Melhorias no layout

Após criar o AlunosAdapter, queremos que na tela do nosso aplicativo aparece a foto, o nome e o telefone dos alunos cadastrados na Agenda. Para fazer isto, voltaremos no AlunosAdapter.java e alterar o método getView(), para termos dois TextView e a imagem.

@Override  
public View getView(int position, View convertView, ViewGroup parent) {  
 TextView view = new TextView(context);  
 Aluno aluno = alunos.get(position);  
 view.setText(aluno.toString());  
  
 return view;  
}

Para adicionar outro TextView, poderíamos apenas duplicar as linhas do código referente a ele. Depois, poderíamos adicionar um ImageView. Este caminho funcionaria. No entanto, estaríamos misturando novamente a parte do layout no nosso código. No começo do curso, vimos que está não era uma boa ideia e por isso, temos o arquivo xml. Clicando na pasta res, seguida por layout, veremos que nós já criamos a representação de um item. Então, só temos que encontrar uma maneira de transformar o arquivo list\_item.xml em uma view.

Anteriormente, nós já trabalhamos com menu.xml, que continha a descrição de todos os itens que iriam entrar no nosso menu. Em seguida, conseguimos transformamos o xml em um Menu de fato. Queremos fazer o mesmo com o layout. O trabalho de transformar um xml no layout receberá o nome de inflar. Nós usaremos o LayoutInflater para realizar esta operação.

De volta ao AlunosAdapter.java, iremos tirar o pedaço referente ao TextView e deixaremos a linha do Aluno. Depois, iremos inflar o xml e transformá-lo em um View. Vamos criar a variável LayoutInflater. Após importar a classe, iremos dizer que ela tem o método estável from. Adicionaremos também o contexto (context).

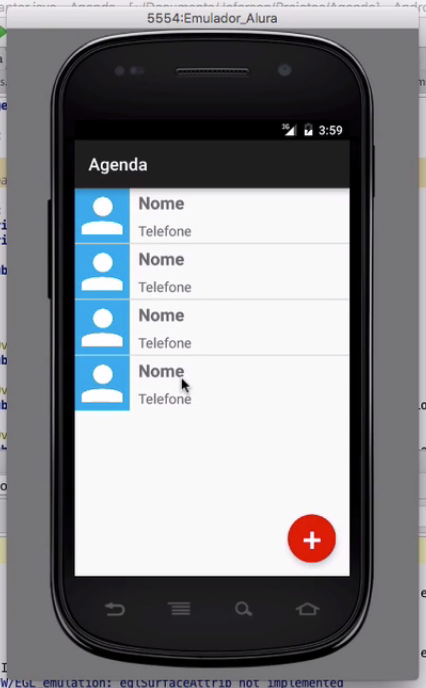
@Override  
public View getView(int position, View convertView, ViewGroup parent) {  
 Aluno aluno = alunos.get(position);  
  
 LayoutInflater inflater = LayoutInflater.from(context);  
  
 return view;  
}

Quando chamamos o método from do LayoutInflater, ele devolverá a instância de um LayoutInflater - especilista em inflar xml.

Em seguida, iremos chamaro método inflate e vamos informar o que ele precisará inflar: R.layout.list\_item. Após importá-la, precisaremos passar mais um parâmetro, usaremos null, por enquanto. Já criamos algo concreto, agora, iremos guardar a view em uma variável com o mesmo nome. Por fim, iremos retorná-la no getView. Nosso código ficará assim:

@Override  
public View getView(int position, View convertView, ViewGroup parent) {  
 Aluno aluno = alunos.get(position);  
  
 LayoutInflater inflater = LayoutInflater.from(context);  
 View view = inflater.inflate(R.layout.list\_item,null);  
  
 return view;  
}

Vamos rodar nossa aplicação no emulador para conferir os resultados da alteração.



Para cada item da lista, construímos uma view inflando o xml. No entanto, os dados dos alunos ainda não estão preenchidos.

Faremos isto no AlunosAdapter.java. Agora, pegaremos uma referência do campo de "Nome", por exemplo, e incluíremos valor dentro.

