

# **Manual técnico CH-MAQUINA**

Instructivo para programadores  
Sistemas operativos

**DOCENTE :Carlos Hernán Gómez**  
**CURSO :sistemas operativos**



**UNIVERSIDAD NACIONAL DE COLOMBIA SEDE  
MANIZALES  
ADMINISTRACIÓN DE SISTEMAS INFORMÁTICOS**

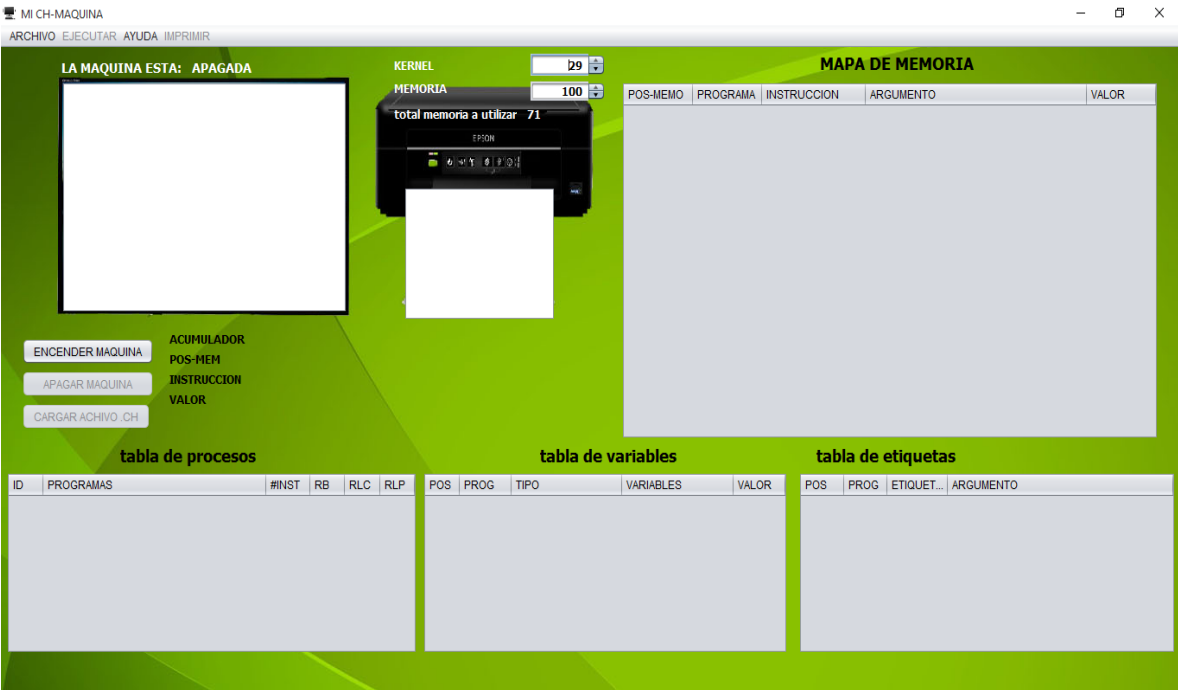
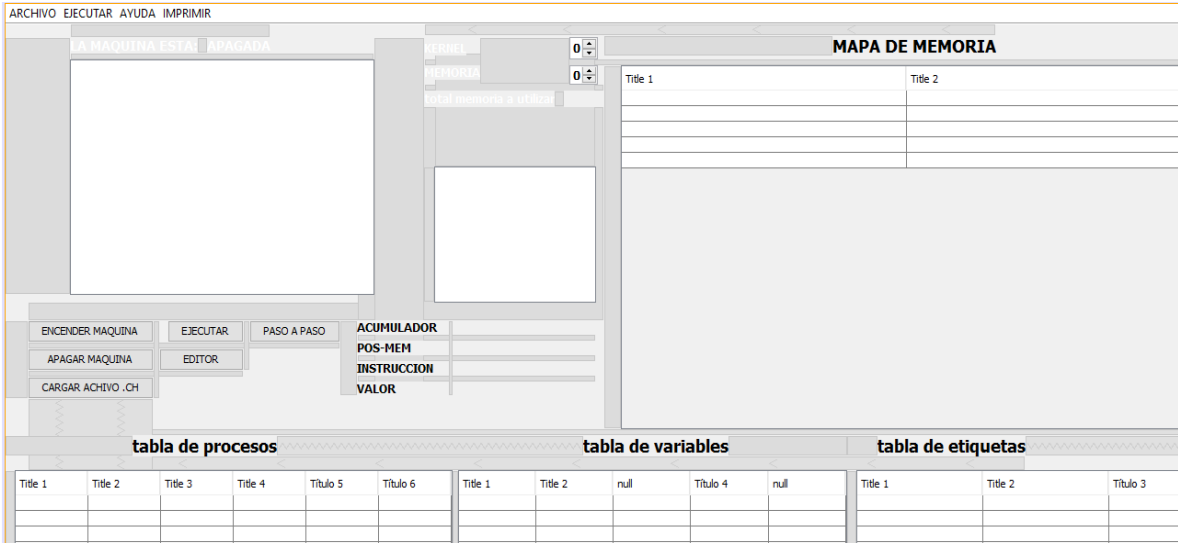
**YEISON AGUIRRE OSORIO COD 913503**  
**Fecha de inicio 5/feb/2016**

# 1. Índice:

<b>1. ÍNDICE:</b> .....	<b>2</b>
<b>1. MUESTRA PLANA DE LA INTERFAZ Y REPRESENTACIÓN FINAL INTERFAZ DE USUARIO</b> .....	<b>4</b>
<b>2. LENGUAJE UTILIZADO EN LA IMPLEMENTACIÓN</b> .....	<b>5</b>
<b>3. INTRODUCCIÓN Y DESCRIPCIÓN DEL PROGRAMA</b> .....	<b>5</b>
<b>4. OBJETO Y ASPECTOS PRINCIPALES DEL PROGRAMA.</b> .....	<b>5</b>
<b>5. QUE HACE EL CH-MÁQUINA Y CUÁL ES SU SINTAXIS.</b> .....	<b>6</b>
Operación Descripción sintaxis: .....	6
Ejecución del programa: .....	8
<b>6. CÓDIGO COMPLETO Y EXPLICACIÓN DE FUNCIONAMIENTO DEL PROGRAMA</b> .....	<b>9</b>
➤ Estructura del proyecto chmaquina:.....	9
➤ Paquete chmaquina clase entrada.java:.....	10
1. Librerías importadas:.....	11
2. Clase entrada .....	12
3. Funciones:.....	19
➤ Función memtotal(): .....	19
➤ Función son():.....	19
➤ Función encender():.....	20
➤ Función temporales(): .....	22
➤ Función apagar(): .....	22
➤ Función cargararchivo(): .....	23
➤ Función actualizar(): .....	24
➤ Función cargue(): .....	40
➤ Función almacene(): .....	40
➤ Función vaya(): .....	41
➤ Función lea():.....	42
➤ Función sume(): .....	42
➤ Función reste():.....	43
➤ Función multiplique(): .....	44
➤ Función divide():.....	45
➤ Función potencia(): .....	46

➤ Función modulo():.....	47
➤ Función concatene(): .....	48
➤ Función elimine():.....	49
➤ Función extraiga():.....	50
➤ Función mostrar(): .....	51
➤ Función imprimir():.....	52
➤ Función ejecutar():.....	53
➤ Función pasoapaso():.....	58
➤ Función initComponets():.....	62
4. eventos por botones:.....	62
➤ evento de botón cargarprogramaActionPerformed(): .....	62
➤ Evento de botón memoriaStateChanged() , kernelStateChanged(): .....	63
➤ Evento de botón encenderActionPerformed(), apagarmaquina2ActionPerformed(), encender2ActionPerformed(), apagarmaquina1ActionPerformed(): .....	64
➤ Evento de botón jMenuItem3ActionPerformed():.....	65
➤ Evento de botón acercadeActionPerformed(); .....	66
➤ Evento de botón encender3ActionPerformed (),encender4ActionPerformed();.....	67
➤ Evento de botón documentacionActionPerformed (); .....	68
➤ Evento de botón botoncargarActionPerformed ();.....	68
➤ Evento de botón ejecutarActionPerformed (); .....	69
➤ Evento de botón editorActionPerformed (); .....	69
➤ Evento de botón pasoapasoActionPerformed (); .....	69
➤ Evento de botón IMPRIActionPerformed ();.....	70
➤ <b>Paquete chmaquina clase ambiente.java:</b> .....	70
1. Librerías importadas:.....	70
2. Clase ambiente .....	71
3. Funciones:.....	71
➤ Función guardar(): .....	72
➤ Función cargararchivo(): .....	73
➤ Función actualizar(): .....	74
➤ Función initComponets():.....	75
4. eventos por botones:.....	75
➤ evento de botón guardarActionPerformed (), limpiarActionPerformed(),terminarActionPerformed(),cargarActionPerformed() : .....	75

# 1. Muestra plana de la interfaz y representación final interfaz de usuario



## 2. Lenguaje utilizado en la implementación

Para la implementación del ch-maquina se utilizó lenguaje JAVA Java Platform (JDK) 8u73 / 8u74 con editor NetBeans IDE 8.0.1 .

## 3. Introducción y descripción del programa

En este documento pretende dar a conocer a quien lo lea la forma como implemente un programa que corra sobre un computador mostrando cada una de sus partes internas conocidas como caja negra , encargada de todas las operaciones e instrucciones demandadas por el usuario.

El proyecto tiene por nombre MI CH-MAQUINA interfaz de usuario encargada de simular el funcionamiento abstracto de un sistema operativo.

Se mostrara paso a paso cada una de las instrucciones planteadas su descripción y funcionamiento de tal forma que quien entre a estudiar o a modificar el código, tenga las herramientas necesarias para hacer más fácil su trabajo.

## 4. Objeto y aspectos principales del programa.

Realizar una simulación gráfica de un sistema operativo de un ch-computador ficticio de funcionamiento básico.

El programa debe simular un procesador muy elemental y una memoria principal a través de un vector de hasta 9999 posiciones, las cuales pueden ser variadas al momento de iniciar el programa, se asume por defecto que el ch-computador empieza con 100 posiciones de memoria para facilitar el proceso de pruebas.

El programa debe estar en capacidad de leer un conjunto de programas en un pseudo lenguaje de máquina que llamaremos CHMAQUINA y los cargara en las posiciones disponibles de la citada memoria, leerá una instrucción por cada línea de entrada.

Las primeras posiciones de la memoria estarán reservadas para el núcleo del sistema operativo (kernel), el tamaño de este deberá poderse ingresar al iniciar la corrida del simulador, su valor por defecto es 29 correspondiente al condicional del proyecto  $10 * \text{último número de mi documento de identidad} = 2 + 9$  dando como resultado 29.

El programa realizara un chequeo de Sintaxis, produciendo un listado de errores si los hay, de lo contrario procederá a la carga definitiva del programa en memoria y quedará listo para ejecución del mismo bajo las reglas de corrida de múltiples programas.

En cualquier momento de la ejecución del programa mostrar el mapa de memoria (es decir el Vector de memoria y sus posiciones, las variables, lo mismo que el valor del acumulador).

## 5. Que hace el ch-máquina y cuál es su sintaxis.

Se asumirá que el sistema operativo ocupa las primeras posiciones de la memoria, su contenido para este proyecto no es importante y su tamaño se podrá variar solo al iniciar el ambiente de trabajo.

El programa utilizará un acumulador para registrar los valores de los cálculos y recibirá como nombre reservado “acumulador”.

Las posiciones de memoria que almacenen datos tendrán un nombre asociado, la inicialización de variables se asume en cero si es numérico y blanco si es alfanumérico. Estas variables deberán ser creadas antes de ser usadas y tendrá un nombre asociado.

Las instrucciones constarán de 2 partes; el código de la operación y el(los) operando(s) dependiendo el tipo de instrucción.

El código de operación corresponde al nemónico del código de operación y éste puede ser:

### Operación Descripción sintaxis:

- **cargue** Cárguese/copie en el acumulador el valor almacenado en la variable indicada por el operando.
- **Almacene** Guarde/copie el valor que hay en el acumulador a la variable indicada por el operando.
- **Vaya** Salte a la instrucción que corresponde a la etiqueta indicada por el operando y siga la ejecución a partir de allí.
- **Vayasi** Salte Si el valor del acumulador es mayor a de cero a la instrucción que corresponde a la etiqueta indicada por el primer operando.

Si el valor del acumulador es menor a cero a la instrucción que corresponde a la etiqueta indicada por el segundo operando o Si el acumulador es cero a la siguiente instrucción adyacente a la instrucción *vayasi* y siga la ejecución a partir de allí.

