

Documentación CH-MAUINA

Requerimientos Clases métodos
funciones

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:

1. ÍNDICE:	2
2. INTRODUCCIÓN Y DESCRIPCIÓN DEL PROGRAMA	3
3. OBJETO Y ASPECTOS PRINCIPALES DEL PROGRAMA.	3
4. QUE HACE EL CH-MÁQUINA Y CUÁL ES SU SINTAXIS.	4
Operación Descripción sintaxis:	4
Ejecución del programa:	6
5. CÓDIGO COMPLETO Y EXPLICACIÓN DE FUNCIONAMIENTO DEL PROGRAMA.	7
➤ Estructura del proyecto chmaquina:	8
➤ Paquete chmaquina clase entrada.java:	8
1. Librerías importadas:	9
2. Clase entrada	10
3. Funciones:	14
➤ Función memtotal():	14
➤ Función son():	14
➤ Función encender():	15
➤ Función apagar():	16
➤ Función initComponets():	16
4. eventos por botones:	17
➤ evento de botón cargarprogramaActionPerformed():	17
➤ Evento de botón memoriaStateChanged() , kernelStateChanged():	17
➤ Evento de botón encenderActionPerformed(), apagarmaquina2ActionPerformed(), encender2ActionPerformed(), apagarmaquina1ActionPerformed():	18
➤ Evento de botón jMenuItem3ActionPerformed():	19
➤ Evento de botón acercadeActionPerformed();	19

2. 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-MAUINA 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.

3. 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 * ultimo 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).

4. 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** Cárgetse/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 suma 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 manejarán 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.

