



Universidad Politécnica de Madrid

**Escuela Técnica Superior de
Ingenieros Informáticos**



Manual de Usuario:
Librería Java de Tabla de Símbolos

Contenido

1. Introducción	3
2. Uso	3
2.1. Crear gestor	3
2.2. Crear tabla de palabras reservadas	3
2.3. Definir atributos	3
2.4. Crear tabla global y local	4
2.5. Añadir entradas	4
2.6. Buscar entradas	4
2.7. Añadir tipo	5
2.8. Dar valor al resto de los atributos	6
2.9. Mostar y escribir la representación de la tabla de símbolos	7
2.10. Destruir una tabla de símbolos	7
2.11. Debug	7
3. API	7
3.1. Funciones	7
3.2. Enumerados	13
4. Ejemplos	14

1. Introducción

Este documento explica cómo instalar, configurar y utilizar la librería Java de Tabla de Símbolos.

La librería consta de cuatro ficheros dentro del paquete `tslib`: `Atributo.java`, `Entrada.java`, `TS.java` y `TS_Gestor.java`. La clase `TS_Gestor` es la que se utiliza para la gestión de tablas de símbolos, siendo las otras tres clases apoyo de esta. Esta clase es un gestor de tablas de símbolos que sirve como apoyo a cualquier compilador. Como requisito para añadir la librería es que la versión del JDK tiene que ser 15 o superior.

2. Uso

En este apartado se proporciona una guía rápida sobre cómo usar la librería. Para usar el gestor de forma correcta primero hay que crear la tabla de palabras reservadas, después definir los atributos y finalmente crear la tabla global. Después de estos pasos, el gestor se puede usar como se desee.

2.1. Crear gestor

Para crear el gestor solo hay que instanciar un objeto `TS_Gestor` con su constructor, al que hay que pasarle un nombre de fichero para cuando se quiera escribir en un fichero del contenido de la tabla de símbolos.

```
public TS_Gestor(String nombre_fichero);
```

Por ejemplo, si se instancia el gestor y se quiere crear un fichero de nombre `"tabla_de_simbolos.txt"` se haría así:

```
TS_Gestor gestor=new TS_Gestor("tabla de símbolos");
```

2.2. Crear tabla de palabras reservadas

Después de instanciar el gestor, hay que crear la tabla de palabras reservadas. Para ello, hay que llamar a la función `createTPalabrasReservadas`, que no tiene argumentos. A partir de aquí, ya se podrían añadir entradas a esta tabla.

```
gestor.createTPalabrasReservadas();
```

2.3. Definir atributos

Después de crear la tabla de palabras reservadas, se definen los atributos que se añadirán automáticamente a cada identificador según su tipo en las Tablas de Símbolos. Hay que tener en cuenta que los nombres de atributos que se pongan aquí son los que se utilizarán luego para referirse a dichos atributos a lo largo del programa.

Para definir un atributo hay que llamar a la función `createAtributo` con tres valores: el nombre del atributo que se le va a dar, la descripción del atributo y el tipo de dato del atributo. La descripción del atributo indica qué atributo predefinido o personalizado se quiere crear. El tipo de dato indica qué tipo de valor va a almacenar el atributo.

```
public int createAtributo(String nombre_atr, DescripcionAtributo tipo_des,
                          TipoDatoAtributo tipo_valor);
```

Un atributo predefinido se definiría de la siguiente manera. Si el atributo es el desplazamiento, se le quiere llamar `"dirección"` y se quiere almacenar un entero:

```
gestor.createAtributo("dirección", DescripcionAtributo.DIR, TipoDatoAtributo.ENTERO);
```

Un atributo personalizado se definiría de la siguiente manera. Si se le quiere llamar `"dimensión"` y se quiere almacenar un entero:

```
gestor.createAtributo("dimensión", DescripcionAtributo.OTROS, TipoDatoAtributo.ENTERO);
```

2.4. Crear tabla global y local

Después de la definición de atributos, ya se puede crear la tabla global. Para ello, se llama a la función `createTSGlobal`, que no necesita argumentos. A continuación, ya se podrían añadir entradas a la tabla global.

```
gestor.createTSGlobal();
```

También se puede crear la tabla local después de haber creado la tabla global. Hay que tener en cuenta que solo se puede tener una tabla local a la vez, ya que esta librería no se puede utilizar en compiladores para lenguajes con anidamiento de funciones. Para crear la tabla local hay que llamar a la función `createTSLocal` sin argumentos.

```
gestor.createTSLocal();
```

2.5. Añadir entradas

Para añadir un lexema en las diferentes tablas hay que llamar a diferentes funciones, una para cada tabla. Estas funciones devolverán la posición en la que se ha añadido el lexema, siendo en el caso de la tabla local una posición negativa y en el caso de la tabla global y la tabla de palabras reservadas una posición positiva.