- **Nueva** Crea una nueva variable cuyo nombre es el especificado en el primer operando, en el segundo operando definirá el tipo de variable (C cadena/alfanumérico, I Entero, R Real/decimal), un tercer operando establecerá un valor de inicialización; a cada variable se le asignará automáticamente una posición en la memoria. Las variables deberán estar definidas antes de ser utilizadas. Las variables no inicializadas tendrán por defecto el valor cero para reales y enteros y espacio para cadenas. El separador de decimales es el punto.
- **etiqueta** La etiqueta es un nombre que opcionalmente se le puede asignar a una instrucción en el programa para evitar trabajar con las posiciones en memoria de las instrucciones y poder utilizar un nombre simbólico independiente de su ubicación.  
Crea una nueva etiqueta cuyo nombre es el especificado en el primer operando y a la cual le asignará automáticamente la posición indicada en el segundo operando (esta será la posición relativa de la instrucción a la que se le asigna este nombre con respecto a la primera instrucción del programa). Las instrucciones que definen etiquetas podrán definirse en cualquier posición del programa, pero en todo caso antes de la instrucción *retorne*.
- **lea** Lee por teclado el valor a ser asignado a la variable indicado por el operando *sume* Incrementa el valor del acumulador en el valor indicado por la variable señalada por el operando.
- **reste** Decrementa el acumulador en el valor indicado por la variable que señala el operando.
- **multiplique** Multiplica el valor del acumulador por el valor indicado por la variable señalada por el operando.
- **Divida** Divida el valor del acumulador por el valor indicado por la variable señalada por el operando.  
El divisor deberá ser una cantidad diferente de cero.
- **potencia** Eleve el acumulador a la potencia señalada por el operando (los exponentes pueden ser valores enteros, positivos o negativos)
- **modulo** Obtenga el modulo al dividir el valor del acumulador por el valor indicado por la variable señalada por el operando.

- **concatene** Genere una cadena que una la cadena dada por el operando a la cadena que hay en el acumulador (Operando alfanumérico).
- **elimine** Genere una subcadena que elimine cualquier aparición del conjunto de caracteres dados por el operando de la cadena que se encuentra en el acumulador (operando alfanumérico)
- **extraiga** Genere una subcadena que extraiga los primeros caracteres (dados por el valor numérico operando) de la cadena que se encuentra en el acumulador (operando numérico).
- **Muestre** Presente por pantalla el valor que hay en la variable indicada por el operando, si el operando es acumulador muestre el valor del acumulador.
- **Imprima** Lo mismo que el anterior pero presentándolo en la impresora.
- **retorne** El programa termina; debe ser la última instrucción del programa y no tiene operando

### Ejecución del programa:

La ejecución de los programas normalmente se hace de forma secuencial de instrucciones, la primera después la segunda, la tercera....etc., las instrucciones de transferencia de control (vaya y vayasi) son la forma de cambiar este orden de ejecución, obligando que el programa no siga en el orden secuencial predeterminado, sino que continúe en la instrucción señalada por una etiqueta (es decir una instrucción que tiene asignado un nombre como referencia).

Vaya y vayasi cumple esta función, la primera de forma incondicional y la segunda condicionada al valor del acumulador como se especifica en su definición.

La inicialización de posiciones de memoria se hará como instrucciones en las cuales se crean las variables y se les asigna valor, como se explicó en la operación Nueva.

El código puede tener comentarios por líneas, los cuales se denotaran por dos backslash (//) en las dos primeras posiciones de la instrucción, de igual manera se podrán insertar líneas en blanco entre instrucciones del programa, cuyo propósito es de legibilidad del programa.

Se podrán realizar operaciones entre valores enteros y reales, los resultados intermedios se manejaran como reales y el resultado final obedecerá al tipo de variable que almacena el resultado.



El programa no debe permitir la sobrecarga del acumulador (Overflow/desborde) por lo cual sacará un mensaje de error que le permita al usuario tomar la decisión que corresponda.

Inicialmente la protección de memoria se hará por registro base y registro límite, esto es, cada programa empieza en una posición de memoria (registro base) y termina en otra posición de memoria denominada (registro límite) con base en las cuales se evitará la violación de las normas básicas de ejecución, también debe tenerse claro que los programas tendrán área de código y área de datos.

El programa podrá ejecutarse en modalidad normal (corrida continua) o paso a paso (instrucción por instrucción), en todo caso se podrá visualizar la instrucción que se esté ejecutando en cada momento y el respectivo valor del acumulador.

Los ch-programas serán almacenados previamente en archivos con extensión ch en cualquier carpeta de algún medio de almacenamiento, de allí podrán ser cargados al señalarlos de la lista.

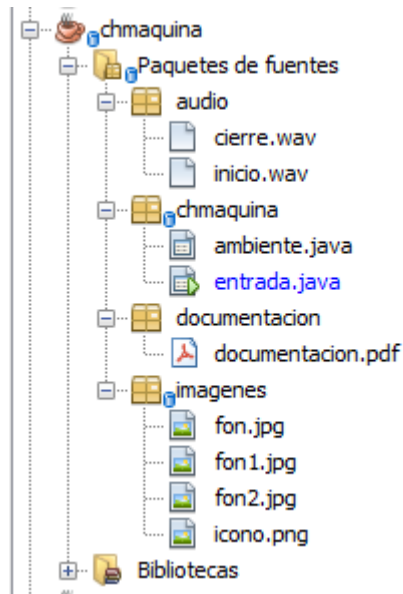
Se podrán cargar y correr varios ch-programas hasta agotar la memoria disponible, para la corrida de los ch-programas en la primera fase se utilizará una cola circular, la cual será visitada con base al orden de llegada (cola simple-primero en entrar primero en ser atendido-FCFS).

El sistema debe indicar por medio de alguna convención si está trabajando en modo usuario (ejecución del programa) o modo kernel (el sistema tiene el control y administración del ambiente), mostrando la acción de cambio de contexto (el paso de un modo al otro). Se podrán ver los distintos estados en los cuales estén los procesos, a nivel de proceso y a nivel de cola.

## 6. Código completo y explicación de funcionamiento del programa.

Se presenta la estructura y contenido de cada paquete por clases del programa y la estructura interna de cada clase:

- **Estructura del proyecto chmaquina:**



El proyecto en su estructura tiene 4 paquetes contenedores que son audio (contiene todos los archivos de sonido utilizados en el proyecto, estos son de extensión .wav), imágenes (contiene todas las imágenes de múltiples tipos utilizadas en el proyecto), Documentación contiene la documentación y manuales de técnico y usuario y chmaquina contenedor de las clases.

➤ **Paquete chmaquina clase entrada.java:**

En este punto se dará una explicación concisa de partes del código encargadas de ciertas funcionalidades de la clase entrada.java.

## 1. Librerías importadas:

```
import java.awt.Desktop;
import java.awt.print.PrinterException;
import java.io.File;
import javax.swing.ImageIcon;
import javax.swing.JFileChooser;
import javax.swing.JLabel;
import javax.swing.JLayeredPane;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JFrame;
import java.io.*;
import java.util.ArrayList;
import java.util.StringTokenizer;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableColumn;
```

Librerías encargadas de interactuar con el código facilitando variedad de operaciones y funciones, tanto en la construcción como en la implementación de código.

## 2. Clase entrada

```
31
32  /**
33   *
34   * @author aguir
35   */
36  public class entrada extends JFrame {
37
38      /**
39       * Creates new form entrada
40       */
41      DefaultTableModel modelo,tprocesos,tvariables, tetiquetas;
42      String arc = "documentacion.pdf";
43      int programa=1; // cantidad de programas cargados
44      String memoriaprin[]; // vector de memoria principal
45      public static ArrayList<String> instrucciones;
46      public static ArrayList<String> nvariables;
47      public static ArrayList<Object[]> var;
48      public static ArrayList<Object[]> etiq;
49
50      int pivote=0;
51      int rlp;
52      int inicialproceso=0, inicialvariables=0,inicialetiquetas=0;
53
54      public entrada() {
55          initComponents();
56          instrucciones=new ArrayList();
57          nvariables=new ArrayList();
58          var=new ArrayList();
59          etiq=new ArrayList();
60          setLocationRelativeTo(null);
```

```

61     setResizable(true); // permite que la ventana principal se pueda maximizar o minimizar
62     setExtendedState(JFrame.MAXIMIZED_BOTH); // hace que la ventana siempre aparesca maximizada
63     setTitle("MI CH-MAQUINA"); // titulo del programa
64     setIconImage(new ImageIcon(getClass().getResource("/imagenes/icono.png")).getImage()); // icono de la ventana del programa
65     cargarprograma.setEnabled(false); // impide cargar programas sin prender maquina
66     // impide apagar la maquina sin encenderla
67     apagarmaquina1.setEnabled(false);
68     apagarmaquina2.setEnabled(false);
69     // desactiva botones que no pueden ser inicializados sin orden previa
70     botoncargar.setEnabled(false);
71     ejecutar.setVisible(false);
72     editor.setVisible(false);
73     pasoapaso.setVisible(false);
74     IMP.setEnabled(false);
75     EJEK.setEnabled(false);
76
77
78     //fundamento encargado de la imagen de fondo del ch-maquina
79     ((JPanel) getContentPane()).setOpaque(false);
80     ImageIcon uno = new ImageIcon(this.getClass().getResource("/imagenes/fon1.jpg"));
81     JLabel fondo = new JLabel();
82     fondo.setIcon(uno);
83     getLayeredPane().add(fondo, JLayeredPane.FRAME_CONTENT_LAYER);
84     fondo.setBounds(0, 0, uno.getIconWidth(), uno.getIconHeight());
85
86
87
88
89     //definen los valores por defecto de la memoria del kernel y la memoria disponible para programas
90     int maxmemo = 9999, maxkerner = 1000;

```

```

91     memoria.setModel(new javax.swing.SpinnerNumberModel(100, 2, maxmemo, 1));
92     kernel.setModel(new javax.swing.SpinnerNumberModel(29, 1, maxkerner, 1));
93     total_memoria.setText("71");
94
95
96     //CREA EL TIPO DE MODELO DE TABLA para mapa de memoria
97     modelo = new DefaultTableModel();
98     tabla.setModel(modelo);
99     // CREA LOS NOMBRES DE LAS COLUMNAS
100    modelo.addColumn("POS-MEMO");
101    modelo.addColumn("PROGRAMA");
102    modelo.addColumn("INSTRUCCION");
103    modelo.addColumn("ARGUMENTO");
104    modelo.addColumn("VALOR");
105    //redimenciona la columna
106    TableColumn columna = tabla.getColumnModel("POS-MEMO");
107    columna.setPreferredWidth(80); // pixeles por defecto
108    columna.setMinWidth(50); //pixeles minimo
109    columna.setMaxWidth(90); // pixeles maximo
110
111    TableColumn PRE = tabla.getColumnModel("PROGRAMA");
112    PRE.setPreferredWidth(80); // pixeles por defecto
113    PRE.setMinWidth(10); //pixeles minimo
114    PRE.setMaxWidth(200); // pixeles maximo
115
116    TableColumn PREE = tabla.getColumnModel("INSTRUCCION");
117    PREE.setPreferredWidth(120); // pixeles por defecto
118    PREE.setMinWidth(10); //pixeles minimo
119    PREE.setMaxWidth(200); // pixeles maximo
120


```

```

121 TableColumn PREEE = tabla.getColumn("VALOR");
122 PREEE.setPreferredWidth(80); // pixeles por defecto
123 PREEE.setMinWidth(10); // pixeles minimo
124 PREEE.setMaxWidth(200); // pixeles maximo
125
126 // CREA EL TIPO DE MODELO DE TABLA para procesos
127 tprocesos = new DefaultTableModel();
128 tabla2.setModel(tprocesos);
129 // CREA LOS NOMBRES DE LAS COLUMNAS
130 tprocesos.addColumn("ID");
131 tprocesos.addColumn("PROGRAMAS");
132 tprocesos.addColumn("#INST");
133 tprocesos.addColumn("RB");
134 tprocesos.addColumn("RLC");
135 tprocesos.addColumn("RLP");
136
137 //redimensiona la columna
138 TableColumn id = tabla2.getColumn("ID");
139 id.setPreferredWidth(40); // pixeles por defecto
140 id.setMinWidth(10); // pixeles minimo
141 id.setMaxWidth(41); // pixeles maximo
142
143 TableColumn pro = tabla2.getColumn("PROGRAMAS");
144 pro.setPreferredWidth(100); // pixeles por defecto
145 pro.setMinWidth(10); // pixeles minimo
146 pro.setMaxWidth(501); // pixeles maximo
147
148 TableColumn ins = tabla2.getColumn("#INST");
149 ins.setPreferredWidth(50); // pixeles por defecto
150 ins.setMinWidth(10); // pixeles minimo

```

```

151      ins.setMaxWidth(51); // pixeles maximo
152
153      TableColumn rb = tabla2.getColumn("RB");
154      rb.setPreferredWidth(40); // pixeles por defecto
155      rb.setMinWidth(10); // pixeles minimo
156      rb.setMaxWidth(41); // pixeles maximo
157
158      TableColumn rcl = tabla2.getColumn("RLC");
159      rcl.setPreferredWidth(40); // pixeles por defecto
160      rcl.setMinWidth(10); // pixeles minimo
161      rcl.setMaxWidth(41); // pixeles maximo
162
163       TableColumn rlp = tabla2.getColumn("RLP");
164      rlp.setPreferredWidth(40); // pixeles por defecto
165      rlp.setMinWidth(10); // pixeles minimo
166      rlp.setMaxWidth(41); // pixeles maximo
167
168
169      // CREA EL TIPO DE MODELO DE TABLA para variables
170      tvariables = new DefaultTableModel();
171      tablavariables.setModel(tvariables);
172      // CREA LOS NOMBRES DE LAS COLUMNAS
173      tvariables.addColumn("POS");
174      tvariables.addColumn("PROG");
175      tvariables.addColumn("TIPO");
176      tvariables.addColumn("VARIABLES");
177      tvariables.addColumn("VALOR");
178
179      //redimensiona la columna
180      TableColumn POS = tablavariables.getColumn("POS");

