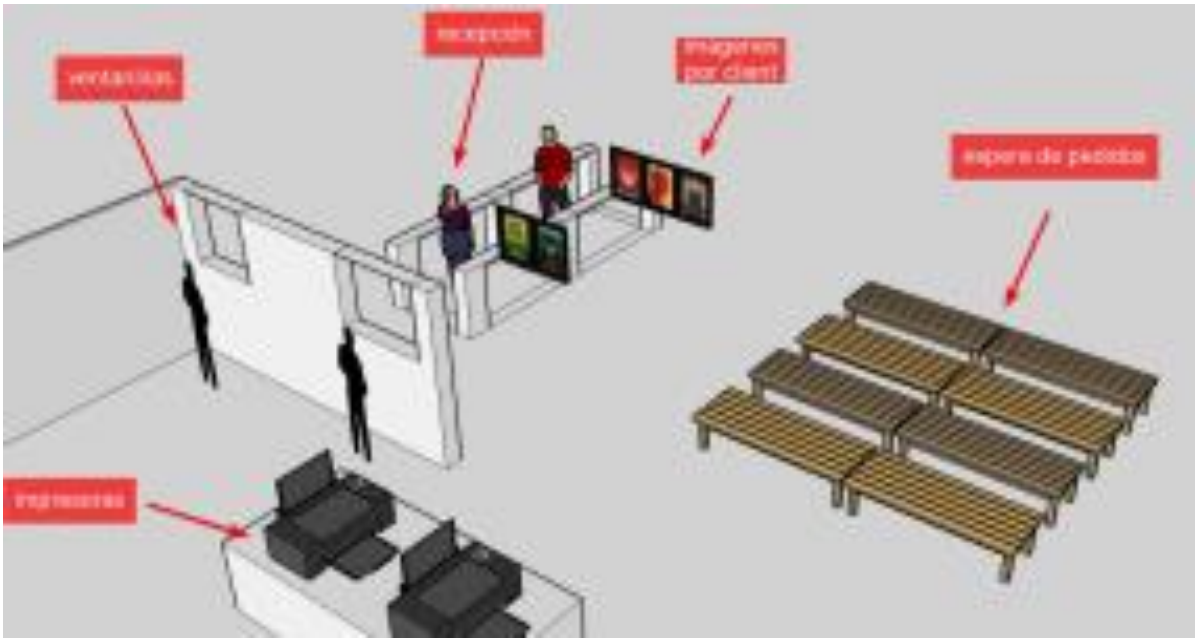


Estructuras de datos sección C
Laboratorio 2022.

Alexander Mejia

Menú de la aplicación

Para realizar el menú debemos de utilizar la librería scanner que nos permite ingresar valores atravez de la consola, y con auda de la setiquetas println y la función switch podemos crear un menú muy eficiente.

```
String el = "";
System.out.println("-------MENU--------");
System.out.println("1. Parametros Iniciales");
System.out.println("2. Ejecutar Pasos");
System.out.println("3. Estado de memoria de las estructuras");
System.out.println("4. Reportes ");
System.out.println("5. Acerca de");
System.out.println("6. Salir");
el = entrada.nextLine();
boolean valido = Val_Numero(el);
if (valido == true) {
    int numero = Integer.parseInt(el);
    switch (numero) {
        case 1:
            System.out.println("Selecciono 1 esta en la funcion Para
            String e2 = "";
            boolean salida2 = false;
            boolean Vala = false;
            boolean Valb = false;
```

Debemos de seleccionar la opción 1, ahí podremos realizar una carga masiva de datos.

C:\Users\braya\OneDrive\Documentos\NetBeansProjects\Extraccion_Json\src\Files_Json\Extaccion.json

C:\Users\braya\Downloads\Entrada.json

Cualquier ruta con un archivo json con la estructura correcta podrá ser cargado en la carga masiva. Adicional Se le agregaran algunos clientes randoms que se auto generan con un algoritmo de ingreso random. De haber ingresado la ruta correcta se nos presentara una pantalla como a continuación:

Funcion Random

Para generar la funcion random debemos de utilizar arreglos y el uso de la librería random que nos permite seleccionar numeros aleatorios, con el fin de obeteer un nombre y un apellido de dos ectores diferentes, y un valor.

```
public static void IngresarRandoms(Lista_Recepcion lstR){
    Lista_Recepcion LR = lstR;
    Random random = new Random();
    String[] Nombres = {"Liam","Noah","Oliver","Elijah","Olivia","Emma","Ava","Charlotte","
    String[] Apellidos = {"Mejia","Barrientos","Garcia","Cabrera","Gramajo","Arevalo","Caba
    int Cant_Imagenes = random.nextInt(4);
    int Cant_Clientes = random.nextInt(3);
    String N = "";
    String A = "";
    String NCompleto = "";
    String id = "";
    int idnumero = 1999;

    String ing_Imagen_Color = "";
    String ing_Imagen_BW = "";


    for(int i=0;i<=Cant_Clientes;i++){
        idnumero = idnumero + i;
        int indexN = random.nextInt(9);
        N = Nombres[indexN];
        int indexA = random.nextInt(9);
        A = Apellidos[indexA];
        int img_Color = 0;
        int img_BW = 0;

        NCompleto = N + " " + A;
        System.out.println(NCompleto);
        for (int j = 0; j <= Cant_Imagenes; j++) {
            int img = random.nextInt(2);
```

Se puede ver como se generan los randoms y se incorporan al final de la cola en caso no logres ingresar la ruta solo se ingresaran los randoms al principio de la estructura Cola recepción.

```
  -_
-->Lafter Orellana-->Noah Gramajo-->Emma Gramajo-->Luiz Higueros-->Juan Perez-->Andres
--------Sub-Menu--------
1. Carga Masiva de Clientes
2. Cantidad de Ventanillas
```

El paso siguiente es indicar el numero de ventanillas donde este puede ser de 1 a infinito se recomienda utilizar entre 1 a 10.

```
--------Sub-Menu--------
1. Carga Masiva de Clientes
2. Cantidad de Ventanillas
2
Selecciono 1.2 Esta dentro de Cantidad de Ventanillas
Ingrese el numero de Ventanillas
4
--->Ventanilla 3--->Ventanilla 2--->Ventanilla 1-->Ventanilla 4
-------MENU--------
```

Como podemos observar se nos muestra una impresión de las ventanillas que se crearon.

Ahora bien lo siguiente seria ejecutar los pasos, para eso debemos seleccionar la opción 2 de menú:

```
-------MENU--------
1. Parametros Iniciales
2. Ejecutar Pasos
3. Estado de memoria de las estructuras
4. Reportes
5. Acerca de
6. Salir
```

Esta opción nos genera un submenú que nos permite realizar paso a paso para ver como se va estructurando o bien realizar todos de una vez.

Estructuras utilizadas

Estas clases son objetos que nos perimte almacenar los atributos con variables String, int, Object en si podemos guradar un objeto dentro de otro para mayor facilidad.

Clase Imagen:

Los atributos de la imagen son:

String tipo

Int impresora : indica a que impresora se le enviara este se lo proporciona el objeto ventanilla

```
 */
public class Imagen {
    String id ;
    String tipo;
    int impresora;

    public Imagen(String tipo, String id,int impresora) {
        this.id = id;
        this.tipo = tipo;
        this.impresora = impresora;
    }

    public String getId() {
        return id;
    }



    public String getTipo() {
        return tipo;
    }
```

Clase Cliente:

Los atributos del cliente son:

String Nombre

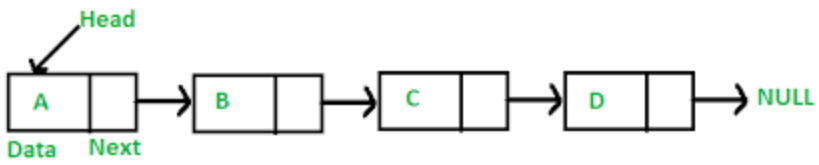
String ID

String Imágenes Color

String Imágenes Blanco y negro

```
public class Cliente {
    String id ;
    String Nombre;
    String Img_C;
    String Img_BW;

    public Cliente(String id, String Nombre, String Img_C, String Img_BW) {
        this.id = id;
        this.Nombre = Nombre;
        this.Img_C = Img_C;
        this.Img_BW = Img_BW;
    }

    public String getId() {
        return id;
    }

    public String getNombre() {
        return Nombre;
    }
```

Estos objetos se almacenan dentro de nodos y ese conjunto de nodos forman la llistas



1. Cola de recepción

Para generar estas estructuras se necesito de una clase nodo_Recepcion y una lista que almacena los datos de forma dinámica y secuencial.

