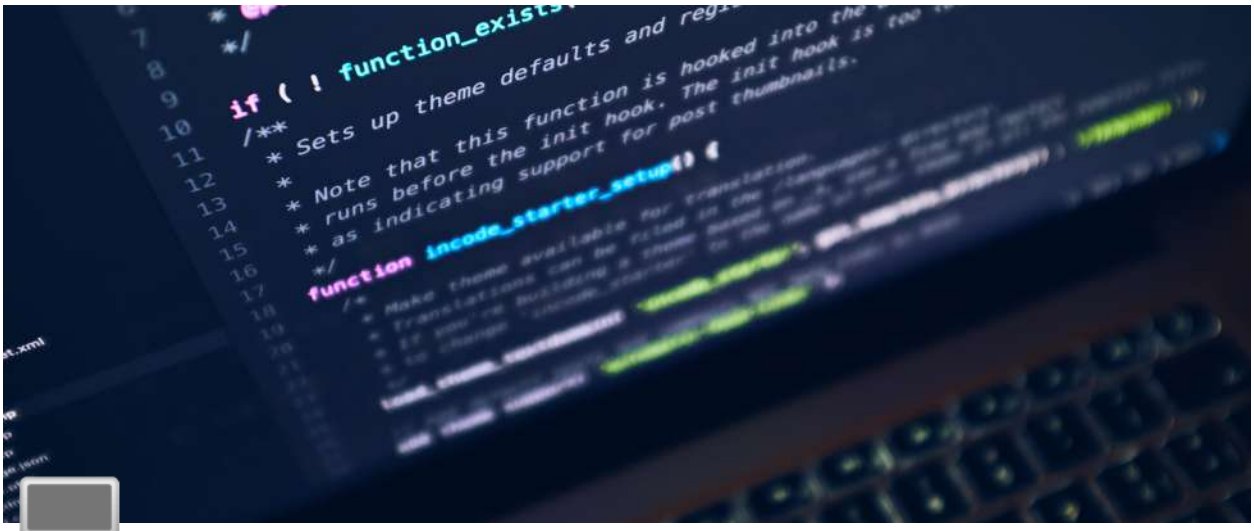


Manual Técnico Proyecto 2 - Estructura De Datos

Compatibilidad

-Windows 7,8,10

-Linux

Especificaciones

En esta sección se mencionará lo que debes conocer de la aplicación.

Lenguaje Utilizado

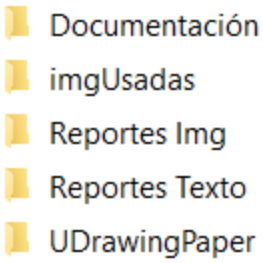
El lenguaje utilizado para la realización de este proyecto fue Java.

IDE y JDK utilizado

El IDE utilizado fue NetBeans en su versión 8.2, y el JDK utilizado fue 1.8.0

Estructura Del Proyecto

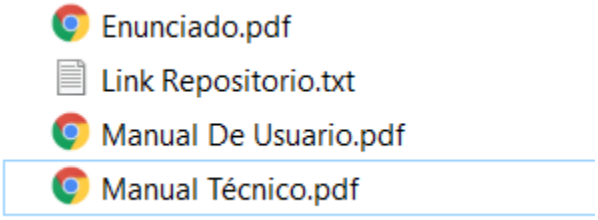
La estructura inicial del proyecto es la siguiente:



Documentación

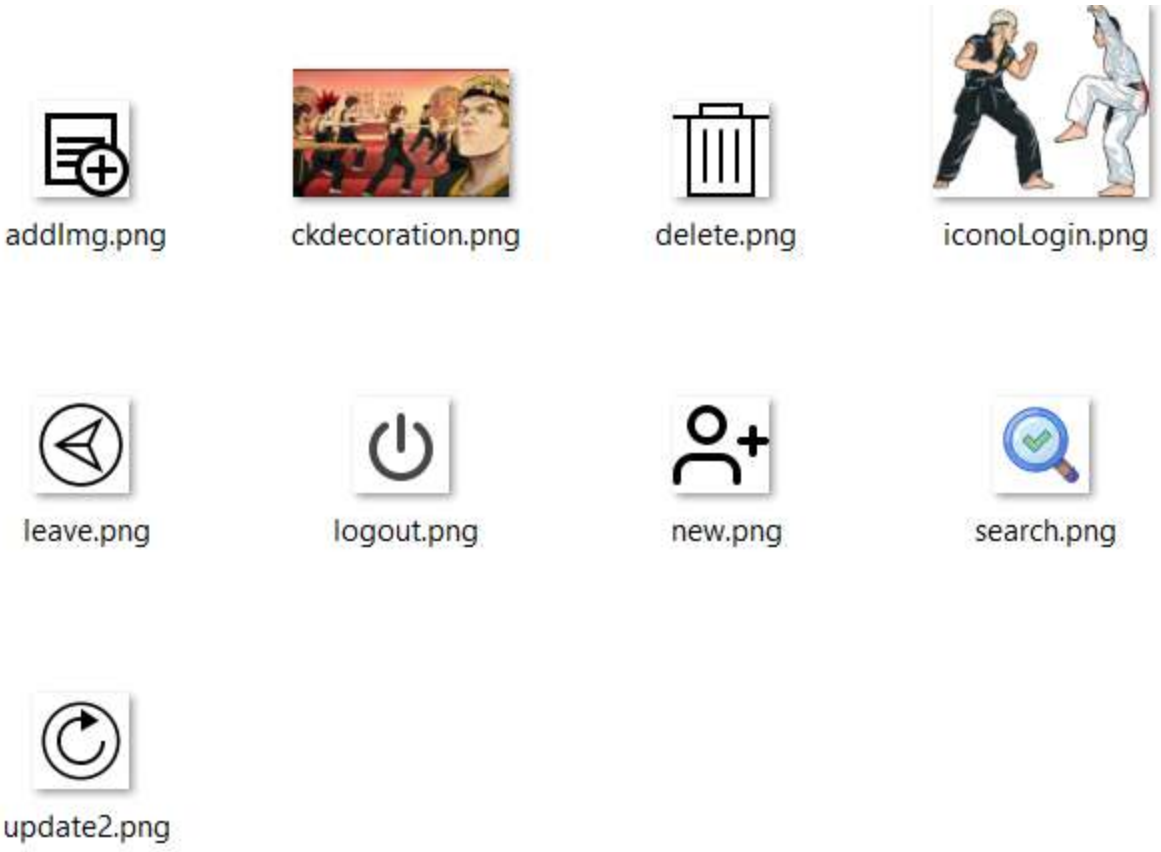
En esta carpeta se encuentra:

- Enunciado del Proyecto
- Manual de usuario
- Manual técnico
- Link al repositorio



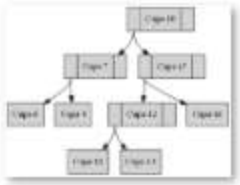
Img Usadas

En esta carpeta se encuentran todas las imágenes utilizadas en la interfaz gráfica

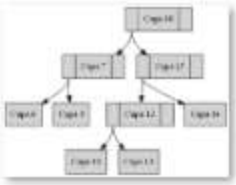


Reportes Img

Carpeta que contiene todas las imágenes generadas por la aplicación, árboles, capas e imágenes.

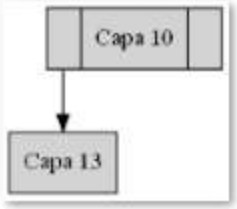

AbbErwin.png


Abbkathy.png


Abbq.png

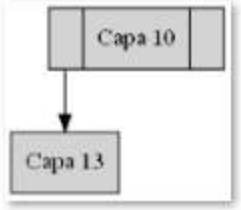

albumErwin.png


ARBOLB.png

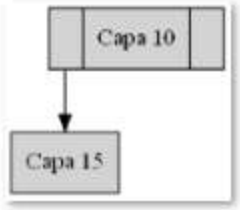


ArbolImagen1Erwin.png

ArbolImagen1kathy.png

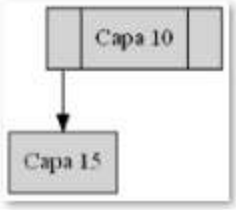


ArbolImagen1q.png

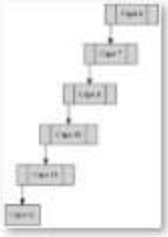


ArbolImagen2Erwin.png

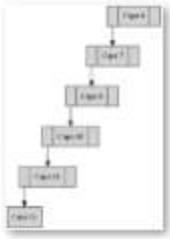
ArbolImagen2kathy.png



ArbolImagen2q.png


ArbolImagen3Erwin.png


ArbolImagen3kathy.png


ArbolImagen3q.png

ArbolImagen4Erwin.png

ArbolImagen4kathy.png

Reportes Texto

Carpeta que contiene todo el código en archivos “txt” generado para graficar en graphviz.

