TABLA DE CONTENIDOS

COMUNICACIONES

Tenemos varias posibilidades en Android para comunicar nuestro dispositivo con otros terminales. Los mecanismos que nos permiten comunicar dos aplicaciones en Internet son los sockets, el uso del protocolo HTTP y los servicios web.

1 COMUNICACIÓN EN INTERNET CON SOCKETS

Un socket es el punto final de una comunicación bidireccional entre dos programas que intercambian información a través de Internet. Una conexión está determinada por un par de sockets que son los extremos de la comunicación. Un socket se identifica por la dirección IP del dispositivo donde está, más un número de puerto (de 16 bits).

Como ejemplo de utilización de sockets en Android veremos una aplicación cliente que realizará peticiones de echo a un servidor.

El servicio de echo suele estar en el puerto 7 y permite comprobar que la máquina está operativa y que se puede establecer una conexión con dicha máquina. El cliente envía datos al servidor y comprueba que los datos que recibe son exactamente iguales a los que envió. El servidor por su parte espera que alguien le envíe algo y le responde con la misma información que recibió.

- CLIENTE ECHO (**Proyecto ClienteECHO**)

```
public class ClienteECHO extends Activity {
    private TextView salida;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        salida = (TextView) findViewById(R.id.TextView01);
        peticionCliente();
    }


    private void peticionCliente() {
        //Socket = IP + PUERTO al que queremos conectarnos
        //En este caso utilizaremos un servidor que está en el mismo ordenador
        String ip = "127.0.0.1";
        int puerto = 7;
        try {
            Socket socket = new Socket(ip,puerto);
            //Para leer del socket
            BufferedReader entrada = new BufferedReader (
                    new InputStreamReader(socket.getInputStream()));
            //Para escribir en el socket
            PrintWriter salida = new PrintWriter(
                    new OutputStreamWriter(socket.getOutputStream()));
            //Enviamos algo al socket, o sea, al servidor
            log("Enviamos una cadena al servidor de echo...");
            salida.println("Hola Cefire Cheste");
            //Mostramos lo que estamos recibiendo
            log("Recibimos del servidor la cadena: " + entrada.readLine());
            socket.close();
        } catch (Exception e) {
            log("Error: " + e.toString());
        }
    }


    private void log(String cadena) {
        salida.append(cadena + "\n");
    }
}
```

No olvidar solicitar el permiso de acceso a Internet, añadiendo la siguiente línea en AndroidManifest.xml:

```
<uses-permission
    android:name="android.permission.INTERNET" />
```

- SERVIDOR ECHO (**Proyecto ServidorECHO**)

Implementaremos un servidor de ECHO en nuestro ordenador. Para eso crearemos un nuevo proyecto pero en este caso NO DE ANDROID sino un PROYECTO JAVA.

```
public class ServidorECHO {

    public static void main(String[] args) {
        try {
            //Creamos un socket en el puerto 7
            ServerSocket socket = new ServerSocket(7);
            System.out.println("Creamos socket");
            //Esperamos a que lleguen peticiones de clientes
            while (true) {
                System.out.println("Espero petición");
                //Obtenemos el cliente desde el que nos están realizando una petición
                Socket cliente = socket.accept();
                System.out.println("Espero petición2");
                //Para leer del socket lo que nos pide el cliente
                BufferedReader entrada = new BufferedReader(
                        new InputStreamReader(cliente.getInputStream()));
                //Para escribir en el socket la respuesta del servidor
                PrintWriter salida = new PrintWriter(
                        new OutputStreamWriter(cliente.getOutputStream()));
                  //Obtenemos la petición del cliente
                  String datos = entrada.readLine();
                  //Escribimos en el socket lo mismo que nos llegó desde el cliente.
                  //De esta forma estamos comportándonos como un servidor de ECHO.
                  salida.println(datos);
                  //Cerramos la conexión con el cliente
                  cliente.close();
               }
            } catch (IOException e) {
                System.out.println(e);
            }
        }

    }
```

2 COMUNICACIÓN EN INTERNET CON EL PROTOCOLO HTTP

La finalidad que tenía en un principio el protocolo HTTP era la de descargar páginas Web. Sin embargo, actualmente se utiliza también para otros fines como el intercambio de ficheros o la comunicación entre aplicaciones.

