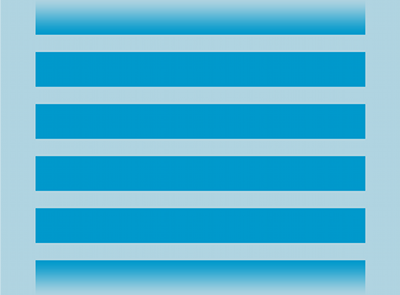
Unidade 6

Aula 1 - Listas

Estamos chegando na reta final da apresentação dos componentes principais do Android. A maioria dos apps tem em suas interfaces uma estrutura de lista, para suportar um conteúdo dinâmico. Vamos ver nesta aula como criar listas no layout do nosso app.

## Listas simples

ListView é um ViewGroup que apresenta uma lista de itens “scrollável” (deslizável). Os itens da lista são inseridos automaticamente na lista usando um Adapter, que puxa o conteúdo de uma fonte, como um array ou uma pesquisa a um banco de dados e converte cada item resultante em uma View que é colocada dentro da lista.



Vamos direto a um exemplo.

Para exemplificar, vamos começar a desenvolver nossa tela de e-mails recebidos (Inbox). Nesta tela devemos inserir uma lista que receberá dinamicamente os e-mails usuário. Para conteúdo didático, não iremos sincronizar com um servidor de email de verdade, pois existe um custo e uma complexidade desnecessária para o objetivo deste curso, mas você terá total capacidade para implementar um app com serviço de e-mail de verdade se você quiser dar continuidade a este app após o término deste curso. Para exemplificar, os dados serão estáticos e pré-definidos.

Dando início a esta prática, descompacte o arquibo **Unidade\_6\_-\_Aula\_1\_-\_Exemplo\_1.zip**, e abra no seu Android Studio o projeto contido nele. Para garantir o funcionamento correto desta nova versão do MailList, delete o aplicativo antigo (algumas atualizações feitas no SharedPreferences podem prejudicar o uso do app).

Execute o projeto e veja que algumas coisas mudaram:

* Para você sair da tela de Login, você deve primeiro entrar na tela **Cadastro** e preencher os campos. Após ter realizado o cadastro, volte na tela de Login e insira o mesmo email cadastrado.
* A tela Inbox agora possui o layout **CoodinatorLayout**, para mostrar o **Float Action Button**, que é um botão flutuante que está sendo aderido nos apps atuais.

Então vamos lá. Siga os seguintes passos:

1. A primeira coisa que devemos fazer é inserir um **ListView** no layout **fragment\_inbox**, então abra o arquivo **fragment\_inbox.xml** e localize o ViewGroup **RelativeLayout.** Insira o ListView dentro deste ViewGroup.

<**ListView**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

**android:id="@+id/inbox\_listView"**

**android:layout\_alignParentTop="true"**

**android:layout\_alignParentStart="true"**

/>

1. Seu seu **fragment\_inbox.xml** deve corresponder com a seguinte listagem (existe uma identificação no arquivo indicando o local que você deve inserir um ListView):

*<?***xml version="1.0" encoding="utf-8"***?>*

<**android.support.design.widget.CoordinatorLayout xmlns:android="http://schemas.android.com/apk/res/android"**

**android:layout\_width="match\_parent"**

**android:layout\_height="match\_parent"**

**android:fitsSystemWindows="true"**>

  <**RelativeLayout**

**android:layout\_width="match\_parent"**

**android:layout\_height="match\_parent"**>

      <**ListView**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

**android:id="@+id/inbox\_listView"**

**android:layout\_alignParentTop="true"**

**android:layout\_alignParentStart="true"**

/>

  </**RelativeLayout**>

  <**android.support.design.widget.FloatingActionButton**

**android:id="@+id/fab"**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

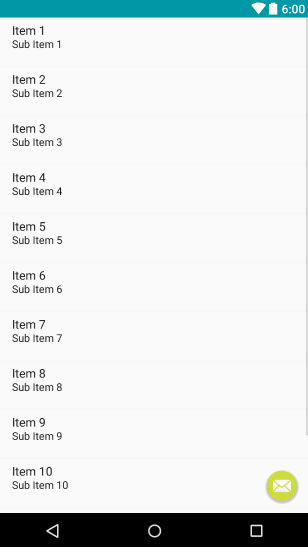
**android:layout\_gravity="bottom|end"**

**android:layout\_margin="@dimen/fab\_margin"**

**android:src="@android:drawable/ic\_dialog\_email"** />

</**android.support.design.widget.CoordinatorLayout**>

Agora que temos nosso ListView adicionado no Layout, você pode ver no **preview** do arquivo o seguinte visual:



1. Agora precisamos inserir itens nesta lista e para nível de demonstração vamos inserir uma simples lista de Strings. Abra o arquivo **InboxFragment.java** e insira o array de itens que aparecerão na lista. Você pode inserir dados diferentes deste exemplo. Fique à voltade.

String[] **emails** = {**"Email 1"**, **"Email 2"**, **"Email 3"**, **"Email 4"**, **"Email 5"**, **"Email 6"**, **"Email 7"**, **"Email 8"**, **"Email 9"**, **"Email 10"**, **"Email 11"**, **"Email 12"**, **"Email 13"**, **"Email 14"**, **"Email 15"**, **"Email 16"**, **"Email 17"**, **"Email 18"**, **"Email 19"**, **"Email 20"**,**"Email 21"**, **"Email 22"**, **"Email 23"**, **"Email 24"**, **"Email 25"**};

1. Você precisa fazer o binding da ListView, então insira o seguinte atributo:

@BindView(R.id.***inbox\_listView***) ListView **listView**;

1. Para inserir os itens do array criado na lista, insira o seguinte método:

**public void** initListView() {

  ArrayAdapter<String> arrayAdapter = **new** ArrayAdapter<String>(

**this**.getActivity().getApplicationContext(),

          android.R.layout.***simple\_list\_item\_1***,

   android.R.id.***text1***,

**emails**

);

**listView**.setAdapter(arrayAdapter);

}

**Entendendo o código:** O conteúdo das ListViews não são renderizados apenas com um array simples, então é necessário um ArrayAdapter, que é um tipo de array especial. Com um ArrayAdapter o ListView infere que seu conteúdo deve ser inserido em um TextView. Para inicializar este Adapter, precisamos de um contexto, um arquivo de layout para o texto a ser exibido (neste caso usamos um layout predefinido do Android).

