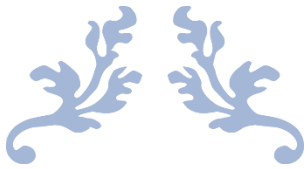

EXPLICACIÓN PROYECTO ACCESO A DATOS 01

ENEKO REBOLLO Y SAÚL MELLADO

Contenido

Paquete Objetos y sus clases

En este paquete hay 4 clases, una por cada fichero csv en el que simplemente creamos las variables de cada una de ellas. Utilizamos lombok para generar getters y setters.

```java
package Objetos;

import ...

@Data
@Builder
public class CalidadAire {
    String provincia,municipio,estacion,magnitud,punto_muestreo,ano,mes,
            dia,h01,v01,h02,v02,h03,v03,h04,v04,h05,v05,h06,v06,h07,v07,h08,
            v08,h09,v09,h10,v10,h11,v11,h12,v12,h13,v13,h14,v14,h15,v15,h16,
            v16,h17,v17,h18,v18,h19,v19,h20,v20,h21,v21,h22,v22,h23,v23,h24,v24;
}
```

```java
package Objetos;

import ...

@Data
@Builder
public class CalidadAireEstaciones {
    @NonNull String estacion_codigo,zona_calidad_aire_descripcion,estacion_municipio,estacion_fecha_alta,
            estacion_tipo_area,estacion_tipo_estacion,estacion_subarea_rural,estacion_direccion_postal,
            estacion_coord_UTM_ETRS89_x,estacion_coord_UTM_ETRS89_y,estacion_coord_longitud,
            estacion_coord_latitud,estacion_altitud,estacion_analizador_NO,estacion_analizador_NO2,
            estacion_analizador_PM10,estacion_analizador_PM2_5,estacion_analizador_O3,estacion_analizador_TOL,
            estacion_analizador_BEN,estacion_analizador_XIL,estacion_analizador_CO,estacion_analizador_SO2,
            estacion_analizador_HCT,estacion_analizador_HNM;
}
```

```java
package Objetos;

import ...

@Data
@Builder
public class CalidadAireZonas {
    @NonNull String zona_calidad_aire_codigo,zona_calidad_aire_descripcion,zona_calidad_aire_municipio;
}
```

```java
package Objetos;

import ...

@Data
@Builder
public class DatosMeteorologicos {
    @NonNull String provincia,municipio,estacion,magnitud,punto_muestreo,ano,mes,dia,h01,v01,h02,
            v02,h03,v03,h04,v04,h05,v05,h06,v06,h07,v07,h08,v08,h09,v09,h10,v10,h11,v11,h12,
            v12,h13,v13,h14,v14,h15,v15,h16,v16,h17,v17,h18,v18,h19,v19,h20,v20,h21,v21,h22,
            v22,h23,v23,h24,v24;
}
```

Paquete Csv y sus clases

Clase CalidadAireReader

Creamos nuestras listas que vamos a utilizar para leer y almacenar los datos de los ficheros csv. Utilizamos el método getInstance() para que solo haya una instancia de esa clase.

Creamos un método que obtiene el path de los ficheros csv y almacena el contenido las líneas del csv correspondiente.

```java
public class CalidadAireReader implements Runnable {

    List<String> calidadAireList;
    List<String> calidadAireZonasList;

    List<CalidadAire> calidadAireObjetosList = new ArrayList<>();
    List<CalidadAireZonas> calidadAireZonasObjetosList = new ArrayList<>();

    private static CalidadAireReader car = null;
    private CalidadAireReader(){}
    public static CalidadAireReader getInstance(){
        if(car==null){
            car=new CalidadAireReader();
        }
        return car;
    }

    /**
     * creacion de las listas de String donde almacenaremos las lineas del csv correspondiente
     */
    private void crearListaCalidadAire(){
        String actualPath = System.getProperty("user.dir");
        String pathAire = actualPath+ File.separator+"Datos"+File.separator+"calidad_aire_datos_mes.csv";
        String pathZonas = actualPath+ File.separator+"Datos"+File.separator+"calidad_aire_zonas.csv";

        Path csvAire = Paths.get(pathAire);
        Path csvZonas = Paths.get(pathZonas);


        try {
            calidadAireList = Files.readAllLines(csvAire);
            calidadAireZonasList = Files.readAllLines(csvZonas, Charset.forName("windows-1252"));
        } catch (IOException e) {
            e.printStackTrace();
        }
```

