**Universidad de Guadalajara**

**Centro Universitario de Ciencias Exactas e Ingenierías**

**Taller de programación de sistemas**

**Reporte # 1**

**ALUMNO:**

Oscar Alberto Palomares López Sección D15

Quiñones Nunñez Karla Patricia Sección D07

**Descripción de las variables y funciones utilizadas**

Para esta primera práctica empleamos 2 clases

* **EnsambladorHC12Raw:**  fd, abierto, bandera, bandera2, bandera3, i.

En esta clase vienen las funciones principales que analizan el texto archivo P1ASM y crean los archivos de error en caso de errores de la práctica1.

* **EnsambladorHC12UI:** temporal, letra, letra2, letra3, archivo[20], etiqueta[20], codop[15], operando[50].

Para esta primera práctica empleamos 2 clases

**Variables de la clase :**

|  |
| --- |
| private String[] contenidoDeArchivo; |
| private String contenidoProcesado; |
| private String contenidoDeArchivoTxt; |
| private String codop; |
| private String operando; |
| private String etiqueta; |
| private final String FOLDER\_ERRORS = "errores"; |
| private String FILE\_NAME; |
| private String FOLDER\_NAME; |

* **Private String[] ContenidoDeArchivo:**

Contiene el contenido del texto del archivo P1ASM.txt separado en palabras cada vez que se lee una linea nueva.

* **Private String ContenidoDeArchivoTxt:**

Contiene el contenido del texto del archivo P1ASM.txt en formato String.

**Variables:**

* **private String contenidoProcesado:**

**Contiene el resultado final de cada línea del archivo P1ASM.txt y en caso que haya errores esta variable también los contendrá**

**Variables:**

* **private String codop:**  contiene el valor de un token codop en caso de ser encontrado.

**Variables:**

* **private String operando:**  contiene el valor de un token operando en caso de ser encontrado.
* **private String etiqueta:**  contiene el valor de un token etiqueta en caso de ser encontrado.
* **private final String FOLDER\_ERRORS:**  contiene el nombre de la carpteta en la que irán los errores encontrados.
* **private final String FILE\_NAME:**  contiene el nombre y la ruta del archivo P1ASM.TXT.

**Constructor:**

|  |
| --- |
| public EnsambladorHC12Raw(String FILE\_NAME, String FOLDER\_NAME) |
| { |
| this.setFILE\_NAME(FILE\_NAME); |
| this.setFOLDER\_NAME(FOLDER\_NAME); |
| new File(getFOLDER\_ERRORS()).mkdirs(); |
| } |
|  |

* **Public EnsambladorHC12Raw(String FILE\_NAME, String FOLDER\_NAME) :**

Aquí se asigna la ruta del archivo y el nombre de la carpeta de los errores.

**Métodos:**

* **private String obtenerResultados() :**  fd, abierto, bandera, bandera2, bandera3, i.

Cuando se manda a.llamar checa el contenido del archivo P1ASM.txt en caso que no allá nada en el archivo regresa una String llamada 's' y crea un archivo de error con el método writeError() con esta String describiendo este error. En caso que si exista el contenido de archivo se separa por lineas de texto usando el metodo getContenidoDeArchivo() y posteriormente analiza linea por linea estas.

Después se crea un arreglo de las palabras del tipo String[] llamado 'palabras' de la línea del archivo P1ASM.TXT para así evitar analizar cualquier tipo de espacio u tabulación no útil usando el método this.separarEnPalabras().

Después se analiza en caso que la linea sea la final decide si es un END y agrega a la String anteriormente mencionada este END, y en caso que sea la última línea y no tenga un END agrega a 's' este error una vez que haya analizado los tokens en las siguientes validaciones.

En caso que la linea esté vacía agrega a la String anteriormente mencionada este tipo de error.

Sino si en caso que analice que sea un comentario agrega a la String anteriormente mencionada este Comentario.

