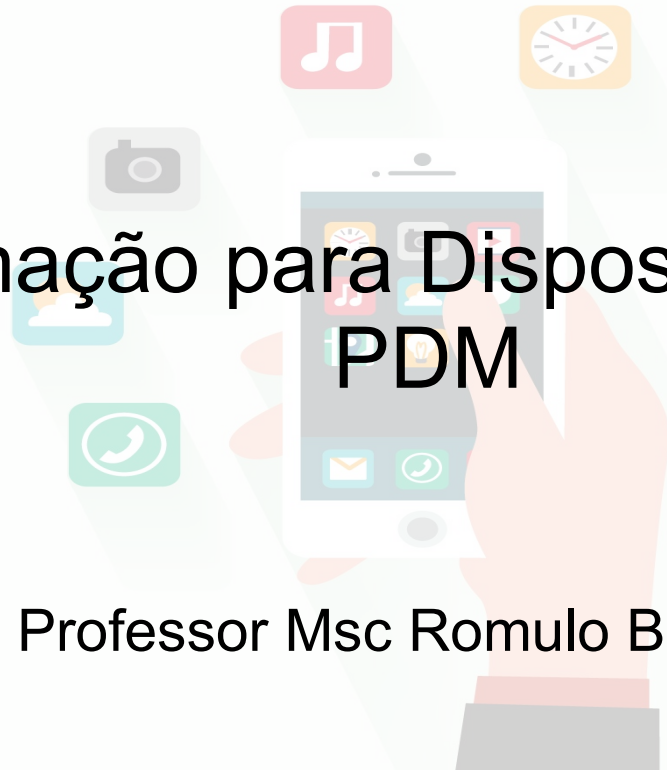


Programação para Dispositivos Móveis PDM

Professor Msc Romulo Beninca

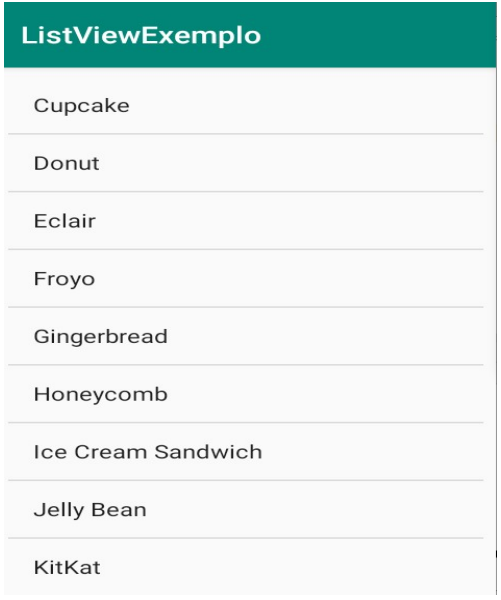
Objetivos da aula

Componentes de listagem

- *ListView*
- *Adapter*
- *ListView* com *template* personalizado
- Adapter Personalizado

Listagens com *ListView*

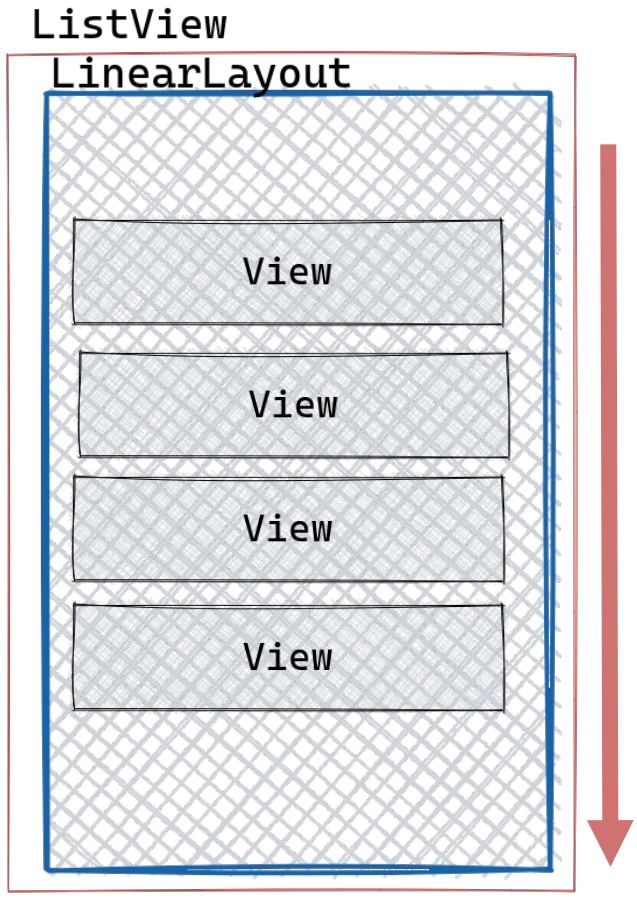
- *ListView* é um componente de listagem que possibilita listar *views.*



Fonte:Autor(2018)

Fonte:Autor(2018)

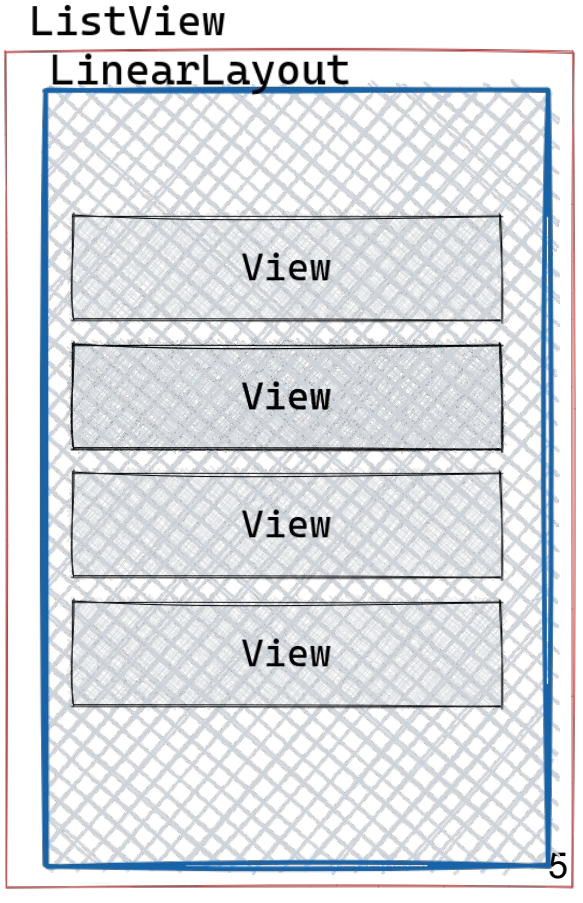
Listview



Listagens com *ListView*
Funcionamento de um ListView

A classe *ListView* estruturalmente contem um *LinearLayout* que pode posicionar *Views* em uma listagem vertical.

