|  |
| --- |
| **Documentación CH-MAUINA** |
| Requerimientos Clases métodos funciones |
| Sistemas operativos |
| **DOCENTE :Carlos Hernán Gómez**  **CURSO :sistemas operativos**  Descripción: Descripción: http://www.regalospublicitariosm.com/UNIVERSIDAD%20NACIONAL%20DE%20COLOMBIA.jpg  **UNIVERSIDAD NACIONAL DE COLOMBIA SEDE MANIZALES**  **ADMINISTRACIÓN DE SISTEMAS INFORMÁTICOS** |
| **YEISON AGUIRRE OSORIO COD 913503** |
| **Fecha de inicio 5/feb/2016**   1. **Índice.** 2. **Introducción y descripción del programa Pag 3** 3. **Objeto y aspectos principales del programa. Pag 3** 4. **Que hace el ch-máquina y cuál es su sintaxis. Pag 4**      1. **Código completo y explicación de funcionamiento del programa. Pag 7** 2. **Explicación de la gestión de errores del programa.** 3. **Explicación de variables, datos, archivos.** 4. **Recomendaciones para el mantenimiento futuro.** |
|  |
|  |

# **Índice:**

1. **Índice. Pag 2**
2. **Introducción y descripción del programa. Pag 3**
3. **Objetos y aspectos principales del programa. Pag 4**
4. **Que hace el ch-máquina y cuál es su sintaxis. Pag 5**

* **Operaciones descripción sintaxis. Pag 5**
* **Ejecución del programa. Pag 7**

1. **Código completo y explicación de funcionamiento del programa. Pag 9**

* **Estructura del proyecto ch-maquina. Pag 9**
* **Paquete chmaquina clase entrada.java Pag 9**

1. **Librerías importadas. Pag 10**
2. **Clase entrada. Pag 11**
3. **Funciones. Pag 15**

* **Función memtotal(): Pag 16**
* **Funcion son(): pag 16**
* **Función encender(): pag 17**
* **Función apagar(): Pag 18**
* **Función initComponets(): Pag 18**

1. **Eventos por botones. Pag 19**

* **evento de botón cargarprogramaActionPerformed(): Pag 19**

# **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.

# **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).

# **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 Incremente 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 continué 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 limite) 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.

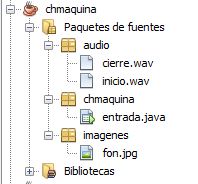
Se podrán cargar y correr varios ch-programas hasta agotar la memoria disponible, para la corrida de los ch-progamas en la primera fase se utilizará una cola circular, la cual será visitada con base al orden de llegada (cola simple-primero en entrar primero en ser atendido-FCFS).

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.

# **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**:

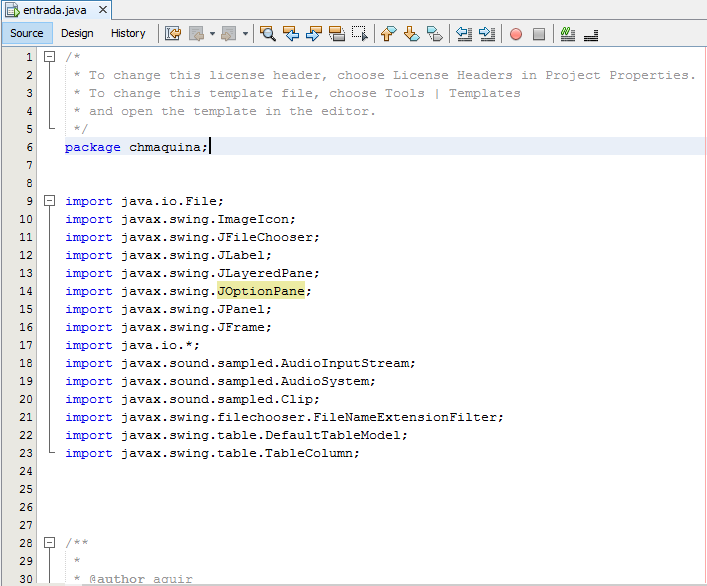


El proyecto en su estructura tiene 3 paquetes contenedores que son audio(contiene todos los archivos de sonido utilizados en el proyecto, estos son de extensión .wav), imágenes(contiene todas las imágenes de múltiples tipos utilizadas en el proyecto), y chmaquina contenedor de las clases.