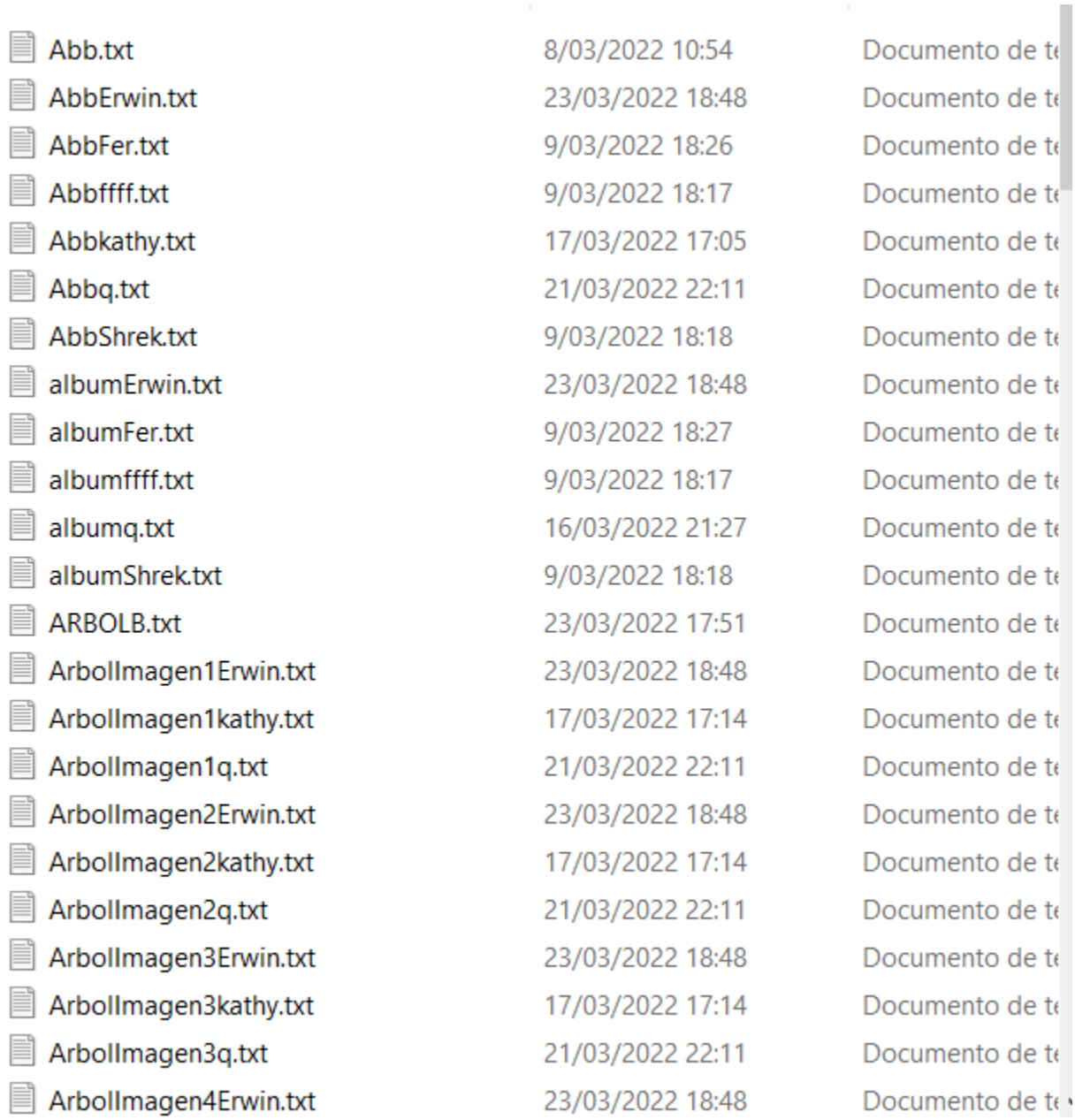
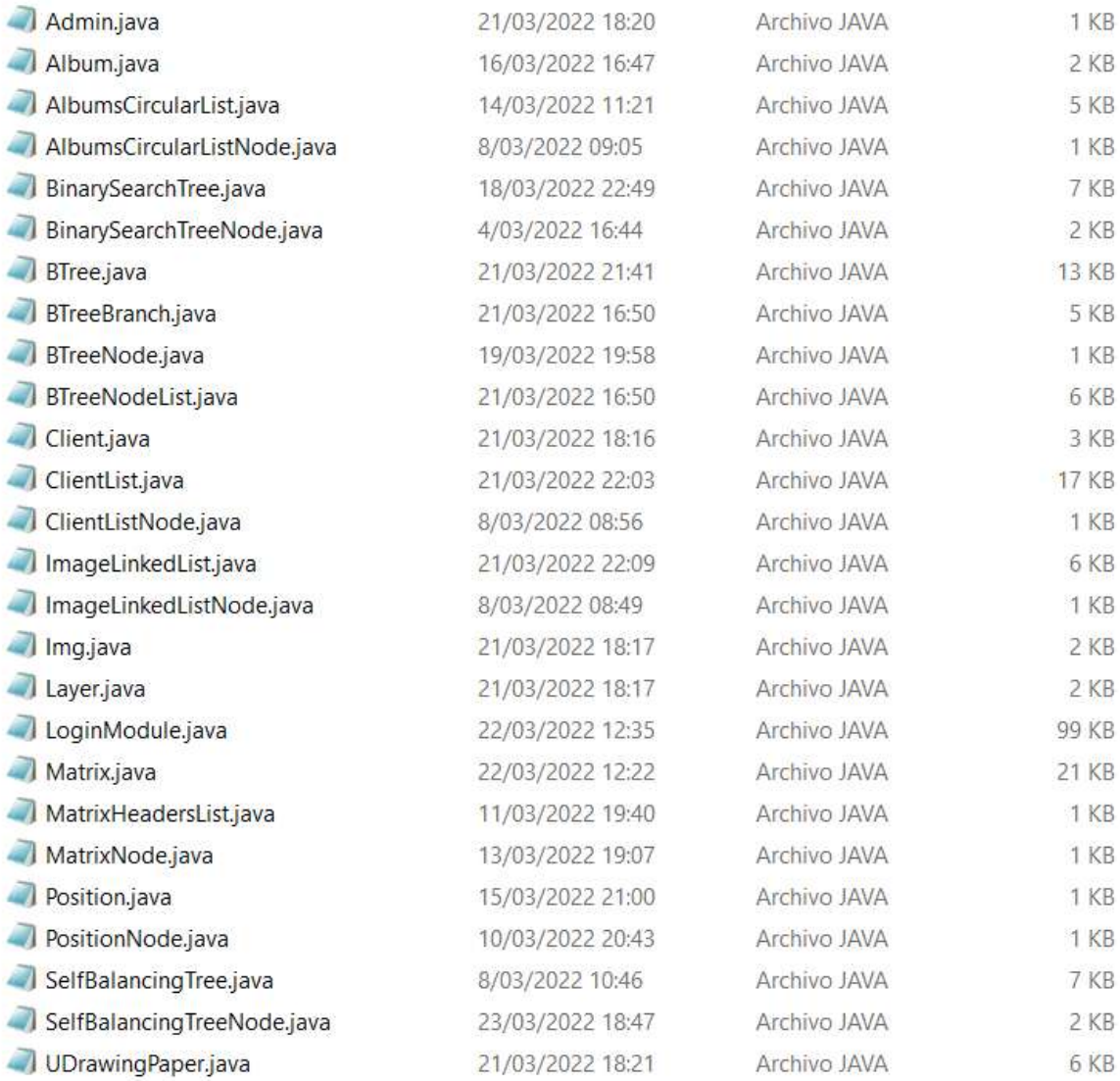


