# TEMA 1

# ACCESO A FICHEIROS, SERIALIZACIÓN DE OBXECTOS, FICHEIROS XML E JSON.

# 1. OBXECTOS SERIALIZABLES

Se temos un obxecto de tipo empregado con varios atributos (o nome, a dirección, o salario, o departamento, o oficio, etc.) e queremos gardalo nun ficheiro, teriamos que gardar cada atributo que forma parte do obxecto, pero de forma separada. Isto vólvese engorroso se temos grandes cantidade de obxectos. Por isto, Java permítenos gardar obxectos en ficheiros binarios, para o que fai falta que o obxecto implemente a interface Serializable. Polo tanto, un obxecto serializable é aquel que se pode almacenar nun ficheiro binario.

A interface Serializable non ten métodos nin atributos e serve só para indicar que os obxectos da clase son serializables. As clases que se usan durante a serialización e deserialización deben implementar unha serie de métodos cos que poderemos gardar e ler obxectos en ficheiros binarios.

Para ler e escribir obxectos serializables a un stream precisamos das seguintes clases:

- ObjectInputStream
  - Dispón do método readObject()
- ObjectOutputStream
  - Dispón do método writeObject()
- FileInputStream
  - Fai referencia ao ficheiro mediante a ruta no sistema de arquivos
- FileOutputStream
  - Fai referencia ao ficheiro mediante a ruta no sistema de arquivos

Os métodos máis importantes a utilizar son:

- void readObject (java.io.ObjectInputStream stream) throws IOException, ClassNotFoundException
  - Para ler un obxecto.
- void writeObject (ObjectOutputStream stream) throws IOException
  - Para escribir un obxecto.

A serialización de obxectos de Java permite tomar calquera obxecto que implemente a interface Serializable e convertilo nunha secuencia de bits, que pode ser posteriormente restaurada para xerar unha instancia exacta do obxecto orixinal.

Ademais, temos que ter en conta unha serie de excepcións que poder aparecer no caso de erros:

- FileNotFountException
  - Cando a ruta ao ficheiro non existe ou hai problemas cos permisos.
- IOException
  - Cando ocurre un problema co fluxo de datos ao ler ou escribir.
- ClassNotFoundException
  - Ao ler os ficheiros non existe na aplicación a clase correspondente para deserializar os obxectos.

Por último, débese pecha o fluxo de datos, primeiro pechando o stream e logo o ficheiro.

```
//Preparar el flujo de salida para escribir
FileOutputStream ficheroSalida = new FileOutputStream("alumnos.dat");
ObjectOutputStream salida = new ObjectOutputStream(ficheroSalida);
//Guardar objetos
salida.writeObject(a1);
//Cerrar fichero
salida.close();
ficheroSalida.close();
```

```java
//Preparar el flujo de entrada de los datos
FileInputStream ficheroEntrada = new FileInputStream("alumnos.dat");
ObjectInputStream entrada = new ObjectInputStream(ficheroEntrada);
//Leer objetos
Alumno alumno = null;
while (ficheroEntrada.available() > 0) {
    alumno = (Alumno) entrada.readObject();
    System.out.println(alumno);
}
//Cerrar ficheiro
entrada.close();
ficheroEntrada.close();
```

## 2. ESTRUTURA DUN DOCUMENTO XML

Un documento XML está composto desde o punto de vista de programación por nodos que poden conter outros nodos. Todo é un nodo:

- O par formado pola etiqueta de apertura (“<etiqueta>”) e pola de cerre (“</etiqueta>”), xunto con todo o seu contido (elementos, atributos e texto do seu interior) é un nodo chamado **elemento** (**Element** desde o punto de vista de programación). Un elemento pode conter outros elementos, é dicir, pode conter no seu interior subetiquetas, de forma aniñada.
- Un **atributo** é un nodo especial chamado atributo (**Attr** desde o punto de vista de programación), que só pode estar dentro dun elemento (concretamente dentro da etiqueta de apertura).
- O **texto** é un nodo especial chamado texto (**Text**), que só pode estar dentro dunha etiqueta.
- Un **comentario** é un nodo especial chamado comentario (**Comment**), que pode estar en calquera lugar do documento XML.
- E por último, un **documento** é un nodo que contén unha xerarquía de nodos no seu interior. Está formado opcionalmente por unha declaración, opcionalmente por un ou varios comentarios e obrigatoriamente por un único elemento (elemento raíz).

