea la más reciente).

## Librería JFlex.jar

Descargaremos la librería JFlex.jar y se tomará en cuenta la dirección donde éste se alojará.

## Estructura del Compilador: Elementos.

|  |  |
| --- | --- |
| Graphical user interface, text, application  Description automatically generated | El Programa está compuesto por un paquete de nombre “código”. Dentro contiene una serie de clases, una librería, un archivo anteriormente vacío, y un jFrame.  Archivo (anteriormente vacío):   * Lexer.flex   Clases:   * Tokens.java * generadora.java * Lexer.java   Librería:   * jFlex.jar   jFrame:   * frmAnalizador.java |

## Creación y contenido:

### Lexer.flex

|  |
| --- |
| Creación de documento: Una vez creado el proyecto y el paquete “código”; presionaremos el segundo botón del ratón para desglozar las siguientes opciones:Crear un archivo nuevo desde: new / other / other / Empty File. |

### Contenido: Lexer.flex y Tokens.java

|  |  |
| --- | --- |
| **Lexer.felx**  package codigo;  import static codigo.Tokens.\*;  %%  %class Lexer  %type Tokens  L=[a-zA-Z\_]+  D=[0-9]+  espacio=[ ,\t,\r]+  %{  public String lexeme;  %}  %%  if {lexeme = yytext(); return entonces;}  else {lexeme = yytext(); return ademas;}  while {lexeme=yytext(); return mientrasque;}  for {lexeme=yytext(); return para;}  int {lexeme = yytext(); return entera;}  char {lexeme = yytext(); return caracter;}  float {lexeme = yytext(); return flotante;}  double {lexeme = yytext(); return doble;}  boolean {lexeme = yytext(); return boleana;}  {espacio} {/\*Ignore\*/}  "//".\* {/\*Ignore\*/}  "\n" {return Linea;}  "=" {lexeme = yytext(); return asignaA;}  "+" {lexeme = yytext(); return Suma;}  "-" {lexeme = yytext(); return Resta;}  "\*" {lexeme = yytext(); return Multiplicacion;}  "/" {lexeme = yytext(); return Division;}  "==" {lexeme = yytext(); return igualA;}  "<" {lexeme = yytext(); return menorQue;}  ">" {lexeme = yytext(); return mayorQue;}  ">=" {lexeme = yytext(); return mayorOigualA;}  "<=" {lexeme = yytext(); return menorOigualA;}  "<<" {lexeme = yytext(); return desplazaIzquierda;}  ">>" {lexeme = yytext(); return desplazaDerecha;}  "!=" {lexeme = yytext(); return diferenteA;}  "&" {lexeme = yytext(); return tambien;}  "&&" {lexeme = yytext(); return comentario;}  "%" {lexeme = yytext(); return modula;}  "|" {lexeme = yytext(); return obien;}  "++" {lexeme = yytext(); return incremento;}  "--" {lexeme = yytext(); return decremento;}  "''" {lexeme = yytext(); return comilladoSimple;}  """" {lexeme = yytext(); return comilladoDobleA;}  ":" {lexeme = yytext(); return dosPuntos;}  "," {lexeme = yytext(); return coma;}  ";" {lexeme = yytext(); return puntoYcoma;}  "(" {lexeme = yytext(); return parentesisA;}  ")" {lexeme = yytext(); return parentesisB;}  "[" {lexeme = yytext(); return corcheteA;}  "]" {lexeme = yytext(); return corcheteB;}  "{" {lexeme = yytext(); return llaveA;}  "}" {lexeme = yytext(); return llaveB;}  {L}({L}|{D})\* {lexeme=yytext(); return Identificador;}  ("(-"{D}+")")|{D}+ {lexeme=yytext(); return Numero;}  . {return ERROR;} | **Tokens.java**  package codigo;  public enum Tokens {    Reservadas,  entonces,  ademas,  mientrasque,  para,  entera,  caracter,  flotante,  doble,  boleana,  Linea,  asignaA,  Igual,  Suma,  Resta,  Multiplicacion,  Division,  igualA,  menorQue,  mayorQue,  mayorOigualA,  menorOigualA,  desplazaIzquierda,  desplazaDerecha,  diferenteA,  tambien,  comentario,  modula,  obien,  incremento,  decremento,  comilladoSimple,  comilladoDobleA,  dosPuntos,  coma,  puntoYcoma,  parentesisA,  parentesisB,  corcheteA,  corcheteB,  llaveA,  llaveB,  Identificador,  Numero,  ERROR  } |

### Descripción: Lexer.flex

En el archivo Lexer.flex se redacta los operadores, delimitadores y palabras reservadas, mediante “ ***lexeme = yytext();***” y donde después de “ ***return*** “ se destribe el elemento que devolverá.

