



11/22/2023

Proyecto Final

Sistemas Operativos

Equipo 2:

- CERVANTES CANDIA SAÚL – 177927
- CUENCA ESQUIVEL ANA KAREN – 177932
- LEANDRO SOSA MICHELL ALEXA – 178630
- RIVERA CARREON BRIAN ISSAI – 178481

CONTENIDO

Objetivo	2
Introducción.....	2
Marco teórico	2
Requerimientos del Sistema	3
Análisis	3
Diseño	3
Implementación de los algoritmos de administración de memoria	8
Espacio.....	8
Primer ajuste	10
Mejor ajuste	12
Próximo ajuste.....	13
Liberación de memoria en ajustes	16
Fragmentación de memoria en ajustes	17
Sistema buddy.....	18
Liberación de memoria en el sistema buddy.....	20
Fragmentación del sistema buddy	21
Códigos auxiliares.....	22
Ejecución	57
Errores presentados	64
Resultados y conclusiones	64
Bibliografía	65
Anexo de responsabilidades	65

OBJETIVO

El objetivo principal de este proyecto es explorar estas técnicas, comprender su funcionamiento interno y evaluar su desempeño en términos de fragmentación, eficiencia y uso de recursos. A través de estas implementaciones en Java, buscamos adquirir una comprensión más profunda de cómo cada algoritmo aborda los desafíos inherentes a la gestión de la memoria en sistemas operativos.

INTRODUCCIÓN

En el mundo de los sistemas operativos, la gestión efectiva de la memoria es fundamental para optimizar el rendimiento del sistema. Este proyecto se enfoca en la implementación y comparación de cuatro técnicas de administración de memoria en Java: el primer ajuste, el mejor ajuste, el siguiente ajuste y el sistema buddy.

El **primer ajuste** asigna el primer bloque de memoria disponible que satisfaga las necesidades de un proceso. Por otro lado, el **mejor ajuste** busca el bloque de memoria más pequeño que pueda acomodar las necesidades del proceso, minimizando así la fragmentación.

El algoritmo del **siguiente ajuste** es similar al primer ajuste, pero comienza la búsqueda desde el último punto de asignación, reduciendo la fragmentación externa. Mientras tanto, el **sistema buddy** divide la memoria en bloques de tamaño potencia de 2 y asigna bloques contiguos para adaptarse a las solicitudes de memoria.

MARCO TEÓRICO

Uno de los métodos más simples para asignar la memoria consiste en dividirla en varias particiones de tamaño fijo. Cada partición puede contener exactamente un proceso, de modo que el grado de multiprogramación estará limitado por el número de particiones disponibles. En este método de particiones múltiples, cuando una partición está libre, se selecciona un proceso de la cola de entrada y se lo carga en dicha partición. Cuando llega un proceso y necesita memoria, el sistema explora ese conjunto en busca de un agujero que sea lo suficientemente grande como para albergar el proceso. Si el agujero es demasiado grande, se lo dividirá en dos partes, asignándose una parte al proceso que acaba de llegar y devolviendo la otra al conjunto de agujeros. Cuando el proceso termina, libera su bloque de memoria, que volverá a colocarse en el conjunto de agujeros.

Si el nuevo agujero es adyacente a otros agujeros, se combinan esos agujeros adyacentes para formar otros de mayor tamaño. Este procedimiento constituye un caso concreto del problema general de asignación dinámica de espacio de almacenamiento, que se ocupa de cómo satisfacer una solicitud de tamaño a partir de una lista de agujeros libres. Hay muchas soluciones a este problema, y las estrategias comúnmente utilizadas para seleccionar un agujero libre entre el conjunto de agujeros disponible son las de primer ajuste, mejor ajuste y peor ajuste.

- **Primer ajuste (first fit).** Se asigna el primer agujero que sea lo suficientemente grande. La acción puede comenzar desde el principio del conjunto de agujeros o en el punto en hubiera terminado la exploración anterior. Podemos detener la exploración en cuanto' encontramos un agujero libre que sea lo suficientemente grande.
- **Mejor ajuste (best fit).** Se asigna el agujero más pequeño que tenga el tamaño suficiente. Debemos^A explorar la lista completa, a menos que ésta esté ordenada según su tamaño. Esta estrategia^A hace que se genere el agujero más pequeño posible con la memoria que sobre en el agujero original.
- **Próximo ajuste (next fit):** En el primer ajuste, ya que todas las búsquedas empiezan al comienzo de la memoria, siempre se ocupan más frecuentemente los huecos que están al comienzo de la memoria que los que están al

final. El “próximo ajuste” intenta mejorar el desempeño al distribuir sus búsquedas más uniformemente sobre todo el espacio de memoria. Hace esto manteniendo el registro de qué hueco fue el último en ser asignado. La próxima búsqueda comienza en el último hueco asignado, no en el comienzo de la memoria.

El sistema de descomposición binaria (**buddy system**) asigna la memoria a partir de un segmento de tamaño fijo compuesto de páginas físicamente contiguas. La memoria se asigna a partir de este segmento mediante un asignador de potencias de 2, que satisface las solicitudes en unidades cuyo tamaño es una potencia de 2 (4 KB, 8 KB, 16 KB, etc.). Toda solicitud cuyas unidades no tengan el tamaño apropiado se redondeará hasta la siguiente potencia de 2 más alta.

REQUERIMIENTOS DEL SISTEMA

- Sistema operativo: cualquier sistema operativo moderno como Windows, Linux o macOS.
- Memoria RAM: se recomienda al menos 2 GB de RAM.
- Espacio en disco: se necesita suficiente espacio en disco para instalar el entorno de ejecución de Java.
- Java y cualquier librería adicional que pueda requerir el programa.
- Procesador: se recomienda un procesador de al menos 1 GHz de velocidad.
- Entorno de ejecución de Java: es necesario tener instalado un entorno de ejecución de Java en la computadora, como Java Development Kit (JDK).

ANÁLISIS

La implementación de los algoritmos de asignación de memoria (primer ajuste, próximo ajuste y mejor ajuste) y del sistema de asignación de memoria Buddy en Java se propone para explorar diversas técnicas de gestión de memoria. El análisis de viabilidad implica evaluar la capacidad de adaptarse dinámicamente a cambios en la demanda de memoria, así como determinar la complejidad y los costos asociados con cada enfoque. Además, se considera el rendimiento de cada algoritmo en términos de fragmentación y eficiencia. El sistema Buddy se examina en función de su capacidad para minimizar la fragmentación y asignar eficientemente bloques de memoria.

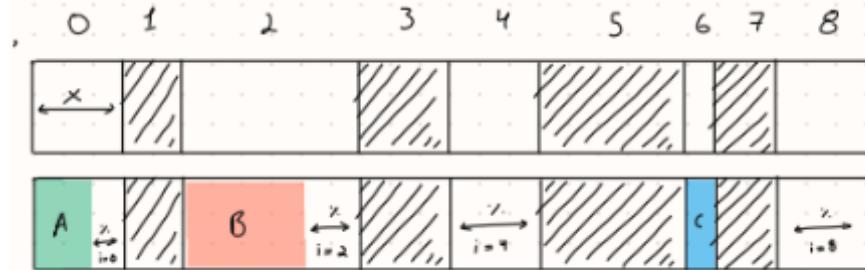
DISEÑO

Para la asignación de memoria se propone una estructura modular que permita la fácil implementación y adaptación, con interfaces claramente definidas para facilitar la interacción con el administrador de memoria. Cada algoritmo incorporará estrategias específicas para mitigar la fragmentación, ya sea mediante técnicas de compactación o mediante la aplicación de asignaciones más avanzadas. La adaptabilidad dinámica se considera esencial, permitiendo que el sistema responda de manera eficiente a cambios en la carga de trabajo, y se presta especial atención a la optimización de recursos para garantizar un rendimiento eficiente.

El sistema buddy su diseño contempla una estructura jerárquica de bloques de memoria que puede dividirse y combinarse según la técnica Buddy. Se implementaron algoritmos eficientes para la búsqueda y asignación de bloques de memoria, con el objetivo de minimizar la fragmentación y garantizar una asignación rápida y eficiente. La adaptabilidad dinámica se integra en el diseño, permitiendo ajustes automáticos en respuesta a cambios en la demanda de memoria, y se presta especial atención a la eficiencia en la gestión de la liberación de memoria y las fusiones de bloques.

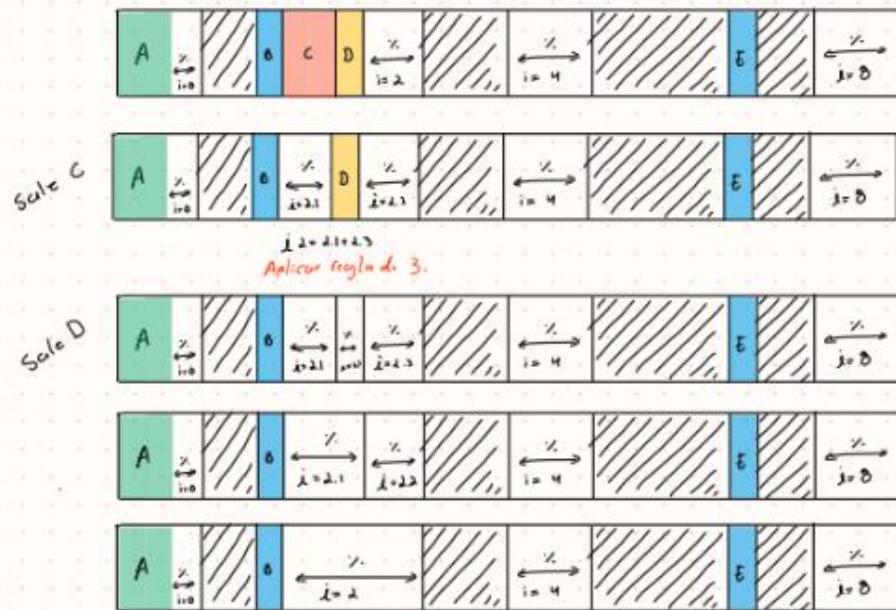
Ambos diseños buscan no solo abordar eficazmente los desafíos técnicos, como la fragmentación y la eficiencia en la asignación, sino también ofrecer una interfaz de programación clara

Fragmentación



- Existe el arreglo de número de espacios
- Inicializamos los no liberables con 0%
- Los espacios de proceso contienen en que Espacio inicial están
- Al momento de insertar se suma todo proceso que ya cuente con el mismo espacio inicial.
 - o Se hace regla de 3 y se actualiza el porcentaje
 - o ¿Se podría omitir el Arreglo si solo cuando salga se muestre la fragmentación?

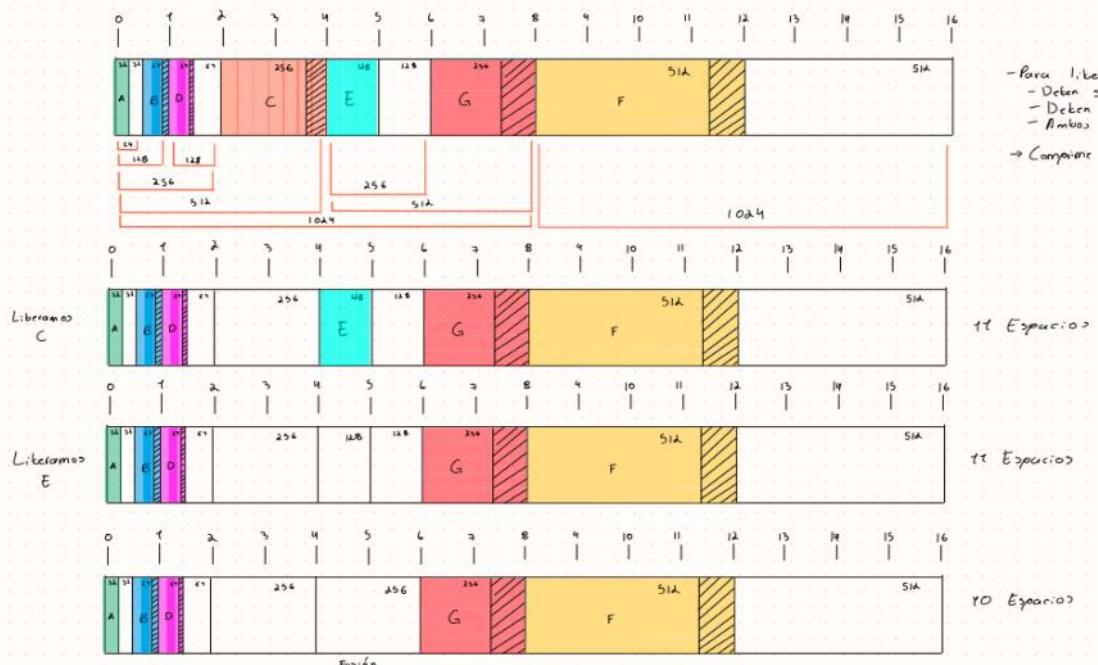
Liberar y Compresión



1. Detectar libres continuas
2. Cambiar tamaño de alguno (¿Cuál?)
3. Cambiar tipo
4. Eliminar el que no cambio tamaño ni tipo
5. ¿Comprobar fragmentación?

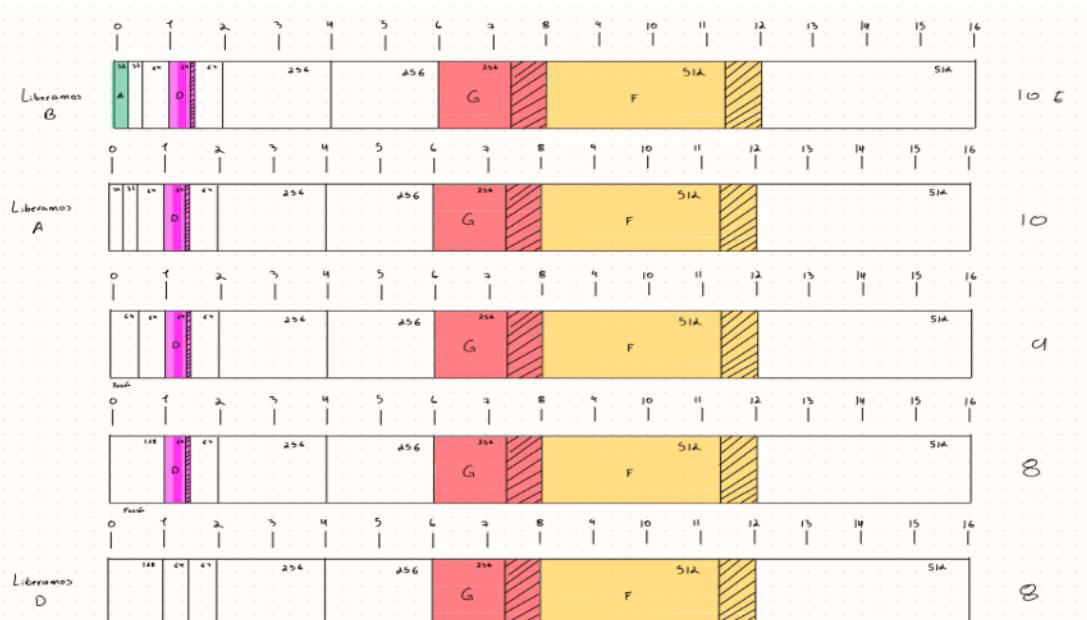
SISTEMAS OPERATIVOS

Memoria Buddy basada en Niveles

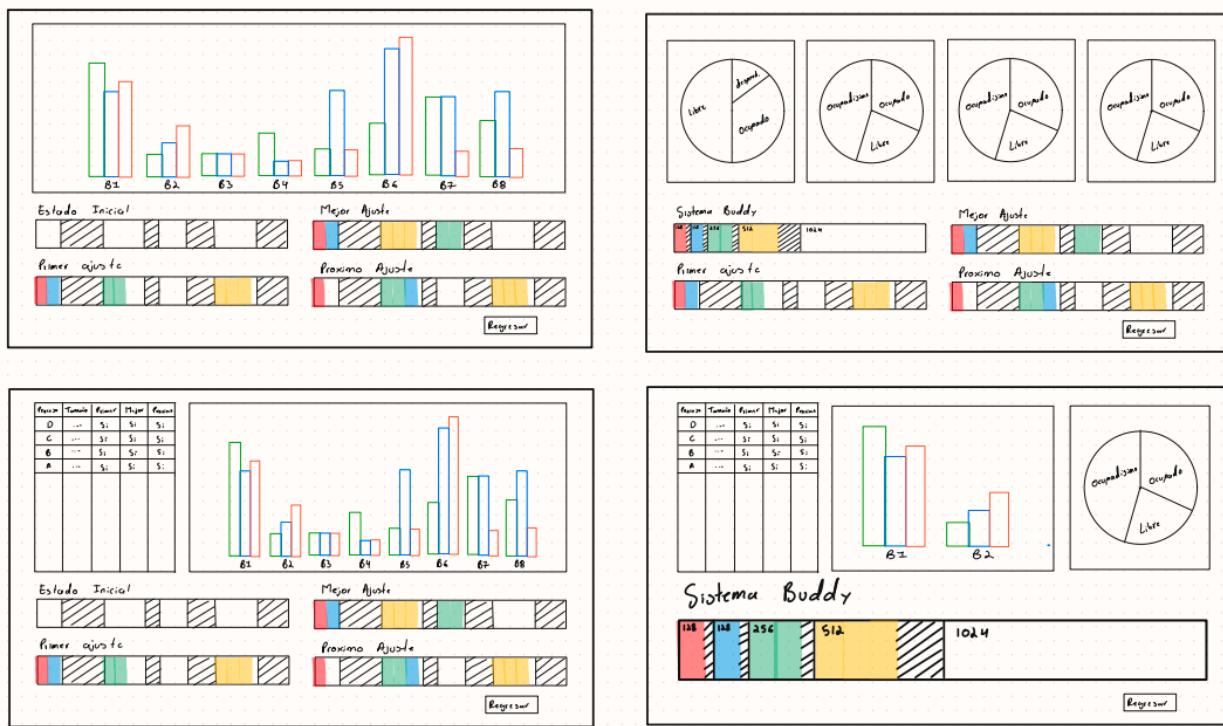


- Para liberar Compara 2 continuos

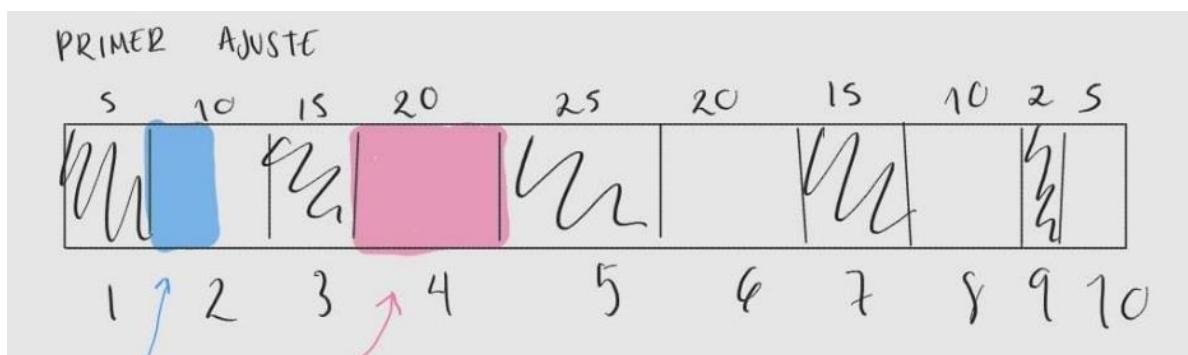
- Deben ser del mismo tamaño
- Deben ser del mismo mega inicial
- Ambos deben estar libres



SISTEMAS OPERATIVOS



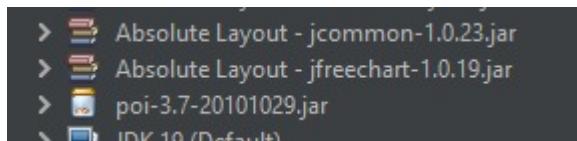
Ideas de la interfaz de los porcentajes de fragmentación



1. Recorre lista
 2. Comparar el tamaño ingresado con el primer bloque vacío
 3. El bloque azul, en caso de no ser del mismo tamaño, crea un espacio en la lista del tamaño de la resta del “vacío-ingresado” y cambiar el tamaño del vacío.
 4. El bloque rosa, en caso de ser del mismo tamaño, solo hay que agregar los datos del ingresado en el bloque vacío

IMPLEMENTACIÓN DE LOS ALGORITMOS DE ADMINISTRACIÓN DE MEMORIA

- **Librerías**



- **Paquetes:**

- **Img**

Contiene apoyo visual

- **Prueba**

Contiene la clase Validaciones, que sirve para limitar y controlar la entrada de datos en el programa.

- **Proyecto**

Contiene las clases:

ESPACIO

- **Espacio.** Clase que contiene variables destinadas a la administración del espacio de los bloques de asignación de procesos.

Todos los bloques de memoria mostrados en los Ajustes y en el sistema buddy son de clase Espacio la cual contiene diferentes parametros como el tipo de espacio que es, si es tipo 0.-Ocupado no liberable, 1.- Espacio libre, 2.- Espacio ocupado liberable. Dependiendo de que tipo de dato sea tendrá diferentes características como el nombre que es un atributo que solamente poseerán los bloques de memoria que han sido generados por procesos que ingresa el usuario con un tamaño predefinido o generado aleatoriamente, los nombres de los procesos se generarán de forma secuencial de la 'A' a la 'Z' y en caso de superarlo obtendrá el formato AA, AB, AC...

El desperdicio es una propiedad para el sistema buddy donde cada bloque de memoria almacena el espacio que fue desperdiciado en caso de que ingrese un proceso que cumpla con las características que lo ubican en ese lugar, pero sobre espacio del bloque.

nBloqueInicial es una propiedad auxiliar del espacio para calcular la fragmentación por bloques, todos los procesos que ingresasen en cualquier ajuste formaran parte del bloque inicial donde hayan sido ingresados y en el caso del sistema buddy es auxiliar en la identificación de los Megabytes individuales que componen la memoria y que de esta manera no se fusionen espacios bloques de memoria libres que no correspondan al mismo Megabyte padre

- **Atributos:**

- String: nombre
- Enteros: tamano, tipo, color [], desperdicio, nBloqueInicial
- Booleano: puntero

```

● ● ● Espacio.java
package proyectooperativos;
import java.util.ArrayList;
public class Espacio {
    String nombre; //Seran letras, si pasa de la Z sera ahora AA, AB, ...
    int tamano; //Tamaño del espacio
    int tipo; //0.-Ocupado no liberable, 1.- Espacio libre, 2.- Espacio ocupado liberable
    int color[] = new int[3];
    int desperdicio; //Para el buddy
    int nBloqueInicial; //Define cual es su espacio para considerar la fragmentación
    ArrayList <Integer> LHermanos = new ArrayList<>();
}

```

- **Métodos:**

- **Espacio.** Método constructor sobrecargado, que recibe determinados parametros, dependiendo de la funcionalidad del espacio que se desea usar:
 - Para cuando se inicializa en cualquier ajuste o buddy: String nombre, int tamano, int tipo

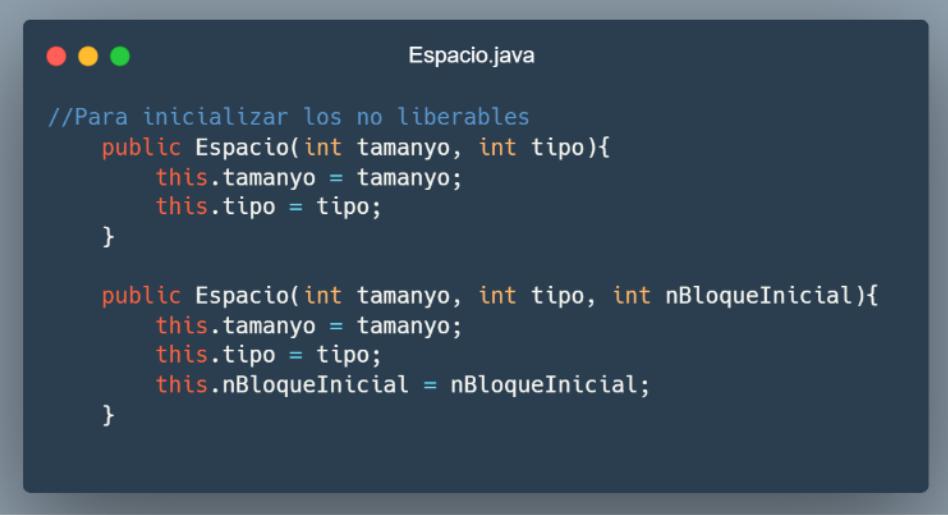
```

● ● ● Espacio.java
public Espacio(){
}

public Espacio(String nombre, int tamano, int tipo){
    this.nombre = nombre;
    this.tamano = tamano;
    this.tipo = tipo;
}

```

- Para inicializar los no liberables: Entero tamaño y tipo.
- Para el bloque inicial: Entero tamaño, tipo y nBloqueInicial



```

Espacio.java

//Para inicializar los no liberables
public Espacio(int tamano, int tipo){
    this.tamano = tamano;
    this.tipo = tipo;
}

public Espacio(int tamano, int tipo, int nBloqueInicial){
    this.tamano = tamano;
    this.tipo = tipo;
    this.nBloqueInicial = nBloqueInicial;
}

```

- Para bloques de procesos: Enteros: tipo, tamano, color0, color1, color2; String nombre.



```

Espacio.java

public Espacio(int tamano, int tipo, int nBloqueInicial, int nivel){
    this.tamano = tamano;
    this.tipo = tipo;
    this.nBloqueInicial = nBloqueInicial;
    this.nivel = nivel;
}

public Espacio(int tipo, String nombre, int tamano, int color0, int color1, int color2){
    this.tipo = tipo;
    this.nombre = nombre;
    this.tamano = tamano;
    this.color[0] = color0;
    this.color[1] = color1;
    this.color[2] = color2;
}

```

PRIMER AJUSTE

Clase encargada de llevar a cabo el proceso lógico para realizar el primer ajuste de memoria al ingresar un nuevo proceso mediante la utilización de la clase Sinc

El primer ajuste se encarga de agregar a la lista correspondiente del primer ajuste en el archivo sincronizado el nuevo bloque de memoria. En este ajuste el algoritmo recorre la lista buscando el primer espacio que cumpla con las características de ser un espacio vacío y que su tamaño sea mayor o igual al del bloque de memoria que se quiere ingresar, en caso de ser así se compara si es que el nuevo espacio de memoria es igual o de un tamaño diferente al que ya existe, en caso de ser igual los parametros del espacio de memoria a ingresar serán copiados en el espacio de memoria ya existente en la lista, en caso de ser diferente se realizará la diferencia entre los tamaños del espacio a ingresar y el existente, el tamaño del espacio que ya se encontraba en memoria se reducirá para que el proceso pueda entrar en el mismo índice de esta manera se conserva la misma cantidad de memoria inicial y se ingresa el proceso.