En Android tenemos dos posibilidades para usar el protocolo Http en nuestras comunicaciones en Internet: java.net.* o org.apache.commons.httpclient.*

En el ejemplo que vamos a ver a continuación para estudiar el funcionamiento del protocolo HTTP en nuestras comunicaciones en Internet aprovecharemos el servicio de RSS que ofrecen muchos periódicos y realizaremos una aplicación mediante la que podremos ver los titulares de las últimas noticias publicadas por dichos periódicos.

Accederemos al servidor mediante la URL: http://www.elpais.com/rss/feed.html?feedId=1022, que nos devuelve una "página" web, aunque en este caso, realmente se tratará de un fichero XML que podremos utilizar para extraer la información que nos interesa.

La respuesta del servidor la obtenemos en una variable de tipo String que decodificaremos más adelante. Buscaremos las apariciones de la cadena "<title>" para extraer los titulares de las noticias.

Por lo tanto, creamos una nueva aplicación "**EjemploHTTP**" y lo primero que hacemos es solicitar el permiso de acceso a Internet, añadiendo la siguiente línea dentro de las etiquetas <application> en AndroidManifest.xml

```
<uses-permission android:name="android.permission.INTERNET" />
```

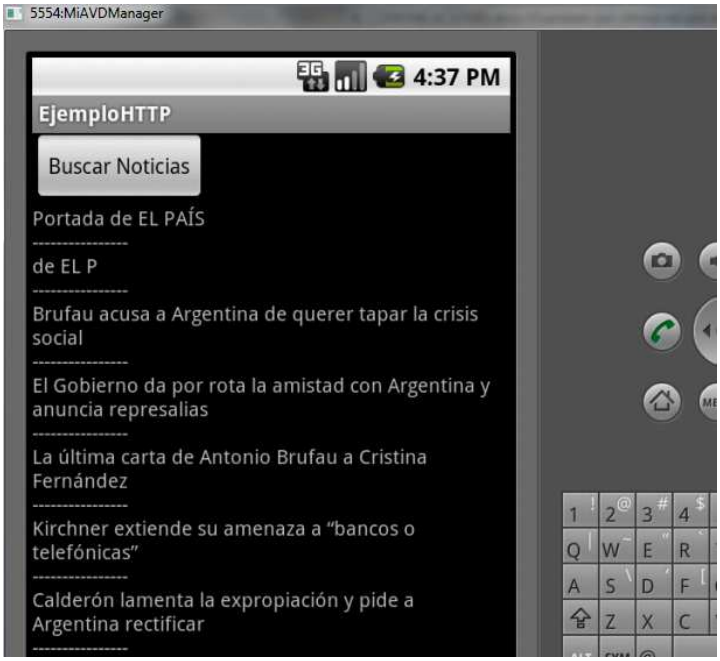
En el Layout main.xml tendremos el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >


    <Button android:id="@+id/Boton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Buscar Noticias" />

    <TextView
        android:id="@+id/texto1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="6pt" />

</LinearLayout>
```

En la clase EjemploHTTP.java tendremos el siguiente código:

```
import java.io.BufferedReader;

public class EjemploHTTP extends Activity {
    private Button boton;
    private TextView texto;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        boton = (Button)findViewById(R.id.Boton1);
        texto = (TextView)findViewById(R.id.texto1);

        //Escuchador del botón. Acciones en caso de pulsar el botón
        boton.setOnClickListener(new OnClickListener() {
            public void onClick(View view) {
                try {
                    String noticias = buscarNoticias();
                    texto.append(noticias);
                } catch (Exception e) {
                    texto.append("Error al conectar");
                    e.printStackTrace();
                }
            }
        });
    }

   private String buscarNoticias() throws Exception {
       String salida="";
       int i=0, j=0;