Entretanto o *ListView* tem a implementação de construção de *Views* ou como associar as view dados.



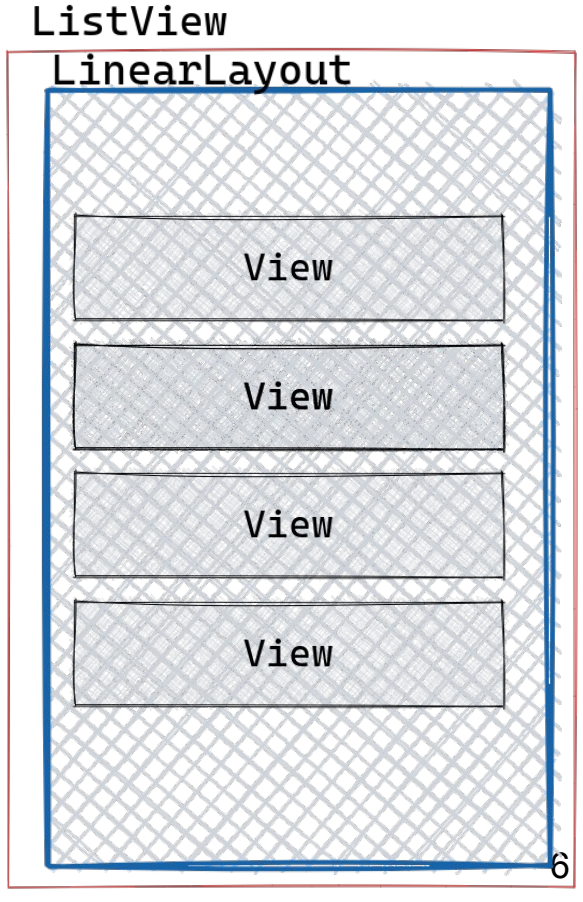
Listagens com *ListView*
Funcionamento de um ListView

Então como podemos passar o *template* de cada item de *view* e o conjunto de dados para o *Listview* e ainda inflar os *templates* e associar dados.

DataSource

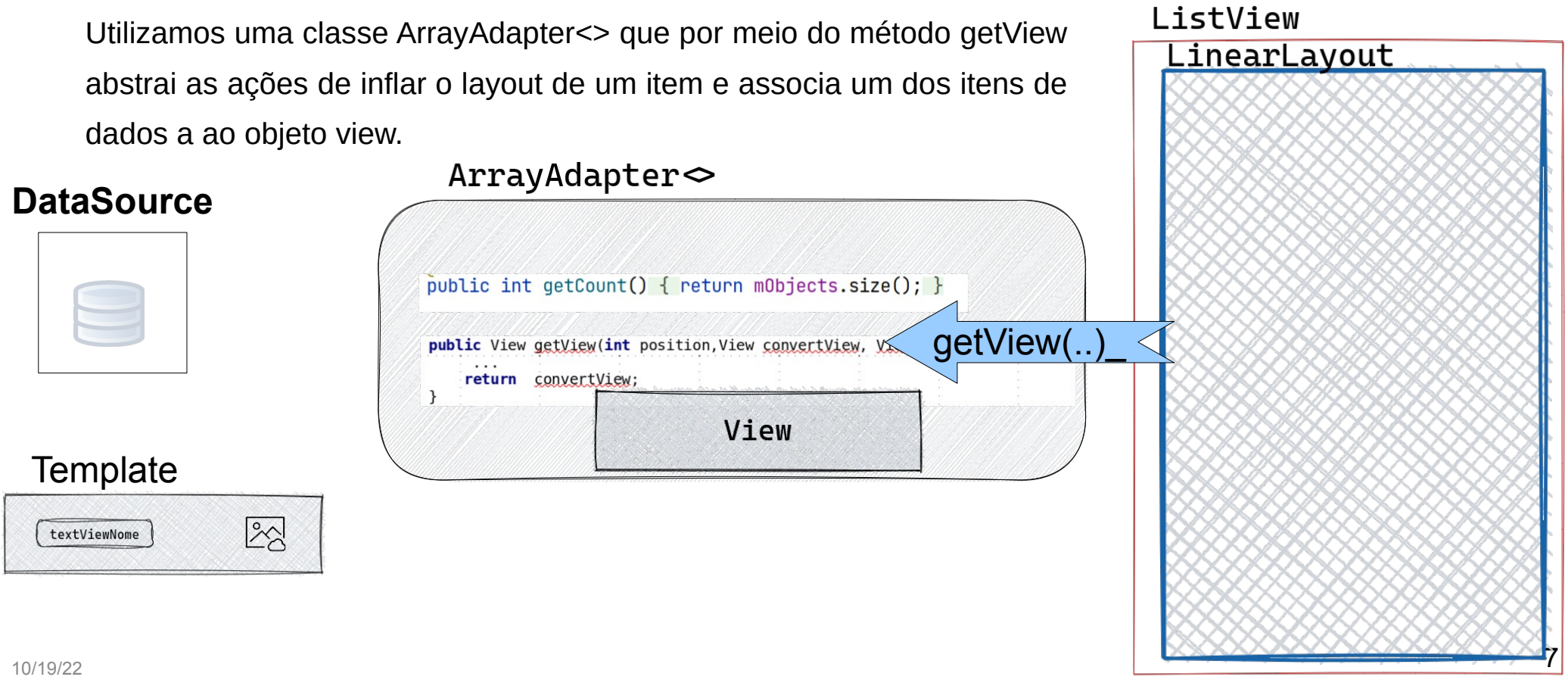
?

