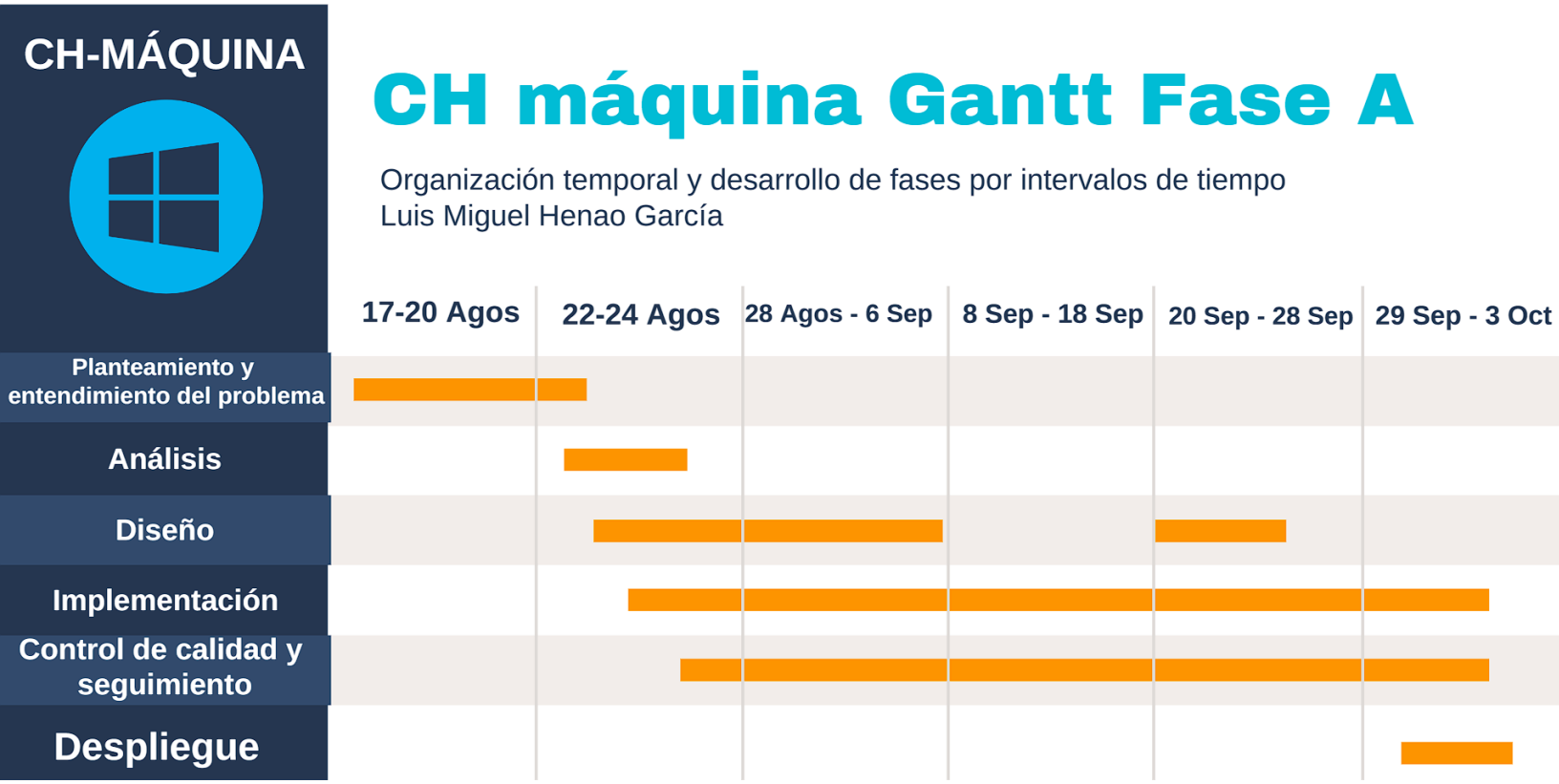
Planeación del proyecto:

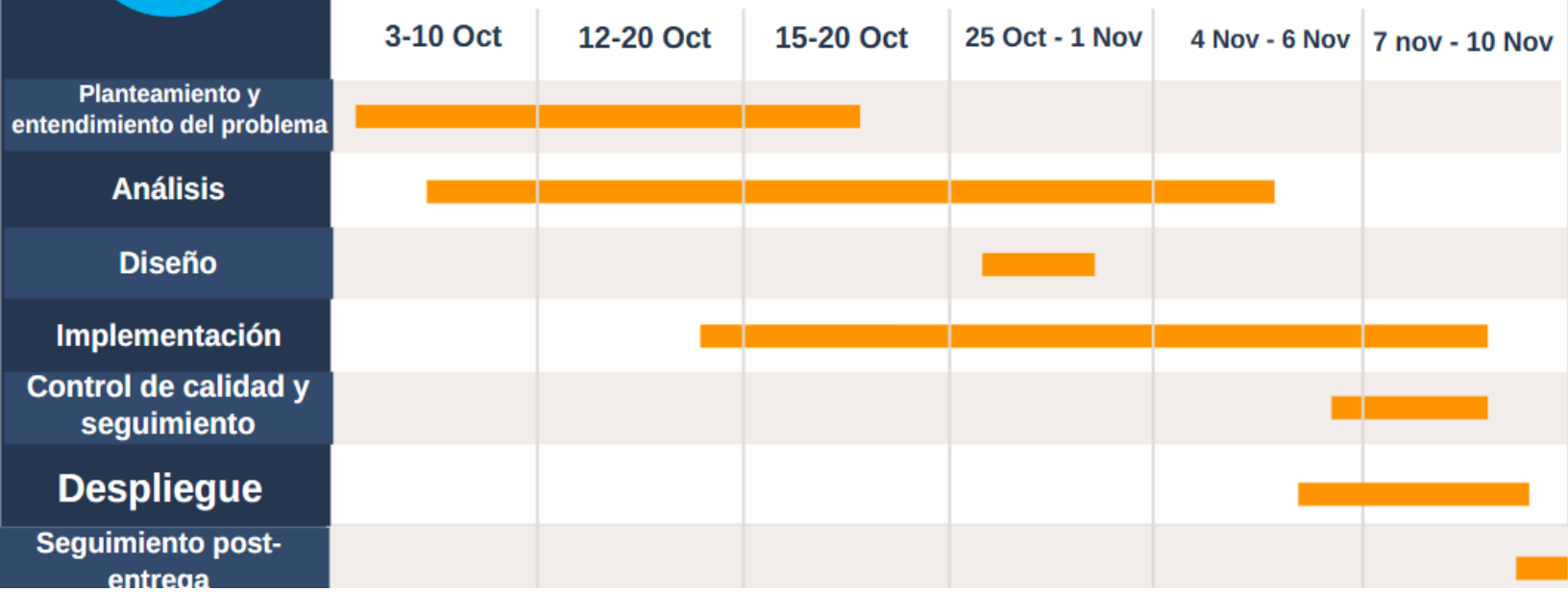
El ch-máquina (en su fase A) es un proyecto que busca crear un entorno virtual donde se pueda simular el funcionamiento de un sistema operativo. Para ello, se utiliza un lenguaje de programación que permite definir archivos con una sintaxis específica que contienen instrucciones para el ch-máquina. Estos archivos se pueden cargar, ejecutar, pausar y observar en el ch-máquina, que cuenta con una interfaz gráfica que muestra la pantalla, la impresora, el espacio en memoria y el espacio kernel del sistema operativo simulado. El ch-máquina también puede detectar y mostrar los errores de sintaxis que puedan tener los archivos, así como modificar los parámetros del entorno virtual según las necesidades del usuario. El objetivo del ch-máquina es ofrecer una herramienta didáctica y divertida para aprender sobre los conceptos básicos de los sistemas operativos y su funcionamiento interno.



CH-MÁQUINA

CH máquina Gantt Fase B

Organización temporal y desarrollo de fases por intervalos de tiempo
Luis Miguel Henao García



Documentación del proyecto:

El ch-máquina es una simulación de un sistema operativo a través de un lenguaje de programación, esta simulación se trata de adjuntar archivos con extensión .ch los cuales tendrán una sintaxis específica de acuerdo a una serie de instrucciones definidas y una serie de parámetros posibles que tienen estas instrucciones, con esto el programa al ser ejecutado tendrá una tabla que representará el espacio en memoria disponible y el espacio kernel (espacio de sistema operativo). Estos dos se podrán modificar de acuerdo a las necesidades del usuario, los espacios en memoria contendrán las líneas de los archivos adjuntos representando la cantidad de memoria que ocupan.

Tendrán después de que terminen las líneas, las variables definidas por el mismo, al adjuntar un archivo el programa de manera automática irá a un método llamado examinar sintaxis y realizará en él todas las pruebas para verificar la correcta sintaxis, de ser correcta, pasará a ocupar la memoria disponible, crear variables, crear etiquetas, las cuales serán asignadas a una instrucción específica y posteriormente mostrar en una tabla tanto las líneas del archivo y en otra tabla los errores.

