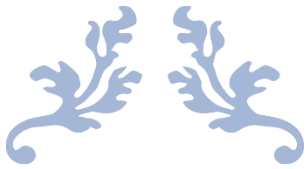

PRÁCTICA XML

ENEKO REBOLLO Y SAÚL MELLADO

Contenido

Paquete Objetos

Clase POJODatos

Creamos nuestra clase POJO con los datos que vamos a necesitar. Utilizamos Lombok

```
@Data
@NoArgsConstructor

public class POJODatos {
    String municipio,estacion,magnitud,fecha;
    List<Double> temperaturas;
    double max,min,media;
}
```

Paquete Csv

Clase CalidadReader

En esta clase creamos un metodo para crear los objetos a partir del csv. Este metodo llama a los métodos(getStringList, que lee las líneas del fichero csv y llama también a getDatos, que mediante el uso del scanner cogemos los datos que necesitamos utilizando el delimitador ;)

```
private void objectGenerator(){

    List<POJODatos> datosCalidad = getStringList().stream().skip(1).map(this::getDatos).collect(Collectors.toList());
    //System.out.println("datos calidad: "+datosCalidad.size());
    listaCalidad = datosCalidad;
}

private List<String> getStringList() {
    try {
        return Files.readAllLines(Path.of(calidadCsv));
    } catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}
```

además en este metodo obtenemos la temperatura máxima, mínima y media llamando al metodo getTemperaturas y getMedia

```
private POJODatos getDatos(String linea){
    POJODatos calidad = new POJODatos();
    Scanner sc = new Scanner(linea);
    sc.useDelimiter(";");
    sc.next();
    sc.next();
    sc.next();
    sc.next();
    String largo = sc.next();
    calidad.setEstacion(largo);
    calidad.setFecha(sc.next()+"/"+sc.next()+"/"+sc.next());
    calidad.setMunicipio(largo.substring(2,5));
    StringTokenizer st = new StringTokenizer(largo, delim: "_");
    st.nextToken();
    calidad.setMagnitud(st.nextToken());

    calidad.setTemperaturas(getTemperaturas(linea));

    Optional<Double> max = calidad.getTemperaturas().stream().max(Comparator.comparing(t->t));
    calidad.setMax(max.get());
    Optional<Double> min = calidad.getTemperaturas().stream().min(Comparator.comparing(t->t));
    calidad.setMin(min.get());
    Double media = getMedia(calidad.getTemperaturas());
    calidad.setMedia(media);

    return calidad;
}
```

El metodo getMedia suma todas las temperaturas y las divide entre el tamaño total de la lista.

El metodo getTemperaturas hacemos uso del delimitador ; para coger los datos que nos interesan y recorremos un for de 24 (que son las horas) y hacemos que si el siguiente valor (que serían las V) es igual a V añadimos el valor.

```java
private Double getMedia(List<Double> temperaturas) {
    double suma= temperaturas.stream().mapToDouble(v → v).sum();
    //System.out.println("suma: "+suma);
    return suma/ temperaturas.size();
}

private List<Double> getTemperaturas(String linea) {
    List<Double> returner = new ArrayList<>();
    //System.out.println(linea);
    Scanner sc = new Scanner(linea);
    sc.useDelimiter(";");

    sc.next();
    sc.next();
    sc.next();
    sc.next();
    sc.next();
    sc.next();
    sc.next();
    sc.next();

    String temp;
    for (int i=0;i<24;i++){
        temp = sc.next();
        if(sc.next().equalsIgnoreCase( anotherString: "V")){
            returner.add(Double.parseDouble(temp));
        }
    }
    //System.out.println("temp: "+returner.size());
    return returner;
}

@Override
public void run() { objectGenerator(); }
```

