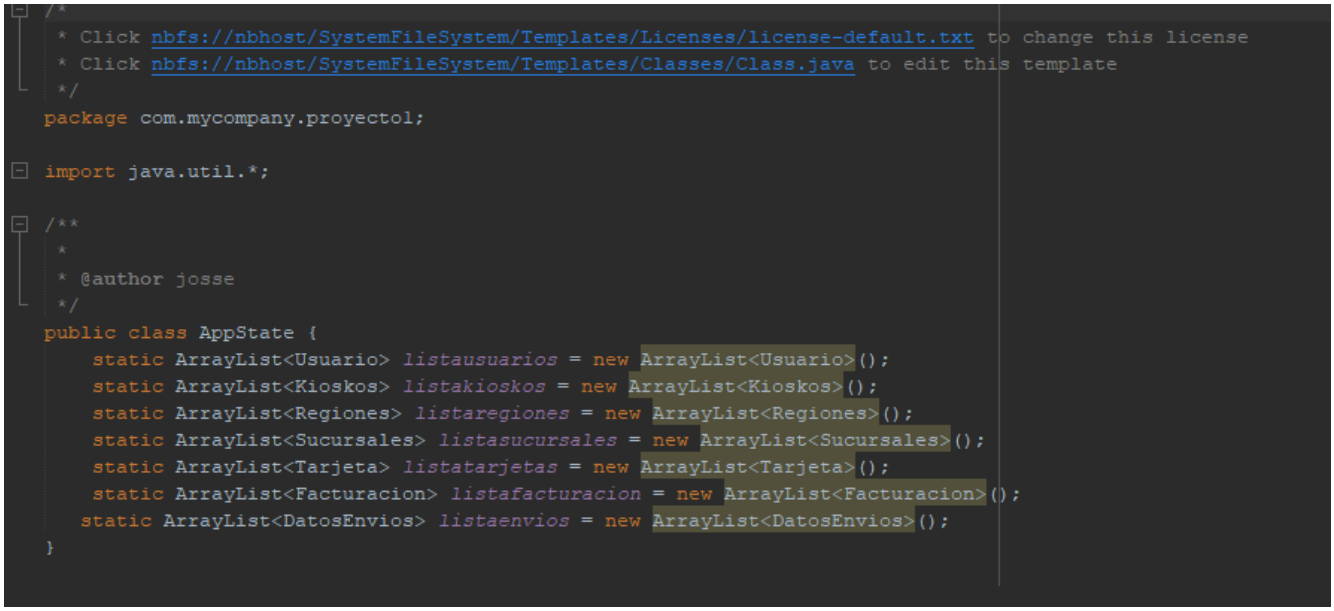
Nombre: Josseline Griselda Montecinos Hernández **Carné:** 202201534 **Curso:** IPC1 Sección A

MANUAL TÉCNICO

CLASE APPSTATE

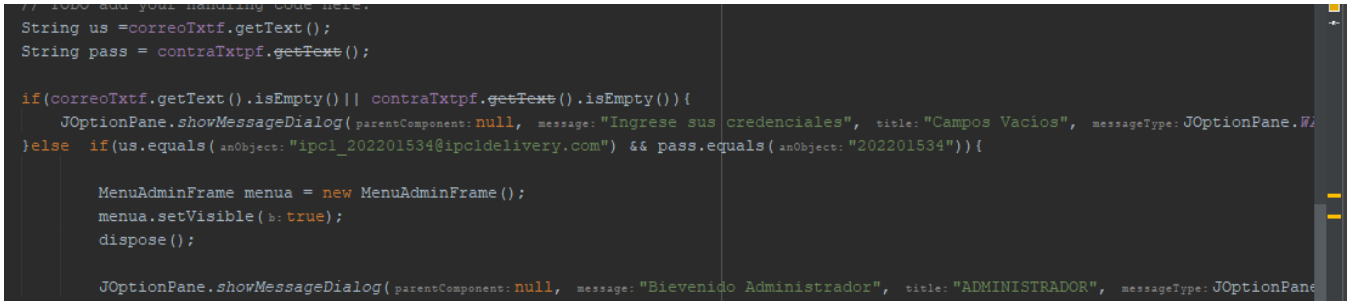
En esta parte del código se encuentran las listas estáticas de cada uno de los objetos manejados en todo el programa, cada una tiene asignado el nombre de lista + objeto relacionado. Para utilizar estas listas en el resto de clases se debe colocar AppState.nombredelalista.



```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package com.mycompany.proyectol;

import java.util.*;

/**
 *
 * @author josse
 */
public class AppState {
    static ArrayList<Usuario> listausuarios = new ArrayList<Usuario>();
    static ArrayList<Kioskos> listakioskos = new ArrayList<Kioskos>();
    static ArrayList<Regiones> listaregiones = new ArrayList<Regiones>();
    static ArrayList<Sucursales> listasucursales = new ArrayList<Sucursales>();
    static ArrayList<Tarjeta> listatarjetas = new ArrayList<Tarjeta>();
    static ArrayList<Facturacion> listafacturacion = new ArrayList<Facturacion>();
   static ArrayList<DatosEnvios> listaenvios = new ArrayList<DatosEnvios>();
}
```