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

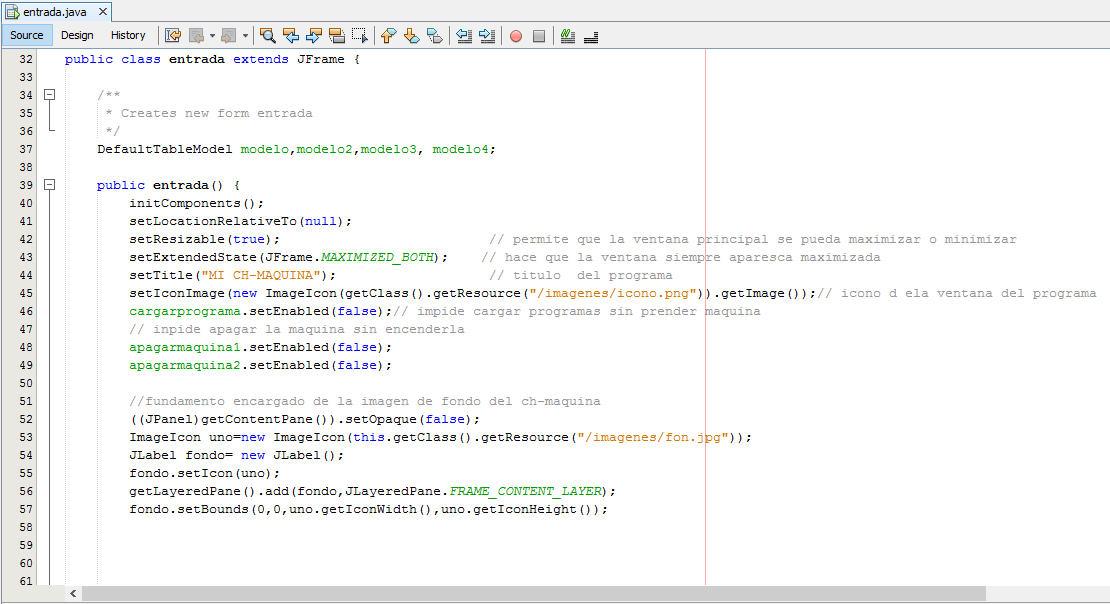
En este punto se dará una explicación concisa de partes del código encargadas de ciertas funcionalidades .

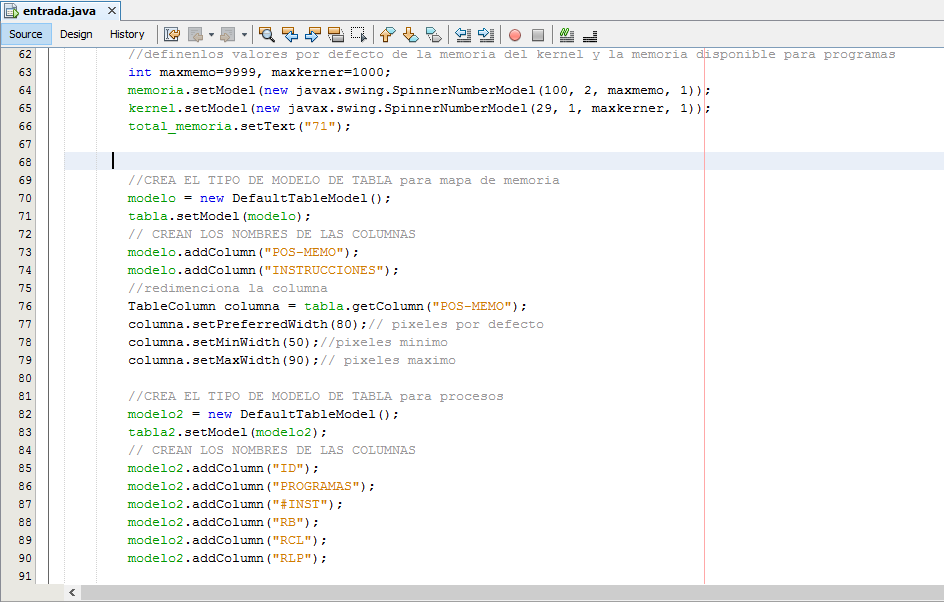
#### Librerías importadas:

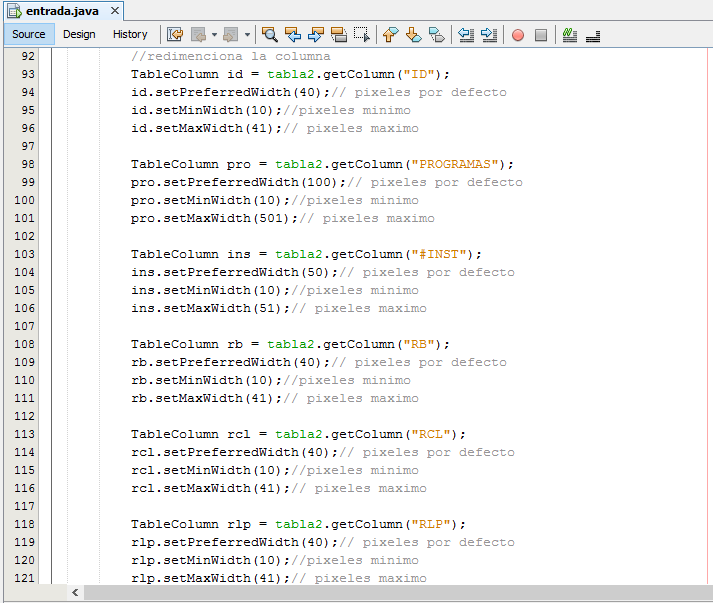


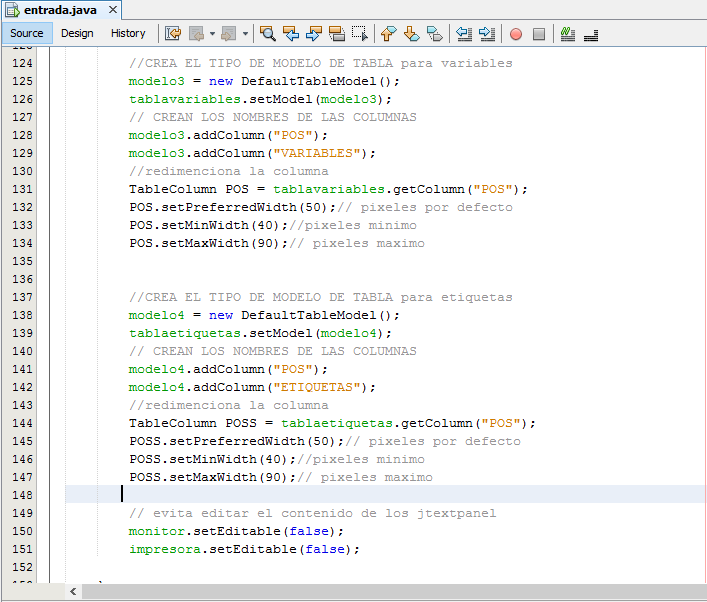
Librerías encargadas de interactuar con el código facilitando variedad de operaciones y funciones, tanto en la construcción como en la implementación de código.

#### Clase entrada





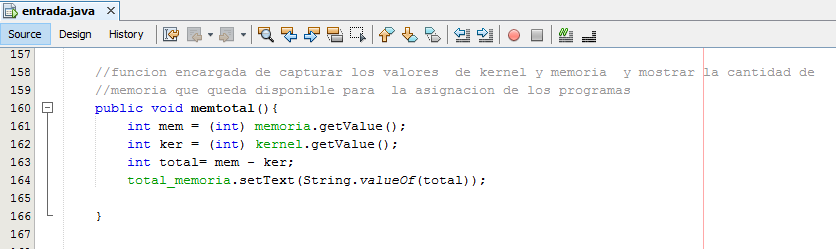


Podemos identificar la función principal **entrada();**  esta se encarga de inicializar todos los componentes de arranque de la interfaz principal inicializando imágenes títulos tablas spinners entre otros y asignándoles valores por defecto , los cuales durante la implementación pueden cambiar .

#### Funciones:

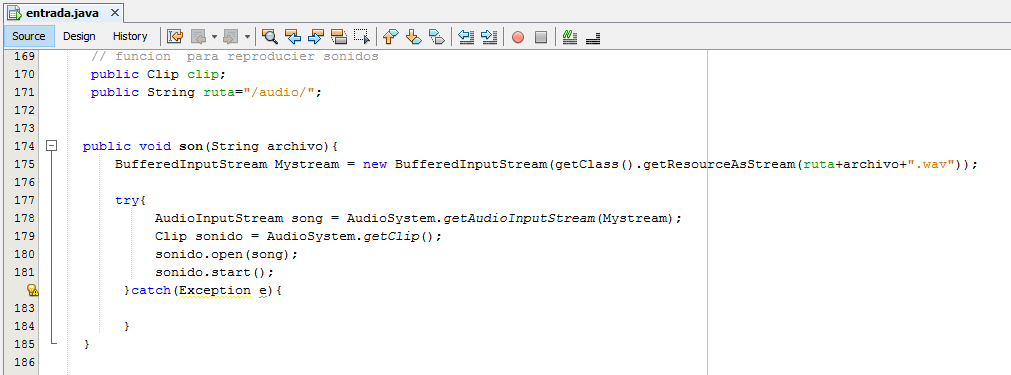
Encargadas de ejecutar instrucciones dadas por el usurario.

