

# Práctica acceso a datos 02

Se ha reciclado casi el 99% del código sobre la práctica anterior

CARLOS MORENO  
ÁNGEL MATEO

# índice

- ▶ Clase ReaderFiles
- ▶ Clase JDOMReader
- ▶ Resultado HTML
- ▶ Clases GenerarHTML y Datos HTML
- ▶ Clase JAXBdbMediciones
- ▶ Clases POJO (mediciones)
- ▶ Clase ConsultasXPath
- ▶ Clase GenerateXSDbyXML
- ▶ Clase ProcesamientoDatos
- ▶ Librerías implementadas
- ▶ Prueba ejecución

# Clase ReaderFiles

```
public static Optional<List<Magnitud>> readDataOfPathMagnitudMeteorizacion() {

    Path path = Paths.get(PATH_MAGNITUDES_METEO);
    String nombreArchivo = XMLConvertUtil.quitarExtensionCSV(path.getFileName().toString());

    if (Files.exists(path)) {
        try (Stream<String> stream = Files.lines(path, Charset.forName("windows-1252"))) {

            Element contenidos = new Element(nombreArchivo);
            Document doc = new Document(contenidos);
            ArrayList<String> elementos = XMLConvertUtil.getTitlesCSV(path);


```

Cada vez que se procesa un csv, se crea un String de nombre de archivo y se coge el nombre que va a tener el csv y se quita la extensión csv con el método `quitarExtensionCSV`. Se crea un nuevo elemento que su nombre será el nombre del archivo y creamos un Document doc con todo el contenido.

Y con el array se almacena la cabecera del csv con el método `getTitlesCSV`. (Esto lo hace por cada uno de los CSV)

```
public static String quitarExtensionCSV(String nombreArchivo) {

    StringBuilder b = new StringBuilder(nombreArchivo);
    b.replace(nombreArchivo.lastIndexOf( str: ".csv"), nombreArchivo.length(), str: "" );
    return b.toString();
}


```

```
public static ArrayList<String> getTitlesCSV(Path path) {
    ArrayList<String> elementosString = new ArrayList<>();
    try (Stream<String> stream = Files.lines(path, Charset.forName("windows-1252"))) {
        Optional<String[]> elementosArray = stream.map(s → s.split( regex: ";", limit: -1)).findFirst();
        elementosString.addAll(Arrays.asList(elementosArray.get()));
    } catch (IOException e) {
        e.printStackTrace();
    }

    return elementosString;
}


```

# Clase ReaderFiles

```
Optional<List<Magnitud>> listaFinal = Optional.of(stream
    .map(s → s.split( regex ":", limit: -1)).skip(1)
    .map(splitted → {

        contenidos.addContent(XMLConvertUtil.createXMLItem(elementos, splitted));

        int codigo_magnitud = Integer.parseInt(splitted[0]);
        String descripcion_magnitud = splitted[1];
        int codigo_tecnica_medida = Integer.parseInt(splitted[2]);
        String unidad = splitted[3];
        String descripcion_unidad = splitted[4];

        return new Magnitud(codigo_magnitud,
            descripcion_magnitud, codigo_tecnica_medida, unidad,
            descripcion_unidad, technicalDescriptionMeasure: null);

    })
    .collect(Collectors.toList());

XMLConvertUtil.generarXML(nombreArchivo, doc); //Generar el XML
return listaFinal;
} catch (IOException ex) {
    System.err.println(ex.getMessage());
    return Optional.empty();
}
}
else{
    return Optional.empty();
}
```

Agregamos a los contenidos el contenido del csv con el método createXMLItem, recibe los elemento (nombre de cada columna) y el contenido. El método generarXML genera el el XML dándole un formato y exportando el XML.

```
public static Element createXMLItem(ArrayList<String> elementos, String[] splitted) {
    Element contenido = new Element( name: "item");
    for(int n = 0; n < elementos.size(); n++) {
        Element c = new Element(elementos.get(n));
        c.setText(splitted[n]);
        contenido.addContent(c);
    }

    /**
     * Genera el XML
     * @param nombreArchivo {@link String}
     * @param doc {@link Document}
     * @throws IOException
     */
    public static void generarXML(String nombreArchivo, Document doc) throws IOException {
        XMLOutputter xml = new XMLOutputter();
        xml.setFormat(Format.getPrettyFormat());
        xml.output(doc, new FileWriter( fileName: ReaderFiles.PATH_FILES + File.separator + nombreArchivo + ".xml"));
    }
}
```

