CEFET-MG
CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

Aplicações Móveis

Aula 6 - Exibição de listas - ListView

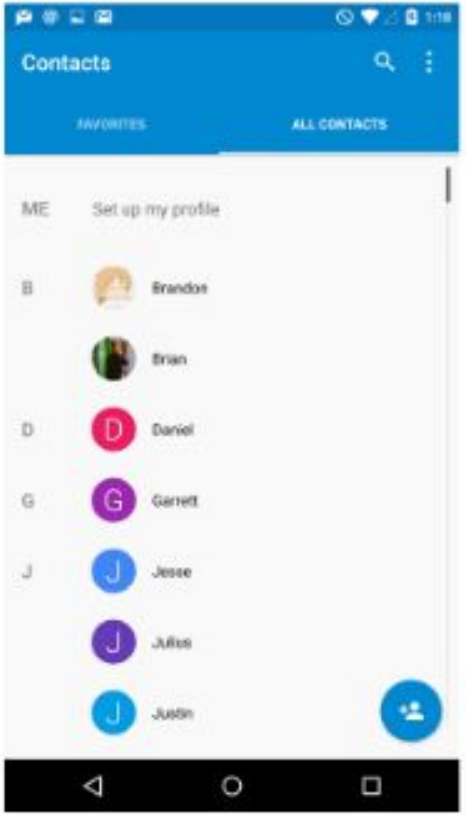
Prof. Dr. Wendell Fioravante da Silva Diniz
3º Ano - Informática Integrado
Centro Federal de Educação Tecnológica de Minas Gerais
Unidade Varginha

Recapitulando...

- Uma Activity pode assumir vários estados
- Cada mudança de estado dispara um evento associado
- A passagem entre os diferentes estados é chamado de Ciclo de Vida
- Uma Activity pode salvar dados para recuperar seu estado
- Usa-se um Intent para passar dados entre Activities
- SharedPreferences são um mecanismo simples de persistência de dados
- A classe Log fornece um meio simples de obter informações sobre a execução do App

Exibição de listas

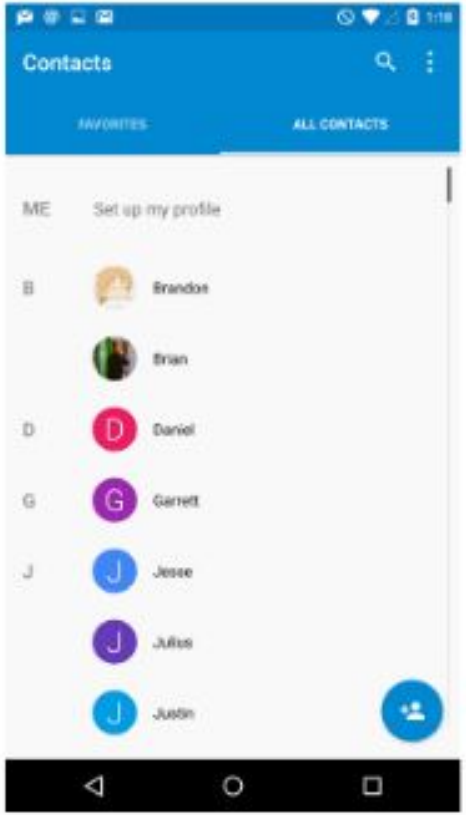
- Memória é um recurso limitado
- Exibição de listas com grandes quantidades de valores deve ser otimizada
- Problema: Imagine uma lista de contatos com 1000 nomes
    - Como exibir os dados?



CEFET-MG

Exibição de listas

- Memória é um recurso limitado
- Exibição de listas com grandes quantidades de valores deve ser otimizada
- Problema: Imagine uma lista de contatos com 1000 nomes
  - Como exibir os dados?
- Solução 1:
  - Usar um laço para incluir TextViews
- Errado! Consome muita memória

