

Acceso a Datos, Practica 1.

Alumno: Javier González Rives.

Asignatura: Acceso a Datos.

Curso: 2º DAM.

1. Creación de Un fichero serializado.	2
a) Explicación Resumida.	2
b) Ejemplo departamentos	2
c) Resultado	8
2 Escritura lectura a través de DOM.	9
a) Explicación resumida	9
b) Ejemplo Departamentos.	9
c) Resultado.	13
3. Creación y lectura de ficheros XML con Xstream.	17
a) Explicación	17
b) Principales Métodos.	17
c) Ejemplo Departamentos.	17
d) Resultado	18
4. Transformaciones XML.	21
a) Explicación.	21
b) Clases y métodos para la transformación	22
Clases.	22
Métodos.	22
c) Ejemplo Departamentos:	23
5. Cuestiones sobre el tema.	25
a) ¿Sigue Xstream la filosofía DOM?	25
b) ¿Qué utilidad crees que tiene la transformación a HTML?	25
c) ¿es posible transformar a otros formatos?	25
Repositorio con el Código:	26
Bibliografía:	26

1. Creación de Un fichero serializado.

a) Explicación Resumida.

Serialización

El paradigma de programación actual por excelencia es el orientado objetos, siendo un objeto una unidad de representación de la realidad dentro de un programa informático formado por atributos con valores y funciones que realizan tareas en tiempo de ejecución, definición que es muy práctica para entender el concepto, los objetos existen en tiempo de ejecución pero una vez que todos los procesos del programa terminan estos desaparecen perdiéndose la información que estos guardaban, para que esta información no se pierda es necesario pasarla a un medio persistente, aquel que no se pierde tras la ejecución, debido a la estructura que tienen los objetos no pueden ser escritos en un fichero de forma directa por lo que nos vemos obligados a guardar sus datos por separado para más tarde volver a recuperarlos y crear un objeto nuevo con esos valores, para evitar este proceso se utiliza la serialización. Cuando un objeto es serializable este puede ser transformado o codificado en un flujo de bits que pueden ser escritos de forma directa en medios como ficheros o transmitidos a través de una red siendo posible más tarde recuperar la totalidad no solo de sus valores sino de su estado teniendo un clon idéntico al objeto almacenado, reduciendo la complejidad y consiguiendo formas más eficientes de gestionar la información.

b) Ejemplo departamentos

Para este ejemplo se ha creado una clase llamada Departamentos, en esta se han implementado ,además de otros métodos que más tarde se explicarán, tanto los métodos para la modificación de valores del objeto como un método para la representación de la información de este por terminal, los valores que almacena Departamentos son:

Nombre	Tipo
id	String
tipo	String
nombre	String
domicilio	String
ciudad	String
codigoPostal	String

provincia	String
país	String

Codigo Departamentos:

```
public class Departamentos implements Serializable{
    private String id;
    private String tipo;
    private String nombre;
    private String domicilio;
    private String ciudad;
    private String codigoPostal;
    private String provincia;
    private String pais;
    private static final long serialVersionUID = 1L;
    /**
     * constructor de departamentos
     */
    public Departamentos(){
        super();
        // creacion de los datos a null
        id = null;
        tipo = null;
        nombre = null;
        domicilio = null;
        ciudad = null;
        codigoPostal = null;
        provincia = null;
        pais = null;
    }
    /**
     * constructor con todos los parametros de la clase
     * @param id
     * @param tipo
     * @param nombre
     * @param domicilio
     * @param ciudad
     * @param codigoPostal
     * @param provincia
     * @param pais
     */
    public Departamentos(String id, String tipo, String nombre, String domicilio, String
ciudad, String codigoPostal, String provincia, String pais) {
        this.id = id;
        this.tipo = tipo;
        this.nombre = nombre;
        this.domicilio = domicilio;
```

```
this.ciudad = ciudad;  
this.codigoPostal = codigoPostal;  
this.provincia = provincia;  
this.pais = pais;  
}
```

```
// GETTERS SETTERS
```

```
public String getId() {  
    return id;  
}
```

```
public void setId(String id) {  
    this.id = id;  
}
```

```
public String getTipo() {  
    return tipo;  
}
```

```
public void setTipo(String tipo) {  
    this.tipo = tipo;  
}
```

```
public String getNombre() {  
    return nombre;  
}
```

```
public void setNombre(String nombre) {  
    this.nombre = nombre;  
}
```

```
public String getDomicilio() {  
    return domicilio;  
}
```

```
public void setDomicilio(String domicilio) {  
    this.domicilio = domicilio;  
}
```

```
public String getCiudad() {  
    return ciudad;  
}
```

```
public void setCiudad(String ciudad) {  
    this.ciudad = ciudad;  
}
```

```

    }

    public String getCodigoPostal() {
        return codigoPostal;
    }

    public void setCodigoPostal(String codigoPostal) {
        this.codigoPostal = codigoPostal;
    }

    public String getProvincia() {
        return provincia;
    }

    public void setProvincia(String provincia) {
        this.provincia = provincia;
    }

    public String getPais() {
        return pais;
    }

    public void setPais(String pais) {
        this.pais = pais;
    }

    @Override
    public String toString() {
        return "departamentos{" + "id=" + id + ", tipo=" + tipo + ", nombre=" + nombre + ",
domicilio=" + domicilio + ", ciudad=" + ciudad + ", codigoPostal=" + codigoPostal + ",
provincia=" + provincia + ", pais=" + pais + '}';
    }

```

Como se puede observar ni el “id” ni el “codigoPostal” son datos de tipo numérico esto es así ya que no son datos con los que se realicen operaciones, siendo más fácil gestionarlos como cadenas de texto quedando eliminadas desventajas como la limitación de tamaño.