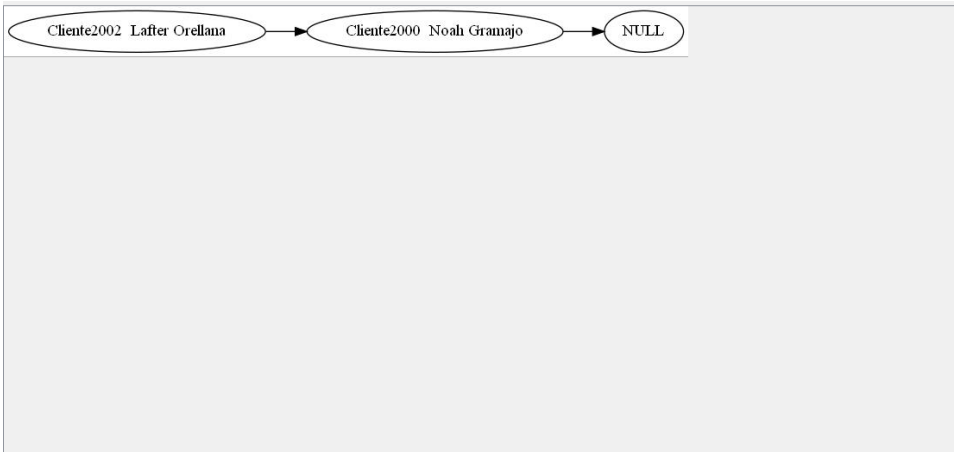
Los nodos_Recepcion almacenan un objeto cliente en este caso se utilizó un casteo de objetos ya que al obtener el atributo de un nodo solo nos entregaba la dirección del objeto y al castearlo nos permite acceder a los atributos del cliente.

A continuación una estructura de un nodo en este caso Nodo_Recepcion

```
*/
public class Nodo_Recepcion {
   Object elemento;
   Nodo_Recepcion siguiente;

    public Nodo_Recepcion(Object elemento) {
        this.elemento = elemento;
        this.siguiente = null;
    }
    public Object getElemento() {
        return elemento;
    }
    public Nodo_Recepcion getSiguiente() {
        return siguiente;
    }
    public void LinkSiguiente(Nodo_Recepcion siguiente) {
        this.siguiente = siguiente;
    }
```

En esta estructura es donde se ingresan los clientes al llegar a la empresa, en el menú deberá existir una opción que permita la carga masiva de los clientes que ingresarán a la empresa. La carga masiva se realizará por medio de un archivo con extensión (json) que contendrá la siguiente estructura:

```
{
"Cliente1": {
"id_cliente": "1",
"nombre_cliente": "Andres Lopez",
"img_color": "3",
"img_bw": "2"
},
"Cliente2": {
"id_cliente": "2",
"nombre_cliente": "Juan Perez",
"img_color": "3",
"img_bw": "0"
},
"Cliente3": {
"id_cliente": "3",
"nombre_cliente": "Luiz Higueros",
"img_color": "2",
"img_bw": "1"
}
}
```



Los clientes ingresados anteriormente estarán en el estado inicial de la aplicación (Revisar las imágenes de pasos adjuntas), adicionalmente en cada paso se generan aleatoriamente más clientes con las siguientes características:

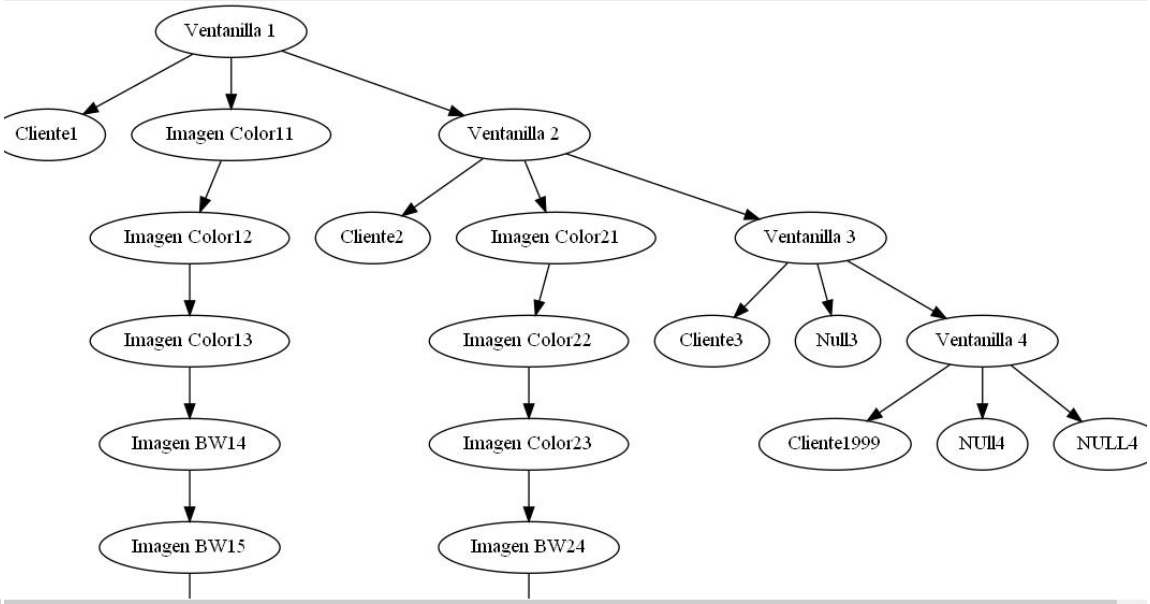
- Cantidad de clientes aleatorios: entre 0 y 3
- Cantidad de imágenes por cada cliente: entre 0 y 4
- Nombre aleatorio: pueden utilizar vectores de nombres y apellidos para generarlos.