# Clase JDOMReader

```
public class JDOMReader {  
    private static JDOMReader controller;  
    public static final String PATH_FILES = System.getProperty("user.dir") + File.separator + "src" + File.separator  
        + "main" + File.separator + "resources" + File.separator + "data";  
    private static final String PATH_ZONAS = PATH_FILES + File.separator + "calidad_aire_zonas.xml";  
    private static final String PATH_UBICA_ESTACIONES = PATH_FILES + File.separator + "calidad_aire_estaciones.xml";  
    private static final String PATH_METEO = PATH_FILES + File.separator + "calidad_aire_datos_meteo_mes.xml";  
    private static final String PATH_CONTAMINACION = PATH_FILES + File.separator + "calidad_aire_datos_mes.xml";  
    private static final String PATH_MAGNITUDES_CONTAMINACION = PATH_FILES + File.separator + "magnitudes_contaminacion.xml";  
    private static final String PATH_MAGNITUDES_METEO = PATH_FILES + File.separator + "magnitudes_meteorizacion.xml";  
    private Document dataZonas;  
    private Document dataUbicaciones;  
    private Document dataMeteo;  
    private Document dataContamina;  
    private Document dataMagContamina;  
    private Document dataMagMeteo;  
}
```

La clase JDOMReader tiene como atributos

- controller: controller para generar el singleton y que solamente haya una instancia de esta clase.
- PATH\_NOMBRE: almacena la dirección donde se encuentran los XML.
- dataNombre: para construir los documentos de cada uno de los XML.

# Clase JDOMReader

```
private JDOMReader() {  
}  
  
/**  
 * JDOMReader  
 * @return JDOMReader  
 */  
public static JDOMReader getInstance() {  
    if (controller == null) {  
        controller = new JDOMReader();  
    }  
    return controller;  
}
```

Método getInstance (Singleton), para que haya una sola instancia de la clase.

```
/**  
 * Carga los datos  
 * @throws IOException Excepción  
 * @throws JDOMException Excepción  
 */  
public void loadData() throws IOException, JDOMException {  
    SAXBuilder builder = new SAXBuilder();  
    File zonasXML = new File(PATH_ZONAS);  
    File ubicacionesXML = new File(PATH_UBICA_ESTACIONES);  
    File meteoXML = new File(PATH_METEO);  
    File contaminacionXML = new File(PATH_CONTAMINACION);  
    File magContaminacionXML = new File(PATH_MAGNITUDES_CONTAMINACION);  
    File magMeteoXML = new File(PATH_MAGNITUDES_METEO);  
    this.dataZonas = builder.build(zonasXML);  
    this.dataUbicaciones = builder.build(ubicacionesXML);  
    this.dataMeteo = builder.build(meteoXML);  
    this.dataContaminacion = builder.build(contaminacionXML);  
    this.dataMagContaminacion = builder.build(magContaminacionXML);  
    this.dataMagMeteo = builder.build(magMeteoXML);  
}
```

Método loadData(), es el que se encargará de cargar los datos.

# Clase JDOMReader

```
/**
 * Devuelve una lista Optional con las zonas del municipio
 * @return Optional<List<ZonasMunicipio>>
 */
public Optional<List<ZonasMunicipio>> getZonas() {
    Element root = this.dataZonas.getRootElement();
    List<Element> listOfZonas = root.getChildren( cname: "item");

    List<ZonasMunicipio> zonasList = new ArrayList<>();

    listOfZonas.forEach(zona -> {
        ZonasMunicipio zonas_m = new ZonasMunicipio();
        zonas_m.setAirCodeQualityZone(zona.getChildText( cname: "zona_calidad_aire_codigo"));
        zonas_m.setMunicipalAirQualityZone(zona.getChildText( cname: "zona_calidad_aire_descripcion"));
        zonas_m.setMunicipalAirQualityZone(zona.getChildText( cname: "zona_calidad_aire_municipio"));
        zonasList.add(zonas_m);
    });
    return Optional.of(zonasList);
}
```

Métodos get, para cada XML, para establecer los datos de los distintos ficheros y estableciendo por el nombre de la etiqueta, los valores en su respectivo atributo de la clase POJO

# Resultado HTML

**INFORME** 

**Servicio meteorológico y contaminación**

**fuenlabrada**

**Estaciones asociadas:**

Alcobendas

Alcorcón

Fuenlabrada

Torrejón de Ardoz

**Fecha de inicio de la medición:**

01/09/2021 - 00:00:00

**Fecha de fin de la medición:**

27/09/2021 - 00:00:00

**Meteorología**

**Velocidad del viento**

Una vez se ejecuta el programa, se genera este HTML, en comparación de la práctica anterior se le ha añadido diseño (CSS)