##### Función memtotal():



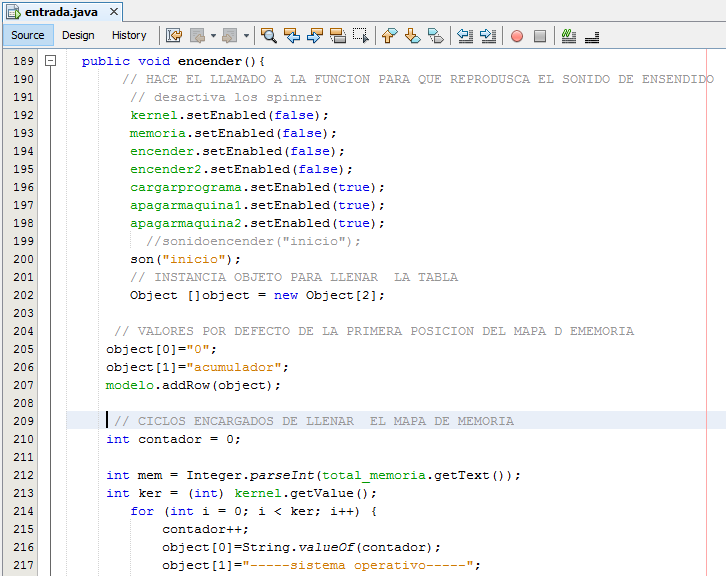
Esta función toma los valores de los spinner de memoria y kernel y saca el total de memoria neta que estará disponible para los programas a cargar y retorna el resultado a un JLabel con nombre total memoria donde puede ser visto por el usuario.

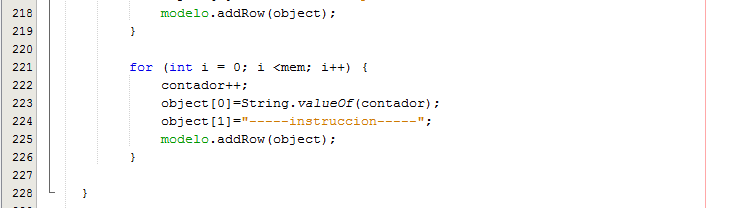
##### Función son():



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.

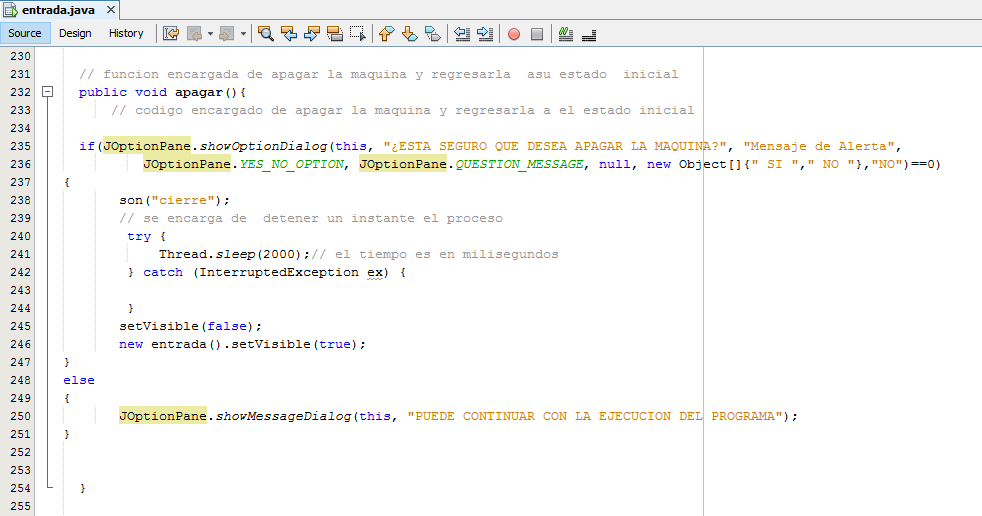
##### Función encender():





