Desarrollo de aplicaciones Android

Sesión 8

Qué vamos a ver?

- Formatos de tramas de respuesta
  - XML
  - JSON

Formato de tramas

- XML
  - Metalenguaje. Nos permite definir nuestra propia estructura de comunicación
  - Fácil de entender, pero para ciertas situaciones, puede resultar engorroso
  - Muy utilizado, gran cantidad de herramientas y librerias para su uso en Java
- JSON
  - Acrónimo de JavaScript Object Notation
  - Es un formato ligero de intercambio de datos (más liviano que XML)
  - Muy utilizado por sitios web 2.0 para intercambio de datos (Facebook, Twitter, etc)

XML

- Métodos disponibles en Android
  - Sax
  - Sax simplificado
  - DOM
  - PullParser (recomendado)

Método SAX

- Acceso secuencial al documento XML, basado en eventos
- Beneficios
  - Memoria utilizada proporcional a la profundidad de documento
  - Como es secuencial, no se tiene reservada memoria para todo el documento
- Contras
  - Existen validaciones que necesitan tener todo el documento cargado en memoria
  - No se podría trabajar con XSLT y XPath

Pasos

- Crear clase `Handler`. Esta tiene que heredar de `DefaultHandler`
- Implementar los siguientes métodos:
  - `characters()`: se lanza cada vez que se encuentra un fragmento de texto.
  - `startDocument()`: se lanza al iniciar el parseo del documento
  - `endDocument()`: se lanza al finalizar de parsear el documento
  - `startElement()`: se lanza cada vez que se encuentra un nuevo tag de apertura.
  - `endElement()`: se lanza cada vez que se encuentra un tag de cierre
- Crear clase que haga uso del parser creado

Archivo XML

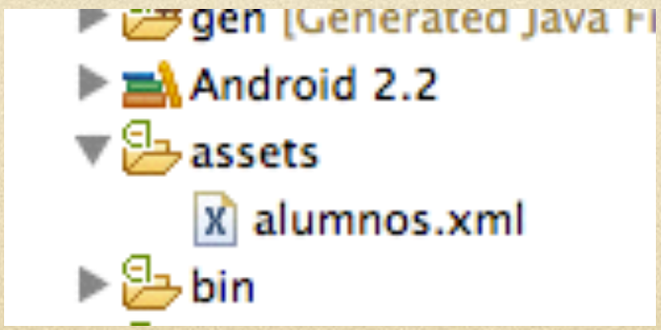
```
<alumnos>
    <alumno id="1" edad="27">Juan Zapata</alumno>
    <alumno id="2" edad="24">Julia Gamarra</alumno>
</alumnos>
```

```java
public class HandlerSAX extends DefaultHandler {

    List<AlumnoBean> alumnos ;
    AlumnoBean alumno;

    @Override
    public void startDocument() throws SAXException {
        Log.i("sesion2", "Se inicio parseo del documento");
        alumnos = new ArrayList<AlumnoBean>();
    }

    @Override
    public void startElement(String uri, String localName, String qName,
            Attributes attributes) throws SAXException {
        if (localName.equals("alumno")){
            alumno = new AlumnoBean();
            alumno.setId(attributes.getValue("","id"));
            alumno.setEdad(attributes.getValue("","edad"));
        }
    }

    @Override
    public void characters(char[] ch, int start, int length)
            throws SAXException {
        if (alumno != null) {
            alumno.setNombre(new String(ch));
        }
    }

    @Override
    public void endElement(String uri, String localName, String qName)
            throws SAXException {
        if (localName.equals("alumno")){
            alumnos.add(alumno);
            alumno = null;
        }
    }
```

- El handler se utiliza así

```
public List<AlumnoBean> dameAlumnos(Context context){
    List<AlumnoBean> lista = null;

    SAXParserFactory factory = SAXParserFactory.newInstance();

    try{
        AssetManager assetManager = context.getAssets();
        InputStream is = assetManager.open("alumnos.xml");

        SAXParser parser = factory.newSAXParser();
        HandlerSAX handler = new HandlerSAX();
        parser.parse(is, handler);
        lista = handler.getListadoAlumnos();
    }
    catch (Exception e){
        throw new RuntimeException(e);
    }

    return lista;
}
```

- El archivo XML debe estar copiado en la carpeta assets:



Método SAX simplificado

- No tenemos que definir una clase independiente para el handler
- Problemas con los tags

```xml
<alumnos>
    <alumno id="1" edad="27" nombre="Juan Zapata">Juan Zapata</alumno>
    <alumno id="2" edad="24" nombre="Julia Gamarra">Julia Gamarra</alumno>
</alumnos>
```

```java
public List<AlumnoBean> dameAlumnosSAXSimplificado(Context context){
    final List<AlumnoBean> lista = new ArrayList<AlumnoBean>();
    try{
        AssetManager assetManager = context.getAssets();
        InputStream is = assetManager.open("alumnos.xml");

        RootElement root = new RootElement("alumnos");
        Element alumnoItem = root.getChild("alumno");

        alumnoItem.setStartElementListener(new StartElementListener() {
            @Override
            public void start(Attributes attributes) {
                alumno = new AlumnoBean();
                alumno.setId(attributes.getValue("","id"));
                alumno.setEdad(attributes.getValue("","edad"));
                alumno.setNombre(attributes.getValue("","nombre"));
            }
        });
        alumnoItem.setEndElementListener(new EndElementListener() {
            @Override
            public void end() {
                lista.add(alumno);
            }
        });
        Xml.parse(is, Xml.Encoding.UTF_8, root.getContentHandler());
    }
    catch (Exception e) {
        throw new RuntimeException(e);
    }
    return lista;
}
```

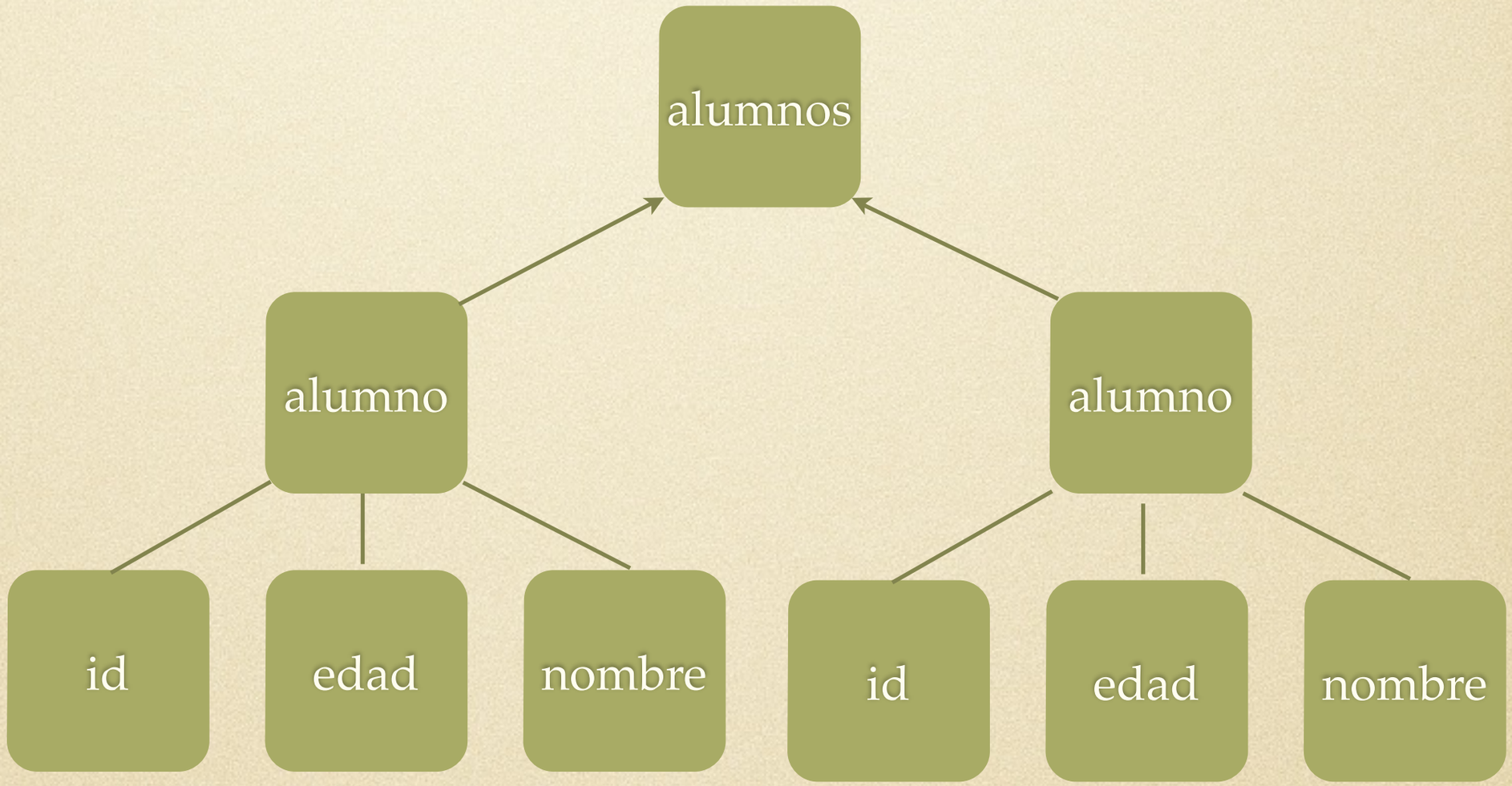
Método DOM

- Pros
  - Mas fácil de codificar (estructura organizada)
  - Se carga todo el documento en memoria, lo que facilita la navegación en este
- Contras
  - Si el documento es grande, puede llenar la memoria
  - Documento complejo, mayor procesamiento