2. Lista de ventanillas (simple)

Se implementará una lista simplemente enlazada para el número de ventanillas que estarán en el proceso de simulación, cada ventanilla contará con una pila que se utilizará para recibir las imágenes que un cliente desea imprimir.

A continuación mostraremos la estructura de como debe de ser el nodo de las ventanillas ya que esta necesita una mayor cantidad de punteros y adicionalmente se le conecta una lista enlazada para poder obtener una lista de listas y se deja inicializada para facilidad de que llevar el objeto no se dificulte ni se pierda entre las las líneas de código

```java
public class Nodo_Ventanillas{
        Object elemento;
        int id;
        Nodo_Recepcion arriba;
        Nodo_Ventanillas siguiente;
        Lista_Imagenes_Ventanilla lstImgVent = new Lista_Imagenes_Ventanilla();

    public Nodo_Ventanillas(Object elemento,int id) {
        this.elemento = elemento;
        this.arriba = null;
        this.siguiente = null;
        this.lstImgVent = lstImgVent;
        this.id = id;
    }

    public Object getElemento() {
        return elemento;
    }

    public Nodo_Ventanillas getSiguiente() {
        return siguiente;
    }

    public void linkSiguiente(Nodo_Ventanillas siguiente) {
        this.siguiente = siguiente;
    }

    public Nodo_Recepcion getArriba() {
        return arriba;
    }
```



}}

2.1. Pila de imágenes

El funcionamiento de la pila de cada ventanilla es el siguiente:

● En cada paso que el cliente está en ventanilla se inserta una imagen a la pila,

con todas las especificaciones de la impresión.

● La pila se vacía cuando el cliente sale de la ventanilla.

Para las imágenes se utilizo una lista igual a la de recepción solo que maneja nodos de tipo imagen es decir que en vez de castear un cliente castearemos un objeto imagen

```java
 */
public class Nodo_Imagenes {
     Object elemento;



     Nodo_Imagenes siguiente;

    public Nodo_Imagenes(Object elemento) {
        this.elemento = elemento;

        this.siguiente = null;
    }
    public Object getElemento() {
        return elemento;
    }
    public Nodo_Imagenes getSiguiente() {
        return siguiente;
    }
    public void linkSiguiente(Nodo_Imagenes siguiente) {
        this.siguiente = siguiente;
    }
}
```

3. Lista de clientes atendidos (simplemente enlazada)

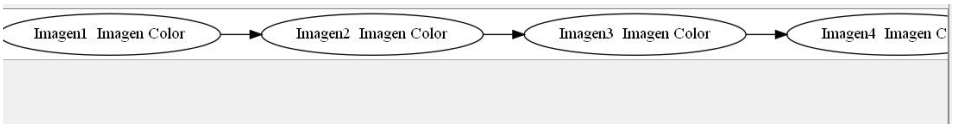
Se guardará la información de todos los clientes que son atendidos por todas las ventanillas habilitadas durante el proceso de simulación.

En cada nodo de la lista se almacenará la siguiente información:

- Nombre del cliente
- Ventanilla que lo atendió
- Número de imágenes impresas
- Cantidad total de pasos en el sistema

4. Cola de impresión

Cada impresora maneja una cola de impresión, al momento que un cliente termina de realizar su orden de impresión, las imágenes son enviadas desde la pila de la ventanilla hacia las distintas colas de acuerdo a las especificaciones solicitadas (imagen a color o en blanco y nego), se debe implementar un mecanismo de clasificación de imágenes que permitirá que en ningún momento una imagen sea enviada a la impresora incorrecta, de tal forma que se cumplan las especificaciones solicitadas por el cliente.



