Manual Técnico

Kelly Mischel Herrera Espino

Introducción

En el siguiente manual se describe a detalle las clases. Así como lo que contiene cada una y porque fue necesaria la creación de esta, para que se usó cada método, variable y lista, esto con el fin de saber que pasaba al ejecutar el programa, explicando de forma breve la lógica aplicada en este proyecto.

Listado de clases:

- **CargaMasiva**
- **Cliente**
- **ClienteConMasPasos**
- **ColaImpresora_bw**
- **ColaImpresora_c**
- **Cola_cliente**
- **Grafos**
- **ListaAtendidos**
- **ListaCircular**
- **ListaEsperaS**
- **ListaTop5Color**
- **ListaVentanillas**
- **Menu**
- **NodoAtendidos**
- **NodoCircular**
- **NodoCola**
- **NodoListaE**
- **NodoPila**
- **NodoVentanilla**
- **Nodo_impresoraBw**
- **Nodo_imprsoraC**
- **Operaciones**
- **PilaImagenes**
- **Top5_bw**

CargaMasiva:

Se crearon de forma global y estática las siguientes instancias de objetos:

```
 */
public class CargaMasiva {

    public static Cola_clientes cola = new Cola_clientes();
    public static ListaVentanillas ventanilla = new ListaVentanillas();
    public static ColaImpresora_bw impresora_bw = new ColaImpresora_bw();
    public static ColaImpresora_c impresora_c = new ColaImpresora_c();
    public static ListaCircular listaCicular = new ListaCircular();
    public static ListaAtendidos listaAtendidos = new ListaAtendidos();
    public static ListaTop5Color listaTop5 = new ListaTop5Color();
    public static Top5Bw listaTop5_bw = new Top5Bw();
    public static ClienteConMasPasos masPasos = new ClienteConMasPasos();
```

Función abrirArchivo:

Esta función rotorna un String, el cual es el contenido del archivo de entrada.json.

```
    public static String abrirArchivo() {

        Scanner entrada = new Scanner(System.in);
        System.out.println("ingrese la ruta:");
        String direccion = entrada.nextLine();
        String cadena = "";
        String texto = "";
        try {
//File abrir = new File(direccion);

            File archivo = new File(direccion);

            BufferedReader leer = new BufferedReader(new FileReader(archivo));

            while ((cadena = leer.readLine()) != null) {

                texto += cadena + "\n";

            }
            leer.close();

        } catch (Exception e) {
```

leerArchivoJson:

Se utilizo la librería Json.org para poder obtener los datos del archivo de entrada y poder guardarlos en la cola. Para ello se utilizó el método .getJSONObject el cual sirve para obtener los datos que estén dentro los diccionarios.

```
public static void leerArchivoJson() {

    System.out.println("_____\n\n");
    //Se cree un objeto de tipo Json a partir de una caddena de entrada.
    //La cadena se obtuvo de la carga masiva de un archivo.json (abrirArchivo funcion de tipo String que contiene la ca
    JSONObject myJSON2 = new JSONObject(abrirArchivo());
    System.out.println(myJSON2.keys());
    for (int i =1; i <=myJSON2.length(); i++) {

        //Se obtiene el nombre de la key del primer diccionario
        String c2 = "Cliente"+String.valueOf(i);
        System.out.println("c2"+c2);
        System.out.println("id_cliente:->" + myJSON2.getJSONObject(c2).get("id_cliente"));

        System.out.println("nombre_cliente:->" + myJSON2.getJSONObject(c2).get("nombre_cliente"));
        System.out.println("img_color:->" + myJSON2.getJSONObject(c2).get("img_color"));

        System.out.println("img_bw:->" + myJSON2.getJSONObject(c2).get("img_bw"));
        cola.encolar(Integer.parseInt(myJSON2.getJSONObject(c2).get("id_cliente").toString()), myJSON2.getJSONObject(c2
        System.out.println("_____");
        ultimoId = Integer.parseInt(myJSON2.getJSONObject(c2).get("id_cliente").toString());
        //con el segundo for se recorre para obtener los datos dentro de los subdiccionarios

    }
```

Clase Cliente

Se creo la clase cliente para poder crear varios objetos de tipo cliente.