```



```

181 POS.setPreferredWidth(40);// pixeles por defecto
182 POS.setMinWidth(40);//pixeles minimo
183 POS.setMaxWidth(41);// pixeles maximo
184
185 TableColumn prog = tablavariables.getColumn("PROG");
186 prog.setPreferredWidth(60);// pixeles por defecto
187 prog.setMinWidth(10);//pixeles minimo
188 prog.setMaxWidth(61);// pixeles maximo
189
190
191
192 TableColumn vLr = tablavariables.getColumn("VALOR");
193 vLr.setPreferredWidth(60);// pixeles por defecto
194 vLr.setMinWidth(10);//pixeles minimo
195 vLr.setMaxWidth(61);// pixeles maximo
196
197
198 //CREA EL TIPO DE MODELO DE TABLA para etiquetas
199 tetiquetas = new DefaultTableModel();
200 tablaetiquetas.setModel(tetiquetas);
201 // CREA LOS NOMBRES DE LAS COLUMNAS
202 tetiquetas.addColumn("POS");
203 tetiquetas.addColumn("PROG");
204 tetiquetas.addColumn("ETIQUETAS");
205 tetiquetas.addColumn("ARGUMENTO");
206 //redimenciona la columna
207 TableColumn POSS = tablaetiquetas.getColumn("POS");
208 POSS.setPreferredWidth(50);// pixeles por defecto
209 POSS.setMinWidth(40);//pixeles minimo
210 POSS.setMaxWidth(90);// pixeles maximo

```

```

211
212     TableColumn POG = tablaetiquetas.getColumn("PROG");
213     POG.setPreferredWidth(50); // pixeles por defecto
214     POG.setMinWidth(40); // pixeles minimo
215     POG.setMaxWidth(90); // pixeles maximo
216
217     TableColumn POGG = tablaetiquetas.getColumn("ETIQUETAS");
218     POGG.setPreferredWidth(70); // pixeles por defecto
219     POGG.setMinWidth(40); // pixeles minimo
220     POGG.setMaxWidth(90); // pixeles maximo
221
222     // evita editar el contenido de los jtextpanel
223     monitor.setEditable(false);
224     impresora.setEditable(false);
225
226 }
227

```

Podemos identificar la función principal **entrada ()**; esta se encarga de inicializar todos los componentes de arranque de la interfaz principal inicializando imágenes títulos tablas spinners entre otros y asignándoles valores por defecto, los cuales durante la implementación pueden cambiar.

### 3. Funciones:

Encargadas de ejecutar instrucciones dadas por el usuario.

#### ➤ Función memtotal():

```
230
231 //funcion encargada de capturar los valores de kernel y memoria y mostrar la cantidad de
232 //memoria que queda disponible para la asignacion de los programas
233 public void memtotal(){
234     int mem = (int) memoria.getValue();
235     int ker = (int) kernel.getValue();
236     int total= mem - ker;
237     total_memoria.setText(String.valueOf(total));
238
239 }
240
```

Esta función toma los valores de los spinner de memoria y kernel y saca el total de memoria neta que estará disponible para los programas a cargar y retorna el resultado a un JLabel con nombre total memoria donde puede ser visto por el usuario.

#### ➤ Función son():

```
241
242 // funcion para reproducir sonidos
243 public Clip clip;
244 public String ruta="/audio/";
245
246
247 public void son(String archivo){
248     //carga en bufer el archivo de audio
249     BufferedInputStream Mystream = new BufferedInputStream(getClass().getResourceAsStream(ruta+archivo+".wav"));
250
251     try{
252         //ejecuta el audio
253         AudioInputStream song = AudioSystem.getAudioInputStream(Mystream);
254         Clip sonido = AudioSystem.getClip();
255         sonido.open(song);
256         sonido.start();
257     }catch(Exception e){
258
259     }
260 }
```

Esta función le envían un nombre de un archivo y este concatena la dirección por defecto del archivo más el nombre más la extensión del archivo por defecto y lo almacena en un buffer de memoria, dentro de esta función va encapsulado la ejecución del sonido ya que puede existir la posibilidad que surjan errores de reproducción por tanto hay un disparador que impida el bloqueo del programa si no se reproduce el archivo sonoro.

➤ Función encender():

```
262
263 // funcion enargada de encender la maquina y cargar la memoria
264 public void encender(){
265     // HACE EL LLAMADO A LA FUNCION PARA QUE REPRODUzca EL SONIDO DE ENSENDIDO
266     // desactiva los spinner
267     kernel.setEnabled(false);
268     memoria.setEnabled(false);
269     encender.setEnabled(false);
270     encender2.setEnabled(false);
271     cargarprograma.setEnabled(true);
272     apagarmaquina1.setEnabled(true);
273     apagarmaquina2.setEnabled(true);
274     botoncargar.setEnabled(true);
275     estado.setText("MODULO KEREL");
276     editor.setVisible(true);
277
278     //sonidoencender("inicio");
279     son("inicio");
280     rlp = (int)kernel.getValue()+1; // inicualiza el primer rcl
281     // INSTANCIA OBJETO PARA LLENAR LA TABLA
282     Object []object = new Object[5];
283     // inicializa la memoria principal con el tamaño de memoria establecido
284     memoriaprin= new String[(int)memoria.getValue()];
285     // VALORES POR DEFECTO DE LA PRIMERA POSICION DEL MAPA D EMEMORIA
286     object[0]="0";
287     object[1]="0000";
288     object[2]="----";
289     object[3]="acumulador";
290     object[4]="0";
```

```

291     memoriaprin[0]="acumulador";// carga en la memoria
292     modelo.addRow(object);
293
294     // CICLOS ENCARGADOS DE LLENAR EL MAPA DE MEMORIA
295     int contador = 0;
296
297     int mem = Integer.parseInt(total_memoria.getText());
298     int ker = (int) kernel.getValue();
299     for (int i = 0; i < ker; i++) {
300         contador++;
301         object[0]=String.valueOf(contador);
302         object[1]="0000";
303         object[2]="----";
304         object[3]="-----sistema operativo-----";
305         object[4]="----";
306         modelo.addRow(object);
307         memoriaprin[i+1]="-----sistema operativo-----";
308     }
309
310     pivote = pivote + ker +1; // crea un pivote marcador de inicio de primer programa
311     for (int i = 0; i <mem-1; i++) {
312         contador++;
313         object[0]=String.valueOf(contador);
314         object[1]="";
315         object[2]="";
316         object[3]="";
317         object[4]="";
318         modelo.addRow(object);
319     }
320
321
322     // se encarga de crear el contenido de un programa en la tabla de procesos
323     Object []objectprocesos = new Object[6];
324     objectprocesos[0]="0000";// # instancias
325     objectprocesos[1]="SISTEMA OPERATIVO ch-maquina"; // nombre del programa
326     objectprocesos[2]=ker; // numero de lineas del programa
327     objectprocesos[3]=1; // rb
328     objectprocesos[4]=ker; //registro limite de el programa
329     objectprocesos[5]=ker+1 ; // crea el rlp
330     tprocesos.addRow(objectprocesos);// adiciona a la tabla
331
332 }
333

```

Esta función se encarga de hacer el segundo arranque de la interfaz aplica la inicialización del mapa de memoria el sonido de encendido y desactivación de botones de carga de archivo, encendido de máquina y los medidores de memoria y kernel.

➤ Función temporales():

```
333
334 // se encarga de borrar los archivos temporales
335 public void temporales() {
336
337     File temp = new File(arc);
338     temp.delete();
339 }
340
```

Esta función se encarga de borrar archivos temporales almacenados en memoria abiertos como la documentación o los manuales.

➤ Función apagar():

```
333
334 // se encarga de borrar los archivos temporales
335 public void temporales() {
336
337     File temp = new File(arc);
338     temp.delete();
339 }
340
341 // funcion encargada de apagar la maquina y regresarla a su estado inicial
342 public void apagar() {
343     // codigo encargado de apagar la maquina y regresarla a el estado inicial
344     temporales(); // borra los archivos temporales
345
346     if(JOptionPane.showOptionDialog(this, "¿ESTA SEGURO QUE DESEA APAGAR LA MAQUINA?", "Mensaje de Alerta",
347         JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE, null, new Object[]{"SI", "NO"}, "NO")==0)
348     {
349         son("cierre");
350         // se encarga de detener un instante el proceso
351         try {
352             Thread.sleep(2000); // el tiempo es en milisegundos
353         } catch (InterruptedException ex) {
354
355         }
356         setVisible(false);
357         new entrada().setVisible(true);
358     }
359     else
360     {

```

```

361         JOptionPane.showMessageDialog(this, "PUEDE CONTINUAR CON LA EJECUCION DEL PROGRAMA");
362     }
363
364
365
366     }

```

Esta función se encarga de hacer un reinicio a los valores por defecto del programa y cuenta con dos opciones si para reiniciar y no para regresar al entorno actual, en esta función se activa un sonido de cierre de sesión y un retardador de tiempo con el objetivo de que del efecto de apagado.

#### ➤ Función cargararchivo():

```

368 public void cargararchivo() {
369     // encargado de abrir el panel de busqueda de archivos y cargarlo a la funcion actualizar.
370     JFileChooser ventana = new JFileChooser();
371     // filtra las extensiones segun la que buscamos
372     ventana.setFileFilter(new FileNameExtensionFilter("todos los archivos "
373         + "*.ch", "CH", "ch"));
374     int sel = ventana.showOpenDialog(entrada.this);
375
376     // incrementan en uno el contador
377     inicialproceso++;
378
379
380     // condicional que le dara el nombre a el programa
381     String prefijo;
382     if (programa<10) {
383         prefijo="000"+String.valueOf(programa);
384         programa++;
385     }else{
386         prefijo="00"+String.valueOf(programa);
387     }
388
389     if (sel == JFileChooser.APPROVE_OPTION) {
390
391         File file = ventana.getSelectedFile();
392
393         String nombrea=file.getName();
394         actualizar(file.getPath(), prefijo,nombrea );
395
396     }
397 }

```

Función encargada de abrir el panel del asistente de búsqueda, que cargara el archivo que contiene las instrucciones ch para ser cargadas en la memoria.

➤ Función actualizar():

```
400 // funcion encargada de leer el archivo y hacer el token
401 public void actualizar(String url, String pre,String nombre){
402     int lexa =0;
403     long lNumeroLineas = 0;// INICIALIZA EL CONTADOR DE LAS LINEAS DEL ARCHIVO
404     try{
405         instrucciones.clear();
406         nvariables.clear();
407         // limpia los arreglos para que no queden rastros del programa anterior
408         etiq.clear();
409         var.clear();
410         // lee el archivo y lo carga en bufer
411         FileReader file = new FileReader(url);
412         BufferedReader leer = new BufferedReader(file);
413
414         //Inicializo todas las variables a leer
415         //de forma general
416
417         String operacion="";// almacena el primer token de la linea examinada
418
419         String variablenueva="", tipo="", valor="";
420         String nombreetiqueta="", numerolinea="";
421         String variablealmacene="", variablecargue="";
422
423
424         // variables para calculos matematicos
425         String variablesume="";
426         String variablereste="";
427         String variablemultiplique="";
428         String variabledivida="";
429         String variablepotencia="";
430         String variablemodulo="";
```



```

433 String variableconcatene="";
434 String variableelimine="",variableextraiga="";
435
436 // ciclos
437 String etiquetaini="";
438 String etiquetainicio="", etiquetafin="";
439
440 // entrega de resultados
441 String variablemuestre="";
442 String variableimprimir="";
443
444
445 //operaciones con cadenas
446 String variablelea="";
447
448 // ALMACENARA LA LISTA DELOS ERRORES ENCONTRADOS
449 String errores= "**** ERRORES ENCONTRADOS ****\n\n";
450
451
452 // SE ENCARGA DE RECORRER EL ARCHIVO Y CONTAR LA CANTIDAD DE LINEAS
453
454 String sCadena;
455 // CICLO QUE RECORRE CADA LINEA HASTA QUE LA LINEA SEA NULL
456 while ((sCadena = leer.readLine())!=null) {
457     lNumeroLineas++;
458 }
459

```

```

462
463     FileReader file2 = new FileReader(url);
464     BufferedReader leer2 = new BufferedReader(file2);
465
466
467     int inicialmemoria=pivote;
468     int inicialprocesos= inicialproceso;
469     int inicialvariable=inicialvariables;
470     int inicialetiqueta=inicialetiquetas;
471
472     int posi=pivote-1; // nos dice en que pision va almacenando instrucciones
473
474
475     // captura la cantidad de filas ocupadas de la tabla d evariables
476     int q=tvariables.getRowCount();
477     // FOR ENCARGADO DE RECORRER EL ARCHIVO LINEA POR LINEA PARA HACER LOS TOKENS
478     for (int i=0; i<lNumeroLineas; i++){
479         //Se usa 'StringTokenizer' para tomar toda la linea examinada
480         posi++; // aumenta en uno las posiciones d ememoria para ocupar
481         String linea=leer2.readLine().trim();
482
483         lexa++;
484         StringTokenizer tk = new StringTokenizer(linea);
485
486         // condiciona la linea para saber si esta vacia
487         if (linea.length()>0) {
488             operacion= (tk.nextToken());
489
490         }else{
491             // en caso tal que la linea este vacia
492             operacion=" ";