1. Faça a chamada do método **initListView()** no método onCreateView() e veja o resultado:



Não era o que você esperava, não é mesmo? Este formato que utilizamos renderiza um TextView com letras brancas. Para ajustar isto, vamos logo criar nosso TextView personalizado.

1. Crie um novo arquivo de layout chamado **listview\_textview.xml**.
2. Apague todo conteúdo deste novo arquivo e insira o seguinte código:

*<?***xml version="1.0" encoding="utf-8"***?>*

<**TextView xmlns:android="http://schemas.android.com/apk/res/android"**

**android:id="@+id/tv"**

**android:textColor="@color/primary\_text"**

**android:padding="5sp"**

**android:layout\_width="fill\_parent"**

**android:singleLine="true"**

**android:gravity="center\_vertical"**

**android:layout\_height="56dp"**/>

1. Agora volte para a classe **InboxFragment** e no método **initListView()**, substitua os parâmetros para que o método corresponda a listagem abaixo:

**public void** initListView() {

  ArrayAdapter<String> arrayAdapter = **new** ArrayAdapter<String>(

**this**.getActivity().getApplicationContext(),

          R.layout.***listview\_textview***,

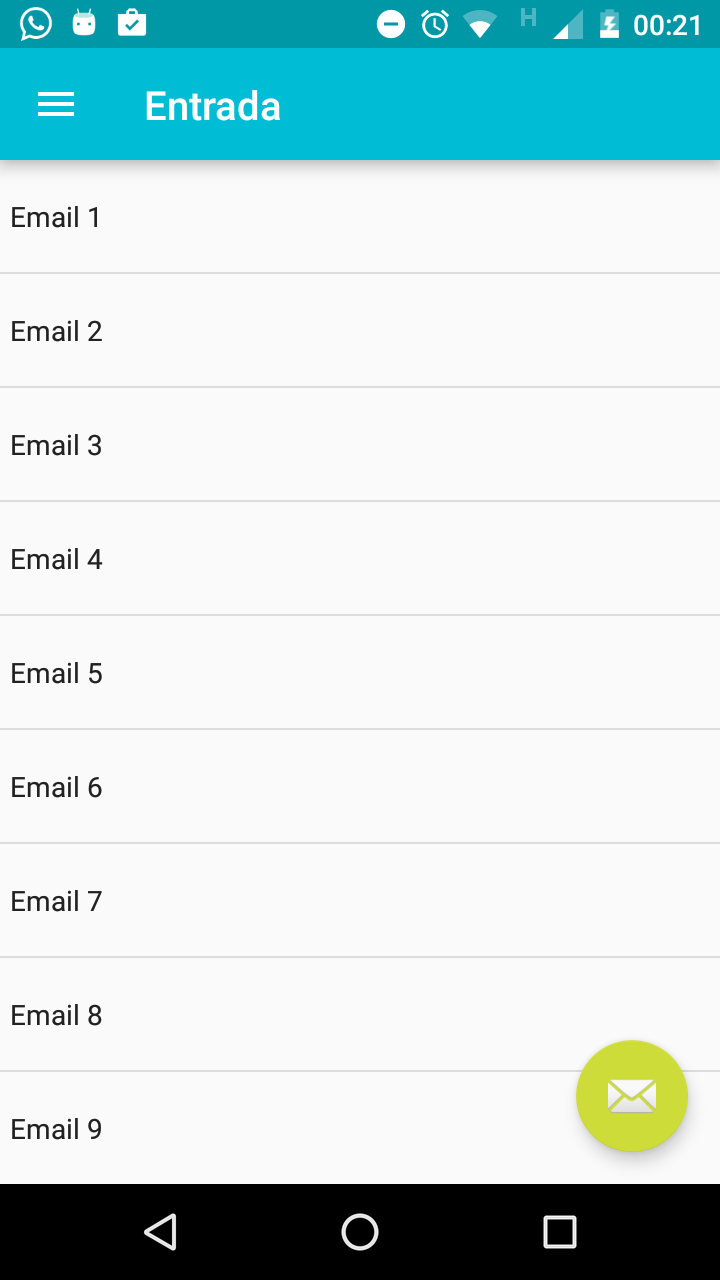
**emails**

);

