
AGENDA 4

**TRABALHANDO
COM LINEARLAYOUT
NO PROJETO**





Você aprendeu até agora diversas formas de trabalhar com o Layout de um aplicativo, alterando cores, inserindo imagens, trabalhando com várias *Activity's* ou telas. Agora vamos aprender que no desenvolvimento por meio do Android Studio podemos utilizar uma outra forma de produção de Activity, tão boa quanto o ConstraintLayout.

O LinearLayout é muito utilizado pelos desenvolvedores mais experientes na ferramenta Android Studio, proporcionando os mesmos recursos encontrados no ConstraintLayout. É importante ressaltar que esse método é apresentado somente agora, por já ter uma certa familiaridade e controle sobre a ferramenta de desenvolvimento. O motivo é que esse método utiliza o arquivo XML (*Extensible Markup Language*), base da Activity, e quando é apresentado a um aluno sem experiência na ferramenta, pode gerar um enorme conflito com os outros conceitos apresentados até aqui.

Arquivo XML (Extensible Markup Language) da Activity

O Android Studio proporciona uma interface de desenvolvimento visual para a produção de uma Activity. A ferramenta de desenvolvimento também nos oferece uma outra forma para efetuar os mesmos processos anteriores, garantindo ao desenvolvedor a construção de tela do aplicativo, baseada nos mesmos processos de desenvolvimento de um site que utiliza o HTML para a sua concepção.

O arquivo XML de uma Activity armazena *tag's* responsáveis por construir a interface no momento da execução do aplicativo. As *tag's* e o arquivo XML são totalmente manipuláveis pelo desenvolvedor, proporcionando o desenvolvimento de Activity por meio de códigos.

A Imagem 3 apresenta uma Activity desenvolvida por meio da interface gráfica do Android Studio e a imagem 4 exibe o código XML responsável por desenvolver a tela apresentada na imagem 3. Veja a seguir:

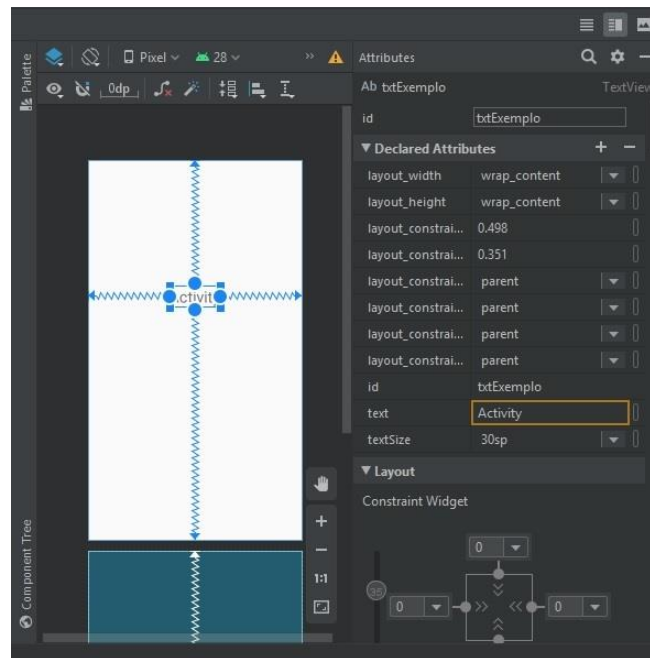


Imagem 3 – Exemplo de Activity.