Abb.txt	8/03/2022 10:54	Documento de te
AbbErwin.txt	23/03/2022 18:48	Documento de te
AbbFer.txt	9/03/2022 18:26	Documento de te
Abbffff.txt	9/03/2022 18:17	Documento de te
Abbkathy.txt	17/03/2022 17:05	Documento de te
Abbq.txt	21/03/2022 22:11	Documento de te
AbbShrek.txt	9/03/2022 18:18	Documento de te
albumErwin.txt	23/03/2022 18:48	Documento de te
albumFer.txt	9/03/2022 18:27	Documento de te
albumffff.txt	9/03/2022 18:17	Documento de te
albumq.txt	16/03/2022 21:27	Documento de te
albumShrek.txt	9/03/2022 18:18	Documento de te
ARBOLB.txt	23/03/2022 17:51	Documento de te
ArbolImagen1Erwin.txt	23/03/2022 18:48	Documento de te
ArbolImagen1kathy.txt	17/03/2022 17:14	Documento de te
ArbolImagen1q.txt	21/03/2022 22:11	Documento de te
ArbolImagen2Erwin.txt	23/03/2022 18:48	Documento de te
ArbolImagen2kathy.txt	17/03/2022 17:14	Documento de te
ArbolImagen2q.txt	21/03/2022 22:11	Documento de te
ArbolImagen3Erwin.txt	23/03/2022 18:48	Documento de te
ArbolImagen3kathy.txt	17/03/2022 17:14	Documento de te
ArbolImagen3q.txt	21/03/2022 22:11	Documento de te
ArbolImagen4Erwin.txt	23/03/2022 18:48	Documento de te

UDrawingPaper

carpeta que contiene todo el código fuente de este proyecto.



Admin.java	21/03/2022 18:20	Archivo JAVA	1 KB
Album.java	16/03/2022 16:47	Archivo JAVA	2 KB
AlbumsCircularList.java	14/03/2022 11:21	Archivo JAVA	5 KB
AlbumsCircularListNode.java	8/03/2022 09:05	Archivo JAVA	1 KB
BinarySearchTree.java	18/03/2022 22:49	Archivo JAVA	7 KB
BinarySearchTreeNode.java	4/03/2022 16:44	Archivo JAVA	2 KB
BTree.java	21/03/2022 21:41	Archivo JAVA	13 KB
BTreeBranch.java	21/03/2022 16:50	Archivo JAVA	5 KB
BTreeNode.java	19/03/2022 19:58	Archivo JAVA	1 KB
BTreeNodeList.java	21/03/2022 16:50	Archivo JAVA	6 KB
Client.java	21/03/2022 18:16	Archivo JAVA	3 KB
ClientList.java	21/03/2022 22:03	Archivo JAVA	17 KB
ClientListNode.java	8/03/2022 08:56	Archivo JAVA	1 KB
ImageLinkedList.java	21/03/2022 22:09	Archivo JAVA	6 KB
ImageLinkedListNode.java	8/03/2022 08:49	Archivo JAVA	1 KB
Img.java	21/03/2022 18:17	Archivo JAVA	2 KB
Layer.java	21/03/2022 18:17	Archivo JAVA	2 KB
LoginModule.java	22/03/2022 12:35	Archivo JAVA	99 KB
Matrix.java	22/03/2022 12:22	Archivo JAVA	21 KB
MatrixHeadersList.java	11/03/2022 19:40	Archivo JAVA	1 KB
MatrixNode.java	13/03/2022 19:07	Archivo JAVA	1 KB
Position.java	15/03/2022 21:00	Archivo JAVA	1 KB
PositionNode.java	10/03/2022 20:43	Archivo JAVA	1 KB
SelfBalancingTree.java	8/03/2022 10:46	Archivo JAVA	7 KB
SelfBalancingTreeNode.java	23/03/2022 18:47	Archivo JAVA	2 KB
UDrawingPaper.java	21/03/2022 18:21	Archivo JAVA	6 KB

Ahora se explican las diferentes estructuras de datos utilizadas

Estructuras De Datos

Lista Circular Doblemente Enlazada

Esta lista guarda en cada nodo un álbum, que a su vez cada álbum tiene una lista enlazada simple d imágenes.