# 3. TRABALLO CON FICHEIROS XML

Quen establece as bases do XML? Pois o W3C ou World Wide Web Consortium é a entidade que establece as bases do XML. Esta entidade, ademais de describir como é o XML internamente, define un montón de tecnoloxías estándar adicionais para verificar, convertir e manipular documentos XML. Nós non imos explorar tódalas tecnoloxías de XML aquí (son moitas), só imos usar dúas delas, aquelas que nos van permitir manexar de forma simple un documento XML: DOM e SAX.

Para ler os ficheiros XML e acceder ó seu contido e estrutura, utilízase un procesador de XML ou parser. O procesador le os documentos e proporciona acceso ó seu contido e estrutura. Algúns dos procesadores máis empregados son: DOM: Modelo de Obxectos de Documento e SAX: API Simple para XML. Son independentes da linguaxe de programación e existen versións particulares para Java, VisualBasic, C, etc. Utilizan dous enfoques moi diferentes:

**DOM**: un procesador XML que utilice este plantexamento almacena toda a estrutura do documento en memoria en forma de árbore con nodos pai, nodos fillo e nodos finais (que son aqueles que non teñen descendentes). Unha vez creada a árbore, vanse percorrendo os diferentes nodos e analízase a que tipo particular pertencen. Ten a súa orixe no W3C. Este tipo de procesamento necesita máis recursos de memoria e tempo sobre todo se os ficheiros XML a procesar son bastante grandes e complexos.

**SAX**: un procesador que utilice este plantexamento le un ficheiro XML de forma secuencial e produce unha secuencia de eventos (comezo/fin de documento, comezo/fin dunha etiqueta, etc.) en función dos resultados da lectura. Cada evento invoca a un método definido polo programador. Este tipo de procesamento practicamente non consume memoria, pero por outra parte, impide ter unha visión global do documento polo que navegar.

## 4. PASO DE XML A ÁRBORE DE OBXECTOS DOM

XML DOM permite transformar un documento XML nun modelo de obxectos (de feito DOM significa Document Object Model), accesible comodamente dende a linguaxe de programación. DOM almacena cada elemento, atributo, texto, comentario, etc. Do documento XML nunha estrutura tipo árbore composta por nodos facilmente accesibles, sen perder a xerarquía do documento. A partir de agora, a estrutura DOM que almacena un XML a chamaremos árbore ou xerarquía de obxectos DOM.

En Java, estas e outras funcións están implementadas na libraría JAXP(Java API for XML Processing) e xa van incorporadas na edición estándar de Java (Java SE). En primeiro lugar imos ver como converter un documento XML a unha árbore DOM, e viceversa, para despois ver como manipular dende Java unha árbore DOM.

Para cargar un documento XML temos que facer uso dun procesador de documentos XML (coñecidos xeralmente como parsers) e dun construtor de documentos DOM. As clases de Java que teremos que usar son:

- javax.xml.parsers.DocumentBuilder: será o procesador e transformará un documento XML a DOM, coñéceselle como construtor de documentos.
- javax.xml.parsers.DocumentBuilderFactory: permite crear un construtor de documentos, é unha fábrica de construtores de documentos.
- org.w3c.dom.Document: unha instancia desta clase é un documento XML pero almacenado en memoria seguindo o modelo DOM. Cando o parser procesa un documento XML creará unha instancia desta clase co contido do documento XML.

