Departamento de Cs. e Ingeniería de la Computación
Universidad Nacional del Sur

Compiladores e Intérpretes

Compilador de MINIJAVA

Manual de usuario

Ricardo Ferro Moreno

Universidad Nacional del Sur
Bahía Blanca - 30 de noviembre de 2016

Índice

1. Introducción

En el siguiente documento se darán detalles de cómo compilar y utilizar el compilador de MINIJAVA, así como también detalles de los mensajes que pueden aparecer en pantalla.

MINIJAVA es una versión reducida del lenguaje JAVA. Las especificaciones del lenguaje se pueden encontrar en los apuntes brindados por la cátedra:

- Enunciado general del proyecto
  http://cs.uns.edu.ar/~lc/cei/downloads/Proyecto/Enunciado%20Gral%20del%20Proyecto.pdf
- Sintaxis de MiniJava
  http://cs.uns.edu.ar/~lc/cei/downloads/Proyecto/Sintaxis%20MiniJava.pdf
- Semántica de MiniJava
  http://cs.uns.edu.ar/~lc/cei/downloads/Proyecto/MiniJava-Semantica.pdf

2. Compilación

El programa fue desarrollado *Java Development Kit 1.8.0_101*, por lo que se sugiere compilar con una versión de JDK superior a 1.8.0.

Para realizar la compilación desde la línea de comandos o terminal, es necesario situarse en la carpeta **src** del código fuente, para luego ejecutar el siguiente comando:

```
javac modulos/Principal.java
```

Con dicho comando se crearán todos los archivos **class** de JAVA.

3. Modo de uso

Una vez compilado el programa, se puede ejecutar el mismo mediante la máquina virtual de JAVA, siguiendo con el uso de la línea de comandos o terminal es necesario escribir el siguiente comando:

```
java modulos.Principal <Entrada> [Salida]
```

Donde *Entrada* (parámetro obligatorio) representa la ruta al archivo de entrada a ser compilado, mientras que *Salida* (parámetro opcional) representa la ruta al archivo de salida donde se generará el código intermedio.

En caso de no estar especificado el archivo de salida, el compilador intentará generar un nuevo archivo bajo el mismo nombre del archivo de entrada, agregándole una extensión **ceiasm**. Por ejemplo, si el archivo de entrada es *Programa.java* y no se especifica archivo de salida, el código intermedio será generado en un nuevo archivo *Programa.java.ceiasm* en la misma carpeta donde está siendo ejecutado el compilador.

Una vez generado el código intermedio, para poder ejecutarlo es necesario utilizar la máquina virtual CEIVM brindada por la cátedra de la siguiente manera:

```
java –jar CEIVM.jar <CódigoIntermedio>
```

Donde *CódigoIntermedio* representa la ruta al archivo de código intermedio, generado en el paso anterior, y *CEIVM.jar* el archivo ejecutable de la CEIVM.

4. Mensajes del compilador

En esta sección se dará detalle de los distintos tipos de mensajes que puede mostrar el compilador.

4.1. Compilación correcta

En caso de que la compilación sea correcta, significa que el código fuente del archivo de entrada es léxica, sintáctica y semánticamente correcto, se mostrará el siguiente mensaje:

```
El analisis lexico, sintactico y semantico se realizo de forma exitosa
```

Si aparece dicho cartel en pantalla, significa que todo el proceso de compilacción se realizó correctamente, y en el archivo especificado de salida estará guardado el código intermedio generado por el compilador.

4.2. Advertencias

El compilador puede advertir al usuario ante la presencia de situaciones particulares, que son merecedoras de informar pero no afectan al proceso de compilación. Dichas advertencias serán notificadas en pantalla, pero la compilación del código fuente seguirá su transcurso.