```

```

494     }
495     // evalua por casos cada linea y hace los tokens correspondientes
496     switch (operacion) {
497     case "cargue":
498         if (tk.countTokens()==1) {
499             // hace el segundo token de la linea
500             variablecargue= (tk.nextToken());
501
502             //agrega la linea completa al mapa de memoria
503             modelo.setValueAt(pre, posi, 1);// guarda en la tabla de memoria el numero del programa
504             modelo.setValueAt(operacion, posi, 2);// guarda en la tabla la instruccion del programa
505             modelo.setValueAt(linea, posi, 3);// guarda en la tabla el argumento de memoria
506             modelo.setValueAt(variablecargue, posi, 4);// guarda en la tabla el valor de memoria
507             memoriaprin[posi]=linea; // guarda en el vector principal de memoria
508             instrucciones.add(pre + " " + linea);
509             break;
510         }else{
511             errores=errores+"debe tener dos argumentos en esta linea";
512             throw new Exception("Invalid entry");
513         }
514
515
516     case "almacene":
517         // hace el segundo token de la linea
518         variablealmacene= (tk.nextToken());
519         //agrega en el array list de instrucciones
520
521         modelo.setValueAt(pre, posi, 1);// guarda en la tabla de memoria el numero del programa
522         modelo.setValueAt(operacion, posi, 2);// guarda en la tabla la instruccion del programa
523         modelo.setValueAt(linea, posi, 3);// guarda en la tabla el argumento de memoria
524         modelo.setValueAt(variablealmacene, posi, 4);// guarda en la tabla el valor de memoria
525         memoriaprin[posi]=linea; // guarda en el vector principal de memoria

```

```

526     instrucciones.add(pre + " " + linea);
527     break;
528
529     case "vaya":
530         // hace el segundo token de la linea
531         etiquetaini= (tk.nextToken());
532         //agrega en el array list de instrucciones
533
534         //agrega la linea completa al mapa de memoria
535         modelo.setValueAt(pre, posi, 1); // guarda en la tabla de memoria el numero del programa
536         modelo.setValueAt(operacion, posi, 2); // guarda en la tabla la instruccion del programa
537         modelo.setValueAt(linea, posi, 3); // guarda en la tabla el argumento de memoria
538         modelo.setValueAt(etiquetaini, posi, 4); // guarda en la tabla el valor de memoria
539
540         memoriaprin[posi]=linea; // guarda en el vector principal de memoria
541         instrucciones.add(pre + " " + linea);
542         break;
543
544     case "vayasi":
545         // hace el segundo token de la linea
546         etiquetaini= (tk.nextToken());
547         etiquetafin=(tk.nextToken());
548         //agrega en el array list de instrucciones
549         //agrega la linea completa al mapa de memoria
550         modelo.setValueAt(pre, posi, 1); // guarda en la tabla de memoria el numero del programa
551         modelo.setValueAt(operacion, posi, 2); // guarda en la tabla la instruccion del programa
552         modelo.setValueAt(linea, posi, 3); // guarda en la tabla el argumento de memoria
553         modelo.setValueAt(etiquetaini+" "+etiquetafin, posi, 4); // guarda en la tabla el valor de memoria
554         memoriaprin[posi]=linea; // guarda en el vector principal de memoria
555         instrucciones.add(pre + " " + linea);
556         break;

```

```

557
558
559 case "nueva":
560     if (tk.countTokens()>=3 || tk.countTokens()==4) {
561         inicialvariables++;
562         // hace el segundo token de la linea
563         variablenueva= (tk.nextToken());
564         tipo=(tk.nextToken());
565         if (tk.countTokens()==3) {
566             if ("c".equals(variablenueva) || "C".equals(variablenueva) ) {
567                 valor=" ";
568             }else{
569                 valor="0";
570             }
571         }else{
572             valor= (tk.nextToken());
573         }
574
575         //agrega en el array list de instrucciones
576         //agrega la linea completa al mapa de memoria
577         modelo.setValueAt(pre, posi, 1); // guarda en la tabla de memoria el numero del programa
578         modelo.setValueAt(operacion, posi, 2); // guarda en la tabla la instruccion del programa
579         modelo.setValueAt(linea, posi, 3); // guarda en la tabla el argumento de memoria
580         modelo.setValueAt(valor, posi, 4); // guarda en la tabla el valor de memoria
581
582         memoriaprin[posi]=linea; // guarda en el vector principal de memoria
583
584         Object nuevo[]=new Object[5];
585         nuevo[0]=" ";
586         nuevo[1]=pre;
587         nuevo[3]=variablenueva;
588         switch (tipo){

```

```

589     case "i":
590     case "I":
591         tipo="ENTERO";
592         break;
593
594     case "r":
595     case "R":
596         tipo="REAL";
597         break;
598
599     case "c":
600     case "C":
601         tipo="CADENA";
602         break;
603     default:
604         errores= errores + "** hay un error de sintaxis en la linea "+lNumeroLineas+"\n"+
605             "parece error en el tipo de variable";
606
607
608     }
609     nuevo[2]=tipo;
610     nuevo[4]=valor;
611     var.add(nuevo); // almacena en la un array list para luego pasarlo a la tabla variables
612
613     nvariables.add(variablenuova) ;
614     nvariables.add(valor) ;
615     break;
616 }else{
617     throw new Exception("Invalid entry");
618 }
619

```

```

620
621 case "etiqueta":
622     // hace el segundo token de la linea
623     nombreetiqueta =(tk.nextToken());
624     numerolinea=(tk.nextToken());
625
626
627     iniciaetiquetas++;
628     //agrega la linea completa al mapa de memoria
629     modelo.setValueAt(pre, posi, 1);// guarda en la tabla de memoria el numero del programa
630     modelo.setValueAt(operacion, posi, 2);// guarda en la tabla la instruccion del programa
631     modelo.setValueAt(linea, posi, 3);// guarda en la tabla el argumento de memoria
632     modelo.setValueAt(nombreetiqueta, posi, 4);// guarda en la tabla el valor de memoria
633     memoriaprin[posi]=linea; // guarda en el vector principal de memoria
634
635     Object netiqueta[]=new Object[4];
636
637     // carga de nuevo el documento para recorrerlo de nuevo y encontrar la linea a etiquetar
638     FileReader file3 = new FileReader(url);
639     BufferedReader leer3 = new BufferedReader(file3);
640     String eti="";
641     int j;
642     // recorre el documento hasta la linea requerida
643     for ( j = 0; j < Integer.parseInt(numerolinea); j++) {
644         eti=leer3.readLine();
645     }
646
647     netiqueta[0]=pivote+j-1;//posicion en memoria
648     netiqueta[1]=pre; //programa al q pertenece
649     netiqueta[2]=nombreetiqueta; // nombre etiqueta mas
650     netiqueta[3]=eti; // la linea renombrada
651     etiq.add(netiqueta); // adiciona el arreglo a la tabla etiquetas

```

```

652         break;
653
654     case "lea":
655         variablelea=(tk.nextToken());
656         //agrega la linea completa al mapa de memoria
657         modelo.setValueAt(pre, posi, 1);// guarda en la tabla de memoria el numero del programa
658         modelo.setValueAt(operacion, posi, 2);// guarda en la tabla la instruccion del programa
659         modelo.setValueAt(linea, posi, 3);// guarda en la tabla el argumento de memoria
660         modelo.setValueAt(variablelea, posi, 4);// guarda en la tabla el valor de memoria
661         memoriaprin[posi]=linea; // guarda en el vector principal de memoria
662         break;
663
664     case "sume":
665         variablesume=(tk.nextToken());
666         //agrega la linea completa al mapa de memoria
667         modelo.setValueAt(pre, posi, 1);// guarda en la tabla de memoria el numero del programa
668         modelo.setValueAt(operacion, posi, 2);// guarda en la tabla la instruccion del programa
669         modelo.setValueAt(linea, posi, 3);// guarda en la tabla el argumento de memoria
670         modelo.setValueAt(variablesume, posi, 4);// guarda en la tabla el valor de memoria
671         memoriaprin[posi]=linea; // guarda en el vector principal de memoria
672         break;
673
674     case "reste":
675         variablereste=(tk.nextToken());
676         //agrega la linea completa al mapa de memoria
677         modelo.setValueAt(pre, posi, 1);// guarda en la tabla de memoria el numero del programa
678         modelo.setValueAt(operacion, posi, 2);// guarda en la tabla la instruccion del programa
679         modelo.setValueAt(linea, posi, 3);// guarda en la tabla el argumento de memoria
680         modelo.setValueAt(variablereste, posi, 4);// guarda en la tabla el valor de memoria
681         memoriaprin[posi]=linea; // guarda en el vector principal de memoria
682         break;

```



```

683
684 case "multiplique":
685     variablemultiplique=(tk.nextToken());
686     //agrega la linea completa al mapa de memoria
687     modelo.setValueAt(pre, posi, 1);// guarda en la tabla de memoria el numero del programa
688     modelo.setValueAt(operacion, posi, 2);// guarda en la tabla la instruccion del programa
689     modelo.setValueAt(linea, posi, 3);// guarda en la tabla el argumento de memoria
690     modelo.setValueAt(variablemultiplique, posi, 4);// guarda en la tabla el valor de memoria
691     memoriaprin[posi]=linea; // guarda en el vector principal de memoria
692     break;
693
694 case "divida":
695     variabledivida=(tk.nextToken());
696     //agrega la linea completa al mapa de memoria
697     modelo.setValueAt(pre, posi, 1);// guarda en la tabla de memoria el numero del programa
698     modelo.setValueAt(operacion, posi, 2);// guarda en la tabla la instruccion del programa
699     modelo.setValueAt(linea, posi, 3);// guarda en la tabla el argumento de memoria
700     modelo.setValueAt(variabledivida, posi, 4);// guarda en la tabla el valor de memoria
701     memoriaprin[posi]=linea; // guarda en el vector principal de memoria
702     break;
703
704 case "potencia":
705     variablepotencia=(tk.nextToken());
706     //agrega la linea completa al mapa de memoria
707     modelo.setValueAt(pre, posi, 1);// guarda en la tabla de memoria el numero del programa
708     modelo.setValueAt(operacion, posi, 2);// guarda en la tabla la instruccion del programa
709     modelo.setValueAt(linea, posi, 3);// guarda en la tabla el argumento de memoria
710     modelo.setValueAt(variablepotencia, posi, 4);// guarda en la tabla el valor de memoria
711     memoriaprin[posi]=linea; // guarda en el vector principal de memoria
712     break;
713

```

```

714 case "modulo":
715     variablemodulo=(tk.nextToken());
716     //agrega la linea completa al mapa de memoria
717     modelo.setValueAt(pre, posi, 1);// guarda en la tabla de memoria el numero del programa
718     modelo.setValueAt(operacion, posi, 2);// guarda en la tabla la instruccion del programa
719     modelo.setValueAt(linea, posi, 3);// guarda en la tabla el argumento de memoria
720     modelo.setValueAt(variablemodulo, posi, 4);// guarda en la tabla el valor de memoria
721     memoriaprin[posi]=linea; // guarda en el vector principal de memoria
722     break;
723
724 case "concatene":
725     variableconcatene=(tk.nextToken());
726     //agrega la linea completa al mapa de memoria
727     modelo.setValueAt(pre, posi, 1);// guarda en la tabla de memoria el numero del programa
728     modelo.setValueAt(operacion, posi, 2);// guarda en la tabla la instruccion del programa
729     modelo.setValueAt(linea, posi, 3);// guarda en la tabla el argumento de memoria
730     modelo.setValueAt(variableconcatene, posi, 4);// guarda en la tabla el valor de memoria
731     memoriaprin[posi]=linea; // guarda en el vector principal de memoria
732     break;
733
734 case "elimine":
735     variableelimine=(tk.nextToken());
736     //agrega la linea completa al mapa de memoria
737     modelo.setValueAt(pre, posi, 1);// guarda en la tabla de memoria el numero del programa
738     modelo.setValueAt(operacion, posi, 2);// guarda en la tabla la instruccion del programa
739     modelo.setValueAt(linea, posi, 3);// guarda en la tabla el argumento de memoria
740     modelo.setValueAt(variableelimine, posi, 4);// guarda en la tabla el valor de memoria
741     memoriaprin[posi]=linea; // guarda en el vector principal de memoria
742     break;
743

