

Android
ListView

SUMÁRIO

1.	Informativo.....	3
2.	Objetivo.....	4
3.	Pré-Requisitos	5
4.	Versão.....	6
4.1.	minSdkVersion	6
4.2.	targetSdkVersion.....	6
5.	ListView	7
5.1.	Problema	8
5.2.	Solução	8
5.3.	Adapter.....	9
5.4.	Criando a ListView	9
5.4.1.	Como?	10
5.4.2.	Resultado	13
5.4.3.	Selecionando um item da lista.....	14
6.	Resumo.....	16
7.	Referências.....	17

1. Informativo

Autor(a): Helena Strada

Data de Criação: jan/2018

2. Objetivo

O objetivo desta apostila é mostrar a criação e a explicação sobre mostrar itens de um determinado tipo com ListView.

3. Pré-Requisitos

Java

Orientação a Objetos;
APIs;
Bibliotecas.

Android

Estrutura do Projeto;
XML;
Activity.

4. Versão

Android Studio 3.0.1

4.1. minSdkVersion

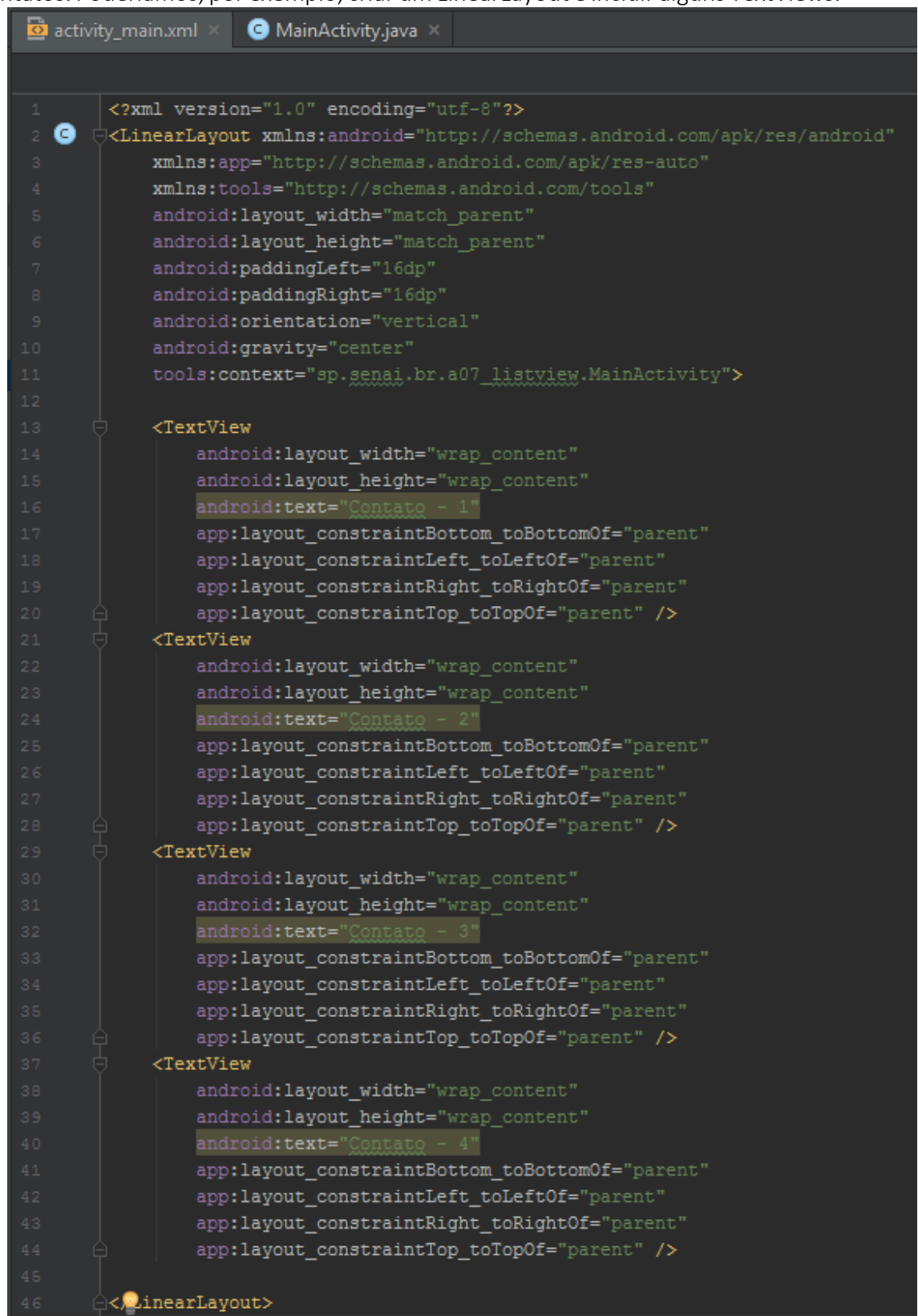
15

4.2. targetSdkVersion

26

5. ListView

Imagina que precisamos mostrar uma lista de contatos. Essa lista possui mais de 500 contatos. Poderíamos, por exemplo, criar um LinearLayout e incluir alguns TextViews.



```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:app="http://schemas.android.com/apk/res-auto"
4      xmlns:tools="http://schemas.android.com/tools"
5      android:layout_width="match_parent"
6      android:layout_height="match_parent"
7      android:paddingLeft="16dp"
8      android:paddingRight="16dp"
9      android:orientation="vertical"
10     android:gravity="center"
11     tools:context="sp.senai.br.a07_listview.MainActivity">
12
13     <TextView
14         android:layout_width="wrap_content"
15         android:layout_height="wrap_content"
16         android:text="Contato - 1"
17         app:layout_constraintBottom_toBottomOf="parent"
18         app:layout_constraintLeft_toLeftOf="parent"