Creamos el método que utilizando un delimitador y recorriendo las listas asigna los datos del csv a las variables que tiene la clase.

```
private void crearObjetosAire(){

    crearListaCalidadAire();

    for(String a : calidadAireList){

        Scanner sc=new Scanner(a);
        sc.useDelimiter(";");
        while(sc.hasNext()){
            calidadAireObjetosList.add(CalidadAire.builder().provincia(sc.next()).municipio(sc.next()).
                    estacion(sc.next()).magnitud(sc.next()).punto_muestreo(sc.next()).ano(sc.next()).mes(sc.next()).
                    dia(sc.next()).h01(sc.next()).v01(sc.next()).h02(sc.next()).v02(sc.next()).h03(sc.next()).
                    v03(sc.next()).h04(sc.next()).v04(sc.next()).h05(sc.next()).v05(sc.next()).h06(sc.next()).
                    v06(sc.next()).h07(sc.next()).v07(sc.next()).h08(sc.next()).v08(sc.next()).h09(sc.next()).
                    v09(sc.next()).h10(sc.next()).v10(sc.next()).h11(sc.next()).v11(sc.next()).h12(sc.next()).
                    v12(sc.next()).h13(sc.next()).v13(sc.next()).h14(sc.next()).v14(sc.next()).h15(sc.next()).
                    v15(sc.next()).h16(sc.next()).v16(sc.next()).h17(sc.next()).v17(sc.next()).h18(sc.next()).
                    v18(sc.next()).h19(sc.next()).v19(sc.next()).h20(sc.next()).v20(sc.next()).h21(sc.next()).
                    v21(sc.next()).h22(sc.next()).v22(sc.next()).h23(sc.next()).v23(sc.next()).h24(sc.next()).
                    v24(sc.next()).build());
        }
    }

    for(String a : calidadAireZonasList){

        Scanner sc=new Scanner(a);
        sc.useDelimiter(";");
        while(sc.hasNext()){
            calidadAireZonasObjetosList.add(CalidadAireZonas.builder().zona_calidad_aire_codigo(sc.next()).zona_calid
                    zona_calidad_aire_municipio(sc.next()).build());
        }
    }
}
```

Clase DatosMeteorologicosReader

```java
public class DatosMeteorologicosReader implements Runnable{

    List<String> datosMeteorologicosList;
    List<DatosMeteorologicos> datosMeteorologicosObjetosList = new ArrayList<>();
    List<String> estacionesList;

    private static DatosMeteorologicosReader dmr = null;
    private DatosMeteorologicosReader(){}
    public static DatosMeteorologicosReader getInstance(){
        if(dmr==null){
            dmr=new DatosMeteorologicosReader();
        }
        return dmr;
    }

    /**
     * cramos las lista de datos a partir de la informacion de los csv
     */
    private void crearListaDatos(){
        String actualPath = System.getProperty("user.dir");
        String pathMeteo = actualPath+ File.separator+"Datos"+File.separator+"calidad_aire_datos_meteo_mes.csv";
        String pathEstaciones =  actualPath+ File.separator+"Datos"+File.separator+"calidad_aire_estaciones.csv";

        Path csvMeteo = Paths.get(pathMeteo);
        Path csvEstaciones = Path.of(pathEstaciones);

        try{
            datosMeteorologicosList = Files.readAllLines(csvMeteo);
            estacionesList = Files.readAllLines(csvEstaciones,Charset.forName("windows-1252"));
            estacionesList.remove( index: 0);
        }catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```java
private void crearObjetoDatos(){
    crearListaDatos();
    for(String a : datosMeteorologicosList){

        Scanner sc=new Scanner(a);
        sc.useDelimiter(";");
        while(sc.hasNext()){
            datosMeteorologicosObjetosList.add(DatosMeteorologicos.builder().provincia(sc.next()).municipio(sc.next(
                    estacion(sc.next()).magnitud(sc.next()).punto_muestreo(sc.next()).ano(sc.next()).mes(sc.next()).
                    dia(sc.next()).h01(sc.next()).v01(sc.next()).h02(sc.next()).v02(sc.next()).h03(sc.next()).
                    v03(sc.next()).h04(sc.next()).v04(sc.next()).h05(sc.next()).v05(sc.next()).h06(sc.next()).
                    v06(sc.next()).h07(sc.next()).v07(sc.next()).h08(sc.next()).v08(sc.next()).h09(sc.next()).
                    v09(sc.next()).h10(sc.next()).v10(sc.next()).h11(sc.next()).v11(sc.next()).h12(sc.next()).
                    v12(sc.next()).h13(sc.next()).v13(sc.next()).h14(sc.next()).v14(sc.next()).h15(sc.next()).
                    v15(sc.next()).h16(sc.next()).v16(sc.next()).h17(sc.next()).v17(sc.next()).h18(sc.next()).
                    v18(sc.next()).h19(sc.next()).v19(sc.next()).h20(sc.next()).v20(sc.next()).h21(sc.next()).
                    v21(sc.next()).h22(sc.next()).v22(sc.next()).h23(sc.next()).v23(sc.next()).h24(sc.next()).
                    v24(sc.next()).build());
        }

    }
    EstacionesMapas em = EstacionesMapas.getInstance();

    for(String a : estacionesList){

        StringTokenizer st = new StringTokenizer(a, delim: ";");

        String codigo = st.nextToken();
        int codigoMunicipio = Integer.parseInt(codigo.substring(2,5));
        System.out.println(codigoMunicipio);
        st.nextElement();
        String nombre = st.nextToken();
        em.fillCodigoMunicipio(codigoMunicipio,nombre);
        em.fillCodigoNacional(Integer.parseInt(codigo),nombre);
    }
```