El ch-máquina después de adjuntar el archivo tendrá opciones de ejecutar paso a paso, ejecutar de corrido y pausar la ejecución, al momento de ejecución se observará línea por línea en la pantalla y se ejecutará lo que dicen, habrá una variable llamada acumulador la cual se encargará de la mayoría de procesos, se podrán actualizar las variables, hacer operaciones entre ellas y mostrar resultados (de manera general lo que pasa con las variables) y en el momento en el que se hayan ejecutado todas las instrucciones deberá mostrar en pantalla que se terminó el archivo y pasará con el siguiente.

Manual técnico:

El ch-máquina está elaborado en el entorno de programación NetBeans y, en el lenguaje de programación Java.

Para entender el funcionamiento de este programa debemos de tener en cuenta cómo deben de estar estructurados los archivos con extensión .ch que este recibirá, para ello, tome en cuenta la siguiente información.

Los archivos están compuestos por instrucciones que ocupan una línea, la cual tendrá la siguiente estructura:

INSTRUCCIÓN OPERANDO1 OPERANDO2... OPERANDO N

Gracias a esto podemos tomar en cuenta ya la documentación de las posibles instrucciones

Operación	Descripción
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 en la posición de memoria que corresponda a la variable indicada por el operando.
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, L lógico o booleano (**1** Verdadero o **0** Falso), 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, espacio para cadenas, 0 para lógicos. El separador de decimales es el punto.
lea	Lee por teclado/pantalla el valor a ser asignado a la variable indicado por la variable referida en el operando.
sume	Incremente/sume al valor del acumulador el valor indicado por la variable señalada por el operando.
reste	Decremente/reste del valor del acumulador el valor indicado por la variable que señala el operando.
multiplique	Multiplique 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). El contenido del acumulador deberá tratarse como cadena en caso de ser numérico.
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 operando con valor numérico) de la cadena que se encuentra en el acumulador.
Y	Produce una operación lógica Y (AND) entre el primer operando y el segundo operando que son variables lógicas y la almacena en el tercer operando.
	son variables lógicas y la almacena en la variable del tercer operando.
NO	Produce una operación de negación lógica para el primer operando que es una variable lógica y el resultado se almacena en la variable del segundo operando.
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	Presente por la impresora el valor que hay en la variable indicada por el operando, si el operando es acumulador muestre el valor del acumulador.
retorne	El programa termina; debe ser la última instrucción del programa y tiene opcionalmente un operando numérico entero.
vaya*	Salte a la instrucción que corresponde a la etiqueta indicada por el operando y siga la ejecución a partir de allí.
vayasi*	Si el valor del acumulador es mayor a cero salte a la instrucción que corresponde a la etiqueta indicada por el primer operando y continue la ejecución a partir de allí. Si el valor del acumulador es menor a cero salte a la instrucción que corresponde a la etiqueta indicada por el segundo operando y continue la ejecución a partir de allí. o Si el acumulador es cero salte a la siguiente instrucción adyacente a la instrucción vayasi y siga la ejecución a partir de allí.
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.
XXX	**Esta será una función complementaria a las anteriores creada por el estudiante, se deja a voluntad su nombre y forma de funcionamiento.**

Con esta información o documentación presente podremos observar ya un ejemplo de la estructura de un archivo .ch:

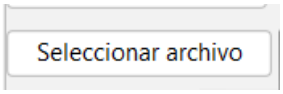
FACTORIAL.CH:

```
nueva unidad I 1

nueva m I 5

nueva respuesta I 1

nueva intermedia I 0

cargue m

almacene respuesta

reste unidad

almacene intermedia

cargue respuesta

multiplique intermedia

almacene respuesta

cargue intermedia

reste unidad

vayasi itere fin

etiqueta itere 8

etiqueta fin 19

muestre respuesta

imprima respuesta

retorne 0
```

Este archivo tiene un total de 19 líneas y como observamos, cada una de estas líneas empieza por una instrucción definida, debemos tener en cuenta entonces, que cada instrucción tiene una cantidad de operandos definida.

EJECUCIÓN:

Para guiarnos fácilmente, nos iremos paso por paso de la ejecución del archivo anteriormente mencionado.



Nos dirigimos entonces, a la interfaz gráfica y daremos click al botón de seleccionar archivo, nos mostrará una ventana emergente que contendrá la exploración de los archivos en nuestro dispositivo, en este caso adjuntaremos el **Factorial.ch.**

Automáticamente el programa se irá a esta función después de adjuntar el archivo

```
524     private void btnseleccionararchivoActionPerformed(java.awt.event.ActionEvent evt) {
525         JFileChooser fc = new JFileChooser(); //Sirve para permitir al usuario seleccionar un archivo
526         fc.showOpenDialog(parent:null);//abrir el seleccionador de archivo sin ningun archivo seleccionado
527         File archivo = fc.getSelectedFile(); //se instancia una variable tipo file y será igual al archivo seleccionado
528         try {
529             FileReader fr = new FileReader (file: archivo); //variable de lector de archivo
530             BufferedReader br = new BufferedReader(in: fr); //variable para mostrar donde queramos el texto del archivo
532             String texto = "";
533             String linea = "";
534             String nombreArchivo = archivo.getName();
535             int contadorii = 0;
536             DefaultTableModel modeloTablaii = (DefaultTableModel)this.tablaEspacioMemoria.getModel();
537             while((modeloTablaii.getValueAt(row:contadorii, column:1)!=null)){ /*Si la fila que indica el contador tiene un texto
538                                                                                 es diferente de nada, entonces es porque hay algo
539                                                                                 y no puedo colocar la variable ahí*/
540                 contadorii++; //aumento el contador para que pase a la siguiente fila
541             }
542             int contadorLinea = 0;
543             int contadorErrores = 0; //inicializo una variable para contar los errores
544 
545             while(((linea = br.readLine())!=null)){ //si la línea leida tiene texto se le agregará a la variable texto
546                 contadorLinea++;
547                 texto+=linea+"\n";
548                 boolean sintaxisCorrecta = examinarSintaxis(linea,contadorLinea,archivo: nombreArchivo);
549                 if(sintaxisCorrecta ==false){
550                     contadorErrores++; //como examino línea por línea si me retorna por lo menos un error se aumentará el contadorErrores
551                 }
552             }
553             String textofinal = this.txArchivo.getText() + "A R C H I V O : "+archivo.getName()+"\n"+texto;
554             this.txArchivo.setText(t: textofinal);
555             JOptionPane.showMessageDialog(parentComponent: null, message: "Archivo leido correctamente");
557             //OrganizarEspacioMemoria();
             if(contadorErrores==0){ //compruebo si hay por lo menos un error en el archivo
                     String linea1=""; //creo una nueva variable linea para no confundirla con la de arriba
                     FileReader fr1 = new FileReader (file: archivo); //creo un nuevo lector de documento
```

En primer lugar tenemos una serie de instancias que nos permitirán al presionar el botón abrir la ventana emergente para seleccionar un archivo, después guardaremos el archivo seleccionado en una variable y con una nueva instancia llamada fr y br leeremos línea por línea el archivo (ignoremos el primer while de momento).

En una variable llamada texto iremos adjuntando cada línea que encontramos y llamaremos después a una función llamada sintaxis correcta que recibe una línea, el

contador de la línea y el nombre del archivo que adjuntamos. Con esto pasaremos a la función examinarSintaxis():

```
String errores = ""; //creo un string para adjuntarlo al textArea de los errores
private boolean examinarSintaxis(String linea, int contadorLinea, String archivo){ //el método recibe una línea y le hace la revisión
    String[] palabras = linea.trim().split(regex: "\\s"); //separo por palabras la frase y las meto en un arreglo
    String[] instrucciones = {"cargue","almacene","nueva","lea","sume","reste","multiplique","divida","potencia","modulo","concatene","elimine","extraiga"
    //la línea de arriba contiene un vector con las palabras que siempre deben ir en la primera posición
    boolean instruccionCorrecta = false;
    boolean sintaxisCorrecta = true;
    palabras = Arrays.stream(array: palabras).filter(palabra -> !palabra.isEmpty()).toArray(String[]::new); //elimino espacios de más entre palabras
    if (linea.trim().startsWith(prefix:"//")) {
    return true; // No se considera un error
    }
    if (palabras.length == 0) {
    errores += "Advertencia: No se encontró contenido en la línea " + contadorLinea + " " + archivo + "\n";
    sintaxisCorrecta = true; // No es un error, solo una advertencia
    txErrores.setText(t: errores);
    return sintaxisCorrecta;
    }
    if(palabras.length>1){
        for(String i:instrucciones){ //recorro el vector que contiene las palabras de la frase
            if(palabras[0].equals(anObject: i)){ //comparo con cada palabra de las instrucciones la primera palabra de la frase
               instruccionCorrecta = true; //la instrucción es correcta si coincide con alguna de las del arreglo
               if(palabras[1].equals(anObject: "")){ //como toda línea debe tener por lo menos una instrucción y un operando, compruebo
                  errores+="Error: no se encuentra operando para la instrucción "+i+" linea "+contadorLinea+" "+archivo+"\n";
                  sintaxisCorrecta = false;
               }
               if(i.equalsIgnoreCase(anotherString:"nueva")){ //desde aquí se empiezan a tomar los casos de instrucciones que tienen más de un operando
                        boolean variableCorrecta = false;
                        if (palabras.length < 3) {
                         errores += "Error: la instrucción 'nueva' requiere al menos 2 palabras adicionales, línea " + contadorLinea + " " + archivo + "\n"
                         sintaxisCorrecta = false;
                         }
                        else{
                        String[]tiposVariables = {"C","I","R","L"}; //guardo los tipos de variables en un arreglo
                        variableCorrecta = false;
                         for(String j:tiposVariables){ //recorro
                         if(palabras[2].equalsIgnoreCase(anotherString:j)){
```

Para no complicarnos mucho, explicaré de manera general lo que hace esta función, tenemos una variable llamada errores y una lista con todas las instrucciones permitidas las cuales estarán posicionadas en la primera palabra de la línea, hacemos la verificación con la línea partida por palabras y según que instrucción sea hacemos las demás verificaciones, por ejemplo, si tenemos que la primera palabra de la línea es la instrucción “nueva”, entonces debemos verificar si la tercera palabra de esa línea corresponde a un tipo de variable definida.