AUTENTIFICACIÓN DE USUARIOS

- Las validaciones a través de los if-else se realizan luego de comprobar que todos los campos estén llenos, en esta parte la validación del usuario administrador es por defecto así que se iguala en contenido de las cajas de texto con el nombre y contraseña fijados.



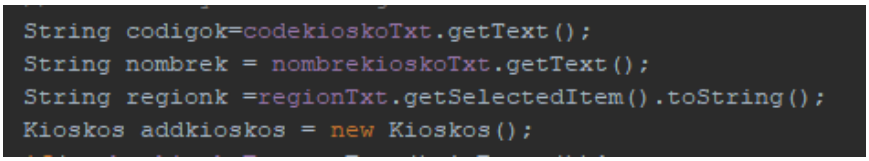
```
String us =correoTxtf.getText();
String pass = contraTxtpf.getText();

if(correoTxtf.getText().isEmpty()|| contraTxtpf.getText().isEmpty()){
    JOptionPane.showMessageDialog( parentComponent: null, message: "Ingrese sus credenciales", title: "Campos Vacios", messageType: JOptionPane.W
}else  if(us.equals( anObject: "ipcl_202201534@ipcldelivery.com") && pass.equals( anObject: "202201534")){

        MenuAdminFrame menua = new MenuAdminFrame();
        menua.setVisible( b: true);
        dispose();

        JOptionPane.showMessageDialog( parentComponent: null, message: "Bievenido Administrador", title: "ADMINISTRADOR", messageType: JOptionPane
```

- Si el usuario es distinto al tipo administrador, procederá a verificar por medio de una estructura for a lo largo de toda la lista de usuarios clientes registrados y si el usuario o contraseña coinciden, una variable booleana tomará el valor de true para realizar validaciones.

  En caso de que la variable booleana sea verdadera, se valida nuevamente en la lista de usuarios que ambas sean correctas para el usuario en la misma posición, ya que puede haber usuarios con contraseñas iguales, pero con correos y contraseñas iguales no lo habrá.
  Si no coinciden ambas credenciales se concluirá que los datos son incorrectos y caso de no cumplir con ninguna de las condiciones anteriores se concluirá que el usuario no existe.

MÉTODO PARA AGREGAR ELEMENTOS NUEVOS A LAS LISTAS

Se pasan los datos de las cajas de texto a variables del tipo que convenga.



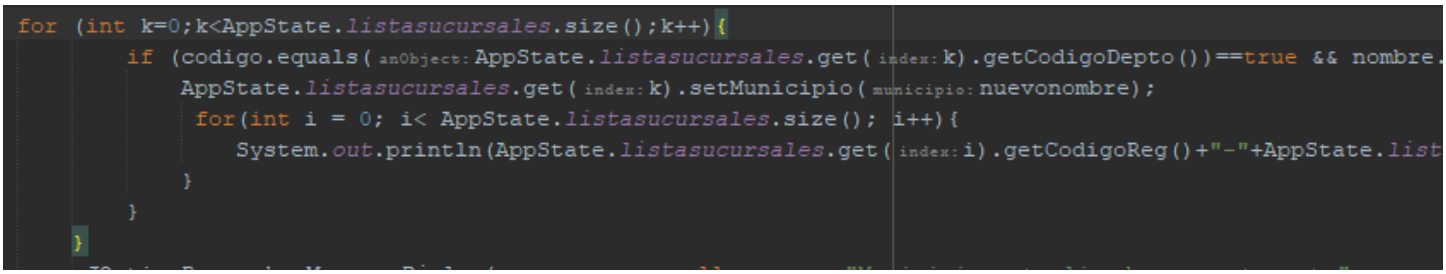
En esta parte también se crea un **objeto (addkiosko)** la Clase de Tipo Kiosko (Clase del objeto) que registraremos, en este caso es Kioskos ya que se agregarán los atributos de un Kiosko, pero se realiza lo mismo para las demás clases.