Para que el objeto instanciado sea serializable debemos implementar la interfaz “Serializable”, para que esta sea funcional debemos crear una constante estática llamada “serialVersionUID” de tipo long, el valor estándar que se le suele dar es 1L, una vez cumplidas estas dos condiciones el objeto instanciado será oficialmente serializable. Para realizar el proceso de escritura he creado una función que le permite al objeto de tipo Departamento guardar en un fichero de datos su información en el directorio que le indiquemos.

Método saveData:

```
public void saveData(String nombreFichero,String directorio)throws IOException{
    File dir = new File(directorio);
    File fichero = new File(directorio,nombreFichero + ".dat");
    // comprobacion que el directorio existe
    if(dir.exists()){
        // creacion del flujo
        FileOutputStream flujo = new FileOutputStream(fichero);
        // formateo de la salida
        ObjectOutputStream escritor = new ObjectOutputStream(flujo);
        // escritura
        escritor.writeObject(this);
        // cerrado de escritor
        escritor.close();
        // cerrado del flujo
        flujo.close();
    }else{
        // caso que la carpeta introducida no exista
        throw new IOException("la carpeta no existe");
    }
}
```

El método a través de los parámetros que le hemos indicado creará un fichero con el objeto, este método requiere de controlar las excepciones, en primer lugar se escribe un flujo de datos al fichero para más tarde usar ese flujo con “ObjectOutputStream” y escribir el objeto como tal, una vez realizada la escritura se cierran todos los flujos.

Para después poder recuperar la información existe un método estático que lee el fichero y devuelve un objeto de tipo Departamentos:

Método leerFichero:

```
public static Departamentos leerFichero(File dir) throws FileNotFoundException,
IOException, ClassNotFoundException{
    // apertura del flujo
    FileInputStream flujo = new FileInputStream(dir);
    // creacion del lector
    ObjectInputStream lector = new ObjectInputStream(flujo);

    // creacion de un departamento nuevo
    Departamentos dep = (Departamentos)lector.readObject();
    // retorno del departamento
    return dep;}
}
```

Se trata simplemente de un flujo de lectura binario que a través de “ObjectInputStream” lee la información del fichero recuperando el objeto anteriormente serializado, como

anteriormente se ha dicho podemos recuperar una gran cantidad de información además de poder manipularla de forma sencilla o incluso operar.

Para poder escribir varios registros en un mismo documentos podemos hacerlo de diferentes maneras, la más sencilla consiste en crear una lista con varios objetos de tipo departamentos, de esta forma guardaremos un único objeto que almacene dentro varios Departamentos.

metodo saveDataList:

```
public static void saveDataList(ArrayList<Departamentos> list,File dir,String nombre) throws
FileNotFoundException, IOException{
    FileOutputStream flujo = new FileOutputStream(new File(dir,nombre+".dat"));
    ObjectOutputStream escritor = new ObjectOutputStream(flujo);
    escritor.writeObject(list);
}
```

Para este método simplemente le pasamos el directorio con el nombre del archivo y el arrayList con los departamentos que vamos a escribir.

para recuperarlo se hace el mismo proceso pero a la inversa

metodo readDataList:

```
public static ArrayList<Departamentos> readDataList(File dir) throws
FileNotFoundException, IOException, ClassNotFoundException{
    ArrayList<Departamentos> dep = null;
    FileInputStream flujo = new FileInputStream(dir);
    ObjectInputStream lector = new ObjectInputStream(flujo);
    dep = (ArrayList<Departamentos>) lector.readObject();
    return dep;
}
```

c) Resultado

para la ejecución se ha creado una clase con un método "main" para iniciar un programa donde se usen los métodos anteriormente señalados.

```
public class PrimerEjercicio {
    public static void main(String[] args) {
        File dir = new File("src/departamentos/departamento.dat");
        Departamentos dep = new Departamentos();
        // definicion de atributos
        dep.setId("0001");
        dep.setCiudad("Almeria");
        dep.setCodigoPostal("04007");
        dep.setDomicilio("calle el peru nº 12");
        dep.setNombre("recursos humanos");
        dep.setPais("Espana");
    }
}
```



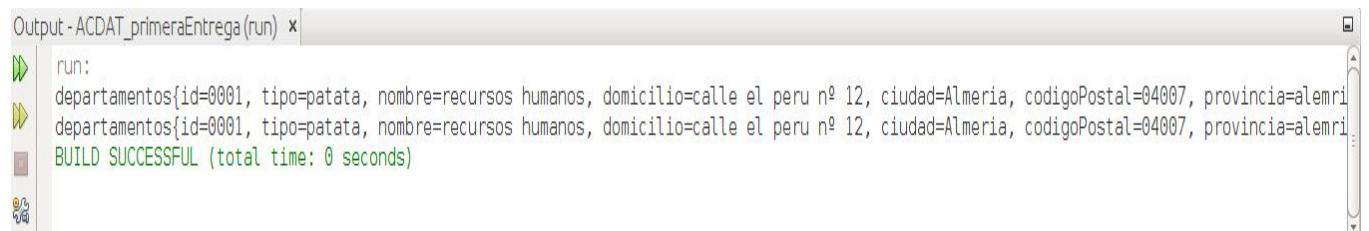
```

        dep.setTipo("patata");
        dep.setProvincia("alemria");
        ArrayList<Departamentos> list = new ArrayList<Departamentos>();
        list.add(dep);
        try{
            // escritura y lectura de comprobacion
            dep.saveData("departamento", "src/departamentos");
            System.out.println(Departamentos.leerFichero(dir).toString());
            Departamentos.saveDataList(list,new
File("src/departamentos"),"departamentosList");
            System.out.println(Departamentos.readDataList(new
File("src/departamentos","departamentosList.dat")).get(0).toString());
        }catch(ClassNotFoundException cl){
            cl.printStackTrace();
        }catch(IOException io){
            io.printStackTrace();
        }
    }
}

```

En este se crea un departamento vacío y con sus modificadores se van añadiendo los valores, más se usa el método saveData para guardar la información en un fichero, Para realizar este proceso con varios departamentos se ha utilizado el método anteriormente mostrado con un arrayList.

