

Compilador Fase 0

Análisis y Diseño



22 de julio de 2013

Universidad Galileo

Hugo Alberto Garrido Carranza 20072115

Contenido

[Suposiciones 1](#_Toc362298472)

[Resumen del Diseño 1](#_Toc362298473)

[Descripción de Diseño Top-Level 2](#_Toc362298474)

# 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 y de salida no tienen extensión alguna especificada.
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.
5. Actualmente la clase Debug.java imprime Debuggin y el stage que se le envié como parámetro, eventualmente en posteriores fases ya recibirá un string probablemente más grande de cada una de las fases que se implemente.
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.

# 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.

# 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. La clase Compiler.java se recupera de errores que se encuentren en los target de entrada ejecutando únicamente lo que es válido y desplegando los errores que se encontraron en el string de entrada.
4. Al no estar especificado la extensión del archivo de entrada o de salida se asume que puede ser cualquier tipo de archivo, probablemente esto cambie en las siguientes etapas sin embargo al no existir una especificación clara en el proyecto 0 el compilador funciona con cualquier extensión del archivo de entrada y de salida.
5. Cada una de las clases posee la escritura al archivo de salida y la validación del debug.