```

```

744 case "extraiga":
745     variableextraiga=(tk.nextToken());
746     //agrega la linea completa al mapa de memoria
747     modelo.setValueAt(pre, posi, 1);// guarda en la tabla de memoria el numero del programa
748     modelo.setValueAt(operacion, posi, 2);// guarda en la tabla la instruccion del programa
749     modelo.setValueAt(linea, posi, 3);// guarda en la tabla el argumento de memoria
750     modelo.setValueAt(variableextraiga, posi, 4);// guarda en la tabla el valor de memoria
751     memoriaprin[posi]=linea; // guarda en el vector principal de memoria
752     break;
753
754 case "muestre":
755     variablemuestre=(tk.nextToken());
756     //agrega la linea completa al mapa de memoria
757     modelo.setValueAt(pre, posi, 1);// guarda en la tabla de memoria el numero del programa
758     modelo.setValueAt(operacion, posi, 2);// guarda en la tabla la instruccion del programa
759     modelo.setValueAt(linea, posi, 3);// guarda en la tabla el argumento de memoria
760     modelo.setValueAt(variablemuestre, posi, 4);// guarda en la tabla el valor de memoria
761     memoriaprin[posi]=linea; // guarda en el vector principal de memoria
762     break;
763
764 case "imprima":
765     variableimprimir=(tk.nextToken());
766     //agrega la linea completa al mapa de memoria
767     modelo.setValueAt(pre, posi, 1);// guarda en la tabla de memoria el numero del programa
768     modelo.setValueAt(operacion, posi, 2);// guarda en la tabla la instruccion del programa
769     modelo.setValueAt(linea, posi, 3);// guarda en la tabla el argumento de memoria
770     modelo.setValueAt(variableimprimir, posi, 4);// guarda en la tabla el valor de memoria
771     memoriaprin[posi]=linea; // guarda en el vector principal de memoria
772     break;
773

```

```

774 case "retorne":
775
776     //agrega la linea completa al mapa de memoria
777     modelo.setValueAt(pre, posi, 1); // guarda en la tabla de memoria el numero del programa
778     modelo.setValueAt(operacion, posi, 2); // guarda en la tabla la instruccion del programa
779     modelo.setValueAt(linea, posi, 3); // guarda en la tabla el argumento de memoria
780     modelo.setValueAt("----", posi, 4); // guarda en la tabla el valor de memoria
781     memoriaprin[posi]=linea; // guarda en el vector principal de memoria
782     break;
783
784 case "//":
785     //agrega la linea completa al mapa de memoria
786     modelo.setValueAt(pre, posi, 1); // guarda en la tabla de memoria el numero del programa
787     modelo.setValueAt("COMENTARIO", posi, 2); // guarda en la tabla la instruccion del programa
788     modelo.setValueAt(linea, posi, 3); // guarda en la tabla el argumento de memoria
789     modelo.setValueAt("----", posi, 4); // guarda en la tabla el valor de memoria
790     memoriaprin[posi]=linea; // guarda en el vector principal de memoria
791     break;
792
793 case " ":
794     //agrega la linea completa al mapa de memoria
795     modelo.setValueAt(pre, posi, 1); // guarda en la tabla de memoria el numero del programa
796     modelo.setValueAt("LINEA VACIA", posi, 2); // guarda en la tabla la instruccion del programa
797     modelo.setValueAt(linea, posi, 3); // guarda en la tabla el argumento de memoria
798     modelo.setValueAt("----", posi, 4); // guarda en la tabla el valor de memoria
799     memoriaprin[posi]=linea; // guarda en el vector principal de memoria
800     break;

```

```

801
802         default:
803             // borra el ultimo programa cargado de la memoria
804             for (int h = inicialmemoria; h < (int)memoria.getValue(); h++) {
805                 modelo.setValueAt("", h, 1);
806                 memoriaprin[h]="";
807             }
808             // hace el llamado a la exeption si algo esta mal en el archivo
809             throw new Exception("Invalid entry");
810         }
811     }
812     // carga si no hay problema las variables a la tabla variables
813     for (int b = 0; b < var.size(); b++) {
814
815         tvariables.addRow(var.get(b));
816
817     }
818
819     // ciclo que le asigna el valor de la posicion de memoria donde esta
820     //hubicado y lo muestra en tabla de variables
821     int tempoposi=posi;
822     // toma el valor de las variables del nuevo archivo mas las filas ocupadas de la tabla
823     int lim=(nvariables.size()/2)+q;
824     for ( int r=q; r < lim; r++) {
825         tempoposi++;
826         // le da el valor de la posicion de la variable en la memoria a la tabala d evariables
827         tvariables.setValueAt(tempoposi, r, 0);
828     }
829     //concatena el prefijo con la instruccion en el mapa de memoria las variables defidas
830     for (int a = 0; a < nvariables.size(); a+=2) {
831         posi++;

```

```

831         posi++;
832         // agrega toda la instruccion a la memoria
833
834         modelo.setValueAt(pre, posi, 1); // guarda en la tabla de memoria el numero del programa
835         modelo.setValueAt("VARIABLE", posi, 2); // guarda en la tabla la instruccion del programa
836         modelo.setValueAt(nvariables.get(a), posi, 3); // guarda en la tabla el argumento de memoria
837         modelo.setValueAt(nvariables.get(a+1), posi, 4); // guarda en la tabla el valor de memoria
838         memoriaprin[posi]=nvariables.get(a); // guarda en el vector principal de memoria
839     }
840     for (int c = 0; c < etiq.size(); c++) {
841         tetiquetas.addRow(etiq.get(c));
842     }
843
844     // se encarga de crear el contenido de un programa en la tabla de procesos
845     Object []objectprocesos = new Object[6];
846     objectprocesos[0]=pre; // # instancias
847     objectprocesos[1]=nombre; // nombre del programa
848     objectprocesos[2]=lNumeroLineas; // numero de lineas del programa
849     objectprocesos[3]=pivote; // rb
850     pivote=posi+1; // crea el nuevo pivote
851     objectprocesos[4]=posi; //registro limite de el programa
852     objectprocesos[5]=pivote ; // crea el rlp
853     tprocesos.addRow(objectprocesos); // adiciona a la tabla
854
855     } catch (Exception e) {
856         // retrocede el ide del programa en 1 pues el programa que lo ocupava no se cargo
857         programa--;
858         // recupera el valor de la memoria restante
859         int memoria restante=((int)memoria.getValue()- pivote);
860         // condicion que define que tipo de error surgio en el proceso
861         if (lNumeroLineas>memoria restante) {

```

```

862 // borra lo que se halla subido a la memoria si por casualidad salta un error
863 int tamaño=tabla2.getRowCount();
864 int posisi=(int) tabla2.getValueAt(tamaño-1, 5);
865 for (int i = posisi; i < (int)memoria.getValue(); i++) {
866     modelo.setValueAt("", i, 1);
867
868 }
869 //Mensaje que se muestra cuando hay error dentro del 'try'
870 JOptionPane.showMessageDialog(null, "Se generó un error al cargar el archivo \n"
871     +"pues el tamaño de este es superior a la memoria restante");
872 }else{
873     int tamaño=tabla2.getRowCount();
874     int posisi=(int) tabla2.getValueAt(tamaño-1, 5);
875     for (int i = posisi; i < (int)memoria.getValue(); i++) {
876         modelo.setValueAt("", i, 1);
877         modelo.setValueAt("", i, 2);
878         modelo.setValueAt("", i, 3);
879         modelo.setValueAt("", i, 4);
880
881
882     }
883     //Mensaje que se muestra cuando hay error dentro del 'try'
884     JOptionPane.showMessageDialog(null, "Se generó un error al cargar el archivo \n"
885         +"en la linea "+lexa+" es posible que uno de los datos del archivo \nno coincida con el formato");
886     }
887
888 }
889
890 }

```

Esta función se encarga de cargar en buffer el archivo .ch y comenzar a analizar línea por línea de este y verificar su sintaxis y si es correcta lo carga en la tabla de memoria clasifica las variables y las etiquetas y las carga a las tablas correspondientes, si todo sale bien carga el dato del programa a la tabla de procesos, si por casualidad algo no coincide saltara un error con el número de la línea q lo tiene el error de estructura del archivo.



➤ Función cargue():

```
891
892 // funcion encargada de tomar el valor de una variable y asignarselo a el acumulador
893 public void cargue(String programa, String variable){
894     int tamaño=tvariables.getRowCount();
895     int filas=0;
896     // recorre la tabla de variable en busca de la condicion
897     while(filas<tamaño){
898         // captura las variables y las caste a a cadenas
899         String prog=(String) tvariables.getValueAt(filas, 1);
900         String vari=(String) tvariables.getValueAt(filas, 3);
901         if (prog.equals(programa) && vari.equals(variable)) {
902             // agrega toda la instruccion a la memoria en el acumulador
903             modelo.setValueAt(tvariables.getValueAt(filas, 4), 0, 4);
904             filas=tamaño;
905         }
906         filas++;
907     }
908 }
909
910 }
```

Esta función se aplica a las operaciones dadas por las instrucciones del ch maquina la cual toma el valor de una variable dada y la carga en el acumulador esto lo hace dándole el número de programa y la variable a cargar en el acumulador.

➤ Función almacene():

```
911
912 // FUNCION ENCARGADA DE RECORRER LA TABLA DE VARIABLES
913 // Y ALMACENAR EL VALOR DEL ACUMULADOR EN UNA VARIABLE
914 //DADA
915 public void almacene (String programa, String variable){
916     String acumu = (String) modelo.getValueAt(0,4).toString();
917     float acumulador =Float.parseFloat(acumu);
918     int filas=0;
919     int tamaño=tvariables.getRowCount();
920     // recorre la tabla de variable en busca de la condicion
921     while(filas<tamaño){
922         String prog=(String) tvariables.getValueAt(filas, 1);
923         String vari=(String) tvariables.getValueAt(filas, 3);
924         if (prog.equals(programa) && vari.equals(variable)) {
925             int posicion= (int) tvariables.getValueAt(filas, 0);
926             // agrega toda la instruccion a la memoria en el acumulador
927             modelo.setValueAt(acumulador, posicion, 4);
928             // agrega el nuevo valor a la tabla de variables
929             tvariables.setValueAt(acumulador, filas, 4);
930
931             filas=tamaño;
932         }
933         filas++;
934     }
935 }
936 }
```

Función encargada de recorrer la tabla de variables y almacenar el valor del acumulador en una variable dada esto lo hace dándole el número de programa y la variable que capturara el contenido del acumulador.



➤ Función vaya():

```
937
938 // funcion que crea un ciclo y retorna la posicion de memoria
939 // donde debe continuar esta funcion se aplica para vaya y vayasi
940 public String vaya(String programa, String etiqueta ){
941
942     int filas=0;
943     String pos="";
944     int tamaño=tetiquetas.getRowCount();
945     // recorre la tabla de variable en busca de la condicion
946     while(filas<tamaño){
947         String prog=(String)tetiquetas.getValueAt(filas, 1);
948         String etique=(String) tetiquetas.getValueAt(filas,2 );
949         if (prog.equals(programa) && etique.equals(etiqueta)) {
950             // optiene el valor de la posicion donde debe iniciar el ciclo
951             pos = (String) tetiquetas.getValueAt(filas, 0).toString();
952             filas=tamaño;
953         }
954         filas++;
955     }
956     return pos;
957 }
```

Esta función se aplica a la condición vaya y vayasi la cual retorna una ubicación donde deberá continuar la ejecución del programa ya sea antes o después de la línea de la instrucción.

➤ Función lea():

```
960 // funcion que le pide al usuario que ingrese un valor requerido
961 //y lo retorna
962 public void lea(String programa, String variable){
963
964     int filas=0;
965     int tamaño=tvariables.getRowCount();
966     String tipo="";
967     int posicion=0,fil=0;
968     // recorre la tabla de variable en busca de la condicion
969     while(filas<tamaño){
970         String prog=(String) tvariables.getValueAt(filas, 1);
971         String vari=(String) tvariables.getValueAt(filas, 3);
972         if (prog.equals(programa) && vari.equals(variable)) {
973             // captura el tipo de la variable con la cual se busca captuara un dato
974             tipo= (String) tvariables.getValueAt(filas, 2);
975             posicion= (int) tvariables.getValueAt(filas, 0);
976             fil=filas;
977             filas=tamaño;
978         }
979         filas++;
980     }
981     //solicita el dato al usuario
982     String datodeusuario=JOptionPane.showInputDialog("INGRESE UN VALOR DE TIPO "+tipo);
983
984     // agrega el nuevo valor a la memoria
985     modelo.setValueAt(datodeusuario, posicion, 4);
986     // agrega el nuevo valor a la tabla de variables
987     tvariables.setValueAt(datodeusuario, fil, 4);
988 }
```

Esta función solicita al usuario un valor por teclado lo captura y lo almacena en la variable dada.

➤ Función sume():

```
989
990 //funcion que suma el valor del acumulador con el valor de una variable
991 public void sume(String programa, String variable){
992     String acumu = (String) modelo.getValueAt(0,4).toString();
993     float acumulador =Float.parseFloat(acumu);
994     int filas=0;
995     int tamaño=tvariables.getRowCount();
996     // recorre la tabla de variable en busca de la condicion
997     while(filas<tamaño){
998         String prog=(String) tvariables.getValueAt(filas, 1);
999         String vari=(String) tvariables.getValueAt(filas, 3);
1000         if (prog.equals(programa) && vari.equals(variable)) {
1001             // captura la posicion de memoria donde esta la variable
1002             String val=String.valueOf(tvariables.getValueAt(filas, 4));
1003             float valor= Float.parseFloat(val);
1004
1005             float resultado=acumulador + valor;
1006             // agrega el nuevo valor a la memoria en el acumulador
1007             modelo.setValueAt(resultado, 0, 4);
1008             filas=tamaño;
1009         }
1010         filas++;
1011     }
1012 }
1013 }
```

Esta función toma el valor del acumulador y el valor de la variable dada y las suma y el resultado lo almacena en el acumulador.