19         app:layout_constraintRight_toRightOf="parent"
20         app:layout_constraintTop_toTopOf="parent" />
21
22     <TextView
23         android:layout_width="wrap_content"
24         android:layout_height="wrap_content"
25         android:text="Contato - 2"
26         app:layout_constraintBottom_toBottomOf="parent"
27         app:layout_constraintLeft_toLeftOf="parent"
28         app:layout_constraintRight_toRightOf="parent"
29         app:layout_constraintTop_toTopOf="parent" />
30
31     <TextView
32         android:layout_width="wrap_content"
33         android:layout_height="wrap_content"
34         android:text="Contato - 3"
35         app:layout_constraintBottom_toBottomOf="parent"
36         app:layout_constraintLeft_toLeftOf="parent"
37         app:layout_constraintRight_toRightOf="parent"
38         app:layout_constraintTop_toTopOf="parent" />
39
40     <TextView
41         android:layout_width="wrap_content"
42         android:layout_height="wrap_content"
43         android:text="Contato - 4"
44         app:layout_constraintBottom_toBottomOf="parent"
45         app:layout_constraintLeft_toLeftOf="parent"
46         app:layout_constraintRight_toRightOf="parent"
47         app:layout_constraintTop_toTopOf="parent" />
48
49 </LinearLayout>
```

Figura 1 – activity_main.xml



Figura 2 – Resultado do nosso exemplo da Figura 2.

5.1. Problema

Isso iria gastar muita memória para armazenar e listar todos esses itens mesmo que eles não estão sendo mostrados na tela ao mesmo tempo.

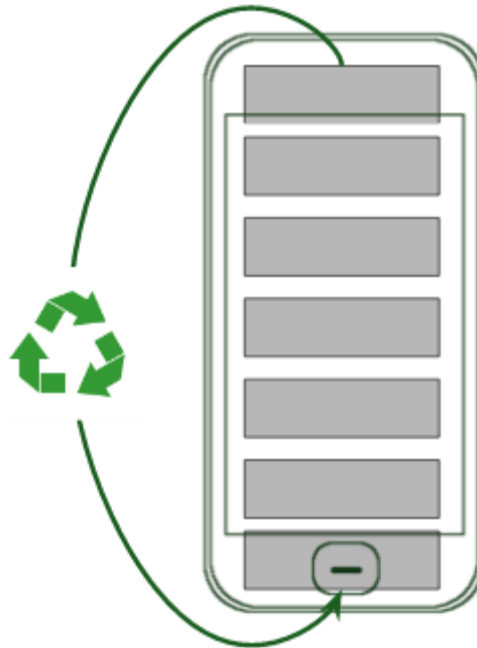
5.2. Solução

Nós, desenvolvedores, devemos sempre lembrar que estamos trabalhando com recursos limitados do celular. Logo, temos que nos atentar ao uso de recursos de memória.

