Universidad de San Carlos de Guatemala
Escuela de Ciencias y Sistemas
Estructuras De Datos

Manual Técnico
Proyecto 1 Fase 1
UDrawing Paper

Arnoldo Luis Antonio González Camey
Carné: 201701548

INTRODUCCION

Este manual va dirigido a todo público que cuente con conocimientos técnicos sobre la programación y que quiera ver la interacción entre clases, objetos y el uso de listas simples, colas, pilas y listas circulares doblemente enlazadas, por medio de una simulación de una empresa "Drawing Paper" la cual se dedica a imprimir imágenes en distinto tamaños y tipos de papel; este describe el funcionamiento de cada uno de los métodos utilizados en el programa. El reporte grafico se realiza a través del Graphviz, para ello se hizo uso del IDE NetBeans 11.0.

REQUERIMIENTOS MÍNIMOS

Para poder operar el programa se necesita: Una computadora con un procesador de 32 o 64 bits, sistema operativo Windows 7 en adelante, Linux o cualquier otro sistema que permita la instalación de JAVA, tener instalado JAVA en la versión más reciente, tener la capacidad de acceder a los comandos de Windows o Linux.

OBJETIVOS

General

- Aplicar los conocimientos del curso de Estructuras de Datos en el desarrollo de una aplicación que permita manipular la información de forma óptima.

Específicos

- Demostrar los conocimientos adquiridos sobre estructuras de datos lineales poniéndolos en practica en una aplicación de simulación.
- Utilizar el lenguaje Java para implementar estructuras de datos lineales.
- Utilizar la herramienta Graphviz para graficar estructuras de datos líneas.
- Definir e implementar algoritmos de búsqueda, recorrido y eliminación

Manual Técnico

- Método insertarCliente, pushColor, pushByN

En estos metodos la lógica es la misma y son utilizados para insertar los clientes, imágenes a color y blanco y negro en su respectiva cola para ello se colocan como parámetros los atributos del objeto, luego dentro del método se crea un nuevo objeto y se valida si la cabeza de la cola es igual a nulo, es decir, no existe ningun elemento, si es así el nuevo objeto se coloca como la nueva cabeza, si esto no se cumple se crea una variable auxiliar, la cual se recorre hasta que el siguiente de esta variable sea nula, al finalizar se establece que el siguiente del actual es el nuevo objeto que se desea insertar.

```
public void insertarCliente(String key_titulo, int id, String nombre, int color, int bYn) {
    Cliente nuevo = new Cliente(key_titulo, id, nombre, color, bYn, 0, 0, 0);
    if (this.getCabeza() == null) {
        this.setCabeza(nuevo);
    } else {
        Cliente actual = this.getCabeza();
        while (actual.getSiguiente() != null) {
            actual = actual.getSiguiente();
        }
        actual.setSiguiente(nuevo);
    }
}
```

```
public void pushColor(int idCliente, boolean tipoImpresion) {
    Imagen nuevo = new Imagen(idCliente, tipoImpresion);
    this.sizeColor++;
    if (this.getCabezaColor() == null) {
        this.setCabezaColor(nuevo);
    } else {
        Imagen actual = this.getCabezaColor();
        while (actual.getSiguiente() != null) {
            actual = actual.getSiguiente();

        }
        actual.setSiguiente(nuevo);
    }
}
```

```
public void pushByN(int idCliente, boolean tipoImpresion) {
    Imagen nuevo = new Imagen(idCliente, tipoImpresion);
    this.sizeByN++;
    if (this.getCabezaByN() == null) {
        this.setCabezaByN(nuevo);
    } else {
        Imagen actual = this.getCabezaByN();
        while (actual.getSiguiente() != null) {
            actual = actual.getSiguiente();

        }
        actual.setSiguiente(nuevo);
    }
}
```

- Método popCliente, popColor, popByN

En estos metodos la lógica es la misma y son utilizados para sacar un cliente, una imagen a color o una imagen en blanco y negro y luego retornarla, para ello se crea un nuevo objeto y se inicializa como nulo, luego se valida si la cabeza no es igual a nula, si es así el valor del retorno es igual a la cabeza de la cola, seguido de esto se valida si el siguiente de la cabeza no es nula, si no lo es se hace que la nueva cabeza sea el siguiente al retorno, si lo anterior no se valia la cabeza se vuelve nula y al final se retorna el objeto en cuestion.

```
public Cliente popCliente() {
    Cliente retorno = null;
    if (this.getCabeza() != null) {
        retorno = this.getCabeza();
        if (this.getCabeza().getSiguiente() != null) {
            this.setCabeza(retorno.getSiguiente());
        } else {
            this.setCabeza(null);
        }
        return retorno;
    }
    return retorno;
}
```

```
public Imagen popColor() {
    Imagen retorno = null;
    if (this.getCabezaColor() != null) {
        retorno = this.getCabezaColor();
        if (this.getCabezaColor().getSiguiente() != null) {
            this.setCabezaColor(retorno.getSiguiente());
        } else {
            this.setCabezaColor(null);
        }
        return retorno;
    }
    return retorno;
}
```

```
public Imagen popByN() {
    Imagen retorno = null;
    if (this.getCabezaByN() != null) {
        retorno = this.getCabezaByN();
        if (this.getCabezaByN().getSiguiente() != null) {
            this.setCabezaByN(retorno.getSiguiente());
        } else {
            this.setCabezaByN(null);
        }
        return retorno;
    }
    return retorno;
}
```