Template



Listagens com *ListView*
Funcionamento de um ListView

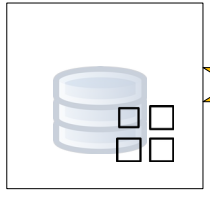
Utilizamos uma classe ArrayAdapter<> que por meio do método getView abstrai as ações de inflar o layout de um item e associa um dos itens de dados a ao objeto view.



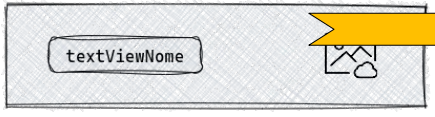
Listagens com *ListView*
Funcionamento de um ListView

Então inicialmente passamos os dados e o template a uma instancia de ArrayAdapter por meio do seu construtor.

DataSource



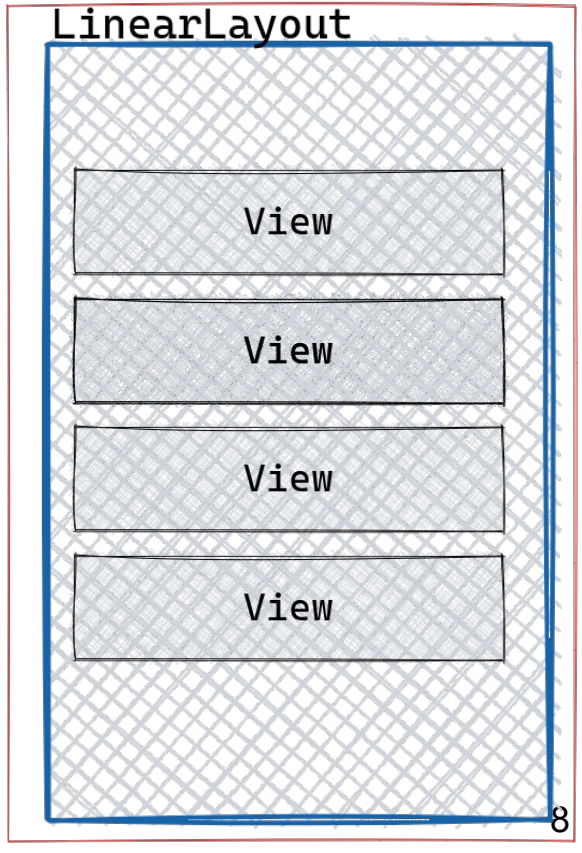
Template



ArrayAdapter<>

```
public int getCount() { return mObjects.size(); }

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



Listagens com *ListView*
Funcionamento de um ListView

Inicialmente o ListView solicita chama o método getCount para saber a quantidade de dados e depois faz uma iteração chamando getView

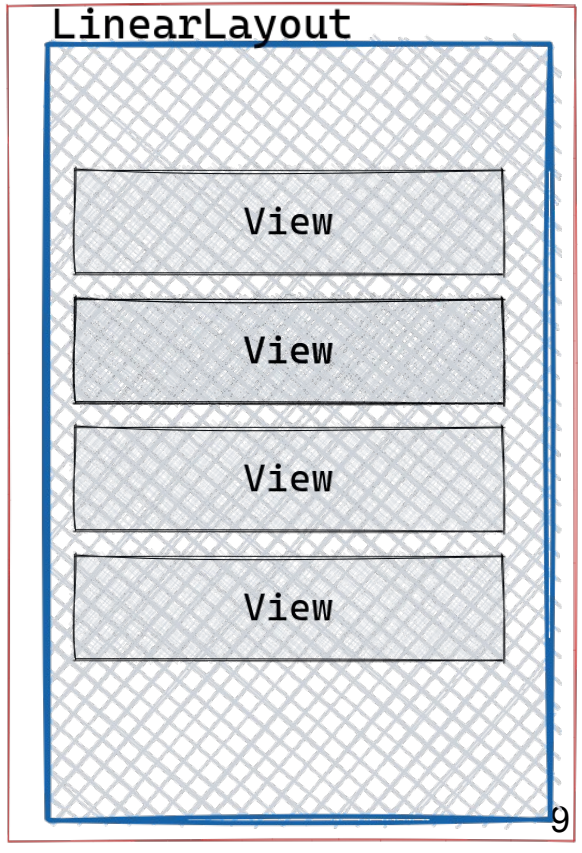
DataSource

Template

ArrayAdapter<>

```
public int getCount() { return mObjects.size(); }

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



Listagens com *ListView*
Funcionamento de um ListView

A cada chamada de getView é inflado um item do layout, e em seguida associado os dados as views internas da view que será retoranada pelo método getView.

DataSource

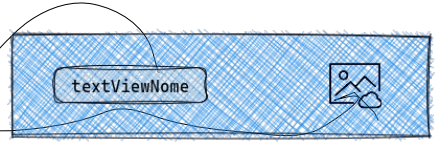
Template

ayAdapter<>

```
int getCount() { return mObjects.size(); }

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

mObject[position]
nome=Pedro;
photo=imgage....



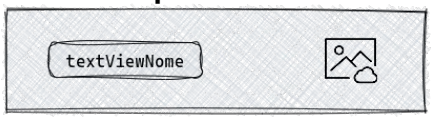
ListView
LinearLayout

Listagens com *ListView*
Funcionamento de um ListView

Após a construção da view que representa o item mesma é retornada para o linear layout que faz a exibição ao final da iteração de todos objetos

DataSource

Template




```
ayAdapter<>

pub  int getCount() { return mObjects.size(); }

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

mObject[position]
nome=Pedro;
photo=imgage….