@Override  
public View getView(int position, View convertView, ViewGroup parent) {  
 Aluno aluno = alunos.get(position);  
  
 LayoutInflater inflater = LayoutInflater.from(context);  
 View view = inflater.inflate(R.layout.list\_item,null);  
  
 campoNome.setText(aluno.getNome());  
  
 return view;  
}

Porém, ainda não temos o campoNome. Precisaremos buscá-lo usando findViewById e passaremos como referência o R.id.item\_nome. O Android Studio não conseguirá encontrar este método, porque não sabe de onde ele vem.

O findViewById é um método que herdamos da Activity. Então, iremos chamar a activity e em seguida, fazer o findViewById.

activity.findViewById(R.id.item\_nome);   
campoNome.setText(aluno.getNome());

Mas se fizermos um findViewById e procurarmos por item\_nome, o Android irá procurar dentro do activity\_lista\_alunos.xml, onde não encontraremos o item. Ele irá nos devolver um null. Nós gostaríamos de fazer uma busca no list\_item.xml, porque nele iremos encontrar o item\_nome.

<LinearLayout   
 android:layout\_width="wrap\_content"  
 android:layout\_heigth="match\_parent"  
 android:orientatio="vertical"  
 android:gravity="center\_vertical">  
  
 <TextView  
 android:"wrap\_content"  
 android:"wrap\_content"  
 android:id="@+id/item\_nome"  
 android:text="Nome"  
 android:textSize="20sp"  
 android:textStyle="bold"  
 android:paddingLeft="10dp"/>  
\...

Precisaremos mudar a referência do activity no AlunosAdapter.java. Na verdade, aview que representa o item acabamos de inflar, com a foto e o nome do aluno. Logo, o findViewById que queremos fazer é na view e não na activity.

@Override  
public View getView(int position, View convertView, ViewGroup parent) {  
 Aluno aluno = alunos.get(position);  
  
 LayoutInflater inflater = LayoutInflater.from(context);  
 View view = inflater.inflate(R.layout.list\_item,null);  
  
 view.findViewById(R.id.item\_nome);  
 campoNome.setText(aluno.getNome());  
  
 return view;  
}

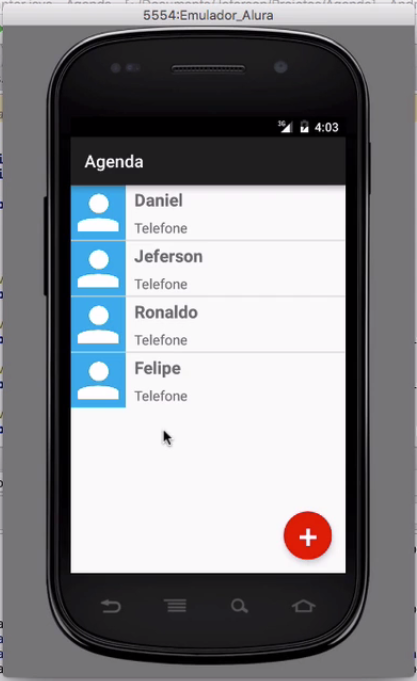
Vamos atribuir uma variável para o findViewById que chamaremos de campoNome e vamos definir que ela é do tipo TextView.

TextView campoNome = (TextView) view.findViewById(R.id.item\_nome);

Fizemos o cast no segundo view. Com isto preenchemos o campo de "Nome". Nosso código ficou assim:

@Override  
public View getView(int position, View convertView, ViewGroup parent) {  
 Aluno aluno = alunos.get(position);  
  
 LayoutInflater inflater = LayoutInflater.from(context);  
 View view = inflater.inflate(R.layout.list\_item,null);  
  
 TextView campoNome = (TextView) view.findViewById(R.id.item\_nome);  
  
 return view;  
}

Vamos testar o aplicativo novamente e ver o resultado.



Conseguimos fazer as alterações e já visualizamos os nomes corretos dos alunos. Agora, iremos adicionar os telefones e as fotos.