XML

```
<alumnos>
     <alumno id="1" edad="27" nombre="Juan Zapata">Juan Zapata</alumno>
     <alumno id="2" edad="24" nombre="Julia Gamarra">Julia Gamarra</alumno>
</alumnos>
```

DOM



```java
public List<AlumnoBean> dameAlumnosDOM(Context context){
    final List<AlumnoBean> lista = new ArrayList<AlumnoBean>();
    //Instanciamos la fábrica para DOM
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    try{
        AssetManager assetManager = context.getAssets();
        InputStream is = assetManager.open("alumnos.xml");

        //Creamos un nuevo parser DOM
        DocumentBuilder builder = factory.newDocumentBuilder();
        //Realizamos la lectura completa del XML
        Document dom = builder.parse(is);
        //Nos posicionamos en el nodo principal del árbol
        org.w3c.dom.Element root = dom.getDocumentElement();
        //Localizamos todos los elementos <alumnos>
        NodeList alumnosList = root.getElementsByTagName("alumno");

        for (int i=0; i<alumnosList.getLength(); i++){
            AlumnoBean bean = new AlumnoBean();
            Node alumnoElement = alumnosList.item(i);
            bean.setId(alumnoElement.getAttributes().getNamedItem("id").getTextContent());
            bean.setEdad(alumnoElement.getAttributes().getNamedItem("edad").getTextContent());
            bean.setNombre(alumnoElement.getAttributes().getNamedItem("nombre").getTextContent());

            lista.add(bean);
        }
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
    return lista;
}
```

Método Pull Parser

- Parecido al método SAX
- Se diferencia en que acá tenemos el control del flujo de parseo

```
final List<AlumnoBean> lista = new ArrayList<AlumnoBean>();
try{
    AssetManager assetManager = context.getAssets();
    InputStream is = assetManager.open("alumnos.xml");
    AlumnoBean alumno = null;

    XmlPullParser parser = Xml.newPullParser();
    parser.setInput(is, "utf-8");
    int evento = parser.getEventType();
    while (evento != XmlPullParser.END_DOCUMENT){
        String etiqueta = null;
        switch(evento){
            case XmlPullParser.START_DOCUMENT:
                Log.i("sesion2", "Se inicio el documento");
            break;
            case XmlPullParser.START_TAG:
                etiqueta = parser.getName();
                if (etiqueta.equals("alumno")){
                    alumno = new AlumnoBean();
                    alumno.setId(parser.getAttributeValue("", "id"));
                    alumno.setNombre(parser.getAttributeValue("", "nombre"));
                    alumno.setEdad(parser.getAttributeValue("", "edad"));
                }
            break;
            case XmlPullParser.END_TAG:
                etiqueta = parser.getName();
                if (etiqueta.equals("alumno")){
                    lista.add(alumno);
                }
            break;
        }
        evento = parser.next();
    }
} catch (Exception e){
    throw new RuntimeException(e);
}
return lista;
```

JSON

- Objeto JSON

```
{
  "alumno": {
    "id": "1",
    "edad": "27",
    "nombre": "Juan Zapata"
  }
}
```

- Arreglo simple

```
{
  "arreglo": [1, 4, 6, 8, 0]
}
```

- Arreglo complejo

```
{
  "alumnos": [
    {
      "id": "1",
      "edad": "27",
      "nombre": "Juan Zapata"
    },
    {
      "id": "22",
      "edad": "24",
      "nombre": "Julia Gamarra"
    }
  ]
}
```

Métodos clave

- Lectura JSON
  - Para el caso de un JSON Array se crea la clase `JSONArray(String json)`
  - Para el caso de un objeto, se crea la clase `JSONObject(String json)`
- Escritura
  - Por medio de `HashMap`
  - Armando en ejecución la estructura (`JSONObject`, `JSONArray`)

```
public List<AlumnoBean> dameAlumnoJSON(Context context){
    final List<AlumnoBean> lista = new ArrayList<AlumnoBean>();

    try{
        AssetManager assetManager = context.getAssets();
        InputStream is = assetManager.open("jsonejem.json");
        String json = convertStreamToString(is);

        JSONObject obj = new JSONObject(json);
        JSONArray arr = obj.getJSONArray("alumnos");
        for (int i=0; i<arr.length(); i++){
            JSONObject jsAlumno = arr.getJSONObject(i);
            AlumnoBean alumno = new AlumnoBean();
            alumno.setId(jsAlumno.getString("id"));
            alumno.setEdad(jsAlumno.getString("edad"));
            alumno.setNombre(jsAlumno.getString("nombre"));
            lista.add(alumno);
        }
    }catch (Exception e){
        throw new RuntimeException(e);
    }
    return lista;
}
```