Criando o projeto

*Neste capítulo faremos apenas uma revisão do conteúdo de outros treinamentos de Android do Alura e não será apresentado nenhuma novidade. Caso você já esteja confortável com o conteúdo desses outros treinamentos, poderá seguir direto para o* [*capítulo 2*](https://cursos.alura.com.br/course/android-chat/task/22366) *onde há um* link *para você baixar o projeto base para continuarmos o treinamento. Bons estudos :)*

Vamos começar criando o nosso projeto no Android Studio, clicando em ***Start a new Android Studio project***, com o nome de **ichat-alura**. Colocaremos a **API 15** como versão mínima e pediremos que uma *activity* seja criada, a ***Empty Activity***, deixamos o nome padrão e concluímos a criação do projeto.

Como foi solicitado, no momento da criação do projeto, a criação também da *activity*, nos são exibidos a classe **MainActivity** e o arquivo **activity\_main.xml**, que é o layout da *activity*. Só que o **activity\_main.xml** vem apenas com o TextView, que para nós não importa muito, já que no layout queremos listar e enviar mensagens. Então o primeiro passo e modificarmos o layout para se adequar ao projeto e às nossas necessidades.

## **Ajustando o layout**

Que componente do Android faz essa listagem de objetos, textos para nós? O **ListView**, logo o usaremos ao invés do **TextView**. E para a lista ocupar a tela toda, utilizaremos a diretiva **match\_parent** tanto na largura quanto na altura:

<ListView  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent" />

Na parte de baixo da tela, teremos os componentes, a imagem de avatar do usuário, o campo de texto para enviar as mensagens e o botão de envio, todos eles um ao lado do outro, de forma linear. Quem faz essa organização de componentes de forma linear no Androis? O **LinearLayout**. Ele também ocupará a largura inteira do dispositivo, logo utilizaremos **match\_parent** no seu width, e na height utilizaremos wrap\_content:

<LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content">  
  
</LinearLayout>

E adicionamos os componentes já citados:

<LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content">  
  
 <ImageView  
 android:layout\_width=""  
 android:layout\_height="" />  
  
 <EditText  
 android:layout\_width=""  
 android:layout\_height="" />  
  
 <Button  
 android:layout\_width=""  
 android:layout\_height="" />  
  
</LinearLayout>

Queremos que o campo de texto tenha o dobro da largura do botão, mas como conseguimos fazer isso no Android? Para isso existe o **layout\_weight**, que permite a nós darmos pesos a alguma dimensão do nosso componente. Logo, se queremos que a largura do campo de texto seja o dobro da do botão, o seu peso será **2** e o do botão será **1**. Mas agora, como queremos **desabilitar** a largura e deixar essa responsabilidade com o **layout\_weight**, vamos colocar o **layout\_width** de ambos os componentes como **0dp**:

<LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content">  
  
 <ImageView  
 android:layout\_width=""  
 android:layout\_height="" />  
  
 <EditText  
 android:layout\_weight="2"  
 android:layout\_width="0dp"  
 android:layout\_height="" />  
  
 <Button  
 android:layout\_weight="1"  
 android:layout\_width="0dp"  
 android:layout\_height="" />  
  
</LinearLayout>

Para a altura, vamos colocar o suficiente para mostrar o componente, logo utilizaremos **wrap\_content**. E para a imagem, tanto a largura quanto a altura terão **50dp**:

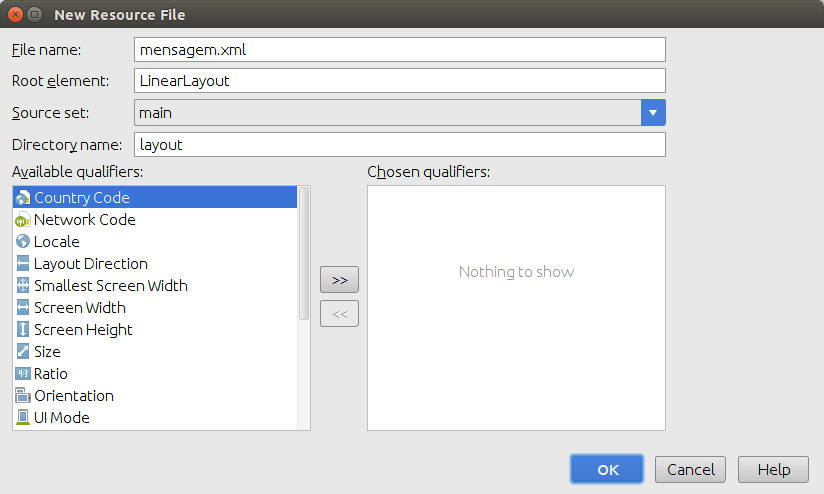
<LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content">  
  
