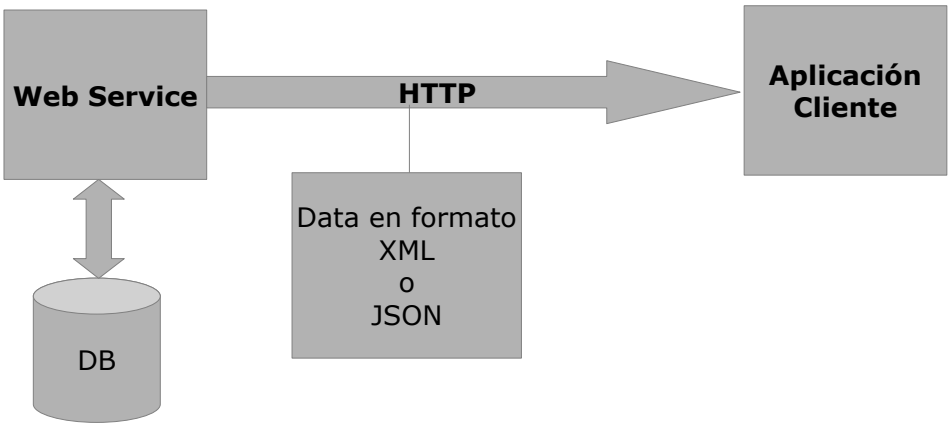
Web Services

Con la biblioteca HttpClient, podemos conectarnos a un servidor HTTP para obtener información (una pagina web, un archivo, etc). Existen formas standard de consumir la información que provee un servidor web, formas de "serializar" la información para que pueda ser interpretada por otro programa, y no por un ser humano, cuando un servidor expone sus datos de esta manera, se lo conoce como **Web Service**.

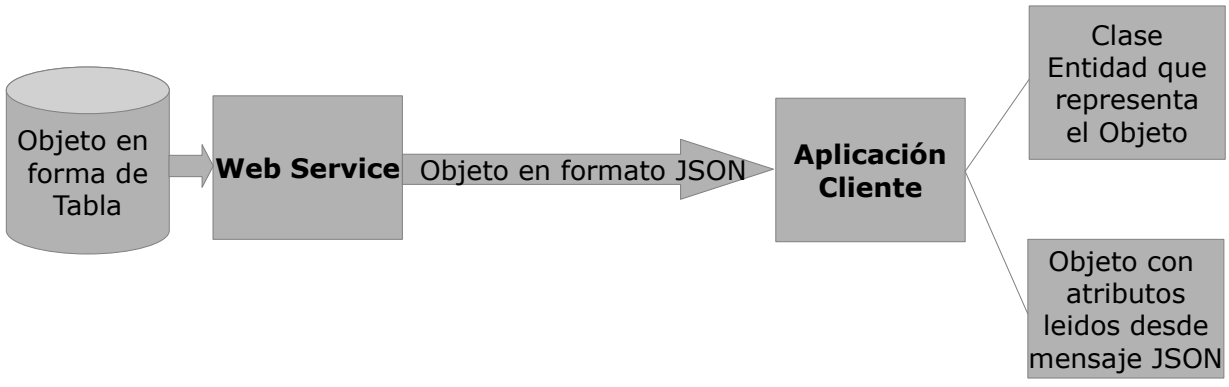
La información que provee un web service se debe estructurar y serializar para poder ser transmitida sobre el protocolo HTTP. Las formas de estructurar y serializar la información son dos:

- XML
- JSON



Este material se centrará en la forma de interpretar información estructurada en el formato JSON.
Android provee una biblioteca que se encarga de, a partir de un String recibido en formato JSON (el mensaje HTTP) , "des-serializar" el mensaje y poder parsear los datos que éste posee.

Generalmente, la estructura de datos de un mensaje JSON corresponde a un objeto, a una entidad que puede ser representada por una clase Java, de modo que la forma de trabajar contra un Web Service, es definir una clase que contenga los atributos que se consumirán del servicio, obtener el String con el mensaje HTTP con formato JSON, parsearlo y cargar los datos obtenidos en los atributos de un objeto de la clase definida.