5. 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:**

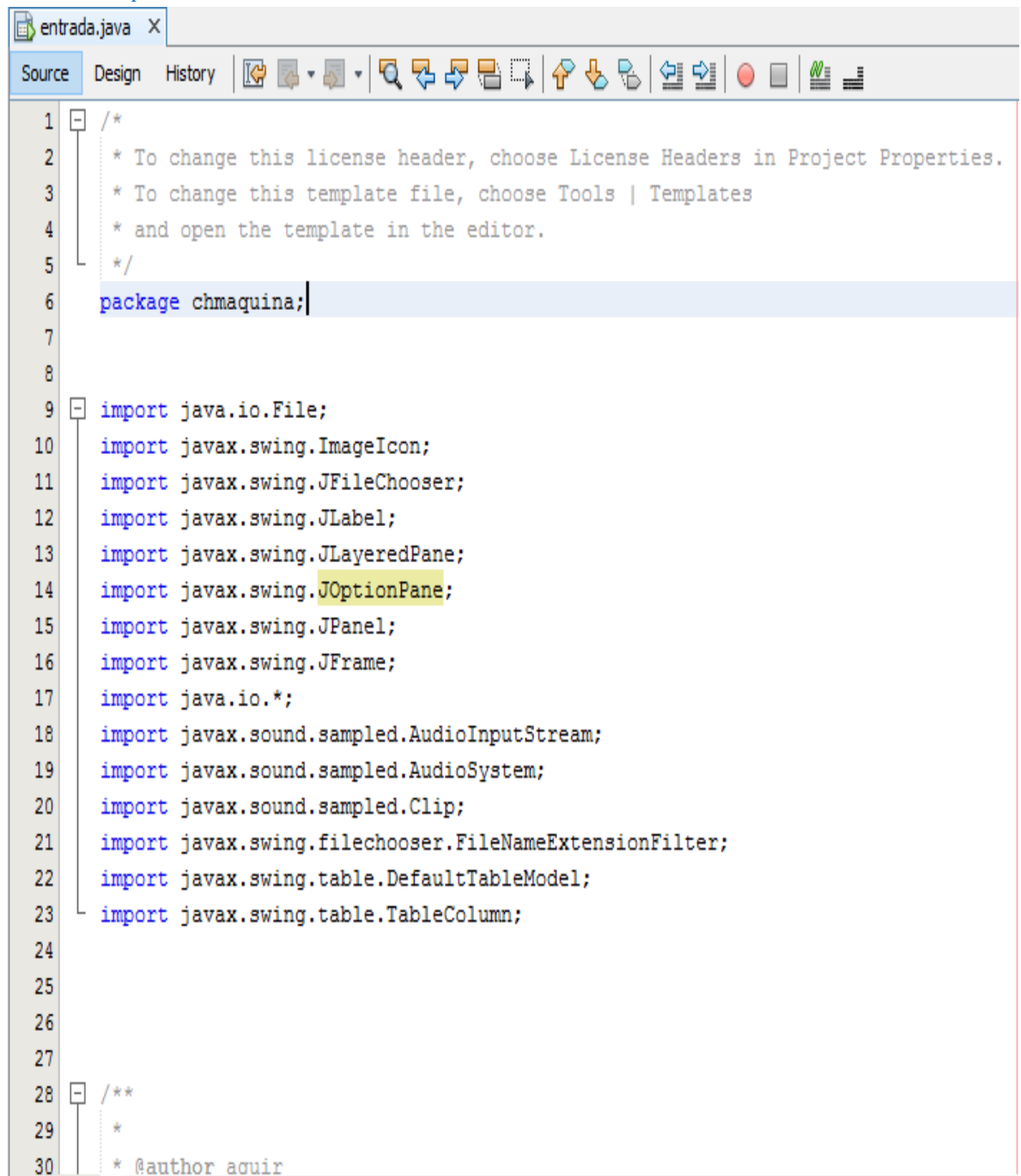


El proyecto en su estructura tiene 3 paquetes contenedores que son audio(contiene todos los archivos de sonido utilizados en el proyecto, estos son de extensión .wav), imágenes(contiene todas las imágenes de múltiples tipos utilizadas en el proyecto), y chmaquina contenedor de las clases.

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

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

1. Librerías importadas:



```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package chmaquina;
7
8
9  import java.io.File;
10 import javax.swing.ImageIcon;
11 import javax.swing.JFileChooser;
12 import javax.swing.JLabel;
13 import javax.swing.JLayeredPane;
14 import javax.swing.JOptionPane;
15 import javax.swing.JPanel;
16 import javax.swing.JFrame;
17 import java.io.*;
18 import javax.sound.sampled.AudioInputStream;
19 import javax.sound.sampled.AudioSystem;
20 import javax.sound.sampled.Clip;
21 import javax.swing.filechooser.FileNameExtensionFilter;
22 import javax.swing.table.DefaultTableModel;
23 import javax.swing.table.TableColumn;
24
25
26
27
28 /**
29  *
30  * @author aguir
```