- Atributos
 - Boolean bandera.
 - Espacio NewEspacio

- Métodos
 - IngresarProceso()

```

● ● ● PrimerAjuste.java

package proyectooperativos;

import java.awt.Graphics;

public class PrimerAjuste {
    public int ingresaProceso(Espacio newEspacio, Graphics panel){
        //Aqui se debe agregar el nuevo espacio en la lista de "Sinc" correspondiente
        //Retorna true en caso de que si lo pueda insertar, de lo contrario false
        int bandera=0;
        int i = 0;

        if(newEspacio.tamanyo>Sinc.limiteSuperior){
            return 2;
        }

        for (i = 0; i < Sinc.LPrimer.size(); i++) {//recorre la lista

            //Actualiza la lista para mostrar la posicion del scanner
            Sinc.dibujaLista(Sinc.LPrimer, panel, i);
            Sinc.posScannerPrimer = i; //Actualiza la ultima posicion del scanner
            Sinc.esperar(0.18);

            if(Sinc.LPrimer.get(i).tipo == 1 ){// si hay un espacio vacio
                if(Sinc.LPrimer.get(i).tamanyo>=newEspacio.tamanyo){//checa si el newEspacio cabe en ese i
espacio

                    if(Sinc.LPrimer.get(i).tamanyo!=newEspacio.tamanyo){//ES DIFERENTE EL ESPACIO
                        Sinc.LPrimer.get(i).tamanyo=Sinc.LPrimer.get(i).tamanyo-newEspacio.tamanyo;
                        //agrega espacio antes uno nuevo con el tamano del nuevo
                        newEspacio.nBloqueInicial = Sinc.LPrimer.get(i).nBloqueInicial;
                        Sinc.LPrimer.add(i,newEspacio);
                    }else{// ES EL MISMO ESPACIO
                        //le ponemos todos los datos del nuevo a ese espacio vacio
                        Sinc.LPrimer.get(i).nombre = newEspacio.nombre;
                        Sinc.LPrimer.get(i).tipo = 2;//ahora es ocupado pero que se puede liberar
                        Sinc.LPrimer.get(i).color[0] = newEspacio.color[0];
                        Sinc.LPrimer.get(i).color[1] = newEspacio.color[1];
                        Sinc.LPrimer.get(i).color[2] = newEspacio.color[2];
                    }
                    bandera= 1;
                    break;
                }else{//es el ultimo y no cupo en algun espacio
                    bandera= 0;
                }
            }
        }

        if(!(i < Sinc.LPrimer.size())){
            Sinc.posScannerPrimer = -1; //En caso de entrar scanner = -1 = no muestra
        }
        return bandera;
    }
}

```

MEJOR AJUSTE

Clase encargada de llevar a cabo el proceso lógico para realizar el mejor ajuste de memoria al ingresar un nuevo proceso mediante la utilización de la clase Sinc.

LA lógica que sigue el algoritmo del mejor ajuste es que este recorre la lista correspondiente a la memoria del mejor ajuste y compara que el nuevo espacio a ingresar sea igual o menor a los espacios libres que se encuentre en su recorrido. Durante el escaneo de la memoria, si se encuentra algún espacio de memoria que cumpla exactamente con el espacio necesario para el nuevo bloque se copiaran los parámetros del proceso a ingresar, en caso de que no sea exacto el espacio se comparara el residuo que generaría ingresar el proceso en ese bloque de memoria, este proceso se realizará hasta terminar de recorrer la lista y el resultado de la variable indiceideal sería el mismo índice, donde la fragmentación de memoria es la menor, al finalizar el escaneo se realizará la diferencia entre el espacio existente en el índice ideal en tamaño con el proceso a ingresar, el espacio ocupará el residuo de la operación y se insertará en el índice ideal de la lista el nuevo bloque de memoria, de esta manera no se verá afectada la memoria total del ajuste.

- Atributos
 - Boolean bandera.
 - Espacio NewEspacio
- Métodos
 - IngresaProceso()

```

MejorAjuste.java

package proyectooperativos;

import java.awt.Graphics;

public class MejorAjuste {
    int indiceIdeal=-1;
    public int ingresaProceso(Espacio newEspacio, Graphics panel) {
        int rf = 0;
        int residuo = Integer.MAX_VALUE;
        indiceIdeal=-1;

        if(newEspacio.tamanyo>Sinc.limiteSuperior){
            return 2;
        }

        for (int i = 0; i < Sinc.LMejor.size(); i++) {
            //Actualiza la lista para mostrar la posicion del scanner
            Sinc.dibujaLista(Sinc.LMejor, panel, i);
            Sinc.posScannerMejor = i; //Actualiza la ultima posicion del scanner
            Sinc.esperar(0.18);

            // Recorre la lista de asignaciones
            if (Sinc.LMejor.get(i).tipo == 1) {
                if (Sinc.LMejor.get(i).tamanyo >= newEspacio.tamanyo) {
                    // Valida que el espacio sea igual o mayor al tamaño solicitado
                    if (Sinc.LMejor.get(i).tamanyo == newEspacio.tamanyo) {
                        // El nuevo espacio cabe exactamente en el espacio existente
                        newEspacio.nBloqueInicial = Sinc.LPrimer.get(i).nBloqueInicial;
                        Sinc.LMejor.remove(i);
                        Sinc.LMejor.add(i,newEspacio);
                        rf = 1; // Devuelve true para indicar que se asignó correctamente
                        indiceIdeal = i;
                        Sinc.posScannerMejor = indiceIdeal; //Actualiza la ultima posicion del scanner
                        return rf;
                    } else {
                        if(Sinc.LMejor.get(i).tamanyo-newEspacio.tamanyo < residuo){
                            indiceIdeal = i;
                            residuo = Sinc.LMejor.get(i).tamanyo-newEspacio.tamanyo;
                        }
                    }
                }
            }
        }

        //En caso de que no entre
        if(indiceIdeal == -1){
            Sinc.posScannerMejor = -1; //Actualiza la ultima posicion del scanner
            return 0;
        }

        // Actualiza el espacio asignado
        Sinc.LMejor.get(indiceIdeal).tamanyo -= newEspacio.tamanyo;

        // Agrega un nuevo espacio ocupado con el tamaño del nuevo
        newEspacio.nBloqueInicial = Sinc.LMejor.get(indiceIdeal).nBloqueInicial;
        Sinc.LMejor.add(indiceIdeal, newEspacio);
        rf = 1; // Devuelve true para indicar que se asignó correctamente
        Sinc.posScannerMejor = indiceIdeal; //Actualiza la ultima posicion del scanner
        return rf;
    }
}

```

PRÓXIMO AJUSTE

SISTEMAS OPERATIVOS

Clase que realiza el proceso lógico necesario para llevar a cabo el próximo ajuste de memoria al ingresar un nuevo proceso utilizando la clase Sinc.

El algoritmo del próximo ajuste es muy parecido al algoritmo de primer ajuste, aunque en este la diferencia es que el índice en el cual se realizó la última inserción a la memoria se almacena en una variable estática (aprovechando las características del lenguaje Java), al inicio la variable estática “posición” tiene un valor de cero el cual está aumentando a medida que se recorre la lista de memoria correspondiente al próximo ajuste desde el archivo sincronizado.

Si hay un campo con las características, se evalúa si es del mismo tamaño o diferente, el espacio de memoria que quiere ingresar ocupará el tamaño del resultado de la diferencia entre el espacio de memoria existente y el espacio que se encontraba en memoria ocupará el tamaño del resultado de la diferencia entre el espacio donde se ingresará y el espacio del proceso a ingresar.

En caso de que el algoritmo de proximoAjuste no encuentre ningún campo que cumpla con el espacio mínimo requerido el nuevo bloque de memoria no podrá ser ingresado y si el puntero llega al final de la lista que representa la memoria significa que no logró entrar por lo que el mismo puntero se reiniciara para comenzar nuevamente.

- Atributos
 - Boolean bandera.
 - Espacio NewEspacio
- Métodos
 - IngresaProceso()

```

● ● ● ProximoAjuste.java

package proyectooperativos;

import java.awt.Graphics;

public class ProximoAjuste {
    static int posicion = 0; //variable estatica

    public int ingresaProceso(Espacio newEspacio, Graphics panel){
        //Aqui se debe agregar el nuevo espacio en la lista de "Sinc" correspondiente
        //Retorna true en caso de que si lo pueda insertar, de lo contrario false
        int bandera=0;

        if(newEspacio.tamano>Sinc.limiteSuperior){
            return 2;
        }

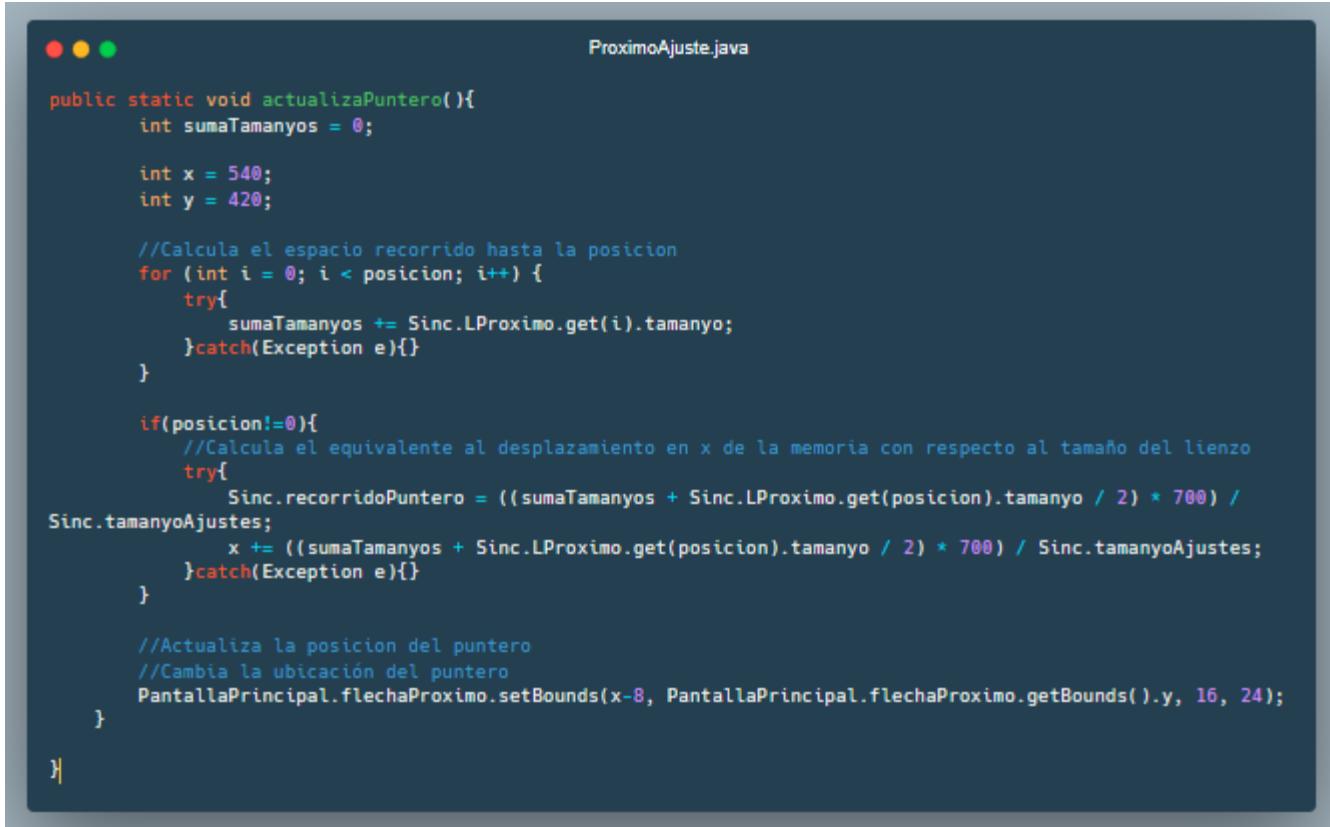
        for (; posicion < Sinc.LProximo.size(); posicion++) { //recorre la lista
            //Actualiza la lista para mostrar la posicion del scanner
            Sinc.dibujaLista(Sinc.LProximo, panel, posicion);
            Sinc.posScannerProximo = posicion; //Actualiza la ultima posicion del scanner
            Sinc.esperar(0.18);

            if(Sinc.LProximo.get(posicion).tipo == 1 ){// si hay un espacio vacio
                if(Sinc.LProximo.get(posicion).tamano>=newEspacio.tamano){//checa si el newEspacio cabe en
                ese i espacio
                    //restamos el espacio en donde si entra, pero si es igual no lo hago? B: Yo digo que no
                    solo cambiale los parametros
                    if(Sinc.LProximo.get(posicion).tamano!=newEspacio.tamano){//ES DIFERENTE EL ESPACIO
                        Sinc.LProximo.get(posicion).tamano=Sinc.LProximo.get(posicion).tamano-
                        newEspacio.tamano;
                        //agrega espacio antes uno nuevo con el tamano del nuevo
                        newEspacio.tipo=2;
                        newEspacio.nBloqueInicial = Sinc.LProximo.get(posicion).nBloqueInicial;
                        Sinc.LProximo.add(posicion,newEspacio);
                    }else{// ES EL MISMO ESPACIO
                        //le ponemos todos los datos del nuevo a ese espacio vacio
                        Sinc.LProximo.get(posicion).nombre = newEspacio.nombre;
                        Sinc.LProximo.get(posicion).tipo = 2;//ahora es ocupado pero que se puede liberar
                        Sinc.LProximo.get(posicion).color[0] = newEspacio.color[0];
                        Sinc.LProximo.get(posicion).color[1] = newEspacio.color[1];
                        Sinc.LProximo.get(posicion).color[2] = newEspacio.color[2];
                    }
                    bandera= 1; //Revisar
                    break;
                }else{//es el ultimo y no cupo en algun espacio
                    bandera= 0;
                }
            }
        }

        //En caso de que se acabe el espacio el puntero se reinicia
        if(! (posicion < Sinc.LProximo.size())){
            posicion = 0;
            Sinc.posScannerProximo = -1; //Actualiza la ultima posicion para que nos e muestre en caso de no
            entrar
        }

        actualizaPuntero();
        return bandera;
    }
}

```



```

public static void actualizaPuntero(){
    int sumaTamanyos = 0;

    int x = 540;
    int y = 420;

    //Calcula el espacio recorrido hasta la posicion
    for (int i = 0; i < posicion; i++) {
        try{
            sumaTamanyos += Sinc.LProximo.get(i).tamanyos;
        }catch(Exception e){}
    }

    if(posicion!=0){
        //Calcula el equivalente al desplazamiento en x de la memoria con respecto al tamaño del lienzo
        try{
            Sinc.recorridoPuntero = ((sumaTamanyos + Sinc.LProximo.get(posicion).tamanyos / 2) * 700) /
Sinc.tamanoAjustes;
            x += ((sumaTamanyos + Sinc.LProximo.get(posicion).tamanyos / 2) * 700) / Sinc.tamanoAjustes;
        }catch(Exception e){}
    }

    //Actualiza la posicion del puntero
    //Cambia la ubicación del puntero
    PantallaPrincipal.flechaProximo.setBounds(x-8, PantallaPrincipal.flechaProximo.getBounds().y, 16, 24);
}
}

```

LIBERACIÓN DE MEMORIA EN AJUSTES

Para liberar un bloque de memoria de un proceso en específico se busca el bloque de espacio que esté en memoria y cuyo nombre coincida con el que se va a liberar. En caso de encontrarse el espacio, este cambiará su tipo a uno o vacío y se ejecutará la fusión de memoria.

La función de la fusión de memoria es combinar los espacios vacíos adyacentes y que sean del mismo bloque inicial. Por medio del algoritmo recursivo se verifica si el espacio es adyacente a otro espacio vacío y se suma el tamaño entre los dos para asignarse al primer bloque de memoria y el segundo se elimina conservando así el total de memoria inicial.

La recursividad forma papel importante dentro de este algoritmo, cuando se fusionan dos bloques de espacio se hace una llamada recursiva para comprobar que no existan más campos que se puedan fusionar, este proceso se repetirá cada que la fusión sea exitosa y se detendrá hasta que llegue al final de la lista de memoria sin fusionar más espacios

- CompresionDeMemoriaAjustes(). Realiza la compresión de memoria en un arreglo específico (LAjuste) para ajustes particulares. Recorre el arreglo y, al encontrar dos espacios libres consecutivos, aumenta el tamaño del primer espacio y elimina el segundo, llamando recursivamente para asegurar la correcta liberación.



```

PantallaPrincipal.java

public void compresionDeMemoriaAjustes(ArrayList <Espacio> LAjuste){
    for (int i = 0; i < LAjuste.size()-1; i++) {
        //Donde encuentre 2 continuos libres
        if(LAjuste.get(i).tipo == 1 && LAjuste.get(i+1).tipo == 1){
            //Aumenta el primer espacio que coincide
            LAjuste.get(i).tamanyo += LAjuste.get(i+1).tamanyo;
            LAjuste.remove(i+1); //Elimina el espacio siguiente
            compresionDeMemoriaAjustes(LAjuste);
        }
    }
}

```

FRAGMENTACIÓN DE MEMORIA EN AJUSTES

Para calcular la fragmentación de memoria correspondiente a los ajustes se recurre a las variables auxiliares que definen a cuál bloque inicial de memoria es al que pertenecen cada espacio existente antes de realizar el cálculo.

Si el bloque al que se quiere calcular la fragmentación es idéntico como el bloque inicialmente, ningún proceso entró y no fragmentó la memoria, por lo que la fragmentación correspondiente a ese bloque es de cero.

la fragmentación de los bloques de memoria se mide en torno a cuánto fue el espacio libre restante después de ingresar un proceso que no ocupara el total del bloque inicial.

En caso de que el bloque inicial memoria esté dividido en más de un espacio se calculará en base a la variable que define el número de bloque inicial cuánto es el espacio vacío y se realizará una regla de tres para obtener el equivalente en porcentaje de la fragmentación sobre la cantidad total de memoria

- **FragmentacionAjuste()**. Calcula y acumula la fragmentación total de bloques de memoria para un tipo de ajuste específico, actualizando datos para gráficas y mostrando resultados en la consola.

```

public void fragmentacionAjuste(ArrayList <Espacio> LAjuste, int numAjuste){
    float fragmentacion = 0;
    float suma = 0;
    String tipo = "";
    //Segun el ajuste
    switch(numAjuste){
        case 0: //Primer
            Sinc.fragTotalPrimerA = 0;
            tipo = "Primer ajuste";
            break;
        case 1: //Mejor
            Sinc.fragTotalMejorA = 0;
            tipo = "Mejor ajuste";
            break;
        case 2: //Proximo
            Sinc.fragTotalProximoA = 0;
            tipo = "Proximo ajuste";
            break;
    }
    for (int i = 0; i < Sinc.LInicial.size(); i++) {
        suma = 0;
        System.out.print("Bloque "+Sinc.LInicial.get(i).nBloqueInicial);
        //Calculamos la fragmentacion por bloques
        for (int j = 0; j < LAjuste.size(); j++) {
            //Si esta libre y es del mismo bloque
            if (LAjuste.get(j).tipo == 1 && LAjuste.get(j).nBloqueInicial == Sinc.LInicial.get(i).nBloqueInicial ) {
                //Suma contiene la suma del espacio libre de un mismo bloque
                suma += LAjuste.get(j).tamanylo; //Acumulamos los tamaños del mismo bloque
            }
        }
        //En caso de que el espacio sea tipo 0 o la fragmentacion sea del 100% reportamos 0 de fragmentacion
        if(Sinc.LInicial.get(i).tipo == 0 || (100*suma)/Sinc.LInicial.get(i).tamanylo == 100){
            fragmentacion = 0;
        }else{
            //Realizamos regla de 3 para conocer el % de fragmentación
            fragmentacion = (100*suma)/Sinc.tamanyoAjustes;
        }
        //Agregamos los datos a la lista de datos de fragmentacion para las graficas
        Sinc.datos.setValue(fragmentacion,tipo,"B"+(i+1));
        System.out.println(", %Frag. = "+ fragmentacion);
        //Acumula la fragmentacion total dependiendo del ajuste
        switch(numAjuste){
            case 0: //Primer
                Sinc.fragTotalPrimerA += fragmentacion;
                break;
            case 1: //Mejor
                Sinc.fragTotalMejorA += fragmentacion;
                break;
            case 2: //Proximo
                Sinc.fragTotalProximoA += fragmentacion;
                break;
        }
    }
}
}

```

SISTEMA BUDDY

Agrega el nuevo espacio en la lista de "Sinc" correspondiente, con la lógica del sistema Buddy.

Para mantener el código lo más optimizado a nivel rendimiento posible el sistema buddy se compone de una lista al igual que los ajustes, se recorre la lista en busca de un espacio libre que al inicio siempre serán los megabytes vacíos y completos, en caso de encontrarse con un bloque de memoria que este vacío comparara que el espacio cumpla con la cualidad del sistema buddy al asignar espacios de memoria:

```
/* EJEMPLO
```

```
256 < A=64 <= 512
```

```
128 < A=64 <= 256
```

```
64 < A=64 <= 128
```

```
32 < A=64 <= 64
```

```
*/
```

Si la condición es verdadera el sistema buddy asignara al espacio actual el nuevo proceso cambiando su tipo a ocupado y calculando la diferencia entre el tamaño del bloque del sistema buddy y el bloque a ingresar se asigna el desperdicio que es la fragmentación de ese bloque de memoria.

En caso de que la condición no se cumpla se generara un nuevo espacio el cual será de la mitad del tamaño del bloque padre, este bloque cambiara de tamaño a la mitad del que tenía anteriormente y se agregara el que fue generado, de esta manera un bloque de memoria será dividido en dos partes respetando así las propiedades del sistema buddy.

Para la compresión de memoria en este sistema es importante conocer cuál es el origen del megabyte de memoria por lo cual cada espacio cuenta con una lista de “hermanos/familiaresDirectos” para identificar cual es el bloque con el que se debe fusionar ya que deben ser del mismo padre y pertenecer al mismo megabyte inicial el cual es el nBloqueInicial.

- Atributos
 - Espacio NewEspacio
- Métodos
 - IngresapProceso()

```

● ● ●
SistemaBuddy.java

package proyectooperativos;

import java.awt.Graphics;

public class SistemaBuddy {
    public int ingresaProceso(Espacio nEspacio, Graphics panelBuddy){
        //Aqui se debe agregar el nuevo espacio en la lista de "Sinc" correspondiente
        //Retorna true en caso de que si lo pueda insertar, de lo contrario false
        if(nEspacio.tamanyo>1024){
            return 2;
        }
        for (int i = 0; i < Sinc.LBuddy.size(); i++) {
            if(Sinc.LBuddy.get(i).tipo == 1 ){
                //o ocupado, 1 libre, 2 ocupado pero se puede liberar
                if(Sinc.LBuddy.get(i).tamanyo >= nEspacio.tamanyo ){
                    //Actualiza lentamente la lista mientras se divide la memoria
                    Sinc.dibujaLista(Sinc.LBuddy, panelBuddy,Sinc.posScannerPrimer);
                    Sinc.esperar(0.30);
                    if(Sinc.LBuddy.get(i).tamanyo/2 < nEspacio.tamanyo && nEspacio.tamanyo<= Sinc.LBuddy.get(i).tamanyo ){
                        //Si entra se calcula el desperdicio de ese bloque de memoria
                        Sinc.LBuddy.get(i).desperdicio = Sinc.LBuddy.get(i).tamanyo - nEspacio.tamanyo;
                        //Se copian los valores del nuevo espacio
                        Sinc.LBuddy.get(i).nombre = nEspacio.nombre;
                        Sinc.LBuddy.get(i).tipo = 2;
                        Sinc.LBuddy.get(i).color[0] = nEspacio.color[0];
                        Sinc.LBuddy.get(i).color[1] = nEspacio.color[1];
                        Sinc.LBuddy.get(i).color[2] = nEspacio.color[2];
                        return 1; //Si entro
                    }else{
                        //Se divide el tamaño de memoria de ese bloque
                        Sinc.LBuddy.get(i).tamanyo /= 2;
                        //Genera un nuevo espacio con las características del espacio que se dividió, de esta forma son "hermanos"
                        Espacio e = new Espacio(
                            Sinc.LBuddy.get(i).tamanyo,
                            1,
                            Sinc.LBuddy.get(i).nBloqueInicial
                        );
                        //Añadimos el hermano a la lista de hermanos del espacio de memoria
                        Sinc.LBuddy.get(i).LHermanos.add(0,Sinc.cuentaHermanos);
                        e.LHermanos.add(0,Sinc.cuentaHermanos);
                        //Se aumenta el contador de hermanos global
                        Sinc.cuentaHermanos++;
                        //Se añade el espacio
                        Sinc.LBuddy.add(i,e);
                        //Utilizamos recursividad hasta que entre el proceso
                        return ingresaProceso(nEspacio,panelBuddy);
                    }
                }
            }
        }
        return 0;
    }
}

```

LIBERACIÓN DE MEMORIA EN EL SISTEMA BUDDY

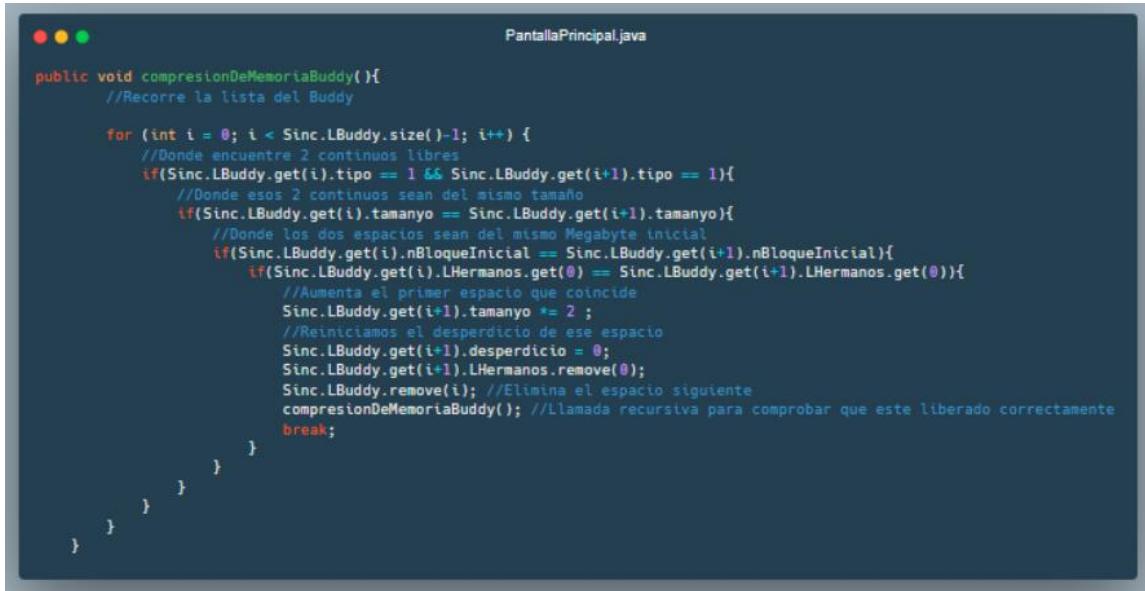
La liberación de los espacios de memoria en el sistema buddy se comporta de una manera parecida a los ajustes, pero los bloques que se tienen que fusionar tienen que cumplir ciertos requisitos.