Para llenar el conjunto de datos de un único Kiosko, se utilizará el método set que se encuentra en la clase del Objeto al que le enviaremos lo que contengan nuestras variables, por ejemplo: **addkioskos.setNombrek(variable)** y para agregarlos a la lista utilizaremos le método add de la siguiente manera: **App.State.lista.add.(objeto)**

```
addkioskos.setCodigoK(codigok);
addkioskos.setNombreK(nombrek);
addkioskos.setRegionK(regionk);
AppState.listakioskos.add( e: addkioskos);
```

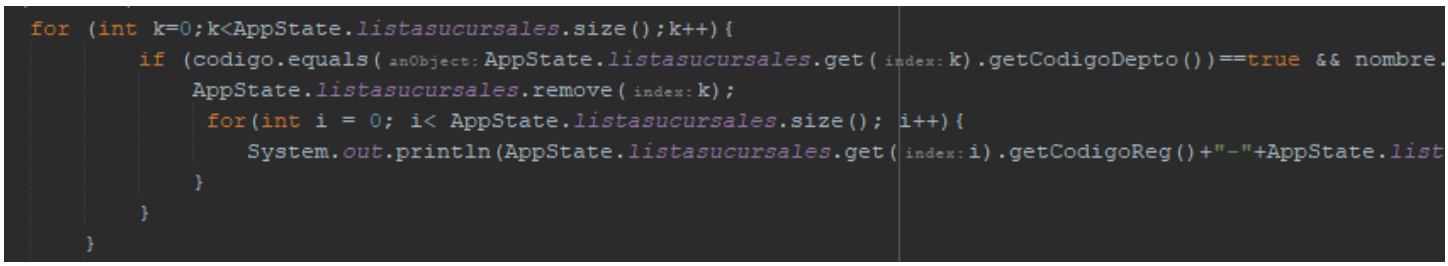
MÉTODO PARA MODIFICAR ATRIBUTOS DE UN ELEMENTO DE UNA LISTA

Para modificar un atributo, se recorre la lista por medio de una estructura for tomando en cuenta que se necesita un dato único para validar que este corresponda a un objeto específico y así obtener la posición que ocupa en el ArrayList. Se valida por medio de un if estás coincidencias y a través del método get se obtiene la posición del objeto para finalmente con el método set enviar el dato que se modificará.



MÉTODO PARA ELIMINAR ELEMENTOS DE UNA LISTA

Para eliminar un elemento, se recorre la lista por medio de una estructura for tomando en cuenta que se necesita un dato único para validar que este corresponda a un objeto específico y así obtener la posición que ocupa en el ArrayList. Se valida por medio de un if estás coincidencias y en caso de cumplirlas se utiliza el método remove al que se le indicará la posición del objeto que se eliminará.



MÉTODO PARA GENERAR CÓDIGOS ALEATORIOS

Para crear códigos aleatorios se utiliza la clase Random, se declara una variable de tipo int inicializada en cero que guardará el número aleatorio en este caso *100 (generará números de 0 a 99). La variable de tipo String se utiliza para lograr formar un código aleatorio con los primeros 5 caracteres fijos + él número aleatorio generado por la clase Random.

```
int aleatorio=0;
Random codigoRdm = new Random();
aleatorio=(int)(codigoRdm.nextDouble()*100);
String codigo="AQRT-"+aleatorio;
codekioskoTxt.setText( t: codigo);
```

MÉTODO PARA RELLENAR JCOMBOBOX RELACIONADO CON OTRO JCOMBOBOX

Se utiliza una función con un array tipo string para cada Item del JComboBox principal con parámetro de tipo String para validar si la selección del JComboBox es igual a la del método. En caso de ser así retornará la lista de elementos que se relacionarán a este Item.

```
public String[] getMetropolitana(String region1){
    String[] listadepartamentos1 = new String[4];
    if(region1.equalsIgnoreCase( anotherString: "(M) Metropolitana")){
        listadepartamentos1[0]="Guatemala";
        listadepartamentos1[1]="Escuintla";
        listadepartamentos1[2]="Chimaltenango";
        listadepartamentos1[3]="Sacatepéquez";
    }
    return listadepartamentos1;
}
```