 <ImageView  
 android:layout\_width="50dp"  
 android:layout\_height="50dp" />  
  
 <EditText  
 android:layout\_weight="2"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content" />  
  
 <Button  
 android:layout\_weight="1"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content" />  
  
</LinearLayout>

Podemos reparar que os componentes ficaram no **topo** do dispositivo, mas queremos que eles fiquem na parte de **baixo**, certo? Como estamos utilizando o **RelativeLayout**, podemos pedir para o **LinearLayout** se alinhar na parte de baixo da tela, passando o valor **true** para o atributo **alignParentBottom**:

<LinearLayout  
 android:layout\_alignParentBottom="true"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content">  
  
 <ImageView  
 android:layout\_width="50dp"  
 android:layout\_height="50dp" />  
  
 <EditText  
 android:layout\_weight="2"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content" />  
  
 <Button  
 android:layout\_weight="1"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content" />  
  
</LinearLayout>

## **Layout das mensagens**

Como já estamos mexendo com o layout, vamos aproveitar para criar o layout das mensagens, da **ListView**. Dentro da pasta **res/layout/**, vamos criar o arquivo **mensagem.xml**:



Nesse arquivo, por enquanto só teremos um **TextView**:

<TextView  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content" />

Mais à frente adicionaremos mais componentes nele!

## **Criando o Adapter**

Agora precisamos criar a classe que conseguirá traduzir cada elemento da estrutura de dados em elementos visuais, e esse é o papel do ***adapter***. Então vamos criar a classe **MensagemAdapter**. Crie-a em um pacote específico para os *adapters*. Podemos aproveitar e criar um pacote exclusivo para as *activitys*, e transferir a classe **MensagemActivity** para ele.

Para efetivamente transformar a classe que criamos em um ***adapter***, ela precisa herdar da classe abstrata **BaseAdapter**. Se ela é uma classe abstrata, precisamos implementar os seus métodos abstratos:

public class MensagemAdapter extends BaseAdapter {  
  
 @Override  
 public int getCount() {  
 return 0;  
 }  
  
 @Override  
 public Object getItem(int i) {  
 return null;  
 }  
  
 @Override  
 public long getItemId(int i) {  
 return 0;  
 }  
  
 @Override  
 public View getView(int i, View view, ViewGroup viewGroup) {  
 return null;  
 }  
}

Falando um pouco sobre esse métodos, o **getCount** retorna o tamanho da lista; o **getItem** retorna o objeto, baseado na posição recebida; o **getItemId** retorna um número único pra cada elemento; e o **getView** retorna o layout do elemento, baseado na posição recebida.

Para implementar todos esses métodos, precisamos de uma **lista de mensagens**, então vamos criá-la:

public class MensagemAdapter extends BaseAdapter {  
  
 private List<Mensagem> mensagens;  
  
 // métodos omitidos  
  
}

Não podemos nos esquecer de importar a classe List do pacote java.util e como ainda não temos a classe **`Mensagem**, vamos criá-la em um pacote específico para os **modelos**:

public class Mensagem {  
  
}

Agora vamos criar um construtor para a classe **MensagemAdapter**, que recebe uma lista de mensagens:

public class MensagemAdapter extends BaseAdapter {  
  
 private List<Mensagem> mensagens;  
  
 public MensagemAdapter(List<Mensagem> mensagens) {  
 this.mensagens = mensagens;  
 }  
  
 // métodos omitidos  
  
}

Com a lista em mãos, conseguimos implementar o restante dos métodos. No **getCount**, retornamos o tamanho da lista chamando seu método **size**; no **getItem**, vamos modificá-lo para retornar uma **Mensagem** (e não mais Object), retornamos o elemento baseado na posição recebida, chamando o método **get**; e no **getItemId**, podemos retornar a própria posição do elemento, já que ela não se repete:

public class MensagemAdapter extends BaseAdapter {  
  
 private List<Mensagem> mensagens;  
  
 public MensagemAdapter(List<Mensagem> mensagens) {  
 this.mensagens = mensagens;  
 }  
  
 @Override  
 public int getCount() {  
 return mensagens.size();  
 }  
  
 @Override  
 public Mensagem getItem(int i) {  
 return mensagens.get(i);  
 }  
  
 @Override  
 public long getItemId(int i) {  
 return i;  
 }  
  
 // getView comentado  
}

Dentro do método **getView** precisamos transformar numa **View** o xml contendo o layout do item da lista, o arquivo **mensagem.xml**, para isso usamos o método **inflate**:

@Override  
public View getView(int i, View view, ViewGroup viewGroup) {  
 View linha = inflate(...);  
}

Então, o que queremos fazer é poder *inflar* o **mensagem.xml** em uma instância de **View**, mas como conseguimos referenciar esse arquivo na nossa classe? Utilizando a classe **R** e o tipo de recurso, **layout**. O método ainda recebe um *parent*, no nosso caso será o próprio **viewGroup** e um booleano que representa se queremos anexar esse item ao *parent*, como não queremos passaremos o valor **false**:

@Override  
public View getView(int i, View view, ViewGroup viewGroup) {  
 View linha = inflate(R.layout.mensagem, viewGroup, false);  
}

Mas a quem pertence esse método inflate? Ao **LayoutInflater**. Pegaremos-o através do seu *getter*:

@Override  
public View getView(int i, View view, ViewGroup viewGroup) {  
 View linha = getLayoutInflater().inflate(R.layout.mensagem, viewGroup, false);  
}

Já esse *getter* pertence à classe **Activity**. Como não temos acesso a ela, vamos criar um atributo e recebê-lo através do construtor:

public class MensagemAdapter extends BaseAdapter {  
  
 private List<Mensagem> mensagens;  
 private Activity activity;  
  
 public MensagemAdapter(List<Mensagem> mensagens, Activity activity) {  
 this.mensagens = mensagens;  
 this.activity = activity;  
 }  
  
 @Override  
 public View getView(int i, View view, ViewGroup viewGroup) {  
 View linha = activity.getLayoutInflater().inflate(R.layout.mensagem, viewGroup, false);  
 }  
  
 // outros métodos omitidos  
}

Agora falta pegarmos a referência dos componentes que ficarão na tela. Por enquanto temos somente o **TextView**, que é o texto da nossa mensagem. Para isso, podemos pegá-lo através do seu **id**, mas antes precisamos criá-lo no **mensagem.xml**. Vamos criar o id **tv\_texto**, e como queremos **criar** o id, precisamos passar também o **@+id/**:

<TextView  
 android:id="@+id/tv\_texto"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content" />

Agora podemos pegar o texto da linha através do seu id, chamando o método **findViewById** e passando novamente a classe **R** com o tipo de recurso, **id**:

@Override  
public View getView(int i, View view, ViewGroup viewGroup) {  
  
 View linha = activity.getLayoutInflater().inflate(R.layout.mensagem, viewGroup, false);  
  
 TextView texto = linha.findViewById(R.id.tv\_texto);  
  
  
}

Esse método retorna uma **View**, mas como estamos trabalhando com **TextView**, precisamos fazer um *cast*:

@Override  
public View getView(int i, View view, ViewGroup viewGroup) {  
  
 View linha = activity.getLayoutInflater().inflate(R.layout.mensagem, viewGroup, false);  
  
 TextView texto = (TextView) linha.findViewById(R.id.tv\_texto);  
  
 return linha;  
}

Agora precisamos preencher o texto, acessando a mensagen na posição recebida por parâmetro pelo getView. Para isso, podemos utilizar o próprio método da classe, **getItem**:

@Override  
public View getView(int i, View view, ViewGroup viewGroup) {  
  
 View linha = activity.getLayoutInflater().inflate(R.layout.mensagem, viewGroup, false);  
  
 TextView texto = (TextView) linha.findViewById(R.id.tv\_texto);  
  
 Mensagem mensagem = getItem(i);  
  
 return linha;  
}

Com a mensagem em mãos, precisamos atribuir o seu texto à **TextView**:

@Override  
public View getView(int i, View view, ViewGroup viewGroup) {  
  
 View linha = activity.getLayoutInflater().inflate(R.layout.mensagem, viewGroup, false);  
  
 TextView texto = (TextView) linha.findViewById(R.id.tv\_texto);  
  
 Mensagem mensagem = getItem(i);  
  
 texto.setText(mensagem.getText());  
  
 return linha;  
}

Como o método getText não existe na classe **Mensagem**, vamos criá-lo e fazê-lo retornar o texto, então vamos também criar esse atributo:

public class Mensagem {  
  
 private String texto;  
  
 public String getText() {  
 return texto;  
 }  
}

No próximo vídeo continuaremos com a implementação do método **getView**, então nos vemos lá!