Nodo

```
 */
public class AlbumsCircularListNode {
    public Album album;
    public AlbumsCircularListNode next;
    public AlbumsCircularListNode previous;
    public AlbumsCircularListNode(Album album){
        this.album=album;
    }
}
```

¿Está vacía?

```
public boolean isEmpty(){
        return first==null;
    }
```

Insertar

```
public void finalInsert(Album album){
    AlbumsCircularListNode node= new AlbumsCircularListNode(album);
    if(first==null){
    first=node;
    }else{
        AlbumsCircularListNode pointer= first;
        while(pointer.next!=null){
            pointer=pointer.next;
            }
        pointer.next=node;
    }

}
```

Graficar

```
public String generatePersonalizeAlbum(String name) throws IOException{
    String route="../Reportes Texto/album"+name+".txt";
    String graph="../Reportes Img/album"+name+".png";
    String tParam = "-Tpng";
    String tOParam = "-o";
    String pathString = "C:\\Program Files\\Graphviz\\bin\\dot.exe";

    String finalText="digraph G{\nnode [shape=box];\n";
    AlbumsCircularListNode temp = first;
    String rankSame="{rank=same; ";
    String conections="";
    String nodes="";
    while(temp != null){
        nodes+="N"+temp.hashCode()+"[label=\""+"Nombre Del Album: \n"+temp.album.getName()+"\"];\n";
        if(!temp.album.getImgList().isEmpty()){
            nodes+=temp.album.getImgList().collectLinkedList();
            conections+="N"+temp.hashCode()+ " -> ";
            conections+=temp.album.getImgList().collectConnections();
            //conections+="start"+ " -> "+"N"+temp.hashCode()+";\n";
        }
        if (temp.next!=null){
            conections+="N"+temp.hashCode()+ " -> "+"N"+temp.next.hashCode()+";\n";
            conections+="N"+temp.next.hashCode()+ " -> "+"N"+temp.hashCode()+";\n";
            rankSame+="N"+temp.hashCode()+",";
        }else{
            conections+="N"+temp.hashCode()+ " -> "+"N"+first.hashCode()+";\n";
            conections+="N"+first.hashCode()+ " -> "+"N"+temp.hashCode()+";\n";
            rankSame+="N"+temp.hashCode();
        }

        temp = temp.next;

    }
    rankSame+="};";
```

```
    finalText+=nodes+"\n";
    finalText+=conections+"\n";
    finalText+="start [shape=Mdiamond label=\"Lista De Álbumes Del Cliente:\n "+name+"\"];";
    finalText+=rankSame;
    finalText+="}\n}";
    FileWriter fw = new FileWriter(route);
    BufferedWriter bw = new BufferedWriter(fw);
    bw.write(finalText);
    bw.close();

    String[] cmd = new String[5];
    cmd[0] = pathString;
    cmd[1] = tParam;
    cmd[2] = route;
    cmd[3] = tOParam;
    cmd[4] = graph;

    Runtime rt = Runtime.getRuntime();

    rt.exec( cmd );

    return finalText;
}
```

Lista Enlazada Simple

Esta lista tiene como objetivo guardar las imágenes de cada álbum, por lo que cada álbum posee una de estas listas.