En el método del evento ItemStateChanged, se validará cada vez que cambia la selección de un Item y se llenará el JComboBox secundario de acuerdo al modelo por defecto, pero con los datos correspondientes en su función.

```
if(evt.getStateChange()== ItemEvent.SELECTED){
    if(this.regionCombo.getSelectedIndex()==0){
        this.deptosCombo.setModel(new DefaultComboBoxModel( items: this.getMetropolitana( region1: this.regionCombo.get
    }else if(this.regionCombo.getSelectedIndex()==1){
        this.deptosCombo.setModel(new DefaultComboBoxModel( items: this.getNorte( region2: this.regionCombo.getSelected
    }else if(this.regionCombo.getSelectedIndex()==2){
        this.deptosCombo.setModel(new DefaultComboBoxModel( items: this.getNororiente( region3: this.regionCombo.getSel
    }else if(this.regionCombo.getSelectedIndex()==3){
        this.deptosCombo.setModel(new DefaultComboBoxModel( items: this.getSuroriente( region4: this.regionCombo.getSel
    }else if(this.regionCombo.getSelectedIndex()==4){
        this.deptosCombo.setModel(new DefaultComboBoxModel( items: this.getNoroccidente( region5: this.regionCombo.getS
    }else if(this.regionCombo.getSelectedIndex()==5){
        this.deptosCombo.setModel(new DefaultComboBoxModel( items: this.getSuroccidente( region6: this.regionCombo.getS
    }
}
```

ENVIAR CORREO DEL CLIENTE A TRÁVES DE JFRAMES

Se crea un objeto del Tipo del JFrame al que lo enviaremos, hacemos visible ese JFrame nuevo y en una variable de tipo String almacenamos el correo contenido en el JFrame actual para enviarlo a un JTextfield del JFrame que hemos abierto (para esto el JTextfield debe cambiarse a público y estático).

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    TajetaFrame tarjeta = new TajetaFrame();
    tarjeta.setVisible( b: true);
    String datocorreo = recibecorreoTxt.getText();
    tarjeta.recibecorreo1Txt.setText( t: datocorreo);
    dispose();
}
```

MÉTODO PARA DESCARGAR FACTURAS, GUÍAS Y COTIZACIONES EN FORMATO HTML

Se crea una función estática de tipo String para llenar el contenido del documento por medio una tabla en HTML, los parámetros serán los datos que mostraremos.

Se utiliza la clase StringBuilder (utilizada para modificar una cadena sin crear un objeto) esta servirá para agregar todos los datos.

Se utiliza el método append para escribir las líneas de código en formato HTML en donde concatenaremos los valores que se pasen a los parámetros.

```
public static String generatetablaHtm3(String Origen,String Destino,String numeropaq, String tamanoPaquete1,String estandar,String especial, String
    StringBuilder sb = new StringBuilder();
    //DATOS PERSONALES

    sb.append( str:"<DIV style=\"margin:50px\">");
    sb.append( str:"<font font face=\"monospace\">");
    sb.append( str:"<h3>").append("Origen: "+Origen).append( str:"</h3>");
    sb.append( str:"<h3>").append("Destino: "+Destino).append( str:"</h3>");
    sb.append( str:"<h3>").append("Numero de Paquetes: "+numeropaq).append( str:"</h3>");
    sb.append( str:"<h3>").append("Tamaño del Paquete: "+tamanoPaquete1+ " lb").append( str:"</h3>");

    sb.append( str:"<h3>").append("Servicio Estándar: "+estandar).append( str:"</h3>");
    sb.append( str:"<h3>").append("Servicio Especial: "+especial).append( str:"</h3>");
    sb.append( str:"<img src=\" https://cdn-icons-png.flaticon.com/512/7164/7164922.png \" width=\"180\" height=\"180\">");
    sb.append( str:"</font>");
    sb.append( str:"</DIV>");

    return sb.toString();
}
```

En el botón de cotización Se declaran variables del tipo que convenga para almacenar el contenido de las cajas de texto.