Esta clase tiene sus atributos con visibilidad private para poder manejar el encapsulamiento y de esta forma solo poder acceder a sus métodos por medio de los getters y setters.

```
    private int totalImagenes;
    private String ventanilla;
    private int pasos = 0;

    public Cliente(int id_cliente, String nombre_cliente, int img_color, int img_bw, int totalImgenes, String ventanilla,
        this.id_cliente = id_cliente;
        this.nombre_cliente = nombre_cliente;
        this.img_color = img_color;
        this.img_bw = img_bw;
        this.totalImagenes = totalImgenes;
        this.img_color2 = img_color2;
        this.img_bw2 = img_bw2;
//----------------------------------------
        this.ventanilla = ventanilla;

    }
```

Clase ClienteConMasPasos

En esta lista se guardan los clientes conforme los pasos que dan de forma descendente y por último en el método mostrar se imprime en consola los datos del cliente con mas pasos.

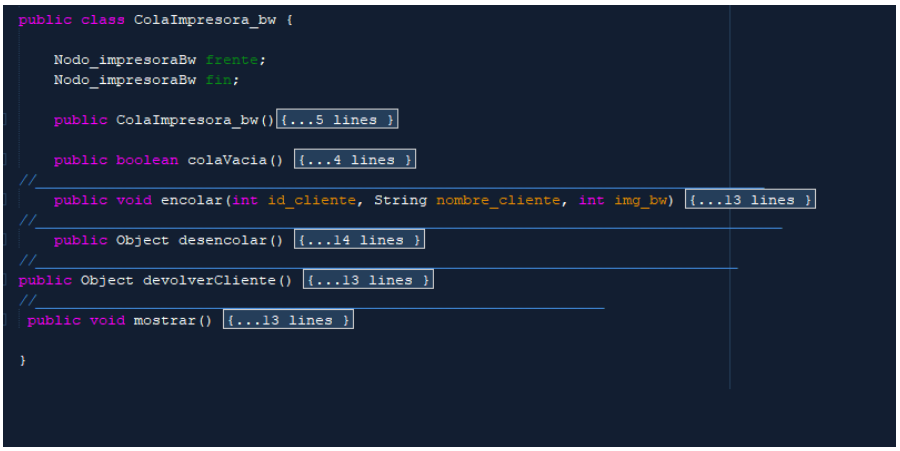
```
public class ClienteConMasPasos {

    NodoAtendidos cabeza;
    public int tamaño = 0;

    public ClienteConMasPasos() {

        this.cabeza = null;

    }

//Orden descendente por medio de los colores
    public void ordenarLista(Cliente cliente) {

        NodoAtendidos nuevo = new NodoAtendidos(cliente);

        if (this.cabeza == null) {
            this.cabeza = nuevo;
        } else if (nuevo.getCliente().getPasos() > this.cabeza.getCliente().getPasos()) {
            nuevo.setSiguiente(this.cabeza);
            this.cabeza = nuevo;

        } else {

            NodoAtendidos aux = this.cabeza;
```

ColaImpresora_bw

Es una cola con dos atributos de tipo Nodo_impresoraBw, estos son frente y fin.

La cola tiene los siguientes métodos:

- ColaImpresora_bw
- colaVacia()
- encolar()
- desencolar()
- devolverCliente()
- mostrar()



```
public class ColaImpresora_bw {

    Nodo_impresoraBw frente;
    Nodo_impresoraBw fin;

    public ColaImpresora_bw(){...5 lines }

    public boolean colaVacia() {...4 lines }
//_____
    public void encolar(int id_cliente, String nombre_cliente, int img_bw) {...13 lines }
//_____
    public Object desencolar() {...14 lines }
//_____
public Object devolverCliente() {...13 lines }
//_____
 public void mostrar() {...13 lines }

}
```