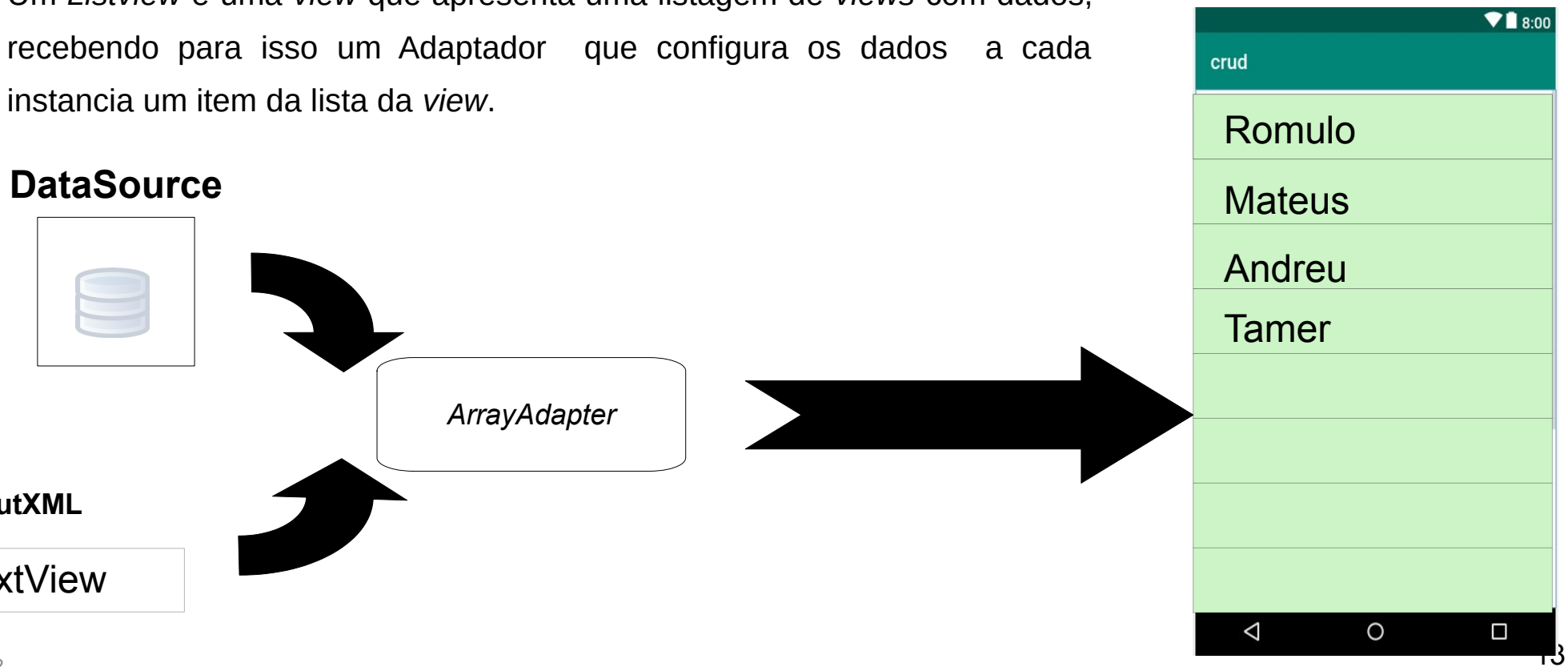
ListView

LinearLayout

Utilizando adapter para Strings

Listagens com *ListView*

Um *Listview* é uma *view* que apresenta uma listagem de *views* com dados, recebendo para isso um Adaptador que configura os dados a cada instancia um item da lista da *view*.



Listagens com *ListView*

O Adaptador *arrayAdapter* ajusta cada campo do a uma item da *view*. Se o templete utilizado para inflar cada campo do *listview* possui apenas *view*, podemos facilmente associar os elementos de um arrayList ou *array* de strings a cada item do *ListView*.

Trecho de código ListaAlunos.java

```
String[] ArrayAlunos = new String[] { "Romulo", "Mateus", "Andreu", "Tamer" };
ArrayAdapter adapter = new ArrayAdapter(
                                  this,
android.R.layout.simple_list_item_1,
android.R.id.text1,
ArrayAlunos);

listView.setAdapter(adapter);
```

1º Parâmetro Contexto ----------> this,

2º Template xml ------> android.R.layout.simple_list_item_1,

3º id do view ----------> android.R.id.text1,

4º Array List com dados -> ArrayAlunos);

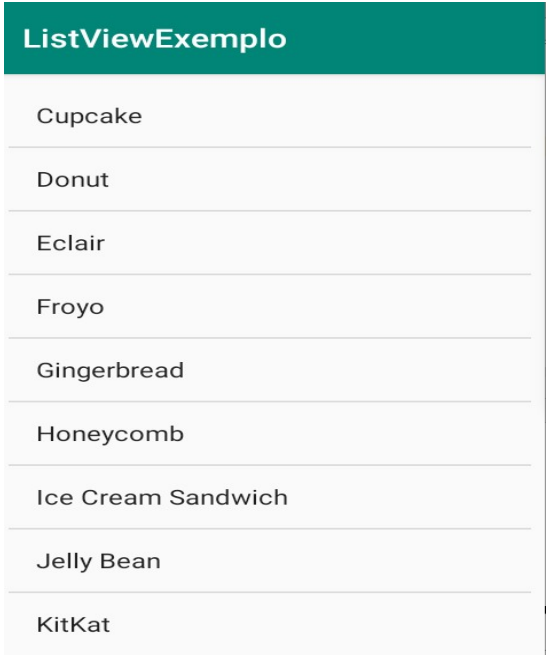
item_1.xml

```
<TextView
    android:id="@android:id/text1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_vertical"
    ... />
```