Iremos fazer um processo semelhante ao que fizemos com o campoNome. Primeiramente vamos buscar na view, e especificar o item: R.id.item\_telefone. Depois, iremos atribuir uma variável que chamaremos de campoTelefone. definiremos que se trata de uma TextView. Logo abaixo, vamos fazer o campoTelefone.setText e passar como parâmetro getTelefone.

TextView campoTelefone = (TextView) view.findViewById(R.id.item\_telefone);  
campoTelefone.setText(aluno.getTelefone());

Faremos agora o mesmo com o item\_nome, com a diferença que dessa vez, iremos trabalhar com uma ImageView.

ImageView campoFoto = view.findViewById(R.id.tem\_foto);

Para colocar a imagem no campoFoto, usaremos o método setImageBitmap, que irá nos pedir um bitmap. Iremos copiar uma parte do código referente à imagem, no formularioHelper.java. Nele criamos um método carregaImagem que colocava um bitmap dentro do ImageView. Iremos trabalhar com o seguinte trecho do formularioHelper.java:

public void carregaImagem(String caminhoFoto) {  
 if (caminhoFoto != null) {  
 Bitmap bitmap = BitmapFactory.decodeFile(caminhFoto);  
 Bitmap bitmapReduzido = Bitmap.createScaledBitmap(bitmap, 300, 300, true);  
 campoFoto.setImageBitmap(bitmapReduzido);  
 campoFoto.setScaleType(ImageView.ScaleType.FIT\_XY);  
 campoFoto.setTag(caminhoFoto);  
 }  
 }

Iremos copiar algumas linhas deste pedaço. Mas não poderemos usar o método como ele está, porque ele está vinculado com o FormularioHelper. No caso, o caminhoFoto virá do aluno.

ImageView campoFoto = (ImageView) view.findViewById(R.id.item\_foto);  
String caminhoFoto = aluno.getCaminhoFoto();  
if (caminhoFoto != null) {  
 Bitmap bitmap = BitmapFactory.decodeFile(caminhoFoto);  
 Bitmap bitmapReduzido = Bitmap.createScaledBitmap(bitmap, 300, 300, true);  
 campoFoto.setImageBitmap(bitmapReduzido);  
 campoFoto.setScaleType(ImageView.ScaleType.FIT\_XY);  
 campoFoto.setTag(caminhoFoto);  
}   
  
return view;

Faremos uma pequena alteração nos parâmetros do tamanho da imagem.

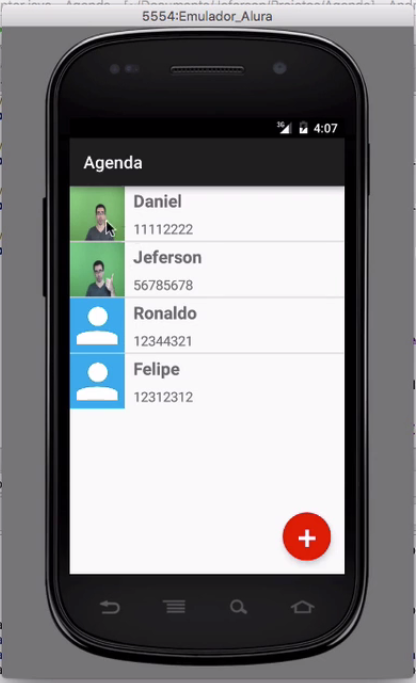
Bitmap bitmapReduzido = Bitmap.createScaledBitmap(bitmap, 100, 100, true);

Iremos eliminar a linha do setTag, que guarde ocaminhoFoto, porque não precisaremos ler isto depois.