Agora ben, isto como se usa? A partir do seguinte exemplo, pódese entender (non esquezas importar as librerías):

```
String rutaArchivoXML = "Ruta ao ficheiro";
Document doc = null;
try {
    // 1º Creamos una nueva instancia de un fabrica de constructores de documentos.
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    // 2º A partir de la instancia anterior, fabricamos un constructor de documentos, que procesará el XML.
    DocumentBuilder db = dbf.newDocumentBuilder();
    // 3º Procesamos el documento (almacenado en un archivo) y lo convertimos en un árbol DOM.
    doc=db.parse(rutaArchivoXML);
} catch (IOException | ParserConfigurationException | SAXException ex) {
    System.out.println("¡Error! Non se pudo cargar o documento XML.");
}
```

Se en lugar de cargar na memoria un documento XML do sistema de arquivos queremos crear na memoria un documento DOM baleiro, con etiquetaRaiz como etiqueta raíz do documento:

```
Document documento = null;
try {
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    DocumentBuilder db = dbf.newDocumentBuilder();
    DOMImplementation implementation = db.getDOMImplementation();
    documento = implementation.createDocument(null, "etiquetaRaiz", null);
    documento.setXmlVersion("1.0"); //asignamos a version do noso XML
} catch (ParserConfigurationException ex) {
    System.out.println("¡Error! Non se pudo parsear o XML.");
}
```

## 5. PASO DE ÁRBORE DE OBXECTOS DOM a XML

E como paso a xerarquía ou árbore de obxectos DOM a XML? En Java isto é un pouco máis complicado cá operación inversa, e require o uso dun montón de clases do paquete java.xml.transform, pois a idea é transformar a árbore DOM nun arquivo de texto que contén o documento XML.

As clases que teremos que usar son:

- javax.xml.transform.TransformerFactory. Fábrica de transformadores, permite crear un nuevo transformador que convertirá el árbol DOM a XML.
- javax.xml.transform.Transformer. Transformador que permite pasar un árbol DOM a XML.
- javax.xml.transform.TransformerException. Excepción lanzada cuando se produce un fallo en la transformación.
- javax.xml.transform.OutputKeys. Clase que contiene opciones de salida para el transformador. Se suele usar para indicar la codificación de salida (generalmente UTF-8) del documento XML generado.
- javax.xml.transform.dom.DOMSource. Clase que actuará de intermediaria entre el árbol DOM y el transformador, permitiendo al transformador acceder a la información del árbol DOM.
- javax.xml.transform.stream.StreamResult. Clase que actuará de intermediaria entre el transformador y el archivo o String donde se almacenará el documento XML generado.
- java.io.File. Clase que, como posiblemente sabrás, permite leer y escribir en un archivo almacenado en disco. El archivo será obviamente el documento XML que vamos a escribir en el disco.

O seguinte é un exemplo de como realizar o proceso de transformación de árbore DOM a XML:

```
try {
    // 1º Creamos una instancia de la clase File para acceder al archivo donde guardaremos el XML.
    File f = new File(rutaArchivoXML);
    //2º Creamos una nueva instancia del transformador a través de la fábrica de transformadores.
    Transformer transformer = TransformerFactory.newInstance().newTransformer();
    //3º Establecemos algunas opciones de salida, como por ejemplo, la codificación de salida.
    transformer.setOutputProperty(OutputKeys.ENCODING, "UTF-8");
    //4º Creamos el StreamResult, intermediria entre el transformador y el archivo de destino.
    StreamResult result = new StreamResult(f);
    //5º Creamos el DOMSource, intermediaria entre el transformador y el árbol DOM.
    DOMSource source = new DOMSource(doc);
    //6º Realizamos la transformación.
    transformer.transform(source, result);
} catch (TransformerException ex) {
    System.out.println("¡Error! No se ha podido llevar a cabo la transformación.");
}
```

# 6. MANIPULACIÓN DE DOCUMENTOS XML

Agora xa sabes cargar un documento XML a DOM y de DOM a XML, pero, como se modifica a árbore DOM? Como xa se dixo antes, unha árbore DOM é unha estrutura en árbore, xerárquica como cabe esperar, formada por nodos de diferentes tipos. O funcionamento do modelo de obxectos DOM é establecido polo organismo W3C, o cal ten unha gran vantaxe, o modelo é practicamente o mesmo en todas as linguaxes de programación.

En Java, practicamente todas as clases que vas necesitar para manipular unha árbore DOM están no paquete org.w3c.dom. Para realizar probas e facer un uso moi intenso de DOM pódese facer unha importación de todas as clases deste paquete ("import org.w3c.dom.*;").