A medida que se van insertando procesos en memoria y se va dividiendo la misma, los bloques de memoria nuevos añaden en su lista de hermanos el identificador del nodo del cual proceden y bloque hermano copia la lista de hermanos también.

Para una fusión de bloques de memoria exitosa, el algoritmo aplica recursividad para garantizar la fusión correcta de toda la memoria hasta donde sea posible.

para que dos bloques de memoria en el sistema buddy se han fusionados es necesario que el último identificador en su lista de hermanos sea el mismo que el bloque vacío con el que se quiere funcionar y también del megabyte bloque inicial

- **comprasionDeMemoriaBuddy()**. Realiza la comprensión de memoria buddy. Recorre el arreglo, si cierta posición y la siguiente se encuentran libres, checa que los dos espacios sean del mismo tamaño para después checar que los bloques iniciales de esos dos espacios sean iguales y al cumplirse, checa que los hermanos de esas posiciones sean iguales. El bloque se duplicará y se eliminará el espacio siguiente para que se forme un bloque más grande. Se llama recursivamente para comprobar que se esté liberando correctamente



```

public void comprasionDeMemoriaBuddy(){
    //Recorre la lista del Buddy

    for (int i = 0; i < Sinc.LBuddy.size()-1; i++) {
        //Donde encuentre 2 continuos libres
        if(Sinc.LBuddy.get(i).tipo == 1 && Sinc.LBuddy.get(i+1).tipo == 1){
            //Donde esos 2 continuos sean del mismo tamaño
            if(Sinc.LBuddy.get(i).tamanyo == Sinc.LBuddy.get(i+1).tamanyo){
                //Donde los dos espacios sean del mismo Megabyte inicial
                if(Sinc.LBuddy.get(i).nBloqueInicial == Sinc.LBuddy.get(i+1).nBloqueInicial){
                    //Si los hermanos son iguales
                    if(Sinc.LBuddy.get(i).LHermanos.get(0) == Sinc.LBuddy.get(i+1).LHermanos.get(0)){
                        //Aumenta el primer espacio que coincide
                        Sinc.LBuddy.get(i+1).tamanyo *= 2;
                        //Reiniciamos el desperdicio de ese espacio
                        Sinc.LBuddy.get(i+1).desperdicio = 0;
                        Sinc.LBuddy.get(i+1).LHermanos.remove(0);
                        Sinc.LBuddy.remove(i); //Elimina el espacio siguiente
                        comprasionDeMemoriaBuddy(); //Llamada recursiva para comprobar que este liberado correctamente
                        break;
                    }
                }
            }
        }
    }
}

```

FRAGMENTACIÓN DEL SISTEMA BUDDY

La fragmentación del sistema buddy se calcula por cada megabyte de memoria inicial con respecto al total de memoria que son 1024 kb por el número de megabytes iniciales.

Cada megabyte inicia siendo su propio bloque inicial y al dividirse este para cumplir con los requisitos para ingresar un nuevo espacio de memoria los procesos hijos de ese bloque de memoria tendrán el mismo valor del bloque inicial.

Después de haber ingresado varios procesos existe la posibilidad de que estos generen un desperdicio de memoria del bloque el cual indica la fragmentación de ese espacio, la suma de los desperdicios o la fragmentación del megabyte con respecto a la memoria total del sistema buddy es el porcentaje de fragmentación total del bloque con respecto al total de memoria y la suma de cada una de estas fragmentaciones es la fragmentación total del sistema

```

public void fragmentacionBuddy(){
    int suma = 0;
    float fragmentacion = 0;
    //Se imprimen los datos de cada megabyte
    for (int i = 0; i < Sinc.LBuddy.size(); i++) {
        System.out.print(i);
        System.out.print(", nMb: "+(Sinc.tamanyoBuddy - Sinc.LBuddy.get(i).nBloqueInicial - 1));
        System.out.println(", desperdicio: "+Sinc.LBuddy.get(i).desperdicio);
    }
    System.out.println("\n");
    //Recorre la lista del sistema buddy y calcula su fragmentacion
    Sinc.fragTotalSistemaBuddy = 0;
    for (int i = 0; i < Sinc.tamanyoBuddy; i++) {
        suma = 0;
        System.out.print("Megabyte N° "+(i));
        for (int j = 0; j < Sinc.LBuddy.size(); j++) {
            //Obtiene el desperdicio por mega
            if(Sinc.LBuddy.get(j).nBloqueInicial==(Sinc.tamanyoBuddy-i-1)){
                suma += Sinc.LBuddy.get(j).desperdicio;
            }
        }
        System.out.print(", Desperdicio: "+suma);
        //Calcula la fragmentacion del mega
        fragmentacion = (float)((float)suma/(float)(1024*Sinc.tamanyoBuddy))*(float)100;
        Sinc.fragTotalSistemaBuddy += fragmentacion;
        System.out.println(", %Frag: "+fragmentacion);
        //Añade los datos a la data de la grafica del sistema buddy
        Sinc.datosFragmentacionBuddy.setValue(fragmentacion,"Fragmentación","Mb "+(i+1));
    }
}

```

CÓDIGOS AUXILIARES

- **ImportGeneral()**. Importación de librerías para uso general

```

import.java

package proyectooperativos;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.util.ArrayList;
import org.jfree.data.category.DefaultCategoryDataset;
import org.jfree.data.general.DefaultPieDataset;

```

- **ProyectoOperativos**. Clase principal que alberga el método main() y realiza la invocación de la clase Sinc. Muestra una portada con los datos de todos los integrantes del equipo, nombre de la materia, título del proyecto.



```

● ● ● ProyectoOperativos.java

package proyectooperativos;

public class ProyectoOperativos {

    public static void main(String[] args) {
        Sinc.portada = new Portada();
        Sinc.portada.setVisible(true);
        Sinc.portada.setSize(1150, 650);
        Sinc.portada.setResizable(false);
        Sinc.portada.setLocationRelativeTo(null); //en medio
    }
}

```

- **IniciarActionPerformed()**. Realiza la invocación de la clase Sinc. Se establecen atributos del panel, como el tamaño y la visibilidad.



```

● ● ● Portada.java

private void IniciarActionPerformed(java.awt.event.ActionEvent evt) {

    Sinc.pantallaPrincipal = new PantallaPrincipal();
    Sinc.pantallaPrincipal.setVisible(true);
    Sinc.pantallaPrincipal.setSize(1300, 650);
    Sinc.pantallaPrincipal.setResizable(false);
    Sinc.pantallaPrincipal.setLocationRelativeTo(null); //en medio
    this.dispose();
}

```

- **PantallaPrincipal**. Módulo de Java encargado de utilizar la paleta para la pantalla principal del proyecto. En esta pantalla, se presentan de manera visual los bloques de asignación de cada ajuste, tales como el estado inicial de la memoria, el primer ajuste, el mejor ajuste, el próximo ajuste y el sistema Buddy. En la parte izquierda de la interfaz se despliega el historial de procesos, el tamaño de la memoria, así como el módulo para el ingreso de nuevos procesos.

▪ **Librerías:**

- java.awt.Color;
- java.awt.Graphics;
- java.util.ArrayList;
- java.util.Iterator;
- java.util.Random;
- javax.swing.JOptionPane;
- javax.swing.event.ChangeEvent;
- javax.swing.table.DefaultTableModel;
- prueba.Validaciones;

▪ **Atributos:**

- ChangeEvent auxEvent;
- Eneros: R, G, B

▪ **Métodos:**

- **ActualizaLienzos()**. Como el nombre indica, actualiza le dibujo de los procesos en memoria de cada ajuste en el panel gráfico.

```

public void actualizaLienzos(){
    //Funcion que llama para actualizar el dibujo de la memoria
    Sinc.dibujaLista(Sinc.LInicial, panelInicial.getGraphics(),-1);
    Sinc.dibujaLista(Sinc.LPrimer, panelPrimer.getGraphics(),Sinc.posScannerPrimer);
    Sinc.dibujaLista(Sinc.LMejor, panelMejor.getGraphics(),Sinc.posScannerMejor);
    Sinc.dibujaLista(Sinc.LProximo, panelProximo.getGraphics(),Sinc.posScannerProximo);
    Sinc.dibujaLista(Sinc.LBuddy, panelBuddy.getGraphics(),-1);
}

```

- **BtnGenerarActionPerformed()**. Genera el espacio de cada proceso. Si el usuario ingresa un numero negativo, flotante o igual a cero, envía una alerta de error hasta que el numero ingresado sea un entero positivo mayor a 0. Manda llamar ActualizaTabla y ActualizaLienzo para agregar a la memoria visual del proceso, sea de cual tamaño.

```

private void btnGenerarActionPerformed(java.awt.event.ActionEvent evt) {
    Registro r = new Registro();
    PrimerAjuste priA = new PrimerAjuste();
    MejorAjuste meA = new MejorAjuste();
    ProximoAjuste proA = new ProximoAjuste();
    SistemaBuddy sisB = new SistemaBuddy();
    String nombre = generaNombre();
    int tamanyo;
    try {
        tamanyo = Integer.parseInt(tfTamanyo.getText());
        if (tamanyo < 0) {
            JOptionPane.showMessageDialog(null, "Error: Por favor, ingresa un número entero positivo mayor que 0");
        } else {
            tfTamanyo.setText("");
            r.tipoES = "Entrada";
            r.nombre = nombre;
            r.tamanyo = tamanyo;
            //ingresaProceso retorna true o falso en caso de no ingresar
            r.resultadoPimerA = priA.ingresaProceso(new Espacio(2,nombre,tamanyo,R,G,B),panelPrimer.getGraphics());
            r.resultadoMejorA = meA.ingresaProceso(new Espacio(2,nombre,tamanyo,R,G,B),panelMejor.getGraphics());
            r.resultadoProximoA = proA.ingresaProceso(new Espacio(2,nombre,tamanyo,R,G,B),panelProximo.getGraphics());
            r.resultadoBuddy = sisB.ingresaProceso(new Espacio(2,nombre,tamanyo,R,G,B),panelBuddy.getGraphics());

            Sinc.LTabla.add(r); //Añadimos al inicio
            actualizaTabla();
            actualizaLienzos();
        }
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(null, "Error: Por favor, ingresa un número entero válido");
    }
}

```

- **DeslizanteColorStateChanged()**. Barra deslizante implementada para la modulación del brillo asociado al color de entrada durante la selección de un nuevo proceso en el cuadro de colores, donde los valores de color se representan en el modelo RGB. Además, la barra deslizante puede aplicar el color resultante al segmento o bloque de proceso recién seleccionado.



```

PantallaPrincipal.java

public void stateChanged(ChangeEvent e) {
    //En caso de cambiar el deslizante de el brillo de color
    //asignamos G de acuerdo a su valor y lo enviamos a "colorea"
    G = deslizanteColor.getValue();
    this.colorea(G);
}

private void colorea(int G) {
    //Coloreamos el cuadro de colores
    Graphics papelC = panelColor.getGraphics();
    for (int i = 0; i < 51; i++) {
        for (int j = 0; j < 51; j++) {
            papelC.setColor(new Color(i*5,G,j*5));
            papelC.fillRect(2+i*2, 2+j*2, 2, 2);
        }
    }
}

```

- PanelColorMousePressed().
 - Atributos: Graphics->papelC, R, B.
 - Recibe un evento, cambia el estado del evento. Se dibuja el cuadro del lugar asignado de la memoria en la tabla.



```

PantallaPrincipal.java

private void deslizanteColorStateChanged(javax.swing.event.ChangeEvent evt){
    this.stateChanged(evt);
}

private void panelColorMousePressed(java.awt.event.MouseEvent evt) {

    Graphics papelC = panelColor.getGraphics();
    R = evt.getX()/2*5;
    B = evt.getY()/2*5;
    //Dibujamos un cuadrito en el lugar que selecciono para indicar el color
    this.colorea(G);
    papelC.setColor(Color.white);
    papelC.drawRect(evt.getX()-2, evt.getY()-2, 4, 4);
}

```

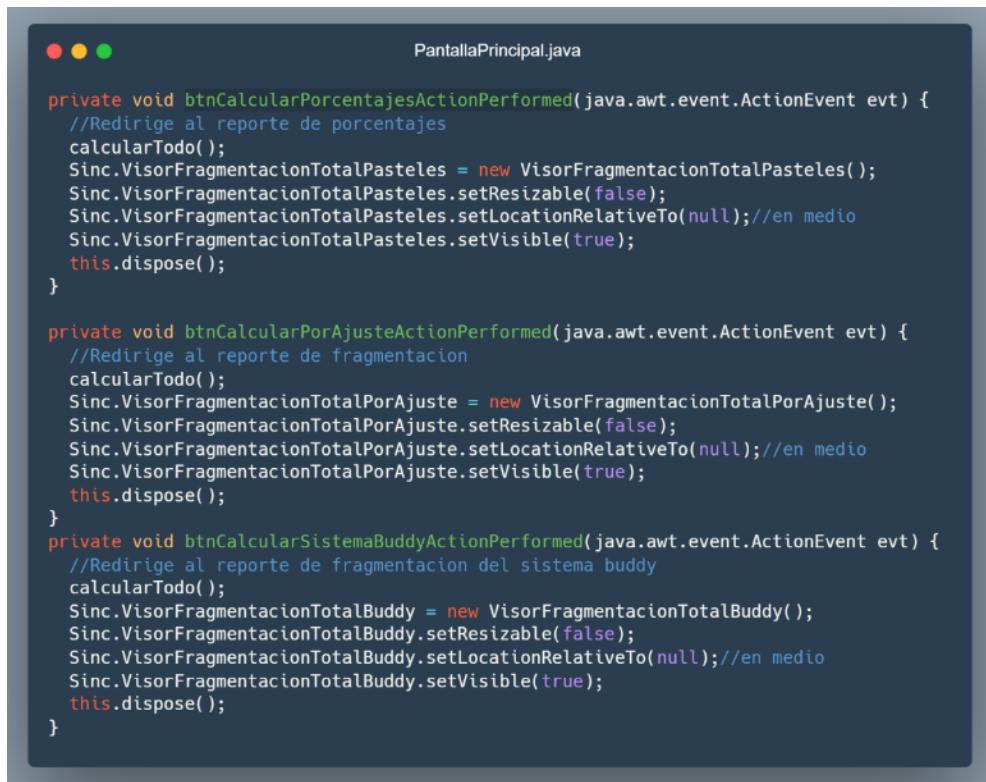
- **BtnLiberarActionPerformed()**. Recibe un evento, que es la selección del evento en tabla, en caso de no haber sido seleccionado, manda un error. De otro modo, escanea el espacio en cada ajuste, saca el proceso a liberar y lo envía.



```
PantallaPrincipal.java

private void btnLiberarActionPerformed(java.awt.event.ActionEvent evt) {
    if(tablaHistorial.getSelectedRow() == -1){ //En caso de no seleccionar nada marca error
        JOptionPane.showInternalMessageDialog(null, "Selecciona un proceso valido","Error",JOptionPane.ERROR_MESSAGE);
    }else{
        Sinc.posScannerPrimer = -1;
        Sinc.posScannerMejor = -1;
        Sinc.posScannerProximo = -1;
        //Obtiene el proceso a liberar y lo envia
        String proceso = (String)tablaHistorial.getValueAt(tablaHistorial.getSelectedRow(), 1);
        liberarProcesoSeleccionado(proceso);
    }
}
```

- **BtnCalcularFragmentacionActionPerformed()**. btnLiberarActionPerformed libera el proceso seleccionado si hay alguno, mostrando un error si no hay selección. btnCalcularFragmentacionActionPerformed redirige a la visualización de datos de fragmentación total, realiza cálculos y cierra la ventana actual. Ambos procesos están relacionados con la gestión de procesos y visualización de fragmentación.



```
PantallaPrincipal.java

private void btnCalcularPorcentajesActionPerformed(java.awt.event.ActionEvent evt) {
    //Redirige al reporte de porcentajes
    calcularTodo();
    Sinc.VisorFragmentacionTotalPasteles = new VisorFragmentacionTotalPasteles();
    Sinc.VisorFragmentacionTotalPasteles.setResizable(false);
    Sinc.VisorFragmentacionTotalPasteles.setLocationRelativeTo(null); //en medio
    Sinc.VisorFragmentacionTotalPasteles.setVisible(true);
    this.dispose();
}

private void btnCalcularPorAjusteActionPerformed(java.awt.event.ActionEvent evt) {
    //Redirige al reporte de fragmentacion
    calcularTodo();
    Sinc.VisorFragmentacionTotalPorAjuste = new VisorFragmentacionTotalPorAjuste();
    Sinc.VisorFragmentacionTotalPorAjuste.setResizable(false);
    Sinc.VisorFragmentacionTotalPorAjuste.setLocationRelativeTo(null); //en medio
    Sinc.VisorFragmentacionTotalPorAjuste.setVisible(true);
    this.dispose();
}

private void btnCalcularSistemaBuddyActionPerformed(java.awt.event.ActionEvent evt) {
    //Redirige al reporte de fragmentacion del sistema buddy
    calcularTodo();
    Sinc.VisorFragmentacionTotalBuddy = new VisorFragmentacionTotalBuddy();
    Sinc.VisorFragmentacionTotalBuddy.setResizable(false);
    Sinc.VisorFragmentacionTotalBuddy.setLocationRelativeTo(null); //en medio
    Sinc.VisorFragmentacionTotalBuddy.setVisible(true);
    this.dispose();
}
```

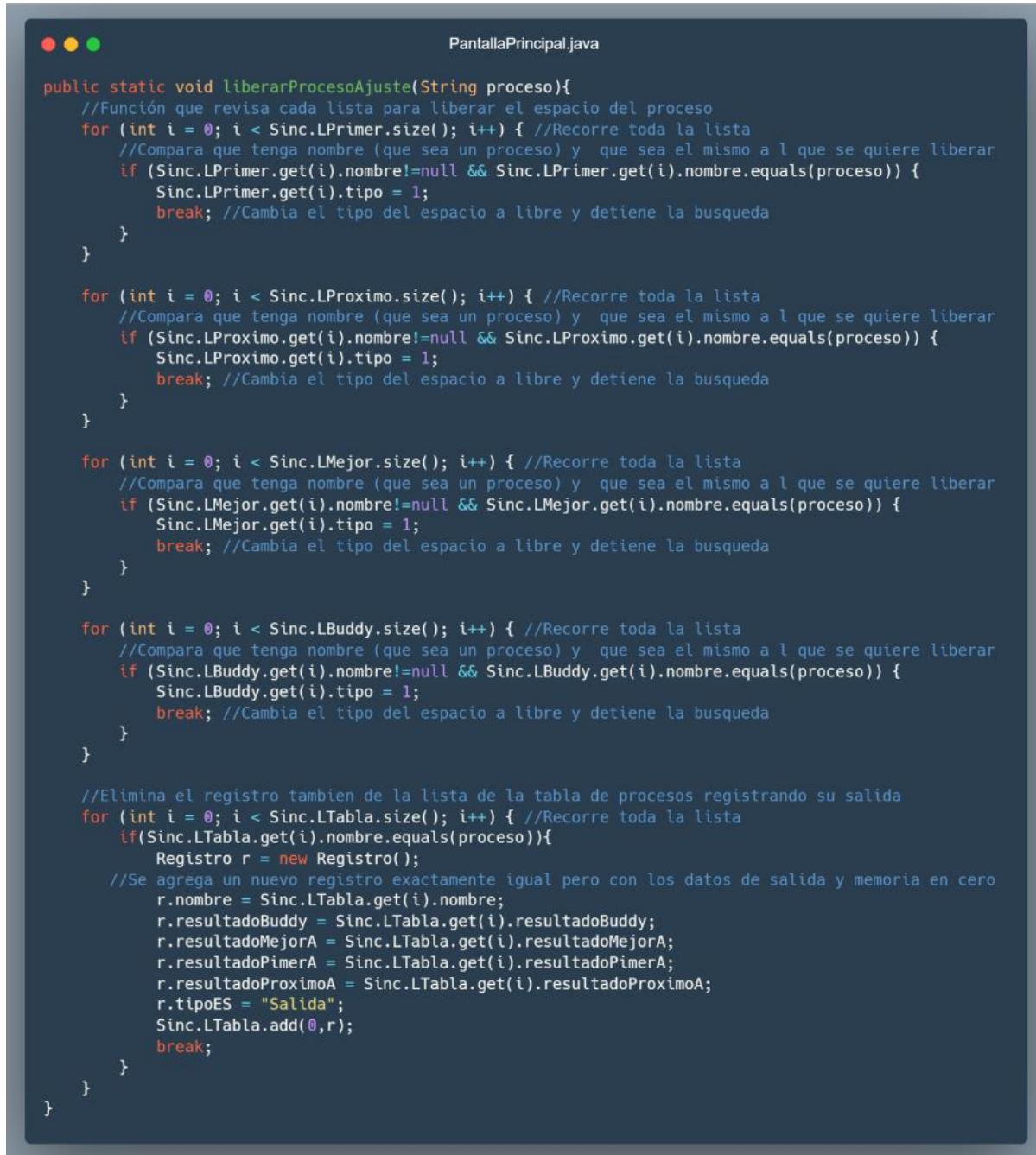
- **LiberarProcesoSeleccionado()**. Verifica si un proceso no ha sido liberado previamente antes de liberarlo, realizando operaciones de compresión de memoria para ajustes específicos y actualizando los lienzo y tabla correspondientes.



```
PantallaPrincipal.java

public void liberarProcesoSeleccionado(String proceso){
    //Comprobamos que el proceso no haya sido ya anteriormente liberado
    if(comprobarProceso(proceso)){
        liberarProcesoAjuste(proceso);
        compresionDeMemoriaAjustes(Sinc.LPrimer);
        compresionDeMemoriaAjustes(Sinc.LMejor);
        compresionDeMemoriaAjustes(Sinc.LProximo);
        compresionDeMemoriaBuddy();
        actualizaLienzos();
        actualizaTabla();
    }
}
```

- `LiberarProcesoAjuste()`. Libera un proceso en listas de memoria específicas, cambiando el tipo del espacio correspondiente a libre y registrando la salida del proceso en la lista **Sinc.LTabla**.



```

public static void liberarProcesoAjuste(String proceso){
    //Función que revisa cada lista para liberar el espacio del proceso
    for (int i = 0; i < Sinc.LPrimer.size(); i++) { //Recorre toda la lista
        //Compara que tenga nombre (que sea un proceso) y que sea el mismo a l que se quiere liberar
        if (Sinc.LPrimer.get(i).nombre!=null && Sinc.LPrimer.get(i).nombre.equals(proceso)) {
            Sinc.LPrimer.get(i).tipo = 1;
            break; //Cambia el tipo del espacio a libre y detiene la búsqueda
        }
    }

    for (int i = 0; i < Sinc.LProximo.size(); i++) { //Recorre toda la lista
        //Compara que tenga nombre (que sea un proceso) y que sea el mismo a l que se quiere liberar
        if (Sinc.LProximo.get(i).nombre!=null && Sinc.LProximo.get(i).nombre.equals(proceso)) {
            Sinc.LProximo.get(i).tipo = 1;
            break; //Cambia el tipo del espacio a libre y detiene la búsqueda
        }
    }

    for (int i = 0; i < Sinc.LMejor.size(); i++) { //Recorre toda la lista
        //Compara que tenga nombre (que sea un proceso) y que sea el mismo a l que se quiere liberar
        if (Sinc.LMejor.get(i).nombre!=null && Sinc.LMejor.get(i).nombre.equals(proceso)) {
            Sinc.LMejor.get(i).tipo = 1;
            break; //Cambia el tipo del espacio a libre y detiene la búsqueda
        }
    }

    for (int i = 0; i < Sinc.LBuddy.size(); i++) { //Recorre toda la lista
        //Compara que tenga nombre (que sea un proceso) y que sea el mismo a l que se quiere liberar
        if (Sinc.LBuddy.get(i).nombre!=null && Sinc.LBuddy.get(i).nombre.equals(proceso)) {
            Sinc.LBuddy.get(i).tipo = 1;
            break; //Cambia el tipo del espacio a libre y detiene la búsqueda
        }
    }

    //Elimina el registro tambien de la lista de la tabla de procesos registrando su salida
    for (int i = 0; i < Sinc.LTabla.size(); i++) { //Recorre toda la lista
        if(Sinc.LTabla.get(i).nombre.equals(proceso)){
            Registro r = new Registro();
            //Se agrega un nuevo registro exactamente igual pero con los datos de salida y memoria en cero
            r.nombre = Sinc.LTabla.get(i).nombre;
            r.resultadoBuddy = Sinc.LTabla.get(i).resultadoBuddy;
            r.resultadoMejorA = Sinc.LTabla.get(i).resultadoMejorA;
            r.resultadoPimerA = Sinc.LTabla.get(i).resultadoPimerA;
            r.resultadoProximoA = Sinc.LTabla.get(i).resultadoProximoA;
            r.tipoES = "Salida";
            Sinc.LTabla.add(0,r);
            break;
        }
    }
}
}

```

- **StateChanged()**. Responde al cambio en el deslizante de brillo de color, asigna el valor de G (componente verde) de acuerdo con el valor del deslizante y luego llama a la función colorea con el nuevo valor de G.