Para añadir un lexema a la tabla de palabras reservadas hay que llamar a la función `addEntradaTPalabrasReservadas` con el lexema como argumento.

```
public int addEntradaTPalabrasReservadas(String lex);
```

En este ejemplo no se ha guardado la posición porque la posición no se considera importante para las palabras reservadas.

```
gestor.addEntradaTPalabrasReservadas("if");
```

Para añadir un lexema a la tabla global hay que llamar a la función `addEntradaTSGlobal` con el lexema como argumento.

```
public int addEntradaTSGlobal(String lex);
```

En este ejemplo, se guarda la posición en la que se ha añadido para más tarde poder consultarlo y dar valor a sus atributos.

```
int pos=gestor.addEntradaTSGlobal("contador");
```

Para añadir un lexema a la tabla local hay que llamar a la función `addEntradaTSLocal` con el lexema como argumento.

```
public int addEntradaTSLocal(String lex);
```

En este ejemplo, también se ha guardado la posición para su uso más tarde.

```
int pos=gestor.addEntradaTSLocal("a");
```

2.6. Buscar entradas

La librería permite buscar la posición en la que está un lexema en cualquiera de las tablas.

Para buscar una entrada en la tabla de palabras reservadas hay que llamar a la función `getEntradaTPalabrasReservadas` con el lexema a buscar como parámetro y devolverá su posición, que en este caso sería positiva, o 0 si el lexema no está en la tabla.

```
public int getEntradaTPalabrasReservadas(String lex);
```

Por ejemplo, si se quiere buscar la palabra reservada "else", habría que hacer:

```
int pos=gestor.getEntradaTPalabrasReservadas("else");
```

Para buscar en las tablas global y local hay dos formas distintas de buscar. Se puede buscar directamente en cada tabla independientemente o se puede hacer una búsqueda en ambas a la vez.

Para la búsqueda solo en la tabla global hay que llamar a la función `getEntradaTSGlobal` con el lexema a buscar como parámetro, y devolverá su posición, en este caso positiva, o 0 si no se ha encontrado en esta tabla.

```
public int getEntradaTSGlobal(String lex);
```

De la siguiente forma se busca el identificador “suma” en la tabla global:

```
int pos=gestor.getEntradaTSGlobal("suma");
```

Para la búsqueda en la tabla local hay que llamar a la función `getEntradaTSLocal` con el lexema a buscar como parámetro, y devolverá su posición, que en este caso es negativa, o 0 si no se ha encontrado en esta tabla.

```
public int getEntradaTSLocal(String lex);
```

El identificador “cadena1” se busca solo en la tabla local:

```
int pos=gestor.getEntradaTSLocal("cadena1");
```

Si se quiere buscar en ambas tablas de símbolos a la vez, hay que llamar a la función `getEntradaTS` con el lexema a buscar como parámetro y devolverá su posición o 0 si no se ha encontrado en ninguna de las dos tablas. Esta búsqueda empieza en la tabla local (si existe) y, si no encuentra el lexema, se busca en la tabla global.

```
public int getEntradaTS(String lex);
```

De esta manera, si se busca el identificador “cadena1” en las tablas de símbolos, se obtendrá una posición negativa si se ha encontrado en la tabla local; si no estaba, se obtendrá una posición positiva si se ha encontrado en la tabla global; si no estaba, se obtendrá un 0.

```
int pos=gestor.getEntradaTS("cadena1");
```

2.7. Añadir tipo

Para añadir el tipo de un identificador hay que llamar a la función `setTipo` con la posición del identificador y el tipo del identificador a asignar como parámetros. El identificador puede estar en cualquiera de las tablas (global o local) y el tipo tiene que ser uno de los valores concretos que se aceptan como válidos. Estos valores del tipo son: función, procedimiento, entero, cadena, real, lógico, puntero y vector. Los valores diferentes a estos darán error y la función devolverá un código de error. Si se quiere añadir el tipo función a un identificador en la tabla local también devolverá un error.

```
public int setTipo(int pos, String tipo_id);
```

Para indicar que el identificador que se encuentra en la posición `pos` es de tipo entero, se pondría:

```
gestor.setTipo(pos, "entero");
```

Al añadir el tipo de un identificador se añaden sus atributos de forma automática. Si el tipo es entero, cadena, real, lógico, puntero o vector se añadirán los atributos dirección y parámetro. Si el tipo es función o procedimiento se añadirán los atributos número de parámetros, tipo de parámetros, modo de parámetros, tipo de retorno y etiqueta. En cualquier caso, también se añadirán los atributos personalizados, si se han definido. Los valores predefinidos de los atributos que se añaden al dar valor al tipo son -1 o *null*, dependiendo del dato que utilice al atributo.

Para saber el tipo de un identificador hay que llamar a la función `getTipo` con la posición del identificador.

```
public String getTipo(int pos);
```

De esta forma, si después de haber fijado el tipo del identificador que está en la posición pos, se consulta su tipo, se obtendrá el tipo asignado.

```
String tipo=gestor.getTipo(pos);
```

2.8. Dar valor al resto de los atributos

Para dar valor a los atributos de un identificador hay que tener en cuenta el tipo de valor del atributo y cómo se ha definido al principio el nombre de este. El identificador puede estar en cualquiera de las tablas de símbolos (global o local).