**listView**.setAdapter(arrayAdapter);

}

1. Execute o projeto e veja o resultado:



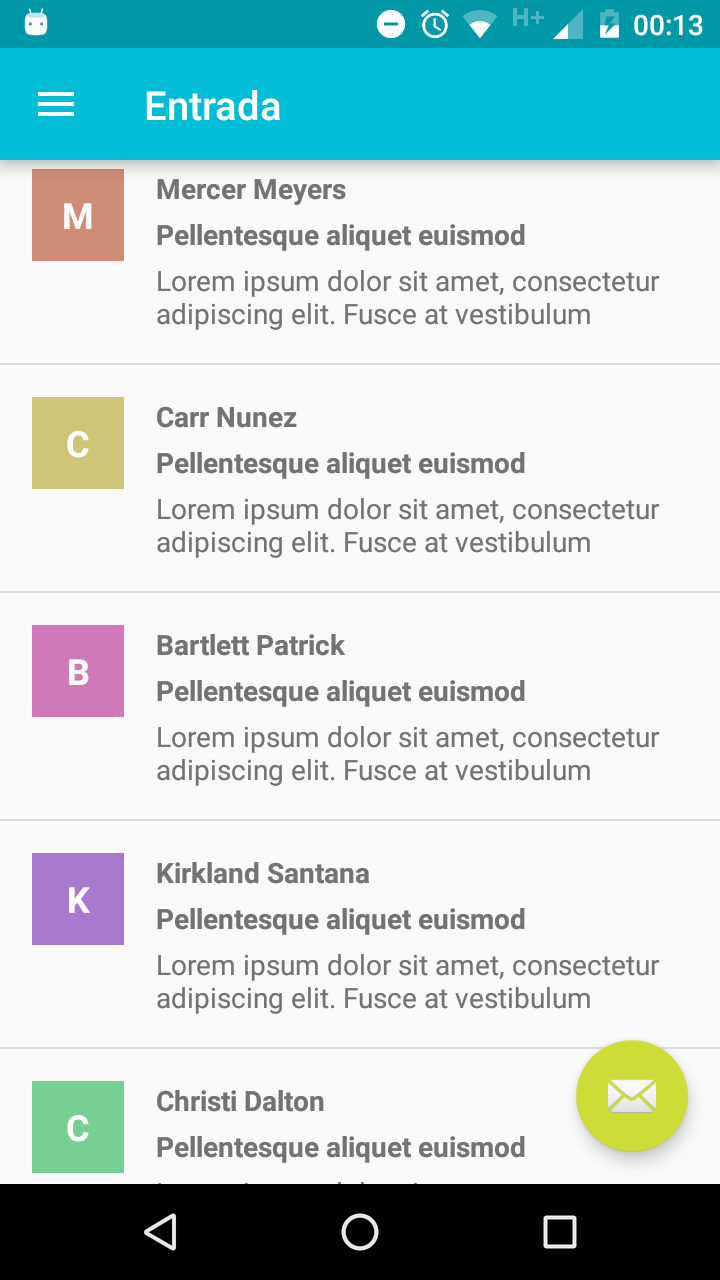
## Listas personalizadas

As listas simples serão úteis em pouquíssimos casos, e em grande parte das vezes sentiremos a necessidade de utilizar listas com células (chamamos de célula a linha de uma lista) personalizadas, que atenderão uma demanda mais robusta de design. Criar listas com células personalizadas é uma atividade corriqueira no desenvolvimento de aplicativos móveis e não poderíamos deixar de mencionar neste curso.

Para criar uma lista personalizada, precisamos de três itens:

* Um layout para a célula;
* Um adapter customizado para a lista, e
* Para atendermos uma boa qualidade de codificação, devemos vincular um **Model** ao adapter.

Ao final desta aula nossa lista ficará parecida como:

**

Vamos começar pelo mais fácil. Criar o Model de e-mail.

### Model Email

Nesta etapa vamos criar um Model para e-mail, e isto facilitará na manipulação dos dados que preencherão nossa lista. Para você ambientar nesta situação, o nosso intuito neste assunto é criar uma lista que possua mais do que uma informação. Queremos mostrar na célula o nome da pessoa, uma parte da mensagem e a foto da pessoa. Ainda não manipularemos a foto, pois deixaremos para a unidade seguinte, mas utilizaremos uma técnica conhecida, que é colocar a inicial do nome da pessoa, como na figura acima.

Vamos então criar nosso model que possua as seguinte informações:

* Remetente
* Destinatário
* Mensagem
* Assunto

Use os padrões aprendidos no curso de POO (nomes em inglês e encapsulamento). Temos abaixo, a classe pronta:

**public class** Email {

**private** String **subject**;

**private** String **from**;

**private** String **to**;

**private** String **message**;

**public** Email(String subject, String from, String message) {

**this**.**subject** = subject;

**this**.**from** = from;

**this**.**message** = message;

  }

**public** String getSubject() {

**return subject**;

  }

**public void** setSubject(String subject) {

**this**.**subject** = subject;

  }

**public** String getFrom() {

**return from**;

  }

**public void** setFrom(String from) {

**this**.**from** = from;

  }

**public** String getTo() {

**return to**;

  }

**public void** setTo(String to) {

**this**.**to** = to;

  }

**public** String getMessage() {

**return message**;

  }

**public void** setMessage(String message) {

**this**.**message** = message;

  }

}

Feito o Model. Vamos criar o layout para a célula da nossa lista.

### Layout da célula

Para criar o Layout é simples. Nossa proposta é exibir o nome do remetente, sua foto, o assunto do email e a mensagem. Caso o remetente não possua foto, exibiremos a inicial dele como na imagem acima.

Para criar um layout para a célula siga os seguintes passos:

1. Crie um arquivo na pasta layouts chamado **row.xml**
2. Remova todo o código XML e insira este:

*<?***xml version="1.0" encoding="utf-8"***?>*

<**RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"**

**android:layout\_width="match\_parent"**

**android:layout\_height="wrap\_content"**

**android:padding="@dimen/activity\_general\_margin"**

>

  <**View**

**android:id="@+id/contact\_picture"**

**android:layout\_width="46dp"**

**android:layout\_height="46dp"**

**android:gravity="center\_horizontal"**

**android:layout\_marginRight="16dp"**

**android:background="@color/colorAccent"**/>

  <**TextView**

**android:id="@+id/contact\_initial"**

**android:layout\_width="46dp"**

**android:layout\_height="46dp"**

**android:gravity="center"**

**android:layout\_marginRight="16dp"**

**android:textColor="@color/icons"**

**android:textStyle="bold"**

**android:textSize="18sp"**

**android:text="L"**

/>

  <**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"**

**android:layout\_width="match\_parent"**

**android:layout\_height="wrap\_content"**

**android:orientation="vertical"**

**android:layout\_toRightOf="@+id/contact\_picture"**>

      <**TextView**

**android:id="@+id/row\_contact"**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

**android:text="Contato"**

**android:textStyle="bold"**

**android:maxLines="1"**

**android:layout\_marginBottom="4dp"**/>

      <**TextView**

**android:id="@+id/row\_subject"**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

**android:text="Assunto"**

**android:textStyle="bold"**

**android:maxLines="1"**

**android:layout\_marginBottom="4dp"**/>

      <**TextView**

**android:id="@+id/row\_message"**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

**android:text="Mensagem"**

**android:maxLines="2"**

/>

  </**LinearLayout**>

</**RelativeLayout**>

Dê uma boa olhada no código para que você entenda o que está sendo feito e tenha ciência dos **ids** mencionados nos passos seguintes.

Com isto já temos o suficiente para o layout da célula de e-mail. Vamos implementar o Adapter.

## Padrão de Projeto - Adapter

Antes de implementarmos nosso próprio adapter, vamos entender como ele funciona. O nome Adapter vem do padrão de projeto de mesmo nome, então vamos falar um pouco dele.

### Objetivos

* O Adapter converte a interface de uma classe em uma outra interface que o cliente está esperando. Ele deixa classes que possuem interfaces incompatíveis trabalharem juntas.
* Encapsula uma classe existente em uma nova interface.
* Dá o poder de deixar componentes antigos compatíveis em um novo sistema.

### Problema

Um componente “fora da estante” oferece uma gama de funcionalidades atraentes que você gostaria de reusar, mas a sua “visão do mundo” não é compatível com a filosofia e arquitetura de seu sistema.