Clase CsvThreadReader

Creamos esta clase para implementar hilos y agilizar la lectura de los csv. Creamos un ThreadGroup, y creamos dos hilos que se encargaran de leer las listas con los datos del csv e inicializarlas una vez acaben de leerlas

```java
public class CsvThreadReader {

    ThreadGroup tg = new ThreadGroup( name: "lectoresCSV");
    private List<CalidadAire> calidadList;
    private List<DatosMeteorologicos> datosList;
    private List<CalidadAireZonas> calidadZonasList;
    //List<CalidadAireEstaciones> aireEstacionesList;

    public CsvThreadReader(){
        try {
            empezar();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    /**
     * ejecuta ambos hilos para agilizar la lectura de los csv, y, una vez terminados
     * generados en cada hilo
     * @throws InterruptedException
     */
    private void empezar() throws InterruptedException {
        CalidadAireReader car = CalidadAireReader.getInstance();
        DatosMeteorologicosReader dmr = DatosMeteorologicosReader.getInstance();

        Thread hilo1 = new Thread(tg,car);
        Thread hilo2 = new Thread(tg,dmr);

        hilo1.start();
        hilo2.start();
        hilo1.join();
        hilo2.join();


        calidadList = car.getCalidadAireObjetosList();
        calidadZonasList = car.getCalidadAireZonasObjetosList();
        datosList = dmr.getDatosMeteorologicosObjetosList();
```

Paquete Filtro y sus clases

Clase Filtro

En la clase filtro hemos hecho dos filtros (uno por cada lista que queremos filtrar) que saque el municipio y la magnitud que se le pasae por parámetro utilizando la Api Stream.

```java
package Filtro;

import ...

public class Filtro {
    /**
     * Metodo que filtra calidadAireObjetosList y returnea solo los que sean iguales a el muncipio pasado por parametro
     * @param municipio al que hacemos referencia
     * @return la lista filtrada
     */
    public List<CalidadAire> filtroCalidadAire(String municipio, String magnitud, List<CalidadAire> datos) {
        return datos.stream().filter(p -> p.getMunicipio().equalsIgnoreCase(municipio)).filter(p -> p.getMagnitud().equalsIgnoreCase(magnitud)).collect(Collectors.toList());
    }

    /**
     * Metodo que filtra datosMeteorologicosObjetosList y returnea solo los que sean iguales a el muncipio pasado por parametro
     * @param municipio al que hacemos referencia
     * @return la lista filtrada
     */
    public List<DatosMeteorologicos> filtroDatosMeteo(String municipio, String magnitud, List<DatosMeteorologicos> datos){
        return datos.stream().filter(p -> p.getMunicipio().equalsIgnoreCase(municipio)).filter(p -> p.getMagnitud().equalsIgnoreCase(magnitud)).collect(Collectors.toList());
    }
}
```

