Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Organización de Lenguajes y Compiladores 1
Primer Semestre 2020
Ing. Mario Bautista, Ing. Manuel Castillo, Ing. Kevin Lajpop
Aux: Luis Mazariegos, Huriel Gómez, Fernando Paz, Juan Landaverde

Proyecto 2

Objetivos

General:

1. El estudiante aplique los conocimientos obtenidos a lo largo del curso y que implemente una solución para generar un detector de copias.

Específico:

1. El estudiante pueda familiarizarse con las herramientas para generar un analizador lexico y sintactico.
2. El estudiante aprenda a desarrollar un detector de copias en un lenguaje orientado a el servicio web.

Descripción

Como estudiante de la carrera de ciencias y sistemas, se le solicita que realice un analizador de copias que reconozca la sintaxis del lenguaje java, debido a que es el primer lenguaje de programación que se ve en la carrera usted tiene el conocimiento del funcionamiento del mismo.

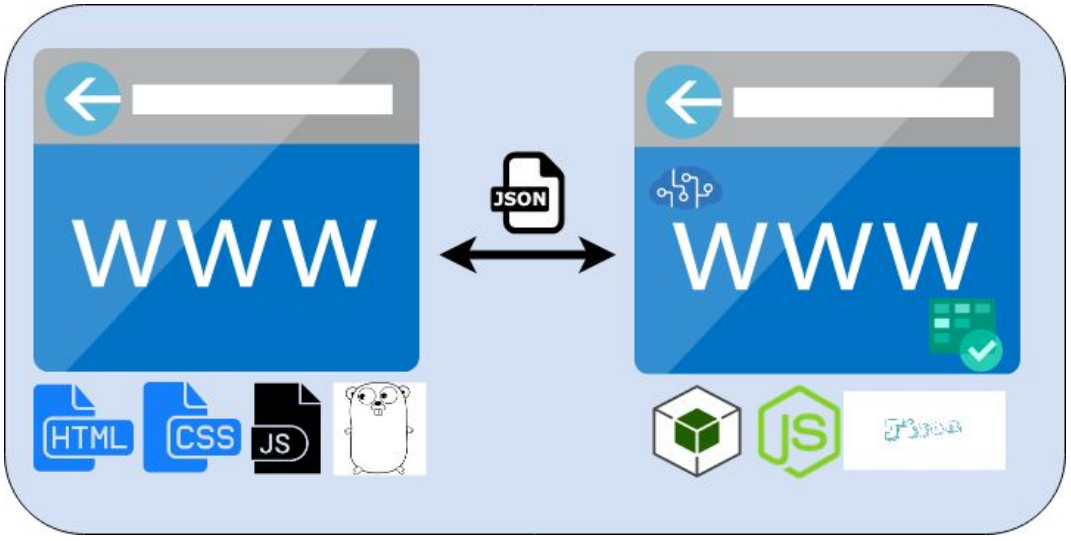
La solución a desarrollar debe ser de tipo cliente-servidor de manera que esta se divide en las siguientes 2 partes:

Frontend (Vista Cliente)
Este deberá ser desarrollado mediante el uso de javascript, css y html para la construcción de una pagina que sea amigable y fácil de utilizar, dicha pagina deberá ser lanzada mediante un servidor http, **para el cual deberán de utilizar el lenguaje Go únicamente.**