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

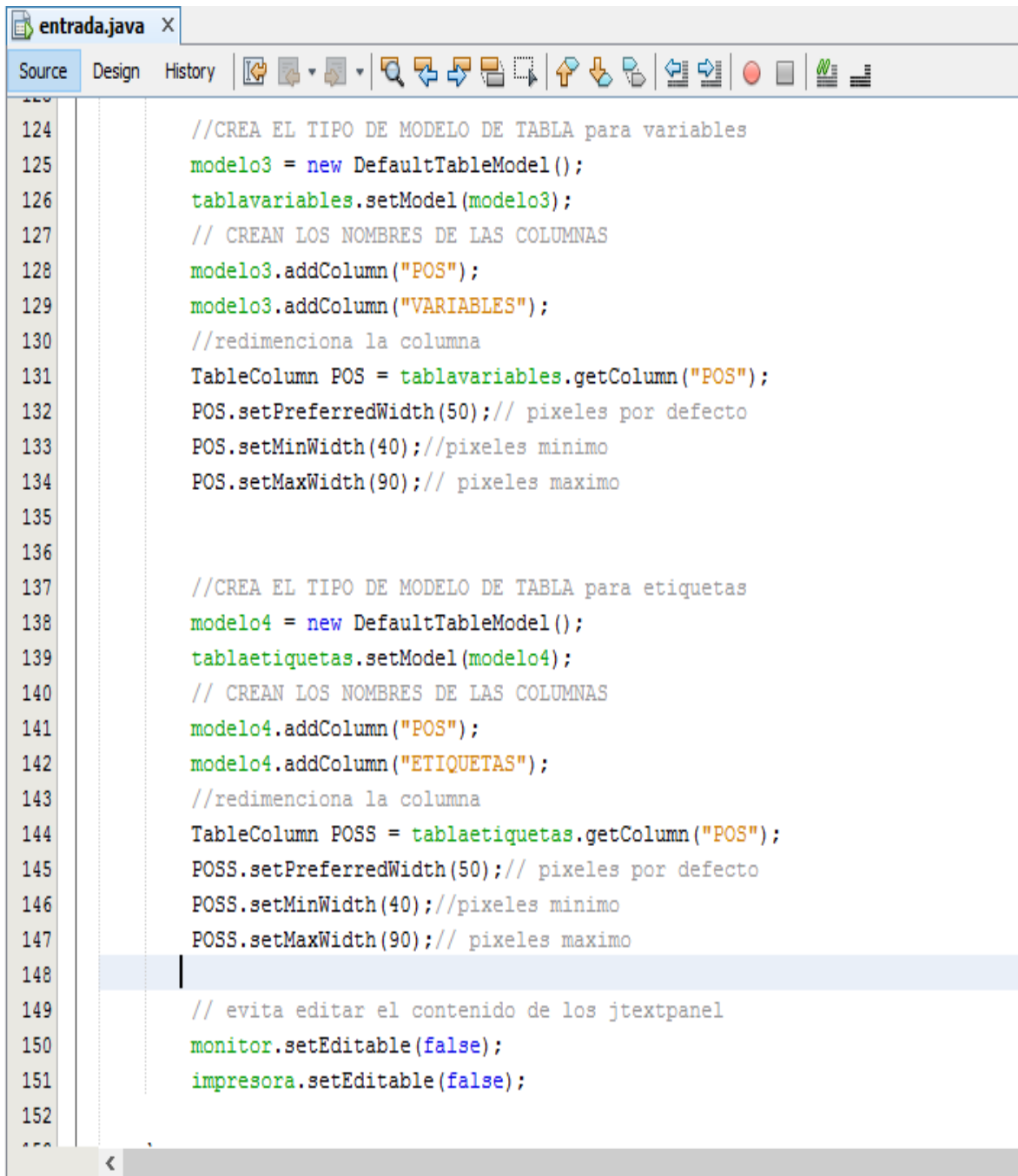
```
entrada.java X
Source Design History
32 public class entrada extends JFrame {
33
34     /**
35      * Creates new form entrada
36      */
37     DefaultTableModel modelo, modelo2, modelo3, modelo4;
38
39     public entrada() {
40         initComponents();
41         setLocationRelativeTo(null);
42         setResizable(true); // permite que la ventana principal se pueda maximizar o minimizar
43         setExtendedState(JFrame.MAXIMIZED_BOTH); // hace que la ventana siempre aparezca maximizada
44         setTitle("MI CH-MAQUINA"); // titulo del programa
45         setIconImage(new ImageIcon(getClass().getResource("/imagenes/icono.png")).getImage()); // icono de la ventana del programa
46         cargarprograma.setEnabled(false); // impide cargar programas sin prender maquina
47         // impide apagar la maquina sin encenderla
48         apagarmaquina1.setEnabled(false);
49         apagarmaquina2.setEnabled(false);
50
51         //fundamento encargado de la imagen de fondo del ch-maquina
52         ((JPanel) getContentPane()).setOpaque(false);
53         ImageIcon uno = new ImageIcon(this.getClass().getResource("/imagenes/fon.jpg"));
54         JLabel fondo = new JLabel();
55         fondo.setIcon(uno);
56         getLayeredPane().add(fondo, JLayeredPane.FRAME_CONTENT_LAYER);
57         fondo.setBounds(0, 0, uno.getWidth(), uno.getHeight());
58
59
60
61
```

```
entrada.java x
Source Design History
62 //definen los valores por defecto de la memoria del kernel y la memoria disponible para programas
63 int maxmemo=9999, maxkerner=1000;
64 memoria.setModel(new javax.swing.SpinnerNumberModel(100, 2, maxmemo, 1));
65 kernel.setModel(new javax.swing.SpinnerNumberModel(29, 1, maxkerner, 1));
66 total_memoria.setText("71");
67
68
69 //CREA EL TIPO DE MODELO DE TABLA para mapa de memoria
70 modelo = new DefaultTableModel();
71 tabla.setModel(modelo);
72 // CREA LOS NOMBRES DE LAS COLUMNAS
73 modelo.addColumn("POS-MEMO");
74 modelo.addColumn("INSTRUCCIONES");
75 //redimensiona la columna
76 TableColumn columna = tabla.getColumnModel("POS-MEMO");
77 columna.setPreferredWidth(80); // pixeles por defecto
78 columna.setMinWidth(50); // pixeles minimo
79 columna.setMaxWidth(90); // pixeles maximo
80
81 //CREA EL TIPO DE MODELO DE TABLA para procesos
82 modelo2 = new DefaultTableModel();
83 tabla2.setModel(modelo2);
84 // CREA LOS NOMBRES DE LAS COLUMNAS
85 modelo2.addColumn("ID");
86 modelo2.addColumn("PROGRAMAS");
87 modelo2.addColumn("#INST");
88 modelo2.addColumn("RB");
89 modelo2.addColumn("RCL");
90 modelo2.addColumn("RLP");
91
```