### Discussão

Apesar de o prezarmos ao máximo por nos oferecer tantos benefícios, o reuso sempre tem sido doloroso e ilusório ao programador. Uma exemplo para esta tribulação é quando estamos desenvolvendo algo novo mas precisamos utilizar algum componente antigo. Sempre vai existir alguma coisa que não se dará bem com o componente antigo, mas o padrão Adapter está aqui para resolver isto.



Podemos assimilar este padrão com um adaptador de tomada. Quem nunca teve um problema com a compatibilidade de tomadas, como quando nos mudamos para uma nova casa e a tomada daquela máquina de costurar que sua mãe tanto ama não entra na tomada. Para resolvermos isto utilizamos um adaptador.

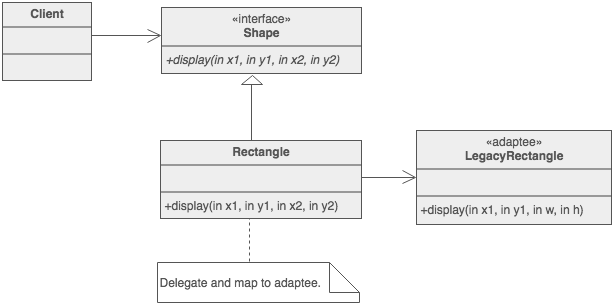


Adaptar é criar uma abstração intermediária que traduz, ou mapeia, o componente antigo para o novo sistema. O Cliente chama métodos no objeto Adapter no qual os redireciona apropriadamente para dentro do componente legado. Esta estratégia pode ser implementada tanto com uma especialização (herança) quando com uma agregação.

Adapter também possui funções como um “invólucro” ou modificador de uma classe existente. Isto garante novas e diferentes formas de tradução e ponto de vista da classe.

### Estrutura

Abaixo, método **display()** do componente legado Retangle espera receber “x, y, w, h” como parâmetro. Mas o cliente quer passar “x e y do canto esquerdo” e “x e y do canto direito” (ou “x1, y1, x2, y2”) como parâmetro. Esta incongruência pode ser reconciliada adicionando um nível a mais nesta relação, como um objeto Adapter.



O Adapter também pode ser visto como um “invólucro”:

### Exemplo Android

Segundo a documentação oficial do Android, um Adapter atua como uma ponte entre um AdapterView e os dados subjacentes desta view. Um AdapterView, por sua vez é uma view cuja sua subview é determinado pelo Adapter. Este Adapter providencia à view o acesso a um conjunto de dados e também é responsável por criar uma View para cada item deste conjunto de dados.

Analisando pelo ListView, Adapter cria uma view para cada filho da ListView (célula) e associa um conjunto de dados a ela (a informação em si a ser exibida). Ele atua como adaptador neste caso pois é impossível que uma ListView possa ser tão flexível com layouts variados de suas células a ponto de não precisar de um adaptador para manipulá-las, então neste caso o BaseAdapter oferece um conjunto de métodos conhecidos por uma ListView a serem implementados.

## Voltando à lista personalizada…

Agora que entendemos como funcionam os Adapters, vamos criar nosso próprio.

1. Crie um arquivo chamado **InboxAdapter.java** no pacote **Model** (o nome e local do arquivo fica a seu critério, sinta-se a vontade para dar os nomes que quiser).
2. Queremos um Adapter, certo? Então com o arquivo InboxAdapter aberto faça com que esta classe estenda **BaseAdapter.**

**public class InboxAdapter extends BaseAdapter**

1. O Android Studio apresentará alguns erros. Isto significa que existem alguns métodos abstratos de BaseAdapter que precisam ser implementados. Faça isto; implemente estes métodos, como na listagem abaixo:

**public class** InboxAdapter **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**;

  }

}

Vamos explicar cada um destes métodos a seguir.

1. Vamos modificar a implementação destes métodos. Antes disso, adicione dois atributos auxiliares. Um Context, pois precisará do contexto para inflar a célula, e o array de dados (do tipo Email) que preencherá a lista. Veja:

Context **context**;

Email[] **data**;

1. Crie o seguinte método Construtor:

**public** InboxAdapter(Context context, Email[] data) {

**this**.**context** = context;

**this**.**data** = data;

*inflater* = (LayoutInflater) context

          .getSystemService(Context.***LAYOUT\_INFLATER\_SERVICE***);

}

**Entendendo o código:** Nas duas primeiras linhas, fazemos a inicialização básica de context e data. Na linha subsequente, obtemos o layoutInflater através de um contexto (que será o contexto do InboxFragment).

1. Agora sim, vamos implementar os métodos abstratos, da nossa maneira. Localize o método **getCount()** e substitua seu conteúdo, como na listagem abaixo:

**Entendendo o código:**  O método getCount() é responsável por informar ao ListView (que atribuir este adapter) quantas células ele deverá renderizar. Neste exemplo pedimos para que renderize a quantidade de emails que estiver contido no array **data.**

1. O segundo método a se implementar é o **getItem().** Faça da seguinte forma:

**Entendendo o código:**  Dado uma posição da lista, o método getItem() pega o conjunto de dados que é referente a View que estará nesta posição.

1. O próximo método não é importante para nós, neste exemplo, mas é obrigatório. No método **getItemId(),** precisamos identificar cada item em exclusivo. Para isto faça igual na seguinte listagem:

@Override

**public** View getView(**int** position, View view, ViewGroup parent) {

*// Se a view for nula, infla do layout*

**if** (view == **null**)

      view = *inflater*.inflate(R.layout.***row***, **null**);

*// A ButterKnife pode ser um pouco problemática em um Adapter, então colete as Views de row do modo tradicional*

TextView brief = (TextView) view.findViewById(R.id.***row\_message***);

  TextView from = (TextView) view.findViewById(R.id.***row\_contact***);

  TextView subject = (TextView) view.findViewById(R.id.***row\_subject***);

  TextView initial = (TextView) view.findViewById(R.id.***contact\_initial***);

  View picture = (View) view.findViewById(R.id.***contact\_picture***);

*// Define os respectivos textos dado uma position*

brief.setText(**data**[position].getMessage());

  from.setText(**data**[position].getFrom());

  subject.setText(**data**[position].getSubject());

  initial.setText(**data**[position].getFirstLetter());

*// Define um número randômico de 0 a 360*

Random r = **new** Random();

**int** hue = r.nextInt(360 - 0) + 0;

*// Determina a cor de fundo para a "TextView de ausência de fotos" (contact\_initial)*

**float**[] hsv = {(**float**)hue, 42.0f/100.0f, 81.0f/100.0f};

  picture.setBackgroundColor(Color.*HSVToColor*(hsv));

**return** view;

}

**Entendendo o código:** Apesar de grande, este método é o mais simples e importante dos Adapters. Através dele, inflamos o layout criado **row.xml**. Após isso, é hora de fazer bind nos itens do layout e atribuir seus devidos valores. Por fim vem a definição de cores para termos quadrados com a inicial das pessoas coloridos.