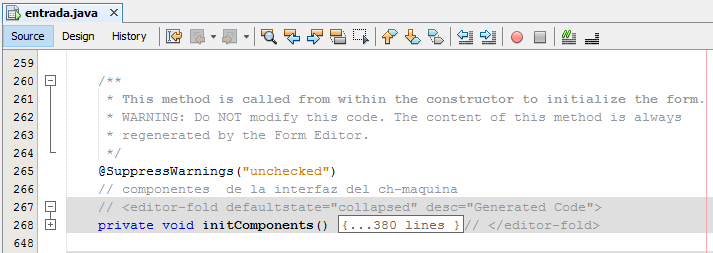
Esta función se encarga de hacer el segundo arranque de la interfaz aplica la inicialización del mapa de memoria el sonido de encendido y desactivación de botones de carga de archivo, encendido de máquina y los medidores de memoria y kernel.

##### Función apagar():



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 initComponets():

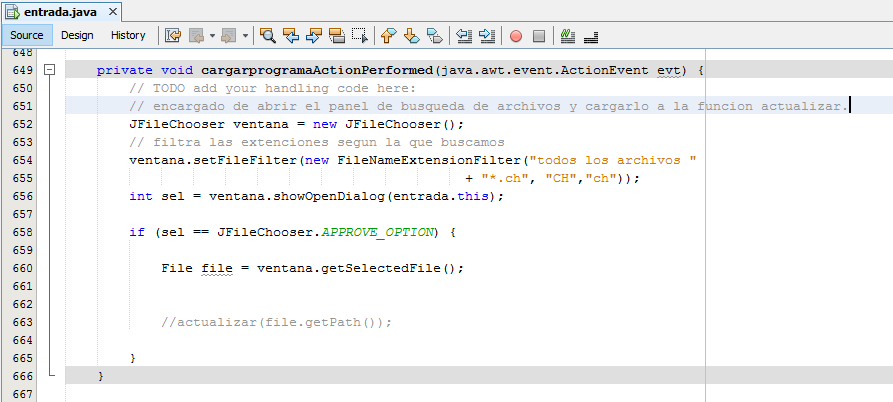


Esta función tiene encapsulado todas las variables y procedimientos instanciados para interfaz grafica y es la primera función llamada en el primer arranque del programa.

#### eventos por botones:

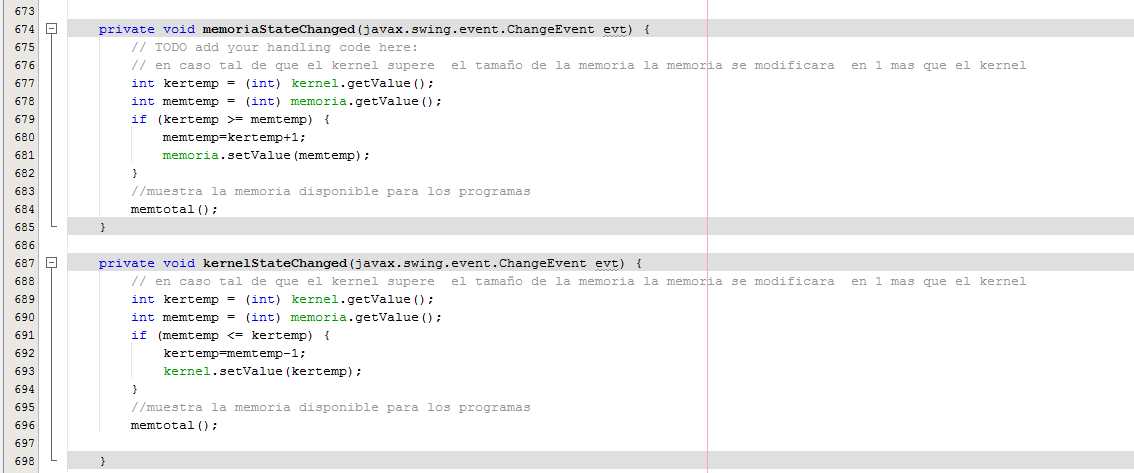
Son aquellas instrucciones ejecutadas al activase un botón en la interfaz.