Clase MeteoReader

Hacemos lo mismo que en la clase CalidadReader pero con el otro csv.

```
private POJODatos getDatos(String linea){
    POJODatos meteo = new POJODatos();
    Scanner sc = new Scanner(linea);
    sc.useDelimiter(";");
    sc.next();
    sc.next();
    meteo.setEstacion(sc.next());
    sc.next();
    String largo = sc.next();
    meteo.setFecha(sc.next()+"/"+sc.next()+"/"+sc.next());
    meteo.setMunicipio(largo.substring(2,5));
    StringTokenizer st = new StringTokenizer(largo, delim: "_");
    st.nextToken();
    meteo.setMagnitud(st.nextToken());

    meteo.setTemperaturas(getTemperaturas(linea));

    if(meteo.getTemperaturas().size()≠0){
        Optional<Double> max = meteo.getTemperaturas().stream().max(Comparator.comparing(t→t));
        meteo.setMax(max.get());
        Optional<Double> min = meteo.getTemperaturas().stream().min(Comparator.comparing(t→t));
        meteo.setMin(min.get());
        Double media = getMedia(meteo.getTemperaturas());
        meteo.setMedia(media);
    }

    return meteo;
}
```

Clase Lanzador

Nos creamos esta clase para implementar el uso de hilos en los que cada una de nuestras 3 clases anteriormente explicadas las metemos en un hilo que a su vez lo metemos en un ThreadGroup.

```
public class Lanzador {
    List<POJODatos> listaCalidad = new ArrayList<>();
    List<POJODatos> listaMeteo = new ArrayList<>();
    ThreadGroup tg = new ThreadGroup( name: "lectoresCSV");

    public void empezar() throws InterruptedException {
        CalidadReader car = CalidadReader.getInstance();
        MeteoReader dmr = MeteoReader.getInstance();
        CreadorMapMunicipios cmm = CreadorMapMunicipios.getInstance();

        Thread hilo1 = new Thread(tg,car);
        Thread hilo2 = new Thread(tg,dmr);
        Thread hilo3 = new Thread(tg,cmm);



        hilo1.start();
        hilo2.start();
        hilo3.start();
        hilo1.join();
        hilo2.join();
        hilo3.join();

        listaCalidad = car.getListaCalidad();
        listaMeteo = dmr.getListaMeteo();

        // System.out.println(calidadList.size()+" "+calidadZonasList.size()+" "+datosList.size());
    }
}
```

Clase CreadorMapMunicipios

En esta clase simplemente recorremos nuestro csv y cogemos el código de municipio y llamamos a los métodos fillCodigoMunicipio y fillCodigoEstaciones que a continuación explicaremos.

```
private void mapMunicipios() throws IOException {
    EstacionesMapas em = EstacionesMapas.getInstance();

    List<String> estacionesList = Files.readAllLines(Path.of(uri), Charset.forName("windows-1252"));

    estacionesList.stream().skip(1).forEach(a->{

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

        String codigo = st.nextToken();
        int codigoMunicipio = Integer.parseInt(codigo.substring(2,5));
        st.nextElement();
        String nombre = st.nextToken();
        nombre = nombre.replace( target: " ", replacement: "_");
        em.fillCodigoMunicipio(codigoMunicipio,nombre);
        em.fillCodigoestaciones(Integer.parseInt(codigo),nombre);

    });
}

@Override
public void run() {
    try {
        mapMunicipios();
    } catch (IOException e) {
        System.err.println("no se pudieron leer los municipios disponibles");
    }
}
```

Paquete Mapas

Clase EstacionesMapas

En esta clase creamos los dos métodos anteriormente comentados, estos métodos rellenan los mapas. El primero asocia el código del municipio a un nombre y el segundo asocia el código de la estación a un nombre.

```java
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> codigoEstacion = new LinkedHashMap<>();

    /**
     * rellenar los mapas
     * @param codigo de la estacion/municipio
     * @param municipio /estacion linkeado al codigo
     */
    public void fillCodigoMunicipio(int codigo, String municipio){
        codigoMunicipio.put(codigo,municipio);
    }

    public void fillCodigoestaciones(int codigo, String estacion) {
        codigoEstacion.put(codigo,estacion);
    }
}
```

Clase MagnitudMap

En esta clase creamos un mapa en el que relaciona el código de la magnitud con el nombre de dicha magnitud.

```
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");
}

public Map<Integer,String> getMapa(){return magnitudes;}
```

Clase UdMedidaMapa

En esta clase asociamos el código de la magnitud con el nombre de la unidad de medida de dicha magnitud.

```
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 XML

Clase GenerarXmlDatos

En esta clase utilizamos Jaxb para crear nuestros dos ficheros xml, creamos dos métodos, uno por cada fichero que llama a los métodos de la clase JaxbCalidad y JaxbMeteo que ahora explicaremos.