Tras converter un documento XML a DOM o que obtemos é unha instancia da clase org.w3c.dom.Document. Esta instancia será o nodo principal que conterá no seu interior toda a xerarquía do documento XML. Dentro dun documento de árbore DOM podemos encontrar os seguintes tipos de clases:

- org.w3c.dom.Node (Nodo). Todos os obxectos contidos na árbore DOM son nodos. A clase Document é tamén un tipo de nodo, considerado o nodo principal.
- org.w3c.dom.Element (Elemento). Corresponde con calquera par de etiquetas ("") e todo o seu contido (atributos, texto, subetiquetas, etc.).
- org.w3c.dom.Attr (Atributo). Corresponde con calquera atributo.
- org.w3c.dom.Comment (Comentario). Corresponde cun comentario.
- org.w3c.dom.Text (Texto). Corresponde co texto que encontramos dentro de dúas etiquetas.

Estes conceptos xa che resultarán familiares. Estas clases terán diferentes métodos para acceder e manipular a información da árbore DOM. A continuación imos a ver as operacións máis importantes sobre unha árbore DOM. En todos os exemplos, "doc" corresponde con una instancia xa creada da clase Document.

**Obter o elemento raíz do documento**.

Como xa sabes, os documentos XML deben ter obrigatoriamente un único elemento ("<pedido></pedido>" por exemplo), considerado elemento raíz, dentro do cal está o resto da información estruturada de forma xerárquica. Para obter dito elemento e poder manipulalo, podemos usar o método getDocumentElement.

```
Element raiz=doc.getDocumentElement();
```

**Buscar un elemento en toda a xerarquía do documento**

Para realizar esta operación pódese usar o método getElementsByTagName disponible tanto na clase Document como na clase Element. Dita operación busca un elemento polo nome da etiqueta e retorna una lista de nodos (NodeList) que cumpran coa condición. Se se usa na clase Element, so buscará entre as subetiquetas (subelementos) de dita clase (non en todo o documento).

```
NodeList nl=doc.getElementsByTagName("cliente");
Element cliente;
if (nl.getLength()==1) cliente=(Element)nl.item(0);
```

O método getLength() da clase NodeList, permite obter o número de elementos (lonxitude da lista) atopados co nome de etiqueta coincidinte. O método item permite acceder a cada un dos elementos atopados, e se lle pasa por argumento o índice do elemento a obter (empezando polo cero e acabando pola lonxitude menos uno). Hai que fixarse que é necesario facer unha conversión de tipos despois de invocar o método item. Isto é porque a clase NodeList almacena un listado de nodos (Node), sen diferenciar el tipo.

**Obter a lista de fillos dun elemento e procesala**

Se trata de obter unha lista cos nodos fillo dun elemento calquera. Estes poden ser un subelemento (subetiqueta) ou texto. Para sacar a lista de nodos, pódese usar o método getChildNodes:

```
nl = doc.getDocumentElement().getChildNodes();
for (int i = 0; i < nl.getLength(); i++) {
    Node n = nl.item(i);
    switch (n.getNodeType()) {
        case Node.ELEMENT_NODE:
            Element e = (Element) n;
            System.out.println("Etiqueta:" + e.getTagName());
            mostraContidoElemento(e);
            break;
        case Node.TEXT_NODE:
            Text t = (Text) n;
            System.out.println("Texto:" + t.getWholeText());
            break;
    }
}
```

No exemplo anterior, úsanse varios métodos. O método "getNodeType()" da clase Node permite saber de que tipo de nodo se trata, xeralmente texto (Node.TEXT_NODE) ou un subelemento (Node.ELEMENT_NODE). Desta forma, poderemos facer a conversión de tipos adecuada e xestionar cada elemento como corresponda. Tamén se usa o método "getTagName" aplicado a un elemento, o cal permitirá obter o nome da etiqueta; e o método "getWholeText" aplicado a un nodo de tipo texto (Text), que permite obter o texto contido no nodo.