- **Colorea ()**. Con el nuevo valor de G. La función ‘colorea’, colorea el cuadro de colores en el panel utilizando un bucle anidado para iterar a través de diferentes combinaciones de colores basadas en el valor de G.



```

public void stateChanged(ChangeEvent e) {
    //En caso de cambiar el deslizante de el brillo de color
    //asignamos G deacuerdo a su valor y lo enviamos a "colorea"
    G = deslizanteColor.getValue();
    this.colorea(G);
}

private void colorea(int G) {
    //Coloreamos el cuadro de colores
    Graphics papelC = panelColor.getGraphics();
    for (int i = 0; i < 51; i+=1) {
        for (int j = 0; j < 51; j+=1) {
            papelC.setColor(new Color(i*5,G,j*5));
            papelC.fillRect(2+i*2, 2+j*2, 2, 2);
        }
    }
}

```

- **ActualizaTabla()**. Dinámicamente actualiza la interfaz gráfica de una tabla utilizando un objeto DefaultTableModel. Itera sobre los registros en Sinc.LTabla y llena una fila con información específica de cada registro. La función agrega la fila a la tabla, filtrando según el tipo de elemento de servicio seleccionado en listaTipoES. Permite visualizar todas las filas, solo aquellas relacionadas con procesos liberados, entradas o salidas, ajustando la interfaz en tiempo real.

```

public void actualizaTabla(){
    DefaultTableModel dtm;
    //Crea un objeto para la fila de la tabla
    Object[] fila = new Object[7];
    //Obtiene el modelo de la tabla
    dtm = (DefaultTableModel) tablaHistorial.getModel();
    dtm.getDataVector().removeAllElements(); //Limpia la tabla
    dtm.fireTableDataChanged();
    //Recorre los registros de la tabla
    for (int l = 0; l < Sinc.LTabla.size(); l++) {
        //Llena el objeto fila con lo correspondiente al registro
        fila[0] = Sinc.LTabla.get(l).tipoES;
        fila[1] = Sinc.LTabla.get(l).nombre;
        fila[2] = Sinc.LTabla.get(l).tamanyo;
        switch(Sinc.LTabla.get(l).resultadoPimerA){
            case 0: //Bloqueado
                fila[3] = "Bloqueado";
                break;
            case 1: //Ingreso
                fila[3] = "Ingresado";
                break;
            case 2: //Inancion
                fila[3] = "Inanición";
                break;
        }
        switch(Sinc.LTabla.get(l).resultadoMejora){
            case 0: //Bloqueado
                fila[4] = "Bloqueado";
                break;
            case 1: //Ingreso
                fila[4] = "Ingresado";
                break;
            case 2: //Inancion
                fila[4] = "Inanición";
                break;
        }
        switch(Sinc.LTabla.get(l).resultadoProximoA){
            case 0: //Bloqueado
                fila[5] = "Bloqueado";
                break;
            case 1: //Ingreso
                fila[5] = "Ingresado";
                break;
            case 2: //Inancion
                fila[5] = "Inanición";
                break;
        }
        switch(Sinc.LTabla.get(l).resultadoBuddy){
            case 0: //Bloqueado
                fila[6] = "Bloqueado";
                break;
            case 1: //Ingreso
                fila[6] = "Ingresado";
                break;
            case 2: //Inancion
                fila[6] = "Inanición";
                break;
        }
        //Añade la fila a la tabla
        if(listaTipoES.getSelectedIndex() == 0){
            dtm.addRow(fila);
        }
        if(listaTipoES.getSelectedIndex() == 1 && compLiberarProceso(fila[1].toString())){ //Flujo normal
            dtm.addRow(fila);
        }
        if(listaTipoES.getSelectedIndex() == 2 && fila[6] == "Entrada"){
            dtm.addRow(fila);
        }
        if(listaTipoES.getSelectedIndex() == 3 && fila[6] == "Salida"){
            dtm.addRow(fila);
        }
    }
}

```

- **Start ()**. Inicializa el sistema solicitando información del tamaño de la memoria para los ajustes de 'primer, próximo y mejor' y el tamaño del sistema buddy. Utiliza la clase Validaciones para asegurar que las entradas sean números válidos. Posteriormente, actualiza etiquetas en la interfaz gráfica con la información ingresada y llama a la función inicializa.

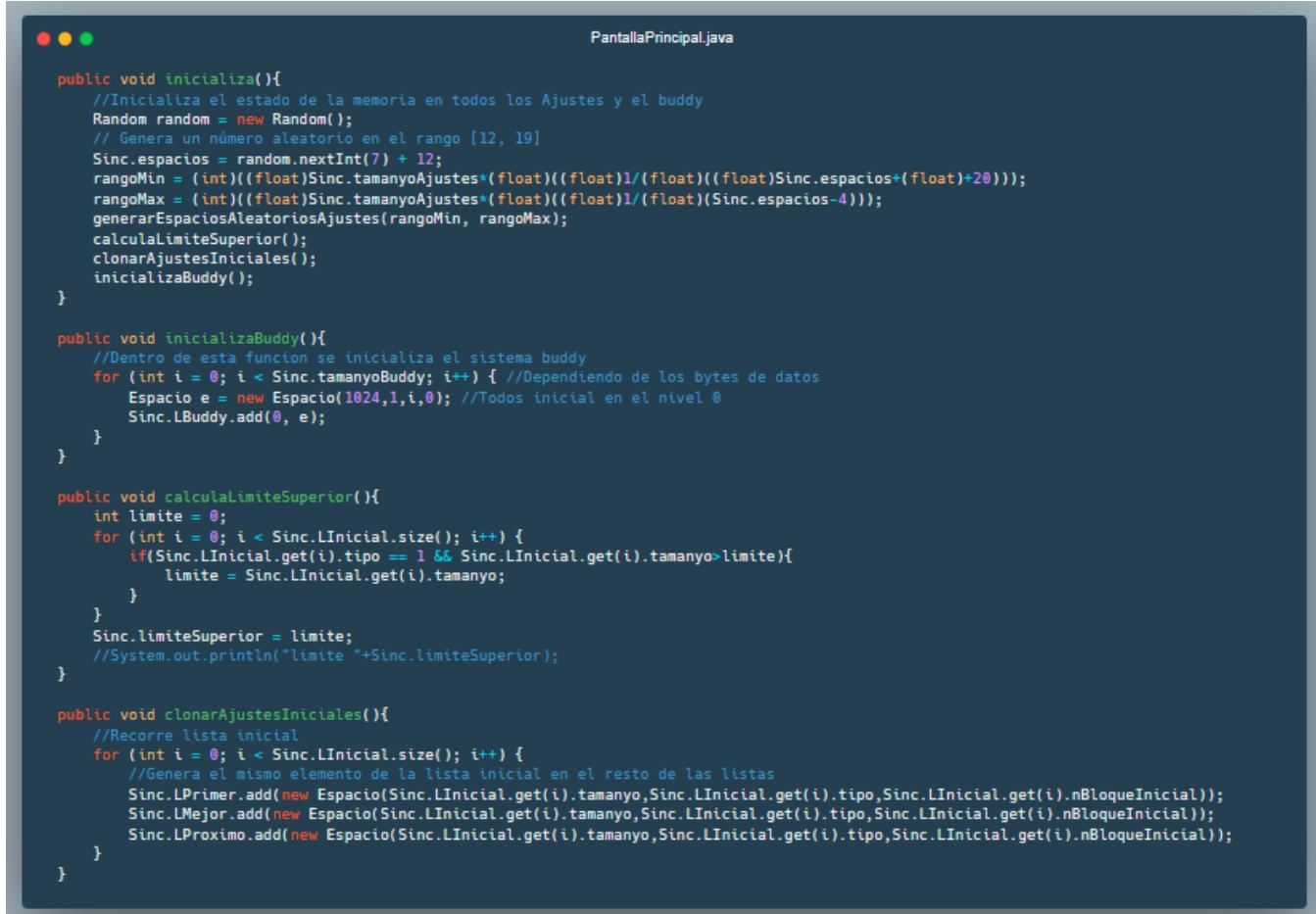


```

public void start(){
    //Se obtienen los valores de los tamaños de memoria
    Sinc.tamanyoAjustes = Validaciones.validarEntero("Ingresa tamaño de memoria para los ajustes 'primer, próximo y mejor':");
    Sinc.tamanyoBuddy = Validaciones.validarPotenciaDeDos("Ingresa el nº de megas en el sistema buddy");
    lbTamAjustes.setText("Tamaño Ajustes: "+Sinc.tamanyoAjustes+ " kilobyte ");
    lbTamBuddy.setText("Tamaño Buddy: "+Sinc.tamanyoBuddy + " Megabyte ");
    inicializa();
}

```

- **Inicializa ()**. Establece el estado inicial de la memoria en ajustes y el sistema buddy. Utiliza un generador de números aleatorios para determinar la cantidad de espacios disponibles y calcula un rango de tamaños para los espacios libres en los ajustes. Luego, invoca funciones específicas para asignar valores iniciales a los espacios de memoria en ajustes y el sistema buddy.
- **InicializaBuddy()**. Inicializa el sistema buddy. Itera a través de los bytes de datos en el sistema buddy y crea un objeto Espacio para cada byte con un tamaño de 1024 bytes, nivel 0, y lo agrega a la lista LBuddy.
- **CalculaLimiteSuperior()**. Recorre la lista inicial LInicial y va obteniendo el elemento mayor de los espacios libres en memoria.
- **ClonarAjustesIniciales()**. Recorre la lista inicial LInicial y genera elementos equivalentes en las listas LPrimer, LMejor, y LProximo. Para cada elemento en LInicial, se crea un nuevo objeto Espacio con el mismo tamaño, tipo y número de bloque inicial, y se agrega a las listas respectivas LPrimer, LMejor, y LProximo.



```

public void inicializa(){
    //Inicializa el estado de la memoria en todos los Ajustes y el buddy
    Random random = new Random();
    // Genera un número aleatorio en el rango [12, 19]
    Sinc.espacios = random.nextInt(7) + 12;
    rangoMin = (int)((float)Sinc.tamanyoAjustes*(float)((float)1//(float)((float)Sinc.espacios+(float)+20)));
    rangoMax = (int)((float)Sinc.tamanyoAjustes*(float)((float)1//(float)(Sinc.espacios-4)));
    generarEspaciosAleatoriosAjustes(rangoMin, rangoMax);
    calculaLimiteSuperior();
    clonarAjustesIniciales();
    inicializaBuddy();
}

public void inicializaBuddy(){
    //Dentro de esta función se inicializa el sistema buddy
    for (int i = 0; i < Sinc.tamanyoBuddy; i++) { //Dependiendo de los bytes de datos
        Espacio e = new Espacio(1024,1,i,0); //Todos inicial en el nivel 0
        Sinc.LBuddy.add(0, e);
    }
}

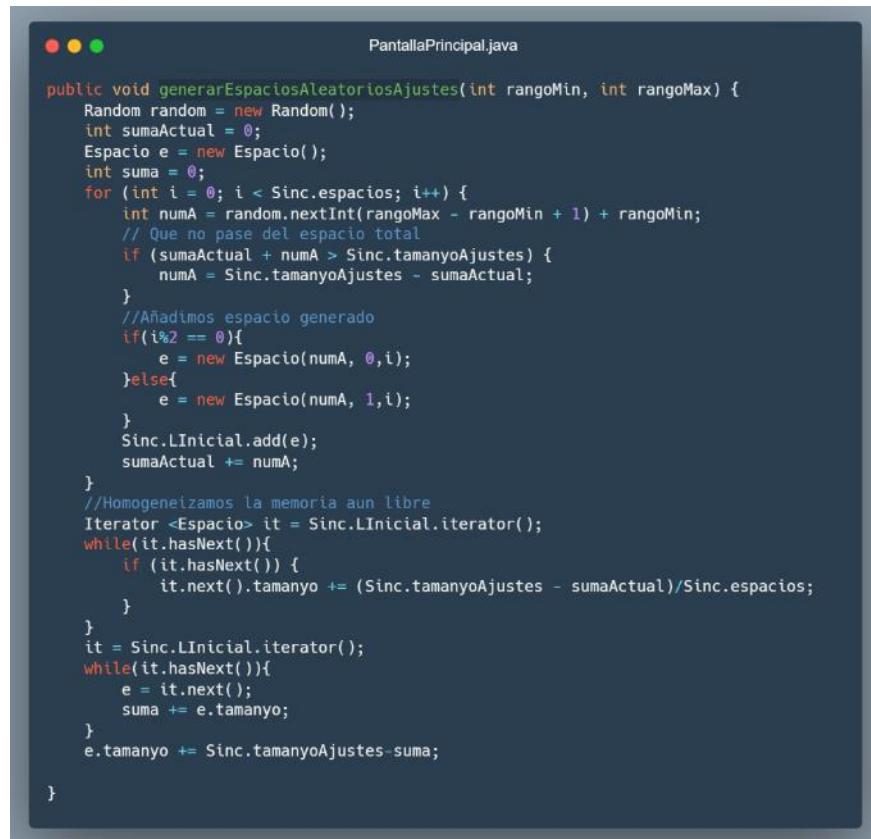
public void calculaLimiteSuperior(){
    int limite = 0;
    for (int i = 0; i < Sinc.LInicial.size(); i++) {
        if(Sinc.LInicial.get(i).tipo == 1 && Sinc.LInicial.get(i).tamanyo>limite){
            limite = Sinc.LInicial.get(i).tamanyo;
        }
    }
    Sinc.limiteSuperior = limite;
    //System.out.println("limite "+Sinc.limiteSuperior);
}

public void clonarAjustesIniciales(){
    //Recorre lista inicial
    for (int i = 0; i < Sinc.LInicial.size(); i++) {
        //Genera el mismo elemento de la lista inicial en el resto de las listas
        Sinc.LPrimer.add(new Espacio(Sinc.LInicial.get(i).tamanyo,Sinc.LInicial.get(i).tipo,Sinc.LInicial.get(i).nBloqueInicial));
        Sinc.LMejor.add(new Espacio(Sinc.LInicial.get(i).tamanyo,Sinc.LInicial.get(i).tipo,Sinc.LInicial.get(i).nBloqueInicial));
        Sinc.LProximo.add(new Espacio(Sinc.LInicial.get(i).tamanyo,Sinc.LInicial.get(i).tipo,Sinc.LInicial.get(i).nBloqueInicial));
    }
}

```

- **GenerarEspaciosAleatoriosAjustes()**. Utiliza un generador de números aleatorios para crear espacios con tamaños aleatorios dentro de un rango especificado (rangoMin a rangoMax). Estos espacios se agregan a la lista LInicial. Además, la función homogeneiza la memoria aún libre y asigna la memoria restante al último

espacio. Finalmente, imprime la lista LInicial

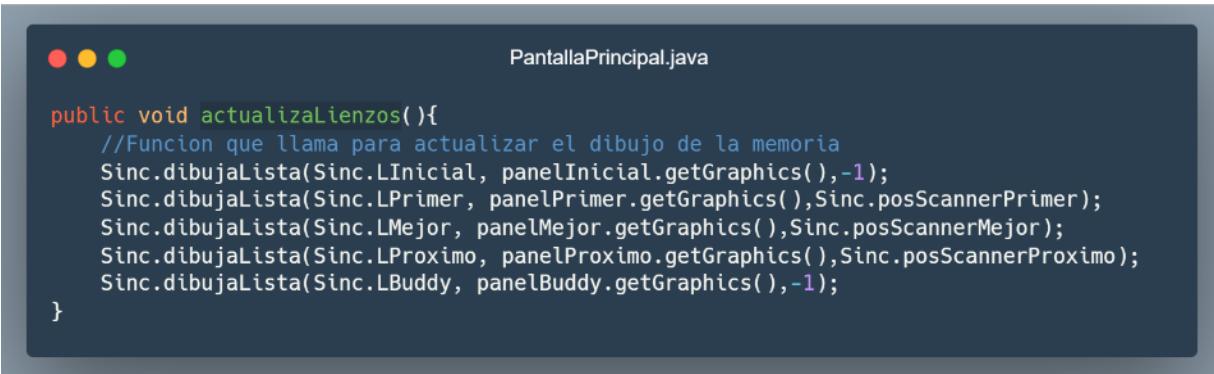


```

public void generarEspaciosAleatoriosAjustes(int rangoMin, int rangoMax) {
    Random random = new Random();
    int sumaActual = 0;
    Espacio e = new Espacio();
    int suma = 0;
    for (int i = 0; i < Sinc.espacios; i++) {
        int numA = random.nextInt(rangoMax - rangoMin + 1) + rangoMin;
        // Que no pase del espacio total
        if (sumaActual + numA > Sinc.tamanyoAjustes) {
            numA = Sinc.tamanyoAjustes - sumaActual;
        }
        //Añadimos espacio generado
        if(i&2 == 0){
            e = new Espacio(numA, 0,i);
        }else{
            e = new Espacio(numA, 1,i);
        }
        Sinc.LInicial.add(e);
        sumaActual += numA;
    }
    //Homogeneizamos la memoria aun libre
    Iterator <Espacio> it = Sinc.LInicial.iterator();
    while(it.hasNext()){
        if (it.hasNext()) {
            it.next().tamanyo += (Sinc.tamanyoAjustes - sumaActual)/Sinc.espacios;
        }
    }
    it = Sinc.LInicial.iterator();
    while(it.hasNext()){
        e = it.next();
        suma += e.tamanyo;
    }
    e.tamanyo += Sinc.tamanyoAjustes-suma;
}

```

- **ActualizaLienzo()**. Actualiza la representación gráfica de las listas de memoria (LInicial, LPrimer, LMejor, LProximo, LBuddy) en sus respectivos paneles. Utiliza la función dibujaLista para dibujar los elementos de cada lista y muestra la posición del escáner en las listas LPrimer, LMejor, y LProximo.



```

public void actualizaLienzos(){
    //Funcion que llama para actualizar el dibujo de la memoria
    Sinc.dibujaLista(Sinc.LInicial, panelInicial.getGraphics(),-1);
    Sinc.dibujaLista(Sinc.LPrimer, panelPrimer.getGraphics(),Sinc.posScannerPrimer);
    Sinc.dibujaLista(Sinc.LMejor, panelMejor.getGraphics(),Sinc.posScannerMejor);
    Sinc.dibujaLista(Sinc.LProximo, panelProximo.getGraphics(),Sinc.posScannerProximo);
    Sinc.dibujaLista(Sinc.LBuddy, panelBuddy.getGraphics(),-1);
}

```

- **Sinc**. Clase que realiza la invocación a PantallaPrincipal. Contiene los atributos y listas de control de memoria correspondientes a cada bloque, así como variables auxiliares globales.
 - Arreglos de listas de Espacio: LInicial, LPrimer, LProximo, LMejor, LBuddy, LTabla
 - Atributos:
 - Variables auxiliares globales: enteros estáticos; tamanyoBuddy, tamanyoAjustes, espacios, cuentaNombre, recorridoPuntero

- Variable tipo PantallaPrincipal pantallaPrincipal
- Almacenamiento en tiempo real (para agregar nuevo bloque, movimiento de puntero): Entero: posScannerPrimer, posScannerMejor, posScannerProximo
- Variables para las gráficas de barras: DefaultCategoryDataset: datos, datosFragmentacionBuddy.
- Para las gráficas de pastel: DefaultPieDataset: porcentajeBuddy, porcentajePrimerAjuste, porcentajeProximoAjuste, porcentajeMejorAjuste
- Para calcular fragmentación: Float; fragTotalPrimerA, fragTotalMejorA, fragTotalBuddyA, FragTotalProximoA
- Para pantallas: Contiene objetos de las clases específicas para visualizar cada pantalla, sea la pantalla principal, la fragmentación, los ajustes, graficas de barras y de pastel.
- Métodos:
 - DibujaLista(). Dibuja el bloque del ajuste, según sea el caso y actualiza correspondiendo al color elegido y el tamaño del proceso.

```

public static void dibujaLista (ArrayList <Espacio> LAjuste, Graphics papel, int pos){
    //Tam lienzo [700, 60]
    int x = 5;
    int y = 5;
    int width = 0;

    //Se pinta el controrno de la memoria de verde si es que logro entrar y de rojo si es que no lo hizo
    if(compruebaUltimaEntrada(LAjuste,papel)){
        papel.setColor(Color.green);
    }else{
        papel.setColor(Color.red);
    }

    papel.fillRect(0, 0, 700, 40);

    papel.setColor(Color.white);
    papel.fillRect(5, 5, 690, 30);
    Font font = new Font("Arial",Font.BOLD,10);
    papel.setFont(font);

    for (int i = 0; i < LAjuste.size(); i++) {
        //En caso de ser el buddy se hace regla de 3 con 1024 kb
        if(LAjuste.containsAll(Sinc.LBuddy)){
            //En caso de que sea el sistema buddy//Regla de 3
            width = (int)((float)694*(float)LAjuste.get(i).tamano)/(float)(Sinc.tamanyoBuddy*1024));
        }else{
            //En caso de ser ajuste//Regla de 3
            width = (int)((float)694*(float)LAjuste.get(i).tamano)/(float)Sinc.tamanyoAjustes);
            //Si el bloque actual es el mismo del scanner agrega marco magenta
            if(pos!=i && i==pos ){
                papel.setColor(new Color(0, 0, 0));//puede ser negro tambien jaja
                papel.fillRect(x-5, 0, width+10, 40);
            }
        }
        //Pinta el cuadro correspondiente al tipo de espacio
        if (LAjuste.get(i).tipo == 0) { //Ocupadisimo
            papel.setColor(Color.gray);
            if(pos!=i && i==pos+1){
                papel.fillRect(x+5, y, width-5, 30);
            }else{
                papel.fillRect(x, y, width, 30);
            }
            papel.setColor(Color.white);
            papel.drawString(Integer.toString(LAjuste.get(i).tamano), x-5+width/2, y+20);
        }
        if (LAjuste.get(i).tipo == 1) { //Libre
            papel.setColor(Color.LIGHT_GRAY);
            if(pos!=i && i==pos+1){
                papel.drawRect(x+5, y, width-5, 30);
            }else{
                papel.drawRect(x, y, width, 30);
            }
            papel.drawString(Integer.toString(LAjuste.get(i).tamano), x-5+width/2, y+20);
        }
        if (LAjuste.get(i).tipo == 2) { //Ocupado
            papel.setColor(new Color(LAjuste.get(i).color[0], LAjuste.get(i).color[1], LAjuste.get(i).color[2]));
            if(pos!=i && i==pos+1){
                papel.fillRect(x+5, y, width-5, 30);
            }else{
                papel.fillRect(x, y, width, 30);
            }
            papel.setColor(Color.black);
            papel.drawString(LAjuste.get(i).nombre, x-5+width/2, y+20);
        }
        x+=width;
    }
}

```

▪ Comprueba ultima entrada () .

- Atributos: boolean->Bandera (para indicar si el proceso ha entrado, o no en el bloque). Espacio->LAjuste, Graphics->Papel (para mostrar de manera grafica).
- Este método recorre la tabla completa (sea de algún ajuste), recibe el espacio del ajuste, y el lienzo a modificar. Si encuentra una entrada al proceso dentro de la tabla, la bandera será puesta en true, si el proceso sale, la bandera será puesta en

falso, de otro modo, regresara el resultado del ajuste, según sea la puesta de la bandera.

```

● ● ● PantallaPrincipal.java

public static int compruebaUltimaEntrada(ArrayList <Espacio> LAjuste, Graphics papel){
    //Comprueba si es que el proceso entro o no
    boolean bandera = true;
    //Recorre la lista completa de la tabla
    for (int i = 0; i < Sinc.LTabla.size(); i++) {
        bandera = true; //reiniciamos bandera

        //En caso de encontrar una entrada
        if(Sinc.LTabla.get(i).tipoES == "Entrada"){
            //Comprobamos que ese registro no haya salido ya
            for (int j = 0; j < Sinc.LTabla.size(); j++) {
                //En caso de haber salido cambiamos bandera a false
                if(Sinc.LTabla.get(i).nombre == Sinc.LTabla.get(j).nombre && Sinc.LTabla.get(j).tipoES == "Salida"){
                    bandera = false;
                }
            }
            //retornamos si es que el ultimo proceso entro exitosamente o no
            if(bandera){
                if(Sinc.LInicial.containsAll(LAjuste)){
                    return 1;
                }
                if(Sinc.LPrimer.containsAll(LAjuste)){
                    return Sinc.LTabla.get(i).resultadoPimerA;
                }
                if(Sinc.LMejor.containsAll(LAjuste)){
                    return Sinc.LTabla.get(i).resultadoMejorA;
                }
                if(Sinc.LProximo.containsAll(LAjuste)){
                    return Sinc.LTabla.get(i).resultadoProximoA;
                }
                if(Sinc.LBuddy.containsAll(LAjuste)){
                    return Sinc.LTabla.get(i).resultadoBuddy;
                }
            }
        }
    }
    return 0;
}

```

■ CalculaLeyenda () .