➤ Función reste():

```
1014
1015 //funcion que reste el valor del acumulador con el valor de una variable
1016 public void reste(String programa, String variable){
1017     String acumu = (String) modelo.getValueAt(0,4).toString();
1018     float acumulador =Float.parseFloat(acumu);
1019     int filas=0;
1020     int tamaño=tvariables.getRowCount();
1021     // recorre la tabla de variable en busca de la condicion
1022     while(filas<tamaño){
1023         String prog=(String) tvariables.getValueAt(filas, 1);
1024         String vari=(String) tvariables.getValueAt(filas, 3);
1025         if (prog.equals(programa) && vari.equals(variable)) {
1026             // captura la posicion de memoria donde esta la variable
1027             int posicion= (int) tvariables.getValueAt(filas, 0);
1028             String val=(String) tvariables.getValueAt(filas, 4);
1029             float valor= Float.parseFloat(val);
1030
1031             float resultado=acumulador - valor;
1032             // agrega el nuevo valor a la memoria en el acumulador
1033             modelo.setValueAt(resultado, 0, 4);
1034             filas=tamaño;
1035         }
1036         filas++;
1037     }
1038
1039
1040 }
```

Esta función toma el valor del acumulador y el valor de la variable dada y las resta y el resultado lo almacena en el acumulador.

➤ Función multiplique():

```
1041
1042 //funcion que multiplique el valor del acumulador con el valor de una variable
1043 public void multiplique(String programa, String variable){
1044     String acumu = (String) modelo.getValueAt(0,4).toString();
1045     float acumulador =Float.parseFloat(acumu);
1046     int filas=0;
1047     int tamaño=tvariables.getRowCount();
1048     // recorre la tabla de variable en busca de la condicion
1049     while(filas<tamaño){
1050         String prog=(String) tvariables.getValueAt(filas, 1);
1051         String vari=(String) tvariables.getValueAt(filas, 3);
1052         if (prog.equals(programa) && vari.equals(variable)) {
1053             // captura la posicion de memoria dond eesta la variable
1054             String val=String.valueOf(tvariables.getValueAt(filas, 4));
1055             float valor= Float.parseFloat(val);
1056
1057             float resultado=acumulador * valor;
1058             // agrega el nuevo valor a la memoria en el acumulador
1059             modelo.setValueAt(resultado, 0, 4);
1060             filas=tamaño;
1061         }
1062         filas++;
1063     }
1064 }
1065 }
```

Esta función toma el valor del acumulador y el valor de la variable dada y las multiplica y el resultado lo almacena en el acumulador

➤ Función divide():

```
1068 public void divide(String programa, String variable){
1069     String acumu = (String) modelo.getValueAt(0,4).toString();
1070     float acumulador =Float.parseFloat(acumu);
1071     int filas=0;
1072     int tamaño=tvariables.getRowCount();
1073     // recorre la tabla de variable en busca de la condicion
1074     while(filas<tamaño){
1075         String prog=(String) tvariables.getValueAt(filas, 1);
1076         String vari=(String) tvariables.getValueAt(filas, 3);
1077         if (prog.equals(programa) && vari.equals(variable)) {
1078             // captura la posicion de memoria donde esta la variable
1079             String val=String.valueOf(tvariables.getValueAt(filas, 4));
1080             float valor= Float.parseFloat(val);
1081             //EN CASO TAL DE QUE LA VARIABLE TENGA UN CERO VERIFICA PRIMERO
1082             if(valor!=0){
1083                 float resultado=acumulador / valor;
1084                 // agrega el nuevo valor a la memoria en el acumulador
1085                 modelo.setValueAt(resultado, 0, 4);
1086                 filas=tamaño;
1087             }else{
1088                 JOptionPane.showMessageDialog(this, "HAY DIVISION CON CERO POR TANTO"
1089                     + "EL ACUMULADOR CONSERVA SU VALOR ORIGINAL", "ALERTA X/0",
1090                     JOptionPane.INFORMATION_MESSAGE, JOptionPane.INFORMATION_MESSAGE, null, new Object[]{" OK "},"OK");
1091             }
1092         }
1093     }
1094     filas++;
1095 }
1096 }
```

Esta función toma el valor del acumulador y el valor de la variable dada y las divide y el resultado lo almacena en el acumulador en caso tal que la variable sea cero deja el acumulador tal cual.

➤ Función potencia():

```
1098 //funcion que potencia el valor del acumulador con el valor de una variable
1099 public void potencia(String programa, String variable){
1100     String acumu = (String) modelo.getValueAt(0,4).toString();
1101     float acumulador =Float.parseFloat(acumu);
1102     int filas=0;
1103     int tamaño=tvariables.getRowCount();
1104     // recorre la tabla de variable en busca de la condicion
1105     while(filas<tamaño){
1106         String prog=(String) tvariables.getValueAt(filas, 1);
1107         String vari=(String) tvariables.getValueAt(filas, 3);
1108         if (prog.equals(programa) && vari.equals(variable)) {
1109             // captura la posicion de memoria donde esta la variable
1110             String val=String.valueOf(tvariables.getValueAt(filas, 4));
1111             float valor= Float.parseFloat(val);
1112
1113             float resultado = (float) Math.pow(acumulador, valor);
1114             // agrega el nuevo valor a la memoria en el acumulador
1115             modelo.setValueAt(resultado, 0, 4);
1116             filas=tamaño;
1117         }
1118         filas++;
1119     }
1120 }
1121
1122
```

Esta función toma el valor del acumulador y lo eleva a la potencia dada por la variable y el resultado lo devuelve al acumulador.

➤ Función modulo():

```
1123 //funcion de modulo el valor del acumulador con el valor de una variable
1124 public void modulo(String programa, String variable){
1125     String acumu = (String) modelo.getValueAt(0,4).toString();
1126     float acumulador =Float.parseFloat(acumu);
1127     int filas=0;
1128     int tamaño=tvariables.getRowCount();
1129     // recorre la tabla de variable en busca de la condicion
1130     while(filas<tamaño){
1131         String prog=(String) tvariables.getValueAt(filas, 1);
1132         String vari=(String) tvariables.getValueAt(filas, 3);
1133         if (prog.equals(programa) && vari.equals(variable)) {
1134             // captura la posicion de memoria donde esta la variable
1135             String val=String.valueOf(tvariables.getValueAt(filas, 4));
1136             float valor= Float.parseFloat(val);
1137             //EN CASO TAL DE QUE LA VARIABLE TENGA UN CERO VERIFICA PRIMERO
1138             if(valor!=0){
1139                 float resultado=acumulador % valor;
1140                 // agrega el nuevo valor a la memoria en el acumulador
1141                 modelo.setValueAt(resultado, 0, 4);
1142             }else{
1143                 JOptionPane.showMessageDialog(this, "HAY DIVISION CON CERO POR TANTO"
1144                     + "EL ACUMULADOR CONSERVA SU VALOR ORIGINAL", "ALERTA X/0",
1145                     JOptionPane.INFORMATION_MESSAGE, JOptionPane.INFORMATION_MESSAGE, null, new Object[]{"OK"}, "OK");
1146                 }filas=tamaño;}
1147             filas++;
1148         }
1149     }
1150 }
1151 }
1152 }
```

Esta función toma el valor del acumulador y el valor de la variable dada y toma el módulo de las dos y el resultado lo almacena en el acumulador en caso tal que la variable sea cero deja el acumulador tal cual.

➤ Función concatene():

```
1153
1154 //funcion de concatenar el valor del acumulador con el valor de una variable
1155 public void concatene(String programa, String variable){
1156     String acumulador= (String) modelo.getValueAt(0, 4).toString();
1157     int filas=0;
1158     int tamaño=tvariables.getRowCount();
1159     // recorre la tabla de variable en busca de la condicion
1160     while(filas<tamaño){
1161         String prog=(String) tvariables.getValueAt(filas, 1);
1162         String vari=(String) tvariables.getValueAt(filas, 3);
1163         if (prog.equals(programa) && vari.equals(variable)) {
1164             // captura la posicion de memoria donde esta la variable
1165             int posicion= (int) tvariables.getValueAt(filas, 0);
1166
1167
1168             // hace la concatenacion del acumulador y el valor de la variable
1169             String resultado = acumulador + tvariables.getValueAt(filas, 4);
1170             // agrega el nuevo valor a la memoria en el acumulador
1171             modelo.setValueAt(resultado, posicion, 4);
1172             tvariables.setValueAt(resultado, filas, 4);
1173
1174             filas=tamaño;
1175         }
1176         filas++;
1177     }
1178 }
1179 }
```

Esta función concatena el valor del acumulador y el de una variable dado y lo retorna a la posición de la variable dada.



➤ Función elimine():

```
1180
1181 //funcion que elimina una parte del acumulador con el valor de una variable
1182 public void elimine(String programa, String variable){
1183     String acumulador= (String) tabla.getValueAt(0, 4),resultado="";
1184     int filas=0;
1185     int tamaño=tvariables.getRowCount();
1186     // recorre la tabla de variable en busca de la condicion
1187     while(filas<tamaño){
1188         String prog=(String) tvariables.getValueAt(filas, 1);
1189         String vari=(String) tvariables.getValueAt(filas, 3);
1190         if (prog.equals(programa) && vari.equals(variable)) {
1191             // captura la posicion de memoria donde esta la variable
1192             int posicion= (int) tvariables.getValueAt(filas, 0);
1193             String valor= (String) tvariables.getValueAt(filas, 4);
1194             // hace la eliminacion de una parte del acumulador con el valor de valor de la variable
1195             resultado = acumulador.replace(valor, "");
1196             // agrega el nuevo valor a la memoria en el acumulador
1197             modelo.setValueAt(resultado, posicion, 4);
1198             tvariables.setValueAt(resultado, filas, 4);
1199
1200             filas=tamaño;
1201         }
1202         filas++;
1203     }
1204
1205 }
```

Esta función toma el valor del acumulador y el de la variable dada y si el acumulador contiene el valor de la variable en alguna parte la elimina y retorna al resultado la cadena.

➤ Función extraiga():

```
1206
1207 //funcion que extraiga los primeros caracteres del acumulador deacuerdo con el valor de una variable
1208 public void extraiga(String programa, String variable){
1209     String acumulador= (String) tabla.getValueAt(0, 4),resultado="";
1210     int filas=0;
1211     int tamaño=tvariables.getRowCount();
1212     // recorre la tabla de variable en busca de la condicion
1213     while(filas<tamaño){
1214         String prog=(String) tvariables.getValueAt(filas, 1);
1215         String vari=(String) tvariables.getValueAt(filas, 3);
1216         if (prog.equals(programa) && vari.equals(variable)) {
1217             // captura la posicion de memoria donde esta la variable
1218             int posicion= (int) tvariables.getValueAt(filas, 0);
1219             int valor= (int) tvariables.getValueAt(filas, 4);
1220             // hace la extraccion de una parte del acumulador con el valor de la variable
1221             resultado = acumulador.substring(0, valor);
1222             // agrega el nuevo valor a la memoria en el acumulador
1223             modelo.setValueAt(resultado, posicion, 4);
1224             tvariables.setValueAt(resultado, filas, 4);
1225
1226             filas=tamaño;
1227         }
1228         filas++;
1229     }
1230
1231 }
```

Esta función toma el acumulador y un número dado por una variable y extrae la cantidad de caracteres que diga la variable del acumulador, y retorna la nueva cadena al acumulador.

➤ Función mostrar():

```
1232
1233 //funcion  mostrar en el monitor los primeros caracteres del acumulador deacuerdo con el valor de una variable
1234 public void mostrar(String programa, String variable){
1235     String resultado="";
1236     int filas=0;
1237     int tamaño=tvariables.getRowCount();
1238     // recorre la tabla de variable en busca de la condicion
1239     while(filas<tamaño){
1240         String prog=(String) tvariables.getValueAt(filas, 1);
1241         String vari=(String) tvariables.getValueAt(filas, 3);
1242         if (prog.equals(programa) && vari.equals(variable)) {
1243
1244             // captura la posicion de memoria donde esta la variable
1245
1246             String valor= (String) tvariables.getValueAt(filas, 4).toString();
1247             // hace la extraccion de una parte del  acumulador con el valor de  la variable
1248             resultado = valor;
1249             String muestra=monitor.getText();
1250             muestra=muestra+"RESULTADO DEL PROGRAMA "+ programa+".ch\nMOSTRANDO VALOR DE LA VARIABLE "+variable+" = "+resultado+"\n\n";
1251             monitor.setText(muestra);
1252
1253             break;
1254         }
1255         filas++;
1256     }
1257 }
```

Esta función muestra en el monitor el valor de la variable dada.