```
public class GenerarXmlDatos {
    JaxbCalidad jc = new JaxbCalidad();
    JaxbMeteo jx = new JaxbMeteo();
    private void generarXmlCalidad() throws InterruptedException, JAXBException {
        jc.cargarDatos();
        JAXBContext jaxbContext = JAXBContext.newInstance(JaxbCalidad.class);
        Marshaller jaxbMarshaller = jaxbContext.createMarshaller();
        jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
        String jaxbxml = System.getProperty("user.dir")+ File.separator+"target"+File.separator+"generated-sources"+
        jaxbMarshaller.marshal(jc, new File(jaxbxml));
    }

    private void generarXmlMeteo() throws InterruptedException, JAXBException{
        jx.cargarDatos();
        JAXBContext jaxbContext = JAXBContext.newInstance(JaxbMeteo.class);
        Marshaller jaxbMarshaller = jaxbContext.createMarshaller();
        jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
        String jaxbxml = System.getProperty("user.dir")+ File.separator+"target"+File.separator+"generated-sources"+
        jaxbMarshaller.marshal(jx, new File(jaxbxml));
    }

    public void crearXMLData() throws JAXBException, InterruptedException {
        generarXmlCalidad();
        generarXmlMeteo();
    }
}
```

Clase JaxbCalidad

En esta clase nos creamos una lista y llamamos a la clase lanzador para rellenar nuestra lista.

```
@Data
@XmlRootElement(name = "DatosCalidad")
@XmlAccessorType(XmlAccessType.FIELD)
public class JaxbCalidad {
    List<POJODatos> listaCalidad = new ArrayList<>();
    public void cargarDatos() throws InterruptedException{
        Lanzador l = Lanzador.getInstance();
        l.empezar();
        listaCalidad= l.getListaCalidad();
    }
}
```

Clase JaxbMeteo

En esta clase nos creamos una lista y llamamos a la clase lanzador para rellenar nuestra lista.

```java
@Data
@XmlRootElement(name = "DatosMeteo")
@XmlAccessorType(XmlAccessType.FIELD)
public class JaxbMeteo {
    List<POJODatos> listaMeteo = new ArrayList<>();
    public void cargarDatos() throws InterruptedException{
        Lanzador l = Lanzador.getInstance();
        l.empezar();
        listaMeteo= l.getListaMeteo();
    }
}
```

Clase XmlCreator

En esta clase nos creamos un metodo que genera un xml pasandole un dom por parámetro.

```java
private void generateXML(Document dom){
    XMLOutputter xml = new XMLOutputter(Format.getPrettyFormat());

    String uri = System.getProperty("user.dir")+File.separator+"target"+File.separator+"db";
    File dir = new File(uri);
        if(!dir.exists()){
            dir.mkdirs();
        }

    try {
        BufferedWriter bw = new BufferedWriter(new FileWriter( fileName: uri+File.separator+"mediciones.xml"));
        xml.output(dom,bw);


    } catch (IOException e) {
        e.printStackTrace();
    }

}
```

Nos creamos también un metodo llamado crear hijo y le pasamos por parámetro el elemento root, el municipio, la magnitud y un boolean que ahora comentaremos.

Este metodo simplemente hace que si el boolean es true añade la etiqueta children listaCalidad al xml y si no añade la etiqueta listaMeteo.

Recorremos una lista y le vamos añadiendo el texto de las etiquetas del xml a la lista, además añadimos a las listas la máxima, mínima y media y por ultimo las agregamos como elemento mediante addcontent.