JSON

JSON es un formato de intercambio de datos basado en el lenguaje de programación JavaScript. Cada String con un mensaje JSON puede ser transformado a un objeto JavaScript.

Formato:

Cada objeto es contenido por llaves, los atributos del objeto se presentan por "nombre:valor" y separados por comas:

```
{
        'nombre': 'Juan',
        'apellido': 'Perez',
        'edad': 19
}
```

En el ejemplo vemos un objeto que podría ser una "Persona" el cual contiene los atributos nombre,apellido y edad.

NOTA: Puede observarse que todos los caracteres de un mensaje JSON son caracteres imprimibles, por lo que pueden ser enviados como texto plano sobre el protocolo HTTP, de esta forma, cuando recibimos este mensaje, podemos almacenarlo en un objeto del tipo String, el cual recibirá la librería que parsee el mensaje.

Formato de Arrays:

Un array de objetos se define mediante corchetes "[]" dentro de los corchetes podemos escribir objetos (que se encierran entre llaves) separados por comas, o otros arrays:

```
[
        {
                'nombre': 'Juan',
                'apellido': 'Perez',
                'edad': 19
        },
        {
                'nombre': 'Pedro',
                'apellido': 'Gonzales',
                'edad': 24
        }
]
```

En el ejemplo vemos un array que contiene dos objetos "Persona", cada objeto se separa por una coma.

Un array también puede ser el valor de un atributo:

```
{
        'nombre': 'Juan',
        'apellido': 'Perez',
        'edad': 19,
        'hermanos' : [
                        {...},
                        {...},
                        {...}
                    ]
}
```

Parseando JSON

Supongamos que tenemos un objeto String el cual posee el mensaje en formato JSON, el cual vendrá de un Web Service.

```
// String con mensaje Json proveniente de un Web Service
String msgJson = "{'nombre': 'Juan','apellido': 'Perez','edad': 19}";

try {
		// Creamos objeto JSONObject pasandole String
		JSONObject objJson = new JSONObject(msgJson);

		// Obtenemos los valores de los atributos del objeto
		// mediante sus nombres
		String nombre = objJson.getString("nombre");
		String apellido = objJson.getString("apellido");
		int edad = objJson.getInt("edad");

		Log.d("json","Objeto parseado:");
		Log.d("json","Nombre:"+nombre);
		Log.d("json","Apellido:"+apellido);
		Log.d("json","Edad:"+edad);


	} catch (JSONException e) {
		e.printStackTrace();
	}
```

En el ejemplo, creamos un objeto del tipo "JSONObject" y en su constructor le pasamos el mensaje Json de tipo String. Un vez creado este objeto, podremos obtener los atributos de la entidad del mensaje llamando a los métodos "getString()" y "getInt()" del objeto JSONObject, a estos métodos le pasamos como parámetro el nombre del atributo definido en el mensaje Json.

Como mencionamos anteriormente, es conveniente representar la entidad que proviene del mensaje Json con una clase Java, por lo que definiremos la clase "Persona" con los mismos atributos que el mensaje, y crearemos los getters y setters para los mismos.

```
public class Persona {
	private String nombre;
	private String apellido;
	private int edad;

	public String getNombre() {
		return nombre;
	}
	public void setNombre(String nombre) {
		this.nombre = nombre;
	}
	public String getApellido() {
		return apellido;
	}
	public void setApellido(String apellido) {
		this.apellido = apellido;
	}
	public int getEdad() {
		return edad;
	}
	public void setEdad(int edad) {
		this.edad = edad;
	}
}
```