Salida:



```

Output - ACDAT_primeraEntrega (run) x
run:
departamentos{id=0001, tipo=patata, nombre=recursos humanos, domicilio=calle el peru nº 12, ciudad=Almeria, codigoPostal=04007, provincia=alemria}
departamentos{id=0001, tipo=patata, nombre=recursos humanos, domicilio=calle el peru nº 12, ciudad=Almeria, codigoPostal=04007, provincia=alemria}
BUILD SUCCESSFUL (total time: 0 seconds)

```

- Ejecución exitosa del programa de guardado

La salida por pantalla corresponde la información del objeto después de haberlo leído desde el fichero, en el primero se ha leído desde un fichero con un departamento y el segundo desde una lista de departamentos.

2 Escritura lectura a través de DOM.

a) Explicación resumida

Para almacenar la información de un objeto en un xml existen diversas formas, una de las principales es a través del método DOM, este método lo que hace es crear una estructura en memoria para representar un XML, esta estructura está representada por “nodos” relacionados entre si, dependiendo de que almacenen estos son de un tipo o de otro, una vez formada esta estructura con otra serie de métodos se hace el proceso de transformación quedando formado un archivo XML. En el caso de la estructura el proceso

es muy parecido, se crea un “parser” con el cual se leen todas las etiquetas del XML y son transformadas en una estructura DOM en memoria, usando algunos métodos como “NodeList” se saca un listado de todos los nodos contenidos y así pudiendo recorrer la información del documento, De cualquier manera es necesario siempre conocer la estructura del documento.

b) Ejemplo Departamentos.

Para este caso se han creado dos métodos distintos a través del método DOM, en el primero a través de una instancia de departamentos se puede guardar en un archivo XML, siendo un solo departamento, en el segundo a través de una lista de Departamentos se guardan en un XML, en este caso siendo varios departamentos.

Para un solo departamento a como se ha dicho es a través de la propia instancia, por lo que un objeto Departamentos se guarda a sí mismo.

método saveXML:

```
public void saveXML(String nombreFichero, String dir) throws ParserConfigurationException,
TransformerConfigurationException, TransformerException, FileNotFoundException{
    // factory para crear el parser
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    // creacion del del parser
    DocumentBuilder builder = factory.newDocumentBuilder();
    // creacion de la implementacion
    DOMImplementation implementacion = builder.getDOMImplementation();
    // creacion del documento
    Document document = implementacion.createDocument(null,"departamentos" , null);
    // cambio version XML
    document.setXmlVersion("1.0");
    // captura del elemento raiz del documento
    Element raiz = document.getDocumentElement();

    // creacion de los sub elementos
    Element idEl = document.createElement("id");
    Element tipoEl = document.createElement("tipo");
    Element nombreEl = document.createElement("nombre");
    Element domicilioEl = document.createElement("domicilio");
    Element ciudadEl = document.createElement("ciudad");
    Element cpEl = document.createElement("cp");
    Element provinciaEl = document.createElement("provincia");
    Element paisEl = document.createElement("pais");

    // creacion del texto
```

```

Text idTx = document.createTextNode(this.id);
Text tipoTx = document.createTextNode(this.tipo);
Text nombreTx = document.createTextNode(this.nombre);
Text domicilioTx = document.createTextNode(this.domicilio);
Text ciudadTx = document.createTextNode(this.ciudad);
Text cpTx = document.createTextNode(this.codigoPostal);
Text provinciaTx = document.createTextNode(this.provincia);
Text paisTx = document.createTextNode(this.pais);

// annadir a la estructura
raiz.appendChild(idEl);
raiz.appendChild(tipoEl);
raiz.appendChild(nombreEl);
raiz.appendChild(domicilioEl);
raiz.appendChild(ciudadEl);
raiz.appendChild(cpEl);
raiz.appendChild(provinciaEl);
raiz.appendChild(paisEl);
// annadir textos
idEl.appendChild(idTx);
tipoEl.appendChild(tipoTx);
nombreEl.appendChild(nombreTx);
domicilioEl.appendChild(domicilioTx);
ciudadEl.appendChild(ciudadTx);
cpEl.appendChild(cpTx);
provinciaEl.appendChild(provinciaTx);
paisEl.appendChild(paisTx);

// escritura del documento
Source source = new DOMSource(document);
//FileOutputStream out = new FileOutputStream(new File(dir,nombreFichero+".xml"));
Result result = new StreamResult(new File(dir,nombreFichero + ".xml"));
// creacion del transformer
Transformer trans = TransformerFactory.newInstance().newTransformer();
// transformacion
trans.transform(source, result);
}

```

El proceso se puede dividir en tres partes diferenciadas, en la primera se crea el Parser para implementar el método DOM con “DocumentBuilderFactory” y “DocumentBuilder”, el primero nos sirve para crear las instancias del parser y el segundo es el parser en sí, el siguiente paso es crear una estructura DOM, el elemento principal es “Document” que almacena toda la estructura además de ser elemento que sirve para crear los Nodos (node). En primer lugar se extrae el Elemento Raíz con el método “getDocumentElement”, para crear el resto de elementos o nodos se usa el método “createElement” o “createText”, una vez creados los nodos se van enlazando con el metodo “appendChild” quedando como “padre.appendChild(hijo)” en caso de los nodos hoja el elemento hijo es un “Text”(Tanto

Element cómo Text heredan de Node). Finalmente se crea un “Transformer” que se encarga de realizar la transformación de DOM en memoria a XML.