# Clases GenerarHTML y Datos HTML

```

public class GeneradorHTML {

    private static final String PROGRAM_NAME = "Servicio meteorológico y contaminación";

    /**
     * Genera el HTML
     * @param nombreCiudad {@link String}
     * @throws IOException Excepción IO
     */
    public static void generarHtml(String nombreCiudad) throws IOException {
        //HEAD
        StringBuilder htmlString = new StringBuilder();
        htmlString.append(String.format("<!DOCTYPE html>\n" +
            "<html lang='es'>\n" +
            "    <head>\n" +
            "        <meta http-equiv='Content-Type' content='text/html; charset=UTF-8'>\n" +
            "        <title>%s</title>\n" +
            "        <link rel='stylesheet' type='text/css' href='style.css' media='screen' />
            </head>\n" +
            "    <body>\n" +
            "        <header>\n" +
            "            <img src='informe.png'>\n" +
            "        </header>\n" +
            "        <section>\n", PROGRAM_NAME));
    }
}

```

Esta clase es la que se encargará de generar el HTML con la estructura vista.

```

22 public class DatosHTML {
23     private String nombreCiudad;
24     static private StringBuilder stringHTMLData;
25     public static StringBuilder getStringHTMLData() { return stringHTMLData; }
26
27     /**
28      * Reset HTML Data
29      */
30     public static void resetHTMLData() {
31         if (stringHTMLData != null) {
32             stringHTMLData.setLength(0);
33         }
34     }
35
36     /**
37      * Procesar los datos por ciudad
38      * @param nombreCiudad {@link String}
39      */
40     public void procesarDatosPorCiudad(String nombreCiudad) {
41         this.nombreCiudad = nombreCiudad;
42
43         Optional<List<UbicacionEstaciones>> listaEstaciones = Utils.filtrarPorCiudad(nombreCiudad);
44         String codigoCiudad = Utils.filtrarPorCiudad(nombreCiudad).get().get(0).getStationCode();
45
46         if (listaEstaciones.isPresent()) {
47             codigoCiudad = listaEstaciones.get().get(0).getStationCode();
48         }
49
50         if (listaEstaciones.isPresent()) {
51             procesarDatosPorCodigo(codigoCiudad);
52         }
53     }
54 }

```

Esta clase es la que se encargará de generar los datos e ir almacenándolos en el HTML.

# Clase JAXBdbMediciones

```
package com.angcar.io;

import ...

public class JAXBdbMediciones {
    private static JAXBdbMediciones controller;
    private Marshaller marshaller;
    private Unmarshaller unmarshaller;

    /**
     * Constructor privado para Singleton
     */
    private JAXBdbMediciones(){
    }

    /**
     * Singleton
     * @return JAXBdbMediciones
     */
    public static JAXBdbMediciones getInstance() {
        if (controller == null) {
            controller = new JAXBdbMediciones();
        }
        return controller;
    }
}
```

Esta clase tiene los atributos:

- controller: para generar una sola instancia de esta clase.
- marshaller: para convertir el XML en un objeto.
- unmarshaller: para convertir el objeto en un XML.

# Clase JAXBdbMediciones

```
/**
 * Crear base de datos y devolver el documento
 * @param resultadoMediciones {@link String}
 * @param uri {@link String}
 * @throws JAXBException excepción
 */
public Document crearBDMediciones(ResultadoMediciones resultadoMediciones, String uri) throws JAXBException, ParserConfigurationException {
    JAXBContext context = JAXBContext.newInstance(ResultadosMediciones.class);

    //Leer archivo e introducir resultados
    File xml = new File(uri);
    this.unmarshaller = context.createUnmarshaller();
    ResultadosMediciones resultados;
    List<ResultadoMediciones> resultadosList;

    if (xml.exists()) {
        resultados = (ResultadosMediciones) unmarshaller.unmarshal(xml);
        resultadosList = resultados.getResultados();
    } else {
        resultados = new ResultadosMediciones();
        resultadosList = new ArrayList<>();
    }

    resultadosList.add(resultadoMediciones);
    resultados.setResultados(resultadosList);
}
```

El método crearBDMediciones(), recoge los resultados de las mediciones y la uri dónde se exportará. Este método se encargará de crear la base de datos y devolverá el DOM para aprovecharlo en las consultas XPATH

```
73 //DOM
74 DocumentBuilderFactory domFactory = DocumentBuilderFactory.newInstance();
75 DocumentBuilder domBuilder = domFactory.newDocumentBuilder();
76 Document doc = domBuilder.newDocument();
77
78 //Ahora escribir los cambios
79 this.marshaller = context.createMarshaller();
80 this.marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
81 this.marshaller.marshal(resultados, xml);
82 this.marshaller.marshal(resultados, doc);
83
84 return doc;
85
86 }
```