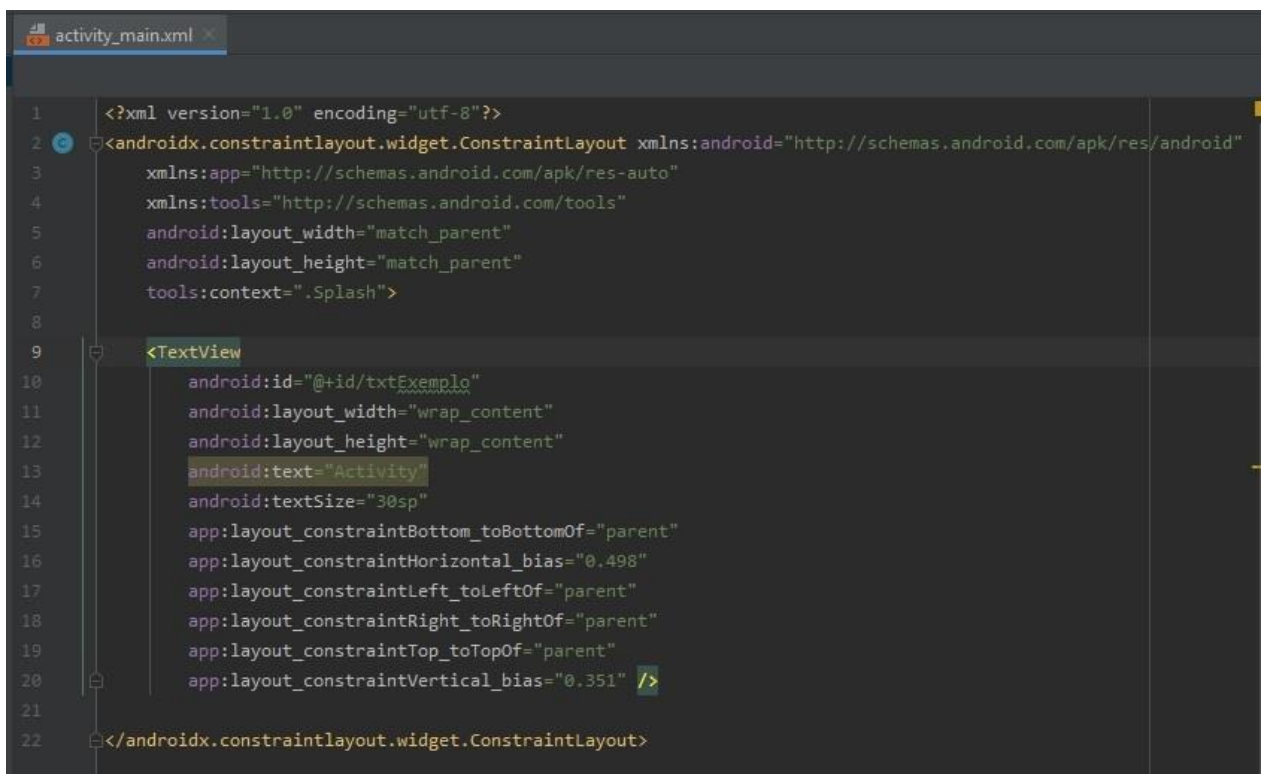


Imagem 4 – Arquivo XML da Activity de exemplo.

Para acessar o código XML de qualquer Activity o desenvolvedor precisa alterar o modo de desenvolvimento. Nas versões anteriores do Android Studio é preciso localizar a aba “Text” como demonstra a imagem 5.

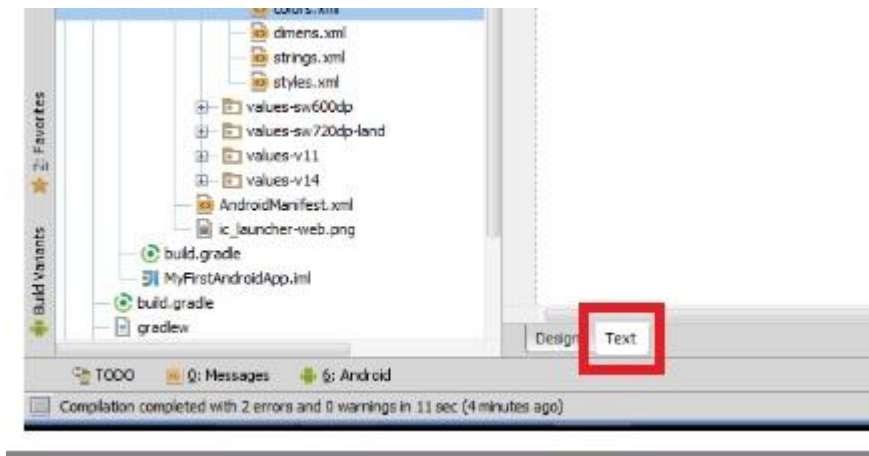


Imagem 5 – Aba “Text” para acesso ao código XML da Activity.

Nas novas versões atuais do Android Studio o desenvolvedor pode escolher entre três opções de desenvolvimento:

- 1- Apenas código XML ou “Code”.
- 2- Apenas por meio da interface gráfica ou “Design”.
- 3- E até mesmo dividir a interface de desenvolvimento em duas, e permitir ambas opções com o modo “Split”.