       //Dirección de la página (RSS) que queremos obtener (Corresponde a titulares del periódico)
       URL url = new URL("http://www.elpais.com/rss/feed.html?feedId=1022");
       //Conexión de tipo HTTP
       HttpURLConnection conexion = (HttpURLConnection) url.openConnection();
       //Añadimos una cabecera HTTP para que identificarnos y evitar obtener un error de aquellos
       //servidores que prohiben la respuesta a aquellos clientes que no se identifican.
       conexion.setRequestProperty("User-Agent", "Mozilla/5.0" +
                                   " (Linux; Android 1.5; es-ES) Ejemplo HTTP");

       if (conexion.getResponseCode() == HttpURLConnection.HTTP_OK) {
           BufferedReader lector = new BufferedReader (
                   new InputStreamReader(conexion.getInputStream()));
           String linea = lector.readLine();
           while (linea != null) {
               //Si encontramos la etiqueta <title> podemos recuperar la noticia
               if (linea.indexOf("<title>") >= 0) {
                   i = linea.indexOf("<title>")+16;
                   j = linea.indexOf("</title>")-3;
                   salida += linea.substring(i,j);
                   salida += "\n----------------\n";
               }
                 //Leemos la siguiente línea
                 linea = lector.readLine();
             }
             lector.close();
       } else {
           salida = "No encontrado";
       }
       conexion.disconnect();

       return salida;
    }
}
```

La salida de la aplicación será algo parecido a esto:



NOTA IMPORTANTE: En caso de estar utilizando un proxy para salir a Internet os dará error en la conexión y no obtendréis los resultados esperados.

3 COMUNICACIÓN EN INTERNET CON SERVICIOS WEB

Un servicio web es una API que es publicada, localizada e invocada a través de la web. Normalmente el transporte de los datos se realiza a través del protocolo HTTP y la representación de los datos mediante XML.

En Internet podemos encontrar varios servicios web públicos mediante los que obtener información interesante. Entre ellos Google nos ofrece uno de tipo REST para obtener datos de una localidad situada en las coordenadas que le digamos. Podemos comenzar a probarlo en un navegador accediendo a la URL: `"http://maps.googleapis.com/maps/api/geocode/json?latlng=" + latitud + "," + longitud + "&sensor=false”`.

Un ejemplo de uso sería:

"http://maps.googleapis.com/maps/api/geocode/json?latlng=38.1525,-0.88972&sensor=false”

Obteniendo como resultado algo parecido a esto (ya sabéis donde buscarme!) codificado en JSON:


```
{
   "results" : [
      {
         "address_components" : [
            {
               "long_name" : "1-4",
               "short_name" : "1-4",
               "types" : [ "street_number" ]
            },
            {
               "long_name" : "Plaza de La Iglesia",
               "short_name" : "Plaza de La Iglesia",
               "types" : [ "route" ]
            },
            {
               "long_name" : "Granja de Rocamora",
               "short_name" : "Granja de Rocamora",
               "types" : [ "locality", "political" ]
            },
            {
               "long_name" : "Alicante",
               "short_name" : "A",
               "types" : [ "administrative_area_level_2", "political" ]
            },
            {
               "long_name" : "Comunidad Valenciana",
               "short_name" : "Comunidad Valenciana",
               "types" : [ "administrative_area_level_1", "political" ]
            },
            {
               "long_name" : "España",
               "short_name" : "ES",
               "types" : [ "country", "political" ]
            },
            ,
```

La respuesta que obtenemos es un objeto JSON (JavaScript Object Notation), que podríamos decir que es “el sucesor de XML”. Lo común cuando se interactúa con Web Services es recibir la respuesta o bien en formato XML, o bien en formato JSON. XML tiene la ventaja de ser muy conocido y de que es increíblemente flexible, además de que está soportado por casi todos los lenguajes. Sin embargo, tiene una gran desventaja con respecto al JSON, y es que requiere de la construcción de un parser, un analizador o escáner, específico para cada tipo de documento XML. Esto significa que para un mismo Web Service, podríamos tener que escribir dos o más escáneres, uno por cada petición distinta que tengamos que hacer (en la práctica, los Web Services intentan minimizar que esto ocurra). Es aquí donde gana el más moderno JSON: funciona por parejas (clave, valor), se adapta a cualquier estructura, cada vez es más soportado, y además, no requiere construir un parser específico, solamente adaptarnos a la respuesta específica.

Realizamos una nueva aplicación (**EjemploWebService**) en la que podremos introducir valores de latitud y longitud, y nos mostrará los datos del lugar que posee dichas coordenadas.

En primer lugar, solicitamos el permiso de acceso a Internet, añadiendo la siguiente línea en AndroidManifest.xml