Para resolver este problema, iremos trabalhar com Reciclagem de *Views*.

Assim, sempre que carregado a nossa lista, ele irá mostrar os itens para o usuário. Caso ele queira visualizar os outros itens daquela lista, basta que ele role a *View* e irá aparecer os outros itens e além disso, irá sumir os itens atuais.

Dessa maneira, não precisamos reescrever a nossa View. Tornando somente os itens que estão visíveis na lista e sempre que eu precisar rolar, irei utilizar os itens já criados, não precisando criar novamente.



Posso assim reutilizar um item da lista que não está mais sendo utilizado, reciclando-o, ao invés de criar um novo item do zero.

5.3. Adapter

O Adapter é responsável por gerenciar e adaptar os dados nas *Views*, ou, para entendermos melhor, um item da nossa lista.



O Adapter infla o layout de cada linha com o método *getView()* que veremos adiante.

Os adapters não somente são utilizados pelas *ListView*s, mas por outros recursos do Android também.

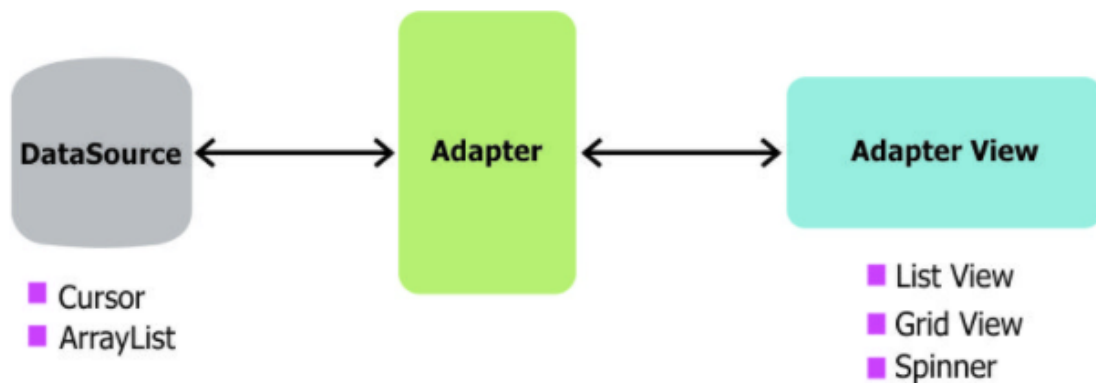
Além disso, existem dois métodos importantes dele:

notifyDataSetChanged(): é chamado se os dados tiverem sido alterados ou se houver novos dados disponíveis.

notifyDataSetInvalidated(): é chamado se os dados não estiverem mais disponíveis.

Então vimos o problema ao carregar 500 contatos em nossa tela, por exemplo com *TextView*.

5.4. Criando a *ListView*



O ArrayAdapter é o que fica entre os dados da sua aplicação e a apresentação em tela para o usuário.

O que precisamos fazer neste caso é saber qual lista devemos utilizar como data source e em seguida como manipular estes dados para a View.

5.4.1. Como?

Uma das maneiras mais utilizadas de listar os dados no Android é utilizando o ListView.

O primeiro para utilizarmos o ListView é colocando em nosso layout XML. Lembrando que o nosso XML é o que conterá as informações que queremos colocar na tela para o usuário (interface).

No nosso exemplo, iremos remover todos os TextView que adicionamos na nossa listagem e iremos inserir o nosso ListView. Inserindo também um id correspondente para a nossa lista.