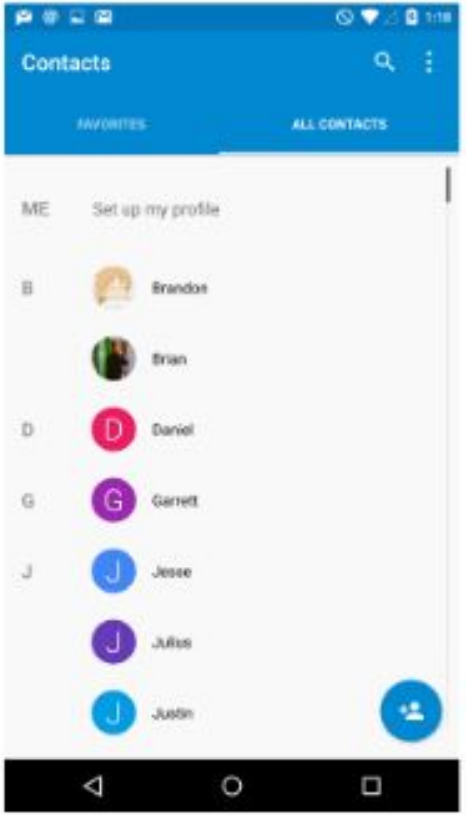


CEFET-MG

Exibição de listas

- Memória é um recurso limitado
- Exibição de listas com grandes quantidades de valores deve ser otimizada
- Problema: Imagine uma lista de contatos com 1000 nomes
  - Como exibir os dados?
- Solução 1:
  - Usar um laço para incluir TextViews
- Errado! Consome muita memória

Como exibir de forma eficiente?



CEFET-MG

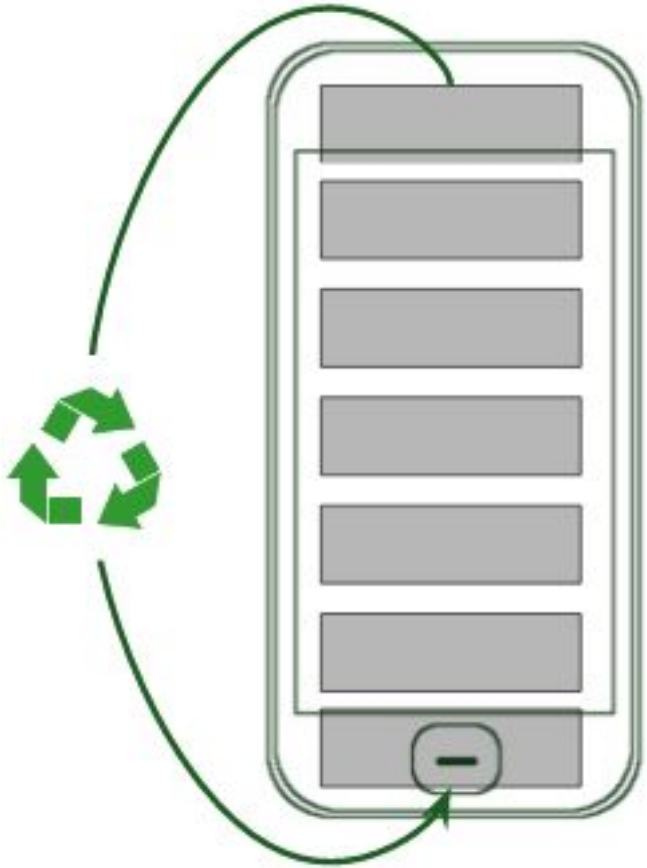
Reciclagem de Views

- A solução do Android é a Reciclagem de Views
- Exibir apenas a quantidade de elementos que o usuário pode ver naquele momento
- Imagine o layout de uma única linha
- O mesmo layout é replicado, mudando-se apenas seus dados
- Quando um item sai da área de visualização, ele é colocado em uma “pilha de sucata”, para ser reutilizado
- Apenas os dados daquela View são mudados, não precisando-se criar uma nova View