Nodo

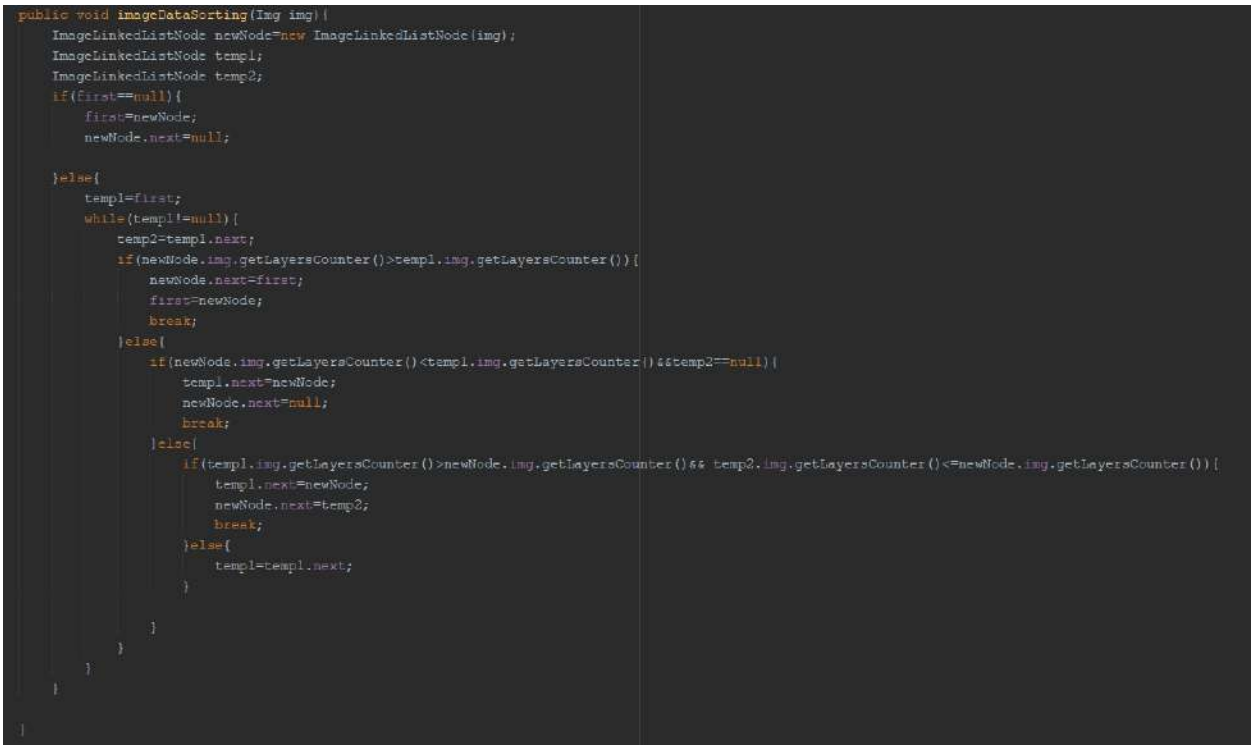
```
public class ImageLinkedListNode {
    public Img img;
    public ImageLinkedListNode next = null;
    public ImageLinkedListNode(Img img){
            this.img = img;
        }

}
```

Insertar

```
public void insert(Img img){
    ImageLinkedListNode node = new ImageLinkedListNode(img);
    node.next=first;
    first=node;
    countImg++;
}
```

Insertar De Forma Ordenada



```
public void imageDataSorting(Img img){
    ImageLinkedListNode newNode=new ImageLinkedListNode(img);
    ImageLinkedListNode temp1;
    ImageLinkedListNode temp2;
    if(first==null){
        first=newNode;
        newNode.next=null;

    }else{
        temp1=first;
        while(temp1!=null){
            temp2=temp1.next;
            if(newNode.img.getLayersCounter()>temp1.img.getLayersCounter()){
                newNode.next=first;
                first=newNode;
                break;
            }else{
                if(newNode.img.getLayersCounter()<temp1.img.getLayersCounter()&&temp2==null){
                    temp1.next=newNode;
                    newNode.next=null;
                    break;
                }else{
                    if(temp1.img.getLayersCounter()>newNode.img.getLayersCounter()&& temp2.img.getLayersCounter()<=newNode.img.getLayersCounter()){
                        temp1.next=newNode;
                        newNode.next=temp2;
                        break;
                    }else{
                        temp1=temp1.next;
                    }

                }
            }
        }
    }

}
```

Matriz Dispersa

Esta matriz nos sirve para guardar la información de las capas y las imágenes, recibe en cada nodo las coordenadas de posicionamiento y un color, además de un contador y un grupo, pero estos 2 últimos mencionados, se usan para la generación del grafo de la misma.

Nodo

```java
public class MatrixNode {
    public int x;
    public int y;
    public String color;
    public MatrixNode up;
    public MatrixNode down;
    public MatrixNode next;
    public MatrixNode previous;
    public int counter;
    public int group;

    public MatrixNode(int x, int y,String color,int counter,int group ){
        this.x=x;
        this.y=y;
        this.color=color;
        up=down=next=previous=null;
        this.counter=counter;
        this.group=group;
    }

    public int getCounter() {
        return counter;
    }

    public void setCounter(int counter) {
        this.counter = counter;
    }

    public int getGroup() {
        return group;
    }

    public void setGroup(int group) {
        this.group = group;
    }


}
```