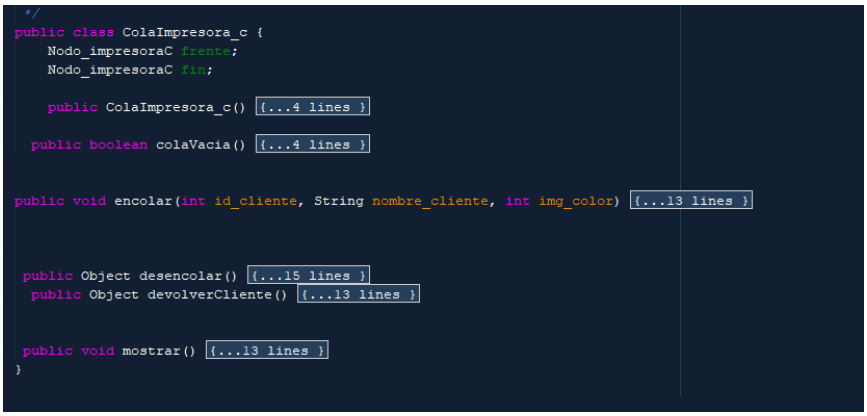
ColaImpresora_C

Es una cola con dos atributos de tipo Nodo_impresoraC, estos son frente y fin.

La cola tiene los siguientes métodos:

- ColaImpresora_C
- colaVacia()
- encolar()
- desencolar()
- devolverCliente()

- mostrar()

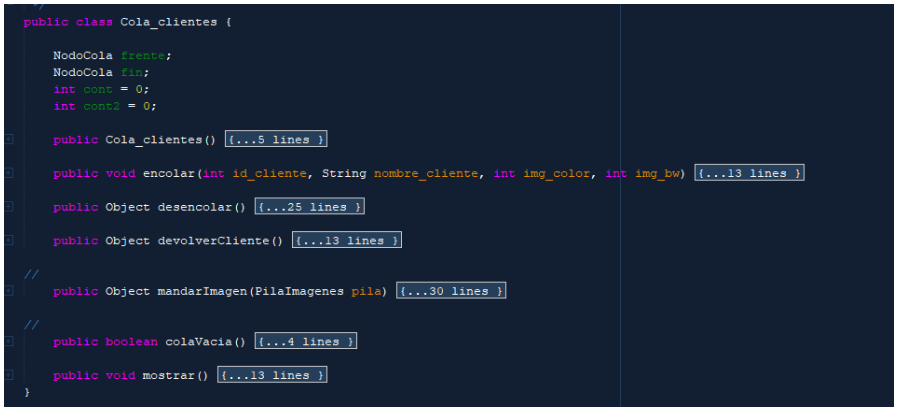


ColaClientes

Es una cola con dos atributos de tipo NodoClientes, estos son frente y fin.

La cola tiene los siguientes métodos:

- ColaClientes()
- colaVacia()
- encolar()
- desencolar()
- devolverCliente()
- mostrar
- MandarImagen()

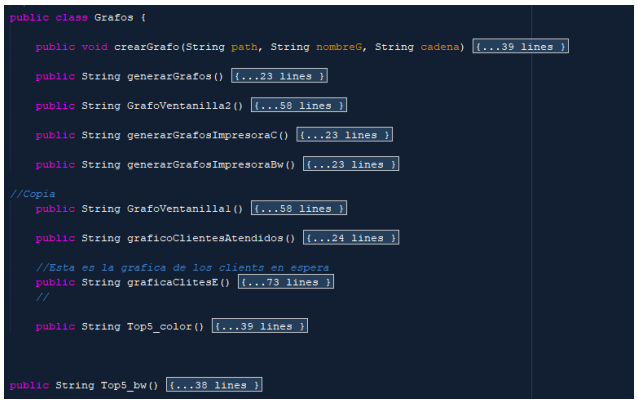


Grafos

En esta clase se encuentran los métodos para crear y genera los grafos con Graphivz de las distintas estructuras utilizadas en todo el programa.