- Método generarClientesRandom

Este método es usado para generar los clientes que ingresan al sistema para ello se declaran dos arreglos con nombres y apellidos, luego se crea una variable entera en la cual se guarda el id del ultimo cliente que se genero, para ello se recorre la cola de clientes; para generar los clientes se recorre mediante un ciclo el cual tiene una longitud entre 0 y 3, dentro de este ciclo se generan más numeros random los cuales son usados para objener una posicion dentro del arreglo de nombres y apellidos. Al final se hace la isntancia del metodo insertarCliente.

```
public void generarClientesRandom() {
    String nombres[] = {"Rossmery", "Ronald", "Daniel", "Oscar", "Samuel", "Eric", "Sergio", "Nicolas", "Gabriela",
        "Stephany", "Emiliano", "Karla", "Angel", "Rebeca", "Lionel", "Guillermo", "Raul", "David", "Lucia", "Emma"};
    String apellidos[] = {"Castillo", "Hernandez", "Lopez", "Martinez", "Rodriguez", "Gonzalez", "Garcia", "Silva", "Ruiz",
        "Fernandez", "Sanchez", "Perez", "Diaz", "Gomez", "Morales", "Suarez", "Santos", "Marquez", "Reyes", "Mora"};
    int ultimoId = 0;
    Cliente actual = this.getCabeza();
    while (actual != null) {
        if (actual.getSiguiente() == null) {
            ultimoId = actual.getId() + 1;
        }
        actual = actual.getSiguiente();
    }
    int cantidadClientes = (int) (Math.random() * (3 - 0 + 1) + 0);
    System.out.println("\nHan ingresado " + cantidadClientes + " nuevos Clientes");
    for (int i = 0; i < cantidadClientes; i++) {
        int nombreRandom = (int) (Math.random() * (19 - 0 + 1) + 0);
        int apellidoRandom = (int) (Math.random() * (19 - 0 + 1) + 0);
        int cantidadColor = (int) (Math.random() * (4 - 0 + 1) + 0);
        int cantidadByN = (int) (Math.random() * (4 - 0 + 1) + 0);
        insertarCliente("Cliente" + (ultimoId + i), (ultimoId + i), nombres[nombreRandom] + " " + apellidos[apellidoRandom],
                cantidadColor, cantidadByN);
          System.out.println("Cliente" + (ultimoId + i) + " - " + (ultimoId + i) + " - " + nombres[nombreRandom] + " " + ape
                  + " - " + cantidadColor + " - " + cantidadByN);
    }
}
```

- Método push

Este método es usado para insertar un nuevo elemento en la pila de imágenes de las ventanillas, para ello se colocan como parámetros los atributos de la imagen, luego dentro del método se crea una nueva imagen, y se aumenta el tamaño de la pila, seguido de esto se valida si la cabeza de la pila es igual a nulo, es decir, no existe ningun elemento, si es así el nuevo objeto es igual a la cabeza, sino el siguiente del nuevo objeto es igual a la cabeza y la nueva cabeza seria la imagen que se quiere insertar.

```
public void push(int idCliente, boolean tipoImpresion) {
    Imagen nuevo = new Imagen(idCliente, tipoImpresion);
    this.size++;
    if (this.getCabeza() == null) {
        this.setCabeza(nuevo);

    } else {
        nuevo.setSiguiente(this.getCabeza());
        this.setCabeza(nuevo);
    }
}
```

- Método insertarImagen, insertarClienteAtendido

En estos metodos la lógica es la misma y son utilizados para insertar los clientes y las imágenes en su respectiva lista simple para ello se colocan como parámetros los atributos del objeto o el objeto en si, luego dentro del método se crea un nuevo objeto y se valida si la cabeza de la lista simple es igual a nulo, es decir, no existe ningun elemento, si es así el nuevo objeto se coloca como la nueva cabeza, si esto no se cumple se crea una variable auxiliar, la cual se recorre hasta que el siguiente de esta variable sea nula, al finalizar se establece que el siguiente del actual es el nuevo objeto que se desea insertar.

```
public void insertarClienteAtendido(String key_titulo, int id, String nombre, int color, int bYn, int cantidadPasos) {
    Cliente nuevo = new Cliente(key_titulo, id, nombre, color, bYn, 0, 0, cantidadPasos);
    if (this.getCabezaCliente() == null) {
        this.setCabezaCliente(nuevo);
    } else {
        Cliente actual = this.getCabezaCliente();
        while (actual.getSiguiente() != null) {
            actual = actual.getSiguiente();
        }
        actual.setSiguiente(nuevo);
    }
}
```

```
public void insertarImagen(Imagen imagen) {
    Imagen nuevo = new Imagen(imagen.getIdCliente(), imagen.isTipoImpresion());
    if (this.getCabezaImg() == null) {
        this.setCabezaImg(nuevo);
    } else {
        Imagen actual = this.getCabezaImg();
        while (actual.getSiguiente() != null) {
            actual = actual.getSiguiente();
        }
        actual.setSiguiente(nuevo);
    }
}
```