En el caso de realizar esta acción con varios Departamentos como se ha dicho se haría a través de una colección de Departamentos (ArrayList) .

método saveXMLList:

```
public static void saveXMLList(ArrayList<Departamentos> deps, File dir, String
nombreFichero) throws ParserConfigurationException, TransformerConfigurationException,
TransformerException{
```

```
    // creacion del factory
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    // creacion de un builder/parser
    DocumentBuilder builder = factory.newDocumentBuilder();
    // creacion de un domImplementation
    DOMImplementation impl = builder.getDOMImplementation();
    // Creacion del documento
    Document document = impl.createDocument(null, "departamentos", null);
    document.setXmlVersion("1.0");
    Element raiz = document.getDocumentElement();
    Element aux;
    for(Departamentos depAux: deps){
        // creacion de un elemento departamento nuevo
        aux = document.createElement("departamento");
        // enlace del departamento con el elemento raiz
        raiz.appendChild(aux);
        // creacion de los sub elementos
```

```
        Element idEl = document.createElement("id");
        Element tipoEl = document.createElement("tipo");
        Element nombreEl = document.createElement("nombre");
        Element domicilioEl = document.createElement("domicilio");
        Element ciudadEl = document.createElement("ciudad");
        Element cpEl = document.createElement("cp");
        Element provinciaEl = document.createElement("provincia");
        Element paisEl = document.createElement("pais");
```

```
        // creacion del texto
        Text idTx = document.createTextNode(depAux.getId());
        Text tipoTx = document.createTextNode(depAux.getTipo());
        Text nombreTx = document.createTextNode(depAux.getNombre());
        Text domicilioTx = document.createTextNode(depAux.getDomicilio());
        Text ciudadTx = document.createTextNode(depAux.getCiudad());
        Text cpTx = document.createTextNode(depAux.getCodigoPostal());
        Text provinciaTx = document.createTextNode(depAux.getProvincia());
```

```

Text paisTx = document.createTextNode(depAux.getPais());

// annadir a la estructura
aux.appendChild(idEl);
aux.appendChild(tipoEl);
aux.appendChild(nombreEl);
aux.appendChild(domicilioEl);
aux.appendChild(ciudadEl);
aux.appendChild(cpEl);
aux.appendChild(provinciaEl);
aux.appendChild(paisEl);
// annadir textos
idEl.appendChild(idTx);
tipoEl.appendChild(tipoTx);
nombreEl.appendChild(nombreTx);
domicilioEl.appendChild(domicilioTx);
ciudadEl.appendChild(ciudadTx);
cpEl.appendChild(cpTx);
provinciaEl.appendChild(provinciaTx);
paisEl.appendChild(paisTx);
}
// escritura del documento
Source source = new DOMSource(document);
//FileOutputStream out = new FileOutputStream(new File(dir,nombreFichero+".xml"));
Result result = new StreamResult(new File(dir,nombreFichero + ".xml"));
// creacion del transformer
Transformer trans = TransformerFactory.newInstance().newTransformer();
// transformacion
trans.transform(source, result);
}

```

En este caso la estructura es prácticamente idéntica al caso anterior con la diferencia que en este se ha usado un bucle para recorrer la colección e ir creando sub elementos del tipo documento con todos sus atributos.

c) Resultado.

Para la ejecución se han creado dos clases distintas, en la primera se guarda un solo Departamento en la segunda se guardan varios.

Ejecución 1: un solo Departamento

```
public class FicheroAXml {
```

```

private static final File dir = new File("src/departamentos/departamento.dat");
public static void main(String[] args) {
    Departamentos dep = null;
    try{
        // lectura del fichero
        dep = Departamentos.leerFichero(dir);
        // salida comprobacion
        System.out.println(dep.toString());
        // escritura de xml
        dep.saveXML("departamento2", "src/departamentos");
    }catch(NullPointerException nu){
        System.out.println("Error, el departamento es nulo");
    }catch(ClassNotFoundException cl){
        System.out.println("la clase no se encontrado");
        cl.printStackTrace();
    }catch(IOException io){
        System.out.println("error en la lectura");
        io.printStackTrace();
    }catch(ParserConfigurationException par){
        System.out.println("error en la configuracion del parser");
        par.printStackTrace();
    }catch(TransformerException tr){
        System.out.println("ha sucedido un error durante la transformacion");
        tr.printStackTrace();
    }
}
}
}