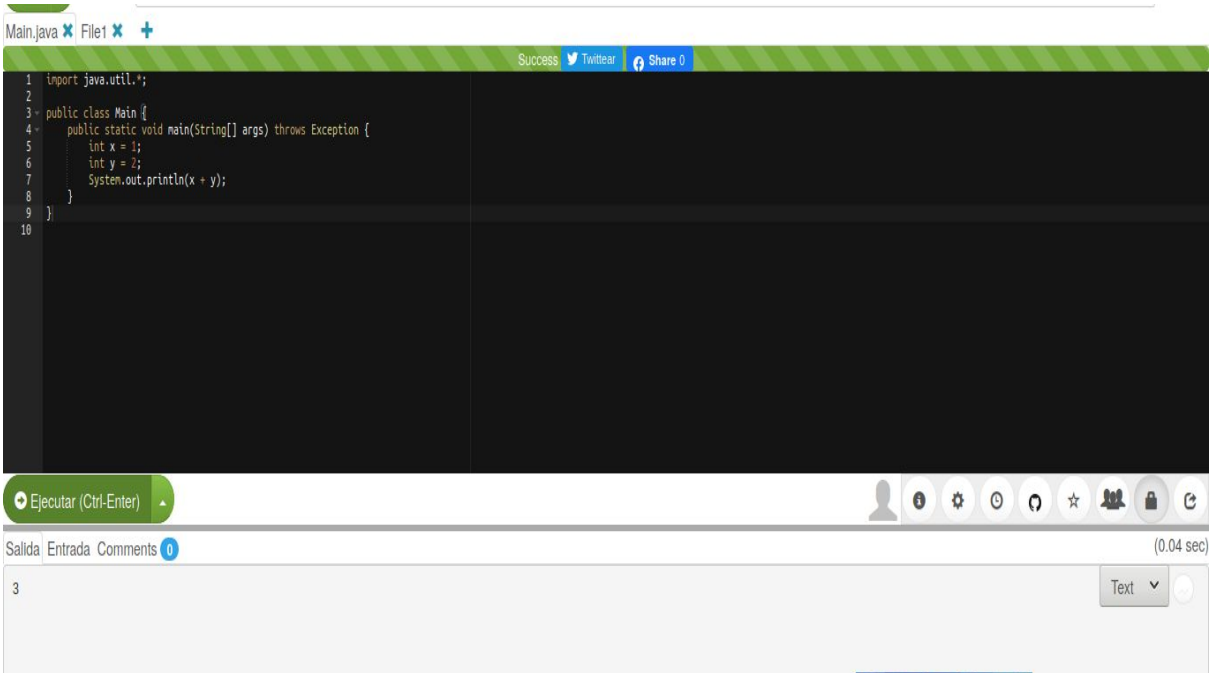
Backend (Servidor Analizador)
Este deberá ser desarrollado utilizando nodejs con javascript para levantar un servidor que recibirá peticiones REST y realizara el analisis lexico, sintactico y la detección de copias entre 2 proyectos desarrollados en java de manera que se obtengan reportes sobre el grado de similitud de los mismos.

Para este caso es permitido utilizar una herramienta para el analisis lexico y sintactico, **para este caso nos referimos al uso de jison.**

Flujo de la aplicación
La aplicación del lado del cliente será una vista web la cual cargará los archivos de entrada, los cuales posteriormente enviará al servidor REST el cual realizará el análisis de los mismos y la detección de posibles similitudes.



Interfaz Sugerida



Referencia: https://paiza.io/es/projects/new

La Interfaz debe contar con:

Un **menú** en el cual se tendrá la opción de: guardar, guardar como, abrir, nuevo, descargar reportes errores analisis lexico y sintactico.

Un **editor de texto** el cual será donde se carguen los archivos de entrada, este debe de contar con un listado de pestañas las cuales deberán crearse de manera libre con un nuevo documento o al abrir un documento.

Un **botón** para enviar el archivo o archivos al servidor y que se genere el análisis de copias.

Un **área de salida** con pestañas en las cuales se muestre como salida si efectivamente se trata de una copia, el reporte de errores léxico, el reporte de errores sintáctico.

Un **área de descarga de reportes** los reportes a realizar se describen adelante.

Definición del Lenguaje

Generalidades

Sensibilidad del Lenguaje

El lenguaje java será case sensitive, es decir que será sensible a las mayúsculas y minúsculas, esto aplica tanto para palabras reservadas del lenguaje como para identificadores. Por ejemplo: *miVariable* no es igual que *mivariable*.

Comentarios

Existirán 2 tipos de comentarios:

- Los comentarios de una línea que serán delimitados al inicio con los símbolos "//" y al final con un carácter de salto de línea.
- Los comentarios con múltiples líneas que empezarán con los símbolos "/*" y terminarán con los símbolos "*/".

Ejemplo:

```
//Comentario de una sola linea

/*   Este es un comentario
   multilínea
*/
```

Identificadores

Un identificador será utilizado para dar un nombre a variables, método, etc.
Un identificador es una secuencia de caracteres alfabéticos [A-Z a-z] incluyendo el guión bajo [_] o dígitos [0-9] que comienzan con un carácter alfabético o guión bajo.