Si se ha declarado el tipo de dato del atributo como entero, se tiene que llamar a la función setValorAtributoEnt con la posición, el nombre del atributo (el usado para definirlo) y el valor que se le quiere dar.

```
public int setValorAtributoEnt(int pos, String atr, int valor);
```

Así, si se ha creado el atributo dirección como entero, se puede poner un valor entero.

```
gestor.createAtributo("dirección", DescripcionAtributo.DIR, TipoDatoAtributo.ENTERO);  
gestor.setValorAtributoEnt(pos, "dirección", 0);
```

Para saber el valor de un atributo cuyo tipo de dato es entero hay que llamar a la función getValorAtributoEnt con la posición y el nombre del atributo (el usado para definirlo).

```
public int getValorAtributoEnt(int pos, String atr);
```

De la siguiente forma, se obtendría el valor entero del atributo dirección de un determinado identificador.

```
int valor=gestor.getValorAtributoEnt(pos, "dirección");
```

Si se ha declarado el tipo de dato del atributo como cadena, se tiene que llamar a la función setValorAtributoCad con la posición, el nombre del atributo (el usado para definirlo) y el valor que se le quiere dar.

```
public int setValorAtributoCad(int pos, String atr, String valor);
```

Así, si se ha creado el atributo etiqueta como cadena, se puede poner un valor de tipo cadena.

```
gestor.createAtributo("etiqueta", DescripcionAtributo.ETIQUETA,  
TipoDatoAtributo.CADENA);  
gestor.setValorAtributoCad(pos, "etiqueta", "sumatorio");
```

Para saber el valor de un atributo cuyo tipo de dato es cadena hay que llamar a la función getValorAtributoCad con la posición y el nombre del atributo (el usado para definirlo).

```
public int getValorAtributoCad(int pos, String atr);
```

De la siguiente forma, se obtendría el valor cadena del atributo etiqueta de un determinado identificador.

```
String valor=gestor.getValorAtributoCad(pos, "etiqueta");
```

Si el tipo de dato del atributo se ha declarado como lista, se tiene que llamar a la función setValorAtributoLista con la posición, el nombre del atributo (el usado para definirlo) y el valor que se le quiere dar.

```
public int setValorAtributoLista(int pos, String atr, String[] valor);
```

Así, si se ha creado el atributo tipo de parámetros como una lista, se puede poner una lista.

```
gestor.createAtributo("tipo de parámetros", DescripcionAtributo.TIPO_PARAM,  
TipoDatoAtributo.LISTA);  
String[] tipos={"int", "int"};  
gestor.setValorAtributoLista(pos, "tipo de parámetros", tipos);
```

Para saber el valor de un atributo cuyo tipo de dato es lista hay que llamar a la función `getValorAtributoLista` con la posición y el nombre del atributo (el usado para definirlo).

```
public int getValorAtributoLista(int pos, String[] atr);
```

De la siguiente forma, se puede saber la lista correspondiente al atributo tipo de parámetros de un determinado identificador.

```
String[] valor=gestor.getValorAtributoLista(pos, "tipo de parámetros");
```

2.9. Mostar y escribir la representación de la tabla de símbolos

Para mostrar por pantalla una tabla de símbolos hay que llamar a la función `show` con la tabla correspondiente.

```
gestor.show(Tabla.GLOBAL);
```

Para escribir una tabla de símbolos en un fichero, hay que llamar a la función `write` con la tabla correspondiente. La gestión del fichero es interna. Se puede llamar varias veces a esta función y no se reescribirá el fichero, sino que se añadirá al final de éste cada tabla de símbolos que se escriba.

```
gestor.write(Tabla.GLOBAL);
```

Los atributos personalizados sin valor de un identificador no aparecerán en esta representación.

2.10. Destruir una tabla de símbolos

Cuando se haya terminado con una tabla se debe destruir llamando a la función `destroy` junto con la tabla que se desee destruir. La destrucción implica la eliminación de todas las entradas.

```
gestor.destroy(Tabla.GLOBAL);
```

2.11. Debug

Todas las funciones de la librería pueden dar algún tipo de error, y las funciones lo muestran a través de un código de error que devuelven. Pero si se quiere una mejor comprensión de los errores es posible activar el modo debug de la librería. Este modo hace que se muestren por pantalla mensajes de los errores que se produzcan. Al crear el gestor esta propiedad está desactivada, pero para activarla solo hay que llamar a la función `activarDebug`.

```
gestor.activarDebug();
```

Esta función se puede activar en cualquier momento. De la misma forma, también se puede desactivar en cualquier momento, utilizando la función `desactivarDebug`.

```
gestor.desactivarDebug();
```

3. API

A continuación, se muestran las descripciones completas de todas las funciones y tipos que el usuario puede utilizar.

3.1. Funciones

TS_Gestor

```
public TS_Gestor(java.lang.String nombre_fichero)
```

Crea el gestor de tablas de símbolos.

Parámetros:

`nombre_fichero` - Fichero en el que se escribirán las tablas cuando se solicite.

activarDebug

```
public void activarDebug()
```

Activa los mensajes de error por pantalla.

desactivarDebug

```
public void desactivarDebug()
```

Desactiva los mensajes de error por pantalla.

createTPalabrasReservadas

```
public int createTPalabrasReservadas()
```

Crea la tabla de palabras reservadas

Returns:

0: si todo ha ido bien

1: si se está intentando crear una tabla de palabras reservadas cuando ya existe una.

createTSGlobal

```
public int createTSGlobal()
```

Crea la tabla global.