# Clases POJO (mediciones)

```
package com.angcar.model.resultados;
import ...

@Data
@XmlType(name = "resultado", propOrder = {"ciudad", "estacionesAsociadas", "fechaInicio",
    "fechaFin", "datosMeteo", "datosContamina"})
public class ResultadoMediciones {
    @Getter(AccessLevel.NONE) private String id;
    private String ciudad;
    @Getter(AccessLevel.NONE) private List<String> estacionesAsociadas;
    private String fechaInicio;
    private String fechaFin;
    @Getter(AccessLevel.NONE) private List<DatosMagnitud> datosMeteo;
    @Getter(AccessLevel.NONE) private List<DatosMagnitud> datosContamina;

    @XmlAttribute(name = "id")
    public String getId() { return id; }

    @XmlElementWrapper(name = "estaciones_asociadas")
    @XmlElement(name = "estacion")
    public List<String> getEstacionesAsociadas() { return estacionesAsociadas; }

    @XmlElementWrapper(name = "datos-metereologicos")
    @XmlElement(name = "dato")
    public List<DatosMagnitud> getDatosMeteo() { return datosMeteo; }

    @XmlElementWrapper(name = "datos-contaminacion")
    @XmlElement(name = "dato")
    public List<DatosMagnitud> getDatosContamina() { return datosContamina; }
}
```

En estas clases se almacenarán los resultados de las mediciones con el marshaller comentado anteriormente, para poder generar la base de datos con el unmarshaller.

- ▼ resultados
  - 📦 DatosDiaMagnitud
  - 📦 DatosMagnitud
  - 📦 DatosMomento
  - 📦 ResultadoMediciones
  - 📦 ResultadosMediciones

```
1 package com.angcar.model.resultados;
2
3 import ...
4
5
6
7
8
9
10
11 @Data
12 @XmlRootElement(name = "resultados")
13 public class ResultadosMediciones {
14
15     @Getter(AccessLevel.NONE) private List<ResultadoMediciones> resultados;
16
17     @XmlElement(name = "resultado")
18     public List<ResultadoMediciones> getResultados() { return resultados; }
19
20
21 }
```

# Clase ConsultasXPath

```
public class ConsultasXPath {  
    /**  
     * Realizar operaciones XPATH, generar archivo '.md' y mostrar por consola  
     * @param ciudad {@link String}  
     * @param path {@link String}  
     * @throws JAXBException Excepción  
     * @throws ParserConfigurationException Excepción  
     * @throws XPathExpressionException Excepción  
     * @throws IOException Excepción  
     */  
    public static void operacionesXPath(String ciudad, String path, Document doc)  
        throws JAXBException, ParserConfigurationException, XPathExpressionException, IOException {  
  
        // Almacenar los datos necesarios para mostrarselo al usuario  
        XPathFactory factory = XPathFactory.newInstance();  
        XPath xpath = factory.newXPath();  
  
        NodeList list2 = (NodeList) xpath.evaluate(  
            expression: "//resultado/datos-metereologicos/dato/media",  
            doc, XPathConstants.NODESET);  
  
        NodeList list1 = (NodeList) xpath.evaluate(  
            expression: "//resultado/datos-metereologicos/dato/@tipo",  
            doc, XPathConstants.NODESET);  
    }  
}
```

Esta clase realizará las operaciones XPATH, generará el archivo Markdown con los datos de las medias filtradas por ciudad y mostrarlo por consola.

Se harán las consultas por expresiones mirando la lista de nodos.

```
NodeList list5 = (NodeList) xpath.evaluate(  
    expression: "//resultado/datos-contaminacion/dato/media",  
    doc, XPathConstants.NODESET);  
  
NodeList list6 = (NodeList) xpath.evaluate(  
    expression: "//resultado/datos-contaminacion/dato/@tipo",  
    doc, XPathConstants.NODESET);  
  
NodeList list3 = (NodeList) xpath.evaluate(  
    expression: "//resultado/@id",  
    doc, XPathConstants.NODESET);  
  
NodeList list4 = (NodeList) xpath.evaluate(  
    expression: "//resultado/ciudad",  
    doc, XPathConstants.NODESET);
```

# Clase ConsultasXPath

En esta clase se aprovecha el DOM de cuando se genera la base de datos y se realizan las operaciones para obtener el número de atributos meteorológicos y contaminación dada una ciudad.

Esta operación se hace porque no sabes el número elementos y también por si en algún caso se quiera expandir los elementos.

