**11 Notas sobre la implementación**

Descripción de la implementación realizada.

**11.1. Descripción de archivos**

Enumeración de los archivos con el código fuente de la implementación, y descripción de lo que contiene cada archivo.

Hemos divido el proyecto es 4 paquetes principales:

· Compilador: Contiene las clases relativas al traductor de lenguaje.  Este paquete contiene 3 paquetes:

· Lexico: contiene la clase AnalizadorLexico, que recibe un flujo de entrada y devuelve un ArrayList de Tokens. Este paquete contiene un paquete Tokens con clases que corresponden a cada uno de los tipos de tokens que maneja el Analizador Lexico.

· TablaSimbolos: Contiene una clase que gestionará las distintas operaciones de la tabla de símbolos. También contiene una clase auxiliar que gestiona los datos que se asignan a cada identificador, en nuestro caso el tipo y la dirección de memoria de cada variable.

· Traductor: Contiene las clases necesarias para traducir un ArrayList de Tokens procedente del analizador léxico a un ArrayList de Objetos que contendrá el código binario. Como hemos implementado dos tipos de traducción (a código P y a código J) tenemos unas clases comunes a ambos que son ErrorTraductor (captura los errores del traductor), Traductor y TraductorCodDual que gestionan la traducción.

· Traductor Codigo P: El código binario generado será ejecutado en nuestro intérprete Pila. La clase del codigo P es Codigo.java.

· Traductor Codigo J: Contiene las clases necesarias para traducir un ArrayList de Tokens procedente del analizador léxico a un ArrayList de Objetos que contendrá el código binario que será ejecutado en la Máquina Virtual de Java. La clase del codigo J es CodigoJVM

· Interfaz: Contiene las interfaces utilizadas de cara al usuario. Como hemos implementado dos programas separados (compilador e intérprete) tenemos dos interfaces:

· Compilador: Contiene la interfaz del compilador. Esta interfaz da la opción de introducir el código en la propia interfaz o cargarla desde un fichero, una vez cargado puedes compilar y ver el código pila o ejecutar (en cuyo caso compilará y luego ejecutará el programa) También nos ofrece ejecutar el código en modo Traza (mostrando el contenido de la pila y la memoria en cada instrucción además de las entradas/salidas del programa) o en modo Normal (mostrando unicamente las entradas/salidas del programa)

· Pila: Contiene una interfaz que hemos utilizado para probar el intérprete a pila. Este panel hace de intermediario entre el bytecode del lenguaje a pila y el explicado en clase (con sentencias alfanumericas como "apila 3" o "suma"). Al decompilar un archivo en bytecode este se mostrará como cadenas alfanumericas. Al compilar, el texto escrito sera traducido a lenguaje de pila, siempre y cuando su sintáxis sea correcta

· org: Contiene las clases de la librería BCEL (http://jakarta.apache.org/bcel/) que se utilizan para generar el código Java y que la Máquina Virtual de Java sea capaz de interpretarlo.

· Pila: Contiene el intérprete encargado de simular la ejecución del código . Como tenemos dos tipos de código (código P y código J) este paquete contiene dos paquetes:

·  Intérprete: Se encarga de ejecutar el código P generado por el compilador de código P. Contiene 3 paquetes que se encargan de gestionar tanto los datos, como las instrucciones y excepciones que pueden surgir en el código y 3 clases que son las principales (EscritorPila, Interprete y LectorPila) que gestionan la entrada/salida de la ejecución y la propia ejecución

**11.2. Otras notas**

Diagramas de clase UML describiendo la arquitectura del sistema.

**Conclusiones**

Qué se ha conseguido y qué se ha dejado pendiente para más adelante.*.*

**Referencias bibliográficas**

Libros, artículos y otras fuentes de información utilizadas (por ejemplo páginas web).

<http://jakarta.apache.org/bcel/>