Atividade 1

Aplicativo Listagens Versões Android

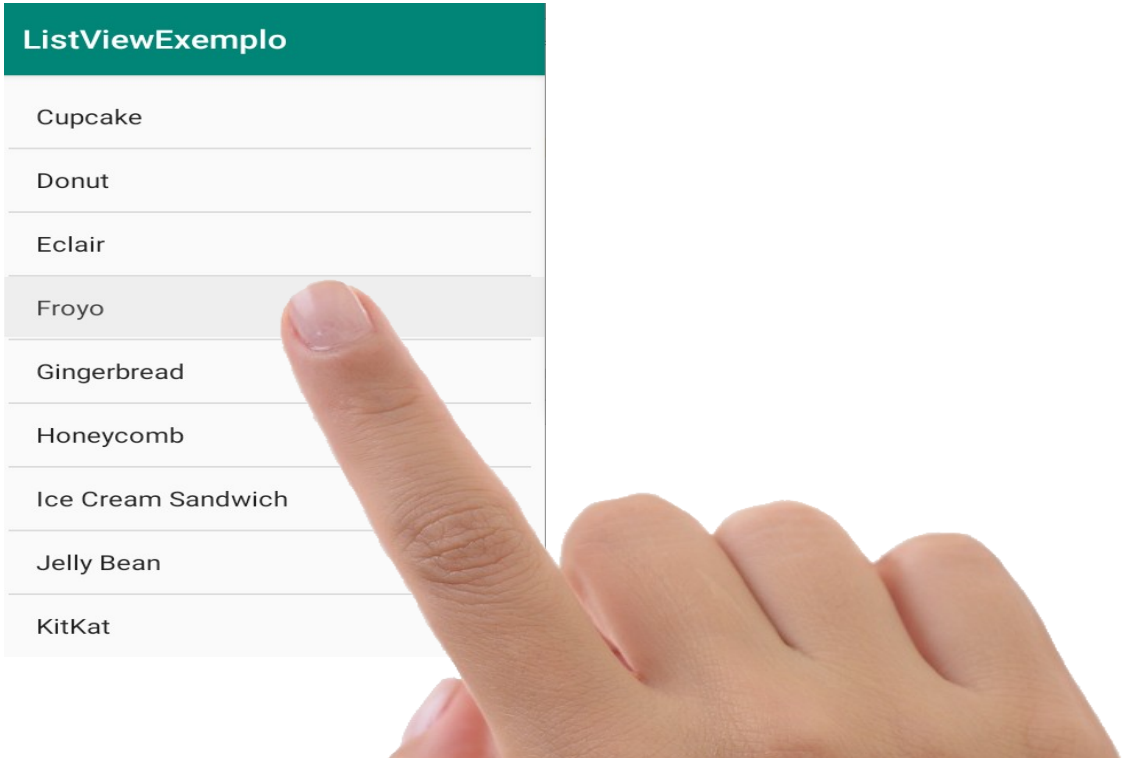
Construa um aplicativo que liste o nome de todas as versões do Android `"Cupcake"`, `"Donut"`, `"Eclair"`, `"Froyo"`, `"Gingerbread"` , `"Honeycomb"`, `"Ice Cream Sandwich"`, `"Jelly Bean"`, `"KitKat"`, `"Lollipop"`, `"Marshmallow"`, `"Nougat"` , `“oreo”`, `“pie”;`



click com *ListView*

Click ListView

Como o *listview* é um *view*, como podemos detectar um clique em um de seus itens?



?

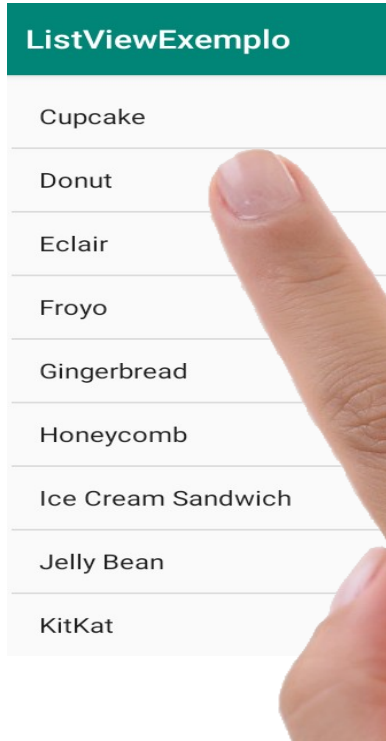
OnClick *ListView* simples

Podemos setar um *AdapterView*.*OnItemClickListener*, que recebe como parâmetros o *AdpaterView.OnItemClickListener*, o *view* que recebeu o evento e a posição do mesmo.

Definindo um ListenerOnItemClickListener

```
//Solicita ao listView que carregue o layout e dados passados no arrayAdapter e apresente
listView.setAdapter(adapter);

listView.setOnItemClickListener( type:AdapterView.OnItemClickListener() );
```



OnClick *ListView* simples

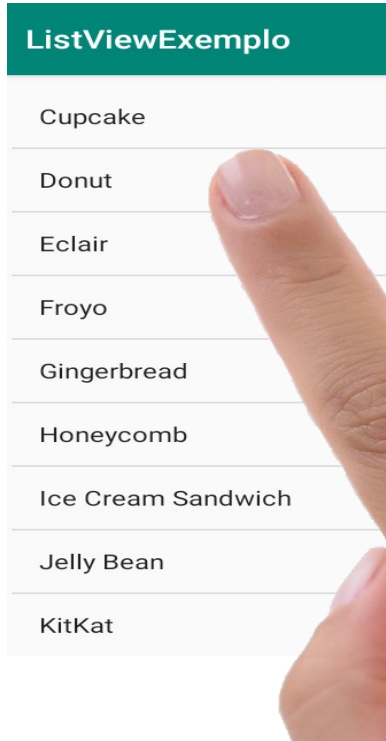
Podemos setar um *AdapterView.OnItemClickListener*, que recebe como parâmetros o *AdpaterView*, o *view* que recebeu o evento e a posição do mesmo

Definindo um ListenerOnItemClickListener

```
//Solicita ao listView que carregue o layout e dados passados no arrayAdapter e apresente
listView.setAdapter(adapter);

listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {

    }
});
```