```

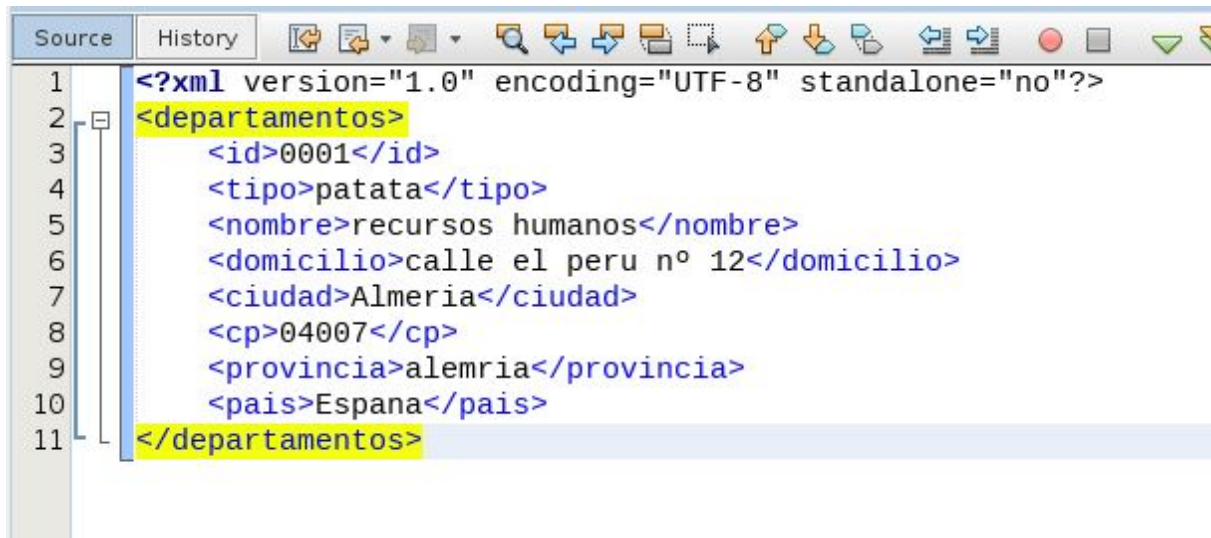
Para la lectura se ha usado el método anteriormente enseñado para recuperar el departamento desde el fichero binario serializado, después se ha usado el método “saveXML” para escribir el fichero.

```

tput - ACDAT_primerEntrega (run)
run:
departamentos{id=0001, tipo=patata, nombre=recursos humanos, domicilio=calle el peru nº 12, ciudad=Almeria, codigoPostal=04007, provincia=alemeria, pais=Espana}
BUILD SUCCESSFUL (total time: 0 seconds)

```

- *ejecución exitosa*



```

1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <departamentos>
3      <id>0001</id>
4      <tipo>patata</tipo>
5      <nombre>recursos humanos</nombre>
6      <domicilio>calle el peru nº 12</domicilio>
7      <ciudad>Almeria</ciudad>
8      <cp>04007</cp>
9      <provincia>alemria</provincia>
10     <pais>Espana</pais>
11 </departamentos>

```

- *fichero xml resultante*

Ejecución 2 Varios departamentos:

```

public class DomMultiDepartamento {
    private static final File dir = new File("src/departamentos");
    // metodo principal
    public static void main(String[] args) {
        // creacion de una lista para almacenar
        ArrayList<Departamentos> deps;

```

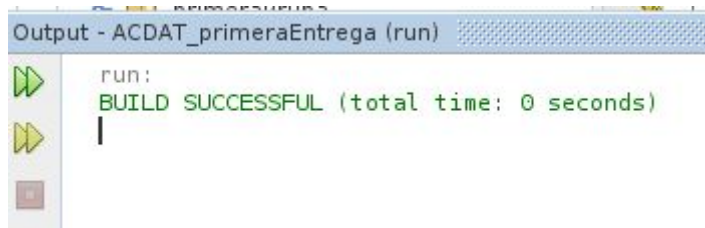
```

try{
// lectura del fichero
deps = Departamentos.readDataList(new File(dir,"departamentosList.dat"));
// escritura del fichero xml
Departamentos.saveXMLList(deps, dir, "departamentosListXml");
}catch(IOException io){
System.out.println("error entrada salida");
io.printStackTrace();
}catch(ClassNotFoundException cl){
System.out.println("clase no encontrada");
cl.printStackTrace();
}catch(ParserConfigurationException par){
System.out.println("error en el parseo");
par.printStackTrace();
}catch(TransformerException tr){
System.out.println("error en la transformacion");
tr.printStackTrace();
}

}
}

```

Para rescatar la información he usado el método “readDataList” que rescata la información en forma de lista de departamentos, para la escritura “saveXMLList”.



- *ejecución exitosa*


```

1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <departamentos>
3    <departamento>
4      <id>0001</id>
5      <tipo>patata</tipo>
6      <nombre>recursos humanos</nombre>
7      <domicilio>calle el peru nº 12</domicilio>
8      <ciudad>Almeria</ciudad>
9      <cp>04007</cp>
10     <provincia>alemria</provincia>
11     <pais>Espana</pais>
12   </departamento>
13   <departamento>
14     <id>0001</id>
15     <tipo>patata</tipo>
16     <nombre>recursos humanos</nombre>
17     <domicilio>calle el peru nº 12</domicilio>
18     <ciudad>Almeria</ciudad>
19     <cp>04007</cp>
20     <provincia>alemria</provincia>
21     <pais>Espana</pais>
22   </departamento>
23 </departamentos>

```

- *xml resultante*

3. Creación y lectura de ficheros XML con Xstream.

a) Explicación

Al igual que con los archivos binarios es posible formar un fichero que almacene un objeto a través de la serialización, esta también se puede usar para guardar su información en un XML, este proceso lo haremos a través de Xstream. Esta librería no pertenece a la api de java por lo que deberá ser incluida en nuestro proyecto de forma manual para ello bien la introduciremos en el árbol de directorios del proyecto o dentro del XML de recursos del proyecto, una vez hecho uno de estos pasos podremos usar las clases de esta librería.

Como se ha dicho anteriormente esta librería permite con un objeto serializable generar un XML este proceso tiene dos maneras de hacerse, una a través de etiquetas dentro de la clase, en este caso no se va a usar, consiste en añadir unos comentarios dentro de clase que más tarde guiarán el proceso de definir la estructura del fichero XML dejando de una manera más concreta muy recomendable si se desea hacer de una forma en específico, el segundo método consiste en indicar la clase con el nombre de su etiqueta principal y el propio Xstream se encargará de analizarla y poner nombre a las etiquetas conteniendo

estas los valores del objeto, en el caso de tener sub objetos deberemos indicarlo de la misma manera que con el principal, finalmente nos devolverá un xml formado.