Itens em exibição
Pilha de sucata

Como funciona a reciclagem

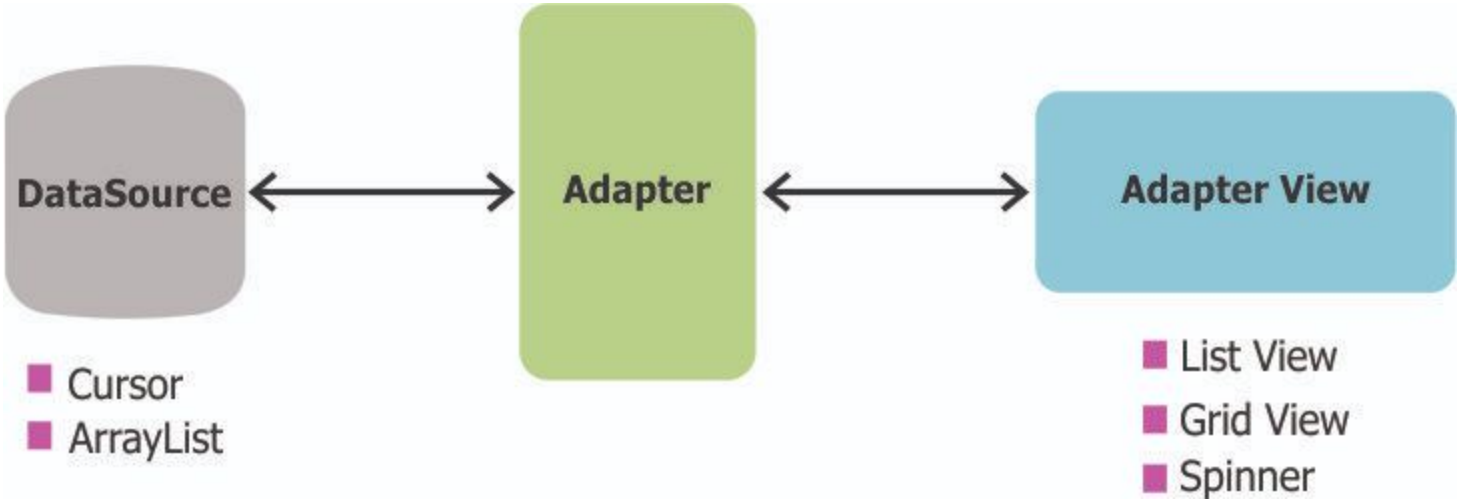
- O usuário rola a tela para cima
- O componente percebe que um novo item deve ser criado
- O item que está na parte superior da pilha de sucata é transferido para baixo
- Os dados são modificados
- O usuário tem a impressão de que os dados sempre estiveram ali



CEFET-MG

Listviews e Adapters

- O que é um Listview?
  - Componente para exibição de listas de valores
  - Usa um layout comum que é replicado para cada item
  - Os dados são gerenciados por um Adapter
- O que é um Adapter?
  - Componente responsável por gerenciar e adaptar os dados no layout da View
  - Atua como uma ponte entre os dados e a View
- O SDK fornece alguns Adapters prontos para uso
  - SimpleCursorAdapter
  - CursorAdapter
  - ArrayAdapter


DataSource
Adapter
Adapter View
Cursor
ArrayList
List View
Grid View
Spinner

ArrayAdapter

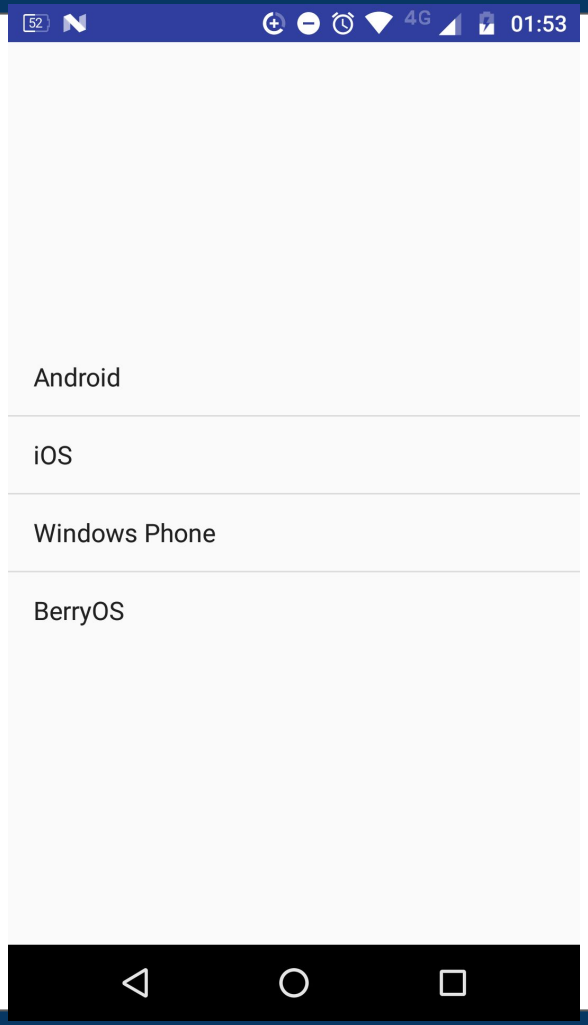
- Manipula dados armazenados em um Array ou uma lista (java.util.List)
- O layout pode ser tão complexo quanto o desenvolvedor precisar
- O Adapter deve ser adaptado de acordo (custom Adapters)
- Métodos importantes:
  - notifyDataSetChanged(): é chamado se os dados forem alterados ou se novos dados ficaram disponíveis
  - notifyDataSetInvalidated(): chamado caso os dados não estiverem mais disponíveis
- Link para a referência

```
String[] values = {"Android","iOS","Windows Phone","BerryOS"};
final ArrayList<String> theArray = new ArrayList<>();
for(int i = 0;i < values.length; ++i) {
    theArray.add(values[i]);
}
ArrayAdapter<String> adapter = new ArrayAdapter<String>(getApplicationContext(),
        R.layout.simple_list_item,R.id.lblFirst,theArray);
```