Listado de metosos:

- crearGrafo: Este método recicbe 3 String los cuales son para el nombre de la imagen, nombre del archivo.dot y la estructura del Graphivz guardada en un String.
- GrafoVentanilla(): Crea el grafo de las ventanillas junto con sus pilas.
- GenerarGrafosImpresroasC: crea el grafo de la cola de imágenes a color.
- GenerarGrafosImpresroasC: crea el grafo de la cola de imágenes a color.
- GraficoClentEsAntendidos: crea el grafo de los cliente atendios.
- GraficaClientesE: crea el grafo de los clientes en espera.
- Top5_color: crea el grafo del top 5 de clientes con mayor cantidad de imagens a color.
- To5_bw: crea el grafo del top 5 de clientes con menor cantidad de imágenes bw.

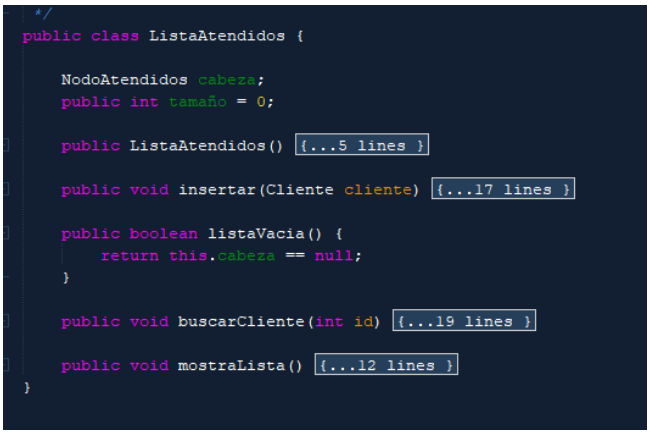


ListaAtendidos

Es una lista simplemente enlazada, la cual tiene un NodoAtendidos como cabeza. En esta lista se almacenan todo los clientes que hayan sido atendidos.

La lista cuenta con los siguientes métodos:

- Insertar
- listaVacia
- buscarliente
- mostrarLista

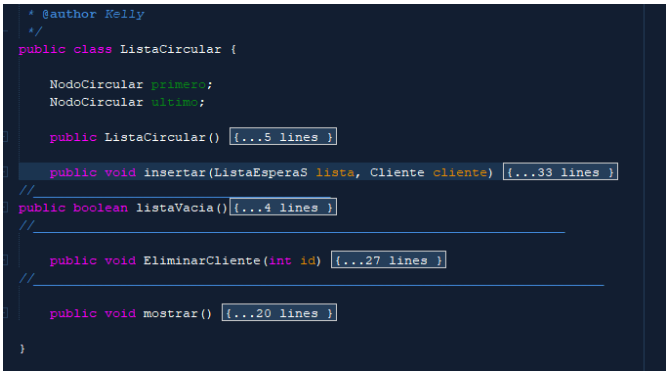


ListaCiruclar

Esta es una lista circular doblemente enlazada, en la cual se almacena listas de imágenes. Para esta lista se utilizo el NodoCircular primero y el NodoCircular ultimo.

La lista cuenta con los siguientes métodos:

- ListaCiruclar(): Es el constructor de la clase.
- Insertar: este método recibe como parámetro una lista y un cliente
- ListaVacia: verifica si esta se encuentra vacia y retorna true.
- ElieminarCliente: Elimina al cliente de la lista
- Mostrar: muestra los cliente en consola.