Veja a imagem 6 os botões de escolha do modo de desenvolvimento da Activity:

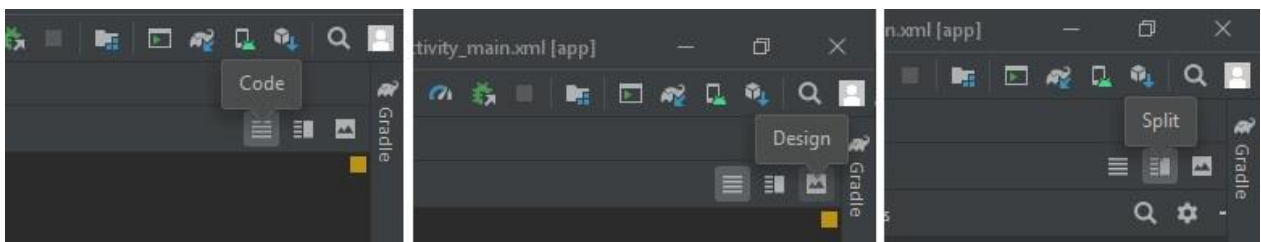


Imagem 6 – Botões de escolha do modo de desenvolvimento da Activity.

LinearLayout

Assim como o `ConstraintLayout`, que é a estrutura padrão quando geramos uma Activity, o **LinearLayout** é uma estrutura utilizada para desenvolver telas. Essa ferramenta gerencia os componentes de uma tela utilizando uma orientação linear vertical ou horizontal.

Cada elemento da tela de um determinado projeto tem a sua posição definida, por meio de uma indicação ao seu objeto anterior.

Para alterar o tipo de Layout de uma aplicação é necessário alterar o arquivo XML referente a Activity presente na pasta **app > res > layout**. O código a seguir é de uma tela utilizando o `ConstraintLayout`. Veja:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Para alterar essa Activity para `LinearLayout` é preciso atuar na tag “<androidx>”. Verifique as alterações necessárias no código a seguir:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.appcompat.widget.LinearLayoutCompat
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="horizontal">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
    />

</androidx.appcompat.widget.LinearLayoutCompat>
```

Após a alteração do código XML, a Activity exibe a aparência exemplificada na imagem 7:

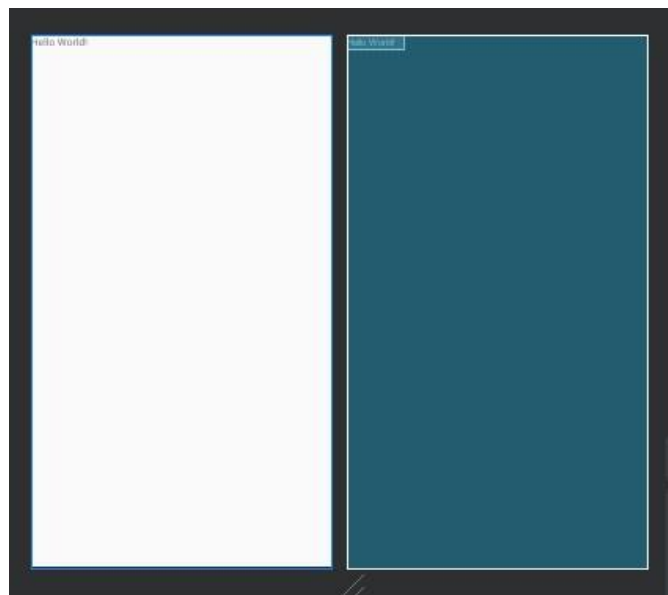


Imagem 7 – Exemplo de Activity com LinearLayout.

Conforme o guia de desenvolvimento do Android Studio, todos os filhos, ou seja, os componentes que são inseridos em um LinearLayout são empilhados um após o outro. Portanto, um Layout definido pelo complemento “**android:orientation=**” como vertical terá somente um filho por linha, independentemente da largura.

E da mesma forma, um Layout com o complemento definido como horizontal terá apenas uma linha, sendo sua altura definida por meio do componente da tela mais alto. Um LinearLayout respeita margens entre filhos e o alinhamento à direita, no centro ou à esquerda de cada filho.

Veja um projeto com três “TextView”, conforme o código a seguir e verifique que foi inserido um “id” para cada componente por meio do comando **android:id="@+id/txtNome"**, respeitando o mesmo processo de identificação dos componentes da tela, assim como é feito no modo visual ou Design de produção da Activity.

O código a seguir também apresenta o conteúdo do texto de cada “TextView”, conforme exemplo. Para isso, alteramos o complemento **android:text="Hello World!"**.

Confira no código a seguir:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.appcompat.widget.LinearLayoutCompat
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="horizontal">

    <TextView
        android:id="@+id/txtNome"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Nome:"
    />

    <TextView
        android:id="@+id/txtTelefone"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Telefone:"
    />

    <TextView
        android:id="@+id/txtEmail"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="E-mail:"
    />

</androidx.appcompat.widget.LinearLayoutCompat>
```

Verifique na imagem 8 o resultado do código apresentado.