Haciendo todas estas verificaciones a través de la variable sintaxisCorrecta verificamos si hay algún error, si lo hay la variable errores aumenta a uno, la variable sintaxisCorrecta se convierte en falso y se retorna y adjunta a un String llamado “errores” el error o los errores que se han identificado y se coloca en la tabla de errores que está en la interfaz gráfica, si no hay errores la variable sigue en 0 y de igual forma se retorna sintaxisCorrecta en verdadero.

En la línea 548 se observa la variable de contadorErrores de la que estábamos hablando, después de haber examinado la sintaxis de todas las líneas, se termina el while y se adjunta al panel de información de archivos el nombre del archivo y las

```
543             while(((linea = br.readLine())!=null)){ //si la línea leida tiene texto se le agregará a la variable texto
544                 contadorLinea++;
545                 texto+=linea+"\n";
546                 boolean sintaxisCorrecta = examinarSintaxis(linea,contadorLinea,archivo: nombreArchivo);
547                 if(sintaxisCorrecta ==false){
548                     contadorErrores++; //como examino línea por línea si me retorna por lo menos un error se aumentará el contadorErrores
549                 }
550             }
551             String textofinal = this.txArchivo.getText() + "A R C H I V O : "+archivo.getName()+"\n"+texto;
552             this.txArchivo.setText(t: textofinal);
553             JOptionPane.showMessageDialog(parentComponent: null, message: "Archivo leido correctamente");
554             //OrganizarEspacioMemoria();
555             if(contadorErrores==0){ //compruebo si hay por lo menos un error en el archivo
                    String linea1=""; //creo una nueva variable linea para no confundirla con la de arriba
557                 FileReader fr1 = new FileReader (file: archivo); //creo un nuevo lector de documento
558                 BufferedReader br1 = new BufferedReader(in: fr1); //creo nuevo extractor de texto
559                 this.variables.removeAll(c: this.variables); //reinicio la lista a vacía para no confundir variables de otros archivos
560                 this.variablessolonumero.removeAll(c: this.variablessolonumero);
561                 this.etiquetas.removeAll(c: this.etiquetas);
562                 memoriaOcupada();
563                 while(((linea1 = br1.readLine())!=null)){ //si la línea leida tiene texto
564                     ocuparMemoria(linea: linea1,archivo); //le mando la línea a la función para que la agregue a la memoria
565                     }
566                 int contadoriiii = 0;
567                 DefaultTableModel modeloTablaj = (DefaultTableModel)this.tablaEspacioMemoria.getModel();
568                  while((modeloTablaj.getValueAt(row:contadoriiii, column:1)!=null)){ /*Si la fila que indica el contador tiene un texto
569                                                                                       es diferente de nada, entonces es porque hay algo
570                                                                                       y no puedo colocar la variable ahí*/
571                     contadoriiii++; //aumento el contador para que pase a la siguiente fila
572                     }
573                 for(String variable:this.variables){
574                 int contador = 0;
575                 DefaultTableModel modeloTabla = (DefaultTableModel)this.tablaEspacioMemoria.getModel();
576                 while((modeloTabla.getValueAt(row:contador, column:1)!=null)){ /*Si la fila que indica el contador tiene un texto
577                                                                                   es diferente de nada, entonces es porque hay algo
```

líneas que lo componen, posteriormente, gracias al retorno de la función examinarSintaxis(), si esta en el while retornó por lo menos una vez falso, el contador de errores será mayor que cero y no pasará a las instrucciones que podemos ver desde la línea 555.

En la línea 563 observamos nuevamente el recorrido de las líneas del archivo, pero esta vez, por cada línea se llama a una función ocuparMemoria la cual recibe una línea y un archivo.

```
434     private void ocuparMemoria(String linea, File archivo) {
435     int memoriaKernel = (int) this.spinMemoriaKernel.getValue(); // obtengo el espacio Kernel seleccionado
436     if (memoriaKernel == 0) { // si el número de espacio en memoria Kernel es igual a 0 (valor al iniciar el programa)
            memoriaKernel = 17; // Configuro el valor predeterminado de la memoria Kernel
438     }
439     if(linea.startsWith(prefix:"//")){
440         return;
441     }
442     String[] palabras = linea.trim().split(regex: "\\s+");
443     palabras = Arrays.stream(array: palabras)
444             .map(String::trim) // Eliminar espacios en blanco al principio y al final de cada palabra
445             .filter(palabra -> !palabra.isEmpty())
446             .toArray(String[]::new);
447     if(palabras.length!=0){
448     if (palabras[0].toLowerCase().equals(anObject: "nueva")) {
449         String valor = "";
450         try { // intenta hallar el valor de la variable
451             if(palabras[2].equalsIgnoreCase(anotherString:"C")){
452                 valor = String.join(delimiter:" ", elements: Arrays.copyOfRange(original: palabras, from: 3, to: palabras.length));
453             }
454             else{
455                 valor = palabras[3];
456             } // El valor se encuentra en la cuarta palabra (índice 3)
457
458         } catch (Exception e) { // si no hay inicialización entonces serán los valores por defecto según tipo de variable
459             try {
460                 if (palabras[2].equals(anObject: "I") || palabras[2].equals(anObject: "R") || palabras[2].equals(anObject: "L")) {
461                     valor = "0";
462
463                 } else {
464                     valor = " ";
465                 }
466             } catch (Exception r) {
467                 System.out.println("Error: " + r);
468              }
469         }
470         this.variables.add(palabras[1]+" : "+valor);
471         this.variablessolonumero.add(e: valor);
472     }
473     else if(palabras[0].toLowerCase().equals(anObject: "etiqueta")){
474         String nombreetiqueta = palabras[1];
475         String pos = palabras[2];
476         try {
477             int posicionlinea = Integer.parseInt(s: pos);
478             if(posicionlinea>=1){
479             FileReader fr = new FileReader (file: archivo); //variable de lector de archivo
480             BufferedReader br = new BufferedReader(in: fr); //variable para mostrar donde queramos el texto del archivo
481             String texto = "";
482             int contadorLinea = 0;
483             while(((linea = br.readLine())!=null&&contadorLinea<posicionlinea)){ //si la línea leida tiene texto se le agregará a la variable texto
484                 contadorLinea++;
485             }
486             if (contadorLinea == posicionlinea) {
487                 this.etiquetas.add("00"+(posicionlinea+(memoriaocupada-1))+"  "+"000"+(this.contadordeArchivos+1)+" "+palabras[1]+"\n");
488             } else {
489              JOptionPane.showMessageDialog(parentComponent: this, "No se encontró la línea correspondiente a la etiqueta " + nombreetiqueta);
490             }
491
492         }
493         }catch (Exception e) {
494             JOptionPane.showMessageDialog(parentComponent: this, "No se pudo castear a int el tercer parámetro de la etiqueta "+pos+"\n"+e);
495             System.out.println(x: e);
496         }
497     }
498     int contador = 0; // inicializo un contador para recorrer las filas de la tabla
499     DefaultTableModel modeloTabla = (DefaultTableModel) tablaEspacioMemoria.getModel(); // convierto el modelo para obtener texto de fila
500     while ((modeloTabla.getValueAt(row:contador, column:1) != null)) {
501         contador++;
502     }
505       this.lineasejecutar.add("00"+contador+" "+archivo.getName()+" "+(this.contadordeArchivos+1)+" "+lineaFormateada);
506       tablaEspacioMemoria.setModel(dataModel:modeloTabla);
507       }
508 }
```

La función ocuparMemoria() verifica si la línea es comentada y si es así, la línea no ocupará espacio en memoria, después se encarga de partirla en palabras y verificar si es de una instrucción nueva, si lo es, toma los operandos y los introduce en una lista de variables definida al inicio del programa. Lo mismo pasa con las etiquetas, si la primera palabra de la línea es una etiqueta también se agrega a una lista. Lo anterior es con el fin de hacer más sencillo la identificación de variables y etiquetas de un archivo (en caso de que se haya ejecutado dos veces el mismo archivo). Se observan dos listas para variable y dos para etiqueta, una se usa para mostrar las etiquetas y variables en los campos de texto correspondientes, y la otra es para identificar más adelante en la ejecución a qué archivo pertenecen las etiquetas o variables.