La variable md, es la que almacenará los datos sobre la media para escribirlo en el Markdown.

```
// Operaciones para obtener el número de atributos meteorologicos y de contaminacion de una ciudad
int numMeteo = list1.getLength()/list3.getLength();
int numConta = list6.getLength()/list3.getLength();

//String idBuscado = list3.item(list3.getLength() -1).getTextContent(); Manera de sacar el último ID generado
StringBuilder md = new StringBuilder();
```

# Clase ConsultasXPath

```
// Sacar los datos por pantalla y guardándolos en la variable md
//for(int i = list3.getLength() -1; i < list4.getLength(); i++) { Otra manera para sacar la última medición añadida
for(int i = 0; i < list4.getLength(); i++) {
    if (ciudad.equals(list4.item(i).getTextContent())) { // Mostrar la ciudad que ha sido pasada por parametro
        System.out.println(list4.item(i).getTextContent() + " id: " + list3.item(i).getTextContent());
        md.append("#").
            append(list4.item(i).getTextContent())
            .append(" id: ")
            .append(list3.item(i).getTextContent())
            .append("\n");
        System.out.println("Meteorización");
        md.append("##").
            append("Meteorización")
            .append("\n");
        for (int j = i * numMeteo; j < (i * numMeteo) + numMeteo; j++) {
            System.out.println(list1.item(j).getTextContent() + ": " + list2.item(j).getTextContent());
            md.append("###").
                append(list1.item(j).getTextContent())
                .append(": ")
                .append(list2.item(j).getTextContent())
                .append("\n");
        }
        System.out.println("Contaminación");
        for (int x = i * numConta; x < (i * numConta) + numConta; x++) {
            System.out.println(list6.item(x).getTextContent() + ": " + list5.item(x).getTextContent());
            md.append("###").
                append(list6.item(x).getTextContent())
                .append(": ")
                .append(list5.item(x).getTextContent())
                .append("\n");
        }
    }
}
```

El for que esta comentado, es por si se quiere sacar los datos de solamente de la última inserción a la base de datos.

Se recorrerán los datos por item y se iran mostrando por pantalla y agregando a la variable md los datos de la media filtrados por ciudad.

# Clase ConsultasXPath

```
FileWriter markdown = new FileWriter( fileName: path+"informe-ciudad-"+ciudad+".md");  
markdown.write(md.toString());  
markdown.close();
```

Se exportará el fichero markdown a la ruta dada y añadiendo al nombre el nombre de la ciudad pasada por parámetro, se escribirán los datos y se cerrará el fichero para que no se quede colgado.



# Clase GenerateXSDbyXML

```

1 package com.angcar.io;
2
3 import lombok.SneakyThrows;
4
5 import javax.xml.bind.SchemaOutputResolver;
6 import javax.xml.transform.Result;
7 import javax.xml.transform.stream.StreamResult;
8 import java.io.File;
9
10 public class GenerateXSDbyXML extends SchemaOutputResolver {
11
12     /**
13      * Crea el xsd en base a un contexto
14      * @param namespaceURI
15      * @param suggestedFileName
16      * @return Result
17      */
18     @SneakyThrows
19     public Result createOutput(String namespaceURI, String suggestedFileName) {
20         String path = "src" + File.separator + "main" + File.separator + "resources" + File.separator;
21         File file = new File(path + suggestedFileName);
22         StreamResult result = new StreamResult(file);
23         result.setSystemId(file.getAbsolutePath());
24         return result;
25     }
26 }

```

```

public Document crearBDMediciones(ResultadoMediciones resultadoMediciones, String uri) throws JAXBException, ParserConfigurationException, IOException {
    JAXBContext context = JAXBContext.newInstance(ResultadosMediciones.class);
    SchemaOutputResolver sor = new GenerateXSDbyXML();
    context.generateSchema(sor);
}

```

Esta clase sirve para generar el xsd de la base de datos en base a un contexto de JAXB. El xsd se crea en base a la etiqueta que le hemos puesto @XmlType de la librería javax.

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema version="1.0" xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="resultados" type="resultadosMediciones"/>

    <xs:complexType name="resultadosMediciones">
        <xs:sequence>
            <xs:element name="resultado" type="resultado" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>

    <xs:complexType name="resultado">
        <xs:sequence>
            <xs:element name="ciudad" type="xs:string" minOccurs="0"/>
            <xs:element name="estaciones_asociadas" minOccurs="0">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="estacion" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="fechaInicio" type="xs:string" minOccurs="0"/>
            <xs:element name="fechaFin" type="xs:string" minOccurs="0"/>
            <xs:element name="datos_meteorologicos" minOccurs="0">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="dato" type="magnitud" minOccurs="0" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>

```

# Clase ProcesamientoDatos

```
/**
 * Ejecución del programa
 */
public void ejecutarPrograma() {
    List<String[]> pares = IntStream.iterate( seed: 0, i → i + 2).limit(ARGS.length / 2)
        .mapToObj(n → new String[]{ARGS[n], ARGS[n + 1]}).collect(Collectors.toList());

    pares.forEach(pair → {
        String ciudad = pair[0]; //Argumento ciudad
        path_destination = pair[1] + File.separator;