4.2.1. Archivo de salida

Si no se especifica el archivo de salida el compilador advertirá la situación e informará el nombre del archivo de salida que intentará usar, en caso de que pueda crearlo y modificarlo. Es necesario mencionar que pueden existir errores posteriores en caso de que no se tuvieran los permisos de creación o modificación de archivos.

Suponiendo que el archivo de entrada es `ejemplo.java`, el mensaje de advertencia sería:

```
[Advertencia] No se especifico archivo de salida.
[Advertencia] Se utilizara la siguiente salida: "ejemplo.java.ceiasm".
```

4.2.2. Múltiples métodos `main`

En caso de que el código fuente contenga múltiples métodos `main`, que sean estáticos, cuyo tipo de retorno sea `void` y no tenga argumentos formales, es necesario que el compilador elija a uno de ellos que será el método principal que dará inicio a la ejecución del programa.

Si se diera esta ocurrencia, el compilador dará anuncio de la situación e informará al usuario cuál es el método `main` a ser considerado.

```
[Advertencia] Se encontraron varias clases con main.
[Advertencia] Se utilizara el main de la clase 'NombreClase'.
```

Donde *NombreClase* es el nombre de la clase cuyo `main` será el considerado.

4.3. Errores de invocación/archivos

A continuación se listarán los mensajes de error que se pueden apreciar al utilizar el compilador. Cabe la aclaración que cuando un mensaje de error es mostrado significa que se ha abortado el proceso de compilación, dado que no se posee ningún mecanismo de recuperación ante errores.

4.3.1. Invocación incorrecta

En caso de que se realice una invocación incorrecta del programa, se informará dicho error dando un ejemplo de cómo se debe invocar al programa correctamente.

```
[Error] Cantidad invalida de argumentos.
Modo de uso: java modulos.Principal <Entrada> [<Salida>]
En caso de no especificarse archivo de salida, se le otorgara
el mismo nombre que el de entrada con extension .ceiasm
```

4.3.2. Archivo de entrada inexistente

Si el archivo de entrada brindado por el usuario no existe, no es posible realizar el proceso de compilación.

```
[Error] No existe el archivo de entrada especificado.
```

4.3.3. Creación del archivo de salida

En caso de que no se posean permisos para crear o modificar el archivo de salida (probablemente por cuestiones de permiso de usuario en el sistema operativo) se informará el siguiente mensaje de error.

```
[Error] Fallo al intentar crear el archivo de salida
```

4.3.4. Lectura en el archivo de entrada

Si fallara la lectura sobre el archivo de entrada (código fuente), se informaría dicha situación al usuario.

```
Error leyendo del archivo de entrada
```

4.3.5. Escritura en el archivo de salida

Análogo al caso anterior. Si fallara la escritura sobre el archivo de salida (el de código intermedio), se informaría dicha situación al usuario.

```
Error al intentar escribir el archivo de salida.
```

4.3.6. Error general de archivos

Cualquier otro error de archivos que no haya entrado en alguno de los casos anteriores se informará de la siguiente manera:

```
Error de archivos. Revisar que el archivo de entrada sea correcto.
```

4.3.7. Error general

Si por algún motivo en el proceso de compilación existe un fallo inesperado y el error no es listado entre los errores que se enumeran en este documento, se mostrará el siguiente mensaje:

```
Se produjo un error.
```

Si existiera alguna limitación por recursos insuficientes el mismo sería mostrado mediante éste mensaje. Sería deseable no llegar a este caso, ya que es el caso extremo de los errores. En caso de errores de compilación (léxicos, sintácticos, semánticos), de archivos o de invocación, se espera que el mensaje se identifique y se muestre correctamente. En cualquier otro tipo de error, se mostrará éste mensaje.

4.4. Errores léxicos

Cuando se produzca un error léxico, además de informar el respectivo error se mostrará el siguiente mensaje.

```
No se pudo completar el analisis lexico.
```

Con dicho mensaje se brinda la facilidad de identificar que el proceso de compilación se abortó durante el análisis léxico y no en las etapas posteriores.