Después de hacer las verificaciones, lo último que hace esta función es insertar línea por línea en una celda de memoria.

Algo muy importante que hace esta función es que si la variable tiene un tipo de variable definido, pero no un valor se inicializa según su tipo. Por ejemplo, si la línea es:

“nueva variable R”, esta variable iniciará con 0 por ser un valor real.

```
567                 while((modeloTablaj.getValueAt(row:contadoriiii, column:1)!=null)){ /*Si la fila que indica el contador tiene un texto
568                                                                                     es diferente de nada, entonces es porque hay algo
569                                                                                     y no puedo colocar la variable ahí*/
570                     contadoriiii++; //aumento el contador para que pase a la siguiente fila
571                     }
572                 for(String variable:this.variables){
573                 int contador = 0;
574                 DefaultTableModel modeloTabla = (DefaultTableModel)this.tablaEspacioMemoria.getModel();
575                 while((modeloTabla.getValueAt(row:contador, column:1)!=null)){ /*Si la fila que indica el contador tiene un texto
576                                                                                     es diferente de nada, entonces es porque hay algo
577                                                                                     y no puedo colocar la variable ahí*/
578                     contador++; //aumento el contador para que pase a la siguiente fila
579                     }
580                 modeloTabla.setValueAt(aValue:variable, row:contador, column:1); //pongo la variable en el espacio vacío
581                 tablaEspacioMemoria.setModel(dataModel:modeloTabla);
582
583                 }
584                 for(String variable:this.variables){
585                     int contador = 0;
586                 DefaultTableModel modeloTabla = (DefaultTableModel)this.tablaEspacioMemoria.getModel();
587                 memoriaOcupada();
588                 while((!modeloTabla.getValueAt(memoriaocupada-1-contador, column:1).equals(obj:variable))){ /*Si la fila que indica el contador tiene un te
589                                                                                     es diferente de nada, entonces es porque hay algo
590                                                                                     y no puedo colocar la variable ahí*/
591                     contador++; //aumento el contador para que pase a la siguiente fila
592                     }
593                     actualizarTablaVariables(memoriaocupada-1-contador,variable);
594                 }
595                 for(String etiqueta:this.etiquetas){
596                 actualizarTablaEtiquetas(etiqueta);
597                 }
598                 int contadoriii = 0;
599                 while((modeloTablaii.getValueAt(row:contadoriii, column:1)!=null)){ /*Si la fila que indica el contador tiene un texto
600                                                                                     es diferente de nada, entonces es porque hay algo
601                                                                                     y no puedo colocar la variable ahí*/
602                     contadoriii++; //aumento el contador para que pase a la siguiente fila
```

Volvemos nuevamente a la función cuando se presiona el botón de adjuntar archivo, después de llamar tantas veces sea necesario la función ocuparMemoria(), observemos únicamente los whiles que hay a partir de las líneas 567, estos 3 que

observamos sirven para tres momentos, uno para saber cuántos espacios en memoria ocupa el archivo en total, otro para saber cuántos ocupa antes de inicializar las variables y el otro es para saber cuántos ocupa después de inicializara las variables. Sabiendo esto observemos los for, los for se encargan de tomar las listas que observamos anteriormente en la función ocuparMemoria() y actualizan las tablas de variables y de etiquetas para mostrarlas en pantalla.

```
509     public void actualizarTablaVariables(int contador, String variable){
            String variables = txVariables.getText();
511         variables +="00" +contador+"    "+"000"+(this.contadordeArchivos+1)+" "+variable+"\n";
512         this.variablesConNombreeIdentificadorArchivo.add("00" +contador+" "+"000"+(this.contadordeArchivos+1)+" "+variable);
513         txVariables.setText(t: variables);
514     }
515     public void actualizarTablaEtiquetas(String etiqueta){
            String etiquetas = txetiquetas.getText();
517         etiquetas += etiqueta;
518         this.etiquetasconNombreeIdentificadorArchivo.add(e: etiqueta);
519         txetiquetas.setText(t: etiquetas);
520     }
```

Estas son las funciones que se llaman en los for, las listas de variablesconNombreeIdentificadorArchivo y etiquetasconNombreeIdentificadorArchivo son fundamentales para la ejecución de las distintas líneas del archivo.

```
604                 String textotxAreaArchivo = this.txArchivos.getText();
605                 this.contadordeArchivos +=1;
606                 textotxAreaArchivo+="000"+this.contadordeArchivos+"   "+archivo.getName()+"_____"+((contadoriii)-(contadorii))+"   "+contadorii+"
607                 this.txArchivos.setText(t: textotxAreaArchivo);
608             }
609             else{
610                 JOptionPane.showMessageDialog(parentComponent: this, message: "El archivo tiene errores sintácticos y no puede ser ejecutado");
611             }
612         } catch (Exception e) {
613             JOptionPane.showMessageDialog(parentComponent: null, message: "No se pudo leer el archivo");
614             System.out.println("Error: "+e);
615         }
616         this.spinMemoriaKernel.setEnabled(enabled: false);
617     }
```

Lo que observamos ahora es la última parte de lo que puede decirse es la función o el pilar más importante de este programa (botón adjuntarArchivo), en esta observamos que si el contador de errores es diferente de 0 entonces sale un aviso que nos mostrará el mensaje de que el archivo tiene errores sintácticos y no puede ser ejecutado. De otra manera en la que ocurrió algún error en todo el proceso que vimos, se observará que no se pudo leer el archivo para evitar errores en la consola.

La línea 616 nos muestra que el spinMemoriaKernel pasará a deshabilitarse después de adjuntar un archivo, este spin es el espacio de memoria que ocupa el sistema operativo, consideré pertinente deshabilitarlo al adjuntar el primer archivo.

Ya observamos qué ocurre entonces cuando adjuntamos un archivo, entonces observemos en la interfaz gráfica qué ocurre cuando adjuntamos el archivo **Factorial.ch:**