        Utils.createEmptyFolders(path_destination);

        if (Utils.inicializarDatosCSV()) {
            ResultadoMediciones datosResultadoMediciones = new ResultadoMediciones();

            //En su lugar, leer los datos desde el XML
            try {
                Utils.inicializarDatosXML();
                ResultadosMedicionService res = new ResultadosMedicionService(ciudad);
                datosResultadoMediciones = res.cargarResultadosMediciones();
            } catch (IOException | JDOMException e) {
                System.err.println("No se ha podido inicializar los datos del XML.");
                System.exit( status: 0);
            }
        }
    });
}
```

Esta clase es la encargada de mover todo el programa con el método ejecutarPrograma.

Dónde se inicializarán los datos del CSV Y los datos XML dada un ciudad. Si no se han podido inicializar los datos del XML entonces mostrará por pantalla un error y se cerrará el programa ya que se requieren esos datos para poder seguir con la ejecución.

# Clase ProcesamientoDatos

```
//Una vez recibidos los datos de las mediciones, trabajar con ellos
//CREAR LA BASE DE DATOS DE MEDICIONES
JAXBdbMediciones bd = JAXBdbMediciones.getInstance();

//Intentar generar Base de datos
try {

    String uri = System.getProperty("user.dir") + File.separator + "src" + File.separator
        + "main" + File.separator + "resources" + File.separator + "data" + File.separator + "db"
        + File.separator + "mediciones.xml";

    //Crear base de datos de mediciones
    ConsultasXPath.operacionesXPath(ciudad,path_destination,
        bd.crearBdMediciones(datosResultadoMediciones, uri)
    );

} catch (JAXBException | ParserConfigurationException e) {
    System.err.println("No se ha podido crear la base de datos de mediciones.");
    System.exit( status: 0);
} catch (XPathExpressionException | IOException e) {
    System.err.println("No se han podido realizar las consultas con XPath.");
}
}
```

Una vez se reciben los datos sobre las mediciones, se podrá trabajar con ellos y se creará la base de datos y si no se puede crear por alguna razón, muestra un error y se cierra el programa ya que no podemos seguir con la ejecución ya que si no se ha podido generar la base de datos no se podrán realizar las consultas XPath.

Si se ha generado correctamente se hacen las operaciones XPath, exportando el Markdown con los datos de la media sobre la ciudad pasada y se mostrarán por consola.

# Clase ProcesamientoDatos

```
//Intentar generar HTML
try {
    DatosHTML datosCiudad = new DatosHTML();
    datosCiudad.procesarDatosPorCiudad(ciudad);
    GeneradorHTML.generarHtml(ciudad);
} catch (IOException e) {
    System.err.println("No se ha podido generar el HTML.");
}
} else {
    System.err.println("Los archivos CSV no se han podido leer.");
    System.exit( status: 0);
}
});
```

En esta parte se generará el HTML con los datos generados, filtrados por la ciudad pasada. Si no se ha podido generar el HTML se mostrará el error.

Se controla también si no se ha podido leer los CSV y se cerrará el programa.

# Clase ProcesamientoDatos

```
/**
 * Mide el tiempo de ejecución del programa y devuelve un informe
 * @return Devuelve cuándo se ha creado el informe y cuánto tiempo ha tardado
 */
public static String tiempoInforme() {
    double tiempo = (double) ((System.currentTimeMillis() - Utils.init_time)/1000);
    LocalDate fecha = LocalDate.now();
    String formatearFecha = "dd/MM/yyyy";
    LocalTime hora = LocalTime.now();
    String formatearHora = "HH:mm:ss";

    return "Informe generado en el día " + fecha.format(DateTimeFormatter.ofPattern(formatearFecha))
        + " a las " + hora.format(DateTimeFormatter.ofPattern(formatearHora)) + " en " + tiempo + " segundos";
}
```

Este método es el que medirá el tiempo de ejecución de todo el programa y el día y la hora en la que se ha generado el informe, se podrá visualizar en el HTML

# Librerías implementadas

```
<dependency>
  <groupId>javax.xml.bind</groupId>
  <artifactId>jaxb-api</artifactId>
  <version>2.3.1</version>
</dependency>

<dependency>
  <groupId>org.glassfish.jaxb</groupId>
  <artifactId>jaxb-runtime</artifactId>
  <version>2.3.2</version>
</dependency>

<dependency>
  <groupId>javax.activation</groupId>
  <artifactId>activation</artifactId>
  <version>1.1.1</version>
</dependency>