- Método insertarVentanilla

Este método es utilizado para insertar las ventanillas en una lista simple para ello se coloca como parámetro la cantidad de ventanillas, luego dentro del método se crea un ciclo while el cual finaliza en el momento que el indice sea menor o igual a la cantidad, dentro de este se crea una nueva ventanilla y se valida si la cabeza de la lista simple es igual a nulo, es decir, no existe ningun elemento, si es así la nueva ventanilla se coloca como la nueva cabeza, si esto no se cumple se crea una variable auxiliar, la cual se recorre hasta que el siguiente de esta variable sea nula, al finalizar se establece que el siguiente del actual es la nueva ventanilla que se desea insertar y se va aumentando en uno el valor del indice en cada iteración.

```java
public void insertarVentanilla(int cantidad) {
    int indice = 1;
    while (indice <= cantidad) {
        Ventanilla nuevo = new Ventanilla(0, indice, false);
        if (this.getCabezaVen() == null) {
            this.setCabezaVen(nuevo);
        } else {
            Ventanilla actual = this.getCabezaVen();
            while (actual.getSiguiente() != null) {
                actual = actual.getSiguiente();
            }
            actual.setSiguiente(nuevo);
        }
        indice++;
    }
}
```

- Método pasarVentanilla

Este método es utilizado para que el cliente acceda a una ventanilla vacía, para ello se coloca como parámetro el cliente a ingresar, luego se crea una variable actualV la cual es igual a la cabeza de ventanillas, seguido de esto se crea un ciclo while el cual finaliza cuando la variable actualV es nula, luego se valida si la ventanilla actual no esta ocupada por otro cliente, si no es asi se crea una variable auxiliar del cliente y se valida si no es igual a nula, si no lo es se agrega un paso al cliente y a la ventanilla se le agrega el cliente, se coloca como ocupada y el cliente de la cola de recepción se elimina.

```java
public void pasarVentanilla(Cola colaCliente) {
    Ventanilla actualV = this.getCabezaVen();
    while (actualV != null) {
        if (!actualV.isOcupada()) {
            Cliente actual = colaCliente.getCabeza();
            if (actual != null) {
                System.out.println("Ingresa a ventanilla " + actualV.getNumVentanilla()
                        + " el Cliente " + actual.getNombre() + " con Id No. " + actual.getId());
                actual.setCantidadPasos(actual.getCantidadPasos() + 1);
                actualV.setOcupada(true);
                actualV.setId_cliente(actual.getId());
                colaCliente.popCliente();
                actualV.setCliente(actual);
            }
        }
        actualV = actualV.getSiguiente();
    }
}
```