Returns:

0: si todo ha ido bien

1: si se está intentando crear una tabla que ya existe

2: si no se ha declarado la tabla de palabras reservadas antes

3: si no se han definido los atributos.

createTSLocal

```
public int createTSLocal()
```

Crea una tabla local.

Returns:

0: si todo ha ido bien.

2: si no se ha creado la tabla global.

14: si no se ha creado la tabla de palabras reservadas.

createAtributo

```
public int createAtributo(java.lang.String nombre_atr,  
TS_Gestor.DescripcionAtributo tipo_des, TS_Gestor.TipoDatoAtributo tipo_valor)
```

Define un atributo.

Parameters:

nombre_atr - Nombre del atributo.

tipo_des - Descripción del atributo.

tipo_valor - Tipo de dato del atributo.

Returns:

0: si todo ha salido bien.

4: si se está definiendo un atributo ya definido.

11: si se quiere utilizar un nombre para un atributo que ya está asignado a otro atributo.

13: si el nombre del atributo no es válido.

addEntradaTPalabrasReservadas

```
public int addEntradaTPalabrasReservadas(java.lang.String lex)
```

Añade una entrada a la tabla de palabras reservadas.

Parameters:

lex - Palabra reservada.

Returns:

La posición en la que se ha añadido.

0: si la tabla de palabras reservadas no existe o si la palabra reservada a añadir ya está en la tabla de palabras reservadas.

addEntradaTSGlobal

```
public int addEntradaTSGlobal(java.lang.String lex)
```

Crea una entrada en la tabla global.

Parameters:

lex - Lexema del identificador.

Returns:

La posición en la que se ha añadido.

0: si la tabla global no existe o si el identificador a añadir ya está en la tabla de símbolos global.

addEntradaTSLocal

```
public int addEntradaTSLocal(java.lang.String lex)
```

Crea una entrada en la tabla local.

Parameters:

lex - Lexema del identificador.

Returns:

La posición en la que se ha añadido.

0: si la tabla local lo existe o si el identificador a añadir ya está en la tabla de símbolos local.

getEntradaTPalabrasReservadas

```
public int getEntradaTPalabrasReservadas(java.lang.String lex)
```

Busca un lexema en la tabla de palabras reservadas.

Parameters:

lex - Palabra Reservada que se quiere buscar

Returns:

La posición de la palabra reservada.

0: si no se ha encontrado el lexema entre las palabras reservadas definidas o si la tabla de palabras reservadas no está creada.

getEntradaTSGlobal

```
public int getEntradaTSGlobal(java.lang.String lex)
```

Busca en la tabla global el lexema de un identificador.

Parameters:

lex - Lexema del identificador.

Returns:

La posición del identificador en la tabla global.

0: si no se ha encontrado el lexema o si la tabla global no está creada.

getEntradaTSLocal

```
public int getEntradaTSLocal(java.lang.String lex)
```

Busca en la tabla local el lexema de un identificador.

Parameters:

lex - Lexema del identificador.

Returns:

La posición del identificador en la tabla local.

0: si no se ha encontrado el lexema o si la tabla local no está creada.

getEntradaTS

```
public int getEntradaTS(java.lang.String lex)
```

Busca en la tabla global y local (si existe).

Parameters:

lex - Lexema del identificador.

Returns:

La posición del identificador en la tabla en la que está (siendo negativo si se encuentra en la tabla local y positivo si se encuentra en la tabla global).

0: si no se ha encontrado o si la tabla global no está creada.

setTipo

```
public int setTipo(int pos, java.lang.String tipo_id)
```

Añade el tipo de un identificador y añade los atributos pertinentes. Si el tipo es entero, cadena, real, lógico, puntero o vector se añadirán los atributos dirección y param. Si el tipo es función o procedimiento se añadirán los atributos número de parámetros, tipo de parámetros, modo de parámetros, tipo de retorno y etiqueta. En cualquier caso, se añadirán los atributos OTROS si se han definido.

Parameters:

pos - Posición del identificador.

tipo_id - Tipo que se le quiere poner al identificador. Los tipos posibles son: función, procedimiento, entero, cadena, real, lógico, puntero y vector.

Returns:

0: si todo ha salido bien.

5: si la posición no es correcta.

6: si el tipo no es correcto.

7: si el tipo ya existía.

10: si la tabla en la que estaría el identificador no existe.

12: si se intenta añadir el tipo función a un identificador de la tabla local.

getTipo

```
public java.lang.String getTipo(int pos)
```

Devuelve el tipo de un identificador.

Parameters:

pos - Posición del identificador.

Returns:

El tipo de la entrada en la posición pos.

null: si hay error, que puede ser porque la posición no sea correcta o porque la tabla en la que estaría el identificador no existe.

setValorAtributoEnt

```
public int setValorAtributoEnt(int pos, java.lang.String atr, int valor)
```

Da valor entero a un atributo de un identificador.

Parameters:

pos - Posición del identificador.

atr - Nombre del atributo.

valor - Valor del atributo.