De este modo, al parsear el objeto Json, crearemos un objeto "Persona" y le cargaremos los atributos:

```
// String con mensaje Json proveniente de un Web Service
String msgJson = "{'nombre': 'Juan','apellido': 'Perez','edad': 19}";

try {
	// Creamos objeto JSONObject pasandole String
		JSONObject objJson = new JSONObject(msgJson);

		// Obtenemos los valores de los atributos del objeto
		// mediante sus nombres
		Persona persona = new Persona();
		persona.setNombre(objJson.getString("nombre"));
		persona.setApellido(objJson.getString("apellido"));
		persona.setEdad(objJson.getInt("edad"));

		Log.d("json","Objeto parseado:");
		Log.d("json","Nombre:"+persona.getNombre());
		Log.d("json","Apellido:"+persona.getApellido());
		Log.d("json","Edad:"+persona.getEdad());


	} catch (JSONException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}
```

En este ejemplo, creamos un objeto "Persona" y mediante sus "setters" cargamos los atributos que leemos desde el mensaje Json, de esta forma obtenemos un objeto Java a partir de un mensaje Json.

Parseando Arrays

Dijimos que era posible definir en un mensaje Json un array de objetos, si queremos parsear este tipo de mensajes, no debemos generar un objeto del tipo JSONObject, sino un JSONArray :

```
// String con mensaje Json proveniente de un Web Service
String msgJson = "[{'nombre': 'Juan','apellido': 'Perez','edad': 19},{'nombre':
                                'Pedro','apellido': 'Gonzales','edad': 24}]";
try {
	//Creamos un objeto JSONArray
		JSONArray arrayJson = new JSONArray(msgJson);
		for(int i=0;i<arrayJson.length();i++)
		{
			// Obtenemos los objetos Json por cada posicion del Array
			JSONObject objJson = arrayJson.getJSONObject(i);

			// Obtenemos los datos como antes
			//...
		}
	} catch (JSONException e) {
		// TODO Auto-generated catch block
		e.printStackTrace();
	}
```

En este ejemplo, el mensaje es un array de dos objetos Persona, por lo que es pasado en el constructor del objeto JSONArray creado. Mediante el método "length()" podemos saber cuantos objetos posee el array, y mediante el método "getJSONObject()" al cual le pasamos el índice del array (comenzando de cero) obtenemos objetos JSONObject de los cuales podemos obtener sus atributos como se mencionó anteriormente.

Si queremos llevar los objetos de el array Json a objetos Java, podemos crear una colección de objetos Persona (por ejemplo ArrayList), cada objeto creado será agregado a la lista:

```
// String con mensaje Json proveniente de un Web Service
String msgJson = "[{'nombre': 'Juan','apellido': 'Perez','edad': 19},{'nombre':
                                    'Pedro','apellido': 'Gonzales','edad': 24}]";

// Creamos un ArrayList de objetos Persona
ArrayList<Persona> listaPersonas = new ArrayList<Persona>();

try {
        //Creamos un objeto JSONArray
        JSONArray arrayJson = new JSONArray(msgJson);
        for(int i=0;i<arrayJson.length();i++)
        {
                // Obtenemos los objetos Json por cada
                // posicion del Array
                JSONObject objJson = arrayJson.getJSONObject(i);

                Persona persona = new Persona();
                persona.setNombre(objJson.getString("nombre"));
                persona.setApellido(objJson.getString("apellido"));
                persona.setEdad(objJson.getInt("edad"));

                // agregamos el objeto Persona a la lista
                listaPersonas.add(persona);
        }
        } catch (JSONException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }
```

En este ejemplo, cada persona creada es agregada al ArrayList de personas, al terminar el bucle “for” el ArrayList “listaPersonas” contendrá dos objetos Persona con sus atributos cargados.

Parseando atributos que son arrays

Como se dijo anteriormente, es posible que un atributo de un objeto sea un array, en ese caso, en vez de ejecutar el método “getString()” o “getInt()” pasándole el nombre del parámetro, puede ejecutarse el método “getJSONArray()” el cual recibe el nombre del parámetro, y devuelve un objeto del tipo JSONArray, el cual puede ser recorrido como se explico anteriormente.