Clase MediasHoras

En esta clase sacamos la media máxima y mínima mediante un for each que si por cada variable V el valor es igual a v almacene la variable H en sumDia de tal forma que así descartamos los valores nulos, además hacemos un contador que vaya incrementando para así sacar la media por los valores que están disponibles. Además, sacamos las estaciones relacionadas y añadimos los valores de H en una lista tipo Double para sacar la mínima y máxima.

```java
public class MediasHoras {

    private int sumaMediaDias=0;

    private static MediasHoras mediaHora = null;
    private MediasHoras(){}
    public static MediasHoras getInstance(){
        if(mediaHora==null){
            mediaHora = new MediasHoras();
        }
        return mediaHora;
    }

    /**
     * a partir de la lista filtrada, sacamos los valores de cada hora
     * @param lista de objetos calidad aire filtrado
     * @return una string con el orden media-maximo-minima mensual
     */
    public List<Object> mediaCalidadAire(List<CalidadAire> lista){
        if(lista.isEmpty()){
            List<Object> returnerList = List.of("no hay datos en este municipio");
            return returnerList;
        }
        List<Double> valores = new ArrayList<>();
        HashSet<String> estaciones = new HashSet<>();
        String returner="";

        lista.forEach(ca ->{
            double sumDia=0;
            int noNulos=0;

            if(ca.getV01().equalsIgnoreCase( anotherString: "V")){
                sumDia+=Double.parseDouble(ca.getH01());
                noNulos += 1;
                estaciones.add(ca.getPunto_muestreo());
                valores.add(Double.parseDouble(ca.getH01()));
            }
            if(ca.getV24().equalsIgnoreCase( anotherString: "V")){
                sumDia+=Double.parseDouble(ca.getH24());
                noNulos += 1;
                estaciones.add(ca.getPunto_muestreo());
                valores.add(Double.parseDouble(ca.getH24()));
            }

            sumaMediaDias+=sumDia/noNulos;

        });

        Optional<Double> max = valores.stream().max(Comparator.comparing(v -> v));
        Optional<Double> min = valores.stream().min(Comparator.comparing(v -> v));

        returner = sumaMediaDias / lista.size() + " " + max.get() + " " + min.get();
        for(String s : estaciones){
            StringTokenizer st = new StringTokenizer(s, delim: "_");
            returner+=" "+st.nextToken()+" ";
        }

        List<Object> returnerLista = List.of(returner,valores);
        return returnerLista;
    }
```

Clase InfoMeteorologica

Esta clase tiene un metodo que dependiendo de los valores de municipio y magnitud llama a un método u otro pasando por parámetro los valores. Si los valores son correctos llama al método y te devuelve la media la máxima la mínima y la estación asociada. Dependiendo de la magnitud llama a una lista u otra.

Este método llama a las clases del paquete mapa que serán explicadas a continuación.

```
public String proveedorDeDatos(List<CalidadAire> datosAire, List<DatosMeteorologicos> datosMeteo,String municipio, String magnitud){
    String returner="";
    MagnitudMap magnitudes = MagnitudMap.getInstance();

    if(Integer.parseInt(magnitud)<81 || Integer.parseInt(magnitud)==431){
        List<Object> returnDatos=datosCalidad(datosAire,municipio,magnitud);
        if(returnDatos.get(0).equals("no hay datos en este municipio")){
            return "<br/><h2 class='error'>"+returnDatos.get((0))+" sobre "+magnitudes.getMapa().get(Integer.parseInt(magnitud))+"</h2>";
        }

        valores = (List<Double>) returnDatos.get(1);
        StringTokenizer st = new StringTokenizer((String)returnDatos.get(0));

        nombre = magnitudes.getMapa().get(Integer.parseInt(magnitud));
        returner+="<br/><h2>"+nombre+":</h2> \n"+
                "<pre>—Media mensual: "+st.nextToken()+" "+unidades.getUdMedida().get(Integer.parseInt(magnitud))+"\n"+
                "-Maximo del mes: "+st.nextToken()+" "+unidades.getUdMedida().get(Integer.parseInt(magnitud))+"\n"+
                "-Minimo del mes: "+st.nextToken()+" "+unidades.getUdMedida().get(Integer.parseInt(magnitud))+"\n"+
                "Estacion/es usadas: ";

        while(st.hasMoreTokens()){
            String estacion = mapas.getCodigoNacional().get(Integer.parseInt(st.nextToken()));
            returner+=" | "+estacion;
        }
        returner+="</pre>";
    }
    if(Integer.parseInt(magnitud)>=81 && Integer.parseInt(magnitud)!=431){
        List<Object> returnDatos = datosMeteorologicos(datosMeteo,municipio,magnitud);
        if(returnDatos.get(0).equals("no hay datos en este municipio")){
            return "<br/><h2 class='error'>"+returnDatos.get(0)+" sobre "+magnitudes.getMapa().get(Integer.parseInt(magnitud))+"</h2>";
        }

        valores = (List<Double>) returnDatos.get(1);
        StringTokenizer st = new StringTokenizer((String)returnDatos.get(0));

        nombre = magnitudes.getMapa().get(Integer.parseInt(magnitud));
```