### Descripción: Tokens.java

Se crea el método “Tokens” donde se incluyen los nombres que se asignaron en el archivo Lexer.flex para así poder ser llamados cuando se les requiera.

### Nombres asignados de los operadores, palabras reservadas y delimitadores:

|  |  |
| --- | --- |
| * if → entonces * else → ademas * while → mientrasque * for → para * int → entera * char → caracter * float → flotante * double → doble * boolean → boleana * "=" → asignaA * "+" → Suma * "-" → Resta * "\*" → Multiplicacion * "/" → Division * "==" → igualA * "<" → menorQue * ">" → mayorQue * ">=" → mayorOigualA * "<=" → menorOigualA | * "<<" → desplazaIzquierda * ">>" → desplazaDerecha * "!=" → diferenteA * "&" → tambien * "&&" → comentario * "%" → modula * "|" → obien * "++" → incremento * "--" → decremento * " **''** " → comilladoSimple * " **""** " → comilladoDobleA * " : " → dosPuntos * "," → coma * ";" → puntoYcoma * "(" → parentesisA * ")" → parentesisB * "[" → corcheteA * "]" → corcheteB * "{" → llaveA * "}" → llaveB |

## Descripción: generadora.java

##### Éste es un archivo ejecutable que contiene dentro de sí la ruta para acceder a la librería “JFlex.jar”, con el cual mediante el método ***generarLexer*** *y* la librería importada “***java.io.File***”, se tendrá acceso a la librería JFlex.jar.

Graphical user interface, application

Description automatically generatedGraphical user interface, application

Description automatically generated

Importamos dentro de las librerías **jFlex.jar**, **java-cup11a.jar**, y **java\_cup.jar**.

### Generar Lexer:

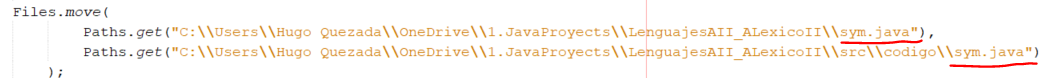
Table

Description automatically generated

##### En este apartado, conforme a las rutas; se generan los archivos con terminación “.java”, como: **lexer.java**, **lexer.java** y finalmente **sym.java**. En los cuales se establecen los símbolos predefinidos por el programador. También, cada que se realicen cambios a los archivos **Lexer.flex, Tokens,** se tendrá que volver a ejecutar el programa para que éste se actualice.

### Precauciones:

##### Como se observa, tenemos rutas muy similares, que generan el mismo archivo en diferentes lugares. Si se descuida, el programa buscará generarlos en dentro de la misma carpeta generando conflicto, y por ello, es necesario crear la ruta donde estos no coincidan.



### Descripción: Lexer.java

##### Una vez ejecutado el programa **generadora.java** automáticamente se generará el archivo: **Lexer.java**.

Text

Description automatically generated

## LexerCup.flex

##### Este apartado lo creamos describiendo todas las palabras reservadas y los operadores con el fin de describirle a nuestro programa las simbologías y el tipo al que corresponden. Como se muestra en la vista siguiente:

|  |  |
| --- | --- |
| **LexerCup.flex** | **LexerCup.java** |
| Text  Description automatically generated | Text  Description automatically generated with low confidence |

Éste tendrá efecto una vez que sea ejecutado nuestro archivo “generadora”.