Un atributo de un objeto Json puede ser un Array, que a su vez puede tener objetos que tengan un atributo que también sea un array, por lo que es posible anidar objetos y arrays tanto como se quiera, y al forma de parsear estas estructuras es ir obteniendo objetos o arrays e iterando los atributos de los mismos.

XML Parsers

Sintaxis

El formato XML (Extensible Markup Languaje) permite crear documentos en un formato que puede ser leído fácilmente tanto para una persona como para una computadora.Se utiliza para representar estructuras de datos.

Ejemplo:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE Edit_Mensaje SYSTEM "Edit_Mensaje.dtd">
<Edit_Mensaje>
    <Mensaje>
        <Remitente>
               <Nombre>Nombre del remitente</Nombre>
               <Mail> Correo del remitente </Mail>
        </Remitente>
        <Destinatario>
               <Nombre>Nombre del destinatario</Nombre>
               <Mail>Correo del destinatario</Mail>
        </Destinatario>
        <Texto>
            <Asunto>
               Este es mi documento con una estructura muy sencilla
               no contiene atributos ni entidades...
            </Asunto>
            <Parrafo>
               Este es mi documento con una estructura muy sencilla
               no contiene atributos ni entidades...
            </Parrafo>
         </Texto>
    </Mensaje>
</Edit_Mensaje>
```

Como se observa, el formato se trata de tags anidados, en donde dentro de cada tag puede haber un texto, u otro/s par/es de tags.

Un tag se abre y se cierra con la siguiente sintaxis:

```
<tag> contenido </tag>
```

El contenido puede ser uno o más pares de tags.
Cada tag puede tener atributos los cuales están compuestos por una clave y un valor:

```
<tag nombre="juan" > Contenido </tag>
```

Cuando el tag no posee contenido, puede abrirse y cerrarse con el mismo tag:

```
<tag nombre="juan" />
```

Parsers

Existen 3 tipos principales de parsers:

- DOM (Document Object Model)
- SAX
- XmlPull

Los parsers tipo DOM cargan en memoria todo el documento. Esto permite acceder a cualquier parte del documento en forma aleatoria.

Los parsers tipo SAX no necesitan cargar en memoria todo el documento. Son "stream-based", parsean el documento a medida que lo leen. No puede accederse a cualquier parte del documento aleatoriamente. El parser comienza a recorrer el XML y genera eventos al encontrar los tags.

Los parsers tipo Pull, son más parecidos a iteradores. El programador generará la petición para "avanzar" y parsear la porción siguiente del archivo.

Parseando con XmlPull

Creamos el objeto parser:

```
XmlPullParser parser = Xml.newPullParser();
```

Le pasamos un InputStream o un Reader (fuente de datos):

```
parser.setInput(new StringReader(xmlTxt));
```

Iteramos al parser y leemos los eventos:

```
event = parser.getEventType();
while (event != XmlPullParser.END_DOCUMENT)
{
	switch (event)
	{
		case XmlPullParser.START_DOCUMENT:
		//...
		case XmlPullParser.START_TAG:
		//...
		case XmlPullParser.END_TAG:
		//...
	}
	event = parser.next();
}
```

Obtenemos el texto dentro del tag, por ejemplo, si el formato es:

<titulo>Texto dentro del tag</titulo>

```
case XmlPullParser.START_TAG:
{
	String tag = parser.getName();
	if (tag.equals("titulo"))
	{
		// inicio del tag <titulo>
		String texto = parser.nextText()
	}
		// Preguntamos por el resto de los tags
		// ...
	break;
}
```

Lectura de atributos. Por ejemplo si el formato es:

<titulo atributo='5' >Texto dentro del tag</titulo>

```
case XmlPullParser.START_TAG:
{
	String tag = parser.getName();
	if (tag.equals("titulo"))
	{
		// inicio del tag <titulo>
		String att = parser.getAttributeValue(null,"atributo");
		String texto = parser.nextText()
	}
	break;
}
```