4.4.1. Operador inválido

Uno de los errores léxicos detectables por el compilador es la formación de un operador inválido. El mensaje de error es el siguiente:

```
[Error Lexico] Operador invalido '<OP>' encontrado en la linea XX.
```

El mismo se puede apreciar cuando se forma un operador && ó || mal formado. La manera de que esto suceda es que el analizador léxico haya encontrado el primer caracter del operador, y al momento de leer el segundo caracter se encontró con algo diferente a lo esperado.

4.4.2. String mal formado

En caso de que un literal `String` esté mal formado, se reportará de la siguiente manera

```
[Error Lexico] Literal String mal formado en la linea XX
```

Para que un `String` esté mal formado, es necesario realizar la apertura del literal (utilizando comillas dobles) y que no hayan sido cerrado, o que haya aparecido un *Enter* antes de su cierre.

4.4.3. Char mal formado

En caso de que un literal `char` esté mal formado, se reportará el siguiente error:

```
[Error Lexico] Literal caracter mal formado en la linea XX
```

Para que un `char` esté mal formado, es necesario realizar la apertura del literal (utilizando comilla simple) y que no hayan sido cerrado, que haya aparecido un *Enter* antes de su cierre, o que la longitud del literal no corresponda con la longitud de un literal (1 ó 2 caracteres, dependiendo de los símbolos utilizados).

4.4.4. Comentario mal formado

En caso de que un comentario esté mal formado, se reportará el siguiente error:

```
[Error Lexico] Comentario mal formado en la linea XX
```

Un comentario se considerará mal formado en caso de que se realice la apertura de un comentario multilínea, y que no haya sido cerrado.

4.4.5. Caracter inválido

En caso de encontrar un caracter que no pertenezca al alfabeto reconocido por el compilador, se mostrará el siguiente mensaje por pantalla:

```
[Error Lexico] Caracter invalido '<CH>' encontrado en la linea XX
```

El alfabeto permitido por el compilador puede encontrarse en el *Manual Técnico*.

4.5. Errores sintácticos

Cuando se produzca un error sintáctico, además de informar el respectivo error se mostrará el siguiente mensaje.

```
No se pudo completar el analisis sintactico.
```

Con dicho mensaje se brinda la facilidad de identificar que el proceso de compilación se abortó durante el análisis sintáctico y no en las etapas posteriores.

4.5.1. Expresión mal formada

En caso de encontrar una expresión mal formada, el compilador mostrará el siguiente mensaje de error:

```
[Error Sintactico] Expresion mal formada en la linea XX
```

Para comprobar la correctitud sintática de una expresión, se recomienda leer el apunte *Sintaxis de Minijava*.

4.5.2. Argumento formal mal formado

En caso de encontrar un argumento formal mal formado, el compilador mostrará el siguiente mensaje de error:

```
[Error Sintactico] Argumento formal mal formado en la linea XX
```

4.5.3. Literal faltante

Cuando se espera por un literal y aparece cualquier otro tipo de expresión, se muestra el siguiente mensaje de error.

```
[Error Sintactico] Error en la linea XX
[Error Sintactico] Se esperaba un literal y se encontro '<EXP>'.
```

4.5.4. Sentencias

En caso de existir algún error sintáctico al momento de leer una lista de sentencias, se mostrará el siguiente error por pantalla:

```
[Error Sintactico] Error en la linea XX
[Error Sintactico] Error leyendo lista de sentencias
```

Además, si una sentencia está mal formada, se mostrará:

```
[Error Sintactico] Error en la linea XX
[Error Sintactico] Sentencia mal formada
```

4.5.5. Asignación o Sentencia

Cuando el analizador sintáctico procesa el código fuente, en caso de leer un *primario* al comenzar una sentencia, se puede tratar de una asignación o de una sentencia llamada. En caso de que la misma estuviera mal formada, se mostrará el siguiente mensaje:

```
[Error Sintactico] Error en la linea XX
[Error Sintactico] Se esperaba una asignacion o una sentencia
```