Returns:

0: si todo ha salido bien.

5: si la posición no es correcta.

7: si el atributo ya tenía un valor asignado.

8: si el tipo de dato del atributo no está definido como ENTERO.

9: si el atributo no existe.

10: si la tabla en la que estaría el identificador no existe.

getValorAtributoEnt

```
public int getValorAtributoEnt(int pos, java.lang.String atr)
```

Devuelve el valor entero de un atributo de un identificador.

Parameters:

pos - Posición del identificador.

atr - Nombre del atributo.

Returns:

El valor entero del atributo atr del identificador de la posición pos

-1: si hay error, que puede ser porque el atributo o la posición no sean correctos, porque la tabla en la que estaría el identificador no existe o porque el atributo no esté declarado como ENTERO.

setValorAtributoCad

```
public int setValorAtributoCad(int pos, java.lang.String atr, java.lang.String valor)
```

Da valor de cadena a un atributo de un identificador.

Parameters:

pos - Posición del identificador.

atr - Nombre del atributo.
valor - Valor del atributo.

Returns:

0: si todo ha salido bien.
5: si la posición no es correcta.
7: si el atributo ya tenía un valor asignado.
8: si el tipo de dato del atributo no está definido como CADENA.
9: si el atributo no existe.
10: si la tabla en la que estaría el identificador no existe.

getValorAtributoCad

```
public java.lang.String getValorAtributoCad(int pos,  
java.lang.String atr)
```

Devuelve el valor de cadena de un atributo de un identificador.

Parameters:

pos - Posición del identificador.
atr - Nombre del atributo.

Returns:

El valor de cadena del atributo atr del identificador de la posición pos.
null: si hay error, que puede ser porque el atributo o la posición no sean correctos, porque la tabla en la que estaría el identificador no existe o porque el atributo no esté declarado como CADENA.

setValorAtributoLista

```
public int setValorAtributoLista(int pos, java.lang.String atr,  
java.lang.String[] valor)
```

Da valor en forma de lista de cadenas a un atributo de un identificador.

Parameters:

pos - Posición del identificador.
atr - Nombre del atributo.
valor - Valor del atributo.

Returns:

0 si todo ha salido bien.
5: si la posición no es correcta.
7: si el atributo ya tenía un valor asignado.
8: si el tipo de dato del atributo no está definido como LISTA.
9: si el atributo no existe.
10: si la tabla en la que estaría el identificador no existe.

getValorAtributoLista

```
public java.lang.String[] getValorAtributoLista(int pos, java.lang.String atr)
```

Devuelve el valor en forma de lista de cadenas para un atributo de un identificador.

Parameters:

pos - Posición del identificador.
atr - Nombre del atributo.

Returns:

El valor en forma de lista de cadenas del atributo atr del identificador de la posición pos.

null: si hay error, que puede ser porque el atributo o la posición no sean correctos, porque la tabla en la que estaría el identificador no existe o porque el atributo no esté declarado como LISTA.

show

```
public int show(TS_Gestor.Tabla tabla)
```

Muestra por pantalla una tabla de símbolos.

Parameters:

tabla - Tabla que se quiere mostrar.

Returns:

0: si todo ha salido bien

10: si la tabla que se quiere mostrar no existe.

write

```
public int write(TS_Gestor.Tabla tabla)
```

Escribe en un fichero el contenido de una tabla de símbolos.

Parameters:

tabla - Tabla que se quiere escribir.

Returns:

0: si se ha escrito correctamente.

10: si la tabla que se quiere escribir no existe.

destroy

```
public int destroy(TS_Gestor.Tabla tabla)
```

Destruye una tabla de símbolos.

Parameters:

tabla - Tabla que se quiere destruir.

Returns:

0: si todo ha salido bien.

10: si la tabla que se quiere destruir no existe.

15: si se está intentando destruir la tabla global sin haber destruido la tabla local.

16: si se está intentando destruir la tabla de palabras reservadas sin haber destruido la tabla global.

3.2. Enumerados

A continuación, se muestran las enumeraciones que el usuario debe utilizar junto con las funciones.

Enum TS_Gestor.DescripcionAtributo

DIR: Dirección o desplazamiento del identificador.

NUM_PARAM: Número de parámetros de una función.

TIPO_PARAM: Tipo de los parámetros de una función.

MODOS_PARAM: Modo de los parámetros de una función.

TIPO_RET: Tipo de retorno de una función.

ETIQUETA: Nombre de la etiqueta asociada a la función.

PARAM: Si la variable es un parámetro de una función o no.

OTROS: Cualquier otro tipo de atributo que se quiera definir.

Enum `TS_Gestor.TipoDatoAtributo`

ENTERO: Valor entero para un atributo.

CADENA: Valor de cadena para un atributo.

LISTA: Valor de lista de cadenas para un atributo.

Enum `TS_Gestor.Tabla`

GLOBAL: Referencia a la tabla global.

LOCAL: Referencia a la tabla local.

PALRES: Referencia a la tabla de palabras reservadas.

