Capítulo 12
Bluetooth

Resumen: En este capítulo nos adentraremos en el uso de bluetooth, el mecanismo para conexión entre dispositivos cercanos mediante radio, que permite construir redes inalámbricas de área personal. Para poder realizar las prácticas de este capítulo, necesitarás, al menos, un dispositivo físico Android con bluetooth, dado que el emulador no soporta bluetooth. Si quieres, además, probar las dos últimas prácticas, con un cliente y un servidor, necesitarás dos.

Práctica 5.1: Activar bluetooth

El punto de entrada a la interacción con bluetooth en Android se consigue a través de la clase BluetoothAdapter, que sirve para controlar un adaptador bluetooth. La clase dispone de un método estático, getDefaultAdapter(), que devuelve una instancia al adaptador predefinido. Fíjate que esto en principio podría hacer pensar que un dispositivo podría contar con más de un adaptador bluetooth; sin embargo, no hay forma de conseguirlos.

Si el dispositivo tiene soporte para bluetooth, conseguiremos un objeto con el que encuestar sobre su estado. El primer método que utilizaremos será isEnabled(), que indica si el usuario tiene o no activa la comunicación porbluetooth.

Si está habilitado, podremos continuar utilizándolo. Si no, podemos pedir al sistema que solicite al usuario que lo active. Para eso, lanzamos un intent con una solicitud predefinida, que muestra un cuadro de diálogo modal. La actividad recibirá la respuesta en su método onActivityResult(), como ocurre siempre con las actividades de este tipo. El cuadro de diálogo devolverá RESULT_OK si el usuario habilitó el adaptador, y RESULT_CANCEL en otro caso.

Vamos a hacer una aplicación que, en el onCreate(), busca el adaptador bluetooth, comprueba si está habilitado, y si no lo está le pide al usuario que lo habilite.

1. Crea un nuevo proyecto y llámalo libro.azul.ActivarBluetooth.
2. Modifica el fichero de manifiesto para requerir el permiso de utilizar el adaptador bluetooth. De otro modo la aplicación fallará en ejecución.

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

3. Añade en el fichero de cadenas los mensajes sobre el estado de bluetooth que daremos al usuario:

- **noBluetooth**: No se encontró adaptador bluetooth
- **bluetoothDesactivado**: Bluetooth deshabilitado
- **bluetoothActivado**: Bluetooth habilitado

4. Define un atributo BluetoothAdapter. En el método onCreate(), inicial ízalo con el objeto para acceder al adaptador predefinido, y comprueba su disponibilidad. Si está habilitado, llama a un método, que escribiremos más adelante, llamado

updateBluetoothUI(). Si no, lanza la actividad pidiendo al usuario que lo habilite.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
....
tvEstadoBluetooth = (TextView)
findViewById(R.id.estadoBluetooth);
bluetooth = BluetoothAdapter.getDefaultAdapter();
if (savedInstanceState != null)
return;
if (bluetooth == null)
tvEstadoBluetooth.setText(R.string.noBluetooth);
else {
if (bluetooth.isEnabled())
updateBluetoothUI();
else {
Intent intent;
intent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
startActivityForResult(intent, CALLBACK_ENABLE_BLUETOOTH);

}
}
}
private BluetoothAdapter bluetooth;
private TextView tvEstadoBluetooth;
private final static int CALLBACK_ENABLE_BLUETOOTH = 1;
```

5. Sobreescribe el método onActivityResult() para que reaccione ante la respuesta del cuadro de diálogo.

```
@Override
protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
if (requestCode == CALLBACK_ENABLE_BLUETOOTH) {
if (resultCode == RESULT_OK) {
updateBluetoothUI();
}
else {
tvEstadoBluetooth.setText(R.string.bluetoothDesactivado);
}
}
}
```

6. Implementa el método updateBluetoothUI() para que se actualice la etiqueta y nos muestre los datos del adaptador.