1. Terminamos nosso Adapte. Para finalizar faltam Instanciar e atribuir este adapter a uma ListView. Criamos um array com muitos emails fake. Entre na sua **EmailFragment** e adicione o seguinte atributo:

Email[] **emails** = {

**new** Email(**"Pellentesque aliquet euismod"**, **"Casey Hamilton"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Sutton Thompson"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Boone Murray"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Sophia Owens"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Bethany Dodson"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Jerry Morgan"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Dickson Snider"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Charity Perry"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Cleo Joseph"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Freeman Keller"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Mercer Meyers"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Carr Nunez"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Bartlett Patrick"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Kirkland Santana"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Christi Dalton"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Sears Velasquez"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Humphrey Crane"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Letitia Warner"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Fannie Suarez"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Luann Harmon"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Janette House"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Stephens Mejia"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Robbie Noel"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Mays Norton"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**),

**new** Email(**"Pellentesque aliquet euismod"**, **"Solis Everett"**, **"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**)

};

1. Por fim, encontre o método **initListView**() e troque por:

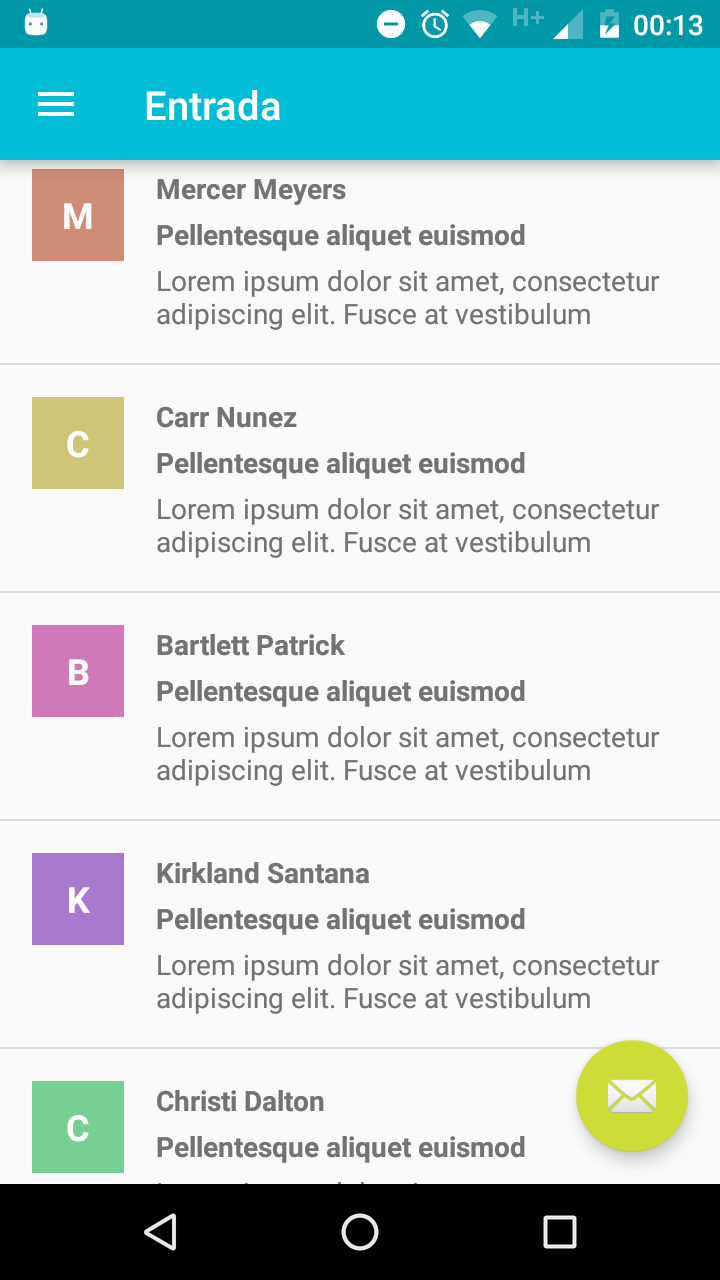
**public void** initListView() {

**listView**.setAdapter(**new** InboxAdapter(**this**.getContext(), **emails**));

}

1. Execute o aplicativo e veja o resultado.

Após ter executado estes passos, você terá a seguinte tela:



Na próxima aula daremos os retoques finais ao visual do nosso aplicativo.

# Aula 2 - Arte final

Aula 2 - Arte final

Apesar do nome, esta aula não é a final. Iremos mostrar alguns componentes que facilitarão e darão uma beleza a mais em seus aplicativos.

Nesta aula mostraremos como criar Views circulares (deveria ser uma coisa simples, mas no Android não é) para inserirmos na nossa lista de email. Também apresentaremos a você um pouco da tipografia do Android e uma biblioteca facilitar a manipulação das fontes dos textos do app. No fim, daremos uma breve pincelada nas animações do Android.

Tudo isto enriquece nosso app e o deixa mais atraente para o usuário. Então vamos lá.

## Drawables arredondados

Tanto em programação web quanto em iOS, deixar os cantos de uma view arredondados ou criar formas circulares é algo muito trivial com uma ou três linhas de código isso é possível. Por algum infortúnio que desconhecemos, no Android esta tarefa pode se tornar uma dor de cabeça. Para nossa sorte existem bibliotecas que auxiliam nisso, prontas para serem utilizadas. Nos iremos conhecer a biblioteca **TextDrawable.**

### TextDrawable

A TextDrawable não chega a ser tão conhecida como a ButterKnife tão pouco como as bibliotecas de suporte da Google. Mas para te mostrar que com uma pequena pesquisa, a maioria (senão todos) os nossos problemas com Android já foram resolvidos por alguém e está disponível na internet.

