# ITCR – IC5701.COMPILADORES E INTÉRPRETES


## TAREA #1 - HERRAMIENTA: JFlex

**Estudiantes:**

William Alfaro Quirós – Carné: 2022437996

Jesus Cordero Díaz - Carné: 2020081049

Pablo Venegas Sánchez - Carné: 2018083497

**Profesor:**

Emmanuel Ramírez S.

**II Semestre 2024**



# JFlex

**Descripción**

**¿Para qué sirve?** JFlex es una herramienta crea analizadores léxicos en Java, similar a Lex para C/C++. Además permite describir patrones de tokens y los genera como código Java.

**¿Tiene algún tipo de licencia?** Sí, está bajo la licencia BSD, la cual es de código abierto y gratuita.

**¿Quien o quienes la desarrollaron?** Fue desarrollada y mantenida por Gerwin Klein, pero cuenta con algunas contribuciones de la comunidad.

**¿Aun cuenta con Mantenimiento?** Sí, en la actualidad aún cuenta con mantenimiento, además cuenta con una versión estable actual es JFlex 1.9.1 lanzada el 11 de marzo del 2023.



# JFlex

**Enlaces de Descarga**

Página oficial: JFlex



# JFlex

**Lenguajes**

JFlex es una herramienta orientada principalmente para trabajar con Java. Su propósito es generar analizadores léxicos en código Java a partir de una especificación escrita en un archivo .jflex.

1. **Lenguaje Principal:** JFlex está diseñado específicamente para generar código Java, lo que lo convierte en una herramienta esencial para proyectos que utilizan este lenguaje. El archivo de salida es un archivo .java que contiene el código de un analizador léxico.
2. **Otros lenguajes:** Aunque JFlex no genera código directamente para otros lenguajes, se puede usar indirectamente en proyectos de multilenguaje o con máquinas virtuales compatibles con Java, como Scala o Kotlin. La compatibilidad se da siempre que la fase de análisis léxico se desarrolle en Java, pero no genera analizadores directamente en lenguajes como C o Python, como lo harían otras herramientas específicas.



# JFlex

## Etapas de generación de procesadores de lenguaje

JFlex se sitúa específicamente en la etapa de **análisis léxico** del proceso de construcción de un compilador o intérprete. El análisis léxico es la primera fase en el ciclo de compilación, donde el flujo de entrada de caracteres (por ejemplo, el código fuente de un programa) se agrupa en **tokens** que luego se pasan al analizador sintáctico.

- **Etapa en la que trabaja**: Análisis Léxico.
- **Algoritmo utilizado**: JFlex utiliza **Autómatas Finitos Deterministas (DFA)**. Primero convierte las expresiones regulares definidas por el usuario en un autómata finito no determinista (NFA), y luego lo optimiza para convertirlo en un DFA, lo que permite reconocer tokens eficientemente.
- JFlex permite definir reglas utilizando **expresiones regulares**. Estas reglas especifican cómo se deben reconocer las secuencias de caracteres (tokens), como identificadores, números, operadores, etc. Cada token puede estar asociado con una acción (generalmente generar un objeto que represente ese token).
- **Producción de tokens**: Estos tokens luego son enviados a la siguiente etapa (normalmente un analizador sintáctico) para continuar con el proceso de compilación.



# JFlex

## Cómo se utiliza?

JFlex se usa a través de archivos de especificación. En estos archivos se definen las reglas de reconocimiento de tokens y las acciones que se van tomar cuando se encuentren dichos tokens. este proceso se podría resumir en los siguientes pasos:

**Creación del archivo de especificación**: Un archivo de especificación **.jflex**, en este se definen las reglas léxicas utilizando expresiones regulares.

**Compilación del archivo de especificación**: Usando JFlex, el archivo **.jflex** se "compila" para generar una clase Java que contiene el analizador léxico.

**Integración en un proyecto**: El archivo generado se incluye en el proyecto y se usa junto con otras herramientas para procesar el código fuente o el texto de entrada.

**Ejecución**: Cuando se ejecuta el programa, el analizador léxico leerá el flujo de caracteres (por ejemplo, desde un archivo de código fuente) y lo convertirá en una secuencia de tokens.



# JFlex

**Compatibilidad con otras herramientas**

**CUP**: JFlex es usado frecuentemente con CUP, el cual es un generador de parsers que maneja el análisis sintáctico. Mientras que JFlex genera los tokens, CUP se encarga de analizar su estructura gramatical.