4.5.6. Operando inválido

En caso de que se espere por un *literal* o un *primario* y no encontrar ningún lexema que corresponda a alguno de éstos, se mostrará el siguiente mensaje:

```
[Error Sintactico] Error en la linea XX
[Error Sintactico] Operando invalido
```

4.5.7. Errores sintácticos generales

Para otros tipos de errores, aparecerá el siguiente mensaje:

```
[Error Sintactico] Error en la linea XX
[Error Sintactico] Se esperaba un <Esperado>
y se encontro un "<Encontrado>"
```

Donde *Esperado* es lo que sintácticamente esperaba encontrar el compilador, y *Encontrado* fue lo que se halló durante la comprobación sintáctica.

4.6. Errores semánticos

Cuando se produzca un error semántico, además de informar el respectivo error se mostrará el siguiente mensaje.

```
No se pudo completar el analisis semantico.
```

Con dicho mensaje se brinda la facilidad de identificar que el proceso de compilación se abortó durante el análisis semántico y no en las etapas posteriores.

4.6.1. Atributo duplicado

En caso de que un atributo (esto es, una variable de instancia) sea declarado en un mismo contexto y el nombre de dicho atributo ya esté en uso, se informará el siguiente error:

```
[Error Semantico] Error en la linea XX
[Error Semantico] El atributo '<ATR>' ya esta declarado.
```

Donde *ATR* es el nombre del atributo en cuestión que ya está declarado.

4.6.2. Atributo inexistente

En caso de que se haga uso de una variable de instancia que no haya sido declarada en ningún contexto previo ni clase de la cual se hereda, se mostrará el siguiente mensaje de error:

```
[Error Semantico] Error en la linea XX
[Error Semantico] La variable de instancia '<ATR>' no existe.
```

Donde *ATR* es el nombre del atributo utilizado pero que no existe.

4.6.3. Atributo no visible

Similar al caso anterior, pero en éste caso ocurre cuando la variable de instancia existe pero no es visible por la clase actual. Esto se da cuando una clase A tiene una variable de instancia privada, y una clase B que hereda de A quiere hacer uso de ésta variable. El mensaje de error que mostrará el compilador es:

```
[Error Semantico] Error en la linea XX
[Error Semantico] La variable de instancia '<ATR>' no es visible.
```

Donde *ATR* es el nombre del atributo que no es visible.

4.6.4. Atributo ya nombrado

Si se quisiera declarar un atributo cuyo nombre ya está siendo utilizado por un método, se mostrará el siguiente error por pantalla:

```
[Error Semantico] Error en la linea XX
[Error Semantico] El nombre de atributo '<ATR>' es el mismo que
el de algun metodo o el de clase.
```

Donde *ATR* es el nombre del atributo que ya está en uso.

4.6.5. Clase Declarada

Para el caso de que se declare una clase más de una vez en un mismo archivo de código fuente, el compilador reportará el siguiente mensaje:

```
[Error Semantico] Error en la linea XX
[Error Semantico] La clase '<NOM>' ya esta declarada.
```

Donde *NOM* es el nombre de la clase ya declarada.

4.6.6. Clase No Declarada

Cuando se utiliza un tipo clase en un atributo, variable local o parámetro, y dicha clase no fue declarada, el compilador mostrará el error:

```
[Error Semantico] Error en la linea XX
[Error Semantico] La clase '<NOM>' no esta declarada.
```

Donde *NOM* es el nombre de la clase utilizada pero no declarada.

4.6.7. Constructor duplicado

Si un constructor de una clase es declarado más de una vez, se mostrará por pantalla el siguiente mensaje:

```
[Error Semantico] Error en la linea XX
[Error Semantico] El constructor '<NOM>' ya esta declarado.
```

Donde *NOM* es el nombre del constructor ya declarado.