En el caso contrario para leer un fichero xml y pasarlo a objeto este mismo objeto nos servirá únicamente indicando el directorio del fichero.

b) Principales Métodos.

- Constructores: sirve para crear una instancia de Xstream, tiene varios con distintos parámetros ya que tiene diversas formas de funcionar un ejemplo sería la implementación de un driver.
- alias: este método nos sirve definir el nombre de la etiqueta de un determinado tipo de objeto (su class) por ejemplo que los tipo "Departamentos" aparezcan como "departamento".
- useAttributeFor: este método nos permite que cierto valor de la clase se muestre en forma de atributo en el xml, para ello indicamos el tipo de clase y el nombre del campo, para definir el nombre del atributo en el xml usaremos el "alias" para cambiarlo.
- toXML: este método nos servirá para hacer la transformación de objeto a XML, nos da varias opciones para hacer esto, desde generar un String con el xml a indicarle un flujo y directamente crear un fichero.
- fromXML: este método des serializa el xml desde una fuente indicada, como un string o un flujo, y devuelve el objeto guardado.

c) Ejemplo Departamentos.

Como en todos los ejemplos anteriores para el caso de solo realizar el proceso en un solo departamento la instancia de este tiene un método que realiza el proceso de guardar su información dentro de un xml, este implementa el método xstream.

```
public void saveXMLXstream(String nombre, String dir) throws FileNotFoundException,
IOException{
```

```
    // para poder usar este metodo es necesario importar el jar
```

```
    // creacion de un Xstream nuevo para la creacion del xml
```

```
    XStream xstream = new XStream();
```

```
    // indicar la clase que se ca usar para combertir
```

```
    xstream.alias("departamento", Departamentos.class);
```

```
    /*
```

```
    en este punto le pasamos la instancia del objeto que queremos combertir
    el dato que nos devuelve es de tipo cadena la cual nos servira para escribir
    mas tarde el documento en un fichero, para escribir directamente debemos
    pasarle tambien un flujo de salida
    */
```

```
    */
```

```
    // creacion de un flujo de salida
```

```
    FileOutputStream flujo = new FileOutputStream(new File(dir,nombre + ".xml"));
```

```
// escritura del xml
xstream.toXML(this, flujo);
flujo.close();
}
```

En primer lugar se instancia un objeto de tipo Xstream que nos servirá para hacer toda la conversión, en este indicamos que los objetos de tipo “Departamentos” tendrán el alias de “departamento” siendo esta la etiqueta principal, a continuación al método toXML le pasamos la instancia que queremos convertir (en este caso this) y el flujo de salida hacia el fichero que deseamos.

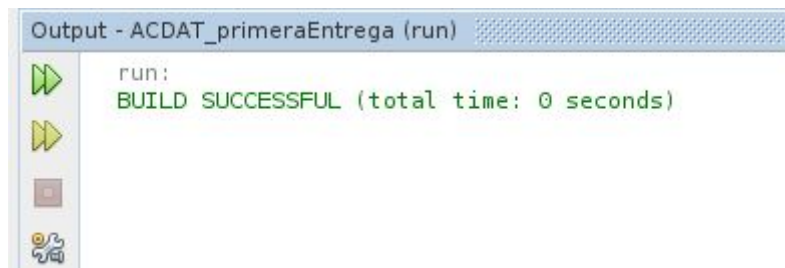
Para el caso de varios departamentos no se ha implementado dentro de “Departamentos” pero en el resultado se muestra un ejemplo de la implementación para este caso.

d) Resultado

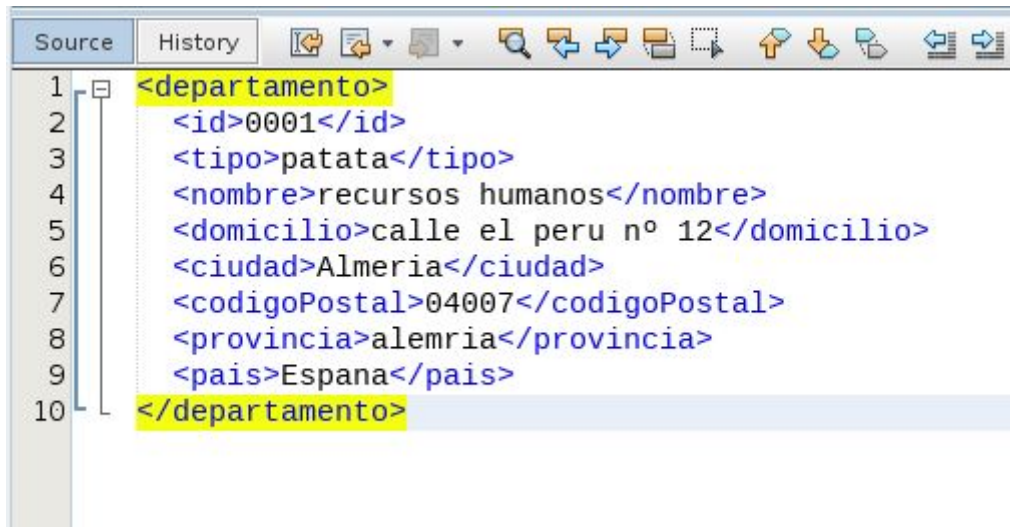
Para el caso de un solo departamento:

```
public static void main(String[] args) {
    // creacion de un nuevo Departamento
    Departamentos departamento;
    try{
        // captura del valor del departamento
        departamento = Departamentos.leerFichero(directorio);
        // escritura del fichero
        departamento.saveXMLXstream("departamentoXstream", "src/departamentos");
        // control de las posibles excepciones
    }catch(ClassNotFoundException cl){
        System.out.println("clase no encontrada");
        cl.printStackTrace();
    }catch(IOException io){
        System.out.println("error en la escritura");
        io.printStackTrace();
    }
}
```

Sencillamente se captura el valor del fichero con el objeto guardado y después es fichero se guarda a sí mismo con el valor indicado.