- **Atributos:** String->leyenda, nombre. Float->ajuste eficiente
- La función evalúa la eficiencia de diversos ajustes, inicialmente asumiendo que el primero es el más eficiente, además de considerar el sistema Buddy. Luego, compara la fragmentación de cada ajuste con la del ajuste inicial. Si alguno de los ajustes (mejor, próximo, primer) tiene una fragmentación menor que el estado inicial, se considera como el más eficiente, y se almacena en el atributo "nombre" un string que indica el tipo de ajuste resultante. Devuelve una leyenda con el nombre y fragmentación total del más eficiente.

```

● ● ● Sinc.java
public static String calculaLeyenda(){
    //Funcion que analiza cual de los ajustes es mas eficiente demas del sistema buddy
    String leyenda = "";
    String nombre = "Primer Ajuste";

    //Se asume que el primer ajuste es el mas eficiente al comienzo
    float ajusteEficiente = fragTotalPrimerA;

    //Se compara con el mejor ajuste
    if(fragTotalMejorA < ajusteEficiente){
        ajusteEficiente = fragTotalMejorA;
        nombre = "Mejor Ajuste";
    }

    //Se compara con el proximo ajuste
    if(fragTotalProximoA < ajusteEficiente){
        ajusteEficiente = fragTotalProximoA;
        nombre = "Proximo Ajuste";
    }

    //Se compone la leyenda
    leyenda = "Ajuste mas eficiente es \"" + nombre + "\": " + ajusteEficiente + "% - Sistema Buddy: " + fragTotalSistemaBuddy + "%";
}

}

```

- **Esperar () .**

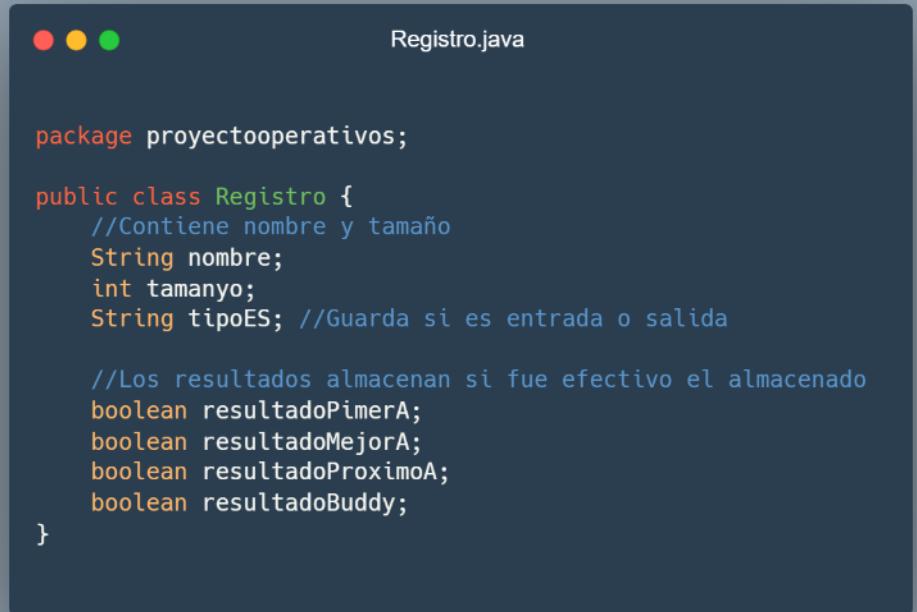
- **Atributos. Double->Segundos**
- Función que no devuelve nada, recibe un doble(segundos), y sirve para crear un espacio de tiempo, garantizando de este modo una mejor visualización del flujo de eventos

```

● ● ● Sinc.java
public static void esperar(double segundos) {
    try {
        Thread.sleep ((int)(segundos*1000));
    } catch (Exception e) {
        // Mensaje en caso de que falle
        System.out.println("Algo ha salido mal (1)");
    }
}

```

- **Registro.** Clase que contiene datos de los procesos registrados, los resultados almacenan si fue efectivo el almacenado.
 - Atributos
 - String nombre;
 - Entero tamanyo;
 - boolean : resultadoPimerA, ResultadoMejorA, ResultadoProximoA, resultadoBuddy.
 - Métodos:
 - No hay métodos en esta clase



```

Registro.java

package proyectooperativos;

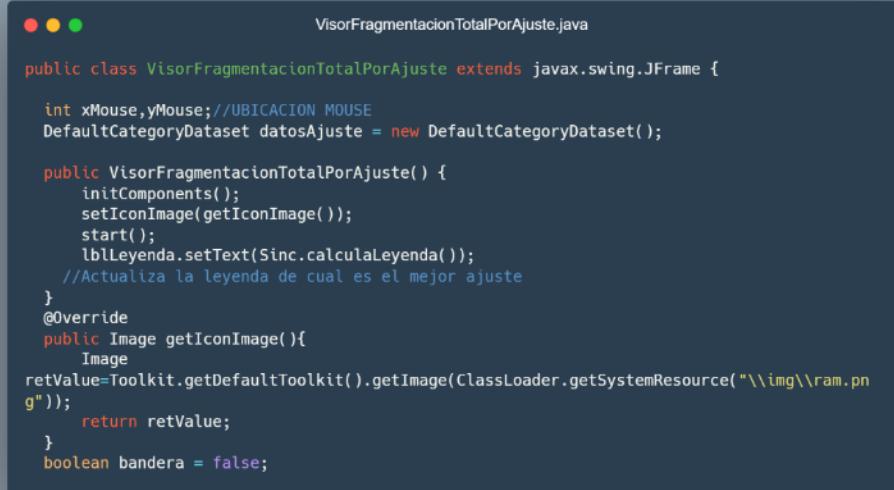
public class Registro {
    //Contiene nombre y tamaño
    String nombre;
    int tamanyo;
    String tipoES; //Guarda si es entrada o salida

    //Los resultados almacenan si fue efectivo el almacenado
    boolean resultadoPimerA;
    boolean resultadoMejorA;
    boolean resultadoProximoA;
    boolean resultadoBuddy;
}

```

- **VisorFragmentacionTotalPorAjuste()**.

- Atributos:



```

VisorFragmentacionTotalPorAjuste.java

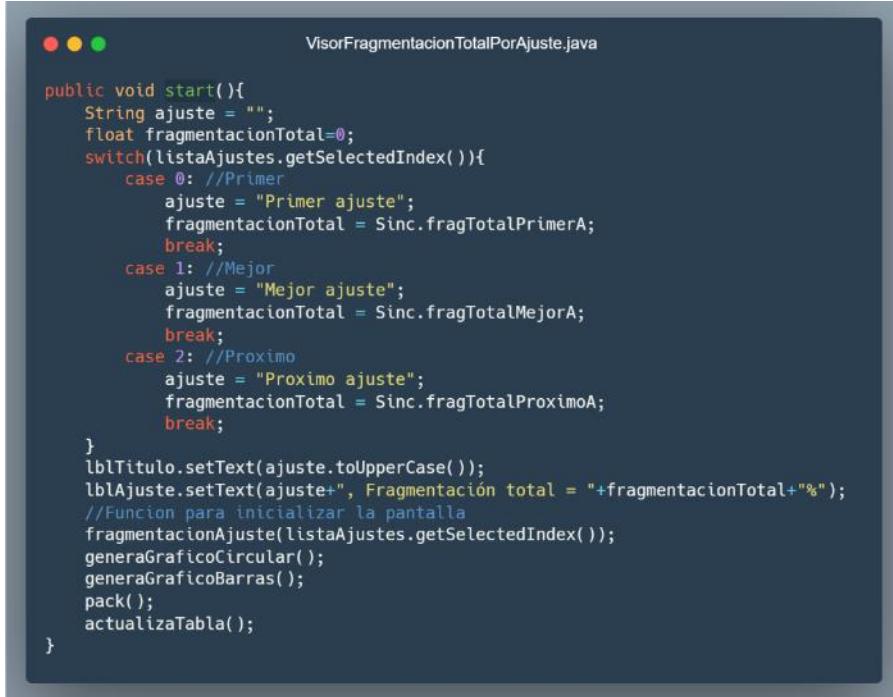
public class VisorFragmentacionTotalPorAjuste extends javax.swing.JFrame {

    int xMouse,yMouse;/UBICACION MOUSE
    DefaultCategoryDataset datosAjuste = new DefaultCategoryDataset();

    public VisorFragmentacionTotalPorAjuste() {
        initComponents();
        setIconImage(getIconImage());
        start();
        lblLeyenda.setText(Sinc.calculaLeyenda());
        //Actualiza la leyenda de cual es el mejor ajuste
    }
    @Override
    public Image getIconImage(){
        Image
        returnValue=Toolkit.getDefaultToolkit().getImage(ClassLoader.getSystemResource("\\\\img\\\\ram.png"));
        return returnValue;
    }
    boolean bandera = false;
}

```

- Start(). Obtiene el índice la lista de ajustes, usando un switch, recoge los datos de fragmentación total. Inicializa la pantalla para mostrar la fragmentación en los gráficos de barras y circular. Además, actualiza la tabla.



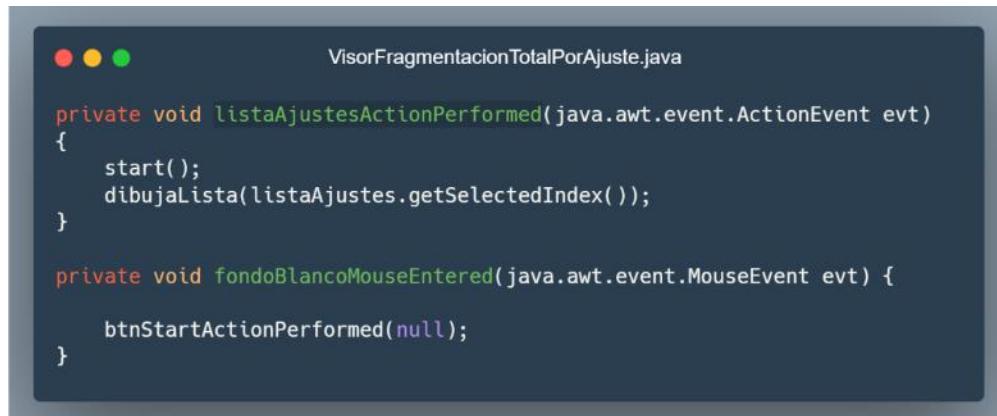
```

VisorFragmentacionTotalPorAjuste.java

public void start(){
    String ajuste = "";
    float fragmentacionTotal=0;
    switch(listaAjustes.getSelectedIndex()){
        case 0: //Primer
            ajuste = "Primer ajuste";
            fragmentacionTotal = Sinc.fragTotalPrimerA;
            break;
        case 1: //Mejor
            ajuste = "Mejor ajuste";
            fragmentacionTotal = Sinc.fragTotalMejorA;
            break;
        case 2: //Proximo
            ajuste = "Proximo ajuste";
            fragmentacionTotal = Sinc.fragTotalProximoA;
            break;
    }
    lblTitulo.setText(ajuste.toUpperCase());
    lblAjuste.setText(ajuste+", Fragmentación total = "+fragmentacionTotal+"%");
    //Funcion para inicializar la pantalla
    fragmentacionAjuste(listaAjustes.getSelectedIndex());
    generaGraficoCircular();
    generaGraficoBarras();
    pack();
    actualizaTabla();
}

```

- **ListaAjustesActionPerformed()**. Al recibir un evento de entrada, llama a start y dibujaLista(envía el índice de la lista de ajustes). En fondoBlancoMouseEntered recibe el evento de entrada del mouse, y manda llamar a btnStartActionPerformed(envía nulo). Estas funciones hacen llamadas a otras cuando reciben su evento con respecto a las listas de ajustes.



```

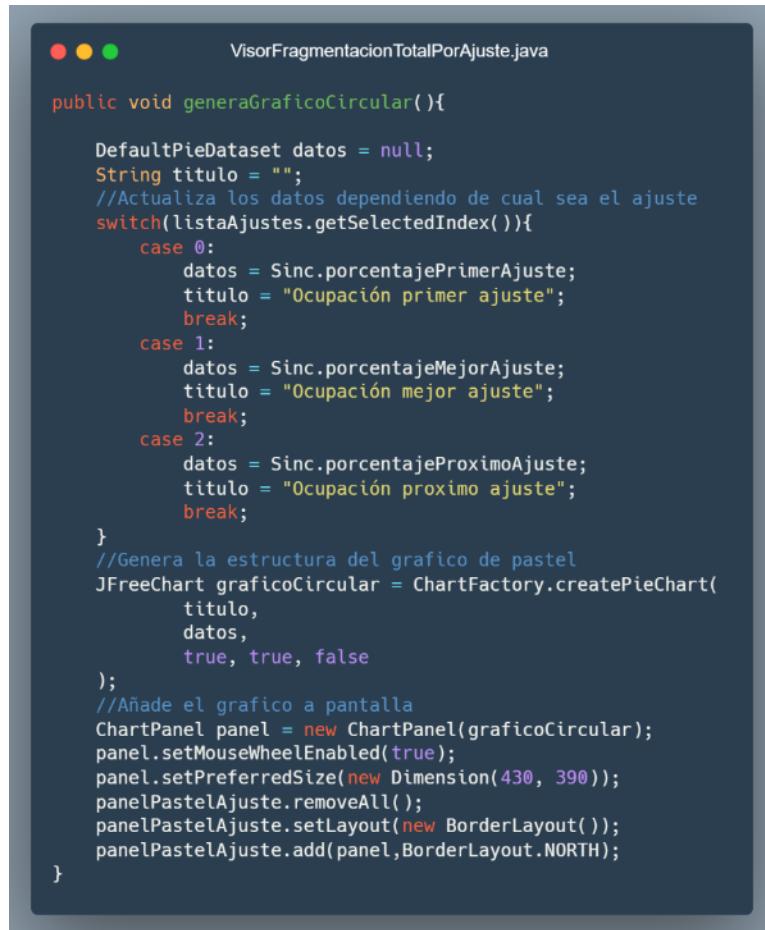
VisorFragmentacionTotalPorAjuste.java

private void listaAjustesActionPerformed(java.awt.event.ActionEvent evt)
{
    start();
    dibujaLista(listaAjustes.getSelectedIndex());
}

private void fondoBlancoMouseEntered(java.awt.event.MouseEvent evt) {
    btnStartActionPerformed(null);
}

```

- **GeneraGraficoCircular()**. Recupera datos de cada ajuste, genera la estructura del grafico de pastel, añade el grafico a la pantalla.



```
VisorFragmentacionTotalPorAjuste.java

public void generaGraficoCircular(){

    DefaultPieDataset datos = null;
    String titulo = "";
    //Actualiza los datos dependiendo de cual sea el ajuste
    switch(listaAjustes.getSelectedIndex()){
        case 0:
            datos = Sinc.porcentajePrimerAjuste;
            titulo = "Ocupación primer ajuste";
            break;
        case 1:
            datos = Sinc.porcentajeMejorAjuste;
            titulo = "Ocupación mejor ajuste";
            break;
        case 2:
            datos = Sinc.porcentajeProximoAjuste;
            titulo = "Ocupación proximo ajuste";
            break;
    }
    //Genera la estructura del grafico de pastel
    JFreeChart graficoCircular = ChartFactory.createPieChart(
        titulo,
        datos,
        true, true, false
    );
    //Añade el grafico a pantalla
    ChartPanel panel = new ChartPanel(graficoCircular);
    panel.setMouseWheelEnabled(true);
    panel.setPreferredSize(new Dimension(430, 390));
    panelPastelAjuste.removeAll();
    panelPastelAjuste.setLayout(new BorderLayout());
    panelPastelAjuste.add(panel,BorderLayout.NORTH);
}
```

- **GeneraGraficoBarras().** Se encarga de asignar el nombre del gráfico, según sea el ajuste del que se muestra. Recoge los datos de fragmentación de cada ajuste y los genera en grafico de barras y lo agrega al panel.

```
VisorFragmentacionTotalPorAjuste.java

public void generaGraficoBarras(){
    String titulo = "";
    //Dependiendo de cual se el ajuste sera el titulo
    switch(listaAjustes.getSelectedIndex()){
        case 0:
            titulo = "Fragmentación primer ajuste";
            break;
        case 1:
            titulo = "Fragmentación mejor ajuste";
            break;
        case 2:
            titulo = "Fragmentación proximo ajuste";
            break;
    }

    //Genera la estructura del grafico con los datos del Ajuste
    JFreeChart graficoAjuste = ChartFactory.createBarChart3D(
        titulo,
        "Fragmentación por bloque de memoria",
        "% de fragmentación total",
        datosAjuste,
        PlotOrientation.VERTICAL,
        true,
        true,
        false
    );

    //Genera la imagen de panel del grafico
    ChartPanel chartPanelAjustes = new ChartPanel(graficoAjuste);
    chartPanelAjustes.setMouseWheelEnabled(true);
    chartPanelAjustes.setPreferredSize(new Dimension(770, 390));

    //Añade el grafico en pantalla
    panelBarrasAjuste.removeAll();
    panelBarrasAjuste.setLayout(new BorderLayout());
    panelBarrasAjuste.add(chartPanelAjustes,BorderLayout.NORTH);
}
```

- **FragmentacionAjuste()**. Calcula y muestra la fragmentación de memoria para un tipo de ajuste (Primer, Mejor o Próximo Ajuste), almacenando los resultados para representación gráfica.

```

VisorFragmentacionTotalPorAjuste.java

public void fragmentacionAjuste( int numAjuste){
    float fragmentacion = 0;
    float suma = 0;
    datosAjuste = new DefaultCategoryDataset();
    ArrayList <Espacio> LAjuste = null;
    String tipo = "";
    //Actualiza los datos dependiendo de cual sea el ajuste
    switch(numAjuste){
        case 0: //Primer
            LAjuste = Sinc.LPrimer;
            tipo = "Primer ajuste";
            break;
        case 1: //Mejor
            LAjuste = Sinc.LMejor;
            tipo = "Mejor ajuste";
            break;
        case 2: //Proximo
            LAjuste = Sinc.LProximo;
            tipo = "Proximo ajuste";
            break;
    }
    //Recorre para comparar con la lista inicial
    for (int i = 0; i < Sinc.LInicial.size(); i++) {
        suma = 0;
        System.out.print("Bloque "+Sinc.LInicial.get(i).nBloqueInicial);
        //Calculamos la fragmentacion por bloques
        for (int j = 0; j < LAjuste.size(); j++) {
            //Si esta libre y es del mismo bloque
            if (LAjuste.get(j).tipo == 1 && LAjuste.get(j).nBloqueInicial == Sinc.LInicial.get(i).nBloqueInicial )
{
                //Suma contiene la suma del espacio libre de un mismo bloque
                suma += LAjuste.get(j).tamanyo; //Acumulamos los tamaños del mismo bloque
            }
        }
        //En caso de que el espacio sea tipo 0 o la fragmentacion sea del 100% reportamos 0 de fragmentacion
        if(Sinc.LInicial.get(i).tipo == 0 || (100*suma)/Sinc.LInicial.get(i).tamanyo == 100){
            fragmentacion = 0;
        }else{
            //Realizamos regla de 3 para conocer el % de fragmentación
            fragmentacion = (100*suma)/Sinc.tamanyoAjustes;
        }
        //Agregamos los datos a la lista de datos de fragmentacion para las graficas
        datosAjuste.setValue(fragmentacion,tipo,"B"+(i+1));
        System.out.println(", %Frag. = "+ fragmentacion);
    }
}

```

- **DibujaLista()**. Dibuja los bloques de memoria para un tipo de ajuste específico (Primer, Mejor o Próximo Ajuste). Utiliza la clase Graphics para visualizar el estado de los bloques, resaltando la ubicación del "scanner" y aplicando colores distintivos para bloques ocupados, libres y ocupados anteriormente. Además, muestra información relevante, como el tamaño y el nombre del proceso, dentro de los bloques ocupados.

SISTEMAS OPERATIVOS

```
● ● ● VisorFragmentacionTotalPorAjuste.java

public void dibujaLista (int numAjuste){
    ArrayList <Espacio> LAjuste = null;
    Graphics papel = panelAjuste.getGraphics();
    int x = 5;
    int y = 5;
    int width = 0;
    int pos = 0;

    //Actualiza los datos dependiendo de cual sea el ajuste
    switch(numAjuste){
        case 0: //Primer
            LAjuste = Sinc.LPrimer;
            pos = Sinc.posScannerPrimer;
            break;
        case 1: //Mejor
            LAjuste = Sinc.LMejor;
            pos = Sinc.posScannerMejor;
            break;
        case 2: //Proximo
            LAjuste = Sinc.LProximo;
            pos = Sinc.posScannerProximo;
            break;
    }
    //Se pinta el controrno de la memoria de verde si es que logro entrar y de rojo si es que no lo hizo
    if(compruebaUltimaEntrada(LAjuste,papel)){
        papel.setColor(Color.green);
    }else{
        papel.setColor(Color.red);
    }

    papel.fillRect(0, 0, 1420, 130);

    papel.setColor(Color.white);
    papel.fillRect(5, 5, 1410, 120);
    Font font = new Font("Arial",Font.BOLD,14);
    papel.setFont(font);

    for (int i = 0; i < LAjuste.size(); i++) {
        //En caso de ser el buddy se hace regla de 3 con 1024 kb
        font = new Font("Arial",Font.BOLD,14);
        papel.setFont(font);
        width = (int)((float)1410*(float)LAjuste.get(i).tamanyo)/(float)Sinc.tamanoAjustes); //Regla de
3

        //En caso de que el scanner se quedara en esta ubicacion se colocara un margen
        if(pos!=i && i==pos){
            papel.setColor(Color.MAGENTA);
            papel.fillRect(x-5, 0, width+10, 130);
        }

        if (LAjuste.get(i).tipo == 0) { //Ocupadisimo
            papel.setColor(Color.gray);
            if(pos!=i && i==pos+1){
                papel.fillRect(x+5, y, width-5, 120);
            }else{
                papel.fillRect(x, y, width, 120);
            }

            papel.setColor(Color.white);
            papel.drawString(Integer.toString(LAjuste.get(i).tamanyo), x-5+width/2, y+65);
        }

        if (LAjuste.get(i).tipo == 1) { //Libre
            papel.setColor(Color.LIGHT_GRAY);
            if(pos!=i && i==pos+1){
                papel.drawRect(x+5, y, width-5, 120);
            }else{
                papel.drawRect(x, y, width, 120);
            }

            papel.drawString(Integer.toString(LAjuste.get(i).tamanyo), x-5+width/2, y+65);
        }
        if (LAjuste.get(i).tipo == 2) { //Ocupado
            papel.setColor(new Color(LAjuste.get(i).color[0], LAjuste.get(i).color[1],
LAjuste.get(i).color[2]));
            if(pos!=i && i==pos+1){
                papel.fillRect(x+5, y, width-5, 120);
            }else{
                papel.fillRect(x, y, width, 120);
            }
            font = new Font("Arial",Font.BOLD,12);
            papel.setFont(font);

            papel.setColor(Color.black);
            papel.drawString("P:"+LAjuste.get(i).nombre, x-5+width/2, y+55);
            papel.drawString("T:"+Integer.toString(LAjuste.get(i).tamanyo), x-5+width/2-5, y+75);
            //PantallaPrincipal.flechaPrimer1.setBounds(x, 205, 16, 15);
            }
            x+=width;
        }
    }
}
```

- **compLiberarProceso_btnRegresar()**. Busca si un proceso específico ha completado su salida. Recorre la lista STabla y retorna false si encuentra una entrada correspondiente al proceso en cuestión con el tipo de evento "Salida", indicando que el proceso aún no ha salido. Si no encuentra ninguna entrada que cumpla con estas condiciones, retorna true, señalando que el proceso ha completado su salida.

```

VisorFragmentacionTotalPorAjuste.java

public boolean compLiberarProceso(String proceso){
    //retorna true si el proceso ya salio
    for (int i = 0; i < Sinc.LTabla.size(); i++) {
        if("Salida".equals(Sinc.LTabla.get(i).tipoES) && Sinc.LTabla.get(i).nombre.equals(proceso)){
            return false;
        }
    }
    return true;
}

private void btnRegresarActionPerformed(java.awt.event.ActionEvent evt) {
    //Regresa a pantalla principal
    Sinc.pantallaPrincipal.setVisible(true);
    Sinc.pantallaPrincipal.setSize(1300, 650);
    Sinc.pantallaPrincipal.setResizable(false);
    Sinc.pantallaPrincipal.setLocationRelativeTo(null); //en medio
    this.dispose();
}

```

- **BtnStartActionPerformed()**. Actualiza el dibujo de la memoria

```

VisorFragmentacionTotalPorAjuste.java

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    start();
}

private void btnStartActionPerformed(java.awt.event.ActionEvent evt) {
    start();
    //Funcion que llama para actualizar el dibujo de la memoria
    dibujaLista(listaAjustes.getSelectedIndex());
    if(!bandera){
        bandera = true;
        btnStartActionPerformed(evt);
    }
}

```

- **ActualizaTabla()**. Actualiza dinámicamente una tabla en la interfaz gráfica según el ajuste seleccionado y si los procesos han completado su salida. Utiliza DefaultTableModel y compLiberarProceso para gestionar los datos y filtrar las filas.