4. Ejemplos

A continuación, se muestra un ejemplo completo de uso de la librería. El código que analizaría el compilador sería el siguiente:

```
int main(){
    int x=2;
    int y=3;
    int res=suma(2,3)
    String cadena1="hola";
    String cadena2="buenos";
    String[] cadenas={cadena1, cadena2};
}

public int suma(int a, int b){
    int res;
    res=a+b
    return res;
}
```

En primer lugar, se crea el gestor.

```
gestor=new TS_Gestor("tabla_de_simbolos.txt");
```

Después se crea la tabla de palabras reservadas y se añaden algunas palabras reservadas a la tabla.

```
int res=gestor.createTPalabrasReservadas();
if(res==0){
    System.out.println("Error en la creación de la tabla de palabras reservadas");
    exit(1);
}
res=gestor.addEntradaTPalabrasReservadas("if");
if(res==0){
    System.out.println("Error: añadir palabra reservada a la tabla de palabras reservadas");
    exit(1);
}
res=gestor.addEntradaTPalabrasReservadas("else");
if(res==0){
    System.out.println("Error: añadir palabra reservada a la tabla de palabras reservadas");
    exit(1);
}
res=gestor.addEntradaTPalabrasReservadas("while");
```

```

if(res==0){
    System.out.println("Error: añadir palabra reservada a la tabla de palabras reservadas");
    exit(1);
}
res=gestor.addEntradaTPalabrasReservadas("do");

if(res==0){
    System.out.println("Error: añadir palabra reservada a la tabla de palabras reservadas");
    exit(1);
}

```

A continuación, se definen los atributos.

```

res=gestor.createAtributo("dirección", DescripcionAtributo.DIR, TipoDatoAtributo.ENTERO);
if(res!=0){
    System.out.println("Error al definir un atributo.");
    exit(1);
}
res=gestor.createAtributo("número de parámetros", DescripcionAtributo.NUM_PARAM,
    TipoDatoAtributo.ENTERO);
if(res!=0){
    System.out.println("Error al definir un atributo.");
    exit(1);
}
res=gestor.createAtributo("tipo de parámetros", DescripcionAtributo.TIPO_PARAM,
    TipoDatoAtributo.LISTA);
if(res!=0){
    System.out.println("Error al definir un atributo.");
    exit(1);
}
res=gestor.createAtributo("modo de parámetros", DescripcionAtributo.MODO_PARAM,
    TipoDatoAtributo.LISTA);
if(res!=0){
    System.out.println("Error al definir un atributo.");
    exit(1);
}
res=gestor.createAtributo("tipo de retorno", DescripcionAtributo.TIPO_RET,
    TipoDatoAtributo.CADENA);
if(res!=0){
    System.out.println("Error al definir un atributo.");
    exit(1);
}
res=gestor.createAtributo("etiqueta", DescripcionAtributo.ETIQUETA, TipoDatoAtributo.CADENA);
if(res!=0){
    System.out.println("Error al definir un atributo.");
    exit(1);
}
res=gestor.createAtributo("parámetro", DescripcionAtributo.PARAM, TipoDatoAtributo.ENTERO);
if(res!=0){
    System.out.println("Error al definir un atributo.");
    exit(1);
}
res=gestor.createAtributo("dimensión", DescripcionAtributo.OTROS, TipoDatoAtributo.ENTERO);
if(res!=0){
    System.out.println("Error al definir un atributo.");
    exit(1);
}
res=gestor.createAtributo("elem", DescripcionAtributo.OTROS, TipoDatoAtributo.CADENA);
if(res!=0){
    System.out.println("Error al definir un atributo.");
    exit(1);
}
}

```

Ahora, se crea la tabla global y se añaden algunos identificadores.

```

res=gestor.createTSGlobal();
if(res!=0){
    System.out.println("Error al intentar crear la tabla de símbolos global");
}

```

```

        exit(1);
    }
    int pos=gestor.addEntradaTSGlobal("x");
    if(pos==0){
        System.out.println("Error al añadir un identificador a la tabla de símbolos global");
        exit(1);
    }
    res=gestor.setTipo(pos, "entero");
    if(res!=0){
        System.out.println("Error al intentar dar valor al tipo de un identificador");
        exit(1);
    }
    res=gestor.setValorAtributoEnt(pos, "dirección", 0);
    if(res!=0){
        System.out.println("Error al intentar dar valor a un atributo de un identificador");
        exit(1);
    }
    res=gestor.setValorAtributoEnt(pos, "parámetro", 0);
    if(res!=0){
        System.out.println("Error al intentar dar valor a un atributo de un identificador");
        exit(1);
    }
    pos=gestor.addEntradaTSGlobal("y");
    if(pos==0){
        System.out.println("Error al añadir un identificador a la tabla de símbolos global");
        exit(1);
    }
    res=gestor.setTipo(pos, "entero");
    if(res!=0){
        System.out.println("Error al intentar dar valor al tipo de un identificador");
        exit(1);
    }
    res=gestor.setValorAtributoEnt(pos, "dirección", 4);
    if(res!=0){
        System.out.println("Error al intentar dar valor a un atributo de un identificador");
        exit(1);
    }
    res=gestor.setValorAtributoEnt(pos, "parámetro", 0);
    if(res!=0){
        System.out.println("Error al intentar dar valor a un atributo de un identificador");
        exit(1);
    }
    pos=gestor.addEntradaTSGlobal("res");
    if(pos==0){
        System.out.println("Error al añadir un identificador a la tabla de símbolos global");
        exit(1);
    }
    res=gestor.setTipo(pos, "entero");
    if(res!=0){
        System.out.println("Error al intentar dar valor al tipo de un identificador");
        exit(1);
    }
    res=gestor.setValorAtributoEnt(pos, "dirección", 8);
    if(res!=0){
        System.out.println("Error al intentar dar valor a un atributo de un identificador");
        exit(1);
    }
    res=gestor.setValorAtributoEnt(pos, "parámetro", 0);
    if(res!=0){
        System.out.println("Error al intentar dar valor a un atributo de un identificador");
        exit(1);
    }
}