Então, com as novas alterações o nosso código ficou assim:

@Override  
public View getView(int position, View convertView, ViewGroup parent) {  
 Aluno aluno = alunos.get(position);  
  
 LayoutInflater inflater = LayoutInflater.from(context);  
 View view = inflater.inflate(R.layout.list\_item,null);  
  
 TextView campoNome = (TextView) view.findViewById(R.id.item\_nome);  
  
 TextView campoTelefone = (TextView) view.findViewById(R.id.item\_telefone);  
 campoTelefone.setText(aluno.getTelefone());  
  
 ImageView campoFoto = (ImageView) view.findViewById(R.id.item\_foto);  
 String caminhoFoto = aluno.getCaminhoFoto();  
 if (caminhoFoto != null) {  
 Bitmap bitmap = BitmapFactory.decodeFile(caminhoFoto);  
 Bitmap bitmapReduzido = Bitmap.createScaledBitmap(bitmap, 100, 100, true);  
 campoFoto.setImageBitmap(bitmapReduzido);  
 campoFoto.setScaleType(ImageView.ScaleType.FIT\_XY);  
 campoFoto.setTag(caminhoFoto);  
 }   
  
 return view;  
  
 }  
 }

Com as modificações, quando testarmos o nosso aplicativo, veremos a foto e o telefone aparecerem juntamente com a lista de nome dos alunos.



Para finalizar, iremos comentar alguns parâmetros que não utilizamos.

LayoutInflater inflater = LayoutInflater.from(context);  
View view = inflater.inflate(R.layout.list\_item,null);

Nós precisaremos alterar o null da linha acima. Neste caso, o list\_item está sendo inflado e ele precisa construir uma view para aparecer na tela. Com o valor nulo, não passamos informações extras. Mas este parâmetro indica aonde a nossa lista deve entrar, "quem é o pai".

Então, por exemplo, vamos testar fazer modificações no list\_item. Iremos alterar os valores de layout\_width e layout\_heigth.

<?xml version="1.0" enconding="utf-8"?>  
<LinearLayout   
xmlns:android="http://schemas.android.com/apk/res/android"  
 android:orientation="horizontal"  
 android:layout\_width="match\_parent"  
 android:layout\_heigth="100dp"  
  
 <ImageView  
 android:layout\_width="64dp"  
 android:layout\_heigth="64dp"  
 android:src="@drawable/person"  
 android:id='@+id/item\_foto"/>

Se quisermos ter este comportamento, com os novos valores, teremos que alterar o null da view. Quem sabe onde o item da lista irá entrar, será a listView, que também chamará o getView. A listView já passou um parâmetro no ViewGroup parent.

@Override  
public View getView(int position, View convertView, ViewGroup parent) {  
 Aluno aluno = alunos.get(position);  
  
}

Temos a posição que a lista nos passa (int position), e ela se passa como parâmetro: "estou no parent". Será o parent que iremos usar no inflate.

LayoutInflater inflater = LayoutInflater.from(context);  
View view = inflater.inflate(R.layout.list\_item, parent);

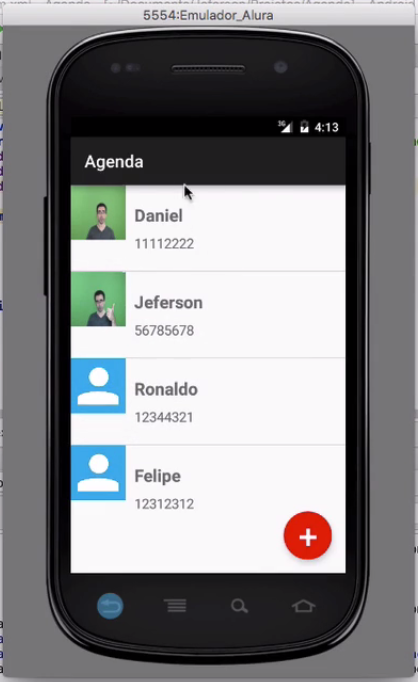
Porém, o inflate irá querer inflar o xml e contruir o item, em seguida, irá criar a view e adicioná-la no parent. O problema estará quando devolvermos a view. Como a lista pediu para colocarmos a view dentro dela, ela pegará o resultado e tentará reinseri-la novamente. Quando tentamos colocar dois elementos exatamente iguais dentro de um componente do Android, iremos receber uma Exception.