En la terminal, el código al que tiene acceso con **LexerCup.java** se encuentra dentro de nuestro botón analizador Léxico.

|  |
| --- |
| Graphical user interface, text  Description automatically generated |

## Syntax.cup

|  |  |
| --- | --- |
| **Syntax.cup** | **Syntax. java** |
|  |  |

##### Como se observa; ingresamos nuestras palabras reservadas y operadores que serán utilizados a través de nuestro analizador Sintáctico, y será ejecutado con nuestra clase “**generadora.java”;** esto para generarnuestro archivo “**Syntax.java**”, donde se registran nuestras palabras asignadas que darán inicio a las sentencias establecidas, ejemplo:

|  |  |
| --- | --- |
| INICIO ::=  **SENTENCIAint**  ; | **SENTENCIAint ::=**  entera parentesisA Identificador asignaA Numero parentesisB puntoYcoma  **;** |
| Y esto quiere decir:  que nuestra SENTENCIAint equivale a =  **int ( i = 3 ) ;** | **Y lo anterior es lo mismo a:** |
| **SENTENCIA int ::=**  **int ( i = 3 ) ;**  **;** |

Graphical user interface, application

Description automatically generated

### Botón Analizador Sintáctico:

Éste se encuentra dentro en la programación del botón “Analizar”, de nuestra terminal.

|  |  |
| --- | --- |
|  | |
| **Symbol: botón Analizador Sintáctico** | | **Symbol: Syntax.java** |
|  | |  |

##### Y es de donde se accede a nuestro Syntax.java generado por nuestra “**generadora.java**” y analiza si el texto ingresado a nuestra cosola corresponde a lo establecido en nuestro **Sytax.cup.**

## Diseño de interfaz gráfica

##### Se generó una interfaz gráfica con nombre: “frmAnalizardor.java”. Y tiene las siguientes características:

|  |  |
| --- | --- |
| Diagrama de dependencias  Graphical user interface, text, application  Description automatically generated | **JPanel1** – contiene dentro dos paneles y una etiqueta (jLabel1) con el nombre Compilador Léxico.  **JPanel2**- resalta con un marco el Área de Texto.  **jPanel3** – contiene dentro de si el campo de texto (txtEntrada), botón Analizar (btnAnalizar), botón Recet (jButton1), y una etiqueta con la instrucción “ingrese el texto que desea analizar”(jLabel2). |
| **Diseño de la interfaz gráfica:** | |

### Color de paneles y botones:

Foreground (color principal): (51,51,51)

A screenshot of a computer

Description automatically generated with medium confidence

### Áreas de texto:

##### Background (color del fondo): (204, 204, 204)

##### Estilo de letra: Thoman, 18

A screenshot of a computer

Description automatically generated with medium confidence

### Texto de etiquetas:

Foreground (color): (204, 204, 204)

Estilo: Sylfaen, 18

A screenshot of a computer

Description automatically generated with medium confidence

## Función de los botones:

Botón Analizar: Está compuesto por dos try-catch:

|  |  |
| --- | --- |
| PrintWriter escribir;  try {  escribir = new PrintWriter(**archivo**);  escribir.print(txtEntrada.getText());  escribir.close();  } catch (FileNotFoundException ex) {  Logger.getLogger(frmAnalizador.class.getName()).log(Level.SEVERE, null, ex);  } | El primer try-catch (PrintWriter) tiene como función crear un archivo de texto dentro del proyecto de nombre “archivo”. El contenido del texto dependerá de lo que el usuario ingrese dentro de él. |
| Reader lector;  try {  lector = new BufferedReader(new FileReader("archivo.txt"));  Lexer lexer = new Lexer(lector);  String resultado = "";  while (true) {  Tokens tokens = lexer.yylex();  if (tokens == null) {  resultado += "FIN";  txtSalida.setText(resultado); //System.out.println(resultado); (proyectar en consola de NetBeans)  return;  }  switch (tokens) {  case ERROR:  resultado += "Error! Símbolo no definido\n";  break;  case Identificador: case Numero: case Reservadas:  resultado += lexer.lexeme + ": Es un " + tokens + "\n";  break;  default:  resultado += "Token: " + tokens + "\n";  break;  }  }  } catch (FileNotFoundException ex) {  Logger.getLogger(frmAnalizador.class.getName()).log(Level.SEVERE, null, ex);  } catch (IOException ex) {  Logger.getLogger(frmAnalizador.class.getName()).log(Level.SEVERE, null, ex);  } | Éste segundo try-catch (Reader lector), tiene como función reconocer cada uno de los elementos introducidos previamente por el usuario.  Estos son introducidos por la función, se almacenaran de un archivo de texto con nombre: “archivo.txt”, y éste leerá su contenido y posteriormente serán impresos en txt.Salida. Error Léxico: Al no poder reconocer alguno de los elementos de texto, éste arrojará una alerta con el enunciado: “Error! Símbolo no definido”.  Sin embargo, cada uno de los elementos identificados han sido definidos previamente en los archivos “Lexer.flex” y “Tokens.java”, de los elementos introducidos por el usuario serán identificados como “**Tokens**”. |