- Método darImagen

Este método es utilizado para que el cliente proporcione las imágenes que desea imprimir a la ventanilla en la que se encuentra, para ello se crea una variable actualV la cual es igual a la cabeza de ventanillas, seguido de esto se crea un ciclo while el cual finaliza cuando la variable actualV es nula, luego se valida si la ventanilla actual no esta ocupada y el cliente en la ventanilla no es nulo, luego se valida si la cantidad de imágenes del cliente en ventanilla, tanto en color como en blanco y negro, es mayor a cero y la cantidad de imágenes es mayor a la cantidad de imágenes que ha entregado hasta el momento, si se cumple entra y entrega una imagen a la ventanilla, la cual se inserta a la respectiva pila y se modifican los valores de entrega, si no se cumple con lo anterior se ingresa al else, el cual significa que el cliente ya no tiene más imágenes que imprimir por lo tanto, el cliente se mueve a la lista de espera, y se realizan las modificaciones de las ventanillas, asi como enviar la pila de impresión a la cola de las impresoras.

```
public void darImagen(Impresora cabezaImprColor, Impresora cabezaImprByN, ListaCircularDoble clienteEspera) {
    Ventanilla actualV = this.getCabezaVen();
    while (actualV != null) {
        if (actualV.isOcupada() && actualV.getCliente() != null) {
            if (actualV.getCliente().getColor() > 0 && actualV.getCliente().getColor() > actualV.getCliente().getEntregaCo
                actualV.getPilaImagen().push(actualV.getCliente().getId(), true);
                actualV.getCliente().setEntregaColor(actualV.getCliente().getEntregaColor() + 1);
                actualV.getCliente().setCantidadPasos(actualV.getCliente().getCantidadPasos() + 1);
                System.out.println("No. Ventanilla: " + actualV.getNumVentanilla() + " - Recibe imagen a Color del Cliente
                        + actualV.getCliente().getNombre() + " con Id No. " + actualV.getCliente().getId());

            } else if (actualV.getCliente().getByN() > 0 && actualV.getCliente().getByN() > actualV.getCliente().getEntreg
                actualV.getPilaImagen().push(actualV.getCliente().getId(), false);
                actualV.getCliente().setEntregaByN(actualV.getCliente().getEntregaByN() + 1);
                actualV.getCliente().setCantidadPasos(actualV.getCliente().getCantidadPasos() + 1);
                System.out.println("No. Ventanilla: " + actualV.getNumVentanilla() + " - Recibe imagen a Blanco y Negro de
                        + actualV.getCliente().getNombre() + " con Id No. " + actualV.getCliente().getId());

            } else {
                System.out.println("\t*Se Retira de Ventanilla " + actualV.getNumVentanilla() + " el Cliente "
                        + actualV.getCliente().getNombre() + " con Id No. " + actualV.getCliente().getId()+"*\n");
                actualV.getCliente().setCantidadPasos(actualV.getCliente().getCantidadPasos() + 1);
                actualV.getCliente().setEntregaByN(0);
                actualV.getCliente().setEntregaColor(0);
                actualV.setOcupada(false);
                actualV.setId_cliente(0);
                enviarColaImpresion(actualV.getPilaImagen(), cabezaImprColor, cabezaImprByN);
                actualV.setPilaImagen(new Pila());
                clienteEspera.insertar(actualV.getCliente());
            }
        }
        actualV = actualV.getSiguiente();
    }
}
```

- Método enviarColaImpresion

El método se usa para enviar la pila de imágenes a la cola de las impresoras, para ello se crea un variable imagenActual, luego mediante un ciclo while la cual se recorre hasta que la imagenActual no sea igual a nula se valida que tipo de imagen es y se hace la instancia del método push de cada impresora.

```
public void enviarColaImpresion(Pila pilaImagen, Impresora cabezaImprColor, Impresora cabezImprByN) {
    Clases.Imagen imagenActual = pilaImagen.getCabeza();
    System.out.println("\t*Enviando Pila de Imagenes a Cola de Impresión*\n");
    while (imagenActual != null) {
        if (imagenActual.isTipoImpresion()) {
            cabezaImprColor.getColaImagen().pushColor(imagenActual.getIdCliente(), true);
        } else {
            cabezImprByN.getColaImagen().pushByN(imagenActual.getIdCliente(), false);
        }
        imagenActual = imagenActual.getSiguiente();
    }
}
```

- Método insertar

Este método es utilizado para insertar los clientes de espera en una lista circular doblemente enlazada, para ello se crea un nuevo objeto y se aumenta de tamaño la cantidad de nodos de la lista, luego se valida si la cabeza de la lista es igual a nulo, es decir, no existe ningun elemento, si es así el siguiente del nuevo cliente es el siguiente de la cabeza, el siguiente de la cabeza es el nuevo cliente a ingresar y el anterior del nuevo cliente es la cabeza. Al finalizar se coloca que la nueva cabeza es el nuevo cliente y el anterior del siguiente es el nuevo.

```
public void insertar(Cliente clienteEspera) {
    Cliente nuevo = new Cliente(clienteEspera.getKey_titulo(), clienteEspera.getId(),
            clienteEspera.getNombre(), clienteEspera.getColor(), clienteEspera.getByN(), clienteEspera.getCantidadPasos());
    this.size++;
    if (this.getCabezaDoble() != null) {
        nuevo.setSiguienteDoble(this.getCabezaDoble().getSiguienteDoble());
        this.getCabezaDoble().setSiguienteDoble(nuevo);
        nuevo.setAnteriorDoble(this.getCabezaDoble());
    }
    this.setCabezaDoble(nuevo);
    this.getCabezaDoble().getSiguienteDoble().setAnteriorDoble(nuevo);
}
```

- Método entregarImagen

Este método es utilizado para entregar a un cliente en la lista de espera una imagen ya impresa, para ello se crea un nuevo objeto y mediante un ciclo for que recorre la cantidad de clientes que hay en la lista de espera, se valida si la imagen enviada no es igual a nulo y si el cliente actual no es igual a nulo, luego se valida si el id del cliente asociada a la imagen es igual a el id del cliente en espera, si es aspi se inserta la imagen al cliente y se retorna.

```
public void entregarImagen(Clases.Imagen imagen) {
    Cliente actual = this.getCabezaDoble();
    for (int i = 0; i < size; i++) {
        if (imagen != null && actual != null) {
            if (imagen.getIdCliente() == actual.getId()) {
                System.out.println("Se Entrega Impresión al Cliente " + actual.getNombre() + " con Id No. " + actual.getId());
                actual.setNumImg(actual.getNumImg() + 1);
                actual.getListaImagenes().insertarImagen(imagen);
                return;
            }
            actual = actual.getSiguienteDoble();
        }
    }
}
```