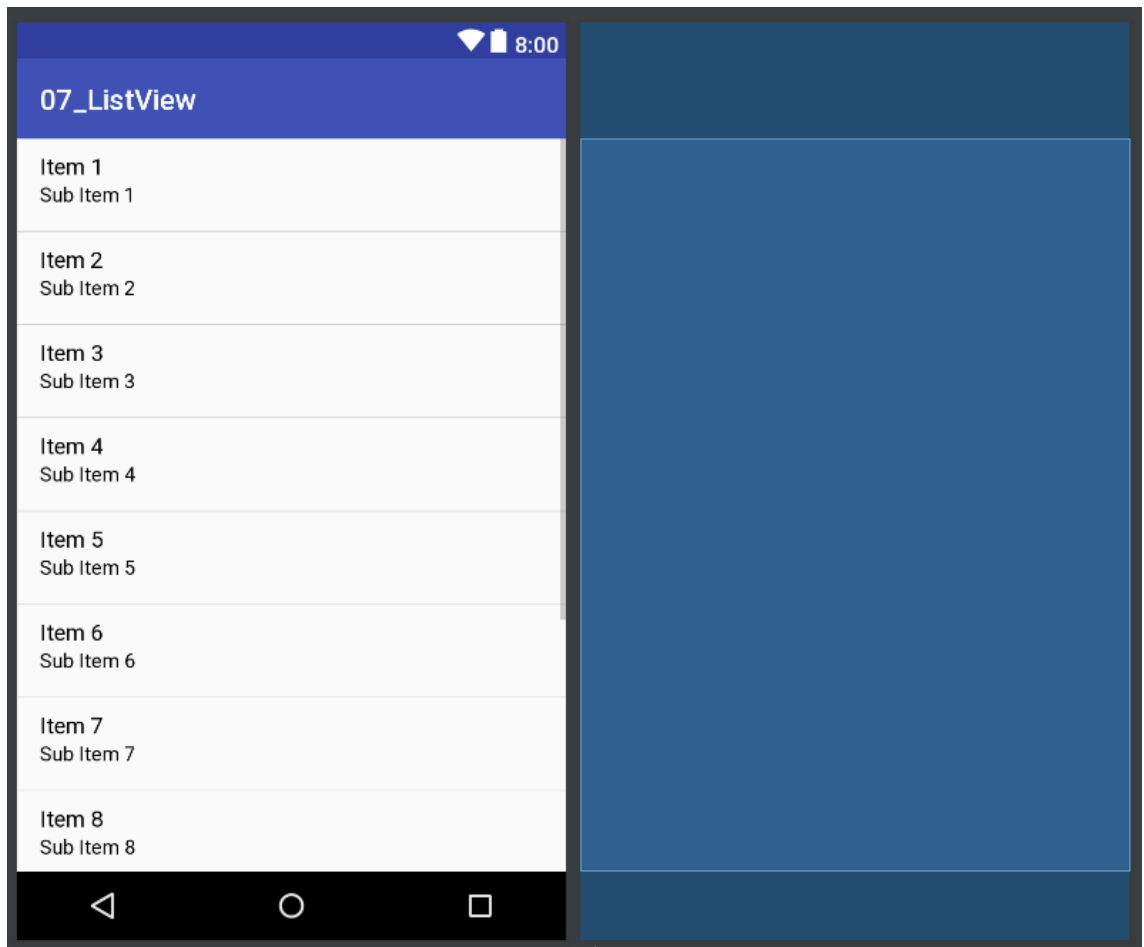
A imagem mostra uma captura de tela do editor de código do Android Studio, com o arquivo activity_main.xml aberto. O código XML define um RelativeLayout contendo um ListView. O ListView possui o id '@+id/lvContatos' e suas dimensões são definidas como wrap_content. O código é o seguinte:

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent">
5
6     <ListView
7         android:id="@+id/lvContatos"
8         android:layout_width="wrap_content"
9         android:layout_height="wrap_content" />
10
11 </RelativeLayout>
```

Figura 3 – activity_main.xml – código com a ListView.

Além disso, iremos deixar nosso layout como *RelativeLayout*.

Em um primeiro momento, se visualizarmos o nosso design, ele deverá ficar assim.



O que devemos fazer agora é buscar a referência do que criamos no nosso XML, em nossa “MainActivity.java”.

