**Documentación**

El objetivo de este proyecto es poder realizar un compilador del lenguaje Decaf en Java. Este documento contiene la estructura de los directorios creados hasta la presente fase (que en este caso es **irt**). También, contiene una descripción de qué realiza cada clase creada, junto con las librerías adicionales empleadas. Concluye con un set de instrucciones para correr el compilador desde el CMD o una línea de comandos similar. La fase de esta segunda entrega abarca solamente el irt funcional, conectado con el CLI del compilador.

**Requisitos:**

* Contar con Java instalado. Entiéndase también, tener instalado el JDK.
* Instalar Graphviz, habilitarlo en el PATH del usuario y agregar el PATH a las variables de ambiente. Es necesario reiniciar la computadora para guardar los cambios.

**Estructura de Directorios:**

/class

/bin

/lib

java\_cup.jar.

java-cup-11a.jar

JFlex.jar

programa3.txt

programa3e.txt

README.md

/src

Compiler.java

/scanner

ScannerRunner.java

GenScanner.java

Scan.java

Lexer.flex

Test.java

Tokens.java

/parser

GenParser.java

GenScannerCup.java

LexerCup.flex

ScannerCup.java

Sintax.cup

Parser.java

ParserRunner.java

sym.java

Test.java

Nodo.java

/ast

/graficas

arbol.dot

arbol.png

Ast.java

/semantic

Error.java

Simbolo.java

TablaParametros.java

Semantic.java

/irt

Irt.java

Fila\_m.java

Fila\_p.java

Fila\_s.java

NodoIrt.java

Tabla\_m.java

Tabla\_p.java

Tabla\_s.java

**Descripción Elementos del Directorio Scanner:**

* ScannerRunner.java: clase generada por JFlex que contiene todas las instrucciones para generar los tokens
* GenScanner.java: clase que guarda lo generado por el scanner en un archivo .txt
* Scan.java: clase generada por el Lexer que contiene las instrucciones para hacer el scan
* Lexer.flex: archivo de definición de las expresiones regulares de Decaf
* Test.java: clase para hacer pruebas internas del scanner
* Tokens.java: clase que contiene las definiciones de los tokens a utilizar

**Descripción Elementos del Directorio Parser:**

* GenParser.java: es el que genera Sintax.java, es decir, hace el parse
* GenScanner.java: genera ScannerCup.java
* LexerCup.flex: es donde se define el léxico del lenguaje, adaptado para Cu
* Sintax.cup: es la definición de la gramática
* sym.java: generado con Sintax y contiene todos los símbolos
* Test.java: archivo para probar un .txt con un ejemplo de un programa en Decaf
* ScannerCup.java: es el archivo generado por el LexerCup que interpreta Parser.java
* ParserRunner.java: es el archivo que ejecuta el parseo
* Parser.java: archivo que contiene el set de la gramática del lenguaje generado por LexerCup
* Nodo.java: archivo que contiene los nodos que se crean para ser procesados por el AST, junto con sus procedimientos.

**Descripción Elementos del Directorio AST:**

* Gráficas: carpeta que contiene el archivo .dot para pruebas pasadas y árboles generados para pruebas pasadas
* Árbol.dot: archivo .dot que contiene los nodos conectados del árbol del código y lo dibuja
* Árbol.png: imagen del archivo .dot generado
* Árbol1.png: imagen del archivo .dot generado para otro código
* Ast.java: archivo que crea el árbol del código con la información de la etapa pasada

**Descripción Elementos del Directorio Semantic:**

* Error.java: archivo para manejar los errores detectados durante el análisis semántico
* Semantic.java: archivo que analiza que la gramática del código detectado se encuentre acorde a lo definido por el lenguaje
* Simbol.java: archivo donde se definió los arrays y algunas estructuras necesarias para el análisis semántico
* TablaParámetros: archivo para imprimir los parámetros que se están analizando y se almacenaron.

**Descripción Elementos del Directorio IRT:**

* Fila\_m: clase que contiene la información de las filas que se guarda en la tabla métodos
* Fila\_p: clase que contiene la información de las filas que se guarda en la tabla parámetros
* Fila\_s: clase que contiene la información de las filas que se guarda en la tabla símbolos
* Tabla\_m: clase que contiene la tabla de métodos, compuesta de un array list de Filas\_m y métodos para irla ejecutando, viendo o modificando
* Tabla\_p: clase que contiene la tabla de parámetros, compuesta de un array list de Filas\_m y métodos para irla ejecutando, viendo o modificando
* Tabla\_s: clase que contiene la tabla de símbolos, compuesta de un array list de Filas\_m y métodos para irla ejecutando, viendo o modificando
* Irt.java: es la clase comando que ejecuta la fase
* NodoIrt.java: clase que contiene la instrucción para ejecutar la lista del IRT y que permite crear varias instancias del mismo para crear el nodo

Compiler.java es el CLI de este proyecto. Esta clase es la que controla todos los elementos que conforman el compilador y es la única forma de correr de forma funcional y completa todo el compilador.

\*Definiciones puestas hasta el punto funcional del proyecto, las demás partes se irán agregando conforme el proyecto vaya avanzando.

**Instrucciones para correr el compilador:**

* Desde el CMD acceder a la dirección *~/class/compiler*
* Correr el comando:

*Makefile.bat*

Makefile prepara Compiler.java para ser inicializado

**SCANNER**

* Para correr el scanner correr el siguiente comando:

*java Compiler programa3.txt -target scan*

Archivo 2:

*java Compiler programa3e.txt -target scan*

* Para correr el debuger hasta el scanner correr el siguiente comando:

*java Compiler programa3.txt -debug scan*

Archivo 2:

*java Compiler programa3e.txt -debug scan*

**PARSER**

* Para correr hasta parser correr el siguiente comando:

*java Compiler programa3.txt -target parse*

Archivo 2:

*java Compiler programa3e.txt -target parse*

* Para correr el debuger hasta el parser correr el siguiente comando:

*java Compiler programa3.txt -debug parse*

Archivo 2:

*java Compiler programa3e.txt -debug parse*

**AST**

* Para correr hasta *ast* correr el siguiente comando:

*java Compiler programa3.txt -target ast*

Archivo 2:

*java Compiler programa3e.txt -target ast*

* Para correr el debuger hasta el ast correr el siguiente comando:

*java Compiler programa3.txt -debug ast*

Archivo 2:

*java Compiler programa3e.txt -debug ast*

**SEMANTIC**

* Para correr hasta semantic correr el siguiente comando:

*java Compiler programa3.txt -target semantic*

Archivo 2:

*java Compiler programa3e.txt -target semantic*

* Para correr el debuger hasta el semantic correr el siguiente comando:

*java Compiler programa3.txt -debug semantic*

Archivo 2:

*java Compiler programa3e.txt -debug semantic*

**IRT**

* Para correr hasta irt correr el siguiente comando:

*java Compiler programa3.txt -target irt*

Archivo 2:

*java Compiler programa3e.txt -target irt*

* Para correr el debuger hasta el irt correr el siguiente comando:

*java Compiler programa3.txt -debug irt*

Archivo 2:

*java Compiler programa3e.txt -debug irt*

NOTA: si desea correr otro archivo, cambiar *archivo.txt* o *test2.txt* de cualquier línea de ejecución. Puede acceder a los archivos .txt y cambiar el contenido de estos. El nuevo archivo que desee correr debe ser .txt