The screenshot shows an IDE window titled 'entrada.java'. The interface includes a 'Source' tab, a 'Design' tab, and a 'History' tab. Below these tabs is a toolbar with various icons for editing and navigation. The main area displays Java code for configuring table columns. The code is as follows:

```
92 //redimenciona la columna
93 TableColumn id = tabla2.getColumn("ID");
94 id.setPreferredWidth(40);// pixeles por defecto
95 id.setMinWidth(10);//pixeles minimo
96 id.setMaxWidth(41);// pixeles maximo
97
98 TableColumn pro = tabla2.getColumn("PROGRAMAS");
99 pro.setPreferredWidth(100);// pixeles por defecto
100 pro.setMinWidth(10);//pixeles minimo
101 pro.setMaxWidth(501);// pixeles maximo
102
103 TableColumn ins = tabla2.getColumn("#INST");
104 ins.setPreferredWidth(50);// pixeles por defecto
105 ins.setMinWidth(10);//pixeles minimo
106 ins.setMaxWidth(51);// pixeles maximo
107
108 TableColumn rb = tabla2.getColumn("RB");
109 rb.setPreferredWidth(40);// pixeles por defecto
110 rb.setMinWidth(10);//pixeles minimo
111 rb.setMaxWidth(41);// pixeles maximo
112
113 TableColumn rcl = tabla2.getColumn("RCL");
114 rcl.setPreferredWidth(40);// pixeles por defecto
115 rcl.setMinWidth(10);//pixeles minimo
116 rcl.setMaxWidth(41);// pixeles maximo
117
118 TableColumn rlp = tabla2.getColumn("RLP");
119 rlp.setPreferredWidth(40);// pixeles por defecto
120 rlp.setMinWidth(10);//pixeles minimo
121 rlp.setMaxWidth(41);// pixeles maximo
```

The image shows a screenshot of a Java IDE window titled 'entrada.java'. The window has a menu bar with 'Source', 'Design', and 'History'. Below the menu bar is a toolbar with various icons for code editing and navigation. The main area displays Java code with line numbers from 124 to 152. The code defines two table models, 'modelo3' and 'modelo4', and sets their column names and widths. It also sets the 'editable' property of 'monitor' and 'impresora' to false.

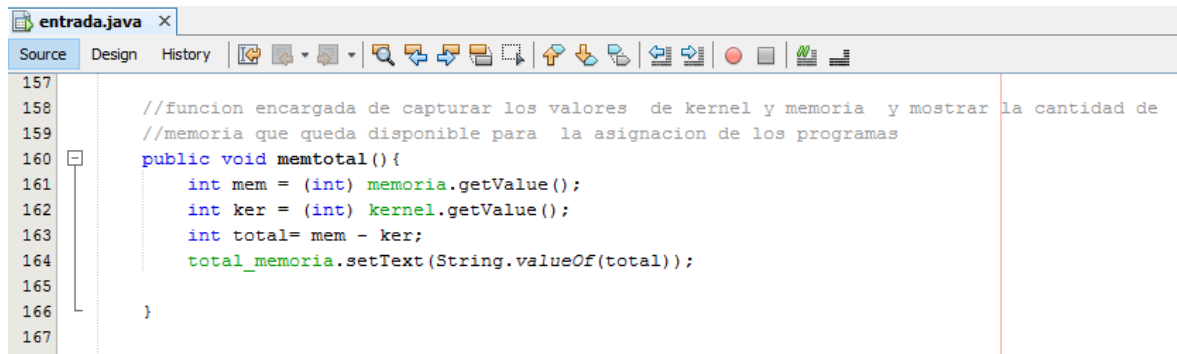
```
124 //CREA EL TIPO DE MODELO DE TABLA para variables
125 modelo3 = new DefaultTableModel();
126 tablavariables.setModel(modelo3);
127 // CREA LOS NOMBRES DE LAS COLUMNAS
128 modelo3.addColumn("POS");
129 modelo3.addColumn("VARIABLES");
130 //redimenciona la columna
131 TableColumn POS = tablavariables.getColumn("POS");
132 POS.setPreferredWidth(50); // pixeles por defecto
133 POS.setMinWidth(40); //pixeles minimo
134 POS.setMaxWidth(90); // pixeles maximo
135
136
137 //CREA EL TIPO DE MODELO DE TABLA para etiquetas
138 modelo4 = new DefaultTableModel();
139 tablaetiquetas.setModel(modelo4);
140 // CREA LOS NOMBRES DE LAS COLUMNAS
141 modelo4.addColumn("POS");
142 modelo4.addColumn("ETIQUETAS");
143 //redimenciona la columna
144 TableColumn POSS = tablaetiquetas.getColumn("POS");
145 POSS.setPreferredWidth(50); // pixeles por defecto
146 POSS.setMinWidth(40); //pixeles minimo
147 POSS.setMaxWidth(90); // pixeles maximo
148
149 // evita editar el contenido de los jtextpanel
150 monitor.setEditable(false);
151 impresora.setEditable(false);
152
```

