02

LayoutInflater para melhorar nossa apresentação

Agora que temos imagens dos alunos vamos melhorar o layout de nossa listagem, customizando a aparência das linhas do `ListView` que apresenta a lista de alunos.

Atualmente a lista de alunos é vinculada ao `ListView` pelo adapter usando um layout do próprio Android:

```
ArrayAdapter<Aluno> adapter = new ArrayAdapter<Aluno>(this,
    android.R.layout.simple_list_item_1, alunos);
```

Esse layout encontra-se dentro do jar da `SDK` do Android de nossa aplicação.

O que queremos é deixar de usar o `android.R.layout.simple_list_item_1` e passar a usar um layout customizado nosso:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fundo"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">

    <ImageView android:id="@+id/foto"
        android:layout_width="50dp"
        android:layout_height="50dp"/>

    <TextView android:id="@+id/nome"
        android:textSize="22dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#000000" />

</LinearLayout>
```

Podemos criar o arquivo acima na pasta **res/layout** com o nome de `item.xml` .

Mas como explicar para o Android que a foto do aluno deve ser carregada e mostrada no `ImageView` , e que o nome do aluno deve ser colocado no `TextView` ?

Para essa tarefa precisaremos criar nosso próprio `Adapter` . Para isso basta herdarmos da classe `BaseAdapter` . Automaticamente somos obrigados a implementar quatro métodos: `getView, getItemId, getCount e getItem` :

```
public class ListaAlunosAdapter extends BaseAdapter {

    @Override
    public View getView(int posicao, View convertView, ViewGroup parent) {
        return ???
    }

    @Override
    public long getItemId(int posicao) {
        return ???;
    }

    @Override
    public Object getItem(int posicao) {
        return ???;
    }

    @Override
    public int getCount() {
        return ???;
    }
}
```

Através do método `getCount` precisamos explicar quantos itens vamos mostrar no `ListView` a partir desse método o Android consegue determinar o tamanho inicial do `ListView` (lembre-se que temos diversos tamanhos possíveis de tela!). Para saber o numero de alunos precisamos saber a quantidade de elementos da lista de alunos, portanto vamos recebê-la no construtor de nossa classe e então chamar o método `size` da lista.

```
public class ListaAlunosAdapter extends BaseAdapter {
    private final List<Aluno> alunos;

    public ListaAlunosAdapter(List<Aluno> alunos) {
        this.alunos = alunos;
    }

    //...

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

No método `getItemId` precisamos criar um identificador único para o item na tela. Usaremos o próprio valor do atributo `id` do aluno:

```
public long getItemId(int posicao) {
    return alunos.get(posicao).getId();
}
```

No método `getItem` precisamos retornar o objeto que se encontra na posicao selecionada de nosso `ListView` . Como desejamos que as posições da lista de alunos e do `ListView` sejam as mesmas, vamos apenas retornar o elemento da lista no índice `posicao` :

```
public Object getItem(int posicao) {
    return alunos.get(posicao);
}
```

Finalmente precisamos implementar o método `getView` o qual retorna uma `View` que será a linha apresentada na tela de nosso dispositivo. Queremos que a linha siga o layout criado anteriormente (`res/layout/item.xml`).

Mas como **preencher** nosso layout com os dados corretos? Para esse fim o Android disponibiliza o `LayoutInflater` : Uma classe capaz de criar um objeto para ser usado na view a partir de um xml. Para obtermos uma instância de um `LayoutInflater` basta chamarmos o método `getLayoutInflater` a partir de uma `Activity` . Para **inflarmos** um `Layout` basta invocarmos o método `inflate` passando o `id` do layout a ser inflado:

```
View view = activity.getLayoutInflater().inflate(R.layout.item, null);
```

Mas precisamos de um `LayoutInflater` em nosso `Adapter` que não é uma `Activity` , portanto vamos pedir nossa dependência também no construtor:

```
public class ListaAlunosAdapter extends BaseAdapter {
    private final List<Aluno> alunos;
    private final Activity activity;

    public ListaAlunosAdapter(Activity activity, List<Aluno> alunos) {
        this.activity = activity;
        this.alunos = alunos;
    }

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

        View view = activity.getLayoutInflater().inflate(R.layout.item, null);

        Aluno aluno = alunos.get(posicao);
        //precisamos colocar os dados do aluno na view aqui...

        return view;
    }

    //...
}
```

Agora basta colocarmos os dados do aluno no `layout` inflado por nós. Para localizar um elemento da view podemos invocar o conhecido método `findViewById` :

```
TextView nome = (TextView) view.findViewById(R.id.nome);
nome.setText(aluno.toString());
```

Para toda a alteração feita no ListView, deveremos recarregar o adapter para o mesmo.