##### evento de botón cargarprogramaActionPerformed():



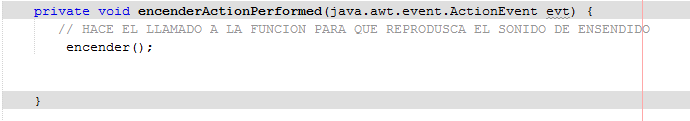
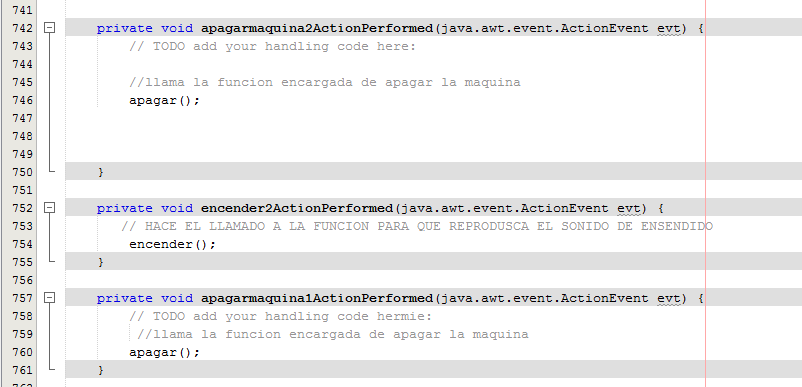
Este evento se encarga de abrir un panel de exploración de archivos para realizar la carga del .ch y ser compilado, este tiene la condición que solo detecta los archivos con extension.ch, este evento es activado por el botón cargar programa del panel de archivo o por el comando ctl+ o .

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



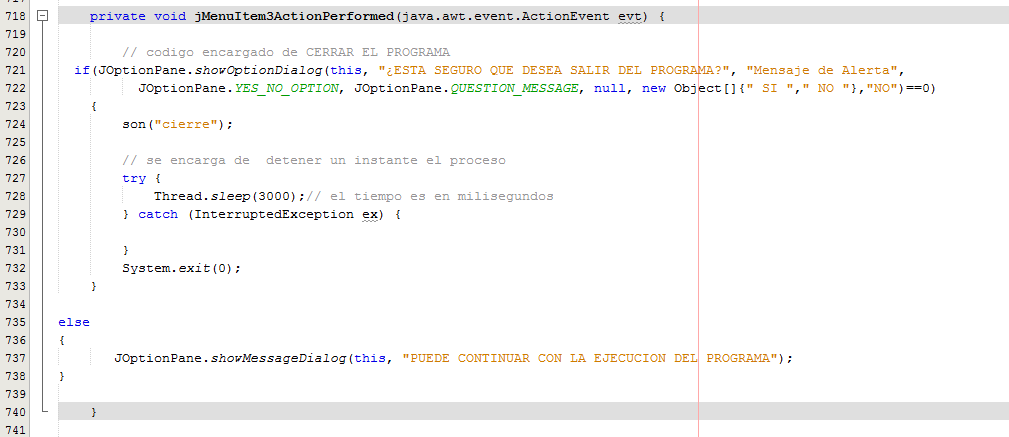
Estos dos eventos se encargan de mantener coherencia en la memoria donde el kernel nunca va poder ser superior o igual a la memoria y la memoria nunca podrá ser menor o igual al kernel.

##### Evento de botón encenderActionPerformed(), apagarmaquina2ActionPerformed(), encender2ActionPerformed(), apagarmaquina1ActionPerformed():

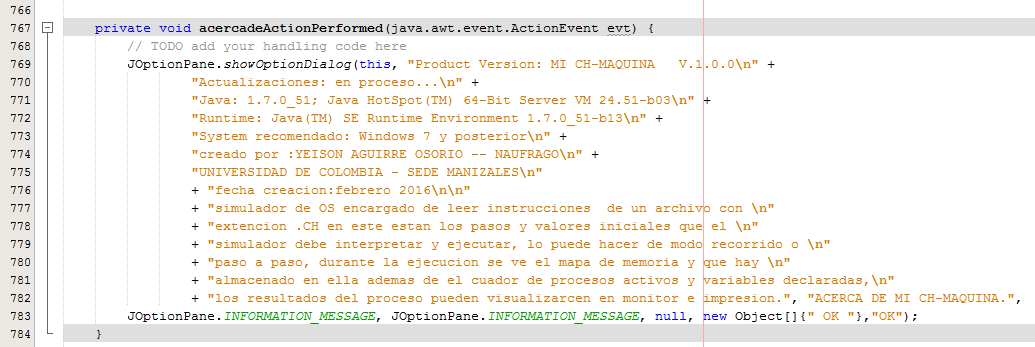
Estos eventos se encargan de llamar las funciones de encender y apagar la maquina son activados por los botones del mismo nombre.

##### Evento de botón jMenuItem3ActionPerformed():



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();



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.