Aquí creamos los dos métodos a los que se les llama para filtrar los parámetros que se necesitan.

```
            nombre = magnitudes.getMapa().get(Integer.parseInt(magnitud));
            returner+="<h2>"+magnitudes.getMapa().get(Integer.parseInt(magnitud))+":</h2> \n"+
                    "<pre>—Media mensual: "+st.nextToken()+" "+unidades.getUdMedida().get(Integer.parseInt(magnitud))+"\n"+
                    "-Maximo del mes: "+st.nextToken()+" "+unidades.getUdMedida().get(Integer.parseInt(magnitud))+"\n"+
                    "-Minimo del mes: "+st.nextToken()+" "+unidades.getUdMedida().get(Integer.parseInt(magnitud))+"\n"+
                    "Estacion/es usadas:";

            while(st.hasMoreTokens()){
                String estacion = mapas.getCodigoNacional().get(Integer.parseInt(st.nextToken()));
                returner+=" | "+estacion;
            }
            returner+="</pre>";
        }
        return returner;
    }

    /**
     *llama al metodo que filtra pasandole los parametros que se necesitan
     * @param datos lista de DatosMeteorologicos a filtrar
     * @param municipio que queremos filtrar
     * @param magnitud que queremos obtener
     * @return una string en orden media max min
     */
    private List<Object> datosMeteorologicos(List<DatosMeteorologicos> datos, String municipio, String magnitud){
        return mh.mediaDatosMeteo(f.filtroDatosMeteo(municipio,magnitud,datos));
    }
    /**
     *llama al metodo que filtra pasandole los parametros que se necesitan
     * @param datos lista de CalidadAire a filtrar
     * @param municipio que queremos filtrar
     * @param magnitud que queremos obtener
     * @return una string en orden media max min
     */
    private List<Object> datosCalidad(List<CalidadAire> datos,String municipio, String magnitud){
        return mh.mediaCalidadAire(f.filtroCalidadAire(municipio,magnitud,datos));
    }
}
```

Paquete Mapas y sus clases

Clase EstacionesMapas

Esta clase creamos dos métodos encargados de rellenar los mapas llamados en la clase datosMeteorologicosReader del paquete csv

```
package Mapas;

import ...

@Data
public class EstacionesMapas {

    private static EstacionesMapas estaciones = null;

    private EstacionesMapas(){}

    public static EstacionesMapas getInstance(){
        if(estaciones==null){
            estaciones = new EstacionesMapas();
        }
        return estaciones;
    }

    private Map<Integer,String> codigoMunicipio = new LinkedHashMap<>();
    private Map<Integer,String> codigoNacional = new LinkedHashMap<>();

    public void fillCodigoMunicipio(int codigo, String municipio) { codigoMunicipio.put(codigo,municipio); }

    public void fillCodigoNacional(int codigo, String estacion) { codigoNacional.put(codigo,estacion); }
}
```

Clase MagnitudMap

En esta clase hemos creado un mapa en el que asignamos la magnitud a un numero de tal manera que en el Método proveedorDeDatos de la clase InfoMeteorologica muestre el nombre de la magnitud a partir de su número correspondiente.

```java
public class MagnitudMap {

    private Map<Integer,String> magnitudes = new HashMap<>();

    private MagnitudMap() { buildMap(); }
    private static MagnitudMap map = null;

    public static MagnitudMap getInstance() {
        if(map==null){
            map = new MagnitudMap();
        }
        return map;
    }

    private void buildMap(){
        magnitudes.put(1,"SO2");
        magnitudes.put(6,"CO");
        magnitudes.put(7,"NO");
        magnitudes.put(8,"NO2");
        magnitudes.put(9,"particulas en suspension PM2,5");
        magnitudes.put(10,"particulas en suspension PM10");
        magnitudes.put(12,"Oxidos de nitrogeno");
        magnitudes.put(14,"O3");
        magnitudes.put(20,"tolueno (C7H8)");
        magnitudes.put(22,"Black Carbon");
        magnitudes.put(30,"Benceno (C6H6)");
        magnitudes.put(42,"Hidrocarburos totales");
        magnitudes.put(44,"Hidrocarburos no metalicos");
        magnitudes.put(431,"MetaParaXileno");
        magnitudes.put(81,"Velocidad del viento");
        magnitudes.put(82,"direccion del viento");
        magnitudes.put(83,"temperatura");
        magnitudes.put(86,"Humedad relativa");
        magnitudes.put(87,"presion atmosferica");
        magnitudes.put(88,"Radiacion solar");
        magnitudes.put(89,"Precipitacion");
    }
```