Fazer uso de bibliotecas externas é fundamental para o processo de desenvolvimento mobile, pois elas reduzem drasticamente o tempo de desenvolvimento do seu app, já que se elas não existissem, você é quem teria que implementar. Vamos então utilizar e ao mesmo tempo testar a biblioteca TextDrawable. Iremos criar os círculos com as iniciais dos remetentes dos e-mails. Siga os seguintes passos.

1. No arquivo **row.xml,** localize logo no início do código os seguinte elementos:

<**View**

**android:id="@+id/contact\_picture"**

**android:layout\_width="46dp"**

**android:layout\_height="46dp"**

**android:gravity="center\_horizontal"**

**android:layout\_marginRight="16dp"**

**android:background="@color/colorAccent"**/>

<**TextView**

**android:id="@+id/contact\_initial"**

**android:layout\_width="46dp"**

**android:layout\_height="46dp"**

**android:gravity="center"**

**android:layout\_marginRight="16dp"**

**android:textColor="@color/icons"**

**android:textStyle="bold"**

**android:textSize="18sp"**

**android:text="L"**

/>

1. Apague-os e no lugar deles você colocará simplesmente esta **ImageView:**

<**ImageView**

**android:id="@+id/contact\_picture"**

**android:layout\_width="46dp"**

**android:layout\_height="46dp"**

**android:layout\_marginRight="16dp"**/>

1. Sua row.xml ficará da seguinte forma:

*<?***xml version="1.0" encoding="utf-8"***?>*

<**RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"**

**android:layout\_width="match\_parent"**

**android:layout\_height="wrap\_content"**

**android:padding="@dimen/activity\_general\_margin"**

>

  <**ImageView**

**android:id="@+id/contact\_picture"**

**android:layout\_width="46dp"**

**android:layout\_height="46dp"**

**android:layout\_marginRight="16dp"**/>

  <**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"**

**android:layout\_width="match\_parent"**

**android:layout\_height="wrap\_content"**

**android:orientation="vertical"**

**android:layout\_toRightOf="@+id/contact\_picture"**>

      <**TextView**

**android:id="@+id/row\_contact"**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

**android:text="Contato"**

**android:textStyle="bold"**

**android:maxLines="1"**

**android:layout\_marginBottom="4dp"**/>

      <**TextView**

**android:id="@+id/row\_subject"**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

**android:text="Assunto"**

**android:textStyle="bold"**

**android:maxLines="1"**

**android:layout\_marginBottom="4dp"**/>

      <**TextView**

**android:id="@+id/row\_message"**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

**android:text="Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**

**android:maxLines="2"**

/>

  </**LinearLayout**>

</**RelativeLayout**>

1. Lembra do arquivo **build.gradle (Module: app)**? Entre nele e adicione a seguinte dependência:

compile **'com.amulyakhare:com.amulyakhare.textdrawable:1.0.1'**

1. Seu arquivo **build.gradle (Module: app)** deverá estar da seguinte maneira:

apply **plugin**: **'com.android.application'**

apply **plugin**: **'android-apt'**

android {

  compileSdkVersion 24

  buildToolsVersion **"24.0.2"**

defaultConfig {

      applicationId **"br.com.pearson.maillist"**

minSdkVersion 19

      targetSdkVersion 24

      versionCode 1

      versionName **"1.0"**

}

  buildTypes {

      release {

          minifyEnabled **false**

proguardFiles getDefaultProguardFile(**'proguard-android.txt'**), **'proguard-rules.pro'**

}

  }

}

dependencies {

  compile fileTree(**dir**: **'libs'**, **include**: [**'\*.jar'**])

  testCompile **'junit:junit:4.12'**

*// ButterKnife*

apt **'com.jakewharton:butterknife-compiler:8.4.0'**

compile **'com.jakewharton:butterknife:8.4.0'**

*// Design*

compile **'com.android.support:appcompat-v7:24.2.0'**

compile **'com.android.support:design:24.2.0'**

compile **'com.android.support:support-v4:24.2.0'**

*// TextDrawer*

compile **'com.amulyakhare:com.amulyakhare.textdrawable:1.0.1'**

}

1. Agora os últimos passos serão no arquivo **InboxAdapter.java**. Entre nele e localize o método **getView()**. Substitua o conteúdo deste método para corresponder com o seguinte código:

@Override

**public** View getView(**int** position, View view, ViewGroup parent) {

*// Se a view for nula, infla do layout*

**if** (view == **null**)

      view = *inflater*.inflate(R.layout.***row***, **null**);

*// A ButterKnife pode ser um pouco problemática em um Adapter, então colete as Views de row do modo tradicional*

TextView brief = (TextView) view.findViewById(R.id.***row\_message***);

  TextView from = (TextView) view.findViewById(R.id.***row\_contact***);

  TextView subject = (TextView) view.findViewById(R.id.***row\_subject***);

  ImageView picture = (ImageView) view.findViewById(R.id.***contact\_picture***);

*// Define os respectivos textos dado uma position*

brief.setText(**data**[position].getMessage());

  from.setText(**data**[position].getFrom());

  subject.setText(**data**[position].getSubject());

  TextDrawable drawable = TextDrawable.*builder*()

          .buildRoundRect(**data**[position].getFirstLetter(), Color.***RED***, 46);