<dependency>
  <groupId>javax.xml</groupId>
  <artifactId>jaxb-api</artifactId>
  <version>2.1</version>
</dependency>
```

Librería para  
poder utilizar JAXB

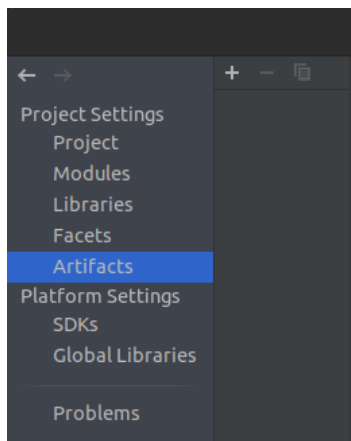
```
<dependency>
  <groupId>org.jdom</groupId>
  <artifactId>jdom</artifactId>
  <version>2.0.2</version>
</dependency>
```

Librería para poder  
trabajar con el parser  
JDOM

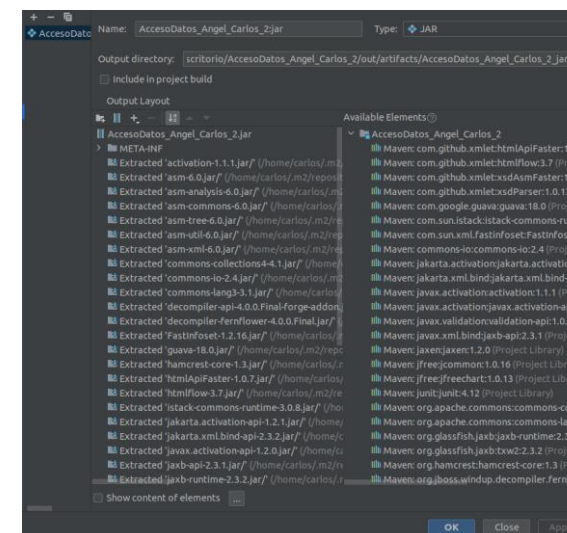
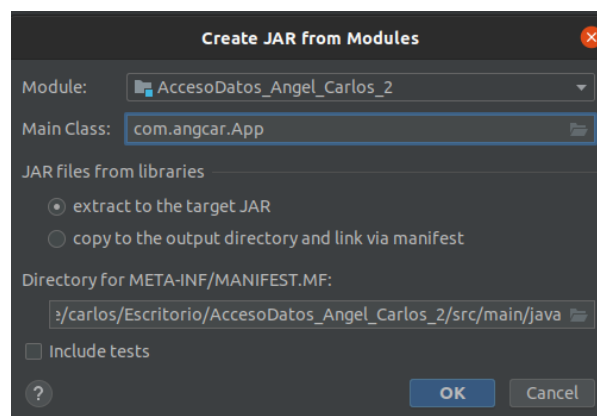
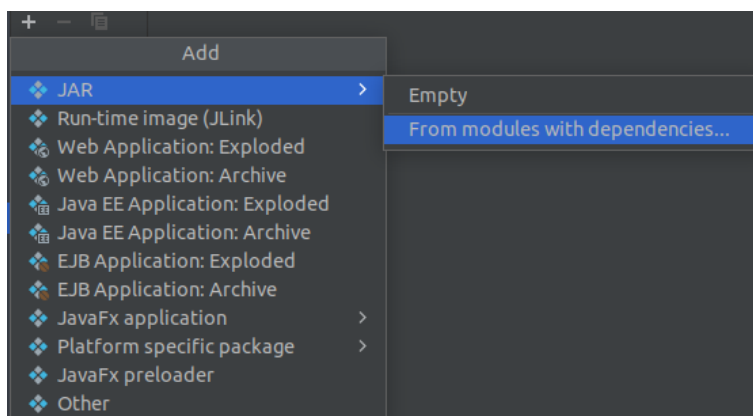
```
<dependency>
  <groupId>jaxen</groupId>
  <artifactId>jaxen</artifactId>
  <version>1.2.0</version>
</dependency>
```

Librería para poder  
realizar las consultas  
de XPATH

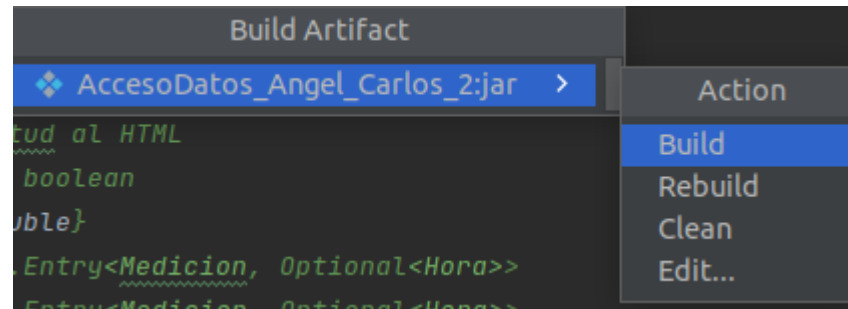
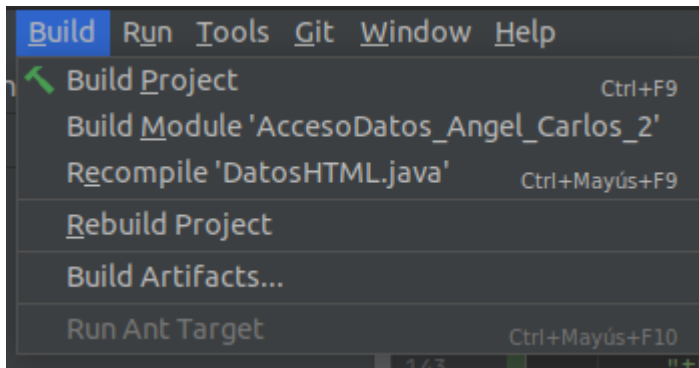
# Prueba ejecución



Esta prueba se ejecutará desde una VM de Linux



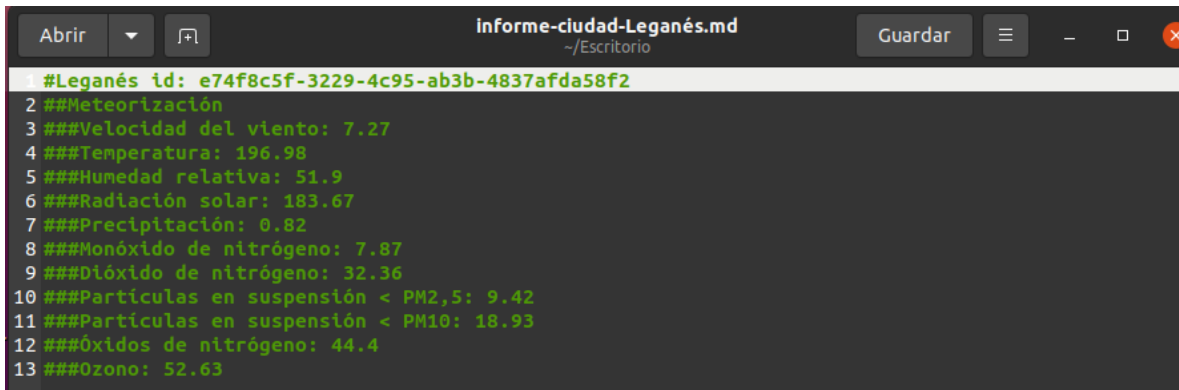
# Prueba ejecución