Clase UdMedidaMapa

Esta hemos creado un método en el que asignamos a cada numero de la magnitud su unidad de medida de tal manera que en el Método proveedorDeDatos de la clase InfoMeteorologica muestre la unidad de medida correspondiente dependiendo de la magnitud.

```
public class UdMedidaMapa {
    private Map<Integer,String> udMedida = new HashMap<>();
    private static UdMedidaMapa udMedidas=null;

    private UdMedidaMapa() { initMapa(); }
    public static UdMedidaMapa getInstance(){
        if (udMedidas==null){
            udMedidas = new UdMedidaMapa();
        }
        return udMedidas;
    }

    private void initMapa(){
        udMedida.put(1,"ug/m\uFFFD (microgramos por metro cubico)");
        udMedida.put(6,"mg/m\uFFFD");
        udMedida.put(7,"ug/m\uFFFD (microgramos por metro cubico)");
        udMedida.put(8,"ug/m\uFFFD (microgramos por metro cubico)");
        udMedida.put(9,"ug/m\uFFFD (microgramos por metro cubico)");
        udMedida.put(10,"ug/m\uFFFD (microgramos por metro cubico)");
        udMedida.put(12,"ug/m\uFFFD (microgramos por metro cubico)");
        udMedida.put(14,"ug/m\uFFFD (microgramos por metro cubico)");
        udMedida.put(20,"ug/m\uFFFD (microgramos por metro cubico)");
        udMedida.put(22,"ug/m\uFFFD (microgramos por metro cubico)");
        udMedida.put(30,"ug/m\uFFFD (microgramos por metro cubico)");
        udMedida.put(42,"mg/m\uFFFD");
        udMedida.put(44,"mg/m\uFFFD");
        udMedida.put(431,"ug/m\uFFFD (microgramos por metro cubico)");
        udMedida.put(81,"m/s");
        udMedida.put(82,"Grd");
        udMedida.put(83,"\uFFFDc");
        udMedida.put(86,"%");
        udMedida.put(87,"mbar");
        udMedida.put(88,"W/m2");
        udMedida.put(89,"l/m2");
    }
}
```

Paquete FreeChart y su clase

En esta clase creamos un método que genere las gráficas utilizando la librería JFreeChart.

Si el nombre que pasamos por parámetro que en este caso es la magnitud es igual a precipitación hace un histograma, si no hace una gráfica lineal que le pasamos con la lista de valores.

```
public String generarPng(List<Double> valores, String nombre) throws IOException {
    String path = System.getProperty("user.dir") + File.separator + "src" + File.separator + "main" + File.separator
    if (nombre.equalsIgnoreCase( anotherString: "precipitacion")) {
        HistogramDataset dataset = new HistogramDataset();
        double[] prueba = new double[valores.size()];
        for (int i = 0; i < valores.size(); i++) {
            prueba[i] = valores.get(i);
        }
        dataset.addSeries( key: "key", prueba,  bins: 50);

        JFreeChart histograma = ChartFactory.createHistogram( title: "Histograma Precipitación",
                xAxisLabel: "Dias",
                yAxisLabel: "Magnitud",
                dataset,
                PlotOrientation.VERTICAL,
                legend: true,
                tooltips: true,
                urls: false);
        ChartUtilities.saveChartAsPNG(new File(path), histograma,  width: 450,  height: 400);
    } else {
        var prueba = new XYSeries( key: "2021");
        for (int i = 0; i < valores.size(); i++) {
            prueba.add(i, valores.get(i));
        }
        var dataset = new XYSeriesCollection();
        dataset.addSeries(prueba);

        JFreeChart chart = ChartFactory.createXYLineChart(
                nombre,
                xAxisLabel: "Dias",
                yAxisLabel: "Magnitud",
                dataset,
                PlotOrientation.VERTICAL,
                legend: true,
                tooltips: true,
                urls: false
            );
            ChartUtilities.saveChartAsPNG(new File(path), chart,  width: 450,  height: 400);
        }
            return path;
        }
}
```