```
<uses-permission android:name="android.permission.INTERNET" />
```

El código del Layout main.xml puede ser el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <EditText android:id="@+id/longitud"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Introducir longitud" />

    <EditText android:id="@+id/latitud"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Introducir latitud" />

    <Button
        android:id="@+id/boton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Buscar posición" />

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:id="@+id/texto"
        android:textSize="6pt" />
</LinearLayout>
```

Reemplazamos el código de EjemploWebService.java por:

```
package org.aguilar.ejemplowebservice;

import org.apache.http.HttpResponse;

public class EjemploWebService extends Activity {

    private EditText longitud;
    private EditText latitud;
    private Button boton;
    private TextView texto;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        longitud = (EditText)findViewById(R.id.longitud);
        latitud = (EditText)findViewById(R.id.latitud);
        boton = (Button)findViewById(R.id.boton);
        texto = (TextView)findViewById(R.id.texto);

       //Escuchador para el botón
       boton.setOnClickListener(new OnClickListener() {
           public void onClick(View view) {
               try {
                   //Recuperamos los valores introducidos en los campos
                   //de latitud y longitud
                   String longitudCad = longitud.getText().toString();
                   String latitudCad = latitud.getText().toString();
                   String resultado = busquedaGoogle(longitudCad, latitudCad);
                   texto.append(resultado);
               } catch (Exception e) {
                   texto.append("Error al conectar\n");
                   e.printStackTrace();
               }
           }
       });
   }
//Método que conectará con el WebService de Google que nos permite obtener
//los datos de una localización dadas sus coordenadas
String busquedaGoogle(String longitud, String latitud) {
    String devuelve = "";

    //Creamos un nuevo objeto HttpClient que será el encargado de realizar la
    //comunicación HTTP con el servidor a partir de los datos que le damos.
    HttpClient comunicacion = new DefaultHttpClient();
    //Creamos una peticion GET indicando la URL de llamada al servicio.
    HttpGet peticion = new HttpGet("http://maps.googleapis.com/maps/api/geocode/json?" +
                                   "latlng=" + latitud + "," + longitud + "&sensor=false");
                                   //"latlng=38.15,-0.89&sensor=false");
    //Modificamos mediante setHeader el atributo http content-type para indicar
    //que el formato de los datos que utilizaremos en la comunicación será JSON.
    peticion.setHeader("content-type", "application/json");


        try {
            //Ejecutamos la petición y obtenemos la respuesta en forma de cadena
            HttpResponse respuesta = comunicacion.execute(peticion);
            String respuestaCad = EntityUtils.toString(respuesta.getEntity());
            //Creamos un objeto JSONObject para poder acceder a los atributos (campos) del objeto.
            JSONObject respuestaJSON = new JSONObject(respuestaCad);
            //Accedemos al vector de resultados
            JSONArray resultJSON = respuestaJSON.getJSONArray("results");
            //Vamos obteniendo todos los campos que nos interesen.
            //En este caso obtenemos la primera dirección de los resultados.
            String direccion="SIN DATOS PARA ESA LONGITUD Y LATITUD";
            if (resultJSON.length()>0){
                direccion = resultJSON.getJSONObject(0).getString("formatted_address");
            }
            devuelve = "Dirección: " + direccion;
        } catch(Exception e) {
            Log.e("Error ies REST", "ERROR!!", e);
        }

        return devuelve;
    }
}
```

4 COMUNICACIÓN EN SOLOBICI DIDÁCTICO

Vamos a introducir la posibilidad de conectar nuestro juego didáctico a Internet para obtener información. Se trata de que al iniciar el juego el número de vidas dependa de la cantidad de respuestas acertadas sobre un determinado tema que se presentarán en pantalla. De esta forma, el jugador (en nuestro caso los alumnos a los que les pasemos el juego) deben contestar una serie de preguntas y dependiendo del número de respuestas acertadas dispondrán de más o menos vidas para jugar.