4.6.8. Constructor mal nombrado

Si el nombre de un constructor no corresponde con el nombre de la clase (esto es, tienen diferentes nombres), se mostrará el mensaje de error:

```
[Error Semantico] Error en la linea XX
[Error Semantico] El nombre del constructor '<NOM>' no corresponde
al nombre de clase.
```

Donde *NOM* es el nombre del constructor en conflicto.

4.6.9. Expresion unaria mal formada

En caso de que un operador unario se utilice con un tipo que no corresponda a dicho operador (por ejemplo, + ó − con `int`, ó `!` con `boolean`), se informará el error por pantalla.

```
[Error Semantico] Error en la linea XX
[Error Semantico] El operador unario <OP> solo permite <Permitidos>.
```

Donde *OP* es el operador en cuestión, mientras que *Permitidos* corresponde al tipo que permite dicho operador.

4.6.10. Expresion binaria mal formada

Análogo al caso anterior, si una expresión binaria tiene dos subexpresiones de tipos que no conformen o que el operador no esté definido para dichos tipos, se mostrará el error por pantalla:

```
[Error Semantico] Error en la linea XX
[Error Semantico] El operador binario <OP> solo permite <Permitidos>.
```

Donde *OP* es el operador en cuestión, mientras que *Permitidos* corresponde a los tipos que permite dicho operador.

4.6.11. Expresion primaria parentizada con tipo primitivo

Para las expresiones primarias parentizadas, se espera que la expresión contenida dentro del paréntesis no sea de tipo primitivo. En caso de existir dicha situación se reportará un error:

```
[Error Semantico] Error en la linea XX
[Error Semantico] La expresion parentizada no puede ser de un tipo primitivo.
```

4.6.12. Falta `main`

Si ninguna de las clases declaradas en el código fuente posee un `main` que sea estático, con tipo de retorno `void` y sin argumentos, se reportará este error, ya que alguna de las clases debe poseer este método.

```
[Error Semantico] Error en la linea XX
[Error Semantico] Alguna clase debe tener el metodo estatico
'main' (sin parametros)
```

Recordar que según la especificación de MINIJAVA alguna de las clases debe poseer dicho método `main`. Esta situación se ha catalogado como un error semántico y no sintáctico.

4.6.13. Herencia circular

En caso de que exista herencia circular entre las clases se repotará dicho error. Existe herencia circular cuando una clase A tiene de ancestro a una clase B, mientras que al mismo tiempo la clase B tiene de ancestro a la clase A. El mensaje mostrado por pantalla será:

```
[Error Semantico] Error en la linea XX
[Error Semantico] La clase '<NOM>' tiene herencia circular.
```

Donde *NOM* es el nombre de la clase que posee el conflicto de herencia circular.

4.6.14. Herencia propia

Análogo al caso anterior, una clase no puede heredar de sí misma. En dicho caso se reportará un error:

```
[Error Semantico] Error en la linea XX
[Error Semantico] La clase '<NOM>' no puede heredar de si misma.
```

Donde *NOM* es el nombre de la clase que posee el conflicto de herencia propia.

4.6.15. Método duplicado

Si se declara un métdo cuyo nombre ya está en uso se informará el error:

```
[Error Semantico] Error en la linea XX
[Error Semantico] El metodo '<NOM>' ya esta declarado.
```

Donde *NOM* es el nombre del método que ya ha sido declarado.

4.6.16. Método inexistente

Si se hiciera una invocación a un método de clase que no existe, se mostrará el siguiente error:

```
[Error Semantico] Error en la linea XX
[Error Semantico] El metodo '<Metodo>' no existe para la clase '<Clase>'.
```

Donde *Metodo* es el nombre del método invocado sobre la clase *Clase*.