nueva unidad l 1

nueva m l 5

nueva respuesta l 1

nueva intermedia l 0

cargue m

almacene respuesta

reste unidad

almacene intermedia

cargue respuesta

multiplique intermedia

almacene respuesta

cargue intermedia

reste unidad

vayasi itere fin

etiqueta itere 8

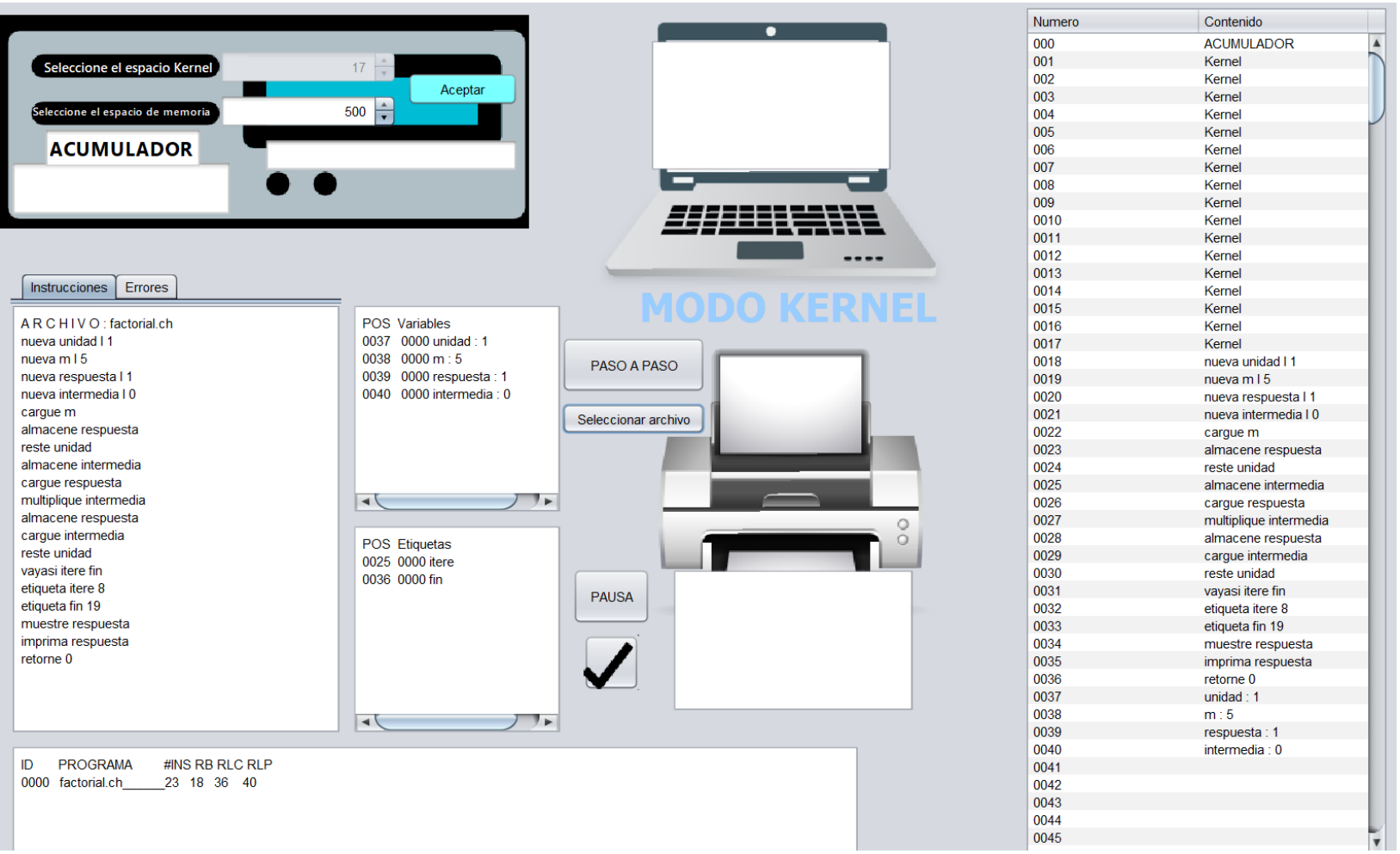
etiqueta fin 19

muestre respuesta

imprima respuesta

retorne 0

Observamos entonces que las instrucciones se muestran en la izquierda a modo de información únicamente, se observa también las etiquetas y la posición a la que corresponden, las variables (NOTA: la tabla de variables mostrará el valor de instancia, es decir, no cambiará en esta tabla, solo en el espacio en memoria), la



información del archivo en la parte inferior y en la memoria se observan las líneas y las variables ya creadas.

Este botón que observamos nos servirá para la ejecución de las instrucciones y hará lo siguiente:

```
    private void btnEjecutararchivosActionPerformed(java.awt.event.ActionEvent evt) {
643     LinkedList<String> lineas = this.lineasejecutar;
644     this.lblModo.setText(text: "MODO USUARIO");
645
646     // Inicializa el timer con un intervalo de 1000 milisegundos (1 segundo)
        timer = new Timer(delay: 500, new ActionListener() {
648         @Override
            public void actionPerformed(ActionEvent e) {
650             if (currentIndex < lineas.size()&&isRunning.get()) {
651                 String linea = lineas.get(index: currentIndex);
652                 String[] palabras = linea.trim().split(regex: "\\s+");
653                 palabras = Arrays.stream(array: palabras)
654                         .map(String::trim)
655                         .filter(palabra -> !palabra.isEmpty())
656                         .toArray(String[]::new);
657
658                 // Inicializa una variable para almacenar la instrucción
659                 StringBuilder instruccion = new StringBuilder();
660
661                 // Comienza desde palabras[3] hasta el final
662                 for (int i = 3; i < palabras.length; i++) {
663                     instruccion.append(palabras[i]);
664                     // Agrega un espacio si no es la última palabra
665                     if (i < palabras.length - 1) {
666                         instruccion.append(str:" ");
667                     }
668                 }
669
670                 // Establece el texto en txInstruccionejecutada
671                 if(!palabras[3].startsWith(prefix:"//")){
672                     txInstruccionejecutada.setText(palabras[0] + ">" + instruccion.toString());
673                 }
674                 lblArchivoejecutando.setText(palabras[1]);
675
676                 currentIndex++; // Avanza al siguiente índice
```

La lista de lineasejecutar, estará compuesta por las líneas válidas que el programa deberá ejecutar, estás tendrán la forma con el ejemplo:

0026 factorial.ch 0 nueva respuesta I 1

Lo que hacemos es juntar toda la información que tiene esa línea con el fin de identificar ¿Dónde está? ¿A qué archivo pertenece? ¿Qué debo hacer?, por eso es que hay un recorrido de todas estas líneas, las cuales son partidas por palabras y a partir de la posición 3 empiezan las órdenes o instrucciones.

Lo que observamos con Timer es una función que se encarga de darle delay al for para poder observar más detalladamente qué función se está ejecutando en ese momento, cuál archivo se está ejecutando y qué está pasando con las variables.

Lo que observamos a partir del switch case en resumidas cuentas es similar a la función examinarSintaxis() pero esta vez, si la primera palabra coincide con una instrucción definida entonces se realizará un recorrido de las variables o de las etiquetas según la necesidad de la instrucción, si coincide la variable indicada con la

```
680                 switch(palabrasInstruccion[0].toLowerCase()){
681                     case "cargue":
682                         for(String variableConIdentificadores:variablesConNombreeIdentificadorArchivo){
683                             String[] partesVariable = variableConIdentificadores.trim().split(regex: "\\s+");
684                             if(("000"+palabras[2]).equals(partesVariable[1]) && partesVariable[2].equals(palabrasInstruccion[1])){
685                                 String variableEnMemoria = (String)tablaEspacioMemoria.getValueAt(row:Integer.parseInt(partesVariable[0]), column:1);
686                                 String[] variableMemoria = variableEnMemoria.split(regex: "\\s+");
687                                 tablaEspacioMemoria.setValueAt(variableMemoria[2], row:0, column:1);
688                                 txtAcumulador.setText(variableMemoria[2]);
689                             }
690                         }
691                         break;
692                     case "almacene":
693                         for(String variableConIdentificadores:variablesConNombreeIdentificadorArchivo){
694                             String[] partesVariable = variableConIdentificadores.trim().split(regex: "\\s+");
695                             if(("000"+palabras[2]).equals(partesVariable[1]) && partesVariable[2].equals(palabrasInstruccion[1])){
696                                 int posicionVariable = Integer.parseInt(partesVariable[0]);
697                                 tablaEspacioMemoria.setValueAt(palabrasInstruccion[1]+" : "+tablaEspacioMemoria.getValueAt(row:0, column:1), row:posicionVar
698                             }
699                         }
700                         break;
701                     case "sume":
702                         for(String variableConIdentificadores:variablesConNombreeIdentificadorArchivo){
703                             String[] partesVariable = variableConIdentificadores.trim().split(regex: "\\s+");
704                             if(("000"+palabras[2]).equals(partesVariable[1]) && partesVariable[2].equals(palabrasInstruccion[1])){
705                                 String variableEnMemoria = (String)tablaEspacioMemoria.getValueAt(row:Integer.parseInt(partesVariable[0]), column:1);
706                                 String[] variableMemoria = variableEnMemoria.split(regex: "\\s+");
707                                 double suma = (Double.parseDouble(s: tablaEspacioMemoria.getValueAt(row:0, column:1).toString())
708                                         +Double.parseDouble(variableMemoria[2]));
709                                 tablaEspacioMemoria.setValueAt(aValue:suma , row:0, column:1);
710                                 txtAcumulador.setText(""+suma);
711                             }
```