- **VisorFragmentacionTotalAjustes()**
 - **VisorFragmentacionTotalAjustesVar()**. implementa la clase VisorFragmentacionTotalAjustes para la interfaz gráfica. La ventana muestra información sobre el tamaño de ajustes y la ubicación de una flecha animada. Además, se establece un ícono personalizado para la aplicación (imagen de RAM).

```
VisorFragmentacionTotalAjustes.java

public class VisorFragmentacionTotalAjustes extends javax.swing.JFrame {
    int xMouse,yMouse;//UBICACION MOUSE
    public VisorFragmentacionTotalAjustes() {
        initComponents();
        setIconImage(getIconImage());
        lbTamAjustes.setText("Tamaño Ajustes: "+Sinc.tamanoAjustes+ " kilobyte ");
        flechaProx.setBounds(760+Sinc.recorridoPuntero-8, flechaProx.getBounds().y, 20, 30);
        lblLeyenda.setText(Sinc.calculaLeyenda());
    }
    @Override
    public Image getIconImage(){
        Image retVal=Toolkit.getDefaultToolkit().getImage(ClassLoader.getSystemResource("\\\\img\\\\ram.png"));
        return retVal;
    }
    boolean bandera = false;
```

- **starter y más.** La acción del botón "jButton1" inicia un proceso con la función start(). El botón "btnStart" invoca la función starter (), que, a su vez, llama a start() y actualiza gráficos y datos. Además, se llama a actualizaPuntero().



```

VisorFragmentacionTotalAjustes.java

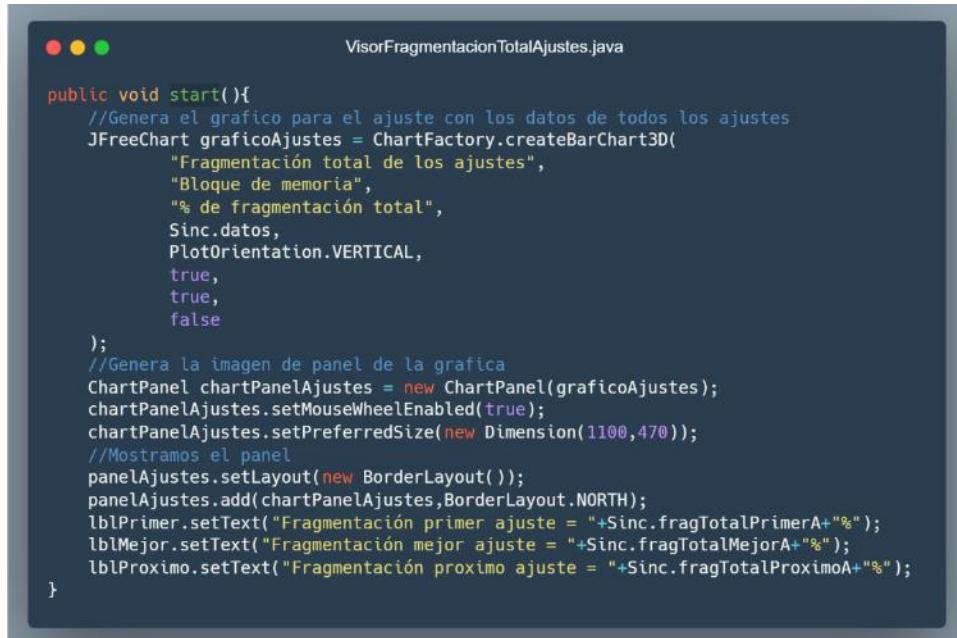
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    start();
}

private void btnStartActionPerformed(java.awt.event.ActionEvent evt) {
    starter();
    actualizaPuntero();
}

public void starter(){
    start();
    //Funcion que llama para actualizar el dibujo de la memoria
    Sinc.dibujaLista(Sinc.LInicial, panelInicial.getGraphics(),-1);
    Sinc.dibujaLista(Sinc.LPrimer, panelPrimer.getGraphics(),Sinc.posScannerPrimer);
    Sinc.dibujaLista(Sinc.LMejor, panelMejor.getGraphics(),Sinc.posScannerMejor);
    Sinc.dibujaLista(Sinc.LProximo, panelProximo.getGraphics(),Sinc.posScannerProximo);
    if(!bandera){
        bandera = true;
        starter();
    }
    //Se actualizan los datos de la tabla
    actualizaTabla();
}

```

- **Start().** Genera y muestra un gráfico de barras 3D que representa la fragmentación total de diferentes ajustes en bloques de memoria. Utiliza la biblioteca JFreeChart. Además, agrega etiquetas con los porcentajes de fragmentación total para los ajustes de Primer, Mejor y Próximo Ajuste.



```

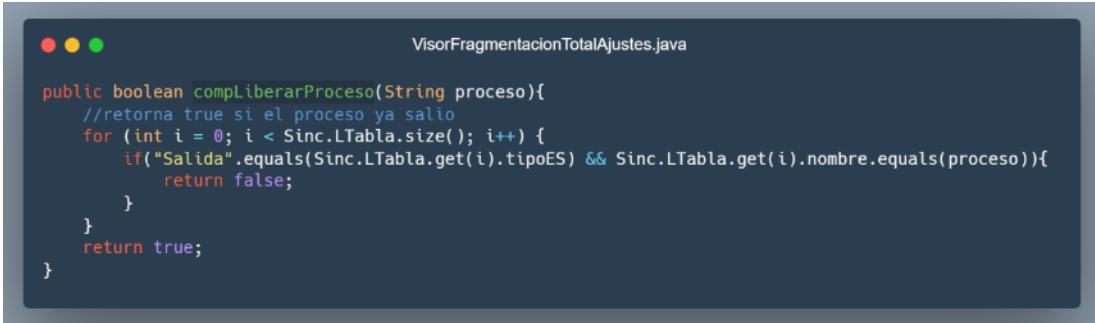
VisorFragmentacionTotalAjustes.java

public void start(){
    //Genera el grafico para el ajuste con los datos de todos los ajustes
    JFreeChart graficoAjustes = ChartFactory.createBarChart3D(
        "Fragmentación total de los ajustes",
        "Bloque de memoria",
        "% de fragmentación total",
        Sinc.datos,
        PlotOrientation.VERTICAL,
        true,
        true,
        false
    );
    //Genera la imagen de panel de la grafica
    ChartPanel chartPanelAjustes = new ChartPanel(graficoAjustes);
    chartPanelAjustes.setMouseWheelEnabled(true);
    chartPanelAjustes.setPreferredSize(new Dimension(1100,470));
    //Mostramos el panel
    panelAjustes.setLayout(new BorderLayout());
    panelAjustes.add(chartPanelAjustes,BorderLayout.NORTH);
    lblPrimer.setText("Fragmentación primer ajuste = "+Sinc.fragTotalPrimerA+"%");
    lblMejor.setText("Fragmentación mejor ajuste = "+Sinc.fragTotalMejorA+"%");
    lblProximo.setText("Fragmentación proximo ajuste = "+Sinc.fragTotalProximoA+"%");
}

```

- **ComLiberarProceso().** Verifica si un proceso ha completado su salida. Recorre la lista STabla y retorna false si encuentra una entrada correspondiente al proceso en cuestión con el tipo de evento "Salida",

indicando que el proceso aún no ha salido. Si no encuentra ninguna entrada que cumpla con estas condiciones, retorna true, señalando que el proceso ha completado su salida.



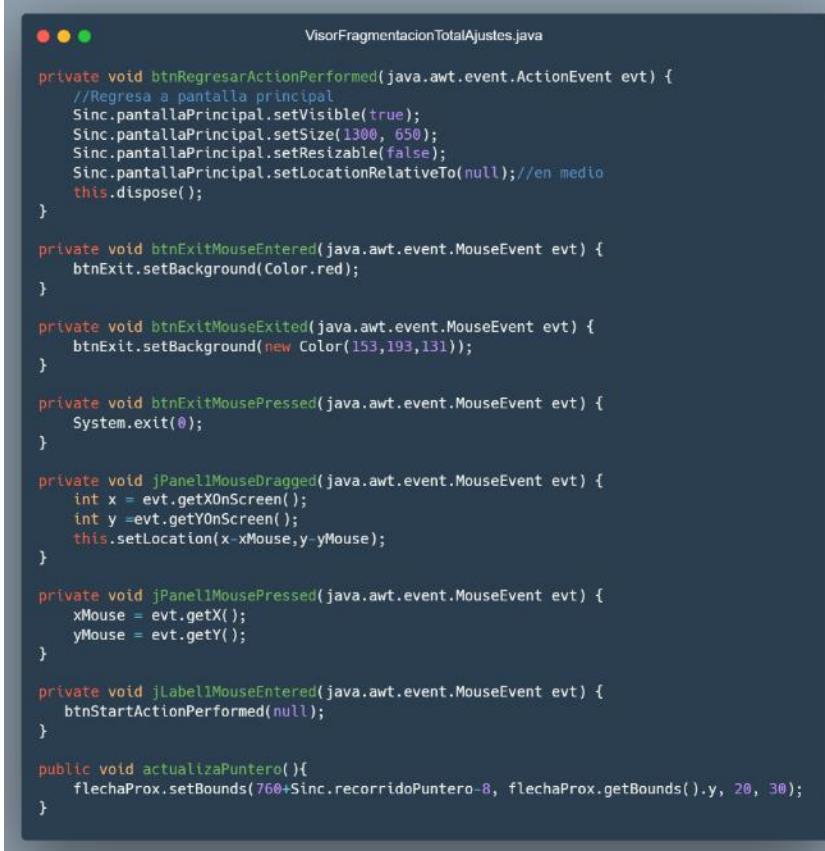
```

VisorFragmentacionTotalAjustes.java

public boolean compLiberarProceso(String proceso){
    //retorna true si el proceso ya salio
    for (int i = 0; i < Sinc.LTabla.size(); i++) {
        if("Salida".equals(Sinc.LTabla.get(i).tipoES) && Sinc.LTabla.get(i).nombre.equals(proceso)){
            return false;
        }
    }
    return true;
}

```

- **Botones ()**. Controla eventos en una interfaz gráfica. El botón **btnRegresar** redirige a la pantalla principal, mientras que **btnExit** cambia su color al pasar el ratón y cierra la aplicación al ser presionado. Las funciones relacionadas con **jPanel1** permiten arrastrar la ventana al mover el ratón. La función **actualizaPuntero()** ajusta la posición de un componente gráfico (**flechaProx**) en respuesta a eventos. La etiqueta **jLabel1** activa la función **btnStartActionPerformed** al pasar el ratón por encima.



```

VisorFragmentacionTotalAjustes.java

private void btnRegresarActionPerformed(java.awt.event.ActionEvent evt) {
    //Regresa a pantalla principal
    Sinc.pantallaPrincipal.setVisible(true);
    Sinc.pantallaPrincipal.setSize(1300, 650);
    Sinc.pantallaPrincipal.setResizable(false);
    Sinc.pantallaPrincipal.setLocationRelativeTo(null); //en medio
    this.dispose();
}

private void btnExitMouseEntered(java.awt.event.MouseEvent evt) {
    btnExit.setBackground(Color.red);
}

private void btnExitMouseExited(java.awt.event.MouseEvent evt) {
    btnExit.setBackground(new Color(153, 193, 131));
}

private void btnExitMousePressed(java.awt.event.MouseEvent evt) {
    System.exit(0);
}

private void jPanel1MouseDragged(java.awt.event.MouseEvent evt) {
    int x = evt.getXOnScreen();
    int y = evt.getYOnScreen();
    this.setLocation(x-xMouse,y-yMouse);
}

private void jPanel1MousePressed(java.awt.event.MouseEvent evt) {
    xMouse = evt.getX();
    yMouse = evt.getY();
}

private void jLabel1MouseEntered(java.awt.event.MouseEvent evt) {
    btnStartActionPerformed(null);
}

public void actualizaPuntero(){
    flechaProx.setBounds(760-Sinc.recorridoPuntero-8, flechaProx.getBounds().y, 20, 30);
}

```

- **ActualizaTabla()**. Actualiza dinámicamente los registros de una tabla en la interfaz gráfica. Utiliza un objeto DefaultTableModel para manipular los datos de la tabla, eliminando todos los elementos existentes y recorriendo la lista STabla. Llena el objeto fila con la información correspondiente a cada

registro (nombre del proceso, tamaño y resultados de los ajustes) y agrega las filas pertinentes al modelo de la tabla (dtm). La función compLiberarProceso filtra los procesos completados.

```
public void actualizaTabla(){
    DefaultTableModel dtm;
    //Crea un objeto para la fila de la tabla
    Object[] fila = new Object[5];
    //Obtiene el modelo de la tabla
    dtm = (DefaultTableModel) tablaHistorial.getModel();
    dtm.getDataVector().removeAllElements(); //Limpia la tabla
    dtm.fireTableDataChanged();
    //Recorre los registros de la tabla
    for (int i = 0; i < Sinc.LTabla.size(); i++) {
        //Llena el objeto fila con lo correspondiente al registro
        fila[0] = Sinc.LTabla.get(i).nombre;
        if(compLiberarProceso(fila[0].toString())){ //Flujo normal
            fila[1] = Sinc.LTabla.get(i).tamanyo;

            if(Sinc.LTabla.get(i).resultadoPimerA){
                fila[2] = "Si";
            }else{
                fila[2] = "No";
            }
            if(Sinc.LTabla.get(i).resultadoMejorA){
                fila[3] = "Si";
            }else{
                fila[3] = "No";
            }
            if(Sinc.LTabla.get(i).resultadoProximoA){
                fila[4] = "Si";
            }else{
                fila[4] = "No";
            }
            dtm.addRow(fila);
        }
    }
}
```

- **VisorFragmentacionTotalPasteles**
 - **VisorFragmentacionTotalPastelesVar()**. Variables de la clase

```

VisorFragmentacionTotalPasteles.java

public class VisorFragmentacionTotalPasteles extends javax.swing.JFrame {

    int xMouse,yMouse;//UBICACION MOUSE
    public VisorFragmentacionTotalPasteles() {
        initComponents();
        setIconImage(getIconImage());
        flechaProx.setBounds(760+Sinc.recorridoPuntero-8, flechaProx.getBounds().y, 20, 30);
        lblLeyenda.setText(Sinc.calculaLeyenda()); //Actualiza la leyenda de cual es el mejor ajuste
    }
    @Override
    public Image getIconImage(){
        Image
    retValue=Toolkit.getDefaultToolkit().getImage(ClassLoader.getSystemResource("\\\\img\\\\ram.png"));
        return retValue;
    }
    boolean bandera = false;

    public void start(){
        //Funcion para inicializar los graficos de pastel
        generaGrafico(0,panelGBuddy);
        generaGrafico(1,panelGPrimer);
        generaGrafico(2,panelGMejor);
        generaGrafico(3,panelGProximo);
    }
}

```

- **GeneraGrafico()**. Genera un gráfico circular dinámicamente en un panel de interfaz gráfica. Utiliza la biblioteca JFreeChart y un conjunto de datos (**DefaultPieDataset**) para generar la estructura del gráfico. El tipo de gráfico (ocupación de Buddy, Primer Ajuste, Mejor Ajuste, Próximo Ajuste) se determina mediante un parámetro (**tipo**). La función luego agrega el gráfico al panel correspondiente en la interfaz gráfica.

```

VisorFragmentacionTotalPasteles.java

public void generaGrafico(int tipo, JPanel gPanel){
    DefaultPieDataset datos = null;
    String titulo = "*";
    //Selecciona los datos y titulo dinamicamente
    switch(tipo){
        case 0:
            datos = Sinc.porcentajeBuddy;
            titulo = "Ocupación buddy";
            break;
        case 1:
            datos = Sinc.porcentajePrimerAjuste;
            titulo = "Ocupación primer ajuste";
            break;
        case 2:
            datos = Sinc.porcentajeMejorAjuste;
            titulo = "Ocupación mejor ajuste";
            break;
        case 3:
            datos = Sinc.porcentajeProximoAjuste;
            titulo = "Ocupación proximo ajuste";
            break;
    }
    //Genera la estructura del grafico
    JFreeChart graficoCircular = ChartFactory.createPieChart(
        titulo,
        datos,
        true, true, false
    );
    //Añade el grafico en su respectivo panel
    ChartPanel panel = new ChartPanel(graficoCircular);
    panel.setMouseWheelEnabled(true);
    panel.setPreferredSize(new Dimension(340,380));
    gPanel.setLayout(new BorderLayout());
    gPanel.add(panel,BorderLayout.NORTH);
    pack();
}

```

- **Botones ()**. El botón "jButton1" y "btnStart" activan la función starter (), que actualiza gráficos de memoria en varios paneles y reinicia el proceso para mantener la actualización continua. El botón "btnRegresar" regresa a la pantalla principal, y los eventos relacionados con "btnExit" cambian su apariencia y cierran la aplicación al ser presionado. El evento formMouseEntered y btnExitMouseEntered

activan la función starter () y actualizaPuntero() al entrar al formulario o al pasar el ratón sobre el botón "btnExit".

```

VisorFragmentacionTotalPasteles.java

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    start();
}

private void btnStartActionPerformed(java.awt.event.ActionEvent evt) {
    starter();
    actualizaPuntero();
}

public void starter(){
    //Funcion que llama para actualizar el dibujo de la memoria
    Sinc.dibujaLista(Sinc.LBuddy, panelBuddy.getGraphics(),-1);
    Sinc.dibujaLista(Sinc.LPrimer, panelPrimer.getGraphics(),Sinc.posScannerPrimer);
    Sinc.dibujaLista(Sinc.LMejor, panelMejor.getGraphics(),Sinc.posScannerMejor);
    Sinc.dibujaLista(Sinc.LProximo, panelProximo.getGraphics(),Sinc.posScannerProximo);
    if(!bandera){
        start();
        bandera = true;
        starter();
    }
}

private void btnRegresarActionPerformed(java.awt.event.ActionEvent evt) {
    //Regresa a pantalla principal
    Sinc.pantallaPrincipal.setVisible(true);
    Sinc.pantallaPrincipal.setSize(1300, 650);
    Sinc.pantallaPrincipal.setResizable(false);
    Sinc.pantallaPrincipal.setLocationRelativeTo(null); //en medio
    this.dispose();
}

private void formMouseEntered(java.awt.event.MouseEvent evt) {
    starter();
    actualizaPuntero();
}
public void actualizaPuntero(){
    flechaProx.setBounds(760+Sinc.recorridoPuntero-8, flechaProx.getBounds().y, 20, 30);
}

```

- **VisorFragmentacionTotalBuddy**

- **VisorFragmentacionTotalBuddyVar()**. Visor de fragmentación total en el sistema Buddy. Inicializa componentes gráficos en el constructor, establece un ícono personalizado para la aplicación (imagen de RAM), y muestra información sobre la fragmentación total del sistema Buddy y una leyenda. La ubicación del mouse (xMouse e yMouse) se utiliza en algún contexto no proporcionado en el fragmento de código. La variable bandera se usa como un indicador de control en algún punto del código que no se muestra aquí.

```
● ● ● VisorFragmentacionTotalBuddy.java

public class VisorFragmentacionTotalBuddy extends javax.swing.JFrame {

    int xMouse,yMouse; //UBICACION MOUSE
    public VisorFragmentacionTotalBuddy() {
        initComponents();
        setIconImage(getIconImage());
        lblSistemaBuddy.setText("Sistema buddy, Fragmentación total = "+Sinc.fragTotalSistemaBuddy+"%");
        lblLeyenda.setText(Sinc.calculaLeyenda());
    }
    @Override
    public Image getIconImage(){
        Image retValue=Toolkit.getDefaultToolkit().getImage(ClassLoader.getSystemResource("\\\\img\\\\ram.png"));
        return retValue;
    }
    boolean bandera = false;

    public void start(){
        //Funcion para inicializar la pantalla
        generaGraficoCircular();
        generaGraficoBarras();
        pack();
        actualizaTabla();
    }
}
```

- **GeneraGraficoCircular()**. Genera un gráfico circular de pastel. Utiliza la biblioteca JFreeChart y un conjunto de datos (**DefaultPieDataset**) para crear la estructura del gráfico. Los datos y el título se obtienen del objeto **Sinc** y se utilizan para generar un gráfico circular de pastel con la función **ChartFactory.createPieChart()**. El gráfico se agrega a un panel en la interfaz gráfica (**panelPastelBuddy**).

```

public void generaGraficoCircular(){
    DefaultPieDataset datos = null;
    String titulo = "";
    //Se obtienen los datos de la grafica desde el archivo
    sincronizado
    datos = Sinc.porcentajeBuddy;
    titulo = "Ocupación buddy";
    //Genera la estructura del grafico de pastel
    JFreeChart graficoCircular = ChartFactory.createPieChart(
        titulo,
        datos,
        true, true, false
    );
    //Añade el grafico a pantalla
    ChartPanel panel = new ChartPanel(graficoCircular);
    panel.setMouseWheelEnabled(true);
    panel.setPreferredSize(new Dimension(430, 390));
    panelPastelBuddy.setLayout(new BorderLayout());
    panelPastelBuddy.add(panel,BorderLayout.NORTH);
}

```

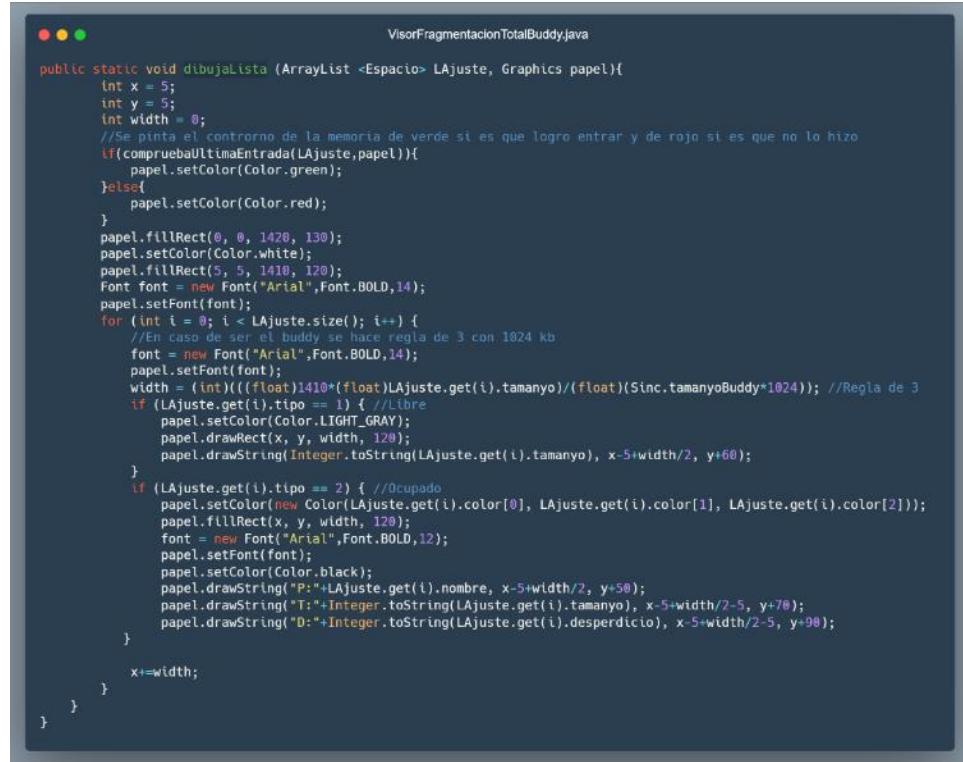
- **GeneraGraficoBarras()**. Genera un gráfico de barras 3D que representa la fragmentación del sistema Buddy. Utiliza la biblioteca JFreeChart y un conjunto de datos (Sinc.datosFragmentacionBuddy) para crear la estructura del gráfico. El gráfico se agrega a un panel en la interfaz gráfica (panelBarrasBuddy).

```

public void generaGraficoBarras(){
    //Genera la estructura del grafico con los datos del sistema buddy
    JFreeChart graficoBuddy = ChartFactory.createBarChart3D(
        "Fragmentación Sistema Buddy",
        "Sistema buddy",
        "Fragmentación",
        Sinc.datosFragmentacionBuddy,
        PlotOrientation.VERTICAL,
        true,
        true,
        false
    );
    //Genera la imagen de panel del grafico
    ChartPanel chartPanelAjustes = new ChartPanel(graficoBuddy);
    chartPanelAjustes.setMouseWheelEnabled(true);
    chartPanelAjustes.setPreferredSize(new Dimension(770, 390));
    //Añade el grafico en pantalla
    panelBarrasBuddy.setLayout(new BorderLayout());
    panelBarrasBuddy.add(chartPanelAjustes,BorderLayout.NORTH);
}

```

- **DibujaLista()**. Los rectángulos representan los espacios, su color y tamaño se ajustan según el tipo (libre u ocupado) y el tamaño total. El contorno de la memoria se colorea en verde o rojo según si se logró la última entrada. Se muestra información para los espacios ocupados, como nombre del proceso, tamaño y desperdicio. Referencias a elementos externos no proporcionados.



```

public static void dibujaLista (ArrayList <Espacio> LAjuste, Graphics papel){
    int x = 5;
    int y = 5;
    int width = 0;
    //Se pinta el controrno de la memoria de verde si es que logro entrar y de rojo si es que no lo hizo
    if(compruebaUltimaEntrada(LAjuste,papel)){
        papel.setColor(Color.green);
    }else{
        papel.setColor(Color.red);
    }
    papel.fillRect(0, 0, 1420, 130);
    papel.setColor(Color.white);
    papel.fillRect(5, 5, 1410, 120);
    Font font = new Font("Arial",Font.BOLD,14);
    papel.setFont(font);
    for (int i = 0; i < LAjuste.size(); i++) {
        //En caso de ser el buddy se hace regla de 3 con 1024 kb
        font = new Font("Arial",Font.BOLD,14);
        papel.setFont(font);
        width = (int)((float)i*1410*(float)LAjuste.get(i).tamano)/(float)(Sinc.tamanoBuddy*1024)); //Regla de 3
        if (LAjuste.get(i).tipo == 1) { //Libre
            papel.setColor(Color.LIGHT_GRAY);
            papel.drawRect(x, y, width, 120);
            papel.drawString(Integer.toString(LAjuste.get(i).tamano), x-5+width/2, y+60);
        }
        if (LAjuste.get(i).tipo == 2) { //Ocupado
            papel.setColor(new Color(LAjuste.get(i).color[0], LAjuste.get(i).color[1], LAjuste.get(i).color[2]));
            papel.fillRect(x, y, width, 120);
            font = new Font("Arial",Font.BOLD,12);
            papel.setFont(font);
            papel.setColor(Color.black);
            papel.drawString("P:"+LAjuste.get(i).nombre, x-5+width/2, y+50);
            papel.drawString("T:"+Integer.toString(LAjuste.get(i).tamano), x-5+width/2-5, y+70);
            papel.drawString("D:"+Integer.toString(LAjuste.get(i).desperdicio), x-5+width/2-5, y+90);
        }
        x+=width;
    }
}

```

- **BtnStartActionPerformed()**. Evento que se dispara al hacer clic en un botón llamado "Start". Al hacer clic, se invoca la función "start()" y se actualiza gráficamente una lista de espacios de memoria tipo "Buddy" en un panel específico. La variable "bandera" se utiliza para controlar la actualización continua, evitando un bucle infinito. La función "dibujaLista" se encarga de visualizar la lista de espacios de memoria en el panel correspondiente.



```

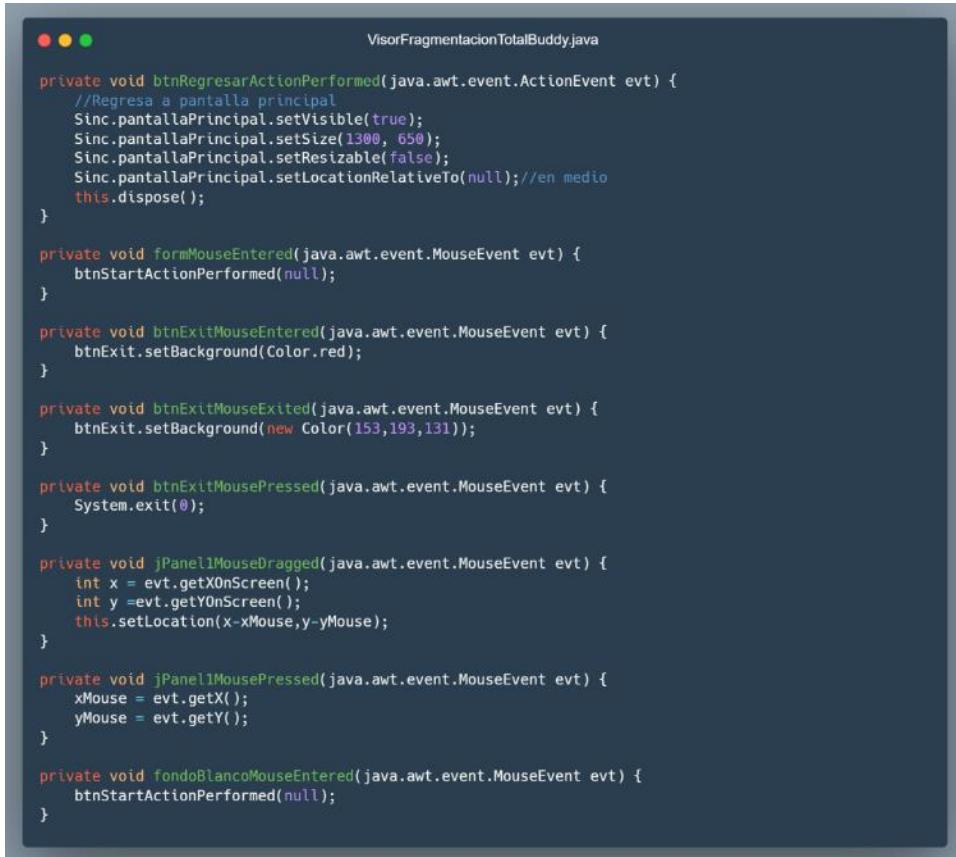
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    start();
}

private void btnStartActionPerformed(java.awt.event.ActionEvent evt) {
    start();
    //Funcion que llama para actualizar el dibujo de la memoria
    dibujaLista(Sinc.LBuddy, panelBuddy.getGraphics());
    if(!bandera){
        bandera = true;
        btnStartActionPerformed(evt);
    }
}

```

- **Botones ()**. La función **btnRegresarActionPerformed** oculta la ventana actual y muestra la pantalla principal al hacer clic en el botón "Regresar". La función **formMouseEntered** invoca automáticamente **btnStartActionPerformed** cuando el ratón entra en la interfaz. Las funciones relacionadas con el botón "Exit" cambian el color del fondo al entrar o salir, y cierran la aplicación al ser presionado. Además, se permite arrastrar la ventana al mantener presionado el botón del ratón en ciertas áreas.