```
private void cotizacionBtnActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    String[] Encabezado = {"Número De Paquetes","Tamaño Del Paquete", "Total De Pago"};
    String Origen = (deptoCombo1.getSelectedItem().toString() + ", " + municipioCombo1.getSelectedItem().toString()+". "+direccion1Txt.getText());
    String Destino =(deptoCombo2.getSelectedItem().toString() + ", " + municipioCombo2.getSelectedItem().toString()+". "+direccion2Txt.getText());
    String numeropaq = npaquetes.getValue().toString();
    String estandar =  totalestandarTxt.getText();
    String especial =  totalespecialTxt.getText();
```

Para pasar los parámetros a la función se utiliza una variable de tipo String que contendrá lo que retorne nuestra función.

La clase File se utiliza para indicar la ruta dónde se guardarán los archivos que generemos y si no existe la crea. Al estar dentro de la carpeta escribe el nombre que tendrá el archivo y lo guarda en esa ruta.

```
    String tablaHtml = generatetablaHtm3(Origen,Destino,numeropaq,tamanoPaquete1,estandar,especial,Encabezado);
    // Crea la carpeta de reportes si no esiste
    File carpeta = new File( pathname: "C:/Users/josse/Downloads/");
    if (!carpeta.exists()) {
        carpeta.mkdirs();
    }

    // Escribe el archivo .html dentro de la carpeta
    try {
        FileWriter fileWriter = new FileWriter("C:/Users/josse/Downloads/COTIZACIÓN"+estandar+especial+".html");
        fileWriter.write( str: tablaHtml);
        fileWriter.close();
         JOptionPane.showMessageDialog( parentComponent: null,  message: "COTIZACIÓN DESCARGADA CORRECTAMENTE",  title: "Cotización Electrónica",  me
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

MÉTODO PARA OFUSCASIÓN DE TARJETAS

Se declara una variable de tipo String que contiene X en lugar de los números que se ocultaran en la tarjeta.

```
cvvLbl.setVisible( aFlag: true);
String ntarjeta= "XXXXXXXXX";
```

Para mostrar solo los valores deseados se recorre la lista de tarjetas existentes por medio de una estructura for, si el usuario tiene tarjetas registradas entonces se mostrará una lista de las tarjetas agregando cada Item al JComboBox con la variable String que creamos, concatenando los últimos 4 dígitos de la tarjeta obtenidos con el método substring.

```
for (int k=0;k<AppState.listatarjetas.size();k++){
        if (recibecorreo3Txt.getText().toString().equals( anObject: AppState.listatarjetas.get( index: k).getPropietarioT().toString())==true ){
            tarjetasCombo.addItem(ntarjeta+(String.valueOf( i: AppState.listatarjetas.get( index: k).getNumeroT())).substring( beginIndex: 9, endIndex: 13));
        }
}
```

PROCEDIMIENTO PARA LLENAR TABLA CON LOS ENVÍOS SOLICITADOS

Se indica el modelo que tendrá la tabla, con cada una de sus columnas y el número de final según el tamaño del ArrayList que contiene los envíos y se coloca con el método set.

```
ago"},  rowCount: AppState.listaenvios.size()){
```

```
DefaultTableModel model = new DefaultTableModel(new String[]{"Código de paquete", "Tipo de servicio","Destinat
    @Override
    public boolean isCellEditable(int row, int column) {
        if(column==5){
            return true;
        }else{
            return false;
        }
    }
 };
enviosTabla.setModel( dataModel:model);
```

Con una variable de tipo int para el número de fila almacenaremos el valor que devuelta el método getRowCount que contará las filas de la tabla. Luego con una estructura for recorre la lista de envios realizados que pertenezcan al usuario en línea.

Se crea un objeto del tipo de la Clase de DatosEnvios que sirve para enviar el valor de ese dato a una posición de la tabla, en este caso el código del paquete irá en la columna 0, el tipo de servicio en la columna 1, el destinatario en la columna 2, el total de pago en la columna 3 y el tipo de pago en la columna 4, como todos esos datos pertenecen a un solo envío todos irán en la misma fila según la posición que tengan en la lista de envíos.

```
int fila = enviosTabla.getRowCount();
for (int i = 0; i < AppState.listaenvios.size(); i++) {
    if(datocorreo.equals( anObject:AppState.listaenvios.get( index:i).getCorreoCliente())==true){
        DatosEnvios envios = AppState.listaenvios.get( index:i);
        modelod.setValueAt( aValue:envios.getCodigoPaq(),  rowIndex:i,  columnIndex:0);
        modelod.setValueAt( aValue:envios.getTipoServicio(),  rowIndex:i,  columnIndex:1);
        modelod.setValueAt( aValue:envios.getDestinatario(),  rowIndex:i,  columnIndex:2);
        modelod.setValueAt( aValue:envios.getTotalPago(),  rowIndex:i,  columnIndex:3);
        modelod.setValueAt( aValue:envios.getTipoPago(),  rowIndex:i,  columnIndex:4);
    }

}
```

MÉTODO PARA CARGAR UNA FOTOGRAFÍA

Se utiliza la clase JFileChooser para mostrar una ventana para la selección de un fichero. Si un usuario abre el cuadro de dialogo, elige una fotografía y presiona aceptar entonces en un objeto de la clase File obtendremos el archivo seleccionado para colocarlo en una caja de texto que contendrá la ruta.