**IDE como Eclipse o IntelliJ**: JFlex se integra en estos entornos para facilitar su uso dentro de proyectos Java, automatizando la generación de analizadores léxicos como parte del proceso de compilación.

# JFlex

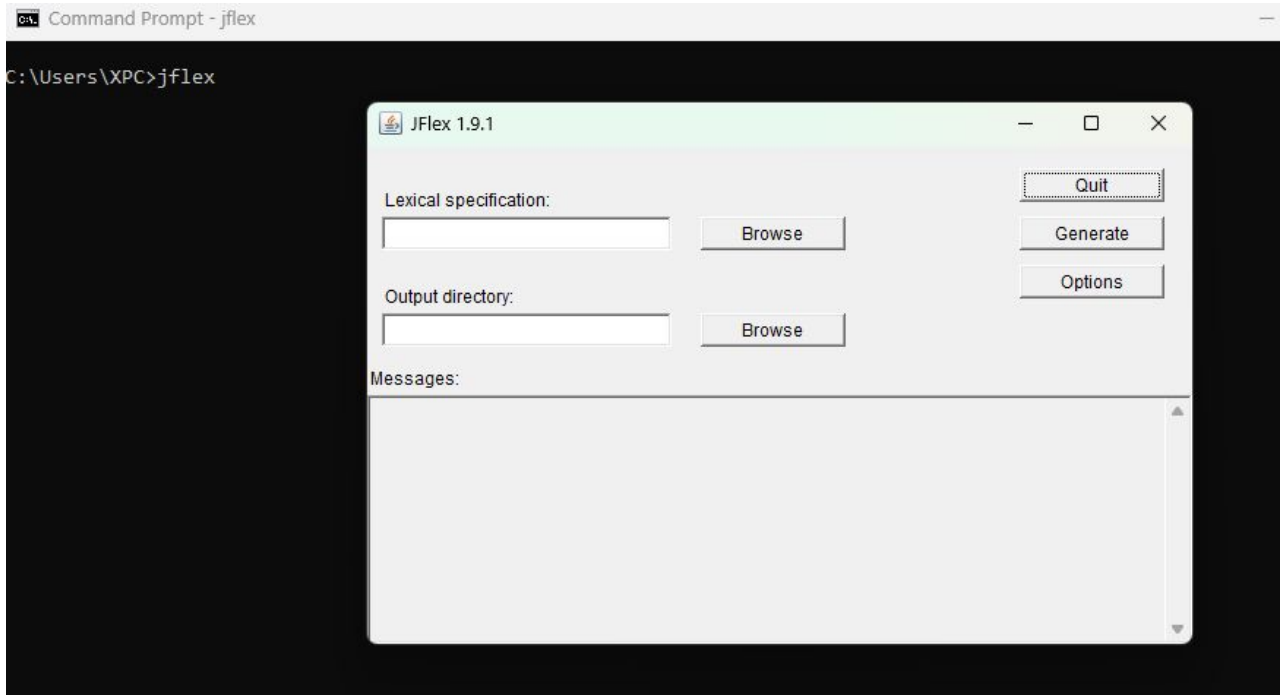
**Ejemplo de uso:**

Creo el archivo .flex donde se escribe lo que se debe retornar con cada expresión.

```
%%
%class Lexer
%type String
L=[a-zA-Z_]+
D=[0-9]+
espacio=[ ,\t,\r,\n]+
%{
public String lexeme;
%}
%%
int {lexeme=yytext(); return "Reservadas: " + lexeme;}
if {lexeme=yytext(); return "Reservadas: " + lexeme;}
else {lexeme=yytext(); return "Reservadas: " + lexeme;}
while {lexeme=yytext(); return "Reservadas: " + lexeme;}
{espacio} {/*Ignore*/}
"//".* {/*Ignore*/}
"=" {lexeme=yytext(); return "Igual: " + lexeme;}
"+" {lexeme=yytext(); return "Suma: " + lexeme;}
"-" {lexeme=yytext(); return "Resta: " + lexeme;}
"*" {lexeme=yytext(); return "Multiplicacion: " + lexeme;}
"/" {lexeme=yytext(); return "Division: " + lexeme;}
"(" {lexeme=yytext(); return "ParIzq: " + lexeme;}
")" {lexeme=yytext(); return "ParDer: " + lexeme;}
"{" {lexeme=yytext(); return "LlaveIzq: " + lexeme;}
"}" {lexeme=yytext(); return "LlaveDer: " + lexeme;}
";" {lexeme=yytext(); return "PuntoYComa: " + lexeme;}
{L} ({L} | {D})* {lexeme=yytext(); return "Identificador: " + lexeme;}
("(-"{D}+")") | {D}+ {lexeme=yytext(); return "Numero: " + lexeme;}
. {lexeme=yytext(); return "Error: " + lexeme;}
```