```
carlos@carlos-virtual-machine:~/Escritorio/AccesoDatos_Angel_Carlos_2$ java -jar /home/carlos/Escritorio/AccesoDatos_Angel_Carlos_2/out/artifacts/AccesoDatos_Angel_Carlos_2_jar/AccesoDatos_Angel_Carlos_2.jar Leganés /home/carlos/Escritorio
Carpeta 'db' creada
Base de datos XML creada.
Leganés id: e74f8c5f-3229-4c95-ab3b-4837afda58f2
Meteorización
Velocidad del viento: 7.27
Temperatura: 196.98
Humedad relativa: 51.9
Radiación solar: 183.67
Precipitación: 0.82
Contaminación
Monóxido de nitrógeno: 7.87
Dióxido de nitrógeno: 32.36
Partículas en suspensión < PM2,5: 9.42
Partículas en suspensión < PM10: 18.93
Óxidos de nitrógeno: 44.4
Ozono: 52.63
```



# Prueba ejecución



A screenshot of a text editor window titled "informe-ciudad-Leganés.md" with a subtitle "~/Escritorio". The editor contains a Markdown document with the following content:

```
#Leganés id: e74f8c5f-3229-4c95-ab3b-4837afda58f2
2 ##Meteorización
3 ###Velocidad del viento: 7.27
4 ###Temperatura: 196.98
5 ###Humedad relativa: 51.9
6 ###Radiación solar: 183.67
7 ###Precipitación: 0.82
8 ###Monóxido de nitrógeno: 7.87
9 ###Dióxido de nitrógeno: 32.36
10 ###Partículas en suspensión < PM2,5: 9.42
11 ###Partículas en suspensión < PM10: 18.93
12 ###Óxidos de nitrógeno: 44.4
13 ###Ozono: 52.63
```

Documento Markdown

## Servicio meteorológico y contaminación

### Leganés

#### Estaciones asociadas:

Leganés

#### Fecha de inicio de la medición:

01/09/2021 - 00:00:00

#### Fecha de fin de la medición:

27/09/2021 - 00:00:00

Documento HTML