Listview

- Componente mais usado para exibição de dados em listas
- Exibe os dados empilhados de forma vertical
- Componente deve estar declarado no layout da Activity
- Link para a referência

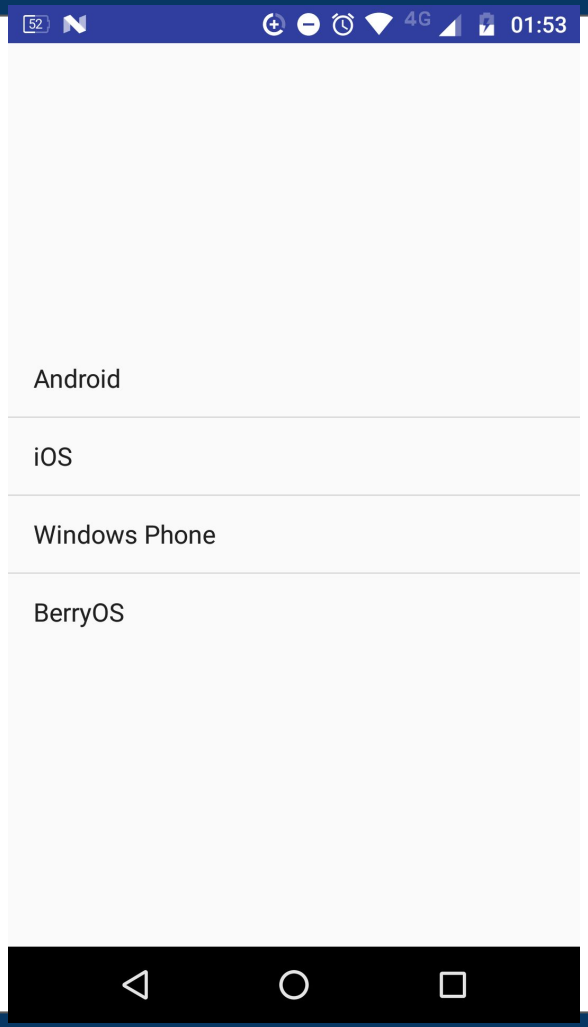
```
1 // Instanciando a ListView...
2 ListView listview = findViewById(R.id.lvItems);
3
4 // Definindo o adapter (com os dados) para a ListView...
5 listView.setAdapter(itemsAdapter);
```


01:53
Android
iOS
Windows Phone
BerryOS

A classe ListActivity

- A classe ListActivity é uma Activity especializada em exibição de listas
- Ela contém um ListView como seu único elemento
- Pode ser usado quando a única função da Activity é exibir uma lista
- Deve-se incluir no layout apenas um componente ListView, que deve obrigatoriamente ter o ID definido para @android:id/list

```java
package com.example.wendell.listviewsimples;

import android.app.ListActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

import java.util.ArrayList;

public class MainActivity extends ListActivity {

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

        String[] values = {"Android", "iOS", "Windows Phone", "BerryOS"};
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(getApplicationContext(),
                android.R.layout.simple_list_item_1, values);
        setListAdapter(adapter);
    }

    @Override
    protected void onListItemClick(ListView l, View v, int position, long id) {

        super.onListItemClick(l, v, position, id);
        // Posição do item tocado
        int itemPosition = position;
        // Recupera o valor do item tocado
        String itemValue = (String) l.getItemAtPosition(position);

        Toast toast = Toast.makeText(getApplicationContext(), itemValue, Toast.LENGTH_SHORT);
        toast.show();

    }
}
```