posición que se ha recorrido entonces se hará con la variable lo que la instrucción diga.

```
713                     case "reste":
714                         for(String variableConIdentificadores:variablesConNombreeIdentificadorArchivo){
715                             String[] partesVariable = variableConIdentificadores.trim().split(regex: "\\s+");
716                             if(("000"+palabras[2]).equals(partesVariable[1]) && partesVariable[2].equals(palabrasInstruccion[1])){
717                                 String variableEnMemoria = (String)tablaEspacioMemoria.getValueAt(row:Integer.parseInt(partesVariable[0]), column:1);
718                                 String[] variableMemoria = variableEnMemoria.split(regex: "\\s+");
719                                 double resta = (Double.parseDouble(s: tablaEspacioMemoria.getValueAt(row:0, column:1).toString())
720                                         -Double.parseDouble(variableMemoria[2]));
721                                 tablaEspacioMemoria.setValueAt(aValue:resta , row:0, column:1);
722                                 txtAcumulador.setText(""+resta);
723                             }
724                         }
725                         break;
726                     case "multiplique":
727                         for(String variableConIdentificadores:variablesConNombreeIdentificadorArchivo){
728                             String[] partesVariable = variableConIdentificadores.trim().split(regex: "\\s+");
729                             if(("000"+palabras[2]).equals(partesVariable[1]) && partesVariable[2].equals(palabrasInstruccion[1])){
730                                 String variableEnMemoria = (String)tablaEspacioMemoria.getValueAt(row:Integer.parseInt(partesVariable[0]), column:1);
731                                 String[] variableMemoria = variableEnMemoria.split(regex: "\\s+");
732                                 double multiplicacion = (Double.parseDouble(s: tablaEspacioMemoria.getValueAt(row:0, column:1).toString())
733                                         *Double.parseDouble(variableMemoria[2]));
734                                 tablaEspacioMemoria.setValueAt(aValue:multiplicacion , row:0, column:1);
735                                 txtAcumulador.setText(""+multiplicacion);
736                             }
737                         }
738                         break;
739                     case "divida":
740                         for(String variableConIdentificadores:variablesConNombreeIdentificadorArchivo){
741                             String[] partesVariable = variableConIdentificadores.trim().split(regex: "\\s+");
742                             if(("000"+palabras[2]).equals(partesVariable[1]) && partesVariable[2].equals(palabrasInstruccion[1])){
743                                 String variableEnMemoria = (String)tablaEspacioMemoria.getValueAt(row:Integer.parseInt(partesVariable[0]), column:1);
744                                 String[] variableMemoria = variableEnMemoria.split(regex: "\\s+");
745                                 if(Double.parseDouble(variableMemoria[2])!=0.0){
```

```
                        double division = (Double.parseDouble(s: tablaEspacioMemoria.getValueAt(row: 0, column: 1).toString())
                                /Double.parseDouble(variableMemoria[2]));
                        tablaEspacioMemoria.setValueAt(aValue: division , row: 0, column: 1);
                        txtAcumulador.setText(""+division);
                        }
                        else{
                            JOptionPane.showMessageDialog(parentComponent: null, message: "El dividendo es 0\nel acumulador seguirá igual");
                        }
                    }
                }
                break;
            case "potencia":
                for(String variableConIdentificadores:variablesConNombreeIdentificadorArchivo){
                    String[] partesVariable = variableConIdentificadores.trim().split(regex: "\\s+");
                    if(("000"+palabras[2]).equals(partesVariable[1]) && partesVariable[2].equals(palabrasInstruccion[1])){
                        String variableEnMemoria = (String)tablaEspacioMemoria.getValueAt(row: Integer.parseInt(partesVariable[0]), column: 1);
                        String[] variableMemoria = variableEnMemoria.split(regex: "\\s+");
                        double potencia = Math.pow(a: Double.parseDouble(s: tablaEspacioMemoria.getValueAt(row: 0, column: 1).toString())
                                ,b: Double.parseDouble(variableMemoria[2]));
                        tablaEspacioMemoria.setValueAt(aValue: potencia , row: 0, column: 1);
                        txtAcumulador.setText(""+potencia);
                    }
                }
                break;
            case "modulo":
                for(String variableConIdentificadores:variablesConNombreeIdentificadorArchivo){
                    String[] partesVariable = variableConIdentificadores.trim().split(regex: "\\s+");
                    if(("000"+palabras[2]).equals(partesVariable[1]) && partesVariable[2].equals(palabrasInstruccion[1])){
                        String variableEnMemoria = (String)tablaEspacioMemoria.getValueAt(row: Integer.parseInt(partesVariable[0]), column: 1);
                        String[] variableMemoria = variableEnMemoria.split(regex: "\\s+");
                        double modulo = (Double.parseDouble(s: tablaEspacioMemoria.getValueAt(row: 0, column: 1).toString())
                              %Double.parseDouble(variableMemoria[2]));
                        tablaEspacioMemoria.setValueAt(aValue: modulo , row: 0, column: 1);
                        txtAcumulador.setText(""+modulo);
                    }
                }
                break;
            case "concatene":
                for(String variableConIdentificadores:variablesConNombreeIdentificadorArchivo){
                    String[] partesVariable = variableConIdentificadores.trim().split(regex: "\\s+");
                    if(("000"+palabras[2]).equals(partesVariable[1]) && partesVariable[2].equals(palabrasInstruccion[1])){
                        String variableEnMemoria = (String)tablaEspacioMemoria.getValueAt(row: Integer.parseInt(partesVariable[0]), column: 1);
                        String[] variableMemoria = variableEnMemoria.split(regex: "\\s+");
                        String nuevaCadena = (tablaEspacioMemoria.getValueAt(row: 0, column: 1).toString())
                              +(variableMemoria[2]);
                        tablaEspacioMemoria.setValueAt(aValue: nuevaCadena , row: 0, column: 1);
                        txtAcumulador.setText(""+nuevaCadena);
                    }
                }
                break;
            case "elimine":
                        String nuevaCadena = (tablaEspacioMemoria.getValueAt(row: 0, column: 1).toString());
                        nuevaCadena = nuevaCadena.replace(palabrasInstruccion[1], replacement: "");
                        tablaEspacioMemoria.setValueAt(aValue: nuevaCadena , row: 0, column: 1);
                        txtAcumulador.setText(t: nuevaCadena);
                break;
            case "extraiga":
                        String extraido = (tablaEspacioMemoria.getValueAt(row: 0, column: 1).toString());
                        if(Integer.parseInt(palabrasInstruccion[1])<=extraido.length()){
                            extraido = extraido.substring(beginIndex: 0,endIndex: Integer.parseInt(palabrasInstruccion[1]));
```

```
                    tablaEspacioMemoria.setValueAt(aValue:extraido , row:0, column:1);
                    txtAcumulador.setText(t: extraido);
                    }
                else{
                    JOptionPane.showMessageDialog(parentComponent: null, "El valor de extracción es mayor"
                            + " que la cantidad de carácteres del acumulador\n La cadena sigue igual");
                }
                break;
            case "y":
                int operando1Y = 0;
                int operando2Y = 0;
                for(String variableConIdentificadores:variablesConNombreeIdentificadorArchivo){
                    String[] partesVariable = variableConIdentificadores.trim().split(regex: "\\s+");
                    if(("000"+palabras[2]).equals(partesVariable[1]) && partesVariable[2].equals(palabrasInstruccion[1])){
                        String variableEnMemoria = (String)tablaEspacioMemoria.getValueAt(row:Integer.parseInt(partesVariable[0]
                        String[] variableMemoria = variableEnMemoria.split(regex: "\\s+");
                        operando1Y = Integer.parseInt(variableMemoria[2]);
                    }
                    if(("000"+palabras[2]).equals(partesVariable[1]) && partesVariable[2].equals(palabrasInstruccion[2])){
                        String variableEnMemoria = (String)tablaEspacioMemoria.getValueAt(row:Integer.parseInt(partesVariable[0]
                        String[] variableMemoria = variableEnMemoria.split(regex: "\\s+");
                        operando2Y = Integer.parseInt(variableMemoria[2]);
                    }
                }
                for(String variableConIdentificadores:variablesConNombreeIdentificadorArchivo){
                    String[] partesVariable = variableConIdentificadores.trim().split(regex: "\\s+");
                    if(("000"+palabras[2]).equals(partesVariable[1]) && partesVariable[2].equals(palabrasInstruccion[3])){
                        int posicionVariable = Integer.parseInt(partesVariable[0]);
                        tablaEspacioMemoria.setValueAt(palabrasInstruccion[3]+" : "+(operando1Y*operando2Y), row:posicionVariabl
                    }
                }
                break;
            case "o":
                int operando1O = 0;
                int operando2O = 0;
                for(String variableConIdentificadores:variablesConNombreeIdentificadorArchivo){
                    String[] partesVariable = variableConIdentificadores.trim().split(regex: "\\s+");
                    if(("000"+palabras[2]).equals(partesVariable[1]) && partesVariable[2].equals(palabrasInstruccion[1])){
                        String variableEnMemoria = (String)tablaEspacioMemoria.getValueAt(row:Integer.parseInt(partesVariable[0]), column:1);
                        String[] variableMemoria = variableEnMemoria.split(regex: "\\s+");
                        operando1O = Integer.parseInt(variableMemoria[2]);
                    }
                    if(("000"+palabras[2]).equals(partesVariable[1]) && partesVariable[2].equals(palabrasInstruccion[2])){
                        String variableEnMemoria = (String)tablaEspacioMemoria.getValueAt(row:Integer.parseInt(partesVariable[0]), column:1);
                        String[] variableMemoria = variableEnMemoria.split(regex: "\\s+");
                        operando2O = Integer.parseInt(variableMemoria[2]);
                    }
                }
                for(String variableConIdentificadores:variablesConNombreeIdentificadorArchivo){
                    String[] partesVariable = variableConIdentificadores.trim().split(regex: "\\s+");
                    if(("000"+palabras[2]).equals(partesVariable[1]) && partesVariable[2].equals(palabrasInstruccion[3])){
                        int posicionVariable = Integer.parseInt(partesVariable[0]);
                        tablaEspacioMemoria.setValueAt(palabrasInstruccion[3]+" : "+(operando1O+operando2O), row:posicionVariable, column:1);
                    }
                }
                break;
            case "no":