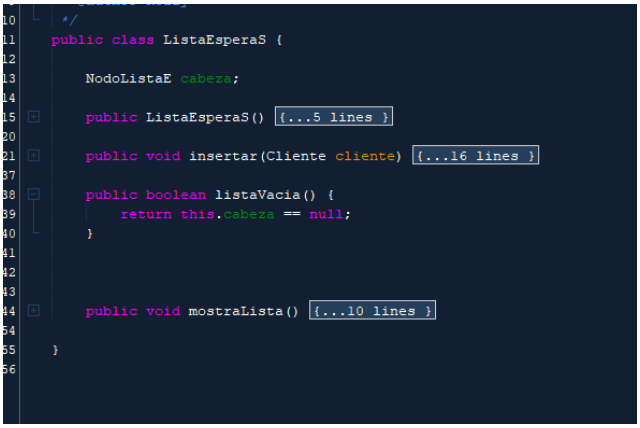


ListaEsperaS

Esta es una lista simplemente enlazada en donde se almacenan las imágenes de los clientes en espera.

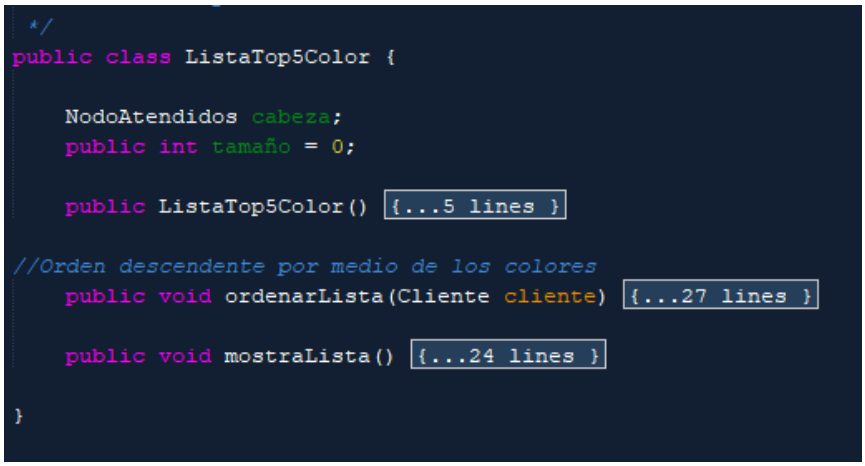
Esta lista tiene los siguientes métodos:

- Insertar()
- listaVacia
- mostrarLista



ListaTop5Color

En esta lista se guardan los clientes conforme las cantidad de imágenes a color que tiene esto de forma descendente y por último en el método mostrar se imprime en consola los datos del cliente con mas imágenes a color.



Lista de Ventanillas

Es una lista enlazada de pilas. En donde se guarda el cliente y pasa sus imágenes a su respectiva pila.

```
 */
public class ListaVentanillas {

    NodoVentanilla cabeza;

    public ListaVentanillas() {...5 lines }

    public void insertar(PilaImagenes pila, String ventanilla, Cliente cliente) {...22 lines }

    public void mostrarLista() {...19 lines }
///

    public String devolverPila() {...19 lines }

}
```

Menu

En esta clase se colocan los métodos del menú principal. Cada método es una opción que cliente ingresa.

```
public class Menu {

    public static int cont1 = 0;
    public static int cont2 = 0;
    public static int pasos = 1;

    public void menu() {...71 lines }
//mostrar una linea

    public void linea() {...4 lines }
//Mostrar las opciones del menu

    public String opciones() {...9 lines }
//Mostrar datos del estudiante

    public String acerDe() {...6 lines }
//opciones de parametros inciales

    public void parametrosIniciales() {...19 lines }
//Menu de reportes

    public void reportes() {...38 lines }

    //Cantidad de ventanillas
    public int cantidadVentanillas() {...9 lines }
}
```

NodoAtendidos

Tiene como atributo un cliente de tipo Cliente y un NodoAtentdidos siguientes. Esta Clase nodo continente el nodo de la listaAtendidos.

```java
 */
public class NodoAtendidos {

    private Cliente cliente;
    private NodoAtendidos siguiente;

    public NodoAtendidos(Cliente cliente) {
        this.cliente = cliente;
        this.siguiente = null;
    }

    public Cliente getCliente() {
        return cliente;
    }

    public void setCliente(Cliente cliente) {
        this.cliente = cliente;
    }

    public NodoAtendidos getSiguiente() {
```