Las preguntas se obtendrán de un servidor de Internet que devolverá un fichero XML con una estructura sencilla donde aparecerán las preguntas y las respuestas. El script que nos devuelve las preguntas estará alojado y será accesible mediante la siguiente URL:

http://f5-preview.awardspace.com/cursoandroid.cursoj2me.com/preguntas.php?categoria=1&dificultad=2

Podéis probar a ejecutarlo en un navegador y ver el código fuente que devuelve.

Como podéis comprobar en la petición se admitirá los siguientes parámetros:

- **categoria** -> Será un número de 1 a 4 para las categorías REDES, PROGRAMACIÓN, BASES DE DATOS y SISTEMAS OPERATIVOS respectivamente.
- **dificultad** -> Es el nivel de dificultad de las preguntas que vamos a obtener. Tenemos del 1 al 10 y lo podemos utilizar según en el nivel del juego en el que nos encontremos.

Un ejemplo del fichero XML que obtendríamos mediante esta llamada lo podemos apreciar a continuación:

```
<?xml version='1.0' encoding='iso-8859-1' standalone='no'?>
<preguntas>
    <pregunta>
            <enunciado>pregunta 1_PROGRAMACION</enunciado>
            <respuesta1>respuesta1</respuesta1>
            <respuesta2>respuesta2_ok</respuesta2>
            <respuesta3>respuesta3</respuesta3>
            <respuesta4>respuesta4</respuesta4>
            <correcta>2</correcta>
    </pregunta>
    <pregunta>
            <enunciado>pregunta 2_PROGRAMACION</enunciado>
            <respuesta1>respuesta1_ok</respuesta1>
            <respuesta2>respuesta2</respuesta2>
            <respuesta3>respuesta3</respuesta3>
            <respuesta4>respuesta4</respuesta4>
            <correcta>1</correcta>
    </pregunta>
</preguntas>
```

Como podéis apreciar el servicio que yo os proporciono os devuelve unas preguntas de prueba. Si a alguno de vosotros le apetece puede crear su propio servicio y devolver las preguntas que desee.

4.1 Ejercicio Guiado 1

Una vez analizada la respuesta que obtenemos del servidor lo que vamos a hacer es completar nuestra aplicación Solobici Didáctico para incorporar esta nueva pantalla de preguntas con la que obtener el número de vidas inicial del juego.

Después de pulsar el botón "Jugar" vamos a abrir una pantalla que contenga la lista de preguntas que deben responder los alumnos para después pasar a la pantalla del juego.

1. Creamos una nueva clase (**Pregunta.java**) para especificar los objetos que contendrá cada una de las preguntas:

```
package juego.solobici;

//Contiene la especificación de cada pregunta para formar
//una estructura (vector) que las contenga todas
public class Pregunta {
    String enunciado;
    String[] respuestas;
    String correcta;

    public Pregunta(){
        enunciado=null;
        respuestas=new String[]{null,null,null,null};
        correcta=null;
    }
}
```