**Engadir un novo elemento fillo a outro elemento**

Xa vimos como inspeccionar o documento XML, pero non como engadir cousas a dito documento. Para engadir un subelemento ou un texto a unha árbore DOM, primeiro temos que crear os nodos correspondentes e despois inserilos na posición que queiramos. Para crear un novo par de etiquetas ou elemento (Element) e un novo nodo texto (Text), o podemos facer da seguinte forma:

```
Element dirTag = doc.createElement("Direccion_entrega");
Text dirTxt =doc.createTextNode("C/Perdida S/N");
```

Agora que os creamos, pero aínda non os engadimos ao documento, temo que usar o método appendChild que engadirá o nodo (sexa do tipo que sexa) ao final da lista de fillos do elemento correspondente:

```
dirTag.appendChild(dirTxt);
doc.getDocumentElement().appendChild(dirTag);
```

No exemplo anterior, o texto engádese como fillo da etiqueta "Direccion_entrega", e á súa vez, a etiqueta "Direccion_entrega" se engade como fillo, ao final de todo, da etiqueta ou elemento raíz do documento. Aparte do método appendChild, que sempre engadirá ao final, pódense utilizar os seguintes métodos para dentro dunha árbore DOM (todos se usan sobre a clase Element):

- insertBefore (Node novo, Node referencia). Engadirá un nodo novo antes do nodo de referencia.
- replaceChild (Node novo, Node anterior). Substituirá un nodo (anterior) por un novo.

**Eliminar un elemento fillo de outro elemento**

Para eliminar un nodo, hai que acceder ao nodo pai de dito nodo. No nodo pai invócase o método removeChild, ao que se lle pasa a instancia da clase Element co nodo a eliminar (non o nome da etiqueta, se non a instancia), o cal implica que primeiro hai que buscar o nodo a eliminar e despois eliminalo.

```
Element parent=(Element)elementoABorrar.getParentNode();
parent.removeChild(elementoABorrar);
```

Obtense o nodo pai do fillo a través do método getParentNode, para así poder eliminar o nodo correspondente co método removeChild.

Non é obrigatorio invocar ao método "getParentNode" se o nodo pai é coñecido. Por exemplo, se o nodo é un fillo do elemento ou etiqueta raíz, chegaría con poro seguinte:

```
doc.getDocumentElement().removeChild(elementoABorrar);
```

**Cambiar o contido dun elemento cando solo é texto**

O métodos getTextContent e setTextContent, aplicados a un elemento, permiten acceder ao texto contido dentro dun elemento ou etiqueta. Débese ter coidado, porque utilizar "setTextContent" significa eliminar calquera fillo (subelemento, por exemplo) que previamente tivera a etiqueta.

```
Element nuevo = doc.createElement("direccion_recogida");
nuevo.setTextContent("C/Del Medio S/N");
System.out.println(nuevo.getTextContent());
```

**Atributos dun elemento**

Por último, queda xestionar os atributos dun elemento. Calquera elemento pode conter un ou varios atributos. Acceder a eles é sinxelo, so se necesitan tres métodos: setAttribute para establecer ou crear o valor dun atributo, getAttribute para obter o valor dun atributo e removeAttribute para eliminar o valor dun atributo.

```
doc.getDocumentElement().setAttribute("urgente","no");
System.out.println(doc.getDocumentElement().getAttribute("urgente"));
```

# 7. ACCESO A FICHEIROS XML CON SAX

SAX (API Simple para XML) é un conxunto de clases e interfaces que ofrecen unha ferramenta moi útil para o procesamento de documentos XML. Permite analizar os documentos de forma secuencial (é dicir, non carga todo o ficheiro en memoria como fai DOM), isto implica pouco consumo de memoria aínda que os documentos sexan de grande tamaño, en contraposición, impide ter unha visión global do documento que se vai analizar. SAX é máis complexo de programar que DOM, é un API totalmente escrita en Java e incluída dentro do JRE que nos permite crear o noso propio parser de XML.

A lectura dun documento XML produce eventos que ocasiona a chamada a métodos, os eventos son atopar a etiqueta de inicio e fin do documento (startDocument() e endDocument()), a etiqueta de inicio e fin dun elemento (startElement() e endElement()), os caracteres entre etiquetas (characters()), etc.