Clase HTMLCodeGenerator

En esta clase creamos un método que devuelve las magnitudes de un municipio pasado por parámetro, recorremos un bucle de 89 posiciones que son las 89 magnitudes y si coincide con nuestro mapa anteriormente explicado en el Paquete Mapas nos devuelve una string con todos los datos, luego si nuestra magnitud existe llama al método generarPng (FreeChart) y muestra los datos de media máxima y mínima + nuestra grafica.

```
public class HTMLCodeGenerator {

    String respuesta="";

    MagnitudMap map = MagnitudMap.getInstance();
    CsvThreadReader ctr = new CsvThreadReader();
    InfoMeteorologica im = InfoMeteorologica.getInstance();
    FreeChart fChart = FreeChart.getInstance();

    /**
     * recorremos una array de 89 posiciones viendo si la i coincide con algun codigo de municipio, si coincide, lo pasa a generar los
     * unos valores asociados, es decir, no es una medicion sin valores, le genera una etiqueta html y suma la String a la String dond
     * codigo del cuerpo del html, y luego lo mismo conla medicion 431 (no se iba a hacer un for de 431 solo para esa medicion)
     * @param municipio al que hace referencia el codigo
     * @return el codigo html con/sin la etiqueta imagen
     * @throws IOException si el generador de imagen tiene problemas para guardar la imagen
     */
    public String devolverMagnitudes(String municipio) throws IOException {

        for (int i = 0; i<=89;i++){
            if(map.getMapa().containsKey(i)){
                String html="<p>"+im.proveedorDeDatos(ctr.getCalidadList(),ctr.getDatosList(),municipio,String.valueOf(i))+"</p>";

                respuesta+=html+"\n";
                List<Double> valores= im.getValores();
                if(html.length()>90){
                    if(valores!=null) {
                        String imagenGenerada = fChart.generarPng(valores, im.getNombre());
                        String imagen = "<img src='" +
                                imagenGenerada+
                                "'>";
                        respuesta += imagen+"\n\n";
                    }
                }
            }
        }
```

Aquí hacemos lo mismo para la magnitud 431, lo hemos hecho en otro método para no hacer un for de 431 integers sabiendo que a partir del valor 89 no se utiliza ninguna magnitud más, solo la 431

```
String html="<p>"+im.proveedorDeDatos(ctr.getCalidadList(),ctr.getDatosList(),municipio,String.valueOf(431))+"</p>";

respuesta+=html+"\n";
List<Double> valores= im.getValores();
if(html.length()>90){
    if(valores!=null) {
        String imagenGenerada = fChart.generarPng(valores, im.getNombre());
        String imagen = "<img src='" +
                imagenGenerada+
                "'>";
        respuesta += imagen+"\n\n";
    }
}

return respuesta;
```

Clase HTMLGenerator

En esta clase generaremos el html con los datos solicitados en la práctica.

```
public class HTMLGenerator {

    EstacionesMapas em = EstacionesMapas.getInstance();

    private  DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss");
    private LocalDateTime now = LocalDateTime.now();
    private String nowSinHora=dtf.format(now).toString().substring(0,10);
    private String fileStart="";
    private String htmlStart ="<html>\n" +
            "    <head>\n" +
            "        <title>Datos Meteorologicos</title>\n" +
            "        <link rel='stylesheet' href='meteorologia.css' type='text/css'/>\n" +
            "\n" +
            "    </head>\n" +
            "    <body>\n";
    private String htmlFinish = "    </body>\n" +
            "</html>";

    private File html;
```

Creamos un método que genere el html en base al incio y fin de la medición, también cuenta cuanto tarda en generar el html y los creadores del fichero.

```
public void htmlGenerator(String municipio,String texto, String uri,Long tiempoNecesitado, int anio, int mes, int dia){
    fileStart+="<pre class='inicio'>Ciudad:"+em.getCodigoMunicipio().get(Integer.parseInt(municipio))+
            "\nMedicion iniciada el "+dia+"/"+mes+"/"+anio+"\n"+
            "Medicion finalizada el "+nowSinHora+"\n</pre>";

    html = new File( pathname: uri+File.separator+municipio+".html");

    String finish = "<p class='generado'>Informe generado el dia "+dtf.format(now)+" en "+tiempoNecesitado+" segundos</p>\n" +
            "creadores: Eneko Rebollo y Saul Mellado";

    try {
        BufferedWriter bw = new BufferedWriter(new FileWriter(html));
        bw.write( str: fileStart+htmlStart+texto+htmlFinish+finish);
        bw.flush();
        bw.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
```