01:53
Android
iOS
Windows Phone
BerryOS

Personalizando Adapters

- Muitas vezes, queremos mostrar uma lista de itens mais elaborada
- Para isso, devemos construir o adaptador para o layout que criamos, desta forma direcionando os dados para seus elementos correspondentes
- O SDK fornece a classe BaseAdapter, que implementa as funcionalidades básicas de um Adapter
- A classe personalizada deve estendê-la, além de implementar as interfaces que o desenvolvedor julgar necessárias

Personalizando Adapters

- Problema: suponhamos que desejamos criar um Adapter para exibir uma lista de cursos de linguagens de programação, com o seguinte layout:

Personalizando Adapters

1. Criar o layout da linha

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="100dp"
    android:orientation="horizontal">


    <ImageView
        android:id="@+id/imgCurso"
        android:layout_width="100dp"
        android:layout_height="match_parent" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <TextView
            android:id="@+id/lblNomeCurso"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Titulo"
            android:textSize="30dp" />

        <TextView
            android:id="@+id/lblDescCurso"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="descriçao"
            android:textSize="20dp" />

    </LinearLayout>

</LinearLayout>
```

Personalizando Adapters

1. Criar o layout da linha
2. Criar uma classe com o modelo dos dados

```java
public class Curso {
    private String nome;
    private String descricao;
    private int imageId;

    void setNome(String val) {
        this.nome = val;
    }

    void setDescricao(String val) {
        this.descricao = val;
    }

    void setImageId(int resId) { this.imageId = resId;  }

    String getNome() {
        return this.nome;
    }

    String getDescricao() {
        return this.descricao;
    }

    int getImageId() { return this.imageId;}
}
```

Personalizando Adapters

1. Criar o layout da linha
2. Criar uma classe com o modelo dos dados
3. Criar a classe do adaptador personalizado, estendendo a classe BaseAdapter

```java
public class CursoAdapter extends BaseAdapter {
    @Override
    public int getCount() {
        return 0;
    }

    @Override
    public Object getItem(int position) {
        return null;
    }

    @Override
    public long getItemId(int position) {
        return 0;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        return null;
    }
}
```

Estes métodos são virtuais, ou seja são definidos na superclasse, mas devem ser obrigatoriamente implementados na subclasse

Personalizando Adapters

1. Criar o layout da linha
2. Criar uma classe com o modelo dos dados
3. Criar a classe do adaptador personalizado, estendendo a classe BaseAdapter
    a. Devemos acrescentar um atributo com uma java.util.List do modelo de dados, que será passado no construtor da classe
    b. Também passaremos no construtor uma referência à Activity que usar o adaptador
    c. Agora podemos implementar o método getCount(), que deve retornar a quantidade de elementos na lista

```java
public class CursoAdapter extends BaseAdapter {

    List<Curso> lista;
    Activity atividade;

    public CursoAdapter(List<Curso> aLista, Activity aAtividade) {
        this.lista = aLista;
        this.atividade = aAtividade;
    }
```

Personalizando Adapters

1. Criar o layout da linha
2. Criar uma classe com o modelo dos dados
3. Criar a classe do adaptador personalizado, estendendo a classe BaseAdapter
    a. Devemos acrescentar um atributo com uma java.util.List do modelo de dados, que será passado no construtor da classe
    b. Também passaremos no construtor uma referência à Activity que usar o adaptador
    c. Agora podemos implementar o método getCount(), que deve retornar a quantidade de elementos na lista
    d. O método getItem(int position) deve retornar um item da lista na posição passada