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.

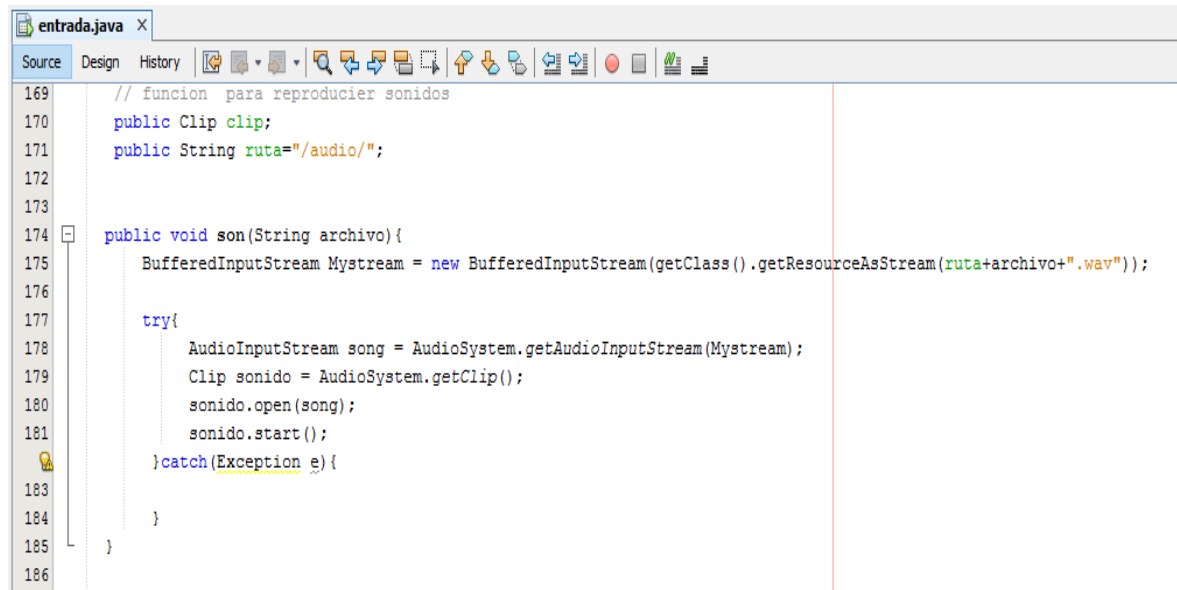
➤ Función memtotal():



```
157
158 //funcion encargada de capturar los valores de kernel y memoria y mostrar la cantidad de
159 //memoria que queda disponible para la asignacion de los programas
160 public void memtotal() {
161     int mem = (int) memoria.getValue();
162     int ker = (int) kernel.getValue();
163     int total = mem - ker;
164     total_memoria.setText(String.valueOf(total));
165
166 }
167
```

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():



```
169 // funcion para reproducir sonidos
170 public Clip clip;
171 public String ruta = "/audio/";
172
173
174 public void son(String archivo) {
175     BufferedInputStream mystream = new BufferedInputStream(getClass().getResourceAsStream(ruta+archivo+".wav"));
176
177     try {
178         AudioInputStream song = AudioSystem.getAudioInputStream(mystream);
179         Clip sonido = AudioSystem.getClip();
180         sonido.open(song);
181         sonido.start();
182     } catch (Exception e) {
183
184     }
185 }
186
```