Os obxectos que posúen métodos que tratarán os eventos serán normalmente implementacións das seguintes interfaces:

- ContentHandler: recibe as notificacións dos eventos que ocorren no documento.
- DTDHandler: recolle eventos relacionados coa DTD
- ErrorHandler: define métodos de tratamentos de erros.
- EntityResolver: os seus métodos chámanse cada vez que se atopa unha referencia a unha entidade.
- DefaultHandler: clase que prove unha implementación por defecto para tódolos seus métodos, o programador definirá os métodos que sexan utilizados polo programa.

## 8. SERIALIZACIÓN DE OBXECTOS A XML

Xa vimos como se poden serializar de forma sinxela obxectos Java a XML e viceversa. Utilizaremos para isto a librería XStream. Para poder utilizala debemos descargar os JAR desde o sitio web: http://xstream.codehaus.org/download.html, para o exemplo descargouse o ficheiro Binary distribution (xstream-distribution-1.4.2-bin.zip) que descomprimimos e buscamos o JAR xstream-1.4.2.jar que está na carpeta lib que é o que usaremos para o exemplo. Tamén necesitamos o ficheiro kxml2-2.3.0.jar que se pode descargar desde o apartado Optional Dependencies. Unha vez que temos os dous ficheiros, os definimos no CLASSPATH.

```
try {
    List<Alumno> listaAlumnos = new ArrayList<>();
    listaAlumnos.add(unAlumno);
    listaAlumnos.add(otroAlumno);

    //Instanciar objeto de XStream
    XStream xstream = new XStream();
    //Configurar XStream
    //Modo NO_REFERENCES para forzar a que ponga todos los elementos
    //aunque no existan referencias
    xstream.setMode(XStream.NO_REFERENCES);
    //Se indican los alias para renombrar las clases a la etiqueta
    xstream.alias("alumno", Alumno.class);
    xstream.alias("alumnos", List.class);
    //Guardamos el objeto en un fichero XML
    xstream.toXML(listaAlumnos, new FileOutputStream("alumnosListaMal.xml"));
} catch (FileNotFoundException ex) {
    Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
}
```

Esta parte es para crearlo mal para ver que si se crea de esta forma despues no se puede leer

```
try {
    ListaAlumnos listaAlumnos = new ListaAlumnos();
    listaAlumnos.getLista().add(unAlumno);
    listaAlumnos.getLista().add(otroAlumno);

    //Instanciar objeto de XStream
    XStream xstream = new XStream();
    xstream.setMode(XStream.NO_REFERENCES);
    xstream.alias("alumno", Alumno.class);
    xstream.alias("alumnos", ListaAlumnos.class);
    xstream.toXML(listaAlumnos, new FileOutputStream("alumnos.xml"));
} catch (FileNotFoundException ex) {
    Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
}
```

Esta es la parte correcta

```
XStream xs = new XStream(new DomDriver());
xs.setMode(XStream.NO_REFERENCES);
xs.alias("alumno", Alumno.class);
xs.alias("alumnos", ListaAlumnos.class);
xs.addPermission(AnyTypePermission.ANY);
ListaAlumnos listaLeida = new ListaAlumnos();
try {
    xs.fromXML(new FileInputStream("alumnos.xml"), listaLeida);
    System.out.println("\nLista de alumnos: \n" + listaLeida);
} catch (FileNotFoundException ex) {
    Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
}
```

leerlo (Mostrarlo por pantalla)

# 9. REPRESENTACIÓN DA INFORMACIÓN USANDO O FORMATO JSON

Hoxe en día, o uso do formato JSON está moi estendido e pode considerarse un estándar na maioría das linguaxes de programación. A realidade é que pode interpretarse e utilizarse de forma similar a XML, polo que non é preciso afondar en detalles aquí.

Podemos comezar a traballar con este formato coa breve introdución que atopamos no seguinte ligazón:

- https://www.javatpoint.com/json-tutorial

## 10. ACCESO DESDE JAVA A FICHEIROS JSON

Hai multitude de librería que permiten o uso de objetos JSON en Java. As máis populares e importantes son org.json.simple e javax.json.

Android Studio traballa ben con org.json. Este é un dos motivos de elixila de exemplo. JSONObject está incluído na API de Android. JsonObject está deseñado para o desenvolvemento de Java EE, onde esencialmente está destinado para aplicacións web e de redes, así como outras moitas.