- *ejecución exitosa*



- *xml resultante*

Para el caso de varios departamentos con un ArrayList:

```
public class XstreamPrueba {
    public static void main(String[] args) {
        Departamentos dep1 = new Departamentos("111", "patata", "hawaii", "c/ el salvador",
            "bucarest", "22222222", "almeria", "espanna");
        Departamentos dep2 = new Departamentos("111", "patata", "hawaii", "c/ el salvador",
            "bucarest", "22222222", "almeria", "espanna");
        Departamentos dep3 = new Departamentos("111", "patata", "hawaii", "c/ el salvador",
            "bucarest", "22222222", "almeria", "espanna");
        ArrayList<Departamentos> deps = new ArrayList<Departamentos>();

        deps.add(dep1);
        deps.add(dep2);
        deps.add(dep3);

        try{
            XStream xstream = new XStream();
            xstream.alias("departamentos", List.class);
            //xstream.aliasField("departamentos", ArrayList.class, "list");
            xstream.alias("departamento", Departamentos.class);
            xstream.toXML(deps, new FileOutputStream(new
            File("src/departamentos/departamentosXstreamList.xml"))));
        }catch(Exception ex){
        }
    }
}
```

Lo único a tener en cuenta a la hora de utilizar Xstream con un arrayList es que si esta esta dentro de otra clase se debe indicar con "impliccitcollection" en un caso como este que se usa directamente un ArrayList debemos tener en cuenta que para que el alias funcione no tenemos que usar "arrayList.class" sino "List.class" al ser esta la interfaz que usa.



The image shows a screenshot of an IDE with two windows. The top window, titled 'Source', displays an XML document with the following structure:

```

1  <departamentos>
2    <departamento>
3      <id>111</id>
4      <tipo>patata</tipo>
5      <nombre>hawaii</nombre>
6      <domicilio>c/ el salvador</domicilio>
7      <ciudad>bucarest</ciudad>
8      <codigoPostal>22222222</codigoPostal>
9      <provincia>almeria</provincia>
10     <pais>espanna</pais>
11   </departamento>
12   <departamento>
13     <id>111</id>
14     <tipo>patata</tipo>
15     <nombre>hawaii</nombre>
16     <domicilio>c/ el salvador</domicilio>
17     <ciudad>bucarest</ciudad>
18     <codigoPostal>22222222</codigoPostal>
19     <provincia>almeria</provincia>
20     <pais>espanna</pais>
21   </departamento>
22   <departamento>
23     <id>111</id>
24     <tipo>patata</tipo>
25     <nombre>hawaii</nombre>
26     <domicilio>c/ el salvador</domicilio>
27     <ciudad>bucarest</ciudad>
28     <codigoPostal>22222222</codigoPostal>
29     <provincia>almeria</provincia>
30     <pais>espanna</pais>
31   </departamento>
32 </departamentos>

```

The bottom window, titled 'Output - ACDAT_primerEntrega (run)', shows the following output:

```

run:
BUILD SUCCESSFUL (total time: 0 seconds)

```

4. Transformaciones XML.

a) Explicación.

Un Documento XML carece de formato ya que se trata de un lenguaje que de forma nativa nos permite estructurar la información a nuestro gusto pero no indica qué hacer con esta a no ser que nosotros mismo programemos esta interpretación. Para los casos que

necesitemos mostrar esta información creando otro tipo de archivo tenemos lo que llaman las transformaciones, estas se hacen a través de un fichero XSL(eXtensible Stylesheet Language) en este fichero se indica cómo debe mostrarse la información contenida en el XML accediendo a su estructura y navegando a través de ella indicando con etiquetas que valor usar.

Para realizar terminar el proceso hay que finalizar con una transformación del XML a través del fichero xsl vinculado a este, un ejemplo sencillo sería el de un navegador que a través de esta plantilla puede crear un “html” con la información del XML, esta implementación en Java es a través del método transform.

b) Clases y métodos para la transformación

Clases.

- Source: A través de esta clase se guarda en memoria la estructura de los documentos a transformar, un objeto de esta clase puede ser creado mediante un objeto ya existente como un “Documento” DOM o bien a través de un flujo de entrada con “StreamSource” que lee un xml, xsl u otro tipo de archivo.
- Result: esta clase define el resultado de la transformación pudiendo esta ser guardada en memoria o a través de StreamResult salir a otro documento en un flujo de salida.
- TransformerFactory: esta clase sirve para crear instancias de Transformer.
- Transformer: esta es la clase que realiza la transformación final a través de un source de entrada un result de salida, para usar una transformación xslt se debe indicar en el constructor.

Métodos.