- Método terminarClienteEspera

Este método es utilizado para sacar un cliente de la lista de espera, para ello se valida si la cabeza no es nula, sino se crea un cliente aux que se iguala a la cabeza y un cliente ant nulo, luego mediante un ciclo for que recorre el tamaño de la lista, se valida si la cantidad de imágenes recibidas por el cliente es igual a la suma de imágenes que el cliente solicito, si es así se valida si el ant es nulo, luego se valida el caso en el que el siguiente del auxiliar es igual a la cabeza, si es asi la cabeza se vuelve nula y se devuele el cliente atendido; sino se hace que la variable ant es igual al anterior del auxiliar, el siguiente del anterior el el siguiente del auxiliar y el ant es anterior del siguiente del siguiente del auxiliar y se retorna el cliente atendido. Si la variable ant no es nula, el anterior del auxiliar se vuelve nulo, el siguiente del ant es el siguiente del auxiliar y el siguiente del siguiente del auxiliar es el ant y se retorna el cliente atendido.

```
public Cliente terminarClienteEspera() {
    if (this.getCabezaDoble() != null) {
        Cliente aux = this.getCabezaDoble();
        Cliente ant = null;
        for (int i = 0; i < this.size; i++) {
            if (aux.getNumImg() == (aux.getColor() + aux.getByN())) {
                if (ant == null) {
                    if (aux.getSiguienteDoble() == this.getCabezaDoble()) {
                        Cliente atendido = aux;
                        this.setCabezaDoble(null);
                        size--;
                        return atendido;
                    } else {
                        Cliente atendido = aux;
                        ant = aux.getAnteriorDoble();
                        ant.setSiguienteDoble(aux.getSiguienteDoble());
                        aux = aux.getSiguienteDoble();
                        aux.setAnteriorDoble(ant);
                        this.setCabezaDoble(aux);
                        ant = null;
                        size--;
                        return atendido;
                    }
                } else {
                    Cliente atendido = aux;
                    aux.setAnteriorDoble(null);
                    ant.setSiguienteDoble(aux.getSiguienteDoble());
                    aux = aux.getSiguienteDoble();
                    aux.setAnteriorDoble(ant);
                    size--;
                    return atendido;
                }
            } else {
                ant = aux;
                aux = aux.getSiguienteDoble();
            }
        }
```

- Método agregarPasoClienteEspera

Método utilizado para agregar un paso a todos los clientes que se encuentran en la lista de espera, para ello se recorre la lista y se aumenta en uno la cantidad de pasos.

```
public void agregarPasoClienteEspera() {
    Cliente actual = this.getCabezaDoble();
    for (int i = 0; i < size; i++) {
        actual.setCantidadPasos(actual.getCantidadPasos() + 1);
        actual = actual.getSiguienteDoble();
    }
}
```

- Método graficarClienteRecepcion, graficaColaColor, graficaColaByN, graficaClienteAtendido