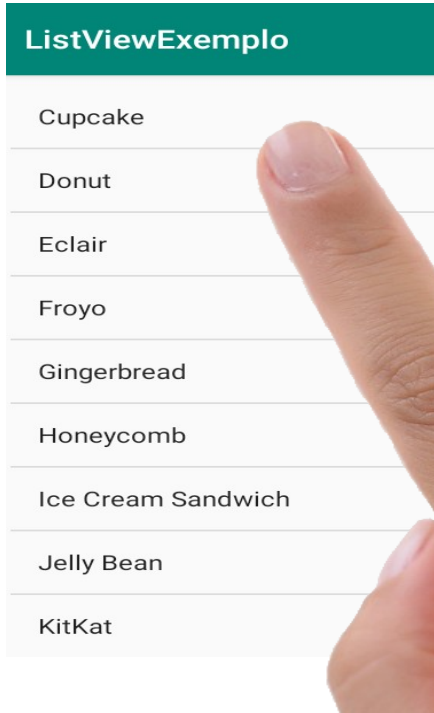


Como ouvir um *click* em um *ListView* Simples

Por fim, como no contexto em que se adiciona o *Listener*, em geral pode-se acessar o Array de dados, passado a listagem podemos recuperar dados do item clicado.

```
//Solicita ao listView que carregue o layout e dados passados no arrayAdapter e apresente
listView.setAdapter(adapter);

listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        String valorTexto = dados[position];
        Toast.makeText(getBaseContext(), valorTexto,Toast.LENGTH_LONG ).show();
    }
});
```



OnClick *ListView* simples

Podemos setar um *AdapterView.OnItemClickListener*, que recebe como parâmetros o *AdpaterView*, o *view* que recebeu o evento e a posição do mesmo

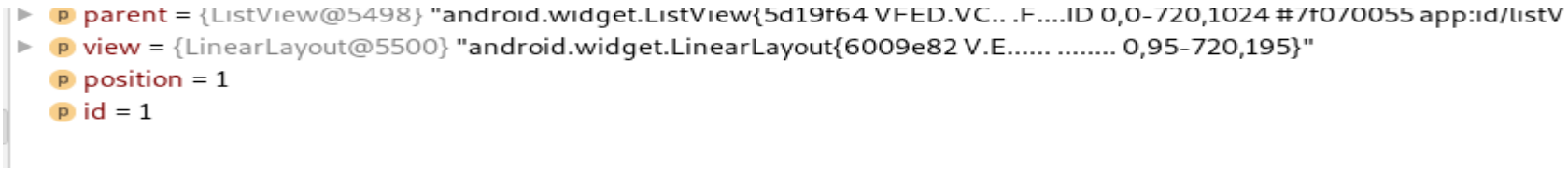
Definindo um ListenerOnItemClickListener

```
//Solicita ao listView que carregue o layout e dados passados no arrayAdapter e apresente
listView.setAdapter(adapter);

listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {

    }
});
```

Exemplo de parametros recebidos pelo ListenerOnItemClickListener



Atividade 2

Crie uma aplicação que apresente um *Toast* com o nome do elemento de dados que fora clicado no *ListView*.

Adapter Personalizado

ArrayAdapter especializado

Embora ter um adaptador pronto que permita a listagem de dados simples, o que torna prático o processo, infelizmente muitas vezes queremos lista de dados personalizadas como é o caso da agenda do celular, ou lista de contatos do *whatsapp*. Nestes casos necessitamos implementar o *template xml* a ser utilizado para exibição de cada item, e também um *Adapter* personalizado, que interligara os dados a exibição do *template*.

Fonte:Autor(2018)

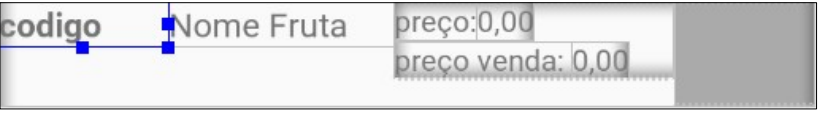
Esquema Gráfico de um *ArrayAdapter*

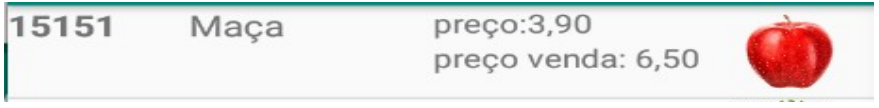
ArrayList de Frutas

Codigo: 15151
Nome: Maça
Preço: 3,90
PreçoVenda: 6,50
Imagem:R.drawer.maca

Adapter ?

Layout XML

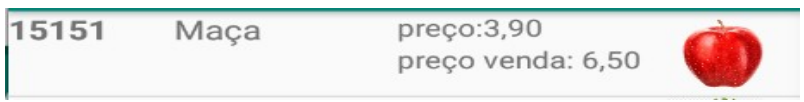




Array Adapter

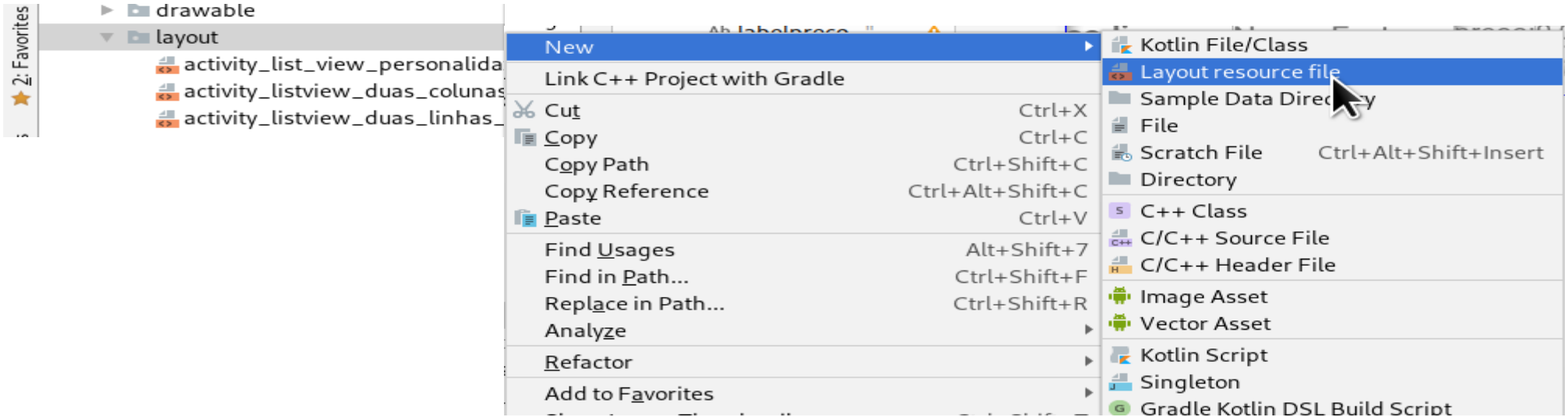
Embora a utilização do *ArrayAdapter* seja simplificada, como podemos apresentar mais de uma informação por item de uma listagem?

```
public static final Fruta MACA = new Fruta("Maça",
        "Avermelhada com traços verdes",
        R.drawable.maca,
        new BigDecimal(3.9),
        new BigDecimal(6.50),
        2922,
        new BigDecimal(4.5),
        15151);
```