Insertar

```java
//Método de la inserción a la matriz
public String insert(int x ,int y,String color){
    MatrixNode newNode=new MatrixNode(x,y,color,-1,-1);
    MatrixNode columnNode=searchColumn(x);
    MatrixNode rowNode= searchRow(y);
    //Caso 1: No existe fila ni columna
    if(rowNode==null && columnNode==null){
        //System.out.println("Caso 1");
        //Creación de cabeceras
        columnNode=columnCreation(x);
        rowNode=rowCreation(y);
        //Inserción a la matriz
        newNode=orderInsertInColumn(newNode, rowNode);
        newNode=orderInsertInRow(newNode, columnNode);
        return "";
    }
    //Caso 2: No existe la fila, pero si la columna
    else if(rowNode==null && columnNode !=null){
        //Creamos las cabeceras
        rowNode=rowCreation(y);
        //Se agrega el contenido.
        newNode=orderInsertInColumn(newNode, rowNode);
        newNode=orderInsertInRow(newNode, columnNode);
        return "";
    }
```

```
//Caso 3: Existe fila pero no columna
else if(rowNode!=null && columnNode==null){
    //System.out.println("Caso 3");
    //Creamos las cabeceras
    columnNode=columnCreation(x);
    //Agregamos el contenido
    newNode=orderInsertInColumn(newNode, rowNode);
    newNode=orderInsertInRow(newNode, columnNode);
    return "";

}
//caso 4: Si existe fila y columna
else {
    //System.out.println("caso 4");
    //Creamos las cabeceras
    //No hay jeje
    //Agregamos el contenido
    newNode=orderInsertInColumn(newNode, rowNode);
    newNode=orderInsertInRow(newNode, columnNode);
    return "";
}
```

Árbol ABB

Este árbol guarda las capas de cada cliente, cada cliente tiene uno de estos árboles, además cada imagen posee uno de estos, ya que cada imagen tiene sus propias capas.

Nodo

```java
public class BinarySearchTreeNode {
    Layer layer;
    BinarySearchTreeNode left=null;
    BinarySearchTreeNode right=null;
    private static int counter=1;
    private  final int id;

    public BinarySearchTreeNode(Layer layer){
            this.layer = layer;
            this.left=null;
            this.right=null;
            this.id=counter++;
        }
```

Insertar

```java
//Inseción al árbol de búsqueda binario
public void insert(Layer layer){
    root=recursiveInsert(layer, root);
}
public BinarySearchTreeNode recursiveInsert(Layer layer,BinarySearchTreeNode root){
    if (root==null){
        BinarySearchTreeNode newNode= new BinarySearchTreeNode(layer);
        return newNode;
    }else{
        if (layer.getId() < root.layer.getId()){
            root.left=recursiveInsert(layer,root.left);
        }else{
            root.right=recursiveInsert(layer, root.right);
        }
    }
    return root;
}
```

Profundidad Del Árbol

```
public int depth(BinarySearchTreeNode root){
    if(root!=null){
        return 1+ Math.max(depth(root.left), depth(root.right));
    }
    return 0;

}
```

Pre-Orden

```
public String preOrder(BinarySearchTreeNode root){

    if(root!=null){
        preOrder+=root.layer.getId()+",";
        preOrder(root.left);
        preOrder(root.right);
    }
    return preOrder;

}
```

In-Orden

```
public String inOrder(BinarySearchTreeNode root){

    if(root!=null){
        inOrder(root.left);
        inOrder+=root.layer.getId()+",";
        inOrder(root.right);
    }
    return inOrder;

}
```

Post-Orden

```
public String postOrder(BinarySearchTreeNode root){

    if(root!=null){

        postOrder(root.left);
        postOrder(root.right);
        postOrder+=root.layer.getId()+",";
    }
    return postOrder;
}
```

Código A Graficar

```
public String bstTreeGraphvizCode(){
    String code="";
    if(left==null && right==null){
        code+="N"+id+" [ label =\""+"Capa "+layer.getId()+"\"];\n";
    }else{
        code="N"+id+" [ label =\"<C0>|"+"Capa "+layer.getId()+"|<C1>\"];\n";
    }
    if(left!=null){
        code=code+left.bstTreeGraphvizCode()+"N"+id+":C0->N"+left.id+"\n";
    }
    if(right!=null){
       code=code+right.bstTreeGraphvizCode()+"N"+id+":C0->N"+right.id+"\n";
        }
    return code;

}
```

Árbol AVL

Árbol que guarda todas las imágenes de los clientes, por lo tanto cada cliente tiene un árbol AVL.

Nodo

```java
public class SelfBalancingTreeNode {
    Img img;
    SelfBalancingTreeNode left;
    SelfBalancingTreeNode right;
    int equilibrium;
    private static int counter=1;
    private  final int id;

    public  SelfBalancingTreeNode (Img img){
        this.img=img;
        this.equilibrium=0;
        this.left=null;
        this.right=null;
        this.id=counter++;

    }
```

Rotación Izquierda

```java
//rotación izquierda
public SelfBalancingTreeNode leftRotation(SelfBalancingTreeNode node){
   SelfBalancingTreeNode temp=node.left;
   node.left=temp.right;
   temp.right=node;
   node.equilibrium=Math.max(getEquilibrium(node.left), getEquilibrium(node.right))+1;
   temp.equilibrium=Math.max(getEquilibrium(temp.left), getEquilibrium(temp.right))+1;
   return temp;
}
```

Rotación Derecha

```
//rotación derecha
public SelfBalancingTreeNode rightRotation(SelfBalancingTreeNode node){
    SelfBalancingTreeNode temp=node.right;
    node.right=temp.left;
    temp.left=node;
    node.equilibrium=Math.max(getEquilibrium(node.left), getEquilibrium(node.right))+1;
    temp.equilibrium=Math.max(getEquilibrium(temp.left), getEquilibrium(temp.right))+1;
    return temp;
}
```