➤ Función imprimir():

```
1259 //funcion  mostrar en el impresora los primeros caracteres del acumulador de acuerdo con el valor de una variable
1260 public void imprimir(String programa, String variable){
1261     String resultado="";
1262     int filas=0;
1263     int tamaño=tvariables.getRowCount();
1264     // recorre la tabla de variable en busca de la condicion
1265     while(filas<tamaño){
1266         String prog=(String) tvariables.getValueAt(filas, 1);
1267         String vari=(String) tvariables.getValueAt(filas, 3);
1268         if (prog.equals(programa) && vari.equals(variable)) {
1269             // captura la posicion de memoria donde esta la variable
1270
1271             String valor= (String) tvariables.getValueAt(filas, 4).toString();
1272             // hace la extraccion de una parte del acumulador con el valor de la variable
1273             resultado = valor;
1274             String muestra=impresora.getText();
1275             muestra=muestra+"RESULTADO DEL PROGRAMA "+ programa +".ch\nMOSTRANDO VALOR DE LA VARIABLE "+variable+" = "+resultado+"\n\n";
1276
1277             impresora.setText(muestra);
1278
1279             break;
1280         }
1281         filas++;
1282     }
1283
1284 }
```

Esta función toma el valor de una variable dada y la muestra en impresora

➤ Función ejecutar():

```

1286 public void ejecutar(){
1287     // toma el valor de el kernel para iniciar a ejecutar
1288     int inicio=(int) kernel.getValue();
1289     // define el limite de las instrucciones en memoria
1290     int ultimaf= tabla2.getRowCount();
1291
1292     int limite= (int) tabla2.getValueAt(ultimaf-1, 5);
1293
1294     // ciclo encargado de recorrer todas las instrucciones en la memoria
1295     for (int i = inicio+1; i < limite; i++) {
1296         // variables capturatoras de cada fila de la tabla de memoria
1297         String pos_memoria= (String) modelo.getValueAt(i, 0).toString();
1298         String programaa= (String) modelo.getValueAt(i, 1).toString();
1299         String instruccion= (String) modelo.getValueAt(i, 2).toString();
1300         String argumento= (String) modelo.getValueAt(i, 3).toString();
1301         String valor= (String) modelo.getValueAt(i, 4).toString();
1302
1303
1304         macumulador.setText(modelo.getValueAt(0, 4).toString());
1305         mpos_mem.setText(pos_memoria);
1306         minst.setText(argumento);
1307         mvalor.setText(valor);
1308         switch (instruccion) {
1309             case "cargue":
1310                 cargue( programaa, valor);
1311                 break;
1312
1313             case "almacene":
1314                 almacene (programaa, valor);
1315                 break;

```

```

1316
1317 case "vaya":
1318     vaya(programaa, valor );
1319
1320     break;
1321
1322 case "vayasi":
1323     StringTokenizer etiquetas = new StringTokenizer(valor, ";");
1324     String inicioo = etiquetas.nextToken();
1325     String fin = etiquetas.nextToken();
1326     String continua= String.valueOf(i);
1327     float acum =(float) modelo.getValueAt(0,4);
1328     if (acum>0.0) {
1329         continua=vaya(programaa, inicioo );
1330         i=Integer.parseInt(continua)-1;
1331     }
1332     else if(acum<0.0){
1333         continua=vaya(programaa, fin );
1334         i=Integer.parseInt(continua)-1;
1335     }
1336     break;
1337
1338
1339 case "lea":
1340     lea( programaa, valor);
1341     break;
1342
1343 case "sume":
1344     sume(programaa,valor);
1345     break;
1346

```

```

1346
1347     case "reste":
1348         reste(programaa, valor);
1349         break;
1350
1351     case "multiplique":
1352         multiplique(programaa, valor);
1353         break;
1354
1355     case "divida":
1356         divide(programaa, valor);
1357         break;
1358
1359     case "potencia":
1360         potencia(programaa, valor);
1361         break;
1362
1363     case "modulo":
1364         modulo(programaa, valor);
1365         break;
1366
1367     case "concatene":
1368         concatene(programaa, valor);
1369         break;
1370
1371     case "elimine":
1372         elimine(programaa, valor);
1373         break;
1374
1375     case "extraiga":
1376         extraiga(programaa, valor);

```



```

1377         break;
1378
1379         case "muestre":
1380             mostrar(programaa,valor);
1381             break;
1382
1383         case "imprima":
1384             imprimir(programaa,valor);
1385             break;
1386
1387         case "retorne":
1388             modelo.setValueAt(0, 0, 4);
1389
1390             break;
1391
1392
1393
1394
1395     }
1396
1397 }
1398
1399
1400 }
1401

```

Esta funcion es la encargada de recorrer el mapa de memoria y según la instrucción invoca otra funcion y le envia parametrs para que los opere y entregue un resultado.

➤ Función pasoapaso():

```
1403 public void pasoapaso(){
1404     // toma el valor de el kernel para iniciar a ejecutar
1405     int inicio=(int) kernel.getValue();
1406     // define el limite de las instrucciones en memoria
1407     int ultimaf= tabla2.getRowCount();
1408
1409     int limite= (int) tabla2.getValueAt(ultimaf-1, 5);
1410     int i = inicio+1;
1411     // ciclo encargado de recorrer todas las instrucciones en la memoria
1412     while ( i < limite) {
1413
1414         // variables capturatoras de cada fila de la tabla de memoria
1415         String pos_memoria= (String) modelo.getValueAt(i, 0).toString();
1416         String programaa= (String) modelo.getValueAt(i, 1).toString();
1417         String instruccion= (String) modelo.getValueAt(i, 2).toString();
1418         String argumento= (String) modelo.getValueAt(i, 3).toString();
1419         String valor= (String) modelo.getValueAt(i, 4).toString();
1420
1421         // muestra en la interfaz los procesos que se estan ejecutando
1422         macumulador.setText(modelo.getValueAt(0, 4).toString());
1423         mpos_mem.setText(pos_memoria);
1424         minst.setText(argumento);
1425         mvalor.setText(valor);
1426         switch (instruccion) {
1427             case "cargue":
1428                 cargue( programaa, valor);
1429                 break;
1430
1431             case "almacene":
```

```

1432     almacene (programaa, valor);
1433     break;
1434
1435     case "vaya":
1436         vaya(programaa, valor );
1437
1438         break;
1439
1440     case "vayasi":
1441         StringTokenizer etiquetas = new StringTokenizer(valor, ";");
1442         String inicioo = etiquetas.nextToken();
1443         String fin = etiquetas.nextToken();
1444         String continua= String.valueOf(i);
1445         float acum =(float) modelo.getValueAt(0,4);
1446         if (acum>0.0) {
1447
1448             continua=vaya(programaa, inicioo );
1449             i=Integer.parseInt(continua)-1;
1450
1451         }
1452         else if(acum<0.0){
1453             continua=vaya(programaa, fin );
1454             i=Integer.parseInt(continua)-1;
1455         }
1456         break;
1457
1458     case "lea":
1459         lea( programaa, valor);
1460         break;
1461

```

```
1461
1462     case "sume":
1463         sume (programaa, valor);
1464         break;
1465
1466     case "reste":
1467         reste (programaa, valor);
1468         break;
1469
1470     case "multiplique":
1471         multiplique (programaa, valor);
1472         break;
1473
1474     case "divida":
1475         divide (programaa, valor);
1476         break;
1477
1478     case "potencia":
1479         potencia (programaa, valor);
1480         break;
1481
1482     case "modulo":
1483         modulo (programaa, valor);
1484         break;
1485
1486     case "concatene":
1487         concatene (programaa, valor);
1488         break;
1489
1490     case "elimine":
1491         elimine (programaa, valor);
```

```

1492         break;
1493
1494         case "extraiga":
1495             extraiga(programaa,valor);
1496             break;
1497
1498         case "muestre":
1499             mostrar(programaa,valor);
1500             break;
1501
1502         case "imprima":
1503             imprimir(programaa,valor);
1504             break;
1505
1506         case "retorne":
1507             modelo.setValueAt(0, 0, 4);
1508
1509             break;
1510
1511     }
1512     if(JOptionPane.showOptionDialog(this, "?DESEA SEGUIR LA EJECUCION PASO A PASO?", "Mensaje de Alerta",
1513         JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE, null, new Object[]{" SI ", " NO "}, "NO")==0)
1514     {
1515         i++;
1516     } else
1517     {
1518         ejecutar();
1519         i=limite;
1520     }
1521 }
1522 }

```

Esta funcion es la encargada de recorrer el mapa de memoria y según la instrucción invoca otra funcion y le envia parametrsos para que los opere y entregue un resultado pero por cada instrucción que lea le pedira al usuario que si desea seguir viendo la siguiente pero si el usuario le dice que no invoca ejecutar() y termina el proceso.

➤ Función initComponents():

```
1524
1525 /**
1526  * This method is called from within the constructor to initialize the form.
1527  * WARNING: Do NOT modify this code. The content of this method is always
1528  * regenerated by the Form Editor.
1529  */
1530 @SuppressWarnings("unchecked")
1531 // componentes de la interfaz del ch-maquina
1532 // <editor-fold defaultstate="collapsed" desc="Generated Code">
1533 private void initComponents() {...543 lines} // </editor-fold>
2076
```

Esta función tiene encapsulado todas las variables y procedimientos instanciados para interfaz gráfica y es la primera función llamada en el primer arranque del programa.

#### 4. eventos por botones:

Son aquellas instrucciones ejecutadas al activarse un botón en la interfaz.

➤ evento de botón cargarprogramaActionPerformed():

```
2077 private void cargarprogramaActionPerformed(java.awt.event.ActionEvent evt) {
2078     // TODO add your handling code here:
2079     cargararchivo();
2080 }
2081
```

Este evento se encarga de abrir un panel de exploración de archivos para realizar la carga del .ch y ser compilado, este tiene la condición que solo detecta los archivos con extensión .ch, este evento es activado por el botón cargar programa del panel de archivo o por el comando ctrl+ o .

➤ Evento de botón memoriaStateChanged() , kernelStateChanged():

```
2088 private void memoriaStateChanged(javax.swing.event.ChangeEvent evt) {
2089     // TODO add your handling code here:
2090     // en caso tal de que el kernel supere el tamaño de la memoria la memoria se modificara en 1 mas que el kernel
2091     int kertemp = (int) kernel.getValue();
2092     int memtemp = (int) memoria.getValue();
2093     if (kertemp >= memtemp) {
2094         memtemp=kertemp+1;
2095         memoria.setValue(memtemp);
2096     }
2097     //muestra la memoria disponible para los programas
2098     mentotal();
2099 }
2100
2101 private void kernelStateChanged(javax.swing.event.ChangeEvent evt) {
2102     // en caso tal de que el kernel supere el tamaño de la memoria la memoria se modificara en 1 mas que el kernel
2103     int kertemp = (int) kernel.getValue();
2104     int memtemp = (int) memoria.getValue();
2105     if (memtemp <= kertemp) {
2106         kertemp=memtemp-1;
2107         kernel.setValue(kertemp);
2108     }
2109     //muestra la memoria disponible para los programas
2110     mentotal();
2111 }
2112 }
2113
```

Estos dos eventos se encargan de mantener coherencia en la memoria donde el kernel nunca va poder ser superior o igual a la memoria y la memoria nunca podrá ser menor o igual al kernel.

- Evento de botón `encenderActionPerformed()`, `apagarmaquina2ActionPerformed()`, `encender2ActionPerformed()`, `apagarmaquina1ActionPerformed()`:

```
2113
2114 private void encenderActionPerformed(java.awt.event.ActionEvent evt) {
2115     // HACE EL LLAMADO A LA FUNCION PARA QUE REPRODUzca EL SONIDO DE ENSENDIDO
2116     encender();
2117
2118
2119 }
2120
2157
2158 private void apagarmaquina2ActionPerformed(java.awt.event.ActionEvent evt) {
2159     // TODO add your handling code here:
2160
2161     //llama la funcion encargada de apagar la maquina
2162     apagar();
2163
2164
2165
2166 }
2167
2168 private void encender2ActionPerformed(java.awt.event.ActionEvent evt) {
2169     // HACE EL LLAMADO A LA FUNCION PARA QUE REPRODUzca EL SONIDO DE ENSENDIDO
2170     encender();
2171 }
2172
2173 private void apagarmaquina1ActionPerformed(java.awt.event.ActionEvent evt) {
2174     // TODO add your handling code hermie:
2175     //llama la funcion encargada de apagar la maquina
2176     apagar();
2177 }
```

Estos eventos se encargan de llamar las funciones de encender y apagar la maquina son activados por los botones del mismo nombre.



➤ Evento de botón jMenuItem3ActionPerformed():

```
2132 private void jMenuItem3ActionPerformed(java.awt.event.ActionEvent evt) {  
2133  
2134     // codigo encargado de CERRAR EL PROGRAMA  
2135  
2136     temporales();// borra los temporales  
2137     if(JOptionPane.showOptionDialog(this, "¿ESTA SEGURO QUE DESEA SALIR DEL PROGRAMA?", "Mensaje de Alerta",  
2138         JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE, null, new Object[]{"SI ", "NO "}, "NO")==0)  
2139     {  
2140         son("cierre");  
2141  
2142         // se encarga de detener un instante el proceso  
2143         try {  
2144             Thread.sleep(3000);// el tiempo es en milisegundos  
2145         } catch (InterruptedException ex) {  
2146  
2147         }  
2148         System.exit(0);  
2149     }  
2150  
2151     else  
2152     {  
2153         JOptionPane.showMessageDialog(this, "PUEDE CONTINUAR CON LA EJECUCION DEL PROGRAMA");  
2154     }  
2155  
2156 }  
2157
```

Se encarga de hacer el cierre definitivo del programa pero tiene la condición si o no para comprobar la decisión del usuario además de activar un sonido de cierre de sesión.