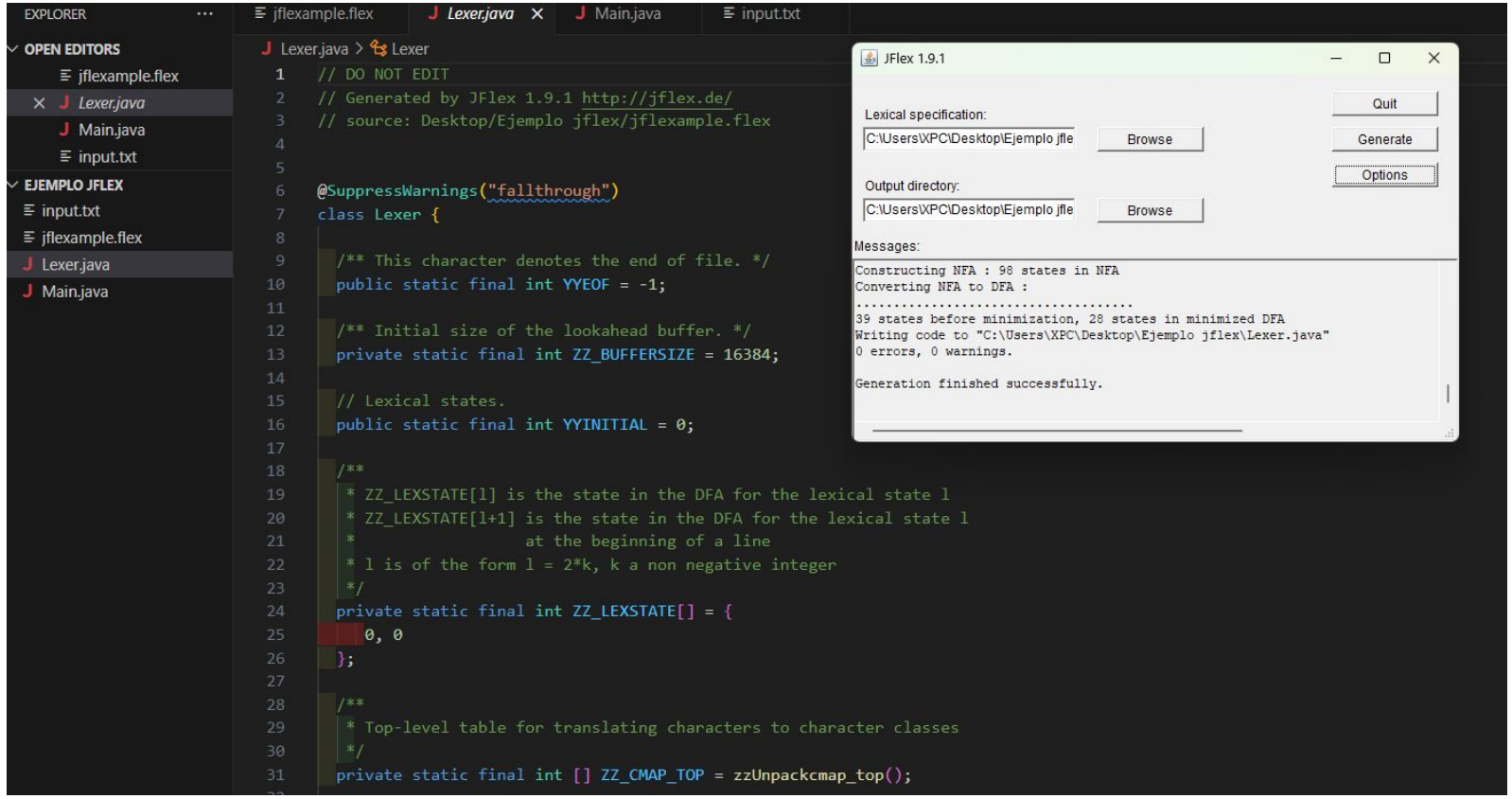
# JFlex

**Ejemplo de uso:**
En la terminal se escribe el comando “jflex” como se muestra a continuación lo que abrirá un cuadro de diálogo para ingresar el directorio donde se encuentra el archivo .flex e ingresar dónde quiero que se genere la clase .java.