```
private Element crearHijo(Element rootElement, int municipio, int magnitud, boolean tipo) {
    if(mm.getMapa().containsKey(magnitud)) {
        Element medicion = new Element( name: "medicion");
        medicion.setAttribute( name: "nombre",mm.getMapa().get(magnitud));
        Element maximo = new Element( name: "maximo_mensual");
        Element minimo = new Element( name: "minimo_mensual");
        Element media = new Element( name: "media_mensual");

        List<Element> datos = null; //aqui esta la mierda
        if(tipo==true){
            datos = rootElement.getChildren( cname: "listaCalidad");
        }
        else datos = rootElement.getChildren( cname: "listaMeteo");
        List<Double> maximas = new ArrayList<>();
        List<Double> minimos = new ArrayList<>();
        List<Double> medias = new ArrayList<>();
        datos.forEach(v -> {
            if (Integer.parseInt(v.getChild("municipio").getText())==municipio && v.getChild("magnitud").getText().equalsIgnoreCase(String.valueOf(magnitud))) {
                maximas.add(Double.parseDouble(v.getChild("max").getText()));
                minimos.add(Double.parseDouble(v.getChild("min").getText()));
                medias.add(Double.parseDouble(v.getChild("media").getText()));
            }
        });
        if(maximas.size()≠0) {
            Optional<Double> max = maximas.stream().max(Comparator.comparing(v -> v));

            maximo.setText(max.get()+" "+umm.getUdMedida().get(magnitud));
        }
        if(minimos.size()≠0) {
            Optional<Double> min = minimos.stream().min(Comparator.comparing(v -> v));
            minimo.setText(min.get()+" "+umm.getUdMedida().get(magnitud));
        }
        if(medias.size()≠0) {
            double med = (medias.stream().mapToDouble(v -> v).sum()) / medias.size();
            media.setText(med + " " + umm.getUdMedida().get(magnitud));
        }
        medicion.addContent(maximo);
        medicion.addContent(minimo);
        medicion.addContent(media);
        return medicion;
    }
    return null;
}
```

Creamos también nuestro metodo crearXML que a partir del municipio pasador por parámetro mediante SAX leemos los datos de nuestros dos ficheros xml y recorre un for para sacar las mediciones, llama al metodo crearHijo, si no es nulo añade el contenido y por ultimo llama al metodo generateXML anteriormente comentado.

```
public void crearXML(int municipio){
    Element root = new Element( name: "datos");
    Element municipioElement = new Element( name: "municipio");
    municipioElement.setAttribute( name: "nombre",em.getCodigoMunicipio().get(municipio));
    Element rootCalidad = null;
    Element rootMeteo = null;
    Element calidad = new Element( name: "calidad_aire");
    Element meteo = new Element( name: "datos_meteorologicos");

    SAXBuilder sax = new SAXBuilder();
    try{
        rootCalidad=sax.build( systemId: uriInput+File.separator+"datosCalidad.xml").getRootElement();
        rootMeteo=sax.build( systemId: uriInput+File.separator+"datosMeteo.xml").getRootElement();
    } catch (IOException | JDOMException e) {
        e.printStackTrace();
    }

    for (int i=0;i<432;i++){
        if(i<81 || i==431) {
            Element elemento = crearHijo(rootCalidad, municipio,i, tipo: true);
            if(elemento≠null){
                calidad.addContent(elemento);
            }
        }
        if(i≥81 && i≠431) {
            Element elemento = crearHijo(rootMeteo, municipio,i, tipo: false);
            if(elemento≠null){
                meteo.addContent(elemento);
            }
        }
    }
    municipioElement.addContent(calidad);
    municipioElement.addContent(meteo);
    root.addContent(municipioElement);
    dom.setRootElement(root);
    generateXML(dom);
```

Paquete Xpath

Clase XpathManager

En esta clase simplemente leemos mediante una expresión de xpath la información de nuestro fichero mediciones.xml y devolvemos los datos del nombre de la medición junto con sus valores.