Rotación Doble Izquierda

```
public SelfBalancingTreeNode doubleLeftRotation(SelfBalancingTreeNode node){
    SelfBalancingTreeNode temp;
    node.left=rightRotation(node.left);
    temp=leftRotation(node);
    return temp;
}
```

Rotación Doble Derecha

```
public SelfBalancingTreeNode doubleRightRotation(SelfBalancingTreeNode node){
    SelfBalancingTreeNode temp;
    node.right=leftRotation(node.right);
    temp=rightRotation(node);
    return temp;
}
```

Insertar

```
//Método Insertar
public SelfBalancingTreeNode recursiveInsert(SelfBalancingTreeNode newNode,SelfBalancingTreeNode subTree){
    SelfBalancingTreeNode newDad=subTree;
    if(newNode.img.getId()<subTree.img.getId()){
        if(subTree.left==null){
            subTree.left=newNode;
        }else{
            subTree.left=recursiveInsert(newNode, subTree.left);
            if((getEquilibrium(subTree.left)- getEquilibrium(subTree.right))==2){
                if(newNode.img.getId()<subTree.left.img.getId()){
                    newDad=leftRotation(subTree);
                }else{
                    newDad=doubleLeftRotation(subTree);
                }
            }
        }
    }else if(newNode.img.getId()>subTree.img.getId()){
        if(subTree.right==null){
            subTree.right=newNode;
```

```
        }else{
            subTree.right=recursiveInsert(newNode, subTree.right);
            if((getEquilibrium(subTree.right)-getEquilibrium(subTree.left))==2){
                if(newNode.img.getId()>subTree.right.img.getId()){
                    newDad=rightRotation(subTree);

                }else{
                    newDad=doubleRightRotation(subTree);
                }
            }
        }

    }else{
        System.out.println("Nodo Duplicado.");
    }
    //Actualizando Equilibrio
    if((subTree.left==null)&&(subTree.right!=null)){
        subTree.equilibrium=subTree.right.equilibrium+1;
    }else if((subTree.right==null)&&(subTree.left!=null)){
        subTree.equilibrium=subTree.left.equilibrium+1;
    }else{
        subTree.equilibrium=Math.max(getEquilibrium(subTree.left), getEquilibrium(subTree.right))+1;
    }
    return newDad;
}
```

Código A Graficar

```
public String selfBalalancingTreeGraphvizCode(){
    String code="";
    if(left==null && right==null){
        code+="N"+id+" [ label =\""+"IMG "+img.getId()+"\"];\n";
    }else{
        code="N"+id+" [ label =\"<C0>|"+"IMG "+img.getId()+"|<C1>\"];\n";
    }
    if(left!=null){
        code=code+left.selfBalalancingTreeGraphvizCode()+"N"+id+":C0->N"+left.id+"\n";
    }
    if(right!=null){
       code=code+right.selfBalalancingTreeGraphvizCode()+"N"+id+":C0->N"+right.id+"\n";
        }
    return code;

}
```

Árbol B

Este árbol guarda a todos los clientes registrados en la aplicación, y solo el administrador puede generar su grafo.

Nodo

```java
public class BTreeNode {
    Client client;
    BTreeNode next;
    BTreeNode previous;
    BTreeBranch left;
    BTreeBranch right;

    public BTreeNode(Client client){
        this.client=client;
        this.next=null;
        this.previous=null;
        this.left=null;
        this.right=null;
    }

}
```

Rama

```
public class BTreeBranch {
    boolean root;
    int max;
    int min;
    int size;
    BTreeNodeList list;
    public BTreeBranch(){
        this.max=4;
        this.min=2;
        this.size=0;
        this.list=new BTreeNodeList();
    }
```

Lista De Nodos En Cada Rama

```
public class BTreeNodeList {
    BTreeNode head;
    BTreeNode end;
    int size;
    public BTreeNodeList(){
        this.head=null;
        this.end=null;
        this.size=0;
    }
```

Insertar

```
public void insert(Client client){
    BTreeNode newNode= new BTreeNode(client);
    if(root==null){
        root=new BTreeBranch();
        root.root=true;
        root=(BTreeBranch)root.insertInBranch(newNode);
    }else{
        if(height==0){
            Object response=root.insertInBranch(newNode);
            //Si la cabeza no se ha dividido:
            if(response instanceof BTreeBranch){
                //System.out.println("rama11");
                root=(BTreeBranch)response;
            }else{
                height++;
                root=new BTreeBranch();
                root=(BTreeBranch) root.insertInBranch( response);
            }
        }else{
            //Hay que validar si hay más de una rama para recorrer el arbol e insertar el nodo.
            if(root==null){
                System.out.println("Raiz Vacia");
                return;
            }
            Object response=travelInsert(newNode,root);
            if(response instanceof BTreeNode){//Se dividió root
                //System.out.println("Nodo11");
                height++;
                root=new BTreeBranch();
                root=(BTreeBranch)root.insertInBranch(response);
            }else if(response instanceof BTreeBranch){
                //System.out.println("rama22");
                root=(BTreeBranch)response;
            }
        }
    }
}
```