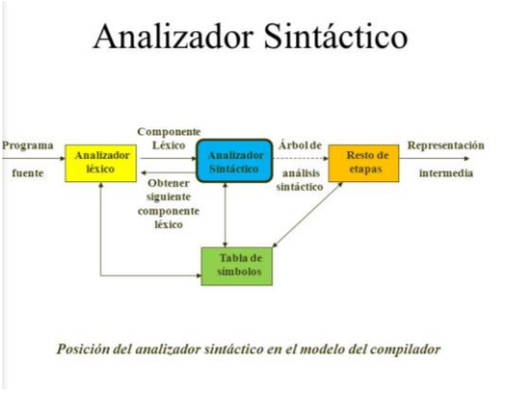
Analizador Sintáctico

El analizador sintáctico o parser es otra de las partes más importantes de todo compilador y su tarea es la de verificar que los tokens que recibe el analizador léxico (scanner) estén debidamente combinados de tal manera que cumplan con las reglas sintácticas del lenguaje fuente, en este caso, que cubra la sintaxis usada por el lenguaje de programación.



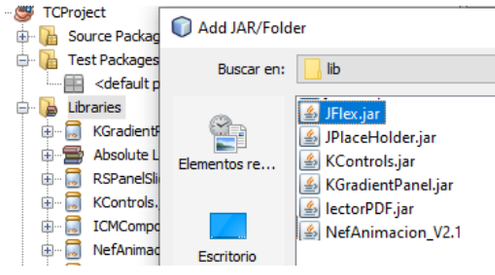
Posición del analizador sintáctico en el modelo del compilador

En esta parte del compilador, es muy importante el manejo de errores, se debería tener en cuenta una política de trabajo cuando suceda que una instrucción ingresada no está sintácticamente bien formada. Usualmente se implementan tablas de errores para el manejo más ordenado de los mismos. A nivel de programación, y por el caso de tratarse de un simulador de compilador, se podría manejar mediante el manejo de Excepciones y lanzamientos (Throw), sin embargo, en el código que se presentará, se hace un corte del desarrollo del programa y se muestra el respectivo mensaje de error.

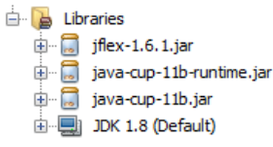
Como crear un Analizador Sintáctico

En este caso nos encontramos con el generador de analisis léxico para java, JFlex. Lo primero que hay que hacer es descargar el analizador.

una vez descargado en el computador, procedemos a importar la librería **jflex** en un nuevo proyecto de Netbeans.



A continuación añadiremos la librería JCup que luego nos servirá para hacer el análisis.

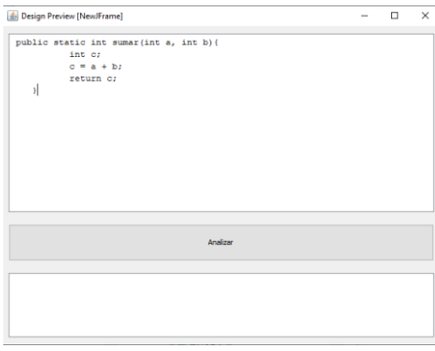


Luego procederemos a crear dos archivos con extensión .flex, llamados CupLexer.flex y Sintasis.flex en el cual dentro de este ultimo iremos creando nuestros tokens.

posterior nuestros Tokens serán generados por la librería de jflex

-En nuestro archivo CupLexer.cup definiremos nuestras propias reglas, el cual serán la guía para el análisis sintáctico.

A continuación, crearemos un Frame con dos JTextarea y un botón.



Luego presionamos doble click en el botón creado y dentro del evento click crearemos la conexión entre Java y el analizador sintáctico y dentro de un Try-Catch iniciaremos el proceso de análisis

```
private void analizadorSintactico(){
    String ST = txtCodigo.getText();
    Sintax sintactico = new Sintax(new LexerCup(new StringReader(ST)));

    try {
        sintactico.parse();
        txtResultado.setText("El Analisis Sintactico Termino con Exito");
        txtResultado.setForeground(new Color(25, 111, 61));


    } catch (Exception ex) {
        txtResultado.setText(sintactico.resultado);
        txtResultado.setForeground(Color.red);
    }
}
```

Procederemos a mejorar la interfaz y luego compilaremos nuestro programa.

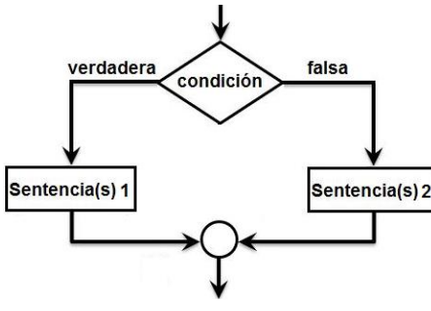
Estructura if Else

La estructura condicional if ... else es la que nos permite tomar ese tipo de decisiones. Traducida literalmente del inglés, se la podría llamar la estructura "si...si no", es decir, "si se cumple la condición, haz esto, y sino, haz esto otro".