Sino este método concluye que será un token de tipo etiqueta, codop u operando y los pasa a analizar siendo que, que analice otro metodo llamado hasETIQUETA( ) que analiza que el primer carácter de la linea sea un carácter pasa a analizar si esta etiqueta contiene un error con el método validarETIQUETA( ) y en caso que sea válida regresa una String con la etiqueta y si tiene algún error concatena ese tipo de error a la String y finalmente la String regresada se manda a la variable etiqueta de la clase llamando setEtiqueta(); posteriormente se analiza si la etiqueta contiene un error con el método writeError() y si lo contiene se agrega a la variable 's' y se crea un archivo de error en la carpeta de errores con el contenido de este error, y si no contenía un error sólo se concatena la etiqueta como está a 's'. Después de hacer esta validación ahora valida en caso que haya más palabras en el arreglo 'palabras' pasa a analizar con el método analizarLinea() si las siguientes palabras son CODOPS u OPERANDOS, y finalmente regresa una String que se concatena a 's' con los resultados del análisis posteriormente se describirá más de este método. Sino en caso que no más palabras en el arreglo 'palabras[]' y como hubo una etiqueta confirmada se pasa a crear y concatenar el error de que no existe mínimo un codop en esa línea y se concatenan el codop y el operando como null. Ahora regresando a la validación de que si fue ETIQUETA la el inicio de la línea se válida en caso que no haya sido se procede a concatenar a 's' que la etiqueta es nula y que se procede a analizar lo que resta de las palabras en el arreglo 'palabras[]', usando de nuevo el método analizarLinea() y concatenando a 's' los resultados. Ahora regresando a la validacion que la línea no haya sido comentario finalmente se analiza en caso que hubiera sido la línea final y como no había sido un END se pasa a concatenar este error que no era una línea que finalizara con END acabando así con todas las validaciones y regresando el contenido de 's' que será el resultado de todas las validaciones de las líneas.

* **Private String readFile(String path, Charset encoding)**  Regresa una String con el contenido del archivo, se usa en el constructor para que la variable contenidoDeArchivoTxt esté inicializada.
* **public String[] separarEnPalabras(String contenido) { :**  Regresa un arreglo de String con el método de Java contenido.trim().split("\\s++"); de la variable contenidoDeArchivoTxt, se usa la la inicialización de la variable local del método obtenerResultados() para crear el arreglo de palabras sin espacios inútiles siendo que la variable local de este método contenido será la línea y el split("\\s++") será la lógica que cree las palabras sin espacios inútiles.
* **public String analizarLinea(int LINE\_NUMBER, String[] palabras) :**  Analiza las palabras de la línea desde el arreglo local palabras llamando a los metodos de validación validaCODOP() y validaOPERANDO() que posteriormente se platicarán y en caso que haya errores concatenarlos a una String local que se regresara y en caso que no haya concatenar estos tokens sin errores a esta String. La lógica de este método es que siempre hay un CODOP y siempre agrega este y lo concatena después lo válida y en caso que haya errores también los concatena, posteriormente si hay más palabras en el arreglo analiza si hay más de 1 después del CODOP si hay más las concatena y las toma como un operando tomando como ejemplo cadenas con comillas etc, posteriormente analiza si es un operando válido (que en realidad en está práctica si hay más palabras después de un operando lo que sea será un operando) lo concatena a la String a regresar. En caso que no haya más palabras depués de un CODOP se concatena un Operando como null a la String local. Finalmente se regresa la String local con el contenido analizado
* **public boolean isComentario(String linea) { :**  Analiza si la linea es un comentario usando el método separarEnPalabras(linea)[0].startsWith(";"); de Java (si empieza con el carácter ';') y regresa verdadero o falso.
* **public boolean hasETIQUETA(String linea ) { :**  Analiza si la linea contiene una etiqueta con el método !(""+linea.charAt(0)).matches("\\s++"); de Java (si la línea en su primer cárater no es un espacio) y regresa verdadero o falso.
* **public String validarOPERANDO(String palabra) :**  Analiza si la línea es un operando(En realidad cualquier cosa es un operando así que regresa siempre una String con el operando, sólo se agregó este método para fines de prácticas futuras).
* **public String validarCODOP(String palabra ) :**  Analiza si la palabra es un CODOP, la lógica es la siguiente, se regresará siempre una String de los resultados que regresen los caso en que si la pálabra es mayor a 5 caracteres se concatenará un error a la String, si no si la palabra no empieza con un carácter alfabético concatenará este tipo de error a la String, si no si la palabra tiene más de 2 caracteres puntos '.' Se concatenará este tipo error a la String, sino si la palabra es mayor a 1 y no contiene algún carácter válido siendo estos sólo alfábeticos y un punto concatena el error de este tipo a la String, si no finalmente regresa la String del CODOP sin errores.
* **public String validarETIQUETA(String palabra )**  Analiza si la palabra es una ETIQUETA, la lógica es la siguiente, se regresará siempre una String de los resultados que regresen los caso en que si la pálabra es mayor a 8 caracteres se concatenará un error a la String, si no si la palabra no empieza con un carácter alfabético concatenará este tipo de error a la String, sino si la palabra es mayor a 1 y no contiene algún carácter válido siendo estos sólo alfánúmericos y el carácter '\_' concatena el error de este tipo a la String, si no finalmente regresa la String del ETIQUETA sin errores.
* **public String writeError(int LINE\_NUMBER, String FILE\_CONTENT) :**  Regresa en una String y en cierto formato el tipo de error que se encontró en alguna validación concatenando también el número de línea en el que fue, y además crea un archivo con esta información en la carpeta de errores.