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():

```

189 public void encender() {
190     // HACE EL LLAMADO A LA FUNCION PARA QUE REPRODUzca EL SONIDO DE ENSENDIDO
191     // desactiva los spinner
192     kernel.setEnabled(false);
193     memoria.setEnabled(false);
194     encender.setEnabled(false);
195     encender2.setEnabled(false);
196     cargarprograma.setEnabled(true);
197     apagarmaquina1.setEnabled(true);
198     apagarmaquina2.setEnabled(true);
199     //sonidoencender("inicio");
200     son("inicio");
201     // INSTANCIA OBJETO PARA LLENAR LA TABLA
202     Object []object = new Object[2];
203
204     // VALORES POR DEFECTO DE LA PRIMERA POSICION DEL MAPA D EMEMORIA
205     object[0]="0";
206     object[1]="acumulador";
207     modelo.addRow(object);
208
209     // CICLOS ENCARGADOS DE LLENAR EL MAPA DE MEMORIA
210     int contador = 0;
211
212     int mem = Integer.parseInt(total_memoria.getText());
213     int ker = (int) kernel.getValue();
214     for (int i = 0; i < ker; i++) {
215         contador++;
216         object[0]=String.valueOf(contador);
217         object[1]="-----sistema operativo-----";
218         modelo.addRow(object);
219     }
220
221     for (int i = 0; i < mem; i++) {
222         contador++;
223         object[0]=String.valueOf(contador);
224         object[1]="-----instruccion-----";
225         modelo.addRow(object);
226     }
227 }

```

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 apagar():

```
entrada.java x
Source Design History
230
231 // funcion encargada de apagar la maquina y regresarla a su estado inicial
232 public void apagar() {
233     // codigo encargado de apagar la maquina y regresarla a el estado inicial
234
235     if(JOptionPane.showOptionDialog(this, "¿ESTA SEGURO QUE DESEA APAGAR LA MAQUINA?", "Mensaje de Alerta",
236         JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE, null, new Object[]{" SI ", " NO "}, "NO")==0)
237     {
238         son("cierre");
239         // se encarga de detener un instante el proceso
240         try {
241             Thread.sleep(2000); // el tiempo es en milisegundos
242         } catch (InterruptedException ex) {
243
244         }
245         setVisible(false);
246         new entrada().setVisible(true);
247     }
248     else
249     {
250         JOptionPane.showMessageDialog(this, "PUEDE CONTINUAR CON LA EJECUCION DEL PROGRAMA");
251     }
252
253
254 }
255
```

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 de el efecto de apagado .

➤ Función initComponents():

```
entrada.java x
Source Design History
259
260 /**
261  * This method is called from within the constructor to initialize the form.
262  * WARNING: Do NOT modify this code. The content of this method is always
263  * regenerated by the Form Editor.
264  */
265 @SuppressWarnings("unchecked")
266 // componentes de la interfaz del ch-maquina
267 // <editor-fold defaultstate="collapsed" desc="Generated Code">
268 private void initComponents() { ...380 lines } // </editor-fold>
648
```

Esta función tiene encapsulado todas las variables y procedimientos instanciados para interfaz grafica 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():

```
entrada.java x
Source Design History
648
649 private void cargarprogramaActionPerformed(java.awt.event.ActionEvent evt) {
650     // TODO add your handling code here:
651     // encargado de abrir el panel de busqueda de archivos y cargarlo a la funcion actualizar.
652     JFileChooser ventana = new JFileChooser();
653     // filtra las extensiones segun la que buscamos
654     ventana.setFileFilter(new FileNameExtensionFilter("todos los archivos "
655     + ".ch", "CH", "ch"));
656     int sel = ventana.showOpenDialog(entrada.this);
657
658     if (sel == JFileChooser.APPROVE_OPTION) {
659
660         File file = ventana.getSelectedFile();
661
662         //actualizar(file.getPath());
663
664     }
665 }
666
667
```