Clase NodoCircular

Tiene como atributo una lista de tipo ListaEsperaS, Cliente de tipo Cliente, NodoCircular siguiente Y NodoCircular anterior. Esta Clase NodoCiruclar continente el nodo de la listaCircular.

```java
 * @author Kelly
 */
public class NodoCircular {

    private ListaEsperaS lista;
    private Cliente cliente;
    private NodoCircular siguiente;
    private NodoCircular anterior;

    public NodoCircular(ListaEsperaS lista, Cliente cliente) {...6 lines }

    public Cliente getCliente() {...3 lines }

    public void setCliente(Cliente cliente) {
        this.cliente = cliente;
    }

    public NodoCircular getSiguiente() {
        return siguiente;
    }

    public void setSiguiente(NodoCircular siguiente) {
```

NodoCola

Tiene como atributo tres enteros, un String y un NodoCola siguiente. Esta Clase NodoCiruclar continente el nodo de la ColaCliente.

```java
*/
public class NodoCola {

    private int id_cliente;
    private String nombre_cliente;
    private int img_color;
    private int img_bw;
    private NodoCola siguiente;

    public NodoCola(int id_cliente, String nombre_cliente, int img_color, int img_bw) {
        this.id_cliente = id_cliente;
        this.nombre_cliente = nombre_cliente;
        this.img_color = img_color;
        this.img_bw = img_bw;
        this.siguiente = null;
    }

    public int getId_cliente() {
        return id_cliente;
    }

    public void setId_cliente(int id_cliente) {
        this.id_cliente = id_cliente;
    }

    public String getNombre_cliente() {
        return nombre_cliente;
    }
```

NodoListaEl

Tiene como atributos cliente de tipo Cliente y un NodoListaEl siguiente . Esta Clase NodoCiruclar continente el nodo de la ListadeEspera.

```java
public class NodoListaE {

    private Cliente cliente;
    private NodoListaE siguiente;

    public NodoListaE(Cliente cliente) {
        this.cliente = cliente;
        this.siguiente = null;
    }

    public Cliente getCliente() {
        return cliente;
    }

    public void setCliente(Cliente cliente) {
        this.cliente = cliente;
    }

    public NodoListaE getSiguiente() {
        return siguiente;
    }

    public void setSiguiente(NodoListaE siguiente) {
        this.siguiente = siguiente;
    }

}
```

NodoPila

Tiene como atributo tres enteros, un String y un NodoPila siguiente. Esta Clase NodoPila es de la Pila.

```java
 */
public class NodoPilaI {

    private int id_cliente;
    private String nombre_cliente;
    private int img_color;
    private int img_bw;
    private NodoPilaI siguiente;

    public NodoPilaI(int id_cliente, String nombre_cliente, int img_color, int img_bw) {
        this.id_cliente = id_cliente;
        this.nombre_cliente = nombre_cliente;
        this.img_color = img_color;
        this.img_bw = img_bw;

        this.siguiente = null;
    }

    public int getId_cliente() {
        return id_cliente;
    }
```

Nodo_impresoraBw

Tiene como atributos dos enteros, un String y un Nodo_impresoraBw siguiente. Esta Clase Nodo_impresoraBw continente el nodo de la Cola_impresorabW.

```java
 * @author Kelly
 */
public class Nodo_impresoraBw {

    private int id_cliente;
    private String nombre_cliente;
    private int img_bw;
    private Nodo_impresoraBw siguiente;

    public Nodo_impresoraBw(int id_cliente, String nombre_cliente, int img_bw) {
        this.id_cliente = id_cliente;
        this.nombre_cliente = nombre_cliente;
        this.img_bw = img_bw;
        this.siguiente = null;
    }

    public int getId_cliente() {
        return id_cliente;
    }

    public void setId_cliente(int id_cliente) {
        this.id_cliente = id_cliente;
```