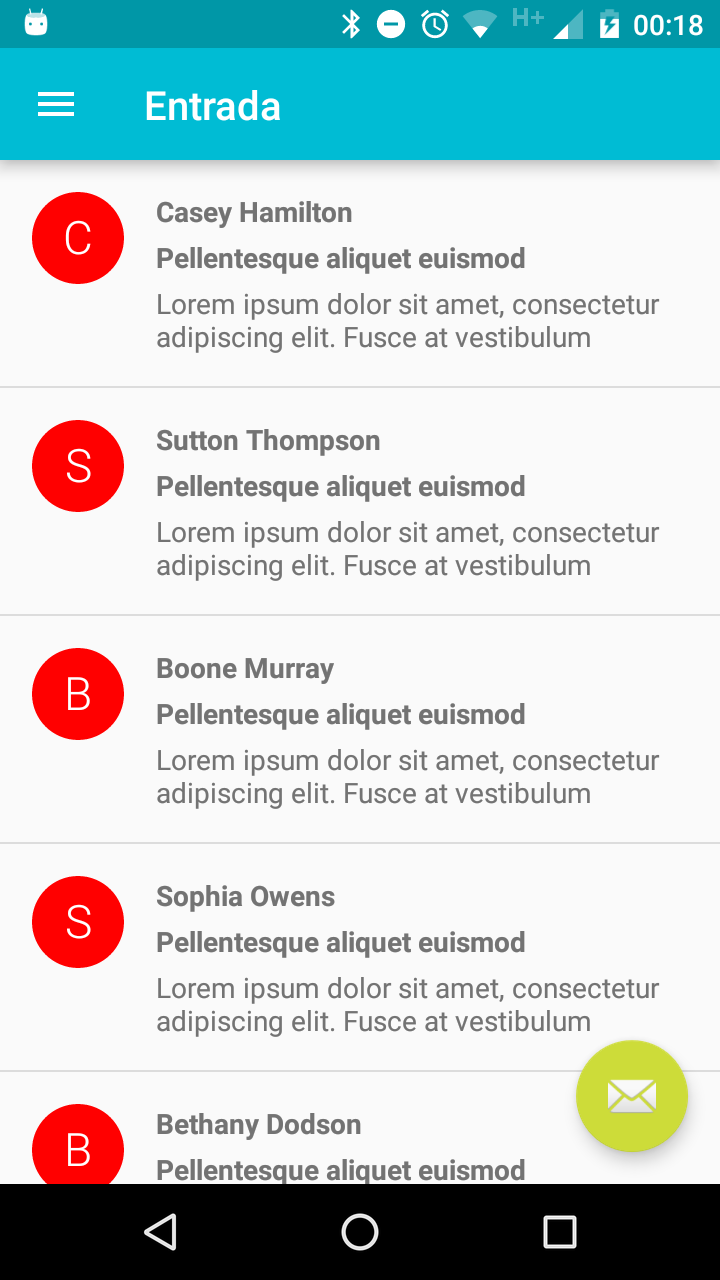
  picture.setImageDrawable(drawable);

**return** view;

}

**Entendendo o código**: Aqui fazemos uso do TextDrawable. O TextDrawable usa o padrão de projeto Builder para criar suas instâncias (que tal pesquisar um pouco sobre o padrão Builder?). Para construir uma instância é necessário um texto (String), que será a inicial do nome do remetente do e-mail, é necessário também uma cor e a curvatura desejada para as bordas desta View. Para obter um formato de círculo, a altura, largura e esta curvatura devem ter o mesmo valor.

1. Rode o app e veja o resultado.



1. Temos as Views circulares! Agora precisamos deixá-las coloridas. Para nossa felicidade, a biblioteca TextDrawable possui um utilitário chamado **ColorGenerator** e ele gera cores muito melhores do que o nosso método de cores aleatórias antigo, então vamos utilizá-lo. Antes de instanciar o TextDrawable, insira o trecho responsável por gerar as cores:

@Override

**public** View getView(**int** position, View view, ViewGroup parent) {

*// Se a view for nula, infla do layout*

**if** (view == **null**)

      view = *inflater*.inflate(R.layout.***row***, **null**);

*// A ButterKnife pode ser um pouco problemática em um Adapter, então colete as Views de row do modo tradicional*

TextView brief = (TextView) view.findViewById(R.id.***row\_message***);

  TextView from = (TextView) view.findViewById(R.id.***row\_contact***);

  TextView subject = (TextView) view.findViewById(R.id.***row\_subject***);

  ImageView picture = (ImageView) view.findViewById(R.id.***contact\_picture***);

*// Define os respectivos textos dado uma position*

brief.setText(**data**[position].getMessage());

  from.setText(**data**[position].getFrom());

  subject.setText(**data**[position].getSubject());

  ColorGenerator generator = ColorGenerator.*MATERIAL*;

  int color = generator.getColor(**data**[position].getFirstLetter());

  TextDrawable drawable = TextDrawable.*builder*()

          .buildRoundRect(**data**[position].getFirstLetter(), color, 46);

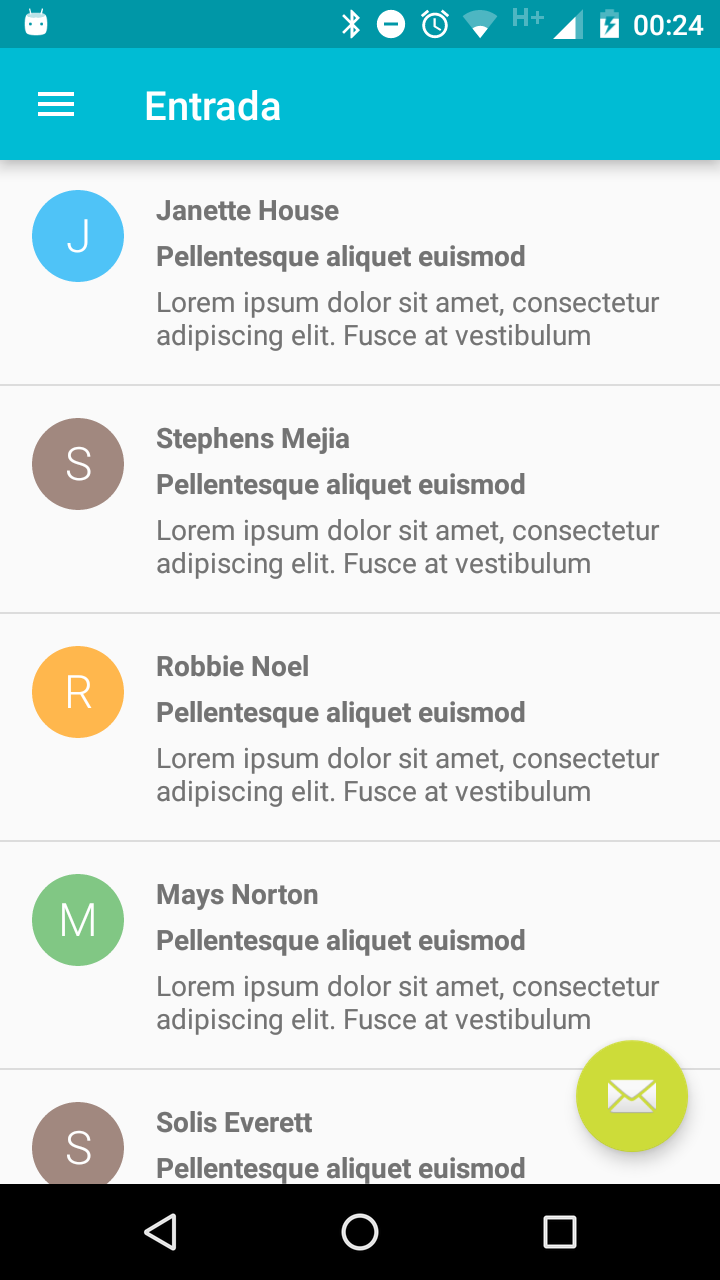
  picture.setImageDrawable(drawable);