Un ejemplo sencillo sería el siguiente (no se trata de un programa completo, sino tan sólo una porción de código):

```
if (condición) {
   sentencias_si_verdadero;
}else{
   sentencias_si_falso;
}
```

```
if (edad < 18){
   System.out.println("No puedes acceder.\n");
}else {
System.out.println("Bienvenido.\n");
}
```

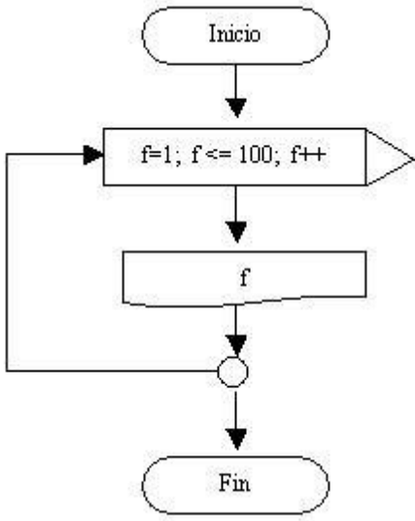


Estructura For

El bucle for es una variante del bucle while y, al igual que éste, puede iterar cero o más veces. Sin embargo, el bucle for sólo se suele usar cuando se conoce el número exacto de veces que tiene que iterar el bucle.

for (<expresión_1> ; <expresión_2> ; <expresión_3>)

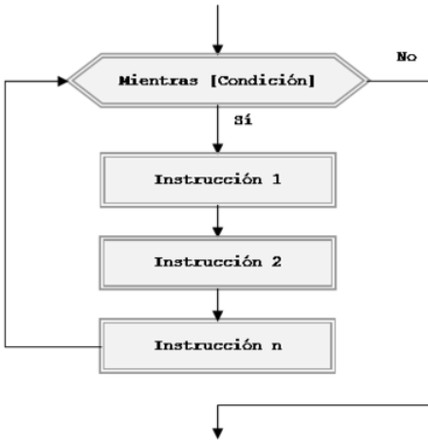
```
{
    <bloque_de_instrucciones>
}

int contador;
System.out.println("");

for ( contador=1 ; contador<=10 ; contador++ )
    System.out.println("# " + contador);
}
```



Estructura While

Estructura While

El bucle while presenta ciertas similitudes y ciertas diferencias con el bucle for. La repetición en este caso se produce no un número predeterminado de veces, sino mientras se cumpla una condición. Conceptualmente el esquema más habitual es el siguiente:



La sintaxis en general es: while (condición) {instrucciones a ejecutarse } donde condición es

una expresión que da un resultado true o false en base al cual el bucle se ejecuta o no.

```
int i = 0;
while (true) {
//Condición trivial: siempre cierta
    i++;
    System.out.println("#i: " + i);
    if (i==9) { break;}
}
```

En este código hemos hecho algo un poco extraño. Como condición a evaluar hemos puesto “true”. Esto significa que la condición es siempre verdadera, lo que en teoría daría lugar a un bucle infinito y a un bloqueo del ordenador. Sin embargo, utilizamos un contador auxiliar que inicializamos en cero y en cada repetición del bucle aumentamos en una unidad. A su vez, introducimos una condición dentro del bucle según la cual cuando el contador alcanza el valor 9 se ejecuta la instrucción break.

Sentencia Do While

La sentencia de iteración do-while es de tipo posprueba. Primero realiza las acciones luego pregunta. La sintaxis es la siguiente:

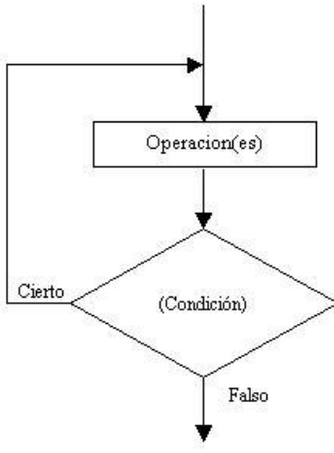
```
do sentencia while ( condición );
```

Observamos que es como un while pero al revés. Primeramente, se ejecuta la sentencia y luego evalúa la

condición. Si la expresión de la condición es verdadera vuelve a dar un ciclo. De lo contrario, termina. Esto nos garantiza que la sentencia se ejecute al menos una vez.

```
do
  System.out.println("Lo veras una vez");
while ( false );
```

Resulta útil para los casos en donde tendremos que realizar ciertas acciones antes de verificar una condición.



EstructuraSwitch-case

La sentencia switch se encarga de estructurar una selección múltiple. Al contrario del enunciado if-else que sólo podemos indicar dos alternativas, maneja un número finito de posibilidades. La estructura general del enunciado switch es la siguiente:

```
switch( expresión ) {
  case constante1:
    sentencia1;
    ...
    break;
    ...
  case constanteN:
    sentenciaN;
    ...
```

```
        break;
    default:
        sentencia;
        ...
        break
  }
```

El valor de la expresión y de las constantes tiene que ser de tipo char, byte, short o int. No hay lugar para booleanos, reales ni long porque, en la ejecución, todos los valores que incorporamos se transforman en valores de tipo int.

Al evaluar la expresión de switch, el intérprete busca una constante con el mismo valor. Si la encuentra, ejecuta las sentencias asociadas a esta constante hasta que tropiece con un break. La sentencia break finaliza la ejecución de esta estructura. Si no encuentra ninguna constante que coincida con la expresión, busca la línea default. Si existe, ejecuta las sentencias que le siguen. La sentencia default es opcional.

```
 switch ( op ) {
default :
        System.out.println("error");
        break;
    case '+':
        System.out.println( a + b );
        break;
}
```