Criação do *layout* para a listagem

Para construirmos uma listagem personalizada devemos criar um novo arquivo de layout(xml), servira como templete para cada item da listagem.



Definição do *layout* personalizado

- Após criarmos o arquivo de layout devemos definir os *views* do layout, lembrando que cada item da listagem terá uma instancia deste layout. Desta forma dificilmente ele terão altura do *ViewGroup* pai. Como no exemplo a seguir:

https://bitbucket.org/professorromulobeninca/modelo-listview/src/master/app/src/main/res/layout/template_listview_personalizada.xml

Exemplo de *Adapter* Personalizado

Como instanciar e utilizar um *Adapter* Personalizado

O *ListView* recebe, pelo método *setAdapter*, um objeto do tipo *ArrayAdapter* com as configurações para exibição da listagem.

Trecho de código com a chamada de um adapter personalizado e listagem do mesmo

```
FrutaListAdapter adapter = new FrutaListAdapter(
Contexto    ----->                  this,
Layout de cada item----->           R.layout.template_listview_personalizada,
ArrayList com dados ----->          listaFrutas);

listView.setAdapter(adapter);
```

Exemplo de *Adapter* Personalizado

Estrutura básica de um *AdapterPersonalizado*

Ao optarmos por construir uma lista personalizada devemos implementar um adaptador personalizado que deve “Adaptar os dados ao layout”. Esta classe adaptador deve estender de *ArrayAdapter*, e também devemos implementar o método construtor e o *getView*.

A assinatura do construtor pode receber como parâmetro o contexto, o layout a ser utilizado para cada linha do layout, e o arraylist com objetos que contenham os dados.

Trecho de código com a chamada de um adapter personalizado e listagem do mesmo

```
public  class FrutaListAdapter extends ArrayAdapter<Fruta> {
    public FrutaListAdapter ( Context context, int resource, ArrayList<Fruta> objects)  {
        super(context, resource, objects);

    }

    @Override
    public View getView(int position,  View convertView,  ViewGroup parent) {
       ...

       ...
      return  convertView;
    }
}
```

Exemplo de *Adapter* Personalizado

Código do construtor do *Adapter* Personalizado

O construtor do adaptador personalizado recebe o contexto e também o inteiro com o recurso só layout que será utilizado para cada item da lista, e também o *ArrayList* com dados. Como o contexto e o layout serão utilizados pelo método *view*, precisamos disponibiliza los aos demais métodos do *Adapter*, e para isso criamos atributos *mContext* e *mResource*.

Trecho de código com a chamada de um adapter personalizado e listagem do mesmo

```
public  class FrutaListAdapter extends ArrayAdapter<Fruta> {
    private Context mContext;
    private  int mResource;
    public FrutaListAdapter ( Context context, int resource, ArrayList<Fruta> objects)  {
        super(context, resource, objects);
        mContext=context;
        mResource=resource;
    }

    @Override
    public View getView(int position,  View convertView,  ViewGroup parent) {
       ...

       ...
      return  convertView;
    }
}
```

Exemplo de *Adapter* Personalizado

Parâmetros do método *getView* e sua funcionalidade.

O método getView é chamado pelo componente listView, e deve retornar layout de um item da lista já instanciado, por isso ele recebe como parâmetros:

- **Position**: um inteiro com a posição da lista a ser construída.
- ***ConvertView:*** referencia para o objeto view a ser disponibilizado.
- **ViewGroup:** ViewGroup em que o Listview esta contido.

Trecho de código do método getView do AdapterPersonalizado

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

Paramêtros passados ao getView para exibição de um item

```
p position = 0
p convertView = null
p parent = {ListView@5548} "android.widget.ListView{605b3f6 VFED.VC.. .....ID 0,0-1184,576 #7f070055 app:id/listView}"
```

```
@Override
public int getCount() {
   return mObjects.size();
}
```

Exemplo de Adapter Personalizado

Associação dos identificadores locais com objetos do *layout* inflado

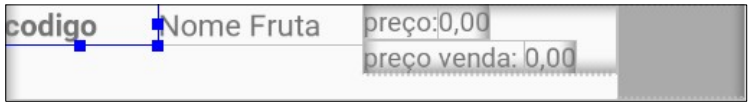
Inicialmente devemos Inflar uma *view* com o layout passado ao construtor como parâmetro, para isso utilizamos um objeto *LayoutManager* vinculado ao contexto e em seguida solicitamos ao método *infla-te* para carregar o layout recebido como recurso.

Trecho de código do método getView do AdapterPersonalizado

```
public View getView(int position,View convertView, ViewGroup parent) {
  LayoutInflater inflater = LayoutInflater.from(mContext);
  convertView= inflater.inflate(mResource, parent, false);


  return  convertView;
}
```

convertView-->



Exemplo de *Adapter* Personalizado

Associação dos identificadores locais com objetos do layout inflado

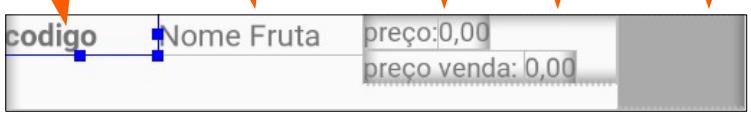
Após inflarmos o *view* a ser retornado, devemos associar os itens do layout inflado com o identificadores locais, para que os mesmos fiquem acessíveis, como apresentado no trecho de código.