Métodos utilizados para la graficacion en graphviz, para ello se crea una variable string con la base del archivo dot a generar, luego se crea una variable objeto y se inicializa con la cabeza, luego se valida si la variable es nula, y se concatena la raiz con el nodo en cuestion; luego mediante un ciclo for que recorre la lista se crean los componentes necesarios para el archivo y los apuntadores entre nodos que debe tener. Al finalizar se retorna una variable string.

```
public String graficaClienteRecepcion() {
    int id = 1;
    String contenido = "digraph L{\n"
            + "node[shape = note fillcolor = \"#F8DEA1\" style = filled]\n"
            + "subgraph cluster_p{\n"
            + "label = \"Cola Recepción\"\n"
            + "bgcolor = \"#8ECBE5\"\n"
            + "raiz[label = \"Recepción\" shape = folder]\n"
            + "edge[dir = \"right\"]\n";
    String nodos = "", apuntador_nodo = "", rank = "{rank = same;raiz";
    Cliente actual = this.getCabeza();
    if (actual != null) {
        contenido += "raiz -> nodo" + id + ";\n";
    }
    for (actual = this.getCabeza(); actual != null; actual = actual.getSiguiente()) {
        nodos += "nodo" + id + "[label = \"" + actual.getKey_titulo() + "\nIMG Color: " + actual.getColor()
                + "\nIMG ByN: " + actual.getByN() + "\", fillcolor = \"#FCF8F7\", group = " + (id + 1) + "]\n";
        if (actual.getSiguiente() != null) {
            apuntador_nodo += "nodo" + id + " -> " + " nodo" + (id + 1) + ";\n";
        }
        rank += ";nodo" + id;
        id++;

    }
    rank += "}\n";
    contenido += nodos + apuntador_nodo + rank + "}\n}";
    return contenido;
}
```

```
public String graficaColaColor() {
    mostrarColor();
    int id = 1;
    String contenido = "digraph L{\n"
            + "node[shape = note fillcolor = \"#F8DEA1\" style = filled]\n"
            + "subgraph cluster_p{\n"
            + "label = \"Cola Impresora Color\"\n"
            + "bgcolor = \"#8ECBE5\"\n"
            + "raiz[label = \"Impresora Color\" shape = folder]\n"
            + "edge[dir = \"right\"]\n";
    String nodos = "", apuntador_nodo = "", rank = "{rank = same;raiz";
    Imagen actual = this.getCabezaColor();
    if (actual != null) {
        contenido += "raiz -> nodo" + id + ";\n";
    }
    for (actual = this.getCabezaColor(); actual != null; actual = actual.getSiguiente()) {
        nodos += "nodo" + id + "[label = \"IMG Color \n Id Cliente: " + actual.getIdCliente() + "\", fi
        if (actual.getSiguiente() != null) {
            apuntador_nodo += "nodo" + id + " -> " + " nodo" + (id + 1) + ";\n";
        }
        rank += ";nodo" + id;
        id++;
    }
    rank += "}\n";
    contenido += nodos + apuntador_nodo + rank + "}\n}";
    return contenido;
}
```

```
public String graficaColaByN() {
    mostrarByN();
    int id = 1;
    String contenido = "digraph L{\n"
            + "node[shape = note fillcolor = \"#F8DEA1\" style = filled]\n"
            + "subgraph cluster_p{\n"
            + "label = \"Cola Impresora Blanco y Negro\"\n"
            + "bgcolor = \"#8ECBE5\"\n"
            + "raiz[label = \"Impresora Blanco Y Negro\" shape = folder]\n"
            + "edge[dir = \"right\"]\n";
    String nodos = "", apuntador_nodo = "", rank = "{rank = same;raiz";
    Imagen actual = this.getCabezaByN();
    if (actual != null) {
        contenido += "raiz -> nodo" + id + ";\n";
    }
    for (actual = this.getCabezaByN(); actual != null; actual = actual.getSiguiente()) {
        nodos += "nodo" + id + "[label = \"IMG Blanco y Negro \n Id Cliente: " + actual.getIdCliente() + "\", f
        if (actual.getSiguiente() != null) {
            apuntador_nodo += "nodo" + id + " -> " + " nodo" + (id + 1) + ";\n";
        }
        rank += ";nodo" + id;
        id++;
    }
    rank += "}\n";
    contenido += nodos + apuntador_nodo + rank + "}\n}";
    return contenido;
}
```

```
public String graficaClienteAtendido() {
    int id = 1;
    String contenido = "digraph L{\n"
            + "node[shape = note fillcolor = \"#F8DEA1\" style = filled]\n"
            + "subgraph cluster_p{\n"
            + "label = \"Lista Clientes Atendidos\"\n"
            + "bgcolor = \"#8ECBE5\"\n"
            + "raiz[label = \"Atendidos\" shape = folder]\n"
            + "edge[dir = \"right\"]\n";
    String nodos = "", apuntador_nodo = "", rank = "{rank = same;raiz";
    Cliente actual = this.getCabezaCliente();
    if (actual != null) {
        contenido += "raiz -> nodo" + id + ";\n";
    }
    for (actual = this.getCabezaCliente(); actual != null; actual = actual.getSiguiente()) {
        nodos += "nodo" + id + "[label = \"Id: " + actual.getId() + "\nNombre: " + actual.getNombre() + "\nIM
                + "\nIMG ByN: " + actual.getByN() + "\", fillcolor = \"#FCF8F7\", group = " + (id + 1) + "]\n
        if (actual.getSiguiente() != null) {
            apuntador_nodo += "nodo" + id + " -> " + " nodo" + (id + 1) + ";\n";
        }
        rank += ";nodo" + id;
        id++;
    }
    rank += "}\n";
    contenido += nodos + apuntador_nodo + rank + "}\n}";
    return contenido;
}
```

- Método solicitarInfoCliente

Método utilizado para obtener la información de un cliente en especifico para los reportesm para ello se recorre la lista de clientes atendidos y cuando el id solicitado sea igual a alguno de los clientes retorna el cliente es cuestion.

```
public Cliente solicitarInfoCliente(int idClienteSolicitado) {
    Cliente actual = getCabezaCliente();
    while (actual != null) {
        if (actual.getId() == idClienteSolicitado) {
            return actual;
        }
        actual = actual.getSiguiente();
    }
    return null;
}
```