Para não termos este tipo de problemas, iremos adicionar mais um parâmetro no inflate não se antecipar e colocar a view diretamente.

LayoutInflater inflater = LayoutInflater.from(context);  
View view = inflater.inflate(R.layout.list\_item, parent, false);

Usaremos o parent como referência, mas não iremos colocar a view na lista. Então, só quando retornarmos a view ela será inserida na lista.

Agora, iremos fazer um novo teste. Vamos ver se ela irá manter os itens com o tamanho pré-definido.



Cada item ficou com um tamanho de 100dp.

No entanto, não queremos estas novas dimensões no nosso layout. Vamos voltar ao list\_item.xmle deixar o código como estava antes.

<?xml version="1.0" enconding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:orientation="horizontal"  
 android:layout\_width="match\_parent"  
 android:layout\_heigth="wrap\_content"

Finalizando, temos um último parâmetro que não usamos: convertView.

Se tivéssemos uma lista com 1 milhão de itens, o aplicativo teria que passar muito tempo instanciando para colocá-los na tela. No entanto, mesmo nestes casos, não conseguiríamos exibir todos os itens na tela do celular. Logo, não será necessário que o Android fique instanciando todos estes itens e já colocá-los na lista. Mas quando quiséssemos rolar a lista, os itens precisariam ser instanciados.

Então, o que de fato acontece é que o Android instancia um número maior de itens que aparecem na lista, assim, quando a rolamos, sempre terá um contato de prontidão. Na hora em que estes acabarem, os itens rolados para cima serão jogados para baixo.

Quando um Android nos passa uma convertView, significa que a view já está pronta e só precisamos modificar os valores que estão nela, antes de mostrá-las na tela. Isto faz com que o inflate não sejá necessário sempre - o que é muito bom para a performance dos aplicativos. Para que isto aconteça, vamos inserir um if.

LayoutInflater inflater = LayoutInflater.from(context);  
if (convertView == null) {  
 View view = inflater.inflate(R.layout.list\_item, parent, false);  
}

Agora, só queremos fazer o inflate, se a convertView for igual a nulo.

Mas se declararmos a view dentro, não poderemos utilizá-la fora. Vamos alterar o if, e adicionar a view fora.

LayoutInflater inflater = LayoutInflater.from(context);  
  
View view = convertView;  
if (view == null) {  
 view = inflater.inflate(R.layout.list\_item, parent, false);  
}

A partir de agora, nossa Adapter irá reaproveitar as views.

@Override  
public View getView(int position, View convertView, ViewGroup parent) {  
 Aluno aluno = alunos.get(position);  
  
 LayoutInflater inflater = LayoutInflater.from(context);  
  
 View view = convertView;  
 if (view == null) {  
 view = inflater.inflate(R.layout.list\_item, parent, false);  
 }  
  
 TextView campoNome = (TextView) view.findViewById(R.id.item\_nome);  
 campoNome.setText(aluno.getNome());  
  
 TextView campoTelefone = (TextView) view.findViewById(R.id.item\_telefone);  
 campoTelefone.setText(aluno.getTelefone());  
  
 ImageView campoFoto = (ImageView) view.findViewById(R.id.item\_foto);  
 String caminhoFoto = aluno.getCaminhoFoto();  
 if (caminhoFoto != null) {  
 Bitmap bitmap = BitmapFactory.decodeFile(caminhoFoto);  
 Bitmap bitmapReduzido = Bitmap.createScaledBitmap(bitmap, 100, 100, true);  
 campoFoto.setImageBitmap(bitmapReduzido);  
 campoFoto.setScaleType(ImageView.ScaleType.FIT\_XY);  
 campoFoto.setTag(caminhoFoto);  
 }   
  
 return view;  
  
}  
}

As mudanças que fizemos não irão alterar nada no código e o aplicativo deve continuar a funcionar como esperamos. Mas nos casos em que temos muitos itens, o Android não irá ficar instanciando todos. Isto irá melhorar bastante a experiência do usuário.

Com isto, encerramos a parte de melhorar o layout da lista e trabalharmos com o adapter.