**Métodos getter y setter:**

* Son los getter y setter de las variables de esta clase.

|  |
| --- |
| public String getCodop() { |

|  |
| --- |
| return codop; |
| } |
|  |
| public void setCodop(String aCodop) { |
| codop = aCodop; |
| } |
|  |
| public String getOperando() { |
| return operando; |
| } |
|  |
| public void setOperando(String aOperando) { |
| operando = aOperando; |
| } |
|  |
| public String getEtiqueta() { |
| return etiqueta; |
| } |
|  |
| public void setEtiqueta(String aEtiqueta) { |
| etiqueta = aEtiqueta; |
| } |
|  |
| public String[] getContenidoDeArchivo() { |
| return contenidoDeArchivo; |
| } |
|  |
| public void setContenidoDeArchivo(String[] aContenidoDeArchivo) { |
| contenidoDeArchivo = aContenidoDeArchivo; |
| } |
|  |
| public String getContenidoDeArchivoTxt() { |
| return contenidoDeArchivoTxt; |
| } |
|  |
| public void setContenidoDeArchivoTxt(String contenidoDeArchivoTxt) { |
| this.contenidoDeArchivoTxt = contenidoDeArchivoTxt; |
| } |
|  |
| public String getContenidoProcesado() { |
| return contenidoProcesado; |
| } |
|  |
|  |
| public void setContenidoProcesado(String contenidoProcesado) { |
| this.contenidoProcesado = contenidoProcesado; |
| } |
|  |
| public String getFILE\_NAME() { |
| return FILE\_NAME; |
| } |
|  |
| public void setFILE\_NAME(String FILE\_NAME) { |
| this.FILE\_NAME = FILE\_NAME; |
| } |
|  |
| public String getFOLDER\_NAME() { |
| return FOLDER\_NAME; |
| } |
|  |
| public void setFOLDER\_NAME(String FOLDER\_NAME) { |
| this.FOLDER\_NAME = FOLDER\_NAME; |
| } |
|  |
| public String getFOLDER\_ERRORS() { |
| return FOLDER\_ERRORS; |

|  |
| --- |
| } |

**Clase public class EnsambladorHC12UI extends javax.swing.JFrame :**

Esta clase contiene la UI en la que se selecciona el archivo P1ASM.txt a analizar, y además contiene el método main que manda a llamar los métodos de la clase EnsambladorHC12Raw para que una vez seleccionado el archivo se analice y en la UI esta clase muestre los resultados del análisis.

**Constructor:**

* **public EnsambladorHC12UI():**  Manda a llamar al método por defecto de Java en Netbeans initComponents() que inicializa los componentes de la UI para usarlos en otros métodos.

**Variables:**

|  |
| --- |
| private javax.swing.JLabel jLabel1; |

|  |
| --- |
| private javax.swing.JLabel jLabel2;  private javax.swing.JLabel jButton1; |
| private javax.swing.JScrollPane jScrollPane1; |
| private javax.swing.JScrollPane jScrollPane2; |
| private javax.swing.JTextArea jTextArea1; |

|  |
| --- |
| private javax.swing.JTextArea jTextArea2; |

* **Variables:**  En el caso de esta clase sólo se usan componentes UI de Java para poder interactuar con ellos, se listan 2 Jlabel para identificar el archivo txt a visualizar y otro llamado resultados para visualizar los resultados del analizador, después 2 textarea donde están los contenidos del archivo txt y de los resultados obetenidos, y finalmente 2 scrollpane para navegar en los textarea.

**Métodos:**

* **initComponents():**  Inicializa los componentes de la UI Java en Netbeands lo crea por defecto al dibujar una UI.
* **public static void main(String args[])**  Una vez que se ejecuta el jar se manda a llamar a este método el cual primero ejecuta una serie de código que Java en Netbeans crea para inicial la UI y posteriormente en un hilo el programador puede instanciar la clase Ensambladorhc12UI para poder hacer uso e interactuar a nivel programación con los componentes de esta usando el evento jButtonActionPerformed cada vez que se elija un archivo.
* **Private void jButtonActionPerformed(Event evt)**  Una vez que se aprieta el botón de Archivo aqui se manda allamar a Ensambladorhc12Raw para poder hacer uso del analizador e interactuar a nivel programación con los componentes de la clase EnsambladorHC12UI, la lógica es Crea un JFIleChooser y si se elige un archivo se carga la ruta y el folder de este en variables String y estas se mandan a un Objeto nuevo del tipo EnsambladorHC12Raw para poder analizar estos archivos, posteriormente inicializa las variables del analizador y se mandan a llamar los contenidos de archivo txt y los resultados para ponerlos en los jtextArea anteriores.