➤ Evento de botón `acercadeActionPerformed()`;

```
2183 private void acercadeActionPerformed(java.awt.event.ActionEvent evt) {  
2184     // TODO add your handling code here  
2185     JOptionPane.showMessageDialog(this, "Product Version: MI CH-MAQUINA V.1.0.0\n" +  
2186         "Actualizaciones: en proceso...\n" +  
2187         "Java: 1.7.0_51; Java HotSpot(TM) 64-Bit Server VM 24.51-b03\n" +  
2188         "Runtime: Java(TM) SE Runtime Environment 1.7.0_51-b13\n" +  
2189         "System recomendado: Windows 7 y posterior\n" +  
2190         "creado por :YEISON AGUIRRE OSORIO -- NAUFRAGO\n" +  
2191         "UNIVERSIDAD DE COLOMBIA - SEDE MANIZALES\n"  
2192         + "fecha creacion:febrero 2016\n\n"  
2193         + "simulador de OS encargado de leer instrucciones de un archivo con \n"  
2194         + "extencion .CH en este estan los pasos y valores iniciales que el \n"  
2195         + "simulador debe interpretar y ejecutar, lo puede hacer de modo recorrido o \n"  
2196         + "paso a paso, durante la ejecucion se ve el mapa de memoria y que hay \n"  
2197         + "almacenado en ella ademas de el cuador de procesos activos y variables declaradas,\n"  
2198         + "los resultados del proceso pueden visualizarcen en monitor e impresion.", "ACERCA DE MI CH-MAQUINA.",  
2199     JOptionPane.INFORMATION_MESSAGE, JOptionPane.INFORMATION_MESSAGE, null, new Object[]{" OK "},"OK");  
2200 }  
2201
```

Este evento se encarga de mostrar una descripción del programa así como su versión programador plataforma fecha de creación y sistema recomendado para ejecución este se puede activar por comando f1 o por el botón acerca de mí del panel ayuda.

➤ Evento de botón encender3ActionPerformed (),encender4ActionPerformed();

```
2201
2202 private void encender3ActionPerformed(java.awt.event.ActionEvent evt) {
2203     // boton ejecutar
2204     botoncargar.setVisible(false);
2205     estado.setText("MODULO USUARIO");
2206     ejecutar();
2207     // TODO add your handling code here:
2208 }
2209
2210 private void encender4ActionPerformed(java.awt.event.ActionEvent evt) {
2211     //boton paso a paso
2212     botoncargar.setVisible(false);
2213     estado.setText("MODULO USUARIO");
2214     pasoapaso();
2215     // TODO add your handling code here:
2216 }
2217
```

Estos eventos se encargan de poner en ejecución el núcleo del programa recorrer la memoria y ejecutar las instrucciones uno de recorrido y el otro paso a paso.

➤ Evento de botón `documentacionActionPerformed ()`;

```
2218 private void documentacionActionPerformed(java.awt.event.ActionEvent evt) {  
2219     // TODO add your handling code here:  
2220     // carga el documento que contiene la documentacion  
2221  
2222     try{  
2223         //nuevo archivo en esa direccion  
2224         File temp = new File(arc);  
2225         InputStream is = this.getClass().getResourceAsStream("/documentacion/documentacion.pdf");  
2226         FileOutputStream archivoDestino = new FileOutputStream(temp);  
2227         //FileWriter fw = new FileWriter(temp);  
2228         byte[] buffer = new byte[1024*1024];  
2229         //lee el archivo hasta que se acabe...  
2230         int nbLectura;  
2231         while ((nbLectura = is.read(buffer)) != -1)  
2232             archivoDestino.write(buffer, 0, nbLectura);  
2233         //cierras el archivo, el inputS y el FileW  
2234         //fw.close();  
2235         archivoDestino.close();  
2236         is.close();  
2237         //abres el archivo temporal  
2238         Desktop.getDesktop().open(temp);  
2239     } catch (IOException ex) {  
2240     }  
2241 }  
2242 }  
2243 }
```

Este evento abre la documentación del programa para que sea leída por el usuario.

➤ Evento de botón `botoncargarActionPerformed ()`;

```
2245 private void botoncargarActionPerformed(java.awt.event.ActionEvent evt) {  
2246     // TODO add your handling code here:  
2247     ejecutar.setVisible(true);  
2248     IMP.setEnabled(true);  
2249     EJEC.setEnabled(true);  
2250  
2251     pasoapaso.setVisible(true);  
2252     cargararchivo();  
2253 }  
2254 }
```

Evento que activa algunos botones desactivados como imprimir ejecutar paso a paso y hace el llamado a la función `cargararchivo()`.

➤ Evento de botón ejecutarActionPerformed ();

```
2254
2255 private void ejecutarActionPerformed(java.awt.event.ActionEvent evt) {
2256     // TODO add your handling code here:
2257     botoncargar.setVisible(false);
2258     estado.setText("MODULO USUARIO");
2259     ejecutar();
2260 }
2261
```

Evento que bloquea el botón cargar y hace el llamado a la función ejecutar().

➤ Evento de botón editorActionPerformed ();

```
2262
2263 private void editorActionPerformed(java.awt.event.ActionEvent evt) {
2264     // TODO add your handling code here:
2265     ambiente des= new ambiente();
2266     des.setVisible(true);
2267 }
2268
2269
2270
```

Evento que abre la ventana del editor.

➤ Evento de botón pasoapasoActionPerformed ();

```
2270
2271 private void pasoapasoActionPerformed(java.awt.event.ActionEvent evt) {
2272     botoncargar.setVisible(false);
2273     estado.setText("MODULO USUARIO");
2274     pasoapaso();
2275     // TODO add your handling code here:
2276 }

```

Evento que desactiva el botón de cargar y ejecuta la función pasoapaso().

➤ Evento de botón IMPRIActionPerformed ();

```
2282
2283 private void IMPRIActionPerformed(java.awt.event.ActionEvent evt) {
2284     // TODO add your handling code here:
2285     //se encarga de imprimir el contenido de la impresora
2286     try {
2287
2288         impresora.print();
2289     } catch (PrinterException ex) {
2290
2291     }
2292 }
```

Evento que ejecuta el comando imprimir y toma el contenido de la impresora y la imprime.

➤ Paquete chmaquina clase ambiente.java:

En este punto se dará una explicación concisa de partes del código encargadas de ciertas funcionalidades de la clase entrada.java.

1. Librerías importadas:

```
8
9 import java.io.BufferedReader;
10 import java.io.BufferedWriter;
11 import java.io.File;
12 import java.io.FileReader;
13 import java.io.FileWriter;
14 import java.io.PrintWriter;
15 import javax.swing.ImageIcon;
16 import javax.swing.JFileChooser;
17 import javax.swing.JLabel;
18 import javax.swing.JLayeredPane;
19 import javax.swing.JOptionPane;
20 import javax.swing.JPanel;
21 import javax.swing.filechooser.FileNameExtensionFilter;
22
```

Librerías encargadas de interactuar con el código facilitando variedad de operaciones y funciones, tanto en la construcción como en la implementación de código.

## 2. Clase ambiente

```
32 String ruta = "";
33 String nombrea="";
34 entrada archivo=new entrada();
35 int programa=archivo.programa;
36
37 public ambiente() {
38     initComponents();
39     setLocationRelativeTo(null);
40     setResizable(false);
41     setTitle("MI CH-MAQUINA EDITOR");
42     setIconImage(new ImageIcon(getClass().getResource("/imagenes/icono.png")).getImage()); // icono d ela ventana del programa
43
44     //fundamento encargado de la imagen de fondo del ch-maquina
45     ((JPanel) getContentPane()).setOpaque(false);
46     ImageIcon uno=new ImageIcon(this.getClass().getResource("/imagenes/fon2.jpg"));
47     JLabel fondo= new JLabel();
48     fondo.setIcon(uno);
49     getLayeredPane().add(fondo,JLayeredPane.FRAME_CONTENT_LAYER);
50     fondo.setBounds(0,0,uno.getIconWidth(),uno.getIconHeight());
51
52 }
53
```

Podemos identificar la función principal **ambiente()**; esta se encarga de inicializar todos los componentes de arranque de la interfaz gráfica del editor inicializando imágenes títulos entre otros y asignándoles valores por defecto , los cuales durante la implementación pueden cambiar.

## 3. Funciones:

Encargadas de ejecutar instrucciones dadas por el usuario.

➤ Función guardar():

```
54
55 public void guardar(){
56     javax.swing.JFileChooser jF1= new javax.swing.JFileChooser();
57     ruta = "";
58     try{
59         if(jF1.showSaveDialog(null)==jF1.APPROVE_OPTION){
60             ruta = jF1.getSelectedFile().getAbsolutePath();
61             //Aqui ya tienes la ruta,,ahora puedes crear un fichero en esa ruta y escribir lo que quieras...
62             String text = panel.getText();
63             nombre=jF1.getName();
64             String nombreArchivo= ruta; // Aqui se le asigna el nombre
65
66             FileWriter file = null; // la extension al archivo
67             try {
68                 file = new FileWriter(nombreArchivo);
69                 BufferedWriter escribir = new BufferedWriter(file);
70                 PrintWriter archivo = new PrintWriter(escribir);
71
72                 archivo.print(text);
73                 archivo.close();
74
75                 JOptionPane.showMessageDialog(null,"SE A CREADO EL NUEVO ARCHIVO ");
76             }
77             catch (Exception e) {
78             }
79         }
80     }catch (Exception ex){
81         ex.printStackTrace();
82     }
83 }
```

Esta función captura el contenido del editor y lo guarda en un archivo .ch, le pide al usuario la ubicación donde lo quiere guardar.



➤ Función cargararchivo():

```
85
86 public void cargararchivo(){
87     // encargado de abrir el panel de busqueda de archivos y cargarlo a la funcion actualizar.
88     JFileChooser ventana = new JFileChooser();
89     // filtra las extensiones segun la que buscamos
90     ventana.setFileFilter(new FileNameExtensionFilter("todos los archivos "
91     + "*.ch", "CH", "ch"));
92     int sel = ventana.showOpenDialog(ambiente.this);
93     // condicional que le dara el nombre a el programa
94     String prefijo;
95     if (programa<10) {
96         prefijo="000"+String.valueOf(programa);
97         programa++;
98     }else{
99         prefijo="00"+String.valueOf(programa);
100     }
101
102     if (sel == JFileChooser.APPROVE_OPTION) {
103
104         File file = ventana.getSelectedFile();
105
106         String nombrea=file.getName();
107         actualizar(file.getPath(), prefijo,nombrea );
108
109     }
110 }
```

Esta función le pide al usuario la ubicación de un archivo.ch para ser montado en el editor y pueda ser modificado y luego guardado para ser utilizado luego.

➤ Función actualizar():

```
111
112 // funcion encargada de leer el archivo y hacer el token
113 public void actualizar(String url, String pre,String nombre){
114
115     try{
116
117         // lee el archivo y lo carga en bufer
118         FileReader file = new FileReader(url);
119         BufferedReader leer = new BufferedReader(file);
120         String sCadena;
121         String contenido="";
122         // CICLO QUE RECORRE CADA LINEA HASTA QUE LA LINEA SEA NULL
123         while ((sCadena = leer.readLine())!=null) {
124             // crea el contenido del .ch
125             contenido=contenido+sCadena+"\r\n";
126         }
127         panel.setText(contenido);// ingresa el contenido del archivo al panel
128         contenido="";
129     }
130     catch (Exception ex){
131         ex.printStackTrace();
132     }
133
134 }
135
```

Esta función toma el contenido de un archivo que el usuario quiere editar y lo monta sobre el editor para ser modificado.

➤ Función initComponents():

```
135
136  /**
137   * This method is called from within the constructor to initialize the form.
138   * WARNING: Do NOT modify this code. The content of this method is always
139   * regenerated by the Form Editor.
140   */
141  @SuppressWarnings("unchecked")
142  // <editor-fold defaultstate="collapsed" desc="Generated Code">
143  private void initComponents() { ...81 lines } // </editor-fold>
224
```

Esta función tiene encapsulado todas las variables y procedimientos instanciados para interfaz gráfica y es la primera función llamada en el primer arranque del programa.

#### 4. Eventos por botones:

Son aquellas instrucciones ejecutadas al activarse un botón en la interfaz.

➤ evento de botón guardarActionPerformed(), limpiarActionPerformed(), terminarActionPerformed(), cargarActionPerformed():

```
225 private void guardarActionPerformed(java.awt.event.ActionEvent evt) {
226     // TODO add your handling code here:
227     // boton encargado de guardar el archivo editado
228     guardar();
229 }
230
231 private void limpiarActionPerformed(java.awt.event.ActionEvent evt) {
232     // boton encargado de limpiar el editor
233     panel.setText("");
234 }
235
236 private void terminarActionPerformed(java.awt.event.ActionEvent evt) {
237     //boton encargado de cerrar el editor
238     dispose();
239 }
240
241 private void cargarActionPerformed(java.awt.event.ActionEvent evt) {
242     //boton encargado de cargar un archivo a editar.
243     cargararchivo();
244     // TODO add your handling code here:
245 }
246
```

Eventos encargados de cargar archivos al editor guardarlos o limpiar el editor o solo cerrar el editor para regresar a la interfaz principal.