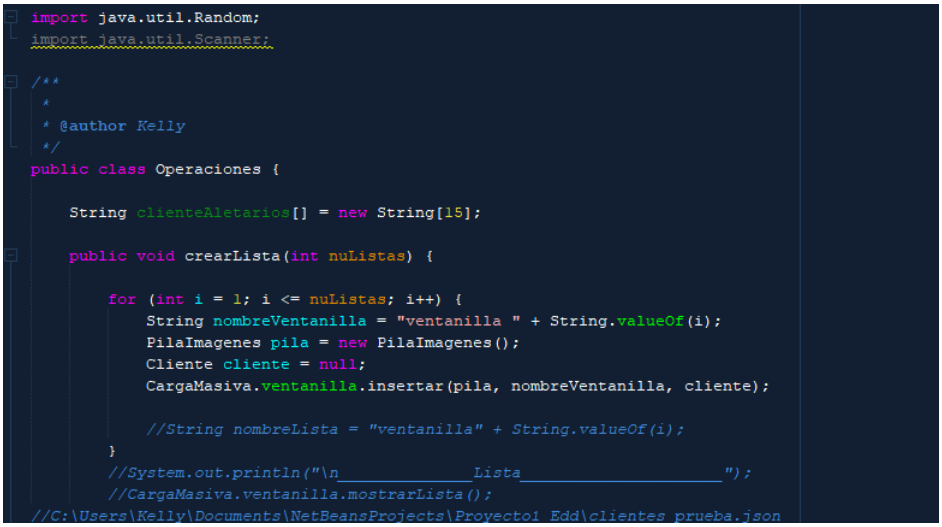
Nodo_impresoraC

Clase Nodo_impresoraBw continente el nodo de la Cola_impresorabW.

```
 */
public class Nodo_impresoraC {

    private int id_cliente;
    private String nombre_cliente;
    private int img_color;

    private Nodo_impresoraC siguiente;

    public Nodo_impresoraC(int id_cliente, String nombre_cliente, int img_color) {
        this.id_cliente = id_cliente;
        this.nombre_cliente = nombre_cliente;
        this.img_color = img_color;

        this.siguiente = null;
    }

    public int getId_cliente() {
        return id_cliente;
    }
```

Clase Operaciones

En esta clase se manejan las diferentes transiciones en las que un cliente se mueve en cada estructura conforme da los distintos pasos.



```
import java.util.Random;
import java.util.Scanner;

/**
 *
 * @author Kelly
 */
public class Operaciones {

    String clienteAletarios[] = new String[15];

    public void crearLista(int nuListas) {

        for (int i = 1; i <= nuListas; i++) {
            String nombreVentanilla = "ventanilla " + String.valueOf(i);
            PilaImagenes pila = new PilaImagenes();
            Cliente cliente = null;
            CargaMasiva.ventanilla.insertar(pila, nombreVentanilla, cliente);

            //String nombreLista = "ventanilla" + String.valueOf(i);
        }
        //System.out.println("\n_____Lista_____");
        //CargaMasiva.ventanilla.mostrarLista();
//C:\Users\Kelly\Documents\NetBeansProjects\Proyecto1_Edd\clientes_prueba.json
```

Clase Pila_imagenes

En esta clase se crea una Pila para almacenar las imágenes de los clientes que ingresan a las ventanillas.

```java
 */
public class PilaImagenes {

    NodoPilaI cima;

    public PilaImagenes() {

        this.cima = null;

    }

    public boolean PilaVacia() {
        return this.cima == null;

    }

    public void push(int id_cliente, String nombre_cliente, int img_color, int img_bw) {

        //el nuevo nodo
        NodoPilaI nuevo = new NodoPilaI(id_cliente, nombre_cliente, img_color, img_bw);
        //el siguiente del nuevo nodo apuntara a la cima actual
```

Conclusión

El programa se logró estructurar y acoplar a los requerimientos que se pedían por parte del cliente. Utilizando el paradigma de programación orientada a objetos se obtuvo una mejor organización del código que hacía más eficaz con una mejor organización a la hora de llamar clases y métodos, también se hizo uso de las distintas estructuras para almacenar los datos de los clientes.