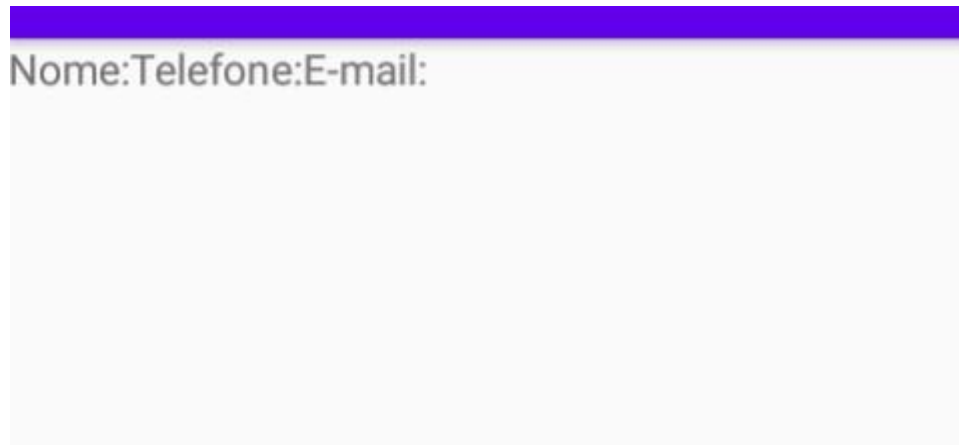


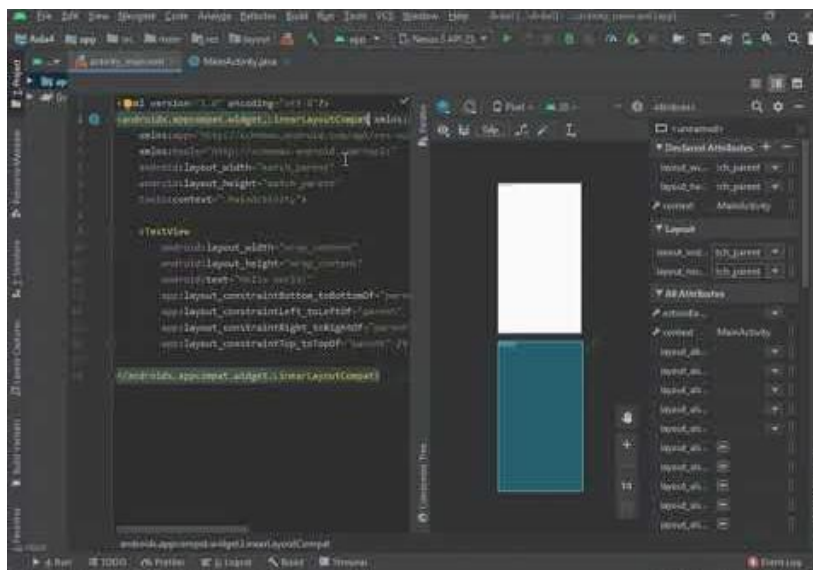
Imagem 8 – Activity com LinearLayout com três componentes “TextView”.



Vamos agora desenvolver o projeto de Activity utilizando o LinearLayout.

Para isso, gere um novo projeto no Android Studio com o nome de “Aula4”.

O vídeo a seguir mostra esse desenvolvimento e, na sequência, explica os conceitos apresentados anteriormente utilizando o arquivo XML da Activity.



Vídeo 1 do Youtube.com: <https://youtu.be/Uf0FvzfoPxY>

Para definir o tamanho de cada componente utilizamos os complementos **android:layout_width=""** e **android:layout_height=""**.

O **layout_width** é utilizado para definir a largura, e o **layout_height** é responsável por definir a altura de cada componente. Para esses dois complementos podemos utilizar os atributos **“match_parent”** para que o componente ocupe todo o espaço da tela ou **“wrap_content”** para que o componente ocupe apenas o espaço necessário com base no seu conteúdo.

Trabalhando com EditText no LinearLayout