Podemos utilizalo, engadindo ao noso CLASSPATH o JAR da librería (json-simple-1.1.1.jar).

```
//Creación de OBJETO JSON
JSONObject obj = new JSONObject();
//Se le añaden datos en formato clave-valor
obj.put("nombre", pers.getNombre());
obj.put("edad", pers.getEdad());
obj.put("salario", pers.getSalario());

//Para las listas, almacenamos los valores en un JSONArray
JSONArray jArray = new JSONArray();
for (String s : pers.getAficiones()) {
    jArray.add(s);
}
obj.put("aficiones", jArray);

//Se guarda escribiendo en el fichero en la ruta
try {
    FileWriter fichero = new FileWriter(ruta);
    fichero.write(obj.toJSONString());
    fichero.close();
} catch (IOException ex) {
    System.err.println(ex);
}
```

```
Persona persona = new Persona();
JSONParser parser = new JSONParser();
//LER FICHEIRO de la ruta
try (FileReader fichero = new FileReader(ruta)) {
    //Se parsea el contenido a un objeto JSON
    JSONObject obj = (JSONObject) parser.parse(fichero);

    //Se recuperan los valores del objeto JSON y se castean
    persona.setNombre((String)obj.get("nombre"));
    persona.setSalario((Double)obj.get("salario"));
    persona.setEdad((Long)obj.get("edad"));

    //Los arrays y listas se recuperan a un JSONArray
    JSONArray jarray = (JSONArray) obj.get("aficiones");
    for (Object o : jarray) {
        persona.getAficiones().add((String)o);
    }

} catch (FileNotFoundException ex) {
    System.err.println(ex);
} catch (IOException | ParseException ex) {
    System.err.println(ex);
}
```

## 11. SERIALIZACIÓN DE OBXECTOS A JSON

Unha das librerías máis empregadas para manexar JSON en Java e Gson, ferramenta desenvolvida por Google e integrada nas versións máis recentes de Android. É un proxecto aberto que ten repositorio en github.

A principal vantaxe do uso desta librería é posibilidade de realizar a conversión directamente de obxectos a un texto (String) que estea en formato JSON sen a necesidade de realizar manualmente as conversións.

Como exemplo, teremos dúas clases (Alumno e Modulo) cos seus atributos, constructores, getters, setters e toString.

```
public class Alumno {

    private String dni;
    private String nombre;
    private String apellidos;
    private Date nacimiento;
    private List<Modulo> modulos;
```

```
public class Modulo {

    private String nombre;
    private String codigo;
```

Necesitaremos incluír no classpath o jar correspondente, que podemos atopar para descargar no repositorio de Maven. Únicamente hai que instanciar un obxecto da clase Gson, facendo o import correspondente do paquete com.google.gson e utilizar os métodos toJson e fromJson.

```
Alumno alumno = new Alumno();
alumno.setNombre("Pedro");
alumno.setApellidos("Pérez López");
alumno.setDni("1A");
alumno.setNacimiento(new GregorianCalendar(1998, Calendar.FEBRUARY, 28).getTime());

Modulo ad = new Modulo("AD", "MP01010");
Modulo di = new Modulo("DI", "MP12121");
alumno.getModulos().add(ad);
alumno.getModulos().add(di);

Gson gson = new Gson();

String jsonAlumno = gson.toJson(alumno);

System.out.println(jsonAlumno);

Alumno alumnoRecuperado = gson.fromJson(jsonAlumno, Alumno.class);

System.out.println(alumnoRecuperado);
```

Para gardar e ler dun ficheiro, faríamos como con json-simple e simplemente utilizaríamos un FileWriter ou FileReader, xa que o ficheiro seguiría sendo de texto plano, pero con extensión *.json*.

```
try (FileWriter file = new FileWriter("alumno.json")) {
        file.write(jsonAlumno);
}catch (IOException ex) {
    Logger.getLogger(EjemploGson.class.getName()).log(Level
}
```

# 12. BIBLIOGRAFÍA


- "Acceso a Datos", Alicia Ramos Martín, Mª Jesús Ramos Martín, Garceta editorial, segunda edición, 2016
- Páxinas web referenciadas ao longo dos apuntamentos.
- Recursos da aula virtual