La función **fondoBlancoMouseEntered** también activa **btnStartActionPerformed** al entrar el ratón en una región específica llamada "fondoBlanco".



```

private void btnRegresarActionPerformed(java.awt.event.ActionEvent evt) {
    //Regresa a pantalla principal
    Sinc.pantallaPrincipal.setVisible(true);
    Sinc.pantallaPrincipal.setSize(1300, 650);
    Sinc.pantallaPrincipal.setResizable(false);
    Sinc.pantallaPrincipal.setLocationRelativeTo(null); //en medio
    this.dispose();
}

private void formMouseEntered(java.awt.event.MouseEvent evt) {
    btnStartActionPerformed(null);
}

private void btnExitMouseEntered(java.awt.event.MouseEvent evt) {
    btnExit.setBackground(Color.red);
}

private void btnExitMouseExited(java.awt.event.MouseEvent evt) {
    btnExit.setBackground(new Color(153,193,131));
}

private void btnExitMousePressed(java.awt.event.MouseEvent evt) {
    System.exit(0);
}

private void jPanel1MouseDragged(java.awt.event.MouseEvent evt) {
    int x = evt.getXOnScreen();
    int y = evt.getYOnScreen();
    this.setLocation(x-xMouse,y-yMouse);
}

private void jPanel1MousePressed(java.awt.event.MouseEvent evt) {
    xMouse = evt.getX();
    yMouse = evt.getY();
}

private void fondoBlancoMouseEntered(java.awt.event.MouseEvent evt) {
    btnStartActionPerformed(null);
}

```

- **actualizaTabla_compLiberar()**. Refleja en una interfaz gráfica una tabla, utilizando un modelo de tabla (**DefaultTableModel**). Primero, obtiene el modelo y lo limpia, eliminando todas las filas existentes. Luego, recorre una lista de registros, representando procesos, y agrega filas al modelo si el proceso aún está activo, basándose en la función **compLiberarProceso**. En resumen, mantiene la tabla actualizada excluyendo procesos finalizados.

```

public void actualizaTabla(){
    DefaultTableModel dtm;
    //Crea un objeto para la fila de la tabla
    Object[] fila = new Object[2];
    //Obtiene el modelo de la tabla
    dtm = (DefaultTableModel) tablaHistorial.getModel();
    dtm.getDataVector().removeAllElements(); //Limpia la tabla
    dtm.fireTableDataChanged();
    //Recorre los registros de la tabla
    for (int i = 0; i < Sinc.LTabla.size(); i++) {
        //Llena el objeto fila con lo correspondiente al registro
        fila[0] = Sinc.LTabla.get(i).nombre;
        fila[1] = Sinc.LTabla.get(i).tamanyos;
        if(compliberarProceso(fila[0].toString())){ //Flujo normal
            dtm.addRow(fila);
        }
    }
}

public boolean compliberarProceso(String proceso){
    //retorna true si el proceso ya salio
    for (int i = 0; i < Sinc.LTabla.size(); i++) {
        if("Salida".equals(Sinc.LTabla.get(i).tipoES) && Sinc.LTabla.get(i).nombre.equals(proceso)){
            return false;
        }
    }
    return true;
}

```

- o excelActionPerformed (). Esta función se encarga de instanciar un objeto de la clase Exportar Excel, mandando a llamar un método donde se recibe la tabla de historial de pantalla inicial, hay que tomar en cuenta que se generara el documento de acuerdo a lo que se tenga en ese momento en la tabla de acuerdo a la filtración.

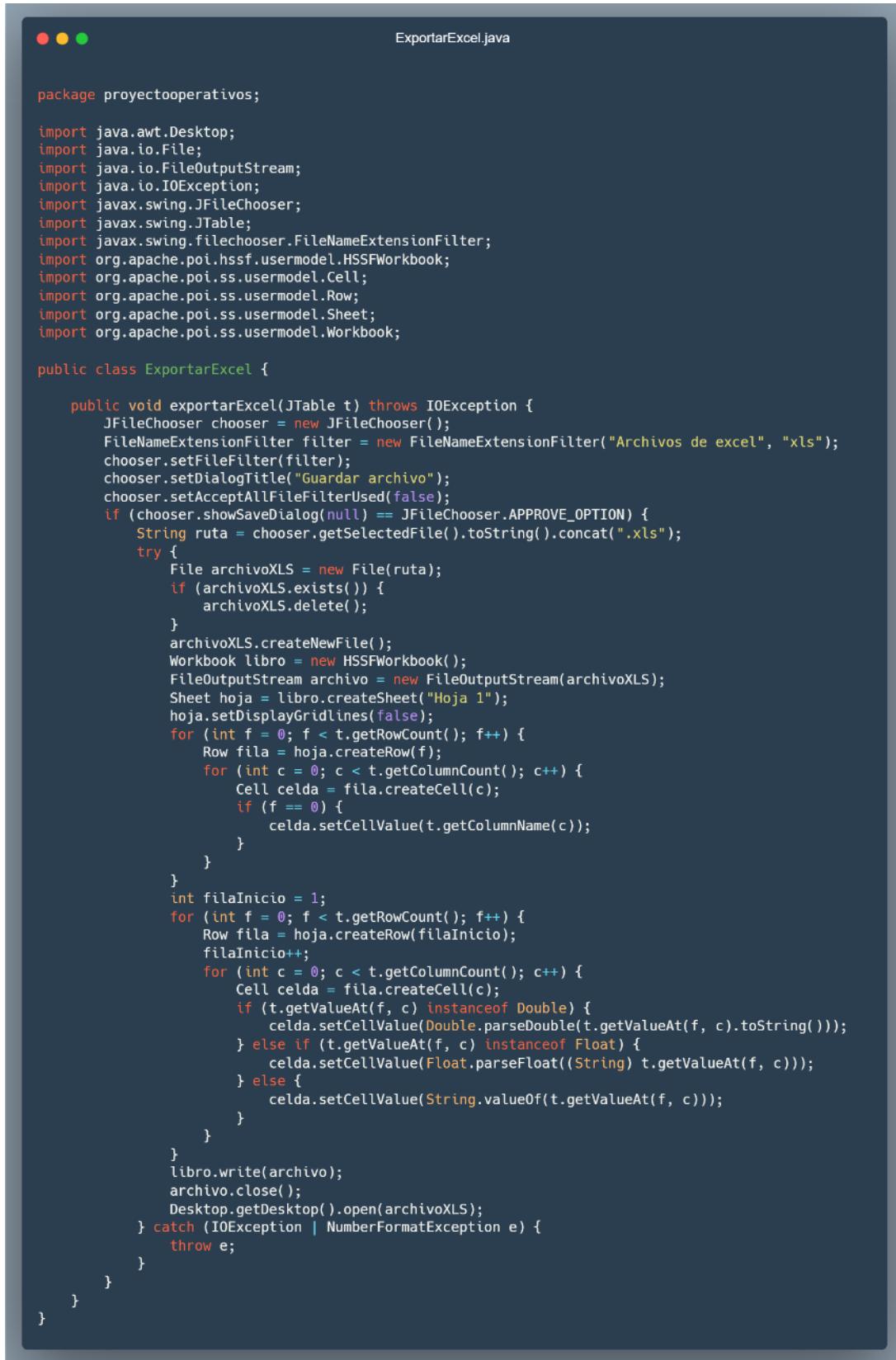
```

private void excelActionPerformed(java.awt.event.ActionEvent evt) {
    ExportarExcel obj;

    try {
        obj = new ExportarExcel();
        obj.exportarExcel(tablaHistorial);
    } catch (IOException ex) {
        System.out.println("Error: " + ex);
    }
}

```

- ExportarExcel
 - o exportarExcelJTable(): genera un Excel con ayuda de la librería poi-3.7, recibiendo como parámetro los datos de la tabla.



```

package proyectooperativos;

import java.awt.Desktop;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import javax.swing.JFileChooser;
import javax.swing.JTable;
import javax.swing.filechooser.FileNameExtensionFilter;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;

public class ExportarExcel {

    public void exportarExcel(JTable t) throws IOException {
        JFileChooser chooser = new JFileChooser();
        FileNameExtensionFilter filter = new FileNameExtensionFilter("Archivos de excel", "xls");
        chooser.setFileFilter(filter);
        chooser.setDialogTitle("Guardar archivo");
        chooser.setAcceptAllFileFilterUsed(false);
        if (chooser.showSaveDialog(null) == JFileChooser.APPROVE_OPTION) {
            String ruta = chooser.getSelectedFile().toString().concat(".xls");
            try {
                File archivoXLS = new File(ruta);
                if (archivoXLS.exists()) {
                    archivoXLS.delete();
                }
                archivoXLS.createNewFile();
                Workbook libro = new HSSFWorkbook();
                FileOutputStream archivo = new FileOutputStream(archivoXLS);
                Sheet hoja = libro.createSheet("Hoja 1");
                hoja.setDisplayGridlines(false);
                for (int f = 0; f < t.getRowCount(); f++) {
                    Row fila = hoja.createRow(f);
                    for (int c = 0; c < t.getColumnCount(); c++) {
                        Cell celda = fila.createCell(c);
                        if (f == 0) {
                            celda.setCellValue(t.getColumnName(c));
                        }
                    }
                }
                int filaInicio = 1;
                for (int f = 0; f < t.getRowCount(); f++) {
                    Row fila = hoja.createRow(filaInicio);
                    filaInicio++;
                    for (int c = 0; c < t.getColumnCount(); c++) {
                        Cell celda = fila.createCell(c);
                        if (t.getValueAt(f, c) instanceof Double) {
                            celda.setCellValue(Double.parseDouble(t.getValueAt(f, c).toString()));
                        } else if (t.getValueAt(f, c) instanceof Float) {
                            celda.setCellValue(Float.parseFloat((String) t.getValueAt(f, c)));
                        } else {
                            celda.setCellValue(String.valueOf(t.getValueAt(f, c)));
                        }
                    }
                }
                libro.write(archivo);
                archivo.close();
                Desktop.getDesktop().open(archivoXLS);
            } catch (IOException | NumberFormatException e) {
                throw e;
            }
        }
    }
}

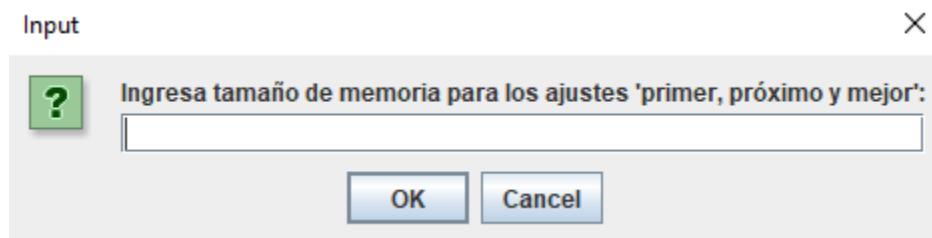
```

EJECUCIÓN

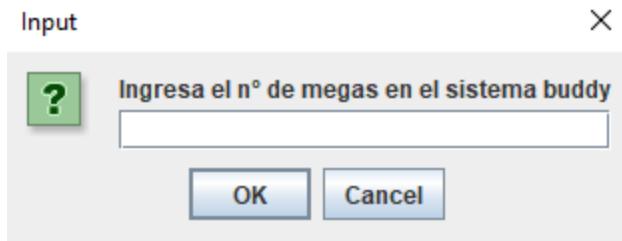
Se mostrará una portada con el título del proyecto y los datos de los integrantes del equipo comenzará después de dar clic en el botón de 'Inicio'



Después de presionar el botón de inicio se le preguntara al usuario el tamaño de la memoria que quiere utilizar para los ajustes. Se debe de ingresar solo números enteros mayores de 200 ya que al no ingresar números, números menor del rango especificado o no ingresar nada, se mostrará una pantalla de error.



Tras haber ingresado el tamaño de memoria que se utilizara, se le preguntara al usuario el número de megas que desea utilizar para el sistema buddy. Se debe de ingresar solo números enteros ya que, al no ingresar números, números menor del rango especificado o no ingresar nada, se mostrará una pantalla de error.



Tras haber ingresado el tamaño de memoria y el número de megas a utilizar, se mostrará una pantalla donde podrás ingresar el tamaño del proceso a ingresar que deseas o un botón de aleatorio para que se genere el tamaño del proceso de forma aleatoria junto a un panel de colores donde el usuario podrá escoger el color que desee.

En la parte inferior de agregar proceso se muestra una tabla con el historial de procesos que fueron ingresando a memoria y al sistema buddy. En la tabla se presenta su estado de E/S si es entrada o ya fue liberado de la memoria (salida), nombre del proceso, el tamaño que se le asignó, si fue Ingresado/Bloqueado en los ajustes y en el sistema buddy.

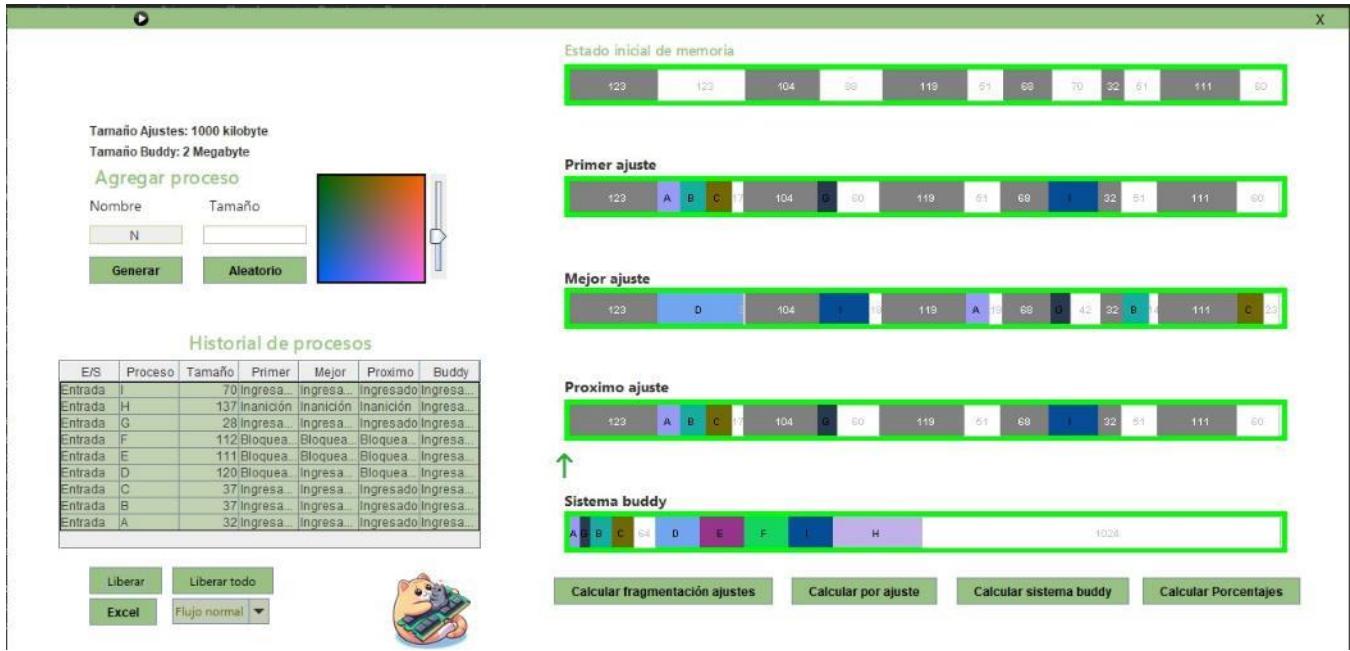
Se presentarán debajo del historial 3 botones el cual el botón de "liberar", liberara el proceso seleccionado desde la tabla. El botón "Liberar todo" liberara todos los procesos que se encuentran en la memoria. El botón "Excel" creará un archivo de Excel con el historial de todos los procesos que fueron ingresados o liberados.



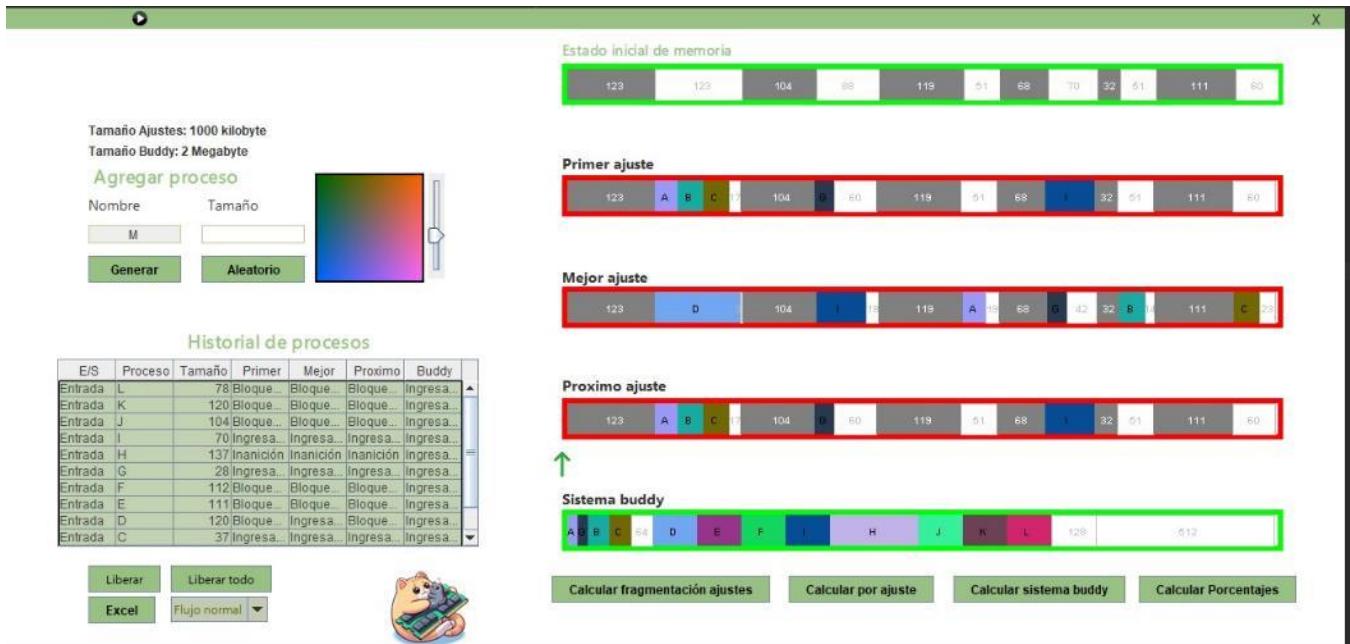
Cuando el proceso a ingresar cumple con el tamaño adecuado, cada ajuste buscará en su respectiva memoria un espacio donde pueda entrar ese proceso, si entra el proceso, la memoria se pondrá en un cuadro de color verde indicando que si entró el proceso. En el próximo ajuste contará con una flecha indicando donde se asignó el proceso para que el siguiente comience desde el último punto.

Para el sistema buddy se verá la forma en la que se va dividiendo los megas para poder ingresar el proceso que se asignó.

SISTEMAS OPERATIVOS

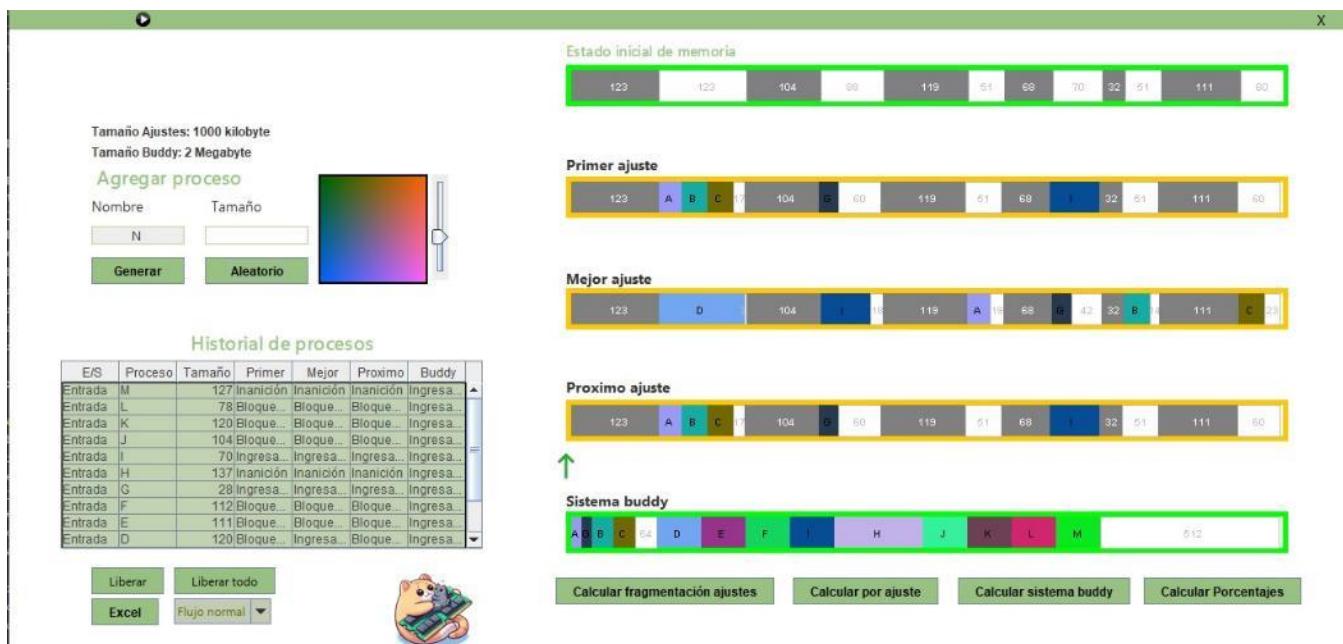


Cuando se va a ingresar un proceso de tamaño adecuado, pero no hay lugar disponible en memoria, la memoria se pondrá en un cuadro de color rojo indicando que no entro, pero como el tamaño es menor al del límite, su estado estará en bloqueado ya que si puede ingresar. En el próximo ajuste al ver que no entro el proceso, se reiniciara el puntero en 0.

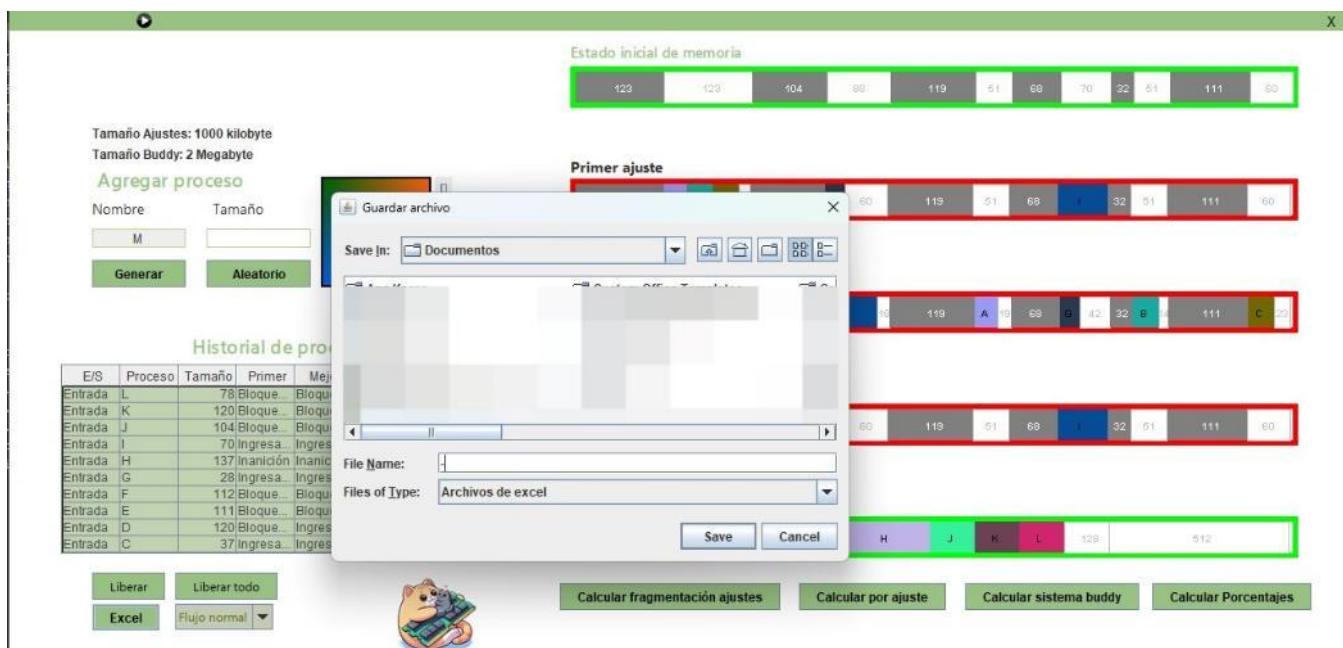


Cuando se ingresa un proceso mayor al límite de los disponibles, la memoria se pondrá en un cuadro amarillo indicando que el proceso no podrá ingresar más adelante por lo que su para los ajustes será de inanición.