Ejemplo de identificadores válidos:

```
id_valido
_otro_valido
yEsteTambien
```

Ejemplo de identificadores no válidos:

```
no-valido
2_invalido
```

Tipos de Datos

Los tipos de datos que serán soportados por el lenguaje son:

Tipo	Descripción	Ejemplo
int	Este tipo de dato acepta únicamente números enteros.	1, 12, 1500, etc.
double	Este tipo de dato acepta valores enteros con decimales	0.5445, 0.25, 50.3, etc.
boolean	Acepta valores lógicos de verdadero o falso.	false, true
char	Solo acepta un carácter y viene delimitado por comillas simples.	‘a’, ‘9’, ‘#’, ‘D’, ‘!’, etc.
String	Acepta cadenas de caracteres.	“hola mundo”, “!! Este es un String??#:”, etc.

Secuencias de Escape

Las secuencias de escape se utilizan para definir ciertos caracteres especiales dentro de cadenas de texto. Las secuencias de escape disponibles son las siguientes:

Secuencia	Descripción
\n	Salto de línea
\t	Tabulador
\r	Retorno de carro
\\	Barra invertida
\”	Comilla doble

Sintaxis del Lenguaje

A continuación se define la sintaxis para el lenguaje:

Operaciones

Operaciones Aritméticas

- **Suma:** El operador de la suma es el signo más (+).
- **Resta:** El operador de la resta es el signo menos (-).
- **Multiplicación:** El operador de la multiplicación es el asterisco (*).
- **División:** El operador de la división es la diagonal (/).
- **Potencia:** El operador de la potencia es el acento circunflejo (^).
- **Módulo:** El operador del módulo es el signo de porcentaje (%).
- **Menos Unario, incremento y decremento**
  - **Menos Unario:** El operador de negación unaria precede su operando y lo niega (*-1)
  - **Incrementó:** El operador de incremento son dos signos más (++).
  - **Decremento:** El operador de decremento son dos signos menos (--).

Ejemplo
-56 = -56 85-- = 84 5++ = 6
-0.25 = -0.25 1.5 -- = 0.5 5.2++ = 6.2

Operaciones Relacionales

- **Igualdad (==)**
- **Distinto (!=)**
- **Mayor que (>)**
- **Mayor o igual que (>=)**
- **Menor que (<)**
- **Menor o igual que (<=)**

Operaciones Lógicas

- **And (&&)**
- **Or(||)**
- **Not(!)**

Definición de clases

La sintaxis para la declaración de una clase es la siguiente, en donde debe establecerse el nombre de la clase:

```
//Esta es la definicion de una clase
class miClase{        /*   Cuerpo de la clase     */          }
```

Imports

Cada clase tendrá la posibilidad de poder importar atributos, métodos y funciones de otra clase indicando el nombre de la clase de la cual se desea obtener toda su estructura.

```
import ClaseA;
import ClaseB;
class ClaseC{
        /*
           Cuerpo de la clase C
        */
}
```

Declaración de Variables

La declaración puede hacerse desde cualquier parte del código ingresado, pudiendo declararlas dentro de ciclos, métodos, funciones o fuera de estos siempre dentro de una clase. La forma normal en que se declaran las variables es la siguiente:

Sintaxis:

```
TIPO LISTA_ID ;
TIPO LISTA_ID = EXPRESION;
```

Ejemplo:

```
int n1 = 4;
double n2,n3,n4;
boolean valor;
String miCadena = “hola\n”+”mundo..!!!”;
```

Asignación de Variables

Sintaxis:

```
IDENTIFICADOR = EXPRESION;
```

Ejemplo:

```
valor = true;
```

```
miCadena = “¿Sale compi1? \n” + valor;
```

Bloque de Sentencias

Es un conjunto de sentencias delimitado por llaves “{ }”, cuando se haga referencia a esto querrá decir que se está definiendo un ámbito local con todo y sus posibles instrucciones.

```
{
        //Sentencias
}
```

Sentencias de Control de Flujo

Sentencia If-Else

Sintaxis:

```
if (CONDICION)  <BLOQUE_SENTENCIAS>
else if (CONDICION)  <BLOQUE_SENTENCIAS>
…
else if (CONDICION)  <BLOQUE_SENTENCIAS>
else <BLOQUE_SENTENCIAS>
```