- newTransformer: este método de TransformerFactory sirve para crear nuevas instancias de Transformer, dentro de este se puede indicar un source para el formateo.
- transform: este método de Transformer realiza el proceso de la transformación pasándole por parámetro un source que será transformado y un resultado que indica la cómo será la salida de la transformación

c) Ejemplo Departamentos:

Para la transformación se ha creado una pequeña clase donde se hace el proceso de transformación.

```
public class TranformarXsl {
    /**
     * metodo principal crear html
     * @param args
     */
    public static void main(String[] args) {
        File dirXsl = new File("src/departamentos/departamentosList.xsl");
        File dirXml = new File("src/departamentos/departamentosXstreamList.xml");
        File dirSalida = new File("src/departamentos/departamentosList.html");
        try{
            // creacion de los elementos source
            Source sourceXsl = new StreamSource(dirXsl);
            Source sourceXml = new StreamSource(dirXml);

            // creacion del transform factory
            TransformerFactory fac = TransformerFactory.newInstance();
            // creacion del transformer
            Transformer trans = fac.newTransformer(sourceXsl);
            trans.transform(sourceXml, new StreamResult(dirSalida));
        }catch(TransformerException tr){
        }catch(TransformerFactoryConfigurationError tra){
        }
    }
}
```

Para este ejemplo se han creado tres objetos File que guardan la información de los directorios, el primero al archivo xsl, el segundo al xml y el tercero al archivo de salida además se crean 3 sources, el primero para el xsl de la transformación y el segundo para el XML a transformar, estos dos archivos funcionan como información de entrada, a continuación se crea el Transformer con su correspondiente TransformerFactory, en la creación del Transformer se pasa por parámetro el source del xsl indicando al transformer como debe ser la transformación, finalmente se llama al método transform con el source del XML a transformar y se crea al mismo tiempo un ResultStream para la salida en forma de HTML.

el fichero xsl:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output method="html"/>

    <!-- TODO customize transformation rules
    syntax recommendation http://www.w3.org/TR/xslt
```

```

-->
<xsl:template match="departamentos">
<html>
<head>
    <title>departamentosList.xml</title>
</head>
<body>
    <table>
    <tr>
    <th>id</th>
    <th>tipo</th>
    <th>nombre</th>
    <th>domicilio</th>
    <th>ciudad</th>
    <th>codigoPostal</th>
    <th>provincia</th>
    <th>pais</th>
    </tr>
    <xsl:for-each select="departamento">
    <tr>
        <td><xsl:value-of select="id"/></td>
        <td><xsl:value-of select="tipo"/></td>
        <td><xsl:value-of select="nombre"/></td>
        <td><xsl:value-of select="domicilio"/></td>
        <td><xsl:value-of select="ciudad"/></td>
        <td><xsl:value-of select="codigoPostal"/></td>
        <td><xsl:value-of select="provincia"/></td>
        <td><xsl:value-of select="pais"/></td>

    </tr>
    </xsl:for-each>
    </table>
</body>
</html>
</xsl:template>

```

</xsl:stylesheet>

En este fichero se estructura la información en forma de tabla usando una etiqueta “for-each” para recorrer las etiquetas de tipo departamento y con “value-of” mostrando la información de sus sub etiquetas.


```
Output - ACDAT_primerEntrega (run)
run:
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
Source History
1 <html>
2 <head>
3 <META http-equiv="Content-Type" content="text/html; charset=UTF-8">
4 <title>departamentosList.xml</title>
5 </head>
6 <body>
7 <table>
8 <tr>
9 <th>id</th><th>tipo</th><th>nombre</th><th>domicilio</th><th>ciudad</th><th>codigoPostal</th><th>provinc
10 </tr>
11 <tr>
12 <td>111</td><td>patata</td><td>hawaii</td><td>c/ el salvador</td><td>bucarest</td><td>22222222</td><td>a
13 </tr>
14 <tr>
15 <td>111</td><td>patata</td><td>hawaii</td><td>c/ el salvador</td><td>bucarest</td><td>22222222</td><td>a
16 </tr>
17 <tr>
18 <td>111</td><td>patata</td><td>hawaii</td><td>c/ el salvador</td><td>bucarest</td><td>22222222</td><td>a
19 </tr>
20 </table>
21 </body>
22 </html>
```

5. Cuestiones sobre el tema.

a) ¿Sigue Xstream la filosofía DOM?

No la sigue por varios motivos, el primero es que Xstream consiste en la transformación de objetos a XML y viceversa mediante la serialización, esto quiere decir que la información obligatoriamente debe almacenarse mediante objetos de forma inevitable en algún momento, en caso de DOM lo que hacemos es crear una estructura en memoria para representar el xml y así crearlo o leerlo pero el origen de la información nos vemos obligados a que sea de un objeto hasta el punto que para guardar la información de este objeto debemos crear nosotros mismos su esquema. La gran similitud estaría en que en ambos casos es necesario en primera instancia guardar la información en memoria.

b) ¿Qué utilidad crees que tiene la transformación a HTML?

Su gran ventaja es que es una forma sencilla y rápida de presentar automáticamente la información con un proceso totalmente reutilizable, permitiéndonos rescatar la información y mostrarla.

c) ¿es posible transformar a otros formatos?

Si, dentro del fichero xsl indicamos que tipos de salida queremos y cómo de ser esta entre otras posibilidades está el texto plano y el Pdf, pero este nos puede servir dependiendo de la interpretación que le demos.

Repositorio con el Código:

https://github.com/likendero/accesoDatos/tree/master/ACDAT_primeraEntrega/src/departamentos

Bibliografía:

Fuente	Tema
http://www.blog.teraswap.com/xstream-conversores/	Xstream
http://x-stream.github.io/tutorial.html	Xstream
http://www.jtech.ua.es/j2ee/publico/lja-2012-13/sesion05-apuntes.html	Serialización
https://en.wikipedia.org/wiki/Object_(computer_science)	objetos
https://docs.oracle.com/javase/8/docs/api/javax/xml/transform/Transformer.html	Transformaciones en Java
https://www.w3schools.com/xml/xsl_intro.asp	XSLT