SISTEMAS OPERATIVOS



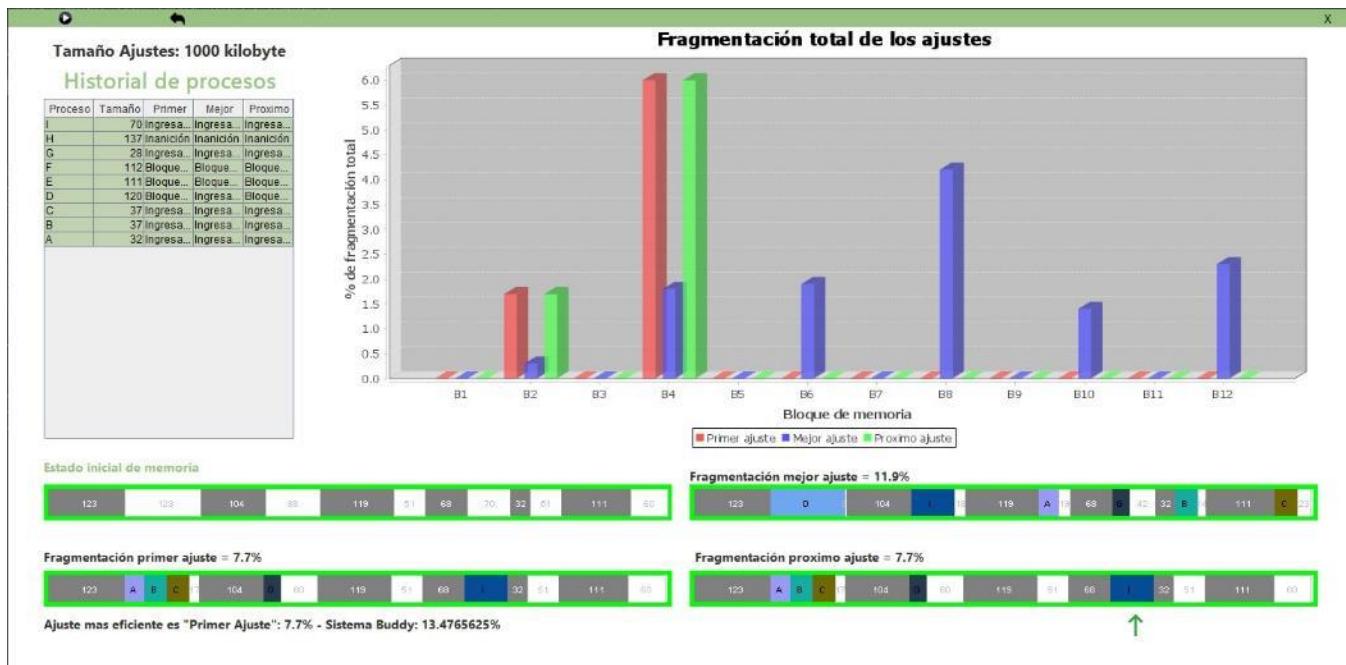
Al presionar el botón “Excel”, genera un documento con los registros de entrada de procesos durante la ejecución, o correspondiente al último ajuste efectuado en cada caso. Muestra si se trató de una entrada de proceso, o una salida, el nombre del mismo, tamaño, y si fue ingresado en la barra de cada ajuste.



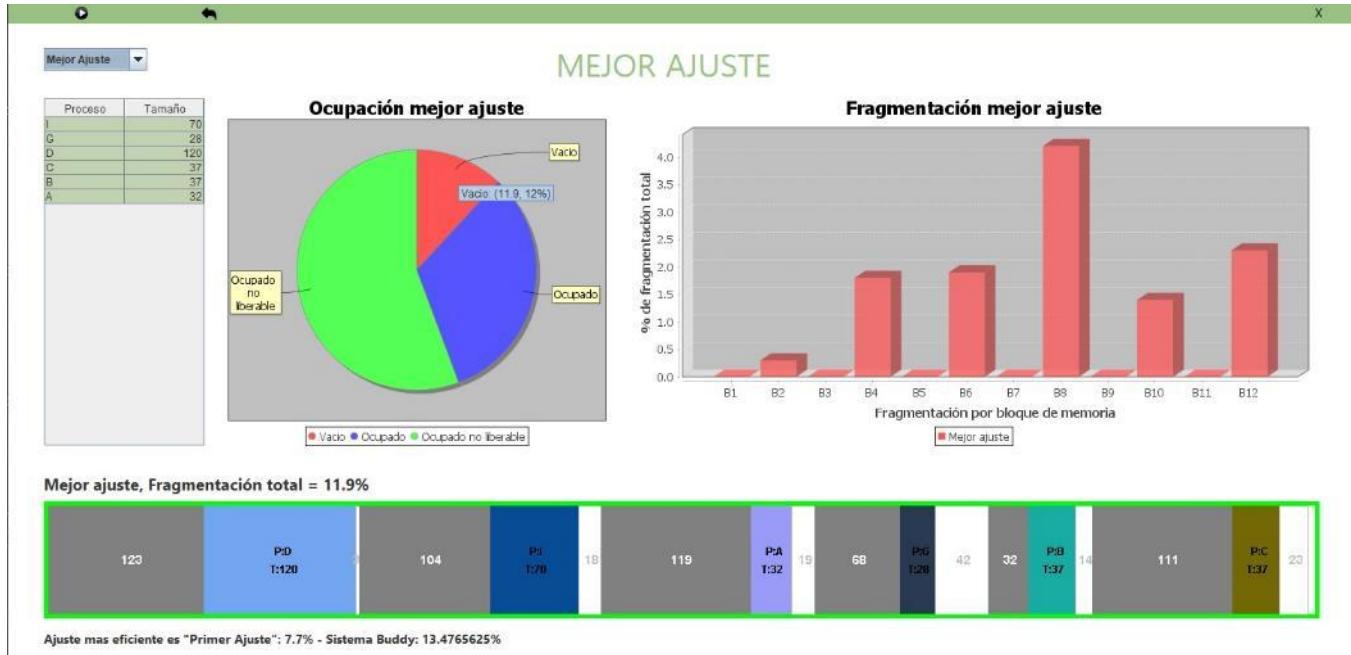
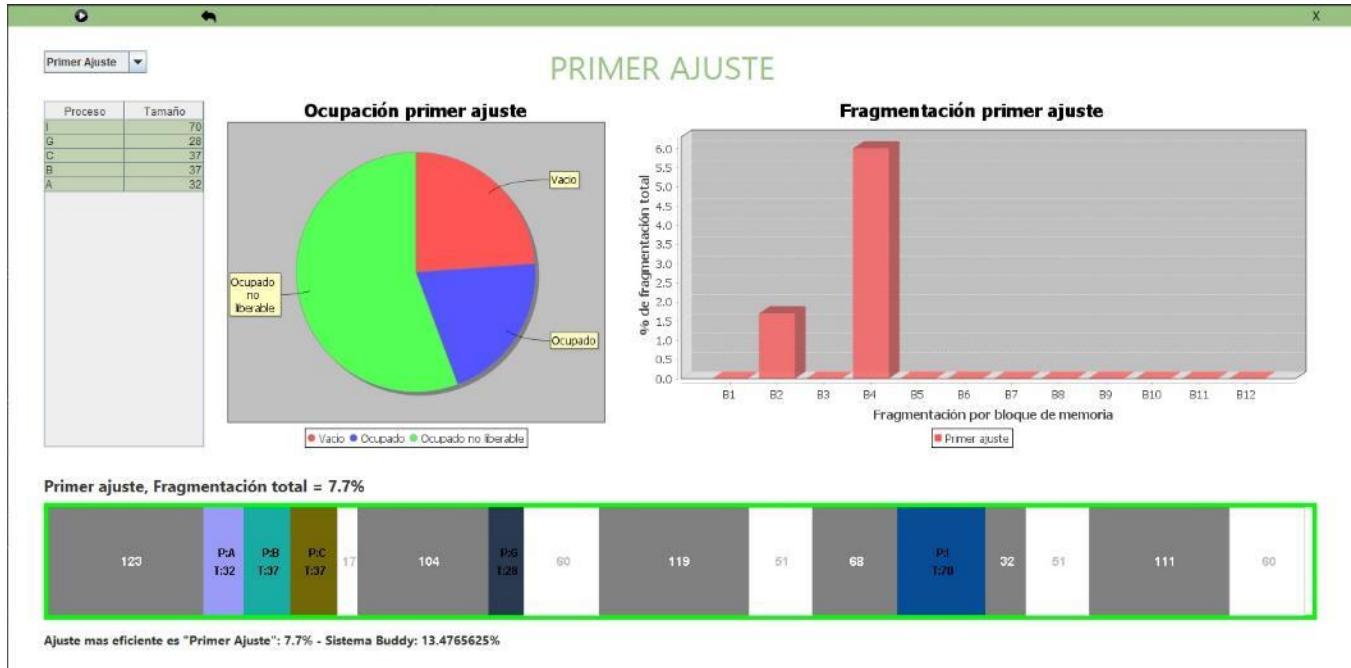
SISTEMAS OPERATIVOS

A	B	C	D	E	F	G	
1	E/S	Proceso	Tamaño	Primer	Mejor	Proximo	Buddy
2	Entrada	K	110	Inanición	Inanición	Inanición	Ingresado
3	Entrada	J	128	Inanición	Inanición	Inanición	Ingresado
4	Entrada	I	77	Bloqueado	Bloqueado	Bloqueado	Ingresado
5	Entrada	H	84	Bloqueado	Bloqueado	Bloqueado	Ingresado
6	Entrada	G	64	Bloqueado	Bloqueado	Bloqueado	Ingresado
7	Entrada	F	60	Ingresado	Ingresado	Ingresado	Ingresado
8	Entrada	E	77	Ingresado	Ingresado	Ingresado	Ingresado
9	Entrada	D	73	Ingresado	Ingresado	Ingresado	Ingresado
0	Entrada	C	57	Ingresado	Ingresado	Ingresado	Ingresado
1	Entrada	B	30	Ingresado	Ingresado	Ingresado	Ingresado
2	Entrada	A	27	Ingresado	Ingresado	Ingresado	Ingresado
3							
4							
5							
6							
7							
8							

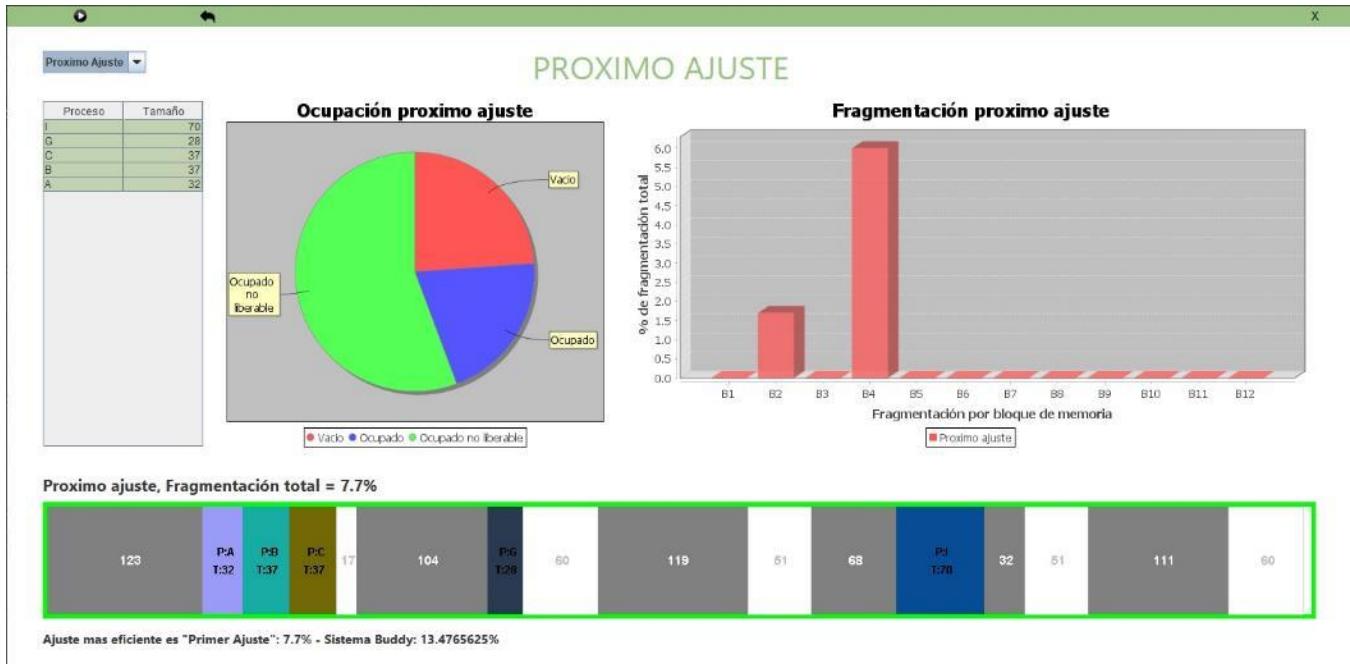
Con el botón “Calcular fragmentación ajustes” se presenta una gráfica representando los porcentajes de fragmentación de los bloques que fueron utilizados por los procesos ingresados. La gráfica muestra cada bloque de memoria del primer, mejor y próximo ajuste por lo que se compara entre los tres ajustes. Del lado izquierdo de la gráfica se muestra una tabla con el historial de los procesos, el orden en el que han entrado, su tamaño, y si entro o no a la memoria de cada ajuste. En la parte inferior se muestra los bloques completos de los ajustes con su porcentaje de fragmentación y mostrando la última asignación de proceso en un cuadro negro. Para finalizar se dará una breve información mencionando cuál de los ajustes fue más eficiente.



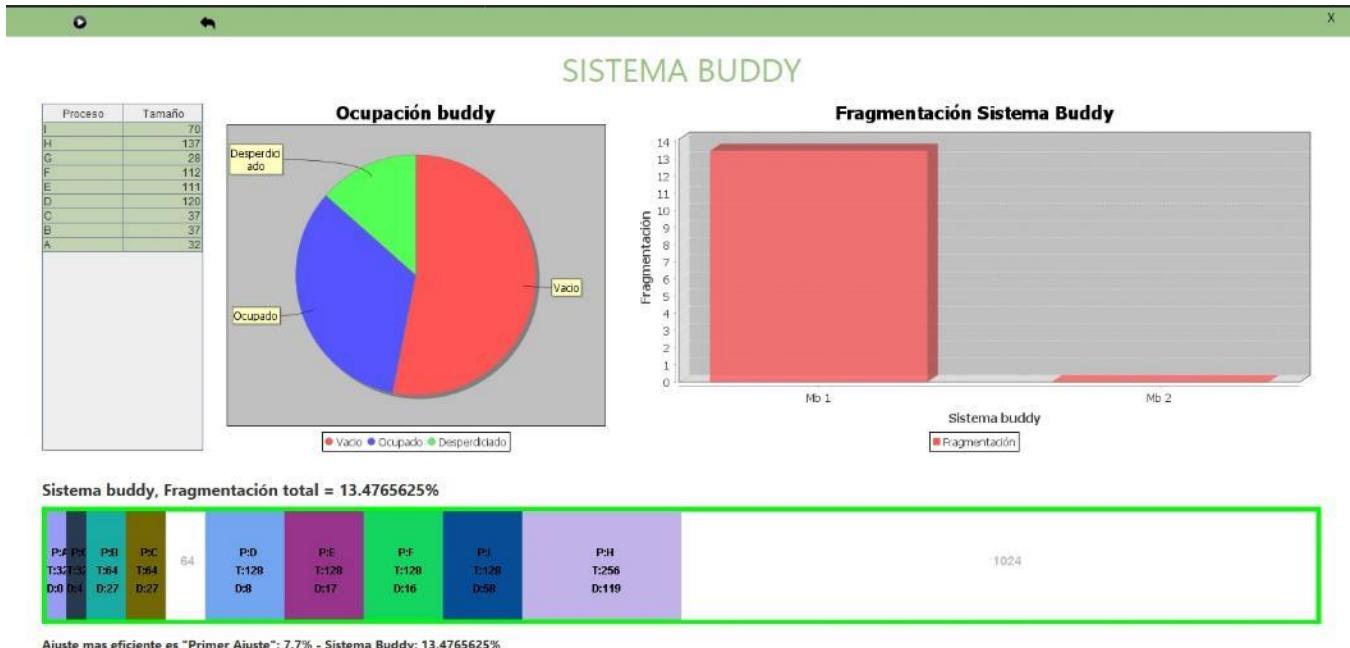
Con el botón “Calcular por ajuste” se brindará información más detallada de cada ajuste. En la parte superior izquierda tenemos un combo box que nos deja elegir el ajuste para informar con los procesos ya ingresados en orden de entrada. A continuación, una gráfica de pastel que muestra los porcentajes de los espacios ocupados no liberables, los vacíos y ocupados y del lado derecho una gráfica de barra con los porcentajes de cada bloque de la memoria. En la parte inferior se mostrará la memoria completa con los procesos ocupados mostrando su tamaño y el último en ser ingresado se mostrará con un cuadro rosa.



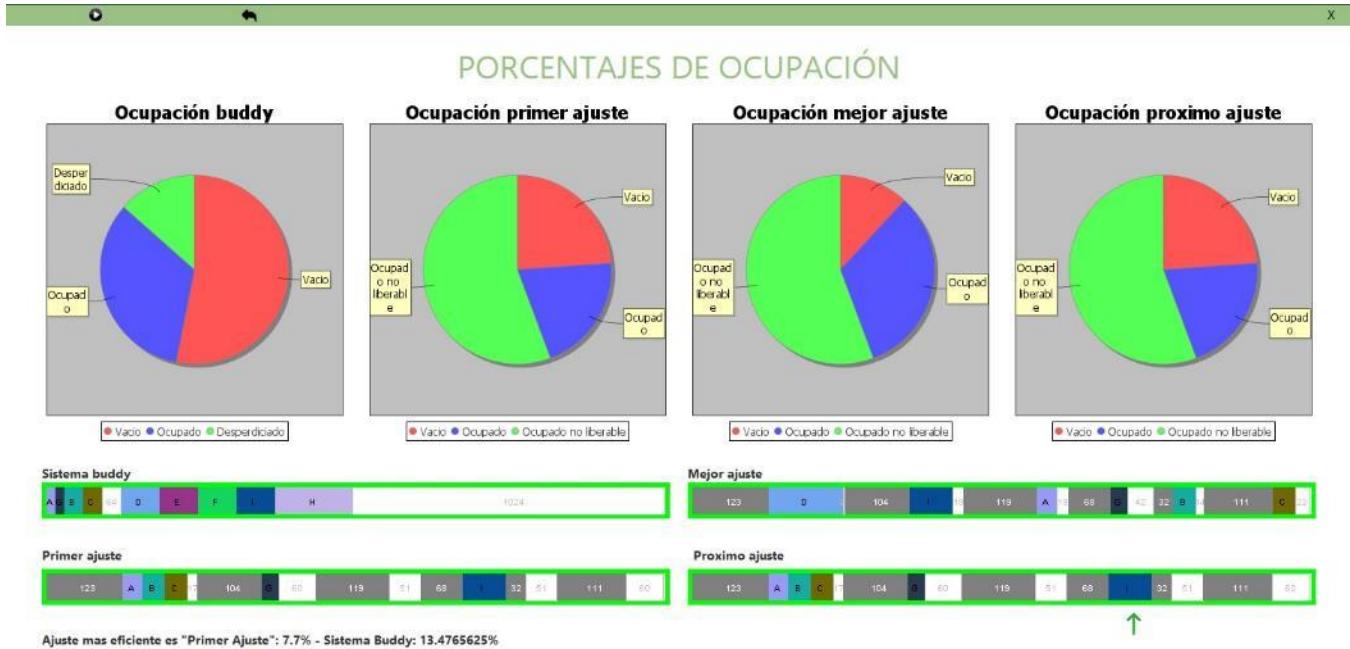
SISTEMAS OPERATIVOS



Con el botón “Calcular sistema buddy” mostrara una tabla con los procesos ya ingresados a memoria con una gráfica de pastel mostrando los porcentajes de los bloques que están vacíos, los que están ocupados y el desperdicio de los bloques que se encuentran ocupados. También una gráfica de barra con el porcentaje de fragmentación de cada mega que se asignó al inicio. En la parte inferior se muestra los bloques del sistema buddy con los procesos asignados mostrando su tamaño y el desperdicio (Lo que sobro) del bloque y la fragmentación total del sistema.



Con el botón “Calcular porcentajes” se mostrarán gráficas de pastel donde se muestra el porcentaje de ocupación total de cada ajuste realizando una comparación entre el sistema buddy y los ajustes.



ERRORES PRESENTADOS

PROBLEMA CON LA COMPRESIÓN EN EL SISTEMA BUDDY

El sistema buddy comprende una lógica diferente a la de los ajustes argumento de comprimir sus espacios de memoria, Sus espacios de memoria solo se pueden Fusionar entre espacios de memoria que fueron generados a partir del mismo bloque padre y que pertenecen al mismo megabyte inicial.

Para aprovechar las funcionalidades creadas para los ajustes se desarrolló una lógica que permite al sistema buddy El historial de bloques de los cuales se deriva su bloque padre. De esta manera fue posible identificar cuál bloque de memoria fusionar con cuál otro y mantener las características que definen un sistema Buddy.

PROBLEMAS EN EL DISEÑO DE LA INTERFAZ

Existió un problema con los elementos de la paleta, puesto que llegaron a moverse o dejar de responder cuando se mandaba a llamar a la función 'dibujaLista()'. Este se solucionó al tener una actualización constante de las listas en pantalla

RESULTADOS Y CONCLUSIONES

Durante el proceso de la realización del proyecto pudimos obtener una mejor comparación entre las asignaciones de memoria y una mejor visualización del sistema buddy. Demostramos con una interfaz las diferencias entre las asignaciones y pudimos observar las diferentes formas en las que se asignan los procesos en la memoria con su ubicación considerando también la liberación de los procesos. Con el sistema buddy pudimos tener una mejor visibilidad en la asignación y las formas en que se va segmentando la memoria para poder asignar el tamaño del proceso en el lugar indicado y como se van fusionando los espacios que son liberados. Gracias a esto, se logró generar gráficas mostrando el porcentaje de fragmentación que hay en los bloques ocupados y pudiendo compararlo con los demás.

En conclusión, la implementación de los algoritmos de asignación de memoria (primer ajuste, próximo ajuste y mejor ajuste) y del sistema de asignación de memoria Buddy representa un enfoque integral y eficiente para abordar los desafíos inherentes a la gestión de la memoria en sistemas operativos. La estructura y las estrategias específicas empleadas en cada algoritmo permiten una adaptabilidad dinámica a cambios en la demanda de memoria, minimizando la fragmentación y optimizando el rendimiento del sistema. Asimismo, el sistema Buddy, con su estructura jerárquica y algoritmos eficientes, demuestra ser una solución efectiva para la asignación y liberación de bloques de memoria, contribuyendo a la optimización de recursos y a la mejora general de la eficiencia del sistema. Estos enfoques de asignación de memoria ofrecen soluciones robustas y versátiles, dando una base para la gestión eficiente de la memoria en entornos operativos Java.

BIBLIOGRAFÍA

Silberschatz, A., I.Galvin, P. & II.Escalona, R. Sistemas operativos (5^a ed.). México: Pearson Educación.

Silberschatz, A., I.Galvin, P. & II.Escalona, R. Fundamentos de sistemas operativos (7ma. Ed.). (2006). Carmelo Sánchez González Com puesto por: Vuelapluma, S. L. Im preso en: C ofas. S. A.

Tanenbaum, A. & I.Palmas Velásco, O. Sistemas operativos modernos. México: Prentice-Hall.

William Stallings, Sistemas Operativos (5^a ed.). México: Prentice-Hall.

ANEXO DE RESPONSABILIDADES

Actividad asignada	Actividad realizada	Responsable	% de cumplimiento
Lógica de próximo ajuste	Algoritmo del próximo ajuste en java	Saúl Cervantes Candia	100%
Documentación	Análisis, Diseño, implementación de algoritmo, ejecución, resultado y conclusión, presentación	Saúl Cervantes Candia	100%
Lógica de primer ajuste Validaciones Excel	Algoritmo del primer ajuste en java Validar entradas de la memoria	Ana Karen Cuenca Esquivel	100%
Diseño del programa	Portada, diseño de cada pestaña, logo y lógica de barra de navegación	Ana Karen Cuenca Esquivel	100%
Documentación	Base del documento y presentación Capturas de códigos	Ana Karen Cuenca Esquivel	100%
Lógica del mejor ajuste	Lógica mejor ajuste en java.	Leandro Sosa Michell Alexa	100%
Documentación	Marco teórico, Implementación de algoritmo, presentación	Leandro Sosa Michell Alexa	100%
Lógica base del programa	Creación de la estructura que rige el programa	Rivera Carreon Brian Issai	100%
Lógica archivo Sinc	Implementación de los modelos sinc	Rivera Carreon Brian Issai	100%

SISTEMAS OPERATIVOS

Algoritmos fusión de memoria	Creación de algoritmos de fusion para ajustes y buddy	Rivera Carreon Brian Issai	100%
Logica del Sistema buddy	Generacion de los algoritmos de insercion en sistema buddy	Rivera Carreon Brian Issai	1000%
Implementacion de graficas	Generación de lógica para la presentación de gráficas	Rivera Carreon Brian Issai	100%