```

Ahora, se añade una función a la tabla global, se añaden sus atributos y se crea la tabla local.

```

pos=gestor.addEntradaTSGlobal("suma");
if(pos==0){

```



```

        System.out.println("Error al añadir un identificador a la tabla de símbolos global");
        exit(1);
    }
    res=gestor.setTipo(pos, "función");
    if(res!=0){
        System.out.println("Error al intentar dar valor al tipo de un identificador");
        exit(1);
    }
    res=gestor.setValorAtributoEnt(pos, "número de parámetros", 2);
    if(res!=0){
        System.out.println("Error al intentar dar valor a un atributo de un identificador");
        exit(1);
    }
    String[] tipos={"int", "int"};
    res=gestor.setValorAtributoLista(pos, "tipo de parámetros", tipos);
    if(res!=0){
        System.out.println("Error al intentar dar valor a un atributo de un identificador");
        exit(1);
    }
    String[] modos={"0", "0"};
    res=gestor.setValorAtributoLista(pos, "modo de parámetros", modos);
    if(res!=0){
        System.out.println("Error al intentar dar valor a un atributo de un identificador");
        exit(1);
    }
    res=gestor.setValorAtributoCad(pos, "tipo de retorno", "int");
    if(res!=0){
        System.out.println("Error al intentar dar valor a un atributo de un identificador");
        exit(1);
    }
    res=gestor.setValorAtributoCad(pos, "etiqueta", "sumatorio");
    if(res!=0){
        System.out.println("Error al intentar dar valor a un atributo de un identificador.");
        exit(1);
    }
    res=gestor.createTSLocal();
    if(res!=0){
        System.out.println("Error al intentar crear una tabla de símbolos local");
        exit(1);
    }
}

```

Se añaden todos los identificadores de ese ámbito, y finalmente, se escribe el contenido de la tabla de símbolos local en el fichero y se destruye la tabla.

```

pos=gestor.addEntradaTSLocal("a");
if(pos==0){
    System.out.println("Error al añadir un identificador a la tabla de símbolos local");
    exit(1);
}
res=gestor.setTipo(pos, "entero");
if(res!=0){
    System.out.println("Error al intentar dar valor al tipo de un identificador");
    exit(1);
}
res=gestor.setValorAtributoEnt(pos, "dirección", 0);
if(res!=0){
    System.out.println("Error al intentar dar valor a un atributo de un identificador");
    exit(1);
}
res=gestor.setValorAtributoEnt(pos, "parámetro", 1);
if(res!=0){
    System.out.println("Error al intentar dar valor a un atributo de un identificador");
    exit(1);
}
pos=gestor.addEntradaTSLocal("b");
if(pos==0){
    System.out.println("Error al añadir un identificador a la tabla de símbolos local");
    exit(1);
}

```

```

}
res=gestor.setTipo(pos, "entero");
if(res!=0){
    System.out.println("Error al intentar dar valor al tipo de un identificador");
    exit(1);
}
res=gestor.setValorAtributoEnt(pos, "dirección", 4);
if(res!=0){
    System.out.println("Error al intentar dar valor a un atributo de un identificador");
    exit(1);
}
res=gestor.setValorAtributoEnt(pos, "parámetro", 1);
if(res!=0){
    System.out.println("Error al intentar dar valor a un atributo de un identificador");
    exit(1);
}
pos=gestor.addEntradaTSLocal("res");
if(pos==0){
    System.out.println("Error al añadir un identificador a la tabla de símbolos local");
    exit(1);
}
res=gestor.setTipo(pos, "entero");
if(res!=0){
    System.out.println("Error al intentar dar valor al tipo de un identificador");
    exit(1);
}
res=gestor.setValorAtributoEnt(pos, "dirección", 8);
if(res!=0){
    System.out.println("Error al intentar dar valor a un atributo de un identificador");
    exit(1);
}
res=gestor.setValorAtributoEnt(pos, "parámetro", 0);
if(res!=0){
    System.out.println("Error al intentar dar valor a un atributo de un identificador");
    exit(1);
}
res=gestor.write(Tabla.LOCAL);
if(res!=0){
    System.out.println("Error al intentar escribir la tabla local en el fichero");
    exit(1);
}
res=gestor.destroy(Tabla.LOCAL);
if(res!=0){
    System.out.println("Error al intentar destruir la tabla local");
    exit(1);
}
}