Finalmente se crea un objeto de la clase Image al que se le asigna la valor de la imagen según la ruta, luego se fija el ancho y alto de la foto según el del Jlabel donde se mostrará y con la clase ImageIcon podremos mostrar de manera gráfica lo que contenga la ruta seleccionada.

```
private void fotoBtnActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    JFileChooser archivo = new JFileChooser();
    int ventana= archivo.showOpenDialog( parent: null);
    if (ventana == JFileChooser.APPROVE_OPTION){
        File file= archivo.getSelectedFile();
        rutaTxt.setText( t: String.valueOf( obj: file));
        Image foto = getToolkit().getImage( filename: rutaTxt.getText());
        foto = foto.getScaledInstance( width: fotografiaLbl.getWidth(),  height: fotografiaLbl.getHeight(),  hints: Image.SCALE_DEFAULT);
        fotografiaLbl.setIcon(new ImageIcon( image: foto));
    }
}
```

MÉTODO PARA VALIDAR REQUISITOS DE LA CONTRASEÑA

Se declaran variables final int ya que nunca cambiarán, para el mínimo de cada carácter que debe ser 1. También se declaran variables de tipo int que serán contadores del número de caracteres que contenga la contraseña según el tipo.

Con una estructura for se recorre cada carácter de la contraseña y se almacena en una variable de tipo char que validará si es mayuscula y aumentará el contador, si el minúscula y aumentará su contador, si es un número y aumentará su contador, si es un carácter especial y aumentará su contador.

```
final int min_mayus=1; // número min de letras mayúsculas
final int min_minus=1; // min de letras minúsculas
final int min_num=1; // min de numeros
final int especial=1; // caracteres espciales

//contadores de caracteres
int countmayus=0;
int countminus=0;
int countnum=0;
int countespecial=0;

for (int i=0; i < contra.length(); i++ ) {
    char c = contra.charAt( index: i);
    if(Character.isUpperCase( ch: c))
        countmayus++;
    else if(Character.isLowerCase( ch: c))
        countminus++;
    else if(Character.isDigit( ch: c))
        countnum++;
    if(c>=33&&c<=46||c==64){
        countespecial++;
    }
}
```

Si todos sus contadores cambian de 0 quiere decir que si cuenta con los requisitos necesarios ya que son mayores o iguales a las variables del mínimo de caracteres.

```
if (countmayus>= min_mayus && countminus>= min_minus && countnum>= min_num && countespecial>= especial){
```

MÉTODO PARA ORDENAR DE MAYOR A MENOR

Para ordenar datos de acuerdo a la lista que se desea, se crea una función estática con parámetros de un ArrayList de tipo Objeto según la lista que se ordene. Con una estructura for se recorrerán los elementos de la lista desde la posición 0 y se creará un fichero temporal de la Clase del objeto;

Dentro de la primera estructura for se crea una nueva estructura for que comenzará en la posición del elemento actual +1, es decir que si la posición del primer for es 3 la del segundo for será 4, adelantándolo una posición. Para poder ordenar de manera ascendente se valida dentro del segundo for, en este caso para comparar cual es mayor se utilizan contadores de cada elemento de la lista, entonces si en la posición actual el contador es menor que el de la posición siguiente, el elemento siguiente tomará la posición del anterior y se almacenará en el fichero temporal.

```
public static void algoritmoburbuja(ArrayList<Regiones> listaregiones){
    for(int i=0;i<AppState.listaregiones.size();i++){
        Regiones temp;
        for(int j=i+1;j<AppState.listaregiones.size();j++){
            if(AppState.listaregiones.get( index:i).getContadorRegion()<AppState.listaregiones.get( index:j).getContadorRegion()){
                temp=AppState.listaregiones.get( index:j);
                AppState.listaregiones.set( index:j, element:AppState.listaregiones.get( index:i));
                AppState.listaregiones.set( index:i, element:temp);
            }
        }
}
```