**return** view;

}

**Entendendo o código:** Primeiro instânciamos um gerador de cor, com o padrão MATERIAL. Existe também o modo DEFAULT, se você preferir. Se você reparou na aula anterior, as cores que criávamos nunca eram mantidas, isto porque o ListView não cria uma quantidade de Views igual a quantidade de células que informamos, pois senão teríamos um problema de memória. O que o ListView faz é criar uma quantidade de Views suficiente para preencher a tela, e depois fica trocando o conteúdo delas, como uma reciclagem de Views. Então, sabendo disso, as cores ficavam se trocando, mas para evitar isso o ColorGenerator possui um método que “memoriza” um identificador para cada cor. No nosso caso teremos uma cor para cada letra apresentada, então nosso identificador será a letra propriamente dita. Por fim esta cor passada é colocada no TextDrawer.

1. Rode o app e veja o resultado:



Agora iremos aprender como gerenciar e combinar as fontes no nosso aplicativo.

## Tipografia

Desde a versão 5.0 do Android, a Google tem se preocupado muito em manter um padrão de design; o Material, e desde então tem encorajado os desenvolvedores e designer a seguirem guidelines (guias) para manter a identidade do padrão Material.

Neste site Android Developer (onde consta a documentação oficial do Android) possui uma seção destinada a Design. Confira em: <https://developer.android.com/design/index.html>

Mas o que vamos tratar aqui é sobre as fontes do nosso texto.

Como dito acima, a Google tem estipulado um padrão para seus apps Android, e a tipografia não ficou de fora. Veja em: <https://material.google.com/style/typography.html>

No Android Studio, as fontes já seguem este padrão, mas para um maior controle, vamos inserir uma bibioteca que permitirá que escolhemos fontes diferente. Estamos falando da biblioteca **Calligraphy**.

### Calligraphy

A biblioteca Calligraphy (<https://github.com/chrisjenx/Calligraphy>) nos da uma série de benefícios, como configuração de fontes manualmente, inserção de novas fontes ao projeto e etc. Vamos entender com um exemplo. Siga os seguintes passos para a instalação:

1. Primeiro devemos inserir as nossas fontes ao projeto, então descompacte o arquivo **assets.zip** dado e entre na pasta do seu projeto. Navegue para a pasta **app > src > main** e coloque a pasta descompactada **asset** dentro deste diretório. Veja a imagem de exemplo:
2. Insira a dependência no arquivo **build.gradle (Module: app)**, como na imagem abaixo:

dependencies {

*// ...*

*// Calligraphy*

compile **'uk.co.chrisjenx:calligraphy:2.2.0'**

}

1. Agora devemos inicializar a biblioteca na classe Application. Se você realmente estou o código a cada novo exercício, você deve ter percebido que existe um arquivo **MailListApplication,** que este é um **Singleton** e também é utilizado para auxílio dos Fragments. Bom, é neste arquivo que devemos inicializar nossa biblioteca. Entre nele e altere o método **onCreate()**:

@Override

**public void** onCreate() {

**super**.onCreate();

  CalligraphyConfig.*initDefault*(**new** CalligraphyConfig.Builder()

          .setDefaultFontPath(**"fonts/Roboto-RobotoRegular.ttf"**)

          .setFontAttrId(R.attr.***fontPath***)

          .build()

  );

*instance* = **this**;

}

1. Feito isto, devemos aplicar a biblioteca no contexto de cada Activity que fará uso desta biblioteca. Como criamos a BaseActivity, não precisaremos replicar esta etapa em todas as Activities, então adicione o segundo método na **BaseActivity**:

@Override

**protected void** attachBaseContext(Context newBase) {

**super**.attachBaseContext(CalligraphyContextWrapper.*wrap*(newBase));

}

1. Agora basta utilizarmos. Entre no layout **row.xml**, e apague a propriedade **textStyle** de todos os TextViews deste layout.
2. Todas as fontes deverão ficar iguais (Roboto-Regular), então vamos usar o atributo **fontPath**, provido pela Calligraphy, para melhorarmos nossa aplicação. Veja o arquivo **row.xml** após esta nova atualização.

*<?***xml version="1.0" encoding="utf-8"***?>*

<**RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"**

**android:layout\_width="match\_parent"**

**android:layout\_height="wrap\_content"**

**android:padding="@dimen/activity\_general\_margin"**

>

  <**ImageView**

**android:id="@+id/contact\_picture"**

**android:layout\_width="46dp"**

**android:layout\_height="46dp"**

**android:layout\_marginRight="16dp"**/>

  <**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"**

**android:layout\_width="match\_parent"**

**android:layout\_height="wrap\_content"**

**android:orientation="vertical"**

**android:layout\_toRightOf="@+id/contact\_picture"**>

      <**TextView**

**android:id="@+id/row\_contact"**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

**android:text="Contato"**

**android:maxLines="1"**

**android:layout\_marginBottom="4dp"**

**fontPath="fonts/Roboto-Bold.ttf"**/>

      <**TextView**

**android:id="@+id/row\_subject"**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

**android:text="Assunto"**

**android:maxLines="1"**

**android:layout\_marginBottom="4dp"**

**fontPath="fonts/Roboto-Medium.ttf"**/>

      <**TextView**

**android:id="@+id/row\_message"**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

**android:text="Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce at vestibulum odio. Vivamus viverra dignissim dictum. Pellentesque aliquet euismod odio eu accumsan."**

**android:maxLines="1"**

**fontPath="fonts/Roboto-Light.ttf"**

/>

  </**LinearLayout**>

</**RelativeLayout**>

1. Execute a aplicação e veja o resultado.