```java
public class XpathManager {

    Document dom = new Document();
    String uri = System.getProperty("user.dir")+File.separator+"target"+File.separator+"db"+File.separator+"mediciones.xml";
    List<String> listaMedias;

    private void loadData() throws IOException, JDOMException {

        SAXBuilder builder = new SAXBuilder();
        File xmlFile = new File(this.uri);
        this.dom = (Document) builder.build(xmlFile);
    }
    public List<String> obtenerMediasMensuales() {
        try {
            loadData();
        } catch (IOException | JDOMException e) {
            e.printStackTrace();
        }
        XPathFactory xpath = XPathFactory.instance();
        XPathExpression<Element> expr = xpath.compile( expression: "//medicion", Filters.element());
        List<Element> medias = expr.evaluate(this.dom);
        List<String> listaMedias = new ArrayList<>();
        medias.forEach(m -> {
            String valor =  m.getChild("media_mensual").getText();
            if (valor.equalsIgnoreCase( anotherString: "")){
                valor = "no hay valores sobre esta medicion";
            }
            String aniadir = m.getAttributeValue( attname: "nombre") + ": " + valor;
            listaMedias.add(aniadir);

        });
        return listaMedias;
    }
```

Paquete MarkDown

Clase MDCreator

Esta clase simplemente tiene un metodo que pasandole una uri y un municipio por parámetro crea un archivo .md y llama al metodo de la clase XpathManager anteriormente comentada para sacar toda la lista de medias mensuales.

```java
public class MDCreator {

    public void mdCreator(String uri, int municipio){
        File archivo = new File(uri);
        if(!archivo.exists()){
            archivo.mkdirs();
        }
        try(BufferedWriter bw = new BufferedWriter(new FileWriter( fileName: uri+File.separator+"datos.md"));) {
            XpathManager manager = new XpathManager();
            bw.write( str: "## Datos de "+municipio+"\n");
            manager.obtenerMediasMensuales().forEach(v->{
                try {
                    bw.write( str: "* "+v+"\n");
                } catch (IOException e) {
                    e.printStackTrace();
                }
            });

            bw.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Clase Funcional

En esta clase nos creamos un metodo que vaya llamando a los diferentes métodos necesarios para que funcione el programa. Le pasamos un municipio y una uri por parámetro. Además también llamamos a los métodos chechFile, municipioExists, crearCarpeta y ejecutarMd que ahora explicaremos.

```java
public void start(String municipio, String uri) throws IOException, InterruptedException, JAXBException {
    Long startTime = System.currentTimeMillis();
    EstacionesMapas em = EstacionesMapas.getInstance();

    checkFile(uri,municipio.toLowerCase());
    municipioExists(municipio);
    crearCarpeta(uri);
    if (codMunicipio==0){
        System.out.println("municipio no encontrado");
    }
    else {
        Lanzador l = Lanzador.getInstance();
        XmlCreator xmlc = new XmlCreator();
        xmlc.crearXML(codMunicipio);
        XpathManager manager = new XpathManager();
        manager.obtenerMediasMensuales().forEach(System.out::println);
        MDCreator md = new MDCreator();
        md.mdCreator(uri,codMunicipio);
    }
}
```

El metodo municipioExists comprueba que el municipio introducido existe mediante el metodo de la clase EstacionesMapas anteriormente comentada.

```java
private void municipioExists(String municipio) throws InterruptedException {
    Lanzador launcher = Lanzador.getInstance();
    launcher.empezar();
    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 = entry.getKey();
            }
        }
    }
}
```

El metodo crear carpeta crea una carpeta en la uri especificada.

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

Este metodo comprueba que el fichero existe y en caso de que exista nos pregunta si queremos sobrescribirlo o no.

```
private void checkFile(String uri,String municipio){
    File file = new File( pathname: uri+File.separator+"datos.md");
    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( anotherString: "no")){
            System.out.println("proceda a sacar de la carpeta el archivo");
            System.exit( status: 0);
        }
    }
}
```

El metodo ejecutarMd simplemente ejecuta nuestro fichero Markdown en nuestro navegador predeterminado del sistema.

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

Clase Main

Nuestra clase main simplemente llama al metodo start de la clase Funcional y nos obliga a meter dos parámetros para que se pueda ejecutar.

```
public static void main(String[] args) throws IOException, InterruptedException, JDOMException, JAXBException {
        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");
        }
    }
}
```