### Ejemplo de funcionamiento de interfaz gráfica:

|  |  |
| --- | --- |
|  | Como se observa de manera un poco más ilustrado, se introducen elementos dentro de la interfaz gráfica, y una vez presionado el botón de “Analizar”, el programa genera un archivo nuevo con nombre “archivo”, dentro de la carpeta del programa; que visualizando el contenido será el mismo al texto introducido previamente. En seguida éste contenido también será impreso dentro del campo de texto (txtSalida), y al finalizar se describe como FIN. |
| Graphical user interface, application  Description automatically generated  Y este a su vez se genera el archivo de texto dentro de la carpeta del proyecto. | |

# Manual de Usuario

## Ejecutar programa:

##### Usando el IDE java NetBeans 8.2, una vez exportado el proyecto LenguajesAII\_ALexicoII, ejecutar la interfaz gráfica: frmAnalizador.java con el segundo botón del ratón, donde si abrirá la interfaz gráfica.

|  |  |
| --- | --- |
| **Cómo ejecutar programa:**  Graphical user interface, application, Teams  Description automatically generated | **Interfaz Gráfica** |

## Uso de Programa:

Tome en cuenta los siguientes símbolos y palabras que el programa puede identificar:

### Nombres asignados de los operadores, palabras reservadas y delimitadores:

|  |  |
| --- | --- |
| * if → entonces * else → ademas * while → mientrasque * for → para * int → entera * char → caracter * float → flotante * double → doble * boolean → boleana * "=" → asignaA * "+" → Suma * "-" → Resta * "\*" → Multiplicacion * "/" → Division * "==" → igualA * "<" → menorQue * ">" → mayorQue * ">=" → mayorOigualA * "<=" → menorOigualA | * "<<" → desplazaIzquierda * ">>" → desplazaDerecha * "!=" → diferenteA * "&" → tambien * "&&" → comentario * "%" → modula * "|" → obien * "++" → incremento * "--" → decremento * " **''** " → comilladoSimple * " **""** " → comilladoDobleA * " : " → dosPuntos * "^" → continuaY * "." → punto * "," → coma * ";" → puntoYcoma * "(" → parentesisA * ")" → parentesisB * "[" → corcheteA * "]" → corcheteB * "{" → llaveA * "}" → llaveB |

Cualquier símbolo que no se enuentre dentro de ésta lista, el programa no podrá identificarlo, y emitirá un mensaje de error de identificación:

Ejemplo: Símbolos no identificados (**$** , **@** ).

## Elementos de la interfaz:

Vista normal de la consola:

Graphical user interface, application

Description automatically generated

Descripción de los elementos de la consola:

Borrar cualquier símbolo introducido e impreso en el campo de salida.

Graphical user interface, application

Description automatically generated

Abrir un archivo con contenido de texto.

Ingresar texto:

Analizar la sintaxis del texto introducido.

Analizar la estructura léxica por el programa del texto introducido.

Analizar los tokens y elementos reconocibles por el programa del texto introducido.

**Nota:**

##### Cualquier elemento analizado en ésta consola será generada en un archivo de texto dentro de la carpeta del proyecto con nombre “archivo.txt”, el cual se irá actualizando cada que se presione el botón “Analizar.

* Para ingresar nuevos elementos al sistema como palabras y/o símbolos, favor de revisar el manual técnico.