Trecho de código do método getView do AdapterPersonalizado

```
public View getView(int position,View convertView, ViewGroup parent) {
  LayoutInflater inflater = LayoutInflater.from(mContext);
  convertView= inflater.inflate(mResource, parent, false);

  //Associando as variaveis do método a os identificadores de views do layout
  TextView tvCodigo = (TextView) convertView.findViewById(R.id.codigo);
  TextView tvNome = (TextView) convertView.findViewById(R.id.nome);
  TextView tvPreco =  (TextView) convertView.findViewById(R.id.preco);
  TextView tvPrecoVenda =(TextView) convertView.findViewById(R.id.preco_venda);
   ImageView imgView = (ImageView) convertView.findViewById(R.id.imageView);
  return  convertView;
}
```

convertView-->



Exemplo de *Adapter* Personalizado

Recuperando objeto do *ArrayList* de dados na posição a ser exibida.

Antes de atribuirmos os valores a cada *view* existente, temos que recuperar do *arraylist* o objeto que contem os dados a serem apresentados no item, para isto utilizamos o método *getItem*, passando a posição que atualmente está sendo construída da listagem.

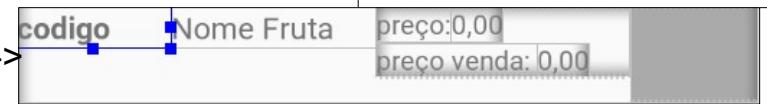
Trecho de código do método getView do AdapterPersonalizado

```
public View getView(int position,View convertView, ViewGroup parent) {
        LayoutInflater inflater = LayoutInflater.from(mContext);
        convertView= inflater.inflate(mResource, parent, false);
  //Associando as variaveis do método a os identificadores de views do layout
        ......
        Fruta fruta = getItem(position);
        NumberFormat nf = new DecimalFormat("#,###.00");
        tvCodigo.setText(Integer.toString(fruta.getCodigo()));
        tvNome.setText(fruta.getNome());
        tvPreco.setText(nf.format (fruta.getPreco()));
        tvPrecoVenda.setText(nf.format (fruta.getPreco_venda()));
        imgView.setImageResource(fruta.getImagem());
  return  convertView;
}
```

fruta-->

Codigo: 15151
Nome: Maça
Preço: 3,90
PreçoVenda: 6,50
Imagem:R.drawer.maca

convertView-->



Exemplo de Adapter Personalizado

Associando dados ao layout apontado por *coventView*

Com o objeto que contém os dados referenciado no identificador fruta, podemos atribuir os dados aos view do layout, como os métodos set respectivos a estes, como apresentado no trecho de código.

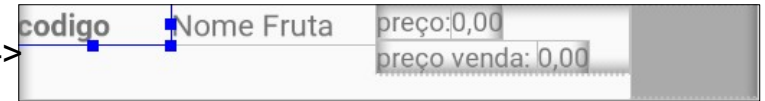
Trecho de código do método getView do AdapterPersonalizado

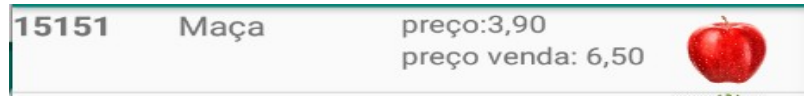
```
public View getView(int position,View convertView, ViewGroup parent) {
        LayoutInflater inflater = LayoutInflater.from(mContext);
        convertView= inflater.inflate(mResource, parent, false);
  //Associando as variaveis do método a os identificadores de views do layout
        ......
        Fruta fruta = getItem(position);
        NumberFormat nf = new DecimalFormat("#,###.00");
        tvCodigo.setText(Integer.toString(fruta.getCodigo()));
        tvNome.setText(fruta.getNome());
        tvPreco.setText(nf.format (fruta.getPreco()));
        tvPrecoVenda.setText(nf.format (fruta.getPreco_venda()));
        imgView.setImageResource(fruta.getImagem());
  return  convertView;
}
```

fruta-->

Codigo: 15151
Nome: Maça
Preço: 3,90
PreçoVenda: 6,50
Imagem:R.drawer.maca

convertView-->





Exemplo de Adapter Personalizado

Associando dados ao layout apontado por coverView

Por fim, o método *getview* é chamado uma vez para cada elemento do *arrayList*, assim construindo a exibição do listview.

```
public View getView(int position, View convertView, ViewGroup parent) {
    LayoutInflater inflater = LayoutInflater.from(mContext);
    convertView= inflater.inflate(mResource, parent, false);

    //Associando identificadores a objetos do layout
    TextView tvCodigo = (TextView) convertView.findViewById(R.id.codigo);
    TextView tvNome = (TextView) convertView.findViewById(R.id.nome);
    TextView tvPreco =  (TextView) convertView.findViewById(R.id.preco);
    TextView tvPrecoVenda =(TextView) convertView.findViewById(R.id.preco_venda);
    ImageView imgView = (ImageView) convertView.findViewById(R.id.imageView);

     Fruta fruta = getItem(position);
     NumberFormat nf = new DecimalFormat("#,###.00");
     tvCodigo.setText(Integer.toString(fruta.getCodigo()));
     tvNome.setText(fruta.getNome());
     tvPreco.setText(nf.format (fruta.getPreco()));
     tvPrecoVenda.setText(nf.format (fruta.getPreco_venda()));
     imgView.setImageResource(fruta.getImagem());
 return convertView;
}
```

Aplicativo desenvolvido com conhecimentos da aula

Repositório da aula:

https://bitbucket.org/professorromulobeninca/modelo-listview/src

Referências

- Lecheta, R. Google Android: Aprenda a criar aplicações para dispositivos móveis. 3a Ed. Novatec, 2013.
- http://developer.android.com/