- Método graficarListaVentanilla

Método utilizado para la graficacion en graphviz, para ello se crea una variable string con la base del archivo dot a generar, luego se crea una variable actual y se inicializa con la cabeza; luego mediante un ciclo for que recorre la lista se crean los componentes necesarios para el archivo, a su vez dentro del ciclo se accede a la pila de cada ventanilla para graficar los nodos de imágenes que contiene cada una de las ventanillas, la cual se obtiene mediante otro ciclo while. Al finalizar se retorna una variable string.

```
public String graficaListaVentanilla() {
    int id = 1;
    String nodoCliente = "", nodoImagen = "", filas = "", apuntador = "", apuntadorCliente = "", apuntadorImagen = "", rank =
    String contenido = "digraph L{\n"
            + "node[shape = note fillcolor = \"#F8DEA1\" style = filled]\n"
            + "subgraph cluster_p{\n"
            + "label = \"Lista de Ventanillas\"\n"
            + "bgcolor = \"#8ECBE5\"\n"
            + "edge[dir = \"right\"]\n";

    Ventanilla actual = this.getCabezaVen();
    for (actual = this.getCabezaVen(); actual != null; actual = actual.getSiguiente()) {

        Boolean primeraImagen = false;
        filas += "Ventanilla" + actual.getNumVentanilla() + "[label = \"Ventanilla " + actual.getNumVentanilla() + "\", fillc

        if (actual.getCliente() != null && actual.isOcupada()) {
            nodoCliente += actual.getCliente().getKey_titulo() + "[label = \"Cliente " + actual.getId_cliente() + "\nIMG Colo
                    + "\nIMG ByN: " + actual.getCliente().getByN() + "\", fillcolor = \"#FCF8F7\"]\n";
        }

        if (actual.getCliente() != null && actual.isOcupada()) {
            apuntadorCliente += actual.getCliente().getKey_titulo() + " -> " + " Ventanilla" + actual.getNumVentanilla() + ";
            rank += ";Ventanilla" + actual.getNumVentanilla() + ";" + actual.getCliente().getKey_titulo();
        }
```

```
    if (actual.getPilaImagen() != null) {
        Imagen actualImg = actual.getPilaImagen().getCabeza();
        while (actualImg != null) {
            if (actualImg.isTipoImpresion()) {
                nodoImagen += "nodo" + id + "[label = \"IMG Color\", fillcolor = \"#FCF8F7\"]\n";
            } else {
                nodoImagen += "nodo" + id + "[label = \"IMG ByN\", fillcolor = \"#FCF8F7\"]\n";
            }
            if (!primeraImagen) {
                apuntadorImagen += " Ventanilla" + actual.getNumVentanilla() + " -> " + " nodo" + id + ";\n";
                primeraImagen = true;
            }
            if (actualImg.getSiguiente() != null) {
                apuntadorImagen += " nodo" + id + " -> " + " nodo" + (id + 1) + ";\n";
            }
            rank += ";nodo" + id;
            id++;
            actualImg = actualImg.getSiguiente();
        }
        rank += "}\n";
    }

    if (actual.getSiguiente() != null) {
        apuntador += "Ventanilla" + actual.getNumVentanilla() + " -> " + " Ventanilla" + actual.getSiguiente().g
        rank += "{rank = same";
    }
}
contenido += filas + apuntador + nodoCliente + apuntadorCliente + nodoImagen + apuntadorImagen + rank + "}\n}";
return contenido;
```

- Método BubbleSortColor, BubbleSortByN, BubbleSortPasos

Este método es usado para ordenar los nodos de la lista de clientes atendidos para los reportes según por el tipo de valor que se desea ordenar, para ello se realiza un ciclo while infinito el cual acaba cuando no se produzca ningún cambio, dentro de este se hace otro ciclo while el cual recorre al siguiente del nodo cabeza, luego se valida si el valor actual es menor o mayor al valor siguiente si es así se valida si el anterior no es igual a nulo, se cambian los respectivos valores, si no se cambian otros, y la bandera cambio se vuelve positiva.

```
public void BubbleSortColor() {
    Cliente actual = this.getCabezaCliente();
    while (true) {
        actual = this.getCabezaCliente();
        Cliente auxAnt = null;
        Cliente auxSig = actual.getSiguiente();
        Boolean cambio = false;
        while (auxSig != null) {
            if (actual.getColor() < auxSig.getColor()) {
                cambio = true;
                Cliente tmp = auxSig.getSiguiente();
                if (auxAnt != null) {
                    auxAnt.setSiguiente(auxSig);
                    auxSig.setSiguiente(actual);
                    actual.setSiguiente(tmp);
                } else {
                    this.setCabezaCliente(auxSig);
                    auxSig.setSiguiente(actual);
                    actual.setSiguiente(tmp);
                }
                auxAnt = auxSig;
                auxSig = actual.getSiguiente();
            } else {
                auxAnt = actual;
                actual = auxSig;
                auxSig = auxSig.getSiguiente();
            }
        }
        if (!cambio) {
            break;
        }
    }
}
```

```
public void BubbleSortByN() {
    Cliente actual = this.getCabezaCliente();
    while (true) {
        actual = this.getCabezaCliente();
        Cliente auxAnt = null;
        Cliente auxSig = actual.getSiguiente();
        Boolean cambio = false;
        while (auxSig != null) {
            if (actual.getByN() > auxSig.getByN()) {
                cambio = true;
                Cliente tmp = auxSig.getSiguiente();
                if (auxAnt != null) {
                    auxAnt.setSiguiente(auxSig);
                    auxSig.setSiguiente(actual);
                    actual.setSiguiente(tmp);
                } else {
                    this.setCabezaCliente(auxSig);
                    auxSig.setSiguiente(actual);
                    actual.setSiguiente(tmp);
                }
                auxAnt = auxSig;
                auxSig = actual.getSiguiente();
            } else {
                auxAnt = actual;
                actual = auxSig;
                auxSig = auxSig.getSiguiente();
            }
        }
        if (!cambio) {
            break;
        }
    }
}
```

```
public void BubbleSortPasos() {
    Cliente actual = this.getCabezaCliente();
    while (true) {
        actual = this.getCabezaCliente();
        Cliente auxAnt = null;
        Cliente auxSig = actual.getSiguiente();
        Boolean cambio = false;
        while (auxSig != null) {
            if (actual.getCantidadPasos() < auxSig.getCantidadPasos()) {
                cambio = true;
                Cliente tmp = auxSig.getSiguiente();
                if (auxAnt != null) {
                    auxAnt.setSiguiente(auxSig);
                    auxSig.setSiguiente(actual);
                    actual.setSiguiente(tmp);
                } else {
                    this.setCabezaCliente(auxSig);
                    auxSig.setSiguiente(actual);
                    actual.setSiguiente(tmp);
                }
                auxAnt = auxSig;
                auxSig = actual.getSiguiente();
            } else {
                auxAnt = actual;
                actual = auxSig;
                auxSig = auxSig.getSiguiente();
            }
        }
        if (!cambio) {
            break;
        }
    }
}
```