Ejemplo:

```
If( 3 < 4) {   /* Sentencias */   }
else if (2 < 5) {   /* Sentencias */  }
else {   /* Sentencias */  }

If(true) {  /* Sentencias */ }

If(false){  /* Sentencias */ }
else {  /* Sentencias */ }

If(false){  /* Sentencias */  }
else if(true) { /* Sentencias */ }
```

Sentencia Switch

Sintaxis:

```
switch (expresión) {
   case expr1:
        // Sentencias asociadas a expr1
   case expr2:
        // Sentencias asociadas a expr2
```

```
    ...
  case exprN:
      // Sentencias asociadas a exprN
  default:
      // Sentencias asociadas a expr1
}
```

Ejemplo:

```
switch (3*54) {
    case 3:   // Sentencias
        break;
    case 5:
            // Sentencias
    case 7:
            // Sentencias
        break;
    default:
            // Sentencias
        break;
}
```

Sentencias de Repetición

Sentencia While

Sintaxis:

```
while ( CONDICION ) <BLOQUE_SENTENCIAS>
```

Ejemplo:

```
while(1>a){
        System.out.println("0777-Compiladores 1");
}
```

Sentencia Do While

Sintaxis:

```
do <BLOQUE_SENTENCIAS> while ( CONDICION )
```

Ejemplo:

```
do{
        System.out.println("0777-Compiladores 1");
}while(1>a);
```

Sentencia For

Sintaxis:

```
for([DEC| ASIG]; CONDICION; [ INC | DEC ]) <BLOQUE_SENTENCIAS>
```

Ejemplo:

```
for(int i=0; i<10; i++){  /* Sentencias */  }

int k;
for(k=15; k>5; k--){  /* Sentencias */  }
```

Sentencias de Transferencia

Break

Ejemplo:

```
for(int i=0; i<10; i++){
    if(i == 5){ break; }
}
```

Continue

Ejemplo:

```
while( i<10){  continue; }
```

Return

Ejemplo:

```
void metodo1(int a) { //Metodo declarado
        if(a==1){ return; }
 }

//Funcion que resuelve la suma de dos numeros
double suma(double n1, double n2){  return n1 + n2; }
```

Funciones

Declaración de Funciones

La declaración de funciones y métodos se realizará de la siguiente manera:

Sintaxis:

```
[ void | TIPO ]  IDENTIFICADOR (LIST_PARAM ) <BLOQUE_SENTENCIAS>
```

Ejemplo:

```
void metodo1() { //Metodo declarado
        System.out.println("0777-Compiladores 1");
 }
```

```
//Funcion que retorna la multiplicacion de dos numeros
double potencia(double n1, double n2){
        double resultado = n1^ n2;
        return resultado;
}
```

Llamada a Funciones

Las funciones y métodos sólo pueden ser llamados desde dentro de algún procedimiento o función definida dentro de la clase.

Sintaxis:

```
IDENTIFICADOR ( LIST_EXPRESIONES );
```

Ejemplo:

```
//Llamada a un metodo
metodo1();
//Llamada a una funcion
double pot = potencia(5,3);
```

Método Principal

```
void main() {
        System.out.println(“7 elevado al cubo es: ”+potencia(7,3));
 }
```

Print

Ejemplo:

```
System.out.print(“7 elevado al cubo es: “);
System.out.println(potencia(7,3));
System.out.println(“:)”);

/*
Salida esperada:
7 elevado al cubo es: 343
:)
*/
```

Reportes

Reporte de Errores

La aplicación debe estar en la capacidad de **recuperarse de errores léxicos y sintácticos**, para poder corregir de manera eficiente los archivos.

Para la recuperación de errores sintácticos se deberá utilizar el método de recuperación modo pánico.

Reporte de errores léxicos, sintácticos:

No.	Tipo error	Línea	Columna	Descripción
1	Léxico	12	6	El carácter '¬' no pertenece al lenguaje.
2	Sintáctico	45	10	Se esperaba… (se encontró id...)...

- El reporte debe ser en formato HTML y de existir errores deberá ser visualizado al momento de terminar el análisis.

Reporte de Clases Copia

Este reporte deberá mostrar un listado de todas las clases que se consideren como copia, para considerar una clase como clase copia se deberá de verificar que tenga el mismo nombre y los mismos métodos y/o funciones. Este reporte debe mostrar el nombre de la clase, cantidad de métodos y/o funciones que contiene.