2. Creamos un fichero (**preguntas.xml**) que contiene unas cuantas preguntas de prueba en el formato XML visto anteriormente y las guardamos en la carpeta "res/raw".
3. Creamos una nueva clase (**ManejadorXML.java**) que nos servirá para leer el fichero XML de preguntas y que depende de dicho fichero. La programación de un parser XML queda fuera de este curso pero el siguiente ejemplo es bastante claro y supongo que no tendréis problemas en entenderlo.

```
package juego.solobici;

import java.util.ArrayList;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class ManejadorXML extends DefaultHandler {
    private StringBuilder cadena;
    private Pregunta pregunta;
    private ArrayList<Pregunta> listaPreguntas;

    public ArrayList<Pregunta> getPreguntas(){
        return listaPreguntas;
    }

    @Override
    public void startDocument() throws SAXException {
        listaPreguntas = new ArrayList<Pregunta>();
        cadena = new StringBuilder();
    }
```

```
    @Override
    public void startElement(String uri, String nombreLocal,
            String nombreCualif, Attributes atr) throws SAXException {
        if (nombreLocal.equals("pregunta")) {
            pregunta = new Pregunta();
        }
    }

    @Override
    public void characters(char ch[], int comienzo, int longitud) {
        cadena.append(ch, comienzo, longitud);
    }

    @Override
    public void endElement(String uri, String nombreLocal,
            String nombreCualif) throws SAXException {
        if (nombreLocal.equals("enunciado")) {
            pregunta.enunciado = cadena.toString();
        } else if (nombreLocal.equals("respuesta1")) {
            pregunta.respuestas[0] = cadena.toString();
        } else if (nombreLocal.equals("respuesta2")) {
            pregunta.respuestas[1] = cadena.toString();
        } else if (nombreLocal.equals("respuesta3")) {
            pregunta.respuestas[2] = cadena.toString();
        } else if (nombreLocal.equals("respuesta4")) {
            pregunta.respuestas[3] = cadena.toString();
        } else if (nombreLocal.equals("correcta")) {
            pregunta.correcta = cadena.toString();
            //Al llegar al último campo añadimos la pregunta
            //a la lista de preguntas
            listaPreguntas.add(pregunta);
        }
        cadena.setLength(0);
    }

    @Override
    public void endDocument() throws SAXException {

    }
}
```

4. Creamos una nueva clase (**AlmacenPreguntas.java**) que se encargará de leer los datos del fichero XML (En este caso no accedemos a una URL para obtener el fichero XML con las preguntas sino que las obtenemos de un fichero XML que hemos introducido en el paso 2 como recurso de la aplicación en la carpeta "res/raw" y que se denomina preguntas.xml).

```
package juego.solobici;

import java.io.InputStream;

public class AlmacenPreguntas {
    private Context contexto;
    private ArrayList<Pregunta> listaPreguntas;
    private boolean cargadaLista;

    public AlmacenPreguntas(Context contexto) {
        this.contexto = contexto;
        listaPreguntas = new ArrayList<Pregunta>();
        cargadaLista = false;
    }


   //Método donde obtenemos la lista de preguntas contenidas
   //en el fichero XML
   public ArrayList<Pregunta> listaPreguntas() {
       try {
           if (!cargadaLista) {
               //A este método le estamos pasando el fichero XML de preguntas abierto
               //Esta será la llamada que tengáis que modificar en el EJERCICIO
               //para leer las preguntas desde un servidor que yo os proporciono
               //en lugar de leerlo desde el fichero
               listaPreguntas = leerXML(contexto.getResources().openRawResource(R.raw.preguntas));
           }
       } catch (Exception e) {
           Log.e("Almacen Preguntas", e.getMessage(), e);
       }
       return listaPreguntas;
   }


    ArrayList<Pregunta> leerXML(InputStream entrada) throws Exception {
        //Las siguientes 3 líneas tienen como objetivo obtener un lector para leer el XML
        SAXParserFactory fabrica = SAXParserFactory.newInstance();
        SAXParser parser = fabrica.newSAXParser();
        XMLReader lector = parser.getXMLReader();

        //Creamos un manejador que nos servirá para leer el archivo XML
        //Este manejador SOLO nos servirá para este archivo XML que tiene una
        //determinada estructura. Para otro XML distinto tendríamos que hacer
        //otro manejador. Esta es una de las desventajas de SAX.
        ManejadorXML manejadorXML = new ManejadorXML();
        lector.setContentHandler(manejadorXML);
        lector.parse(new InputSource(entrada));

        cargadaLista=true;

        return manejadorXML.getPreguntas();
    }
}
```

5. Abrimos una nueva pantalla con los controles de tipo Spinner y TextView que contendrán las preguntas y respuestas.
    a. Creamos un layout denominado “listapreguntas.xml” con un elemento principal al que después le asignaremos con código los elementos correspondientes (etiquetas con las preguntas y listas con las respuestas).

```xml
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:id="@+id/miLinearLayout" >
</LinearLayout>
```

    b. Creamos una nueva actividad “Preguntas.java” que contendrá el layout creado anteriormente y por lo tanto la pantalla con las preguntas y respuestas.

```java
package juego.solobici;

import java.util.ArrayList;

public class Preguntas extends Activity{
    private LinearLayout interfaz;
    private ArrayList<Pregunta> listaPreguntas;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.listapreguntas);

        //Rellenamos las estructuras de datos con la información que recibimos del servidor
        //o del fichero de preguntas al que invocamos para obtener preguntas y respuestas
        AlmacenPreguntas almacen = new AlmacenPreguntas(this);
        listaPreguntas = almacen.listaPreguntas();
           /////System.out.println("Pregunta0" + listaPreguntas.get(0).enunciado);

        //Accedemos al elemento principal y raíz del layout
        interfaz = (LinearLayout)findViewById(R.id.miLinearLayout);


       //Preparamos los campos de texto y las listas desplegables correpondientes
       TextView[] pregunta=new TextView[listaPreguntas.size()];
       final Spinner[] respuesta=new Spinner[listaPreguntas.size()];
       //Creación del adaptador al que pasamos 3 parámetros
       //1. El contexto, que normalmente será una referencia a la actividad
       //   donde se crea el adaptador.
       //2. El ID del layout sobre el que se mostrarán los datos del control.
       //   En este caso le pasamos el ID de un layout predefinido en Android
       //   (android.R.layout.simple_spinner_item), formado únicamente por un
       //   control TextView, pero podríamos pasarle el ID de cualquier layout
       //   de nuestro proyecto con cualquier estructura y conjunto de controles.
       //3. El array que contiene los datos a mostrar.
       ArrayAdapter<String> adaptador=null;
```

```
        //Añadimos cada pregunta y lista desplegable al layout
        for (int i=0; i<listaPreguntas.size(); i++) {
            listaPreguntas.get(i);
            pregunta[i] = new TextView(this);
            pregunta[i].setText(listaPreguntas.get(i).enunciado);
            interfaz.addView(pregunta[i]);
            respuesta[i] = new Spinner(this);
            interfaz.addView(respuesta[i]);

            //Preparamos el adaptador que contendrá los datos/respuestas de cada pregunta
            adaptador = new ArrayAdapter<String>(this,
                        android.R.layout.simple_spinner_item, listaPreguntas.get(i).respuestas);
            //Podemos personalizar también el aspecto de cada elemento en la lista emergente
            adaptador.setDropDownViewResource(
                       android.R.layout.simple_spinner_dropdown_item);
            //Asignamos el adaptador al control (o sea, a la lista de respuestas
            respuesta[i].setAdapter(adaptador);
```

```
        }
    //Añadimos un botón para realizar la corrección
        //de las preguntas y pasar al juego
    Button corregir = new Button(this);
    corregir.setText("Corregir");
    interfaz.addView(corregir);
    //Añadimos un escuchador y cuando pulsen el botón para corregir
    //comprobaremos el número de preguntas acertadas y pasaremos al juego
       corregir.setOnClickListener(new OnClickListener(){
           public void onClick(View view) {
               int acertadas=0;
               for (int i=0; i<listaPreguntas.size(); i++) {
                   int seleccionada = respuesta[i].getSelectedItemPosition();
                   String correcta = listaPreguntas.get(i).correcta;
                   //System.out.println("seleccionada=" + String.valueOf(seleccionada+1) + " correcta
                   //System.out.println("enunciado" + listaPreguntas.get(i).enunciado);
                   if (String.valueOf(seleccionada+1).equals(correcta)) {
                       acertadas++;
                       //System.out.println("Pregunta" + (i+1) + " acertada");
                   }
               }
               System.out.println("Acertadas=" + acertadas);
           }
       });
    }
}
```

c. Al pulsar el botón “Corregir” informamos mediante una pantalla o Toast de las preguntas acertadas y falladas.
   Para ello incorporamos al final del método onClick() de la clase Preguntas.java la llamada a un método que también mostramos a continuación:

```
//Mostramos el número de aciertos mediante un Toast.
mostrarAciertos(acertadas);
```

```
public void mostrarAciertos(int acertadas) {
   Toast mensaje = Toast.makeText(Preguntas.this, "Has acertado " + acertadas +
           " preguntas y por lo tanto el número de vidas en el juego será de " + acertadas + ".",
           Toast.LENGTH_LONG);
   //Posicionamos el mensaje en la pantalla
   mensaje.setGravity(Gravity.CENTER_VERTICAL|Gravity.CENTER, 0, 0);
   mensaje.show();
}
```

d. Al pulsar el botón “Jugar” de la pantalla de preguntas abrimos la pantalla con el juego Solobici y dispondremos como número de vidas las preguntas acertadas.
Para poder hacer eso incorporamos después de la llamada al método mostraraciertos() una llamada al método lanzaJuego que mostramos a continuación.

```
//Cambiamos de pantalla y abrimos la del juego
lanzaJuego(acertadas);

public void lanzaJuego(int numVidas) {
   Intent i = new Intent(Preguntas.this, Juego.class);
   startActivityForResult(i, numVidas);
}
```

6. Debemos hacer que en lugar de abrir la pantalla del juego directamente al pulsar en el botón “Jugar” del menú principal se abra la pantalla de preguntas. Para ello en la clase Solobici.java, cambiamos la llamada a lanzarJuego() por una llamada a lanzarPreguntas(). Creamos entonces el método lanzarPreguntas:

```
public void lanzarPreguntas(){
    Intent i = new Intent(this, Preguntas.class);
    startActivity(i);
}
```

7. Añadimos por último la actividad Preguntas al fichero AndroidManifest.xml:

```
<activity android:name=".Preguntas"
    android:label="Preguntas" >
</activity>
```

4.2 Ejercicio Práctico 1

Realizar los cambios necesarios en la clase AlmacenPreguntas.java para que en lugar de obtener las preguntas y respuestas de un fichero XML facilitado como recurso de la aplicación, se obtengan de la URL: http://f5-preview.awardspace.com/cursoandroid.cursoj2me.com/preguntas.php?categoria=1&dificultad=2 donde los parámetros categoría y dificultad pueden cambiar de la forma que ya sabemos.

AYUDA: Seguramente necesitaréis convertir un String (que será lo que se reciba del servidor) en un InputStream (que es lo que necesita como parámetro el método leerXML de la clase AlmacenPreguntas). Pues bien, para realizar esa transformación podéis utilizar la instrucción:

```
ByteArrayInputStream preguntas = new ByteArrayInputStream(xmlPreguntas.getBytes());
```

Donde “preguntas” sería el InputStream que podéis pasar a “leerXML” y xmlPreguntas será el String que obtenemos del servidor.

EN EL SIGUIENTE EJERCICIO SE INDICA LO QUE DEBÉIS ENTREGAR EN ESTE TEMA.

4.3 Ejercicio Práctico 2

Realizar los cambios necesarios para que al terminar una partida se devuelva la puntuación a Solobici.java, tal y como ocurría antes de introducir la pantalla de las preguntas.

La secuencia de llamadas que tenemos es la siguiente:

Solobici -> Llama a la actividad Preguntas

Preguntas -> Llama a la actividad Juego

Cuando Juego termina devuelve la puntuación obtenida a la actividad que lo llamó, en este caso Preguntas. Así que deberíamos hacer que Preguntas devuelva lo que le llega desde la actividad Juego a la actividad que lo llamó, o sea, Solobici.

ENVIAR CAPTURAS DE PANTALLA DONDE SE PUEDA APRECIAR TODO EL CICLO DEL JUEGO, O SEA, PANTALLA DE MENÚ PRINCIPAL, PANTALLA DE PREGUNTAS, PANTALLA CON EL TOAST DE PREGUNTAS ACERTADAS, PANTALLA CON PUNTUACIONES AL FINALIZAR LA PARTIDA PULSANDO "Q".

4.4 Ejercicio Práctico 3 (OPCIONAL)

Antes de mostrar la pantalla de preguntas, mostrar una pantalla en la que preguntemos la "categoría" de preguntas que queremos obtener y el "nivel" de dificultad. Estos datos se deben utilizar para realizar la petición de preguntas al servidor.

ENVIAR CAPTURA DE PANTALLA DONDE SE VEA ESTA NUEVA PANTALLA.