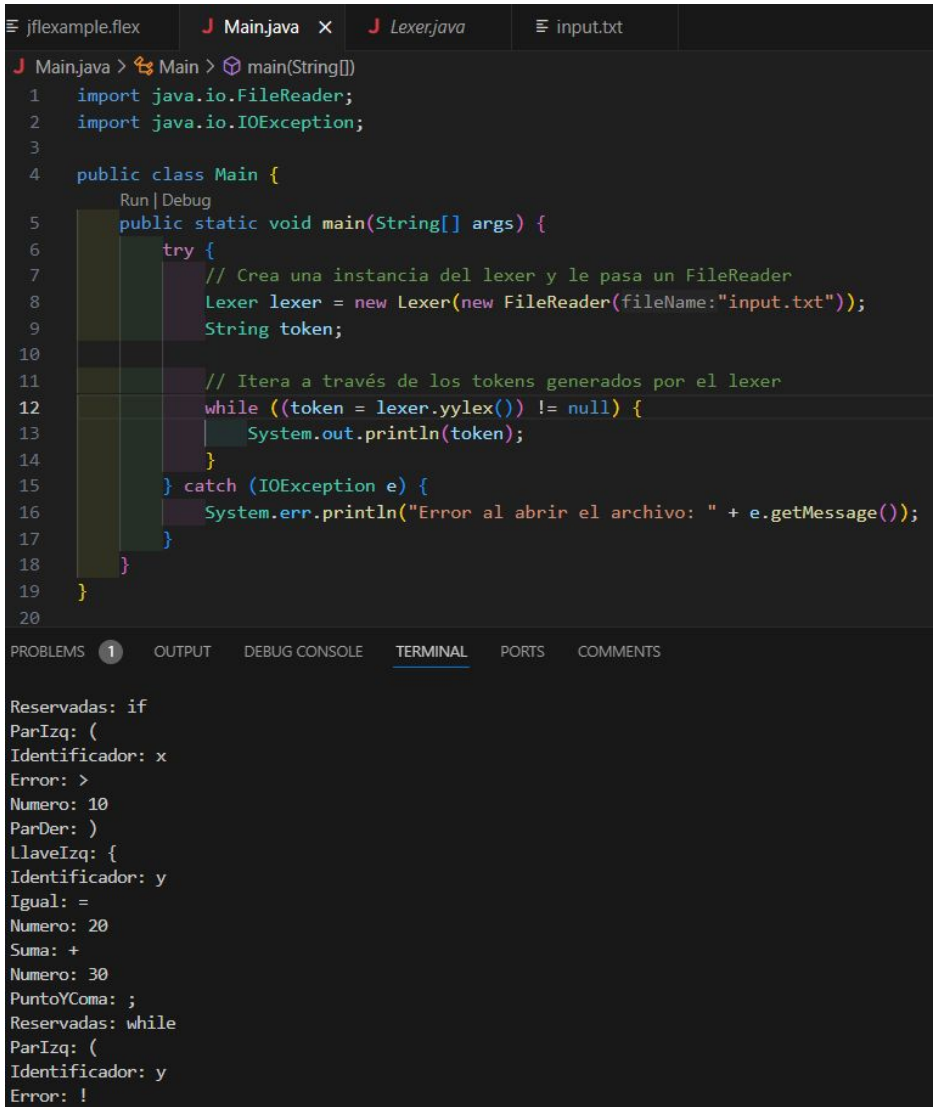
# JFlex

**Ejemplo de uso:**
Se presiona el botón Generate lo que va a crear el archivo Lexer.java.



# JFlex

**Ejemplo de uso:**
Por último se prueba el Lexer generado y vemos que funciona.





# JFlex

- **Espacio para consultas**



# JFlex

- **Bibliografía**

Documentación oficial: JFlex

Repositorio en GitHub:JFlex github

Tutorial Analizador léxico con Java: Tutorial