- Método cargaMasivaJson

Este método es utilizado para realizar la carga masiva del archivo json, para ello se hace la instancia del método leerArchivoJson, luego se hace uso de la librería gson y se hace un parser de la información obtenido, luego se convierte el Json de tipo elemento a tipo objeto. Luego se hace uso de las librería java.util y los métodos Set y Map los cuales retornan los miembros del objeto; mediante un ciclo for – each el cual se le establece el tipo de dato y el Json Element que se debe recorrer, seguido de esto se hace uso del método getKey el cual devuelve las claves del json, para obtener los valores se concatena a una variable string el contenido del método getValue más unos corchetes, luego esta variable se realiza un parse con un JsonArray, seguido de esto se recorre el JsonArray y se obtiene los atributos y se hace la instancia del método insertarCliente.

```
public boolean cargaMasivaJson() {
    String Jsontxt = Archivo.leerArchivoJson();
    JsonParser parser = new JsonParser();
    JsonElement element = parser.parse(Jsontxt); // Convertir el texto a JsonElement
    JsonObject obj = element.getAsJsonObject(); //JsonElement a JsonObject
    Set<Map.Entry<String, JsonElement>> entries = obj.entrySet();//retorna miembros objeto

    for (Map.Entry<String, JsonElement> entry : entries) {
        String key = entry.getKey(); //Claves del Json
        String valor = "[" + entry.getValue().toString(); //Informacin del cliente
        valor += "]";
        JsonArray gsonArr = parser.parse(valor).getAsJsonArray(); //Info -> JsonArray
        for (JsonElement objt : gsonArr) {
            JsonObject gsonObj = objt.getAsJsonObject();
            int id_cliente = gsonObj.get("id_cliente").getAsInt();
            String nombre_cliente = gsonObj.get("nombre_cliente").getAsString();
            int img_color = gsonObj.get("img_color").getAsInt();
            int img_bw = gsonObj.get("img_bw").getAsInt();
            clientes.insertarCliente(key, id_cliente, nombre_cliente, img_color, img_bw);
        }
    }
    return true;
}
```

- Método crearGrafico

Este método es utilizado para escribir el archivo para generar el grafo, para ello dentro de un try – catch se hace la instancia del método FileWriter en el cual se escribe como parámetro el nombre del archivo, luego se escribe el archivo con el método PrintWriter y se cierra dicho archivo; seguido de esto se hace uso de la librería Runtime la cual sirve para acceder a los comandos como si fuese una consola, dentro de los parámetros de este se escribe el comando para convertir un archivo con extensión dot a png, luego se imprime un mensaje en consola.

```
public void crearGrafico(String contenidoGrafica, String nombre) {
    try {
        FileWriter archivo = new FileWriter(nombre + ".dot", false);
        PrintWriter pw = new PrintWriter(archivo);
        pw.write(contenidoGrafica);
        pw.close();
        archivo.close();
        Runtime.getRuntime().exec("dot -Tpng " + nombre + ".dot -o " + nombre + ".png");
        System.out.println("Gráfica generada exitosamente");

    } catch (IOException e) {
        System.out.println("Error Archivo: " + e.getMessage());
    }
}
```