```
private void updateBluetoothUI() {
String msj;
msj = getString(R.string.bluetoothActivado) +
": " + bluetooth.getName() +
" (" + bluetooth.getAddress() + ")";
tvEstadoBluetooth.setText(msj);
} // initBluetoothUI
```

7. Lanza la aplicación sobre un dispositivo físico y pruébala. Los AVD no tienen emulación; puedes probar a ejecutar la aplicación y deberías ver el mensaje correspondiente.

Práctica 5.2: Detectar cambios de estado

En la práctica anterior configuramos la vista en función del estado del adaptador al inicio de la ejecución, pero éste podría cambiar en cualquier momento.

Cuando el estado del adaptador cambia, Android lo notifica globalmente a través de un mensaje de difusión. Podemos registrarnos de esos mensajes para reaccionar en corcondancia. En esta práctica, vamos a añadir un receptor de esos mensajes para actualizar el interfaz ante un cambio de estado.

1. Crea una clase interna y llámala BluetoothMonitor. Haz que herede de BroadcastReceiver. En su método onReceive() obtén del intent el nuevo estado del bluetooth y llama al método updateBluetoothUI() de la clase principal pasándo ese estado como parámetro. Añadiremos el argumento al método más adelante.
2. Crea un atributo nuevo de esa clase.

```
class BluetoothMonitor extends BroadcastReceiver {
@Override
public void onReceive(Context context, Intent intent) {
String action = intent.getAction();
if (action.equals(BluetoothAdapter.ACTION_STATE_CHANGED)) {
int state = intent.getIntExtra(
BluetoothAdapter.EXTRA_STATE, -1);
// En EXTRA_PREVIOUS_STATE tenemos el estado anterior,
// pero no lo usaremos.
updateBluetoothUI(state);
}
} // onReceive
} // BluetoothMonitor
private BluetoothMonitor bluetoothMonitor = new BluetoothMonitor();
```

3. En el onCreate() registra el receptor de los mensajes de difusión. Tendrás que hacerlo siempre, incluso cuando se recree la actividad. Dado que vamos a añadir un parámetro a updateBluetoothUI, lo aprovecharemos para que nuestra actividad indique correctamente el estado si se rota y el onCreate() podemos reestructurarlo un poco.

```
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
tvEstadoBluetooth = (TextView)
findViewById(R.id.estadoBluetooth);
bluetooth = BluetoothAdapter.getDefaultAdapter();
if (bluetooth == null)
tvEstadoBluetooth.setText(R.string.noBluetooth);
else {
registerReceiver(bluetoothMonitor,
new IntentFilter(
BluetoothAdapter.ACTION_STATE_CHANGED));
```

```
updateBluetoothUI(bluetooth.getState());
}
if (savedInstanceState != null)
return;
if (!bluetooth.isEnabled()) {
Intent intent =
new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
startActivityForResult(intent, CALLBACK_ENABLE_BLUETOOTH);
} // if-else bluetooth está activo
} // onCreate
```

4. Implementa el método onDestroy() para desregistrar el receptor.

```
@Override
protected void onDestroy() {
super.onDestroy();
unregisterReceiver(bluetoothMonitor);
}
```

5. Dado que ahora recibiremos por mensajes los cambios en el estado del bluetooth, no necesitamos procesar la respuesta del cuadro de diálogo en el que pedíamos al usuario que habilitara el bluetooth. Elimina el método onActivityResult().
6. Añade dos nuevas cadenas en el fichero de recursos:

- **bluetoothActivandose**: Habilitando bluetooth...
- **bluetoothDesactivandose**: Deshabilitando bluetooth...

7. Modifica el método updateBluetoothUI() para que reciba un parámetro con el estado y muestre la cadena que corresponda. Por ser más complejo, lleva a un método diferente la reacción ante la detección de bluetooth activado.

```
private void updateBluetoothUI(int newState) {
int str;
switch(newState) {
case BluetoothAdapter.STATE_ON:
updateONBluetoothUI();
return;
case BluetoothAdapter.STATE_TURNING_ON:
str = R.string.bluetoothActivandose;
break;
case BluetoothAdapter.STATE_TURNING_OFF:
str = R.string.bluetoothDesactivandose;
break;
case BluetoothAdapter.STATE_OFF:
str = R.string.bluetoothDesactivado;
break;
default:
return; // ¿?
} // switch
tvEstadoBluetooth.setText(str);
} // initBluetoothUI
private void updateONBluetoothUI() {
String msj = getString(R.string.bluetoothActivado) +
": " + bluetooth.getName() +
```

```
"(" + bluetooth.getAddress() + ")";
tvEstadoBluetooth.setText(msj);
} // updateONBluetoothUI
```

8. Ejecuta la práctica en el móvil. Modifica el estado del bluetooth y comprueba cómo se actualiza la etiqueta.

Práctica 5.3: Visibilidad del dispositivo

Aunque el bluetooth esté activado, por seguridad Android oculta su existencia de manera que otros dispositivos no puedan encontrarle aunque hagan un escaneo. Como ocurría con el estado global, es posible preguntar al adaptador sobre su estado de visibilidad, recibir notificaciones de difusión cuando ésta cambia, y pedir al usuario que haga visible el dispositivo durante 120 segundos.

En esta práctica vamos a mostrar al usuario la visibilidad de su dispositivo cuando el bluetooth esté habilitado, permitiéndole, además, hacer visible si no lo estaba ya.

1. Modifica el layout para que el nodo raíz sea un LinearLayout con orientación en vertical.
2. Añade dentro un RelativeLayout debajo de la etiqueta, y dale como identificador @+id/panelBTOn. En él incluiremos los controles que mostraremos cuando el bluetooth esté habilitado, por lo que lo mostraremos u ocultaremos en bloque.
3. Añade dentro una etiqueta, donde mostraremos el estado de visibilidad del dispositivo a través de bluetooth, y un botón que el usuario podrá pulsar para hacerlo visible temporalmente.

```
<LinearLayout ...
android:orientation="vertical">
...
<RelativeLayout
android:id="@+id/panelBTOn"
android:layout_width="match_parent"
android:layout_height="match_parent">
<TextView
android:id="@+id/estadoVisibilidad"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_alignParentTop="true"/>
<Button
android:id="@+id/btHacerVisible"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_below="@+id/estadoVisibilidad"
android:text="@string/hacerVisible"
android:onClick="onHacerVisible"/>
</RelativeLayout>
</LinearLayout>
```

4. Añade las nuevas cadenas:

- **bluetoothVisible**: El dispositivo es visible por bluetooth
- **bluetoothConectable**: El dispositivo es visible para dispositivos emparejados

- **bluetoothInvisible**: El dispositivo es invisible por bluetooth
- **hacerVisible**: Hacer visible

5. Añade un método updateVisibilityUI() que será llamado para actualizar la etiqueta y el botón relacionados con la visibilidad del dispositivo. Recibirá como parámetro el tipo de visibilidad actual, y mostrar á una cadena u otra, y activará o no el botón. El método será llamado únicamente cuando el bluetooth esté activado.

```
private void updateVisitilityUI(int state) {
int str;
switch(state) {
case BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE:
str = R.string.bluetoothVisible;
break;
case BluetoothAdapter.SCAN_MODE_CONNECTABLE:
str = R.string.bluetoothConectable;
break;
case BluetoothAdapter.SCAN_MODE_NONE:
str = R.string.bluetoothInvisible;
break;
default:
return;
}
TextView tv = (TextView) findViewById(R.id.estadoVisibilidad);
tv.setText(str);
boolean botonHabilitado;
botonHabilitado =
(state !=
BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE)
findViewById(R.id.btHacerVisible).setEnabled(botonHabilitado);
}
```

6. Amplía el receptor de mensajes del sistema para que atienda también a las notificaciones relacionadas con la visibilidad. Modifica el onCreate() para registrarlo también para ese tipo de mensajes.

```
protected void onCreate(Bundle savedInstanceState) {
...
else {
registerReceiver(bluetoothMonitor,
new IntentFilter(
BluetoothAdapter.ACTION_STATE_CHANGED));
registerReceiver(bluetoothMonitor,
new IntentFilter(
BluetoothAdapter.ACTION_SCAN_MODE_CHANGED));
updateBluetoothUI(bluetooth.getState());
}
...
}
class BluetoothMonitor extends BroadcastReceiver {
@Override
public void onReceive(Context context, Intent intent) {
String action = intent.getAction();
if (action.equals(BluetoothAdapter.ACTION_STATE_CHANGED)) {
```

```
...
}
else if (action.equals(
BluetoothAdapter.ACTION_SCAN_MODE_CHANGED)) {
int state = intent.getIntExtra(
BluetoothAdapter.EXTRA_SCAN_MODE, -1);
// En EXTRA_PREVIOUS_SCAN_MODE tenemos el estado anterior,
// pero no lo usaremos.
updateVisitilityUI(state);
}
} // onReceive
} // BluetoothMonitor
```

7. Implementa el método asociado al evento de pulsación, para que se pida a Android que le pregunte al usuario si quiere hacer visible el dispositivo. Declara una constante en la clase con el valor que nos devolverá el cuadro de diálogo en onActivityResult(), aunque dado que tenemos el receptor de los mensajes de difusión no lo atenderemos.

```
public void onHacerVisible(View v) {
startActivityForResult(
new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE),
CALLBACK_REQUEST_DISCOVERABLE);
}
private final static int CALLBACK_REQUEST_DISCOVERABLE = 2;
```

8. Cuando se actualiza la vista general del estado de bluetooth hay que ocultar todos los controles relacionados con el estado "habilitado" de bluetooth. Modifica updateBluetoothUI() para que oculte siempre el panel, dado que si el bluetooth estaba visible salíamos antes.

```
private void updateBluetoothUI(int newState) {
...
switch(newState) {
...
} // switch
...
findViewById(R.id.panelBTOn).setVisibility(View.INVISIBLE);
} // initBluetoothUI
```

9. Del mismo modo, en el método llamado cuando bluetooth está habilitado, haz visible el panel y actualiza la etiqueta de visibilidad.

```
private void updateONBluetoothUI() {
...
updateVisitilityUI(bluetooth.getScanMode());
findViewById(R.id.panelBTOn).setVisibility(View.VISIBLE);
} // updateONBluetoothUI
```

10. Ejecuta la práctica. Comprueba que funciona correctamente.

Práctica 5.4: Dispositivos emparejados

Aunque es posible desde una actividad poner al adaptador en modo "búsqueda", e informar al usuario de los dispositivos encontrados, La búsqueda es una tarea

asíncrona que requiere trabajo en segundo plano que metería mucho ruido en la práctica.

Lo que es sencillo es averiguar los dispositivos emparejados, es decir aquellos con los que ya se ha realizado alguna comunicación previa y son capaces de comunicarse de manera encriptada. Para eso, basta con llamar al método getBoundedDevices() del objeto BluetoothAdapter. Obtendremos un Set<BluetoothDevice>, con todos los dispositivos, a los que les podremos pedir el nombre, el tipo, dirección y mucha otra información de interés.

En esta práctica vamos a incluir una lista a nuestra ventana, que muestre los dispositivos bluetooth emparejados. Por simplicidad, utilizaremos una lista básica aprovechando el soporte de Android. Dado que la conversión de los BluetoothDevice a cadena devuelven la dirección física (y no el nombre) será eso lo que mostremos.

1. Añade una nueva cadena noDevices con texto No hay dispositivos emparejados.
2. Modifica el layout para añadir, dentro del bloque visible sólo cuando bluetooth está activo, una lista y una etiqueta que se mostrará si no hay dispositivos emparejados.

```
<ListView
android:id="@+id/listaDispositivos"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_below="@+id/btHacerVisible"
android:layout_alignParentBottom="true"/>
<TextView
android:id="@+id/noDevices"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:gravity="center"
android:layout_below="@+id/btHacerVisible"
android:layout_alignParentBottom="true"
android:text="@string/noDevices"/>
```

3. Crea en la clase el atributo listaDispositivos de tipo ListView. En el onCreate(), busca la lista, asígnala al atributo, y asóciala la vista a mostrar cuando la lista esté vacía.

```
protected void onCreate(Bundle savedInstanceState) {
...
listaDispositivos = (ListView)
findViewById(R.id.listaDispositivos);
listaDispositivos.setEmptyView(findViewById(R.id.noDevices));
...
}
private ListView listaDispositivos;
```

4. Añade en el método updateONBluetoothUI() código para obtener la lista de dispositivos emparejados y añadirlos a la lista:

```
private void updateONBluetoothUI() {
...
Set<BluetoothDevice> devices;
```

```
devices = bluetooth.getBondedDevices();
final ArrayAdapter<BluetoothDevice> aa;
aa = new ArrayAdapter(this,
android.R.layout.simple_list_item_1,
devices.toArray());
listaDispositivos.setAdapter(aa);
for (BluetoothDevice bt : devices) {
android.util.Log.i("BT", bt.getName());
}
} // updateONBluetoothUI
```

5. Prueba la práctica. Deberías ver aparecer los nombres de los dispositivos con los que hayas intercambiado algo por bluetooth con el móvil. Si no ves nada, ve a la configuración de bluetooth y empareja tu dispositivo con el de algún compañero y vuelve a probar.

Práctica 5.5: Servidor bluetooth

En esta práctica haremos que la aplicación se quede escuchando a la espera de algún cliente. Para poderla probar, necesitaremos hacer también la práctica siguiente, y ejecutarlas en dos móviles diferentes que estén emparejados.

En este caso, haremos que el dispositivo se quede siempre escuchando en el adaptador. En bluetooth no existe el concepto de _puerto_. En lugar de eso, las aplicaciones indican su deseo de quedarse escuchando en un UUID (Universally unique identifier[1]. Los clientes de ese servicio se conectan usando ese UUID, sabiendo que en el otro extremo estará la aplicación servidora.

No obstante, dado que sobre bluetooth se ejecutan diferentes tipos de servicios, existen algunos UUID predeterminados. Más allá de eso, los primeros bytes de los UUID especifican la clase de servicio que se proporciona, de lo que depende en última instancia cómo funcionará el protocolo. Para el tipo de comunicación que nos ocupa, el UUID deberá comenzar por 00001101.

Como servidores, el primer paso es indicar al adaptador nuestro deseo de escuchar en un UUID. Obtendremos un server socket de bluetooth, sobre el que nos sentaremos a esperar un cliente. Cuando llegue, el sistema nos devolver á un socket con el que podremos comunicarnos con el cliente. El server socket lo cerraremos y, llegado el caso, lo abriremos otra vez posteriormente. Esto es necesario porque bluetooth no soporta varios clientes simultáneos sobre el mismo UUID.

Ten en cuenta que muchas de las etapas del proceso requieren tiempo (en concreto, la espera del cliente es bloqueante). Debido a eso tendremos que realizarlo en una hebra independiente para no bloquear la hebra principal de la aplicación.

En la práctica, crearemos una nueva hebra con todo el código del servidor. Cuando detectemos que el bluetooth está activado, lanzaremos la hebra sobre un UUID para

[1] Un UUID es un número de 16 bytes aleatorio que sirve como identificador universal. Cuando se necesita un identificador único para algo, se genera un UUID nuevo (de manera aleatoria). Dado el ancho de su representación, se considera estadísticamente imposible que el UUID obtenido vaya a coincidir con el obtenido por nadie más en el mundo: si cada habitante de la tierra se dedicara a obtener 1.000 UUID por segundo se necesitarían más de 100 millones de veces la edad del universo para obtenerlos todos.

que espere al cliente. En cuanto llegue, le mandaremos una cadena de saludo, nos desconectaremos, y empezaremos otra vez.

Cuando la actividad se destruya, detendremos también la hebra a la espera de clientes. Para eso, basta con "cancelar" el server socket, lo que hará que la llamada al método que espera un cliente vuelva inmediatamente.

1. Haz una copia de la práctica de la sección anterior.
2. Añade el permiso BLUETOOTH_ADMIN para que la aplicación pueda hacer uso de comunicación por bluetooth.
3. Crea la clase ServerThread que heredará de Thread y que hará todo el trabajo del lado del servidor. Crea un atributo de dicha clase, sin inicializar.

```
UUID SERVICE_UUID = UUID.fromString("00001101-....");
private class ServerThread extends Thread {
BluetoothServerSocket _serverSocket;
public void run() {
while (!isInterrupted()) {
try {
_serverSocket =
bluetooth.listenUsingRfcommWithServiceRecord(
"LibroAzul", SERVICE_UUID);
} catch (IOException e) {
escribe("ERROR: no pude obtener el server socket");
break;
}
BluetoothSocket socket = null;
try {
socket = _serverSocket.accept();
} catch (IOException e) {
// No hacemos nada. Quizá nos cancelaron...
}
if (socket != null) {
// ¡Tenemos un cliente! Cerramos el server socket.
try {
_serverSocket.close();
_serverSocket = null;
} catch (IOException e) {
}
escribe("¡¡NOS HA LLEGADO UN CLIENTE!!");
try {
socket.close();
} catch (IOException e) {
}
}
} // while
} // run
// Genera un toast con el mensaje.
private void escribe(final String str) {
runOnUiThread(new Runnable() {
@Override
public void run() {
Toast.makeText(MainActivity.this, str,
Toast.LENGTH_LONG).show();
}
```

```
});
}
// Detenemos la hebra
public void cancel() {
try {
interrupt();
if (_serverSocket != null)
_serverSocket.close();
} catch (IOException e) { }
}
}
ServerThread serverThread;
```

4. En el método updateONBluetoothUI() crea una hebra nueva de la clase y iniciala.

```
serverThread = new ServerThread();
serverThread.start();
```

5. En el método onDestroy() anula la ejecución de la hebra del servidor para liberar recursos.

```
if (serverThread != null)
serverThread.cancel();
```

De momento, no podrás probar la práctica hasta que no hagas la siguiente.

Práctica 5.6: Cliente bluetooth

El lado del cliente es similar. Necesitamos conocer el UUID, y usarlo para obtener un socket. Ten en cuenta que el server socket lo conseguimos a través del BluetoothAdapter; en el lado del cliente sin embargo hay que utilizar ya BluetoothDevice, pues el socket se crea con el objetivo de conectarse directamente a un dispositivo. Una vez conseguido el socket, se solicita la conexión, y se espera. Si todo va bien, conseguiremos una conexión. La conexión y comunicación con un dispositivo se ve seriamente afectada si el adaptador está en modo búsqueda, por lo que antes de intentar conectarnos con el servidor se recomienda detener el potencial escaneo activo. Además, la conexión puede requerir bastante tiempo por lo que, de nuevo, se recomienda utilizar una hebra auxiliar.

En esta práctica vamos a permitir al usuario seleccionar uno de los dispositivos emparejados en la lista, y lanzaremos una hebra que intentará conectarse a él. Si en el otro extremo se ha lanzado la práctica anterior (o ésta, pues compartirán el código) se realizará la conexión.

1. Haz una copia de la práctica anterior.
2. Crea una nueva clase ClientThread que herede de Thread. En el constructor deberá recibir el BluetoothDevice al que se intentará conectar. Cuando se lance, creará el socket, intentará conectarse, y mostrará un mensaje cuando lo haga. Crea también un atributo de esa clase.

```
private class ClientThread extends Thread {
private BluetoothDevice _device;
private BluetoothSocket _socket;
public ClientThread(BluetoothDevice device) {
```

```
_device = device;
}
public void run() {
try {
_socket = _device.
createRfcommSocketToServiceRecord(
SERVICE_UUID);
} catch (IOException e) {
escribe("ERROR: no pude obtener el socket");
_socket = null;
return;
}
bluetooth.cancelDiscovery();
try {
_socket.connect();
} catch (IOException connectException) {
escribe("ERROR: no conseguí conexión");
try {
_socket.close();
} catch (IOException closeException) { }
_socket = null;
return;
}
escribe("¡¡CONECTADOS!!");
try {
_socket.close();
} catch (IOException e) {
}
_socket = null;
}
// Genera un toast con el mensaje.
private void escribe(final String str) {
runOnUiThread(new Runnable() {
@Override
public void run() {
Toast.makeText(MainActivity.this, str,
Toast.LENGTH_LONG).show();
}
});
}
public void cancel() {
try {
_socket.close();
} catch (IOException e) { }
}
} // ClientThread
```

3. En el método updateONBluetoothUI() añade a la lista un listener para detectar la pulsación sobre la lista. Cuando lo hagas, lanza una de esas hebras sobre el dispositivo seleccionado.

```
private void updateONBluetoothUI() {
...
listaDispositivos.setOnItemClickListener(
new AdapterView.OnItemClickListener() {
```

```
@Override
public void onItemClick(AdapterView<?> parent,
View view, int position, long id) {
clientThread = new ClientThread(aa.getItem(position));
clientThread.start();
}
});
...
}
```

4. En el método onDestroy() cancela la ejecución de la hebra del cliente si existe.

```
if (clientThread != null)
clientThread.cancel();
```

Notas bibliográficas

Curso: IFC02CM15 - Programación avanzada de dispositivos móviles - Julio 2015
Pedro Pablo Gómez Martín
http://developer.android.com/guide/topics/connectivity/bluetooth.html
http://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html
http://developer.android.com/reference/android/bluetooth/BluetoothDevice.html
Bluetooth Assigned Numbers: https://www.bluetooth.org/en-us/specification/assigned-numbers