4.6.17. Método mal heredado

Para las redifiniciones de los métodos en MINIJAVA, es necesario que los métodos posean la misma signatura que su ancestro. Esto es, que la forma, el tipo de retorno, y la cantidad y tipo de los parámetros coincidan. En caso de no suceder esto, se mostrará el siguiente mensaje:

```
[Error Semantico] Error en la linea XX
[Error Semantico] El metodo '<NOM>' no posee la misma signatura que
su ancestro.
```

Donde *NOM* es el nombre del método en conflicto.

4.6.18. Parámetro duplicado

En el caso de que se utilice un mismo nombre de identificador de variable en una lista de parámetros formales, se reportará dicho error:

```
[Error Semantico] Error en la linea XX
[Error Semantico] Ya existe un parametro con el nombre '<NOM>'.
```

Donde *NOM* es el nombre del parámetro duplicado.

4.6.19. Parámetro no conforma

Si el tipo de la expresión de un parámetro actual no conforma con el tipo del parámetro formal se informará el mensaje de error:

```
[Error Semantico] Error en la linea XX
[Error Semantico] Un parametro actual en la llamada del metodo '<NOM>'
no conforma con el formal
```

Donde *NOM* es el nombre del método donde no conforman los parámetros.

4.6.20. Parámetros no coinciden

Si se realiza una invocación a un método o constructor cuya cantidad de parámetros es incorrecta se mostrará el siguiente mensaje:

```
[Error Semantico] Error en la linea XX
[Error Semantico] La cantidad de parametros no coincide.
```

4.6.21. Receptor de primitivo o `void`

Cuando se realiza una llamada encadenada, ya sea de identificadores o de métodos, se espera que el tipo receptor sea de tipo clase. Es decir, un receptor no debe ser por ejemplo de tipo `int`, `boolean`, o cualquier tipo primitivo. Si existiera este error, se mostraría por pantalla de la siguiente forma:

```
[Error Semantico] Error en la linea XX
[Error Semantico] El receptor de la llamada debe ser
un tipo clase (y no tipo primitivo)
```

Además, si el tipo receptor de la llamada encadenada fuera de tipo `void`, se informaría un error similar:

```
[Error Semantico] Error en la linea XX
[Error Semantico] El receptor de la llamada debe ser
un tipo clase (y no tipo void)
```

4.6.22. Tipo inválido

Si durante el proceso de declaración de variables locales se utilizara un tipo clase que no ha sido declarado en ningún lugar del código fuente, entonces se informaría dicho error:

```
[Error Semantico] Error en la linea XX
[Error Semantico] El tipo <Clase> no corresponde a un tipo valido
```

Junto con este caso se puede encontrar un mensaje en caso de que una expresión de `casting` u `instanceof` contengan un tipo que no haya sido declarado. En dicho caso el mensaje es similar:

```
[Error Semantico] Error en la linea XX
[Error Semantico] El tipo <Clase> es invalido
```

Donde *Clase* es el nombre del tipo clase utilizado que no existe (es inválido).

4.6.23. Tipos no conforman

En ciertas ocasiones puede producirse que dos tipos no conformen, como por ejemplo el caso del `instanceof` o del operador de comparación ==. Dichos casos de no conformidad corresponden a errores semánticos. La forma en la que se reportará este tipo de error es de la siguiente manera:

```
[Error Semantico] Error en la linea XX
[Error Semantico] El tipo '<ClaseA>' no conforma con '<ClaseB>'.
```

Donde *ClaseA* y *ClaseB* son los nombres de las clases en conflicto.