```

Ahora, se continúa añadiendo identificadores a la tabla global hasta finalizar con todos los identificadores.

```

pos=gestor.addEntradaTSGlobal("cadena1");
if(pos==0){
    System.out.println("Error al añadir un identificador a la tabla de símbolos global");
    exit(1);
}
res=gestor.setTipo(pos, "cadena");
if(res!=0){
    System.out.println("Error al intentar dar valor al tipo de un identificador");
    exit(1);
}
res=gestor.setValorAtributoEnt(pos, "dirección", 12);
if(res!=0){
    System.out.println("Error al intentar dar valor a un atributo de un identificador");
    exit(1);
}
res=gestor.setValorAtributoEnt(pos, "parámetro", 0);
if(res!=0){
    System.out.println("Error al intentar dar valor a un atributo de un identificador");
    exit(1);
}

```

```

        exit(1);
    }
    pos=gestor.addEntradaTSGlobal("cadena2");
    if(pos==0){
        System.out.println("Error al añadir un identificador a la tabla de símbolos global");
        exit(1);
    }
    res=gestor.setTipo(pos, "cadena");
    if(res!=0){
        System.out.println("Error al intentar dar valor al tipo de un identificador");
        exit(1);
    }
    res=gestor.setValorAtributoEnt(pos, "dirección", 20);
    if(res!=0){
        System.out.println("Error al intentar dar valor a un atributo de un identificador");
        exit(1);
    }
    res=gestor.setValorAtributoEnt(pos, "parámetro", 0);
    if(res!=0){
        System.out.println("Error al intentar dar valor a un atributo de un identificador");
        exit(1);
    }
    pos=gestor.addEntradaTSGlobal("cadenas");
    if(pos==0){
        System.out.println("Error al añadir un identificador a la tabla de símbolos global");
        exit(1);
    }
    res=gestor.setTipo(pos, "vector");
    if(res!=0){
        System.out.println("Error al intentar dar valor al tipo de un identificador");
        exit(1);
    }
    res=gestor.setValorAtributoEnt(pos, "dirección", 50);
    if(res!=0){
        System.out.println("Error al intentar dar valor a un atributo de un identificador");
        exit(1);
    }
    res=gestor.setValorAtributoEnt(pos, "parámetro", 0);
    if(res!=0){
        System.out.println("Error al intentar dar valor a un atributo de un identificador");
        exit(1);
    }
    res=gestor.setValorAtributoEnt(pos, "dimensión", 2);
    if(res!=0){
        System.out.println("Error al intentar dar valor a un atributo de un identificador");
        exit(1);
    }
    res=gestor.setValorAtributoCad(pos, "elem", "String");
    if(res!=0){
        System.out.println("Error al intentar dar valor a un atributo de un identificador");
        exit(1);
    }
}

```

Finalmente, se escribe la tabla global en el fichero y se destruyen tanto la tabla global como la tabla de palabras reservadas.

```

res=gestor.write(Tabla.GLOBAL);
if(res!=0){
    System.out.println("Error al intentar escribir la tabla global en el fichero");
    exit(1);
}
res=gestor.destroy(Tabla.GLOBAL);
if(res!=0){
    System.out.println("Error al intentar destruir la tabla global");
    exit(1);
}
res=gestor.destroy(Tabla.PALRES);
if(res!=0){

```

```

        System.out.println("Error al intentar destruir la tabla de palabras reservadas");
        exit(1);
    }
}

```

La salida de fichero del compilador sería la siguiente:

```

TABLA LOCAL #2:
*LEXEMA: 'a'
  Atributos:
  + Tipo: 'entero'
  + Despl: 0
  + Param: 1
*LEXEMA: 'b'
  Atributos:
  + Tipo: 'entero'
  + Despl: 4
  + Param: 1
*LEXEMA: 'res'
  Atributos:
  + Tipo: 'entero'
  + Despl: 8
  + Param: 0
-----
TABLA PRINCIPAL #1:
*LEXEMA: 'x'
  Atributos:
  + Tipo: 'entero'
  + Despl: 0
  + Param: 0
*LEXEMA: 'y'
  Atributos:
  + Tipo: 'entero'
  + Despl: 4
  + Param: 0
*LEXEMA: 'res'
  Atributos:
  + Tipo: 'entero'
  + Despl: 8
  + Param: 0
*LEXEMA: 'suma'
  Atributos:
  + Tipo: 'funcion'
  + numParam: 2
  + TipoParam0: 'int'
  + TipoParam1: 'int'
  + ModoParam0: '0'
  + ModoParam1: '0'
  + TipoRetorno: 'int'
  + EtiqFuncion: 'sumatorio'
*LEXEMA: 'cadena1'
  Atributos:
  + Tipo: 'cadena'
  + Despl: 12
  + Param: 0
*LEXEMA: 'cadena2'
  Atributos:
  + Tipo: 'cadena'
  + Despl: 20
  + Param: 0
*LEXEMA: 'cadenas'
  Atributos:
  + Tipo: 'vector'
  + Despl: 50
  + Param: 0
  + elem: 'String'
  + dimension: 2

```