                for(String variableConIdentificadores:variablesConNombreeIdentificadorArchivo){
                    String[] partesVariable = variableConIdentificadores.trim().split(regex: "\\s+");
                    if(("000"+palabras[2]).equals(partesVariable[1]) && partesVariable[2].equals(palabrasInstruccion[1])){
                        String variableEnMemoria = (String)tablaEspacioMemoria.getValueAt(row:Integer.parseInt(partesVariable[0]), column:1);
                        String[] variableMemoria = variableEnMemoria.split(regex: "\\s+");
                        int operandoNO = Integer.parseInt(variableMemoria[2]);
                        if(operandoNO == 1){
                            operandoNO = 0;
                        }
                        else{
                            operandoNO = 1;
                        }
```

```
                    for(String variableConIdentificadores2:variablesConNombreeIdentificadorArchivo){
                        String[] partesVariable2 = variableConIdentificadores2.trim().split(regex: "\\s+");
                        if(("000"+palabras[2]).equals(partesVariable2[1]) && partesVariable2[2].equals(palabrasInstruccion[2])){
                            int posicionVariable = Integer.parseInt(partesVariable2[0]);
                            tablaEspacioMemoria.setValueAt(palabrasInstruccion[2]+" : "+(operandoNO), row:posicionVariable, column:1);
                        }
                    }
                }
            }
            break;
        case "lea":
            for(String variableConIdentificadores:variablesConNombreeIdentificadorArchivo){
                String[] partesVariable = variableConIdentificadores.trim().split(regex: "\\s+");
                if(("000"+palabras[2]).equals(partesVariable[1]) && partesVariable[2].equals(palabrasInstruccion[1])){
                    String variableEnMemoria = (String)tablaEspacioMemoria.getValueAt(row:Integer.parseInt(partesVariable[0]), column:1);
                    String textoPantalla = txPantalla.getText();
                    textoPantalla += "archivo 000"+palabras[2]+": "+variableEnMemoria+"\n";
                    txPantalla.setText(t: textoPantalla);
                }
            }
            break;
        case "muestre":
            boolean mostro = false;
            for(String variableConIdentificadores:variablesConNombreeIdentificadorArchivo){
                String[] partesVariable = variableConIdentificadores.trim().split(regex: "\\s+");
                if(("000"+palabras[2]).equals(partesVariable[1]) && partesVariable[2].equals(palabrasInstruccion[1])){
                    String variableEnMemoria = (String)tablaEspacioMemoria.getValueAt(row:Integer.parseInt(partesVariable[0]), column:1);
                    String textoPantalla = txPantalla.getText();
                    textoPantalla += "archivo 000"+palabras[2]+": "+variableEnMemoria+"\n";
                    txPantalla.setText(t: textoPantalla);
                    mostro = true;
                }
            }
            if(palabrasInstruccion[1].equalsIgnoreCase(anotherString:"acumulador")){
                String textoPantalla = txPantalla.getText();
                textoPantalla += "archivo 000"+palabras[2]+": ACUMULADOR = "+
                tablaEspacioMemoria.getValueAt(row:0, column:1).toString()+"\n";
                txPantalla.setText(t: textoPantalla);
                mostro = true;
            }
            if(mostro!=true){
                String textoPantalla = txPantalla.getText();
                textoPantalla += palabrasInstruccion[1]+"\n";
                txPantalla.setText(t: textoPantalla);
            }

            break;
        case "imprima":
            boolean mostro2 = false;
            for(String variableConIdentificadores:variablesConNombreeIdentificadorArchivo){
                String[] partesVariable = variableConIdentificadores.trim().split(regex: "\\s+");
                if(("000"+palabras[2]).equals(partesVariable[1]) && partesVariable[2].equals(palabrasInstruccion[1])){
                    String variableEnMemoria = (String)tablaEspacioMemoria.getValueAt(row:Integer.parseInt(partesVariable[0]), column:1);
                    String textoPantalla = txImpresora.getText();
                    textoPantalla += "archivo 000"+palabras[2]+": "+variableEnMemoria+"\n";
                    txImpresora.setText(t: textoPantalla);
                    mostro = true;
                }
            }
            if(palabrasInstruccion[1].equalsIgnoreCase(anotherString:"acumulador")){
                String textoImpresora = txImpresora.getText();
                textoImpresora += "archivo 000"+palabras[2]+": ACUMULADOR = "+
                tablaEspacioMemoria.getValueAt(row:0, column:1).toString()+"\n";
                txImpresora.setText(t: textoImpresora);
                mostro = true;
            }
            if(mostro2!=true){
                String textoImpresora = txImpresora.getText();
                textoImpresora += palabrasInstruccion[1]+"\n";
                txImpresora.setText(t: textoImpresora);
```

```
945                     }
946
947                 break;
948             case "retorne":
949                 String texto = txPantalla.getText();
950                 texto += "Termino el programa llamado "+palabras[1]+" con identificador 000"+palabras[2]+"\n";
951                 txPantalla.setText(t: texto);
952                 break;
953             case "vaya":
954                 for(String etiquetaConIdentificadores:etiquetasconNombreeIdentificadorArchivo){
955                     String[] partesEtiqueta = etiquetaConIdentificadores.trim().split(regex: "\\s+");
956                     if(("000"+palabras[2]).equals(partesEtiqueta[1]) && partesEtiqueta[2].equals(palabrasInstruccion[1])){
957                         int posicionEtiqueta = Integer.parseInt(partesEtiqueta[0]);
958                         int contador = 0;
959                         int posicionDefinitiva = 0;
960                         int contadorlineascomentadas = 0;
961                         for(String i:lineasejecutar){
962                             String[] instruccionPartida = i.split(regex: "\\s+");
963                             System.out.println(instruccionPartida[0]+"es diferente de "+posicionEtiqueta);
964                             if(instruccionPartida[0].equals("00"+(posicionEtiqueta-contadorlineascomentadas))){
965                                 posicionDefinitiva = contador;
966                             }
967                             else{
968                                  if(i.startsWith(prefix:"//")){
969                                         contadorlineascomentadas++;
970                                     }
971                                     contador++;
972                             }
973                         }
974                         currentIndex = currentIndex-(currentIndex-posicionDefinitiva);
975                     }
976                 }
977                 break;
978             case "vayasi":
979                 if(Double.parseDouble(s: tablaEspacioMemoria.getValueAt(row:0, column:1).toString())>0.0){
980                     for(String etiquetaConIdentificadores:etiquetasconNombreeIdentificadorArchivo){
981                         String[] partesEtiqueta = etiquetaConIdentificadores.trim().split(regex: "\\s+");
982                         if(("000"+palabras[2]).equals(partesEtiqueta[1]) && partesEtiqueta[2].equals(palabrasInstruccion[1])){
983                             System.out.println("000"+palabras[2]+" es igual a "+partesEtiqueta[1]+" y "+partesEtiqueta[2]+" es igual a "+pal
984                             int posicionEtiqueta = Integer.parseInt(partesEtiqueta[0]);
985                             int contador = 0;
986                             int contadorlineascomentadas = 0;
987                             int posicionDefinitiva = 0;
988                             for(String i:lineasejecutar){
989                                 System.out.println(x: i);
990                                 String[] instruccionPartida = i.split(regex: "\\s+");
991                                 if(instruccionPartida[0].equals("00"+(posicionEtiqueta-contadorlineascomentadas))){
992                                     posicionDefinitiva = contador;
993                                 }
994                                 else{
995                                     if(i.startsWith(prefix:"//")){
996                                         contadorlineascomentadas++;
997                                     }
998                                     contador++;
999                                 }
1000                            }
1001                            System.out.println(currentIndex+"-"+(currentIndex)+"-"+posicionDefinitiva);
1002                            currentIndex = currentIndex-(currentIndex-posicionDefinitiva);
1003                        }
1004                    }
1005                }
1006                if(Double.parseDouble(s: tablaEspacioMemoria.getValueAt(row:0, column:1).toString())<0){
1007                    for(String etiquetaConIdentificadores:etiquetasconNombreeIdentificadorArchivo){
1008                        String[] partesEtiqueta = etiquetaConIdentificadores.trim().split(regex: "\\s+");
1009                        if(("000"+palabras[2]).equals(partesEtiqueta[1]) && partesEtiqueta[2].equals(palabrasInstruccion[2])){
1010                            System.out.println("000"+palabras[2]+" es igual a "+partesEtiqueta[1]+" y "+partesEtiqueta[2]+" es igual a "+pal
1011                            int posicionEtiqueta = Integer.parseInt(partesEtiqueta[0]);
1012                            int contador = 0;
1013                            int posicionDefinitiva = 0;
1014                            int contadorlineascomentadas = 0;
```

```
                                    for(String i:lineasejecutar){
                                        System.out.println(x: i);
                                        String[] instruccionPartida = i.split(regex: "\\s+");
                                        System.out.println(instruccionPartida[0]+"es diferente de "+posicionEtiqueta);
                                        if(instruccionPartida[0].equals("00"+(posicionEtiqueta-contadorlineascomentadas))){
                                            posicionDefinitiva = contador;
                                        }
                                        else{
                                            if(i.startsWith(prefix:"//")){
                                                contadorlineascomentadas++;
                                            }
                                            contador++;
                                            }
                                        }

                                    System.out.println(currentIndex+"-"+(currentIndex)+"-"+posicionDefinitiva);
                                    currentIndex = currentIndex-(currentIndex-posicionDefinitiva);
                                }
                            }
                        }
                    break;
                case "dinosaurio":
                        DinosaurAnimation dinosaurAnimation = new DinosaurAnimation();
                        break;
            }
            if(modoPasoAPaso){
                int respuesta = JOptionPane.showConfirmDialog(parentComponent: null, message: "¿Continuar con modo paso a paso?",title
                if(respuesta == JOptionPane.NO_OPTION){
                    modoPasoAPaso = false;
                }
            }
        }
    }
});

// Inicia el timer
```