```java
@Override
public Object getItem(int position) {
    return lista.get(position);
}
```

Personalizando Adapters

1. Criar o layout da linha
2. Criar uma classe com o modelo dos dados
3. Criar a classe do adaptador personalizado, estendendo a classe BaseAdapter
    a. Devemos acrescentar um atributo com uma java.util.List do modelo de dados, que será passado no construtor da classe
    b. Também passaremos no construtor uma referência à Activity que usar o adaptador
    c. Agora podemos implementar o método getCount(), que deve retornar a quantidade de elementos na lista
    d. O método getItem(int position) deve retornar um item da lista na posição passada
    e. O método getItemId(int position) espera retornar o id do item na posição passada. Como nosso modelo não possui um ID, podemos deixar retornando 0. Quando usarmos itens que são carregados de um banco de dados, usaremos o id correspondente

Personalizando Adapters

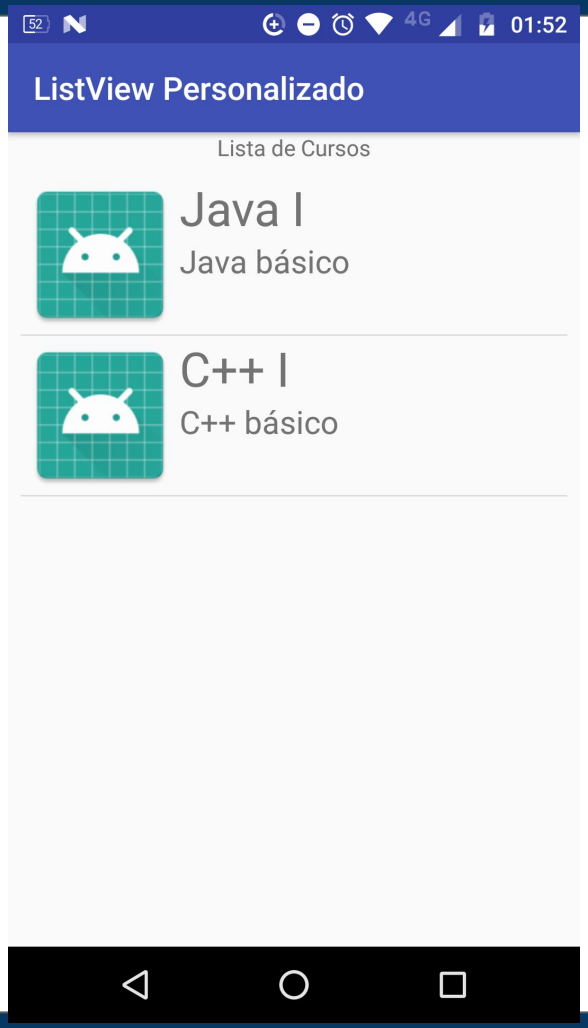
3. …
    f. Agora, vamos implementar o método getView(...).
        i. Repare que o método deve retornar uma View, ou seja, é este método que retorna uma linha da lista, já formatada. Ou seja, em vez de carregarmos os elementos com findElementById(...), vamos criá-los, com base no layout que definimos
        ii. Deve-se usar um Inflater, que é um método da classe Activity, responsável por inflar (preencher) uma View.

```java
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        View v = atividade.getLayoutInflater().inflate(R.layout.item_lista, parent, false);
        Curso curso = lista.get(position);

        //pegando as referências das Views
        TextView nome = (TextView)
                v.findViewById(R.id.lblNomeCurso);
        TextView descricao = (TextView)
                v.findViewById(R.id.lblDescCurso);
        ImageView imagem = (ImageView)
                v.findViewById(R.id.imgCurso);

        //populando as Views
        nome.setText(curso.getNome());
        descricao.setText(curso.getDescricao());
        imagem.setImageResource(curso.getImageId());

        return v;
    }
```

Personalizando Adapters

3. …
    f. Agora, vamos implementar o método getView(...).
        i. Repare que o método deve retornar uma View, ou seja, é este método que retorna uma linha da lista, já formatada. Ou seja, em vez de carregarmos os elementos com findElementById(...), vamos criá-los, com base no layout que definimos
        ii. Deve-se usar um Inflater, que é um método da classe Activity, responsável por inflar (preencher) uma View.
4. Agora, basta usar o novo adaptador no lugar do adaptador padrão

```java
public class MainActivity extends AppCompatActivity {

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

        ArrayList<Curso> cursos = listaTodosOsCursos();
        ListView listaDeCursos = (ListView) findViewById(R.id.listView);
        CursoAdapter adapter = new CursoAdapter(cursos, this);
        listaDeCursos.setAdapter(adapter);
    }

    private ArrayList<Curso> listaTodosOsCursos() {
        ArrayList<Curso> cursos = new ArrayList<>();
        Curso java = new Curso();
        java.setNome("Java I");
        java.setDescricao("Java básico");
        java.setImageId(R.mipmap.ic_launcher);
        cursos.add(java);

        Curso cpp = new Curso();
        cpp.setNome("C++ I");
        cpp.setDescricao("C++ básico");
        cpp.setImageId(R.mipmap.ic_launcher);
        cursos.add(cpp);

        return cursos;
    }
}
```


01:52
ListView Personalizado
Lista de Cursos
Java I
Java básico
C++ I
C++ básico

Leitura complementar recomendada

- Capítulo 7.17 e 7.18 do livro **Google Android** de Ricardo Lecheta

CEFET-MG