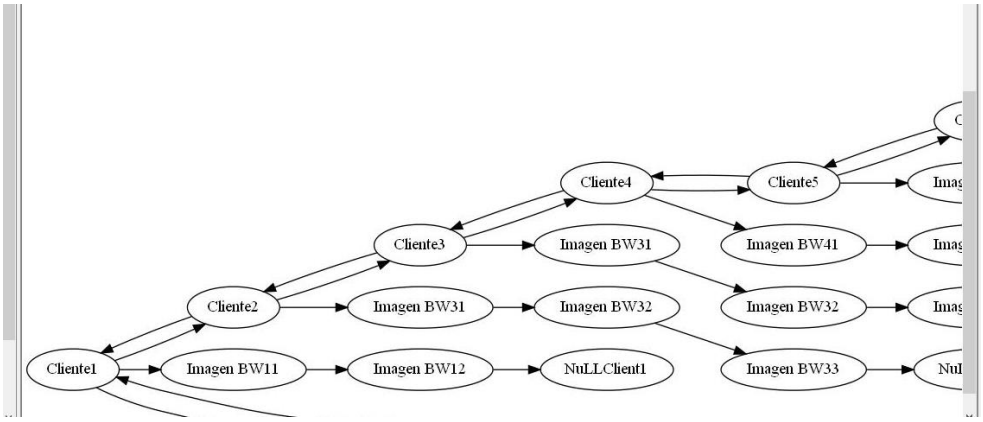
En este caso la cola de impresión negro ya esta vacia

5. Lista de clientes en espera (Lista de listas)

Luego de terminar el proceso en ventanilla, cada uno de los clientes se almacenará en una única lista de espera, en la que uno de los clientes (nodos) contará con una lista de imágenes impresas, estas estarán mezcladas..

Después de que la imagen se genere, esta buscará a que cliente pertenece, y se almacenará en la lista de imágenes del cliente sin importar el tipo de impresión. Una vez que la lista de imágenes del cliente esté completa, se eliminará al cliente de la lista de espera.

La lista de clientes debe ser una lista circular doblemente enlazada, la lista de imágenes de cada cliente no tiene restricción de implementación.



De esta manera quedaría la lista circular graficada.

Para la lista circular doblemente enlazada es mas complicado debido a que tenemos dos punteros y una cabeza o primero como referencia y una cola o ultimo que nos permite orientarnos ya que no tenemos los null para poder tener un final y para mejorar el asunto estas dos referencias ultimo y primero están conectados entres si haciendo que si uno se desplaza a la derecha o a la izquierda siempre va volver al mismo punto.

Ejemplo de Nodo:

```
*/
public class Nodo_2 {
    Object datos;
    Nodo_2 siguiente;
    Nodo_2 anterior;
    Lista_Imagenes lstImg = new Lista_Imagenes();
}
```

A continuación un ejemplo de como se manejo la lista doble circular

```
public class Lista_Circular {
    Nodo_2 primero;
    Nodo_2 ultimo;
    String dato;

    public Lista_Circular() {
        primero = null;
        ultimo = null;
    }

    public void ingresar(Object x) {
        Nodo_2 nuevo = new Nodo_2();
        nuevo.datos = x;
        if (primero == null) {
            primero = nuevo;
            primero.siguiente = primero;
            primero.anterior = ultimo;
            ultimo = nuevo;
        } else {
            ultimo.siguiente = nuevo;
            nuevo.siguiente = primero;
            ultimo = nuevo;
            primero.anterior = ultimo;

        }

    }
```

Reportes:

Para poder ver las graficas tenemos que presionar en el menú ver estructuras.

Y presione la opción 4:

En los reportes encontraremos la opción de buscar un cliente en específico ingresando su id nos mostrara su nombre, id, y objetos imágenes que almacena.

Acerca de:

Se encuentran los datos del autor de programa.

Salir:

Salir cierra el programa.

Anexos:

Estas son las librerías que se utilizar para el correcto funcionamiento de el programa

```
import java.util.Scanner;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.Iterator;
import java.util.Scanner;
import javax.swing.JFileChooser;
import javax.swing.filechooser.FileNameExtensionFilter;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;
import java.io.File;
import static java.lang.Math.random;
import javax.swing.JFileChooser;
import javax.swing.filechooser.FileNameExtensionFilter;
import java.util.Random;
```