**Programação Bluetooth no Android**

[26 de May de 2015](http://dragaosemchama.com.br/2015/05/programacao-bluetooth-no-android/)  [C. David](http://dragaosemchama.com.br/author/david/)  [Android](http://dragaosemchama.com.br/category/android/?lang=pt_br), [Programação](http://dragaosemchama.com.br/category/programacao/?lang=pt_br)

Uma das coisas mais excitantes na programação de computadores e dispositivos móveis é a possibilidade de comunicação sem fio. É quase mágico quando um clique em um botão no seu celular gera uma ação em outro dispositivo. Dá uma sensação de controle, poder criar uma aplicação que envia comandos à distância para outro celular, notebook ou componentes eletrônicos feitos por você mesmo.

Neste post, vamos aprender a utilizar o controlador Bluetooth de um dispositivo Android. Ao final, seremos capazes de compartilhar dados entre dispositivos Android. No processo, vamos escrever um pequeno aplicativo capaz de buscar dispositivos Bluetooth, conectar-se e trocar mensagens.

1. Ativando o adaptador Bluetooth;
2. Buscando por dispositivos Bluetooth pareados;
3. Descobrindo dispositivos Bluetooth nas proximidades;
4. Conectando dois dispositivos;
5. Transferência de dados bidirecional.

**Alguns conceitos sobre Bluetooth**

É sempre interessante deixar claro o que é quê antes de aprender algo. Então reunimos alguns conceitos que você deve entender antes de começar a brincar com seu dispositivo Bluetooth. Claro, não vamos demorar muito nesses conceitos, você tem todo o tempo do mundo e a Internet, caso queira entender tudo sobre Bluetooth. Rapidamente falando, Bluetooth é um tecnologia de comunicação baseada em ondas eletromagnéticas na faixa de frequência em torno de 2.4 GHz. Agora, alguns conceitos que nos serão úteis:

**BluetoothAdapter**: No Android, é classe que representa o dispositivo Bluetooth local. Esta classe contem métodos que realizam ações fundamentais como descoberta de dispositivos, criar sockets Bluetooth.

**Pareamento**: No protocolo Bluetooth, o pareamento é a maneira pela qual dois dispositivos se tornam conhecidos um para o outro. Os dispositivos só poderão se comunicar se estiverem pareados.

**Descoberta**: Uma maneira de saber quais dispositivos Bluetooth existem nas proximidades. Se um dispositivo não está pareado ao dispositivo com o qual deseja se comunicar, é necessário realizar a descoberta antes de fazer o pareamento.

**Visibilidade**: Seu dispositivo só será visível a outros dispositivos se você habilitar essa opção. Então, durante a descoberta, apenas os dispositivos vísiveis serão encontrados.

**Conexão**: Quando dois dispositivos já estão pareados, a realização de uma conexão cria um canal de comunicação bidirecional entre os aparelhos.

**Sockets**: Estrutura que representa a conexão realizada entre os dois dispositivos. Através do socket adquirido na conexão, é possível obter as streams de transmissão e recepção de dados.

**1. Ativando o adaptador Bluetooth**

**1.1 Obtendo permissão**

Para utilizar o Bluetooth no Android, é necessário declarar que o aplicativo utiliza essa funcionalidade. Isso é feito ao configurar as permissões do aplicativo no arquivo *AndroidManifest.xml*, que pode ser encontrado na pasta *manifests*. Abaixo você pode ver como ficará o manifesto com a adição das duas permissões Bluetooth. Na linha 5, a permissão BLUETOOTH declara que queremos utilizar o adaptador para iniciar conexões e transferir dados. A segunda permissão, BLUETOOTH\_ADMIN, na linha 6, declara que o aplicativo poderá ter acesso às configurações do adaptador Bluetooth e também poderá realizar buscas por dispositivos próximos ainda não pareados.

AndroidManifest.xml



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | <?xml version="1.0" encoding="utf-8"?>  <manifest xmlns:android="http://schemas.android.com/apk/res/android"      package="com.project.dragaosemchama.superbluetooth" >        <uses-permission android:name="android.permission.BLUETOOTH" />      <uses-permission android:name="android.permission.BLUETOOTH\_ADMIN" />        <application          android:allowBackup="true"          android:icon="@drawable/ic\_launcher"          android:label="@string/app\_name"          android:theme="@style/AppTheme" >          <activity              android:name=".MainBluetoothActivity"              android:label="@string/app\_name" >              <intent-filter>                  <action android:name="android.intent.action.MAIN" />                    <category android:name="android.intent.category.LAUNCHER" />              </intent-filter>          </activity>      </application>    </manifest> |

**1.2 Garantindo que o adaptador Bluetooth funciona**

Agora que nosso aplicativo possui permissão, podemos iniciar as experiências.

Vamos criar um objeto [TextView](http://developer.android.com/reference/android/widget/TextView.html) chamado statusMessage, que nos permitirá imprimir mensagens na tela do aplicativo Android. Para isso, basta adicionar este widget ao arquivo *activity\_main\_bluetooth.xml*:

Adição de um TextView ao layout.



|  |  |
| --- | --- |
| 1  2  3  4  5 | <TextView      android:id="@+id/statusMessage"      android:text="@string/hello\_world"      android:layout\_width="wrap\_content"      android:layout\_height="wrap\_content" /> |

Já no arquivo *MainBluetoothActivity.java*, você pode definir o objeto [TextView](http://developer.android.com/reference/android/widget/TextView.html) dentro da definição de classe da atividade principal. Aproveite e adicione essas três linhas que definem as variáveis estáticas ENABLE\_BLUETOOTH, SELECT\_PAIRED\_DEVICE e SELECT\_DISCOVERED\_DEVICE. Esses valores serão utilizados durante o processo de habilitação do Bluetooth, seleção de um dispositivo pareado ou descoberto. Confie…

Objeto TextView declarado como variável da classe MainBluetoothActivity.

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | ...    public class MainBluetoothActivity extends ActionBarActivity {        public static int ENABLE\_BLUETOOTH = 1;      public static int SELECT\_PAIRED\_DEVICE = 2;      public static int SELECT\_DISCOVERED\_DEVICE = 3;        static TextView statusMessage;        ... |

E vincule statusMessage ao widget previamente definido, dentro do método onCreate():

Instanciação de uma TextView no método onCreate().

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | ...    *@Override*  protected void onCreate(Bundle savedInstanceState) {      super.onCreate(savedInstanceState);      setContentView(R.layout.activity\_main\_bluetooth);        statusMessage = (TextView) findViewById(R.id.statusMessage);        ... |

Com isso, teremos um [TextView](http://developer.android.com/reference/android/widget/TextView.html) pronto para exibir mensagens de status e poderemos saber o que está acontecendo nas entranhas do código.

Antes de tentar buscar ou conectar a outros dispositivos, é importante garantir que o aplicativo está sendo executado em um aparelho que suporta a funcionalidade Bluetooth e cujo hardware está em pleno funcionamento. Isso pode ser feito verificando se existe um [BluetoothAdapter](http://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html) padrão para o dispositivo.

Verificando se o adaptador Bluetooth existe e está em pleno funcionamento.

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | BluetoothAdapter btAdapter = BluetoothAdapter.getDefaultAdapter();  if (btAdapter == null) {      statusMessage.setText("Que pena! Hardware Bluetooth não está funcionando :(");      } else {      statusMessage.setText("Ótimo! Hardware Bluetooth está funcionando :)");  } |

**1.3 Ativando Bluetooth com permissão do usuário**

Existem duas maneiras de ativar o adaptador Bluetooth. A documentação online do Android recomenda solicitar a ativação ao sistema, que automaticamente solicitará ao usuário uma permissão. O código a seguir inicialmente verifica se o adaptador Bluetooth está ativado. Se não, envia uma solicitação ao sistema na forma de um [Intent](http://developer.android.com/reference/android/content/Intent.html). Caso o adaptador já esteja ativado, você pode ficar tranquilo.

Solicitando ativação do adaptador Bluetooth ao usuário.

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | if(!btAdapter.isEnabled()) {      Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION\_REQUEST\_ENABLE);      startActivityForResult(enableBtIntent, ENABLE\_BLUETOOTH);      statusMessage.setText("Solicitando ativação do Bluetooth...");  } else {      statusMessage.setText("Bluetooth já ativado :)");  } |

Vejamos o que está acontecendo. A classe [Intent](http://developer.android.com/reference/android/content/Intent.html) é uma abstração que representa uma operação a ser realizada. Aqui podemos imaginar que o objeto [Intent](http://developer.android.com/reference/android/content/Intent.html) que criamos, enableBtIntent, representa a operação de ativar o adaptador Bluetooth. Este [Intent](http://developer.android.com/reference/android/content/Intent.html) é utilizado para iniciar uma nova [Activity](http://developer.android.com/reference/android/app/Activity.html), esperando uma resposta, como fazemos na linha 3 acima. A nova [Activity](http://developer.android.com/reference/android/app/Activity.html) apresenta ao usuário a escolha de ativar ou não a funcionalidade Bluetooth. Se isso tudo ainda ficou meio obscuro e você tiver interesse, essas páginas provavelmente vão fornecer uma explicação bem melhor:

Legal. Mas com isso, apenas poderemos solicitar a ativação do Bluetooth e continuar a execução do código. O problema é que, se fizermos apenas isso, as linhas seguintes do código serão executadas sem sabermos se o usuário permitiu ou não a ativação. Lembre-se que a execução do código se dá muitas vezes mais rápido do que o usuário será capaz de responder à solicitação. Então teremos que esperar que o usuário responda à solicitação, para então decidir o que fazer. Para isso, vamos adicionar o método[onActivityResult()](http://developer.android.com/reference/android/app/Activity.html#onActivityResult(int,%20int, android.content.Intent)) à classe MainBluetoothActivity.

Capturando a resposta do usuário sobre a ativação do adaptador Bluetooth.

Java

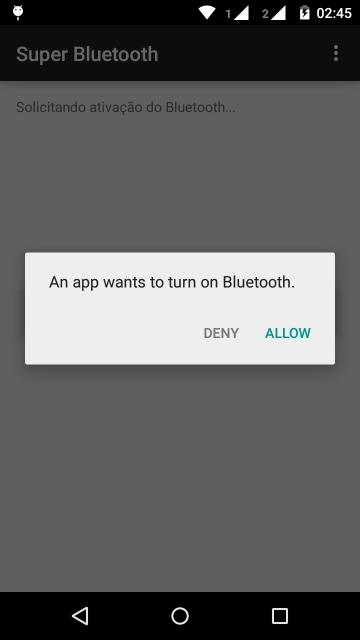


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | *@Override*  protected void onActivityResult(int requestCode, int resultCode, Intent data) {        if(requestCode == ENABLE\_BLUETOOTH) {          if(resultCode == RESULT\_OK) {              statusMessage.setText("Bluetooth ativado :D");          }          else {              statusMessage.setText("Bluetooth não ativado :(");          }      }  } |

Este método sempre será executado quando o algoritmo retornar à MainBluetoothActivity, após a execução do método [startActivityForResult()](https://developer.android.com/reference/android/app/Activity.html#startActivityForResult(android.content.Intent,%20int)) chamado internamente a MainBluetoothActivity. Por isso podemos capturar o momento em que o usuário fez sua escolha e tomar as devidas providências. É importante entender o significado de algumas das variáveis, para esse caso específico:

requestCode: funciona como um identificador sobre qual [Activity](http://developer.android.com/reference/android/app/Activity.html) está retornando um resultado. Esse valor é definido na chamada do método startActivityForResult(). Veja o valor ENABLE\_BLUETOOTH. É um inteiro. Com isso, podemos saber qual [Activity](http://developer.android.com/reference/android/app/Activity.html) está retornando o resultado.

resultCode: traz a informação sobre a decisão do usuário. RESULT\_OK, nesse caso, significa que o adaptador Bluetooth foi ativado, qualquer outro valor significa que o usuário negou a permissão ou que um erro ocorreu.

[](http://dragaosemchama.com.br/wp-content/uploads/2015/02/Screenshot_2015-02-02-02-45-06.png?93852b)Seja gentil, pergunte ao usuário se é isso o que ele quer, se é isso o que ele deseja.

**1.4 Ativando o adaptador Bluetooth automaticamente**

Certo, mas agora vamos supor que você não está feliz em pedir permissão ao sistema e ao usuário para ativar o adaptador Bluetooth. Você quer que isso aconteça automaticamente. Existe uma maneira muito simples de fazer isso.

Ativando o adaptador Bluetooth automaticamente.

Java



|  |  |
| --- | --- |
| 1 | btAdapter.enable(); |

Pronto, com isso, se não houver erros misteriosos, o adaptador Bluetooth estará ativado. Mas preste atenção, a documentação recomenda fortemente, ou exige, que o Bluetooth não seja ativado sem o consentimento do usuário. O método [enable()](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#enable()) é utilizado quando o aplicativo fornece uma interface gráfica em que o usuário pode escolher ativar ou não a funcionalidade. De qualquer forma, é possível ativar o Bluetooth dentro de seu código mesmo sem a permissão explícita do usuário com o código acima. Use quando estiver se sentindo rebelde ou quando sua aplicação exigir isso. A mesma coisa vale para o método[disable()](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#disable()), que desativa o adaptador.

Desativando o adaptador Bluetooth automaticamente.

Java



|  |  |
| --- | --- |
| 1 | btAdapter.disable(); |

**2. Buscando por dispositivos Bluetooth pareados**

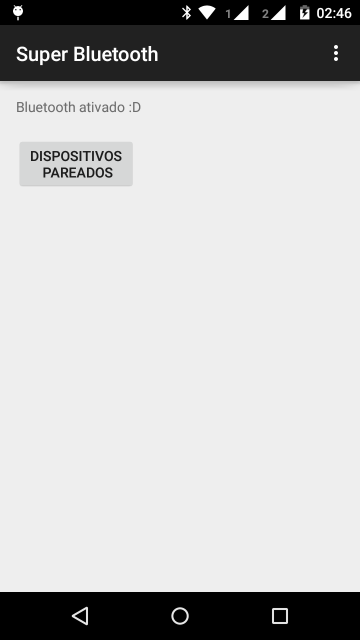
Para iniciar uma conexão com outro dispositivo, tudo o que precisamos é o seu endereço MAC. A não ser que sua aplicação seja bem específica e você já saiba o endereço ao qual o app deve se conectar, o app terá de obter o endereço MAC de alguma maneira. Que tal obter uma lista com todos os dispositivos conhecidos? O gerenciador Bluetooth do Android guarda na memória uma lista com vários dispositivos Bluetooth já pareados. Assim, é possível que o endereço do dispositivo com o qual o usuário deseja conectar o app já seja conhecido. Veremos como obter os dispositivos pareados, selecionar um deles e obter seu endereço MAC.

Vamos criar um botão que, quando pressionado, exibe uma lista contendo dispositivos pareados. Adicione o seguinte widget ao arquivo *activity\_main\_bluetooth.xml*:

Botão para visualizar dispositivos pareados



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <Button      android:layout\_width="wrap\_content"      android:layout\_height="wrap\_content"      android:text="Dispositivos\n pareados"      android:id="@+id/button\_PairedDevices"      android:layout\_below="@+id/statusMessage"      android:layout\_alignParentLeft="true"      android:layout\_alignParentStart="true"      android:layout\_marginTop="20dp"      android:onClick="searchPairedDevices"/> |

[](http://dragaosemchama.com.br/wp-content/uploads/2015/02/Screenshot_2015-02-02-02-46-09.png?93852b)Um botão ajuda a organizar as coisas. Não deixe tudo tão automático ou o usuário ficará triste.

Note que, como estamos usando um RelativeLayout, a posição do botão é configurada usando a posição do [TextView](http://developer.android.com/reference/android/widget/TextView.html) statusMessage, que criamos anteriormente. Adicionalmente, o botão já está configurado para chamar o método searchPairedDevices(), quando pressionado. Bem, esse método ainda não existe. Então vamos defini-lo. Adicione o método abaixo à classe MainBluetoothActivity:

Iniciando a Activity PairedDevices.

Java



|  |  |
| --- | --- |
| 1  2  3  4  5 | public void searchPairedDevices(View view) {        Intent searchPairedDevicesIntent = new Intent(this, PairedDevices.class);      startActivityForResult(searchPairedDevicesIntent, SELECT\_PAIRED\_DEVICE);  } |

O código acima define um [Intent](http://developer.android.com/reference/android/content/Intent.html) para iniciar uma nova [Activity](http://developer.android.com/reference/android/app/Activity.html), o que significa que faremos uma transição de uma tela do aplicativo para outra. Da mesma forma, que a atividade inicial, MainBluetoothActivity, devemos ter uma classe, subclasse de [Activity](http://developer.android.com/reference/android/app/Activity.html), para especificar o que a nova tela vai exibir. Vamos chamar essa nova classe de PairedDevices. Agora veja que na chamada ao método startActivityForResult(), usamos o SELECT\_PAIRED\_DEVICE como requestCode, para diferenciar da chamada que usamos para solicitar a ativação do adaptador Bluetooth.

Faça uma pausa, descanse… Quando voltar, vá até a pasta do seu app -> src -> main -> res -> layout. Clique com o botão direito na pasta layout e selecione New -> Android resource file. Isso adiciona um novo arquivo de layout ao projeto. Escolhemos *text\_header.xml*. Apenas misteriosamente copie e cole o seguinte código. Não se preocupe, não é vírus. Isso nos servirá no futuro, para melhorar o visual do app:

Layout misterioso



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | <?xml version="1.0" encoding="utf-8"?>  <LinearLayout      xmlns:android="http://schemas.android.com/apk/res/android"      android:orientation="vertical"      android:layout\_width="match\_parent"      android:layout\_height="match\_parent">        <TextView          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content"          android:textAppearance="?android:attr/textAppearanceLarge"          android:text="..."          android:id="@+id/textView"          android:layout\_gravity="center\_horizontal" />  </LinearLayout> |

Seguindo…

Normalmente, também é necessário um arquivo xml para o layout da nova [Activity](http://developer.android.com/reference/android/app/Activity.html), mas, nesse caso, faremos de uma maneira em que isso não é uma necessidade. Vamos criar uma subclasse de [ListActivity](https://developer.android.com/reference/android/app/ListActivity.html), que já fornece um layout padrão de lista. Para isso, crie uma nova classe Java chamada PairedDevices, no mesmo diretório da classe MainBluetoothActivity. Use o seguinte código para essa nova classe:

Activity PairedDevices: exibe uma lista selecionável de dispositivos pareados.

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96 | package com.project.dragaosemchama.superbluetooth;    import android.bluetooth.BluetoothAdapter;  import android.bluetooth.BluetoothDevice;  import android.content.Intent;  import android.os.Bundle;  import android.view.LayoutInflater;  import android.view.Menu;  import android.view.MenuItem;  import android.app.ListActivity;  import android.view.View;  import android.widget.ArrayAdapter;  import android.widget.ListView;  import android.widget.TextView;    import java.util.Set;    public class PairedDevices extends ListActivity {    *@Override*      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);            /\*  Esse trecho não é essencial, mas dá um melhor visual à lista.              Adiciona um título à lista de dispositivos pareados utilizando          o layout text\_header.xml.          \*/          ListView lv = getListView();          LayoutInflater inflater = getLayoutInflater();          View header = inflater.inflate(R.layout.text\_header, lv, false);          ((TextView) header.findViewById(R.id.textView)).setText("\nDispositivos pareados\n");          lv.addHeaderView(header, null, false);            /\*  Usa o adaptador Bluetooth para obter uma lista de dispositivos pareados.           \*/          BluetoothAdapter btAdapter = BluetoothAdapter.getDefaultAdapter();          Set<BluetoothDevice> pairedDevices = btAdapter.getBondedDevices();            /\*  Cria um modelo para a lista e o adiciona à tela.              Se houver dispositivos pareados, adiciona cada um à lista.           \*/          ArrayAdapter<String> adapter = new ArrayAdapter<>(this, android.R.layout.simple\_list\_item\_1);          setListAdapter(adapter);          if (pairedDevices.size() > 0) {              for (BluetoothDevice device : pairedDevices) {                  adapter.add(device.getName() + "\n" + device.getAddress());              }          }      }        /\*  Este método é executado quando o usuário seleciona um elemento da lista.       \*/  *@Override*      protected void onListItemClick(ListView l, View v, int position, long id) {            /\*  Extrai nome e endereço a partir do conteúdo do elemento selecionado.              Nota: position-1 é utilizado pois adicionamos um título à lista e o          valor de position recebido pelo método é deslocado em uma unidade.           \*/          String item = (String) getListAdapter().getItem(position-1);          String devName = item.substring(0, item.indexOf("\n"));          String devAddress = item.substring(item.indexOf("\n")+1, item.length());            /\*  Utiliza um Intent para encapsular as informações de nome e endereço.              Informa à Activity principal que tudo foi um sucesso!              Finaliza e retorna à Activity principal.           \*/          Intent returnIntent = new Intent();          returnIntent.putExtra("btDevName", devName);          returnIntent.putExtra("btDevAddress", devAddress);          setResult(RESULT\_OK, returnIntent);          finish();      }    *@Override*      public boolean onCreateOptionsMenu(Menu menu) {          // Inflate the menu; this adds items to the action bar if it is present.          getMenuInflater().inflate(R.menu.menu\_paired\_devices, menu);          return true;      }    *@Override*      public boolean onOptionsItemSelected(MenuItem item) {          // Handle action bar item clicks here. The action bar will          // automatically handle clicks on the Home/Up button, so long          // as you specify a parent activity in AndroidManifest.xml.          int id = item.getItemId();            //noinspection SimplifiableIfStatement          if (id == R.id.action\_settings) {              return true;          }            return super.onOptionsItemSelected(item);      }  } |

Sempre que adicionar uma nova [Activity](http://developer.android.com/reference/android/app/Activity.html) ao app, você deve declará-la no manifesto. Então vá ao arquivo*AndroidManifest.xml* e adicione o seguinte código dentro da tag <application>:

Declarando uma Activity no manifesto.



|  |  |
| --- | --- |
| 1  2  3  4  5 | <activity      android:name=".PairedDevices"      android:label="Paired Devices"      android:parentActivityName=".MainBluetoothActivity" >  </activity> |

Muito bem, temos uma [Activity](http://developer.android.com/reference/android/app/Activity.html) para visualizar os dispositivos pareados. Acredito que o código está bem comentado, caso haja seções importantes obscuras, não fique com vergonha de perguntar.

Vejamos uma breve descrição sobre o que acontece nesta [Activity](http://developer.android.com/reference/android/app/Activity.html).

Usando um [BluetoothAdapter](http://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html), chamamos o método [getBondedDevices()](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#getBondedDevices()) para obter um conjunto de objetos BluetoothDevice. Cada BluetoothDevice representa um dispositivo pareado.

Extraímos informação de nome e endereço de cada [BluetoothDevice](https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html) e populamos uma lista selecionável.

Definimos instruções a serem realizadas quando o usuário selecionar um elemento da lista. Quando isso acontecer, o método [onListItemClick()](https://developer.android.com/reference/android/app/ListActivity.html#onListItemClick(android.widget.ListView,%20android.view.View, int, long)) será invocado.

No método onListItemClick(), extraímos o nome e endereço do dispositivo selecionado e os transferimos de volta a MainBluetoothActivity, empacotados em um [Intent](http://developer.android.com/reference/android/content/Intent.html) com o método putExtra(). Adicionamos um pequeno relatório sobre o sucesso dessas ações com o método setResult().

Com isso, teremos na tela inicial um botão que quando clicado exibe a lista de dispositivos pareados. Quando um item da lista é selecionado, obtemos o nome e endereço do dispositivo Bluetooth. Opa, ainda falta uma coisa… Precisamos preparar a classe MainBluetoothActivity para lidar com a chegada dessas informações. Sabemos que, quando iniciamos a [Activity](http://developer.android.com/reference/android/app/Activity.html) PairedDevices usando startActivityForResult() e então retornamos a MainBluetoothActivity, o método [onActivityResult()](http://developer.android.com/reference/android/app/Activity.html#onActivityResult(int,%20int, android.content.Intent)) é executado. Então vamos adicionar alguma cláusulas a esse método…

Lidando com nome e endereço selecionados dentre os dispositivos pareados.

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | *@Override*  protected void onActivityResult(int requestCode, int resultCode, Intent data) {        if(requestCode == ENABLE\_BLUETOOTH) {          if(resultCode == RESULT\_OK) {              statusMessage.setText("Bluetooth ativado :D");          }          else {              statusMessage.setText("Bluetooth não ativado :(");          }      }      else if(requestCode == SELECT\_PAIRED\_DEVICE) {          if(resultCode == RESULT\_OK) {              statusMessage.setText("Você selecionou " + data.getStringExtra("btDevName") + "\n"                                      + data.getStringExtra("btDevAddress"));          }          else {              statusMessage.setText("Nenhum dispositivo selecionado :(");          }      }  } |

As novas linhas são muito semelhantes ao que já havíamos feito para ativar o Bluetooth, então acho que podemos pular maiores explicações. Observe nas linhas 14 e 15 como obter a String que anteriormente empacotamos no [Intent](http://developer.android.com/reference/android/content/Intent.html) com o método putExtra(). Basta chamar getStringExtra() e utilizar como parâmetro a mesma *key* usada ao empacotar.

Ótimo, com isso podemos obter o endereço MAC de dispositivos Bluetooth já pareados. Mas o que acontece se o que queremos é encontrar um novo dispositivo? Teremos que descobri-lo!

**3. Descobrindo dispositivos Bluetooth nas proximidades**

**3.1 Descobrindo dispositivos Bluetooth**

Iniciar uma descoberta de dispositivos Bluetooth é muito simples. Basta chamar [startDiscovery()](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#startDiscovery()). O problema é ver os dispositivos encontrados. Isso ocorre porque o processo de descoberta ocorre de forma assíncrona, em uma thread separada da principal. Para saber que um dispositivo foi descoberto, precisamos esperar um aviso do sistema operacional, que vem na forma de um broadcast. Para capturar esse aviso, vamos criar um objeto do tipo [BroadcastReceiver](https://developer.android.com/reference/android/content/BroadcastReceiver.html), que será responsável pela captura. Então vamos sobrescrever seu método [onReceive()](https://developer.android.com/reference/android/content/BroadcastReceiver.html#onReceive(android.content.Context,%20android.content.Intent)) para que sejam executadas as instruções que queremos.

Antes de mergulhar nos detalhes do descobrimento de dispositivos Bluetooth, vamos criar um botão para podermos clicar quando desejarmos executar essa ação. O código abaixo define um botão à direita do que já criamos e que chama o método discoverDevices() quando pressionado.

Botão para ativar a descoberta de dispositivos Bluetooth.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <Button      android:layout\_width="wrap\_content"      android:layout\_height="wrap\_content"      android:text="Iniciar descoberta\n de dispositivos"      android:id="@+id/button\_DiscoveredDevices"      android:layout\_alignTop="@+id/button\_PairedDevices"      android:layout\_toRightOf="@+id/button\_PairedDevices"      android:layout\_toEndOf="@+id/button\_PairedDevices"      android:layout\_marginLeft="5dp"      android:onClick="discoverDevices"/> |

[](http://dragaosemchama.com.br/wp-content/uploads/2015/02/Screenshot_2015-02-03-20-30-02.png?93852b)É uma boa ideia utilizar o editor de design visual para escolher a posição de um widget e o editor de texto para ajeitar os detalhes.

discoverDevices() ainda não existe, então adicione-o à classe MainBluetoothActivity. É a terceira vez que fazemos algo parecido, iniciar uma [Activity](http://developer.android.com/reference/android/app/Activity.html) para obter um resultado. Lembre-se de usar um requestCode diferente para cada tipo de chamada. Aqui, usamos o valor SELECT\_DISCOVERED\_DEVICE.

Iniciando a Activity DiscoveredDevices.



|  |  |
| --- | --- |
| 1  2  3  4  5 | public void discoverDevices(View view) {        Intent searchPairedDevicesIntent = new Intent(this, DiscoveredDevices.class);      startActivityForResult(searchPairedDevicesIntent, SELECT\_DISCOVERED\_DEVICE);  } |

Façamos o seguinte: que tal utilizarmos a mesma estrutura de lista que usamos para ver os dispositivos pareados? Vai facilitar bastante a minha vida. Espero que também a sua. Então, go! Crie uma classe, da mesma maneira que já fizemos na seção anterior. Aqui, usei o nome DiscoveredDevices para a nova classe. Agora use sua ferramenta de busca favorita e faça o código para descoberta, visualização e seleção de um novo dispositivo sozinho. Brincadeira! Ok, apenas use o seguinte código:

Activity DiscoveredDevices: exibe uma lista de dispositivos descobertos.

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135 | package com.project.dragaosemchama.superbluetooth;    import android.app.ListActivity;  import android.bluetooth.BluetoothAdapter;  import android.bluetooth.BluetoothDevice;  import android.content.BroadcastReceiver;  import android.content.Context;  import android.content.Intent;  import android.content.IntentFilter;  import android.os.Bundle;  import android.view.LayoutInflater;  import android.view.Menu;  import android.view.MenuItem;  import android.view.View;  import android.widget.ArrayAdapter;  import android.widget.ListView;  import android.widget.TextView;    public class DiscoveredDevices extends ListActivity {        /\*  Um adaptador para conter os elementos da lista de dispositivos descobertos.       \*/      ArrayAdapter<String> arrayAdapter;    *@Override*      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);            /\*  Esse trecho não é essencial, mas dá um melhor visual à lista.              Adiciona um título à lista de dispositivos pareados utilizando          o layout text\_header.xml.          \*/          ListView lv = getListView();          LayoutInflater inflater = getLayoutInflater();          View header = inflater.inflate(R.layout.text\_header, lv, false);          ((TextView) header.findViewById(R.id.textView)).setText("\nDispositivos próximos\n");          lv.addHeaderView(header, null, false);            /\*  Cria um modelo para a lista e o adiciona à tela.              Para adicionar um elemento à lista, usa-se arrayAdapter.add().           \*/          arrayAdapter = new ArrayAdapter<>(this, android.R.layout.simple\_list\_item\_1);          setListAdapter(arrayAdapter);            /\*  Usa o adaptador Bluetooth padrão para iniciar o processo de descoberta.           \*/          BluetoothAdapter btAdapter = BluetoothAdapter.getDefaultAdapter();          btAdapter.startDiscovery();            /\*  Cria um filtro que captura o momento em que um dispositivo é descoberto.              Registra o filtro e define um receptor para o evento de descoberta.           \*/          IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION\_FOUND);          registerReceiver(receiver, filter);      }        /\*  Este método é executado quando o usuário seleciona um elemento da lista.       \*/  *@Override*      protected void onListItemClick(ListView l, View v, int position, long id) {            /\*  Extrai nome e endereço a partir do conteúdo do elemento selecionado.              Nota: position-1 é utilizado pois adicionamos um título à lista e o          valor de position recebido pelo método é deslocado em uma unidade.           \*/          String item = (String) getListAdapter().getItem(position-1);          String devName = item.substring(0, item.indexOf("\n"));          String devAddress = item.substring(item.indexOf("\n")+1, item.length());            /\*  Utiliza um Intent para encapsular as informações de nome e endereço.              Informa à Activity principal que tudo foi um sucesso!              Finaliza e retorna à Activity principal.           \*/          Intent returnIntent = new Intent();          returnIntent.putExtra("btDevName", devName);          returnIntent.putExtra("btDevAddress", devAddress);          setResult(RESULT\_OK, returnIntent);          finish();      }        /\*  Define um receptor para o evento de descoberta de dispositivo.       \*/      private final BroadcastReceiver receiver = new BroadcastReceiver() {            /\*  Este método é executado sempre que um novo dispositivo for descoberto.           \*/          public void onReceive(Context context, Intent intent) {                /\*  Obtem o Intent que gerou a ação.                  Verifica se a ação corresponde à descoberta de um novo dispositivo.                  Obtem um objeto que representa o dispositivo Bluetooth descoberto.                  Exibe seu nome e endereço na lista.               \*/              String action = intent.getAction();              if (BluetoothDevice.ACTION\_FOUND.equals(action)) {                  BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA\_DEVICE);                  arrayAdapter.add(device.getName() + "\n" + device.getAddress());              }          }      };        /\*  Executado quando a Activity é finalizada.       \*/  *@Override*      protected void onDestroy() {            super.onDestroy();            /\*  Remove o filtro de descoberta de dispositivos do registro.           \*/          unregisterReceiver(receiver);      }    *@Override*      public boolean onCreateOptionsMenu(Menu menu) {          // Inflate the menu; this adds items to the action bar if it is present.          getMenuInflater().inflate(R.menu.menu\_paired\_devices, menu);          return true;      }    *@Override*      public boolean onOptionsItemSelected(MenuItem item) {          // Handle action bar item clicks here. The action bar will          // automatically handle clicks on the Home/Up button, so long          // as you specify a parent activity in AndroidManifest.xml.          int id = item.getItemId();            //noinspection SimplifiableIfStatement          if (id == R.id.action\_settings) {              return true;          }            return super.onOptionsItemSelected(item);      }  } |

Não esqueça de declarar esta nova [Activity](http://developer.android.com/reference/android/app/Activity.html) ao manifesto, novamente dentro da tag <application>.

Declarando a Activity Discovered Devices no manifesto.



|  |  |
| --- | --- |
| 1  2  3  4  5 | <activity      android:name=".DiscoveredDevices"      android:label="Discovered Devices"      android:parentActivityName=".MainBluetoothActivity" >  </activity> |

Vamos a um resumo sobre o que está acontecendo na [Activity](http://developer.android.com/reference/android/app/Activity.html) DiscoveredDevices.

Criamos uma lista e a exibimos na tela. Conforme os dispositivos forem sendo descobertos, serão adicionados à lista.

Solicitamos ao adaptador Bluetooth que inicie o processo de descoberta com startDiscovery(), que dura aproximadamente 12 segundos. Apenas os dispositivos descobertos nesse período serão exibidos.

Criamos um filtro para capturar o momento em que um dispositivo é descoberto. Depois disso, registramos o filtro para o recebimento de broadcasts.

O truque! Definimos um receptor de broadcasts. A classe BroadcastReceiver fornece um método chamado onReceive(), que é executado quando o sistema operacional Android transmite um broadcast. Entre seus argumentos está um [Intent](http://developer.android.com/reference/android/content/Intent.html), que vem com algumas informações, por exemplo, sobre o que causou o broadcast. Extraimos essa informação e testamos se a causa é a descoberta de um dispositivo Bluetooth. Caso positivo, extraimos os dados do dispositivo e exibimos na lista. Voila.

Definimos instruções a serem realizadas quando o usuário selecionar um elemento da lista. Quando isso acontecer, o método onListItemClick() será invocado.

No método onListItemClick(), extraímos o nome e endereço do dispositivo selecionado e os transferimos de volta a MainBluetoothActivity, empacotados em um [Intent](http://developer.android.com/reference/android/content/Intent.html) com o método putExtra(). Adicionamos um pequeno relatório sobre o sucesso dessas ações com o método setResult().

Sobrescrevemos o método onDestroy() de nossa [Activity](http://developer.android.com/reference/android/app/Activity.html). onDestroy() é sempre executado quando a[Activity](http://developer.android.com/reference/android/app/Activity.html) é finalizada e foi implementado para liberar recursos alocados pela [Activity](http://developer.android.com/reference/android/app/Activity.html). Usamos unregisterReceiver() para remover o filtro de broadcasts. É boa prática nunca esquecer de finalizar corretamente um BroadcastReceiver, caso contrário, seu aplicativo poderá apresentar vazamentos de dados.

Precisamos de apenas mais um ajuste. Quando obtivermos os dados do dispositivo selecionado pelo usuário, retornaremos à MainBluetoothActivity e ao método [onActivityResult()](http://developer.android.com/reference/android/app/Activity.html#onActivityResult(int,%20int, android.content.Intent)). Mas teremos que decidir o que fazer com os dados. Vamos aproveitar o que já foi feito com os dispositivos pareados. Apenas mostrar o nome e o endereço, por enquanto. Vá até o código de [onActivityResult()](http://developer.android.com/reference/android/app/Activity.html#onActivityResult(int,%20int, android.content.Intent)) na classe MainBluetoothActivity e altere a linha destacada a seguir:

Adicionando uma ação após a seleção de um dispositivo descoberto.

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | *@Override*  protected void onActivityResult(int requestCode, int resultCode, Intent data) {        if(requestCode == ENABLE\_BLUETOOTH) {          if(resultCode == RESULT\_OK) {              statusMessage.setText("Bluetooth ativado :D");          }          else {              statusMessage.setText("Bluetooth não ativado :(");          }       }      else if(requestCode == SELECT\_PAIRED\_DEVICE || requestCode == SELECT\_DISCOVERED\_DEVICE) {          if(resultCode == RESULT\_OK) {              statusMessage.setText("Você selecionou " + data.getStringExtra("btDevName") + "\n"                                          + data.getStringExtra("btDevAddress"));            }          else {              statusMessage.setText("Nenhum dispositivo selecionado :(");          }      }  } |

Isso faz com que o mesmo código seja executado quando o usuário selecionar um dispositivo já pareado ou recém-descoberto.

**3.2 Habilitando visibilidade do dispositivo Bluetooth**

Só será possível descobrir um dispositivo Bluetooth se ele estiver visível. Então vamos adicionar à interface gráfica um botão. Faremos com que o smartphone torne-se visível quando esse botão for pressionado. Adicione o seguinte widget ao arquivo *activity\_main\_bluetooth.xml:*

Adicionando um botão para habilitar visibilidade do dispositivo.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | <Button      android:layout\_width="wrap\_content"      android:layout\_height="wrap\_content"      android:text="Habilitar\nVisibilidade"      android:id="@+id/button\_Visibility"      android:layout\_below="@+id/button\_PairedDevices"      android:layout\_alignParentLeft="true"      android:layout\_alignParentStart="true"      android:onClick="enableVisibility" /> |

[](http://dragaosemchama.com.br/wp-content/uploads/2015/02/Screenshot_2015-02-05-00-42-05-e1423107801369.png?93852b)Não precisa se preocupar agora com a beleza do app.

 Legal. Agora basta adicionar o método enableVisibility() à classe MainBluetoothActivity:

Solicitando permissão para habilitar visibilidade.



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public void enableVisibility(View view) {        Intent discoverableIntent = new Intent(BluetoothAdapter.ACTION\_REQUEST\_DISCOVERABLE);      discoverableIntent.putExtra(BluetoothAdapter.EXTRA\_DISCOVERABLE\_DURATION, 30);      startActivity(discoverableIntent);  } |

O código acima solicita permissão ao usuário para que o dispositivo se torne visível por 30 segundos.  A duração da visibilidade pode ser alterada. Qualquer valor entre 1 e 3600 segundos é possível. O valor 0 significa que o dispositivo estará visível sempre.

Ufa… então somos capazes de obter o endereço MAC de dispositivos já pareados e de dispositivos visíveis nas proximidades. Lembre-se que para solicitar o início de uma conexão Bluetooth a outro dispositivo, precisamos apenas de seu endereço MAC. Agora estamos prontos. Sigam-me os bons!

**4. Conectando dois dispositivos**

Quando escrevemos um aplicativo que conecta-se via Bluetooth, há alguns detalhes a considerar.

Na conexão, temos um servidor e um cliente. Chamamos de servidor o dispositivo que cria um socket e espera pela conexão, ou seja, ele torna-se disponível para aceitar uma conexão. E chamamos de cliente aquele que solicita a conexão com o servidor. Quando programamos aplicações que atuam em rede, o código para iniciar uma conexão é diferente para o servidor e o cliente. Aqui, dois dispositivos são considerados conectados quando compartilham um objeto [BluetoothSocket](https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html). Vamos aprender a obter um BluetoothSocket tanto para o servidor quanto para o cliente.

Outro detalhe é o fato de que vários métodos utilizados para realizar a conexão ou para receber mensagens são capazes de bloquear o fluxo do código. Por exemplo, quando o servidor está esperando por uma conexão, nenhuma outra instrução poderá ser executada na sua thread. O algoritmo ficará empacado naquela linha até que alguém solicite o início de uma conexão. Isso significa que as ações de conexão e transmissão de mensagens deverão ocorrer em uma thread diferente da principal. Caso contrário, muito provavelmente o app dará belas e maravilhosas travadas, lindos crashes. Prepare-se para trabalhar com threads.

Em Java, para definirmos uma thread, criamos uma nova classe, subclasse de [Thread](https://developer.android.com/reference/java/lang/Thread.html), e escrevemos as instruções que devem ser executadas paralelamente dentro de seu método [run()](https://developer.android.com/reference/java/lang/Thread.html#run()). Veja só, a seguir temos o esqueleto dessa classe, que chamo de ConnectionThread. Ela contém código que permite que o app haja tanto como servidor quanto como cliente. Digo esqueleto porque contem apenas o código relacionado ao início da conexão. O gerenciamento da conexão, transmissão de mensagens é assunto da próxima seção. Então, você já sabe! Adicione essa classe ao seu projeto.

Nota: como esta classe não é uma [Activity](http://developer.android.com/reference/android/app/Activity.html), não é necessário modificar o manifesto.

Conectando seu app via Bluetooth em uma nova thread.

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158 | package com.project.dragaosemchama.superbluetooth;    import android.bluetooth.BluetoothAdapter;  import android.bluetooth.BluetoothDevice;  import android.bluetooth.BluetoothServerSocket;  import android.bluetooth.BluetoothSocket;  import android.os.Bundle;  import android.os.Message;    import java.io.IOException;  import java.util.UUID;    public class ConnectionThread extends Thread{        BluetoothSocket btSocket = null;      BluetoothServerSocket btServerSocket = null;      String btDevAddress = null;      String myUUID = "00001101-0000-1000-8000-00805F9B34FB";      boolean server;      boolean running = false;        /\*  Este construtor prepara o dispositivo para atuar como servidor.       \*/      public ConnectionThread() {            this.server = true;      }        /\*  Este construtor prepara o dispositivo para atuar como cliente.          Tem como argumento uma string contendo o endereço MAC do dispositivo      Bluetooth para o qual deve ser solicitada uma conexão.       \*/      public ConnectionThread(String btDevAddress) {            this.server = false;          this.btDevAddress = btDevAddress;      }        /\*  O método run() contem as instruções que serão efetivamente realizadas      em uma nova thread.       \*/      public void run() {            /\*  Anuncia que a thread está sendo executada.              Pega uma referência para o adaptador Bluetooth padrão.           \*/          this.running = true;          BluetoothAdapter btAdapter = BluetoothAdapter.getDefaultAdapter();            /\*  Determina que ações executar dependendo se a thread está configurada          para atuar como servidor ou cliente.           \*/          if(this.server) {                /\*  Servidor.               \*/              try {                    /\*  Cria um socket de servidor Bluetooth.                      O socket servidor será usado apenas para iniciar a conexão.                      Permanece em estado de espera até que algum cliente                  estabeleça uma conexão.                   \*/                  btServerSocket = btAdapter.listenUsingRfcommWithServiceRecord("Super Bluetooth", UUID.fromString(myUUID));                  btSocket = btServerSocket.accept();                    /\*  Se a conexão foi estabelecida corretamente, o socket                  servidor pode ser liberado.                   \*/                  if(btSocket != null) {                        btServerSocket.close();                  }                } catch (IOException e) {                    /\*  Caso ocorra alguma exceção, exibe o stack trace para debug.                      Envia um código para a Activity principal, informando que                  a conexão falhou.                   \*/                  e.printStackTrace();                  toMainActivity("---N".getBytes());              }              } else {                /\*  Cliente.               \*/              try {                    /\*  Obtem uma representação do dispositivo Bluetooth com                  endereço btDevAddress.                      Cria um socket Bluetooth.                   \*/                  BluetoothDevice btDevice = btAdapter.getRemoteDevice(btDevAddress);                  btSocket = btDevice.createRfcommSocketToServiceRecord(UUID.fromString(myUUID));                    /\*  Envia ao sistema um comando para cancelar qualquer processo                  de descoberta em execução.                   \*/                  btAdapter.cancelDiscovery();                    /\*  Solicita uma conexão ao dispositivo cujo endereço é                  btDevAddress.                      Permanece em estado de espera até que a conexão seja                  estabelecida.                   \*/                  if (btSocket != null)                      btSocket.connect();                } catch (IOException e) {                    /\*  Caso ocorra alguma exceção, exibe o stack trace para debug.                      Envia um código para a Activity principal, informando que                  a conexão falhou.                   \*/                  e.printStackTrace();                  toMainActivity("---N".getBytes());              }            }            /\*  Pronto, estamos conectados! Agora, só precisamos gerenciar a conexão.              ...          \*/        }        /\*  Utiliza um handler para enviar um byte array à Activity principal.          O byte array é encapsulado em um Bundle e posteriormente em uma Message      antes de ser enviado.       \*/      private void toMainActivity(byte[] data) {            Message message = new Message();          Bundle bundle = new Bundle();          bundle.putByteArray("data", data);          message.setData(bundle);          MainBluetoothActivity.handler.sendMessage(message);      }        /\*  Método utilizado pela Activity principal para encerrar a conexão       \*/      public void cancel() {            try {                running = false;              btServerSocket.close();              btSocket.close();            } catch (IOException e) {              e.printStackTrace();          }          running = false;      }  } |

Isso certamente vai conectar duas instâncias do seu app.

Handler responsável por receber mensagens da thread de conexão

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | public static Handler handler = new Handler() {  *@Override*      public void handleMessage(Message msg) {            Bundle bundle = msg.getData();          byte[] data = bundle.getByteArray("data");          String dataString= new String(data);            if(dataString.equals("---N"))              statusMessage.setText("Ocorreu um erro durante a conexão D:");          else if(dataString.equals("---S"))              statusMessage.setText("Conectado :D");        }  }; |

O código acima define um [Handler](https://developer.android.com/reference/android/os/Handler.html). Esse objeto será responsável por receber mensagens vindas de outra thread, nesse caso ConnectionThread, e tomar as decisões apropriadas. Aqui, fazemos com que ela mostre na tela o status da conexão, dependendo do código enviado pela thread de conexão.

Para testar você vai precisar de dois dispositivos Android com o app Super Bluetooth instalado e mais alguns botões! Mas façamos o seguinte antes: vamos ao resumo do que vai acontecer nessa thread. Depois disso veremos como iniciá-la e criaremos um botão para isso.

A thread é representada por uma classe que possui dois construtores, um para iniciar um servidor e outra para iniciar um cliente. Se você é um cliente, ou seja, aquele que busca a conexão, precisa do endereço do servidor. Então, se você criar um objeto ConnectionThread utilizando como argumento uma string contendo um endereço MAC, nossa thread define automaticamente que atuará como cliente. Caso não haja um argumento, ela atuará como servidor. A variável booleana server armazena essa informação. Ela é verdadeira se a thread atua como servidor e falsa caso atue como cliente.

Iniciando a thread de conexão como cliente.

Java



|  |  |
| --- | --- |
| 1  2 | ConnectionThread connect = new ConnectionThread("12-34-56-78-9A-BC");  connect.start(); |

Iniciando a thread de conexão como servidor.



|  |  |
| --- | --- |
| 1  2 | ConnectionThread connect = new ConnectionThread();  connect.start(); |

O método run() contem as intruções que serão executadas paralelamente à [Activity](http://developer.android.com/reference/android/app/Activity.html) principal. Isso inclui operações de conexão e leitura de dados, que deixariam a thread principal travada caso fossem executadas nela.

Quando a thread inicia, verificamos sua configuração de servidor ou cliente para que as ações corretas sejam realizadas.

**O servidor**

[listenUsingRfcommWithServiceRecord()](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter.html#listenUsingRfcommWithServiceRecord(java.lang.String,%20java.util.UUID)) é um método com nome muito grande. É usado para obter um ServerSocket, uma representação de servidor de um pontos de comunicação dentro da rede. Seus argumentos são uma String que identifica o app e um [UUID](https://developer.android.com/reference/java/util/UUID.html), que identifica o serviço disponibilizado pelo servidor. O UUID tem um formato bem específico, como você pode ver na String myUUID e a ideia é que cada serviço disponibilizado via Bluetooth possua um UUID diferente, apesar de existirem alguns padrões. Mas não se preocupe, myUUID pode ser trocado por qualquer UUID válido. Para que a conexão seja realizada, o cliente deverá informar a mesma UUID do servidor no ato da conexão. Então é importante que o cliente e o servidor compartilhem essa informação previamente.

O [ServerSocket](https://developer.android.com/reference/java/net/ServerSocket.html) é utilizado apenas para aguardar uma solicitação de conexão. O método [accept()](https://developer.android.com/reference/java/net/ServerSocket.html#accept()) deixa o socket de servidor na escuta por solicitações de clientes. Quando uma ocorre, retorna um BluetoothSocket e este será efetivamente usado para gerenciar a conexão. Se a conexão foi estabelecida, não precisamos mais do ServerSocket, que pode ser finalizado com [close()](https://developer.android.com/reference/java/net/ServerSocket.html#close()).

No caso em que uma excessão ocorre durante esse processo na thread, utilizamos o método toMainActivity() para enviar um código de erro informando que a conexão não foi bem sucedida. toMainActivity() utiliza um handler da [Activity](http://developer.android.com/reference/android/app/Activity.html) principal para transmitir esse código. Handlers fornecem a funcionalidade de transmissão de dados entre threads.

**O cliente**

Se a thread age como cliente, ela possui um endereço ao qual deve solicitar conexão. O endereço é usado para obter a representação de um dispositivo Bluetooth em um objeto BluetoothDevice. Isso é feito para que possamos chamar [createRfcommSocketToServiceRecord()](https://developer.android.com/reference/android/bluetooth/BluetoothDevice.html#createRfcommSocketToServiceRecord(java.util.UUID)) e obter um BluetoothSocket. Aqui, precisamos utilizar o mesmo UUID do servidor, caso contrário, a conexão não ocorrerá. Isso é uma forma de assegurar que o serviço que estamos buscando é realmente aquele que o servidor está fornecendo.

O processo de descoberta consome muitos recursos do adaptador Bluetooth. Se uma tentativa de conexão é feita enquanto o processo ocorre, é muito provável que falhas de conexão ocorram, portanto cancelamos a descoberta antes de solicitar uma conexão com o servidor.

A solicitação de conexão com o servidor é feita com o método [connect()](https://developer.android.com/reference/android/bluetooth/BluetoothSocket.html#connect()).

Assim como no servidor, caso haja uma exceção, enviamos uma mensagem de erro para a [Activity](http://developer.android.com/reference/android/app/Activity.html)principal.

Com isso descrevemos o início da conexão para os dois lados, servidor e cliente. Os dois métodos restantes na classe são toMainActivity() e cancel(). O primeiro é responsável por enviar mensagens à [Activity](http://developer.android.com/reference/android/app/Activity.html) principal do aplicativo, nesse caso, a thread que está executando os componentes da interface gráfica. O segundo método fecha os sockets e finaliza a conexão. Poderíamos criar um botão e fornecer ao usuário a capacidade de encerrar a conexão Bluetooth através desse método. Essa tarefa fica para você. Boa sorte!

Mas peraí! O que acontece se você simplesmente iniciar o aplicativo e esperar que os dispositivos se conectem? Absolutamente nada! É por isso que precisamos configurar para que o servidor espere pela conexão e o cliente solicite a conexão.

**4.1 Conectando como servidor**

Para o servidor, é bem simples. Você pode criar um objeto ConnectionThread em MainBluetoothActivity:

Yep.

Java



|  |  |
| --- | --- |
| 1 | ConnectionThread connect; |

Depois criamos um botão e fazemos com que ela execute o seguinte método:

Iniciando espera como servidor

Java



|  |  |
| --- | --- |
| 1  2  3  4  5 | public void waitConnection(View view) {        connect = new ConnectionThread();      connect.start();  } |

Como você já entende o que acontece ao chamarmos esses métodos, não preciso explicar nada. Mas, já explicando, criamos uma thread de conexão e a executamos. Como não passamos nenhum argumento ao construtor da thread, ela assume que deve atuar como servidor e já inicia a espera. Mágico. Só adicione o método à MainBluetoothActivity e o seguinte botão à *activity\_main\_bluetooth.xml*.

Botão para iniciar o servidor Bluetooth



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <Button      android:layout\_width="wrap\_content"      android:layout\_height="wrap\_content"      android:text="Esperar\nconexão"      android:id="@+id/button\_WaitConnection"      android:layout\_marginLeft="5dp"      android:layout\_alignBottom="@+id/button\_Visibility"      android:layout\_toRightOf="@+id/button\_Visibility"      android:layout\_toEndOf="@+id/button\_Visibility"      android:onClick="waitConnection" /> |

**4.2 Conectando como cliente**

Ainda mais simples! Lembra de como fizemos para selecionar um dispositivo pareado ou recém-pareado? Já tínhamos até seu endereço MAC, não é? Então faremos isso. Vamos modificar de leve o método[onActivityResult()](http://developer.android.com/reference/android/app/Activity.html#onActivityResult(int,%20int, android.content.Intent)) da [Activity](http://developer.android.com/reference/android/app/Activity.html) principal.

Conectando como cliente!

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | *@Override*  protected void onActivityResult(int requestCode, int resultCode, Intent data) {        if(requestCode == ENABLE\_BLUETOOTH) {          if(resultCode == RESULT\_OK) {              statusMessage.setText("Bluetooth ativado :D");          }          else {              statusMessage.setText("Bluetooth não ativado :(");          }      }      else if(requestCode == SELECT\_PAIRED\_DEVICE || requestCode == SELECT\_DISCOVERED\_DEVICE) {          if(resultCode == RESULT\_OK) {              statusMessage.setText("Você selecionou " + data.getStringExtra("btDevName") + "\n"                                      + data.getStringExtra("btDevAddress"));                connect = new ConnectionThread(data.getStringExtra("btDevAddress"));              connect.start();          }          else {              statusMessage.setText("Nenhum dispositivo selecionado :(");          }      }  } |

Veja as linhas 17 e 18, que adicionamos. Isso significa que iniciamos a thread de conexão com o endereço MAC de um dispositivo Bluetooth. A thread automaticamente configura-se para atuar como cliente, solicitando conexão ao dispositivo selecionado. Estamos conectados! A vida é bela. Mas apenas se o dispositivo selecionado estiver esperando como servidor e os UUIDs forem iguais. Caso contrário, a vida é bug.

**5. Transferência de dados bidirecional**

Uma conexão bem sucedida significa que os dois aparelhos estão prontos para trocar mensagens entre si. As duas operações, de leitura e escrita, agora podem ser realizadas a partir dos fluxos de entrada e saída dos sockets. Mas vamos pensar um pouco…

A operação de escrita é simples. Basta capturar o fluxo de saída do socket da conexão e usar o método write(). Este método tem como entrada um vetor de bytes que representa o dado a ser transmitido. Então, se você quer enviar uma string, primeiro transforme-a em bytes e depois use write(). Todo tipo de dado que pode ser transmitido pode ser representado como um vetor de bytes. Então basta encontrar uma função na extensa lista de pacotes do Java para representar a informação desejada em um vetor de bytes antes de transmiti-la. Do outro lado da conexão, usa-se algum outro método que obtenha a informação original a partir do vetor de bytes recebido. Calma, logo teremos código.

Já a operação de leitura é um pouco mais complicada. Isso porque ela bloqueia a execução do algoritmo. Quando se usa o método read(), a thread sofre uma pausa enquanto o método espera por algum dado vindo do outro lado da conexão. Então como fazer com que a mesma thread possa receber e transmitir dados ao mesmo tempo? A ideia é a seguinte.

Façamos com que, depois de realizada a conexão, a thread entre em um loop infinito na qual ela apenas espera para receber dados. Quando um dado é recebido, enviamos o dado para outra thread para processamento. No entanto, a thread de conexão volta ao estado de espera por novos dados. Isso garante que estejamos sempre na escuta de novas transmissões. Para transmitir um dado, criamos um método, dentro da thread, que tem como entrada um vetor de bytes e apenas o transmite ao outro lado da conexão. Deixaremos esse método público de form que possa ser invocado a partir de outras threads. E pronto! Com isso, basta chamar o método de transmissão, que será executado na thread em que foi invocado e não haverá conflito com a operação de leitura. Assim o aplicativo pode ficar sempre na escuta e sempre capaz de realizar uma transmissão.

Finalmente, código… Veja a seguir a versão final da classe ConnectionThread. Agora completa.

Versão final de ConnectionThread

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236 | package com.project.dragaosemchama.superbluetooth;    import android.bluetooth.BluetoothAdapter;  import android.bluetooth.BluetoothDevice;  import android.bluetooth.BluetoothServerSocket;  import android.bluetooth.BluetoothSocket;  import android.os.Bundle;  import android.os.Message;    import java.io.IOException;  import java.io.InputStream;  import java.io.OutputStream;  import java.util.Arrays;  import java.util.UUID;    public class ConnectionThread extends Thread{        BluetoothSocket btSocket = null;      BluetoothServerSocket btServerSocket = null;      InputStream input = null;      OutputStream output = null;      String btDevAddress = null;      String myUUID = "00001101-0000-1000-8000-00805F9B34FB";      boolean server;      boolean running = false;        /\*  Este construtor prepara o dispositivo para atuar como servidor.       \*/      public ConnectionThread() {            this.server = true;      }        /\*  Este construtor prepara o dispositivo para atuar como cliente.          Tem como argumento uma string contendo o endereço MAC do dispositivo      Bluetooth para o qual deve ser solicitada uma conexão.       \*/      public ConnectionThread(String btDevAddress) {            this.server = false;          this.btDevAddress = btDevAddress;      }        /\*  O método run() contem as instruções que serão efetivamente realizadas      em uma nova thread.       \*/      public void run() {            /\*  Anuncia que a thread está sendo executada.              Pega uma referência para o adaptador Bluetooth padrão.           \*/          this.running = true;          BluetoothAdapter btAdapter = BluetoothAdapter.getDefaultAdapter();            /\*  Determina que ações executar dependendo se a thread está configurada          para atuar como servidor ou cliente.           \*/          if(this.server) {                /\*  Servidor.               \*/              try {                    /\*  Cria um socket de servidor Bluetooth.                      O socket servidor será usado apenas para iniciar a conexão.                      Permanece em estado de espera até que algum cliente                  estabeleça uma conexão.                   \*/                  btServerSocket = btAdapter.listenUsingRfcommWithServiceRecord("Super Bluetooth", UUID.fromString(myUUID));                  btSocket = btServerSocket.accept();                    /\*  Se a conexão foi estabelecida corretamente, o socket                  servidor pode ser liberado.                   \*/                  if(btSocket != null) {                        btServerSocket.close();                  }                } catch (IOException e) {                    /\*  Caso ocorra alguma exceção, exibe o stack trace para debug.                      Envia um código para a Activity principal, informando que                  a conexão falhou.                   \*/                  e.printStackTrace();                  toMainActivity("---N".getBytes());              }              } else {                /\*  Cliente.               \*/              try {                    /\*  Obtem uma representação do dispositivo Bluetooth com                  endereço btDevAddress.                      Cria um socket Bluetooth.                   \*/                  BluetoothDevice btDevice = btAdapter.getRemoteDevice(btDevAddress);                  btSocket = btDevice.createRfcommSocketToServiceRecord(UUID.fromString(myUUID));                    /\*  Envia ao sistema um comando para cancelar qualquer processo                  de descoberta em execução.                   \*/                  btAdapter.cancelDiscovery();                    /\*  Solicita uma conexão ao dispositivo cujo endereço é                  btDevAddress.                      Permanece em estado de espera até que a conexão seja                  estabelecida.                   \*/                  if (btSocket != null)                      btSocket.connect();                } catch (IOException e) {                    /\*  Caso ocorra alguma exceção, exibe o stack trace para debug.                      Envia um código para a Activity principal, informando que                  a conexão falhou.                   \*/                  e.printStackTrace();                  toMainActivity("---N".getBytes());              }            }            /\*  Pronto, estamos conectados! Agora, só precisamos gerenciar a conexão.              ...           \*/            if(btSocket != null) {                /\*  Envia um código para a Activity principal informando que a              a conexão ocorreu com sucesso.               \*/              toMainActivity("---S".getBytes());                try {                    /\*  Obtem referências para os fluxos de entrada e saída do                  socket Bluetooth.                   \*/                  input = btSocket.getInputStream();                  output = btSocket.getOutputStream();                    /\*  Cria um byte array para armazenar temporariamente uma                  mensagem recebida.                      O inteiro bytes representará o número de bytes lidos na                  última mensagem recebida.                   \*/                  byte[] buffer = new byte[1024];                  int bytes;                    /\*  Permanece em estado de espera até que uma mensagem seja                  recebida.                      Armazena a mensagem recebida no buffer.                      Envia a mensagem recebida para a Activity principal, do                  primeiro ao último byte lido.                      Esta thread permanecerá em estado de escuta até que                  a variável running assuma o valor false.                   \*/                  while(running) {                        bytes = input.read(buffer);                      toMainActivity(Arrays.copyOfRange(buffer, 0, bytes));                    }                } catch (IOException e) {                    /\*  Caso ocorra alguma exceção, exibe o stack trace para debug.                      Envia um código para a Activity principal, informando que                  a conexão falhou.                   \*/                  e.printStackTrace();                  toMainActivity("---N".getBytes());              }          }        }        /\*  Utiliza um handler para enviar um byte array à Activity principal.          O byte array é encapsulado em um Bundle e posteriormente em uma Message      antes de ser enviado.       \*/      private void toMainActivity(byte[] data) {            Message message = new Message();          Bundle bundle = new Bundle();          bundle.putByteArray("data", data);          message.setData(bundle);          MainBluetoothActivity.handler.sendMessage(message);      }        /\*  Método utilizado pela Activity principal para transmitir uma mensagem ao       outro lado da conexão.          A mensagem deve ser representada por um byte array.       \*/      public void write(byte[] data) {            if(output != null) {              try {                    /\*  Transmite a mensagem.                   \*/                  output.write(data);                } catch (IOException e) {                  e.printStackTrace();              }          } else {                /\*  Envia à Activity principal um código de erro durante a conexão.               \*/              toMainActivity("---N".getBytes());          }      }        /\*  Método utilizado pela Activity principal para encerrar a conexão       \*/      public void cancel() {            try {                running = false;              btServerSocket.close();              btSocket.close();            } catch (IOException e) {              e.printStackTrace();          }          running = false;      }  } |

Show. Com essas adições implementamos a capacidade de transmissão de dados via Bluetooth. Agora para completar nosso app de demonstração, vamos adicionar três elementos à interface gráfica:

* Um EditText, para que possamos digitar uma mensagem;
* Um botão, para transmitir a mensagem;
* Um [TextView](http://developer.android.com/reference/android/widget/TextView.html), para visualizarmos as mensagens recebidas.

Alguns widgets para visualização e envio de mensagens



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | <EditText      android:layout\_width="match\_parent"      android:layout\_height="wrap\_content"      android:id="@+id/editText\_MessageBox"      android:layout\_alignParentBottom="true"      android:layout\_alignParentRight="true"      android:layout\_alignParentEnd="true" />    <Button      style="?android:attr/buttonStyleSmall"      android:layout\_width="wrap\_content"      android:layout\_height="wrap\_content"      android:text="Send"      android:id="@+id/button\_Send"      android:layout\_above="@+id/editText\_MessageBox"      android:layout\_alignRight="@+id/editText\_MessageBox"      android:layout\_alignEnd="@+id/editText\_MessageBox"      android:onClick="sendMessage" />    <TextView      android:layout\_width="wrap\_content"      android:layout\_height="wrap\_content"      android:text=""      android:id="@+id/textSpace"      android:layout\_centerVertical="true"      android:layout\_centerHorizontal="true" /> |

Adicione os widgets acima ao arquivo *activity\_main\_bluetooth.xml* e veja o resultado.



Interface gráfica pronta. Precisamos apenas terminar de programar suas funcionalidades.

Primeiramente o botão. Quando pressionado, o app deve ler o conteúdo da caixa de texto e fazer a transmissão. Veja o método a seguir. É só adicioná-lo à MainBluetoothActivity.

Método executado quando o botão Send é pressionado

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | public void sendMessage(View view) {        EditText messageBox = (EditText) findViewById(R.id.editText\_MessageBox);      String messageBoxString = messageBox.getText().toString();      byte[] data =  messageBoxString.getBytes();      connect.write(data);  } |

Agora, quando uma mensagem for recebida, o app deve exibi-la na tela. Para isso, declare um [TextView](http://developer.android.com/reference/android/widget/TextView.html) no topo da classe MainBluetoothActivity.

Declaração de um TextView

Java



|  |  |
| --- | --- |
| 1 | static TextView textSpace; |

Faça a ligação desse objeto com o widget da interface gráfica. O código abaixo vai para dentro do método onCreate().

Conectando o widget ao objeto TextView

Java



|  |  |
| --- | --- |
| 1 | textSpace = (TextView) findViewById(R.id.textSpace); |

E, finalmente, modifique o código do Handler. Adicione as linhas grifadas a seguir.

Modificação do método Handler

Java



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | public static Handler handler = new Handler() {  *@Override*      public void handleMessage(Message msg) {            Bundle bundle = msg.getData();          byte[] data = bundle.getByteArray("data");          String dataString= new String(data);            if(dataString.equals("---N"))              statusMessage.setText("Ocorreu um erro durante a conexão D:");          else if(dataString.equals("---S"))              statusMessage.setText("Conectado :D");          else {                textSpace.setText(new String(data));          }      }  }; |

Ótimo! Vamos testar. Consiga dois dispositivos Android e instale o app Super Bluetooth em ambos. Escolha qual iniciará como servidor e qual será agirá como cliente.

No dispositivo servidor, se os dispositivos não estiverem pareados, pressione o botão “Habilitar visibilidade” e, em seguida, “Esperar conexão”. Se os dispositivos estiverem pareado, basta clicar “Esperar conexão”.

Agora, no dispositivo cliente, se o pareamento ainda não ocorreu, pressione “Iniciar descoberta de dispositivos” e torça para o servidor aparecer na lista. Se já estiverem pareados, pressione “Dispositivos pareados”. Lembre-se que o servidor deve estar esperando a conexão para que haja sucesso.

Supondo que tudo deu certo, agora você pode digitar uma mensagem na caixa de texto, clicar “Send” e vê-la do outro lado da conexão. Parabéns!

Problems? Você pode baixar o [código completo](https://github.com/dragaosemchama/SuperBluetooth) no [github do Dragão](https://github.com/dragaosemchama), copiar tudo e melhorar para fazer seus projetos sem se preocupar com direitos autorais, tudo free. Ainda muitos bugs? Fale com a gente nos comentários!