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 extension.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():

```
673
674 private void memoriaStateChanged(javax.swing.event.ChangeEvent evt) {
675     // TODO add your handling code here:
676     // en caso tal de que el kernel supere el tamaño de la memoria la memoria se modificara en 1 mas que el kernel
677     int kertemp = (int) kernel.getValue();
678     int memtemp = (int) memoria.getValue();
679     if (kertemp >= memtemp) {
680         memtemp=kertemp+1;
681         memoria.setValue(memtemp);
682     }
683     //muestra la memoria disponible para los programas
684     memtotal();
685 }
686
687 private void kernelStateChanged(javax.swing.event.ChangeEvent evt) {
688     // en caso tal de que el kernel supere el tamaño de la memoria la memoria se modificara en 1 mas que el kernel
689     int kertemp = (int) kernel.getValue();
690     int memtemp = (int) memoria.getValue();
691     if (memtemp <= kertemp) {
692         kertemp=memtemp-1;
693         kernel.setValue(kertemp);
694     }
695     //muestra la memoria disponible para los programas
696     memtotal();
697 }
698
```

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()`:

```
private void encenderActionPerformed(java.awt.event.ActionEvent evt) {  
    // HACE EL LLAMADO A LA FUNCION PARA QUE REPRODUzca EL SONIDO DE ENSENDIDO  
    encender();  
}  
  
741  
742 private void apagarmaquina2ActionPerformed(java.awt.event.ActionEvent evt) {  
743     // TODO add your handling code here:  
744  
745     //llama la funcion encargada de apagar la maquina  
746     apagar();  
747  
748  
749  
750 }  
751  
752 private void encender2ActionPerformed(java.awt.event.ActionEvent evt) {  
753     // HACE EL LLAMADO A LA FUNCION PARA QUE REPRODUzca EL SONIDO DE ENSENDIDO  
754     encender();  
755 }  
756  
757 private void apagarmaquina1ActionPerformed(java.awt.event.ActionEvent evt) {  
758     // TODO add your handling code hermie:  
759     //llama la funcion encargada de apagar la maquina  
760     apagar();  
761 }  
762
```

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():

```

718 private void jMenuItem3ActionPerformed(java.awt.event.ActionEvent evt) {
719
720     // codigo encargado de CERRAR EL PROGRAMA
721     if(JOptionPane.showOptionDialog(this, "¿ESTA SEGURO QUE DESEA SALIR DEL PROGRAMA?", "Mensaje de Alerta",
722         JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE, null, new Object[]{" SI ", " NO "}, "NO")==0)
723     {
724         son("cierre");
725
726         // se encarga de detener un instante el proceso
727         try {
728             Thread.sleep(3000); // el tiempo es en milisegundos
729         } catch (InterruptedException ex) {
730
731         }
732         System.exit(0);
733     }
734
735     else
736     {
737         JOptionPane.showMessageDialog(this, "PUEDE CONTINUAR CON LA EJECUCION DEL PROGRAMA");
738     }
739
740 }
741

```

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 acercaActionPerformed();

```

766
767 private void acercaActionPerformed(java.awt.event.ActionEvent evt) {
768     // TODO add your handling code here
769     JOptionPane.showOptionDialog(this, "Product Version: MI CH-MAQUINA V.1.0.0\n" +
770         "Actualizaciones: en proceso...\n" +
771         "Java: 1.7.0_51; Java HotSpot(TM) 64-Bit Server VM 24.51-b03\n" +
772         "Runtime: Java(TM) SE Runtime Environment 1.7.0_51-b13\n" +
773         "System recomendado: Windows 7 y posterior\n" +
774         "creado por :YEISON AGUIRRE OSORIO -- NAUFRAGO\n" +
775         "UNIVERSIDAD DE COLOMBIA - SEDE MANIZALES\n"
776         + "fecha creacion: febrero 2016\n\n"
777         + "simulador de OS encargado de leer instrucciones de un archivo con \n"
778         + "extencion .CH en este estan los pasos y valores iniciales que el \n"
779         + "simulador debe interpretar y ejecutar, lo puede hacer de modo recorrido o \n"
780         + "paso a paso, durante la ejecucion se ve el mapa de memoria y que hay \n"
781         + "almacenado en ella ademas de el cuador de procesos activos y variables declaradas,\n"
782         + "los resultados del proceso pueden visualizarcen en monitor e impresion.", "ACERCA DE MI CH-MAQUINA.",
783         JOptionPane.INFORMATION_MESSAGE, JOptionPane.INFORMATION_MESSAGE, null, new Object[]{" OK "}, "OK");
784 }

```

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.