

Compilador Fase 1

Análisis y Diseño



01 de Septiembre de 2014

Universidad Galileo

Hugo Alberto Garrido Carranza 20072115

Contenido

[Suposiciones 2](#_Toc397375617)

[Resumen del Diseño 2](#_Toc397375618)

[Scanner 3](#_Toc397375619)

[Parser 3](#_Toc397375620)

[Descripción de Diseño Top-Level 4](#_Toc397375621)

# Suposiciones

1. En la clase ErrorHandler.java la escritura a un archivo genérico ErrorHandler.txt para posteriormente poder implementar un log de errores que se tengan en el compilador.
2. El archivo de entrada debe de ser extensión .decaf, el archivo de salida que se generara será .s.
3. La opción –opt únicamente funcionara cuando el –target sea codegen esto debido a que únicamente pueden existir optimizaciones cuando se genere código.
4. Al existir un error el target o en el debug por ejemplo se desplegará el error en pantalla, sin embargo la ejecución continuara, la única falla que hace que la ejecución no continúe es que el archivo no existe sin embargo esta falla la soporta el CLI, esto será asi hasta que las especificaciones del proyecto mencionen lo contrario, los reportes únicamente se reportaran pero no detendrán la ejecución del proyecto.
5. Al no especificarse en el proyecto se decidió imprimir lo mismo en el archivo como en el debug en el scan se imprimirán el token, el valor que hizo match y la línea en la que se encuentra este token en el archivo mientras que en el parser se imprimirán las diferentes reglas de parseo con las que hizo match.
6. Actualmente en la clase de errores únicamente se escribirá al archivo ErrorHandler.txt sin embargo esto es parametrizable por ejemplo si en un futuro se quiere que escriba un archivo de errores para cada una de las fases la clase lo puede hacer.
7. Si se escribe –debug semantic por ejemplo y el target solo llega al scan no se desplegara el debugging ya que no se puede debuggear algo que no se está instanciado.
8. Al momento de instanciar el parser no se muestra el resultado del scan en el archivo esto se asumió debido a que al momento de llegar a la generación de código por ejemplo no vamos a querer que cuando ese sea el target en el archivo aparezca el resultado de todas las fases anteriores cuando en realidad lo que nos interesaría seria el programa final.

# Resumen del Diseño

Se realizó una estructura de directorios descrita en el proyecto en donde cada clase es perteneciente al paquete (directorio) que está en la raíz.

Básicamente se creó una estructura hash en la clase Compiler.java la cual tendrá de keys cada una de las options que puede generar el CLI.

Como primer paso se parsea todo lo ingresado por el usuario, se ingresa a la hash table y se realizan algunas verificaciones para realizar el análisis si lo ingresado es válido o no.

Previamente ya se había validado que el archivo de ingreso existiera.

Posterior al almacenamiento de cada una de las opciones se recorre la hash table buscando las keys de las opciones esperadas y se generan cada una de las instancias de las clases que los target, degub y opt necesiten para finalizar satisfactoriamente las instrucciones ingresadas por el usuario.

Si existe algún error en tiempo de ejecución se reporta a la clase ErrorHandler.java para que se almacene en un log de errores y proporcione en consola un System.err.println del error que ocurrió en tiempo de ejecución.

A medida que se recorren cada una de las opciones la clase Compiler.java verifica que los parámetros sean los esperados si no es así reporta un error a la clase ErrorHandler.java.

### Scanner

1. Todos los tokens definidos para decaf se encuentran en el archivo decaf.g dentro del directorio scanner.
2. Al momento de ejecutar esto con las especificaciones de antlr4 esto genera el archivo decaf.java y se genera todo el token vocab en el archivo Decaf.tokens.
3. Dentro de la clase Scanner.java se recorren todos el archivo y se imprimen los tokens en el archivo o en pantalla dependiendo si el debug esta en true y si el print que se le envía de flag al archivo esta en true.
4. Si el flag no esta en true y el target es posterior al scan se instancia un objeto de esta clase para poder optar al nombre del archivo por ejemplo y ejecutarlo sin mostrar nada en pantalla ni en el archivo, funciona como una ejecución del scan en modo silence.

### Parser

1. El parser toma como primera instancia los tokens generados en el scan y se importan en el members de la gramatica.
2. Por cada regla de parseo que hace match se inserta el string de esa regla de producción en dos stacks, uno para el debug y otro para para la impresión en el archivo.
3. Dentro de la clase CC4Parser.java se ejecuta el parseo mediante las reglas de producción en este momento ambos stacks ya están llenos con las diferentes reglas de producción que hicieron match al momento de la ejecución.
4. Si el print está en true y el debug está en true se vacían ambos stacks y se imprimen en pantalla y el archivo sin imprimir el scanner tal como se mencionó previamente.
5. Las gramática del parser determina errores a nivel de la misma reportándolos mediante la clase ErroHandler, estos errores se imprimirán en pantalla únicamente como advertencia no interrumpirán en flujo de la ejecución esta se interrumpirá únicamente si la clase Compiler no es capaz de recuperarse de algún error.

# Descripción de Diseño Top-Level

1. Si el –target es codegen se valida el –opt si existe en la hash table para poder generar el algebraic o el constant folding en su defecto esto debido a que no puede existir una optimización si no existe una generación de código.
2. La clase ErrorHandler.java escribirá en un archivo de errores al momento que exista algún tipo de excepciones, esto debido a que en un futuro es probable que se quieran personalizar los errores con un archivo o llevar un log de errores por fase.
3. Se realizaron modificaciones a la clase compiler respecto al proyecto 0 en esta fase ya no se tendrán impresiones en todas las fases únicamente en la fase que especifique el target si esta fase no se especifica por default según las especificaciones del proyecto se ejecutara hasta el parser.
4. Fases de ejecución de la clase compiler
   1. Verifica que exista el archivo de entrada.
   2. Si no existe un archivo de entrada únicamente muestra la ayuda.
   3. Si el archivo de entrada existe y no hay target únicamente se ejecuta el compilador hasta el parser.
   4. Por cada una de las opciones que aparece especificada en el cmd al momento de la ejecución se realiza un put a una hash table.
   5. Posteriormente se realiza un pop con cada una de las opciones y dependiendo de lo que se encuentre se ejecuta una instancia de cada una de las fases que fueron especificadas.
   6. Existen 2 Banderas por cada ejecución 1 que se utiliza para verificar el debug de cada una de las fases y otra que se especifica para verificar si la fase debe realizar print en el archivo o únicamente instanciarse sin hacer print al archivo.
   7. La clase ErrorHandler maneja tres tipos de errores, los genéricos de flujo de ejecución y manejo de excepciones, los generados en el scanner y los generados por el parser.