4.6.24. Otras inconformidades de tipo

Además del ítem del punto anterior, existen otros errores de inconformidad de tipos. Por ejemplo en una asignación, si los tipos no conforman se mostrará el siguiente mensaje:

```
[Error Semantico] Error en la linea XX
[Error Semantico] Asignacion incorrecta. No se puede asignar algo
de tipo <ClaseA> a algo de tipo <ClaseB>
```

Donde *ClaseA* y *ClaseB* son los nombres de las clases en conflicto.
También puede suceder que el tipo de retorno de una expresión dentro de un `return` no conforme con el tipo de retorno del método. Para dicho caso el mensaje de error será este:

```
[Error Semantico] Error en la linea XX
[Error Semantico] El tipo de retorno de la expresion no conforma
con el tipo de retorno del metodo
```

4.6.25. Variable local duplicada

El error para informar que se está declarando una variable local cuyo nombre ya está en uso es el siguiente:

```
[Error Semantico] Error en la linea XX
[Error Semantico] La variable local '<VAR>' ya fue declarada
```

Donde *VAR* es el nombre de la variable local que ya ha sido declarada.

4.6.26. Sentencia no termina en llamada

Toda sentencia de llamada (ya sea simple o encadenada) debe finalizar con un nodo de llamada. En caso de que esto no sucediera, el mensaje que se mostrará por pantalla es el siguiente:

```
[Error Semantico] Error en la linea XX
[Error Semantico] La sentencia no termina en una llamada
```

4.6.27. Expresión no lógica en condición

Para los casos del `if` y `while`, se requiere que la expresión que conforma la condición sea de tipo lógico. Esto es, que el resultado de evaluar toda la expresión devuelva un tipo `boolean`. En caso de no suceder esto, se informará el siguiente error:

```
[Error Semantico] Error en la linea XX
[Error Semantico] La condicion del If/While no es una expresion logica
```

Dependiendo si se tratara de un `if` o `while`.

4.6.28. Constructor con `return`

Un caso particular existe con los constructores de clase, los cuales no pueden contener sentencia return. De esta forma, en caso de que un constructor contenga return (ya sea con o sin expresión), se mostrará el siguiente mensaje:

```
[Error Semantico] Error en la linea XX
[Error Semantico] El constructor no puede contener un return
```

4.6.29. `return` con expresión en método `void`

En los métodos que tienen tipo de retorno `void` se permite utilizar la sentencia `return` con expresión vacía. Sin embargo cuando la expresión no es vacía se mostrará el siguiente error:

```
[Error Semantico] Error en la linea XX
[Error Semantico] El metodo es void y no puede contener
un return con una expresion
```

4.6.30. `return` vacío

Similar al caso anterior, pero de la forma contraria. Los métodos que tienen tipo de retorno no pueden contener un return vacío, por lo tanto se muestra el siguiente mensaje de error:

```
[Error Semantico] Error en la linea XX
[Error Semantico] El metodo no puede contener un return vacio
```

4.6.31. Llamadas a expresiones de tipo `void`

Dentro de las expresiones se pueden realizar llamadas a métodos. Se espera que el tipo de retorno de dichos métodos sean tipos válidos (por ejemplo `boolean`, `int`, etc.). En el caso de que el método llamado tenga tipo de retorno `void`, la expresión no puede tomar ningún valor, por lo que se mostrará un error.

```
[Error Semantico] Error en la linea XX
[Error Semantico] La expresion debe devolver un tipo valido.
El metodo llamado es void
```

4.6.32. Asignación incorrecta

Para la sentencia de asignación, en caso de que el lado izquierdo de la misma no termine en una variable se reportará un error:

```
[Error Semantico] Error en la linea XX
[Error Semantico] Asignacion incorrecta. El lado izquierdo
debe terminar en una variable.
```

Por otra parte, también se pide que el lado izquierdo de una asignación no termine en una llamada. En dicho caso, surgirá el siguiente mensaje:

```
[Error Semantico] Error en la linea XX
[Error Semantico] Asignacion incorrecta. El lado izquierdo
no puede terminar en una llamada.
```

4.7. Errores de generación de código intermedio

En el proceso de compilación no existen los errores a nivel generación de código intermedio. Los únicos errores que pueden surgir durante la generación de código intermedio son los de escritura del archivo de salida, mencionado anteriormente.