Reporte de Funciones Copia

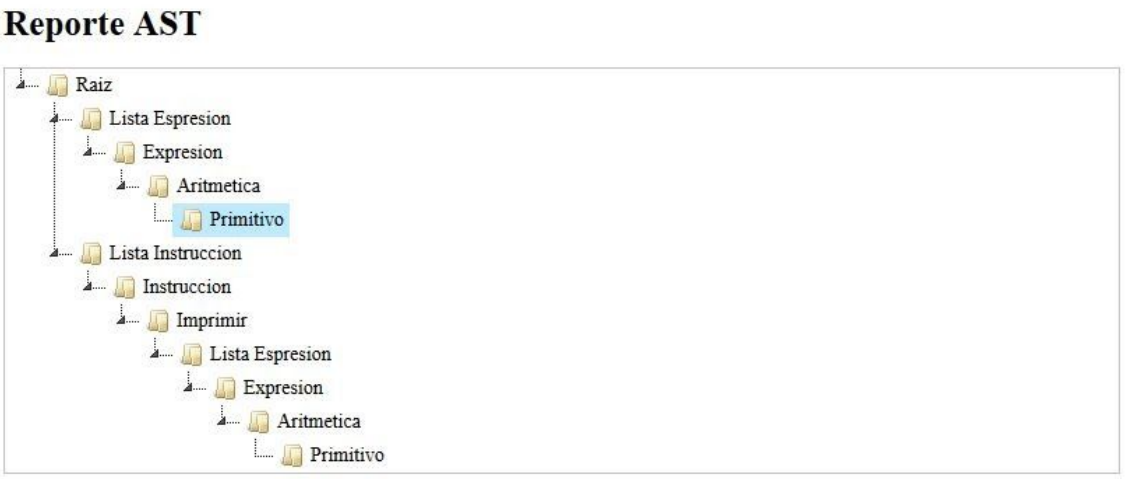
Este reporte deberá de retornar un listado de todas las funciones que se encontraron como copia, para definir una función copia se debe tomar en cuenta que la función tenga la misma cantidad de parámetros y que sean del mismo tipo y en el mismo orden, además de encontrarse dentro de una clase con el mismo nombre y el tipo de retorno. Este reporte mostrará el tipo de retorno del método y/o función, nombre del mismo, listado de sus parámetros con tipo y nombre, nombre de la clase al que pertenece.

Reporte de Variables Copia

Este reporte deberá mostrar un listado de las variables que se consideran copia, para considerar una variable como copia deberá pertenecer al mismo métodos y/o función y a la misma clase, así como el mismo tipo. Este reporte mostrará el tipo de la variable, nombre, la función y/o método al que pertenece, la clase a la que pertenece.

Reporte AST

Este reporte mostrará el árbol de análisis sintáctico que se produjo al analizar el archivo de entrada.Este será un árbol dinámico de manera que al hacer clic sobre alguno de los nodos este despliegue el subárbol que contiene. El Estudiante deberá mostrar los nodos que considere necesarios y se realizarán preguntas al momento de la calificación para que explique su funcionamiento.



Entregables

---

1. Código fuente de la solución
2. Manual Técnico y de Usuario
3. Link del repositorio de versiones

Ciclo de ejecución básico

Para que se pueda realizar la calificación de este proyecto es necesario que se cuente con un **ciclo de ejecución básica** la cual incluye:

1. Página web cliente utilizando (javascript, html, css y go), para poder cargar archivos y ver los resultados esperados.
2. Servidor backend para realizar el análisis del archivo de entrada y retorno de errores obtenidos y salida esperada.
3. Reporte AST del análisis sintáctico obtenido.

Consideraciones

---

1. Trabajar de manera individual
2. Es permitido el uso de herramientas para el análisis (JISON).
3. Lenguajes de programación: Javascript (Typescript), go.
4. El sistema operativo y el IDE son libres, debe utilizarse únicamente nodejs para el backend y go para el frontend.
5. Cualquier copia total o parcial será reportada.
6. Los archivos de entrada serán proporcionados el día de la calificación.
7. No habrá prórroga y entregas fuera de horario tienen una nota de CERO.
8. Fecha de entrega **viernes 14 de mayo a las 23:59.**