Clase Funcional

Esta clase tiene un metodo start() que a partir del municipio y la uri pasados por parámetros llama al resto de métodos de nuestro programa para que se cree el html con todos los datos correctos. Tiene un metodo que comprueba si el municipio pasado por parámetro no existe no lo generará.

```java
public class Funcional {

    private String codMunicipio=null;

    private static Funcional funcional = null;
    private Funcional() {}
    public static Funcional getInstance() {
        if (funcional == null) {
            funcional = new Funcional();
        }
        return funcional;
    }

    /**
     * a partir de el codigo de municipio y la uri de guardado del html generamos este
     *
     * @param municipio es el codigo del municipio que queremos buscar
     * @param uri       la direccion donde guardaremos el html
     * @throws IOException si el generador de imagenes tiene problemas para guardar la imagen
     */
    public void start(String municipio, String uri) throws IOException, InterruptedException {
        Long startTime = System.currentTimeMillis();
        EstacionesMapas em = EstacionesMapas.getInstance();
        CsvThreadReader ctr = CsvThreadReader.getInstance();
        HTMLCodeGenerator generator = new HTMLCodeGenerator();
        HTMLGenerator htmlGenerator = new HTMLGenerator();
        DatosMeteorologicosReader dmr = DatosMeteorologicosReader.getInstance();

        municipioExists(municipio);
        if (codMunicipio==null){
            System.out.println("municipio no encontrado");
        }
        else {
            String html = (generator.devolverMagnitudes(codMunicipio, uri));

            Long finishTime = (System.currentTimeMillis() - startTime) / 1000;
            htmlGenerator.htmlGenerator(codMunicipio, html, uri, finishTime,
                    Integer.parseInt(dmr.getDatosMeteorologicosObjetosList().get(4).getAno()),
                    Integer.parseInt(dmr.getDatosMeteorologicosObjetosList().get(4).getMes()),
                    Integer.parseInt(dmr.getDatosMeteorologicosObjetosList().get(4).getDia()));
        }
    }
```

Este es el método que comprueba que el municipio existe.

```java
private void municipioExists(String municipio) {
    EstacionesMapas em = EstacionesMapas.getInstance();
    if (em.getCodigoMunicipio().containsValue(municipio)) {
        for (Map.Entry<Integer, String> entry : em.getCodigoMunicipio().entrySet()) {
            if (Objects.equals(entry.getValue(), municipio)) {
                codMunicipio = String.valueOf(entry.getKey());
            }
        }
    }
```

Creamos un metodo que crea una carpeta en la uri si no existe

Creamos otro metodo que comprueba si el fichero html existe y nos pregunta si queremos o no reemplazarlo.

Y por último creamos otro metodo que ejecuta el html en nuestro navegador por defecto

```
private void crearCarpeta(String uri){
    File dir = new File(uri);
    if (!dir.exists()){
        dir.mkdirs();
    }
}

private void checkFile(String uri,String municipio){
    File file = new File(uri+File.separator+municipio+".html");
    if(file.exists()){
        System.out.println("parece que ya existe el archivo");
        System.out.println("quiere reemplazar el archivo existente? si/no");
        Scanner sc = new Scanner(System.in);
        String ans = sc.next();
        if(ans.equalsIgnoreCase("no")){
            System.out.println("proceda a sacar de la carpeta el archivo");
            System.exit(0);
        }
    }
}

private void ejecutarHtml(String uri, String municipio){
    String urii = uri+File.separator+municipio+".html";
    File htmlFile = new File(urii);
    try {
        Desktop.getDesktop().browse(htmlFile.toURI());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Clase Main

Nuestro main simplemente comprueba que hemos pasado dos valores por parámetro y si llama a la clase funcional para que realice toda la lógica del programa. Si no hay dos valores pasados por parámetro no llamará a la clase.

```
public class Main {
    public static void main(String[] args) throws IOException, InterruptedException {

        if(args.length==2){
            Funcional funcional = Funcional.getInstance();
            funcional.start(args[0],args[1]);
        }
        else{
            System.out.println("valores aportados no validos, se busca: municipio uri");
        }

        //funcional.start("9", "C:\\Users\\eneko\\Desktop\\prueba");
    }
}
```