En los últimos 3 case considero que una explicación no sobra, en el vaya y vaya si comprobamos si las etiquetas dadas por los operandos coinciden tanto con las etiquetas instanciadas como con el identificador del archivo (esto con el fin de no confundir etiquetas en caso de que se adjunte el mismo archivo dos veces), hacemos el recorrido en la lista de lineasEjecutar y si coincide la posición de la etiqueta con la línea entonces el recorrido nos da un número, a través de una variable llamada CurrentIndex podemos verificar en qué instrucción vamos, entonces, si el vaya si indica que nos retrocedamos le restaremos un número a currentIndex para que este se devuelva y siga la ejecución a partir de ahí.

Documentación FASE B:

Debido a que las líneas a ejecutar se guardan en una lista, la implementación de los algoritmos de planificación fue un poco más sencilla y trabajable.

Podemos observar un cambio en la interfaz gráfica.

Se puede ver un selector desplegable que nos permite indicar a la máquina el tipo de algoritmo de planificación que queremos usar, así mismo se observa un spin que será el quantum de tiempo en caso requerido por el algoritmo. Cabe aclarar que ambas opciones nuevamente agregadas son desactivadas al adjuntar un archivo y que si el usuario no ha seleccionado un algoritmo requerido no podrá ejecutar ni adjuntar archivos.

```
391     public void OrganizarAlgoritmoDePlanificacion(){
392         LinkedList<String> listaAuxiliar = new LinkedList<>();
            switch((String)this.comboPlanificación.getSelectedItem()){
394             case "Planificación por prioridad (Seleccionar número de prioridad) No Expropiativo UNIX":
395                 boolean condicion = false;
396                 while (!condicion){
397                     condicion = true;
398                     for (int i = this.currentIndex; i < this.archivosPrioridad.size()-1; i++) {
399                         String[] prioridadporPartes = this.archivosPrioridad.get(index: i).trim().split(regex: "\\s");
400                         String[] prioridadporPartes1 = this.archivosPrioridad.get(i+1).trim().split(regex: "\\s");
401                         if (Integer.parseInt(prioridadporPartes[1]) > Integer.parseInt(prioridadporPartes1[1])){
402                             String aux = this.archivosPrioridad.get(index: i);
403                             this.archivosPrioridad.set(index: i, element: this.archivosPrioridad.get(i+1));
404                             this.archivosPrioridad.set(i+1,element: aux);
405                             condicion = false;
406                         }
407                     }
408                 }
409                 for(String prioridad:this.archivosPrioridad){
410                     for (int j = 0; j < this.lineasejecutar.size()-1; j++) {
411                         String[] prioridadPorPartes = prioridad.trim().split(regex: "\\s");
412                         String[] lineaPorPartes = this.lineasejecutar.get(index: j).trim().split(regex: "\\s");
413                         if(lineaPorPartes[2].equals(prioridadPorPartes[0])){
414                             listaAuxiliar.add(e: this.lineasejecutar.get(index: j));
415                         }
416                     }
417                 }
418                 this.lineasejecutar.clear();
419                 this.lineasejecutar.addAll(c: listaAuxiliar);
420                 break;
421             case "SJF (Tiempo más corto, primero en ser atendido) No expropiativo":
422                 boolean condicion4 = false;
423                 while (!condicion4){
424                     condicion4 = true;
425                     for (int i = this.currentIndex; i < this.archivosTiempo.size()-1; i++) {
426                         String[] prioridadporPartes = this.archivosTiempo.get(index: i).trim().split(regex: "\\s");
427                         String[] prioridadporPartes1 = this.archivosTiempo.get(i+1).trim().split(regex: "\\s");
428                         if (Integer.parseInt(prioridadporPartes[1]) > Integer.parseInt(prioridadporPartes1[1])){
429                             String aux = this.archivosTiempo.get(index: i);
430                             this.archivosTiempo.set(index: i, element: this.archivosTiempo.get(i+1));
431                             this.archivosTiempo.set(i+1,element: aux);
432                             condicion4 = false;
433                         }
434                     }
435                 }
436                 for(String tiempo:this.archivosTiempo){
437                     for (int j = 0; j < this.lineasejecutar.size(); j++) {
438                         String[] prioridadPorPartes = tiempo.trim().split(regex: "\\s");
439                         String[] lineaPorPartes = this.lineasejecutar.get(index: j).trim().split(regex: "\\s");
440                         if(lineaPorPartes[2].equals(prioridadPorPartes[0])){
441                             listaAuxiliar.add(e: this.lineasejecutar.get(index: j));
442                         }
443                     }
444                 }
445                 this.lineasejecutar.clear();
446                 this.lineasejecutar.addAll(c: listaAuxiliar);
447                 break;
448             case "SJF (Tiempo más corto, primero en ser atendido) Expropiativo":
449                 boolean condicion3 = false;
450                 while (!condicion3){
451                     condicion3 = true;
452                     for (int i = currentIndex; i < this.archivosTiempo.size()-1; i++) {
453                         String[] prioridadporPartes = this.archivosTiempo.get(index: i).trim().split(regex: "\\s");
454                         String[] prioridadporPartes1 = this.archivosTiempo.get(i+1).trim().split(regex: "\\s");
455                         if (Integer.parseInt(prioridadporPartes[1]) > Integer.parseInt(prioridadporPartes1[1])){
456                             String aux = this.archivosTiempo.get(index: i);
457                             this.archivosTiempo.set(index: i, element: this.archivosTiempo.get(i+1));
458                             this.archivosTiempo.set(i+1,element: aux);
459                             condicion3 = false;
```

```
                        }
                    }
                }
                for(String tiempo:this.archivosTiempo){
                        for (int j = 0; j < this.lineasejecutar.size(); j++) {
                            String[] prioridadPorPartes = tiempo.trim().split(regex: "\\s");
                            String[] lineaPorPartes = this.lineasejecutar.get(index: j).trim().split(regex: "\\s");
                            if(lineaPorPartes[2].equals(prioridadPorPartes[0])){
                                listaAuxiliar.add(e: this.lineasejecutar.get(index: j));
                            }
                        }
                }
                this.lineasejecutar.clear();
                this.lineasejecutar.addAll(c: listaAuxiliar);
                break;
            case "RR (Cada proceso será atendido por un lapso de tiempo y se irá a la cola)":
                int quantum = (int) this.spinQuantum.getValue();

                // Crear objetos Archivo y organizar las instrucciones
                for (String archivo : this.archivosTiempo) {
                    String[] archivoporpartes = archivo.trim().split(regex: "\\s");
                    Archivo nuevoArchivo = new Archivo(archivoporpartes[0]);

                    // Verificar si el archivo ya existe en programa.getMisArchivos()
                    boolean archivoExistente = false;
                    for (Archivo arch : programa.getMisArchivos()) {
                        if (arch.getIdentificador().equals(anObject: nuevoArchivo.getIdentificador())) {
                            archivoExistente = true;
                            break;
                        }
                    }

                    // Si el archivo no existe, agregarlo y organizar sus instrucciones
                    if (!archivoExistente) {
                        programa.getMisArchivos().add(e: nuevoArchivo);
                        for (String linea : this.lineasejecutar) {
                            String[] lineaporpartes = linea.trim().split(regex: "\\s");
                            if (lineaporpartes[2].equals(archivoporpartes[0])) {
                                nuevoArchivo.getMislineas().add(e: linea);
                            }
                        }
                    }
                }

                int contadorlinea = 0;

                // Iterar hasta que se recorran todas las instrucciones de todos los archivos
                while (true) {
                    boolean algunaInstruccionAgregada = false;

                    // Iterar sobre cada archivo y agregar las próximas instrucciones según el quantum
                    for (int j = contadorArchivos; j < programa.getMisArchivos().size(); j++) {
                        int totalInstrucciones = programa.getMisArchivos().get(index: j).getMislineas().size();
                        for (int i = contadorlinea; i < contadorlinea + quantum && i < totalInstrucciones; i++) {
                            listaAuxiliar.add(e: programa.getMisArchivos().get(index: j).getMislineas().get(index: i));
                            algunaInstruccionAgregada = true;
                        }
                    }

                    // Si no se agregó ninguna instrucción en esta iteración, salir del bucle
                    if (!algunaInstruccionAgregada) {
                        break;
                    }

                    // Incrementar el contador de línea para la próxima iteración
                    contadorlinea += quantum;
                }

                // Actualizar las instrucciones a ejecutar con las instrucciones organizadas
                this.lineasejecutar.clear();
```

```
530                 this.lineasejecutar.addAll(c: listaAuxiliar);
531                 this.contadorArchivos = programa.getMisArchivos().size();
532                 currentIndex = 0;
533                 break;
534
535         }
536     }
537
```

El código que acabamos de ver es el código que organiza la lista según el algoritmo seleccionado, este código se llama en el botón de ejecutar archivos justo antes de hacer todas las validaciones y operaciones

INSTRUCCIÓN CREADA POR MÍ:

La instrucción “dinosaurio” es una instrucción que agregué yo, cuando se llame en la línea solo tendrá esta instrucción y mostrará una ventana emergente en la que se observa un muñequito de 8 bits corriendo de un lado a otro y servirá a modo de escape del programa, si se cierra la ventana emergente se sale de la ejecución automáticamente.