```
package sp.senai.br.a07_listview;

import ...

public class MainActivity extends AppCompatActivity {

    private ListView lvContatos;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        lvContatos = findViewById(R.id.lvContatos);
    }
}
```

Figura 4 – Buscando a referência do id que criamos no nosso XML, na nossa classe.

Como estamos querendo demonstrar o uso da nossa *ListView*, iremos apenas inserir aqui uma lista de Strings dentro de um array para utilizarmos como exemplo.

```
String[] contatos = new String[] {"Helena", "Wilson", "Gustavo", "João",  
"Mateus"};
```

```
public class MainActivity extends AppCompatActivity {  
  
    private ListView lvContatos;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        lvContatos = findViewById(R.id.lvContatos);  
  
        String[] contatos = new String[] {"Helena", "Wilson", "Gustavo", "João", "Mateus"};  
    }  
}
```

Figura 5 – Criando um array de Strings.

Nosso próximo passo é criar o nosso *ArrayAdapter* identificando qual será o nosso *Context*, o ID do layout que iremos utilizar e o último parâmetro será a lista, o nosso *Array* de dados.

```
public class MainActivity extends AppCompatActivity {  
  
    private ListView lvContatos;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        lvContatos = findViewById(R.id.lvContatos);  
  
        String[] contatos = new String[] {"Helena", "Wilson", "Gustavo", "João", "Mateus"};  
  
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(context: this, android.R.layout.simple_list_item_1, contatos);  
    }  
}
```

Figura 6 – Neste caso, estamos utilizando o layout padrão do Android: *android.R.layout.simple_list_item_1*.

Falta apenas um passo. Fizemos referência a nossa lista do XML. Criamos a nossa lista de contatos. Criamos também o nosso adapter. Porém, percebe-se no próprio Android Studio que o nosso adapter não está sendo utilizado. Isso ocorre por quê ainda não dissemos para a nossa lista que o adapter que queremos utilizar, é este mesmo que criamos. Para isso, fazemos a nossa última etapa:

```

public class MainActivity extends AppCompatActivity {

    private ListView lvContatos;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        lvContatos = findViewById(R.id.lvContatos);

        String[] contatos = new String[] { "Helena", "Wilson", "Gustavo", "João", "Mateus" };

        ArrayAdapter<String> adapter = new ArrayAdapter<String>( context: this, android.R.layout.simple_list_item_1, contatos);

        lvContatos.setAdapter(adapter);
    }
}

```

Figura 7 – Inserindo o nosso adapter na nossa lista.

Basta executarmos a nossa aplicação para ver o resultado da nossa lista.

5.4.2. Resultado

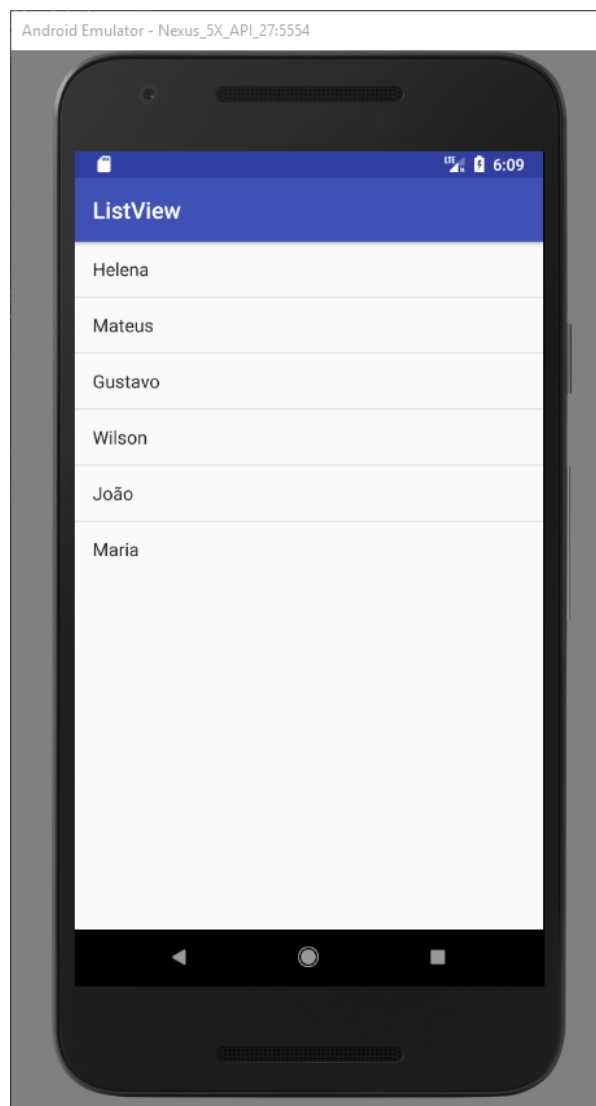


Figura 8 – Resultado da Listagem utilizando ListView

5.4.3. Selecionando um item da lista

Imagina que precisamos selecionar um item da lista, como faríamos? Para isso, podemos utilizar o método `setOnItemClickListener`.

Na nossa `MainActivity`, iremos adicionar a informação que segue:

```
lvContatos.setOnItemClickListener(new AdapterView.OnItemClickListener() {  
    @Override  
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {  
        String valor = (String) parent.getAdapter().getItem(position);  
        Toast.makeText(getApplicationContext(), valor, Toast.LENGTH_SHORT).show();  
    }  
});
```

Nosso código final da nossa listagem ficará assim:

```
public class MainActivity extends AppCompatActivity {  
    private ListView lvContatos;  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        lvContatos = findViewById(R.id.lvContatos);  
        String[] contatos = new String[] { "Helena", "Mateus", "Gustavo", "Wilson", "João", "Maria" };  
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(context: this, android.R.layout.simple_list_item_1, contatos);  
        lvContatos.setAdapter(adapter);  
        lvContatos.setOnItemClickListener(new AdapterView.OnItemClickListener() {  
            @Override  
            public void onItemClick(AdapterView<?> parent, View view, int position, long id) {  
                String valor = (String) parent.getAdapter().getItem(position);  
                Toast.makeText(getApplicationContext(), valor, Toast.LENGTH_SHORT).show();  
            }  
        });  
    }  
}
```

Figura 9 – Código final da activity

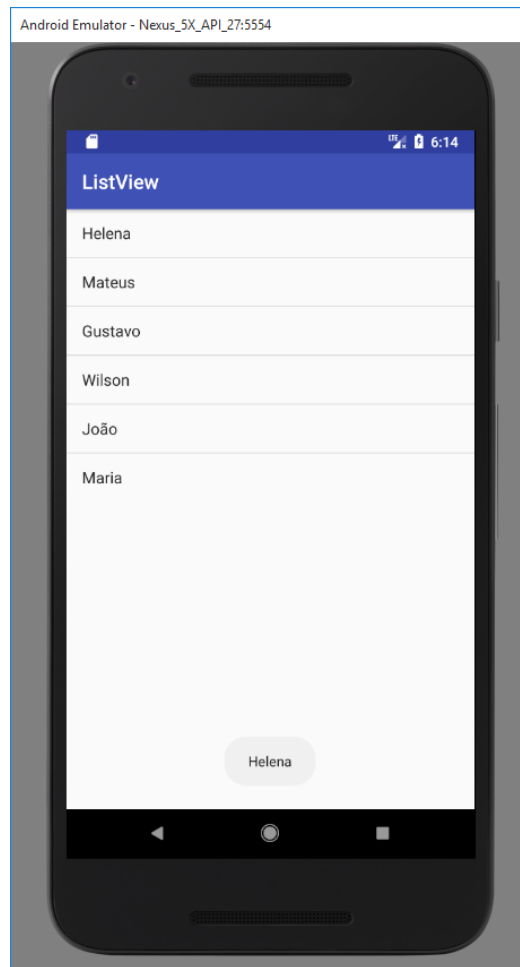


Figura 10 – Selecionando um item da lista

6. Resumo

Nesta apostila, mostramos como listar e apresentar os componentes da tela com `ListView`.

7. Referências

blog.alura.com.br/personalizando-uma-listview-no-android/

<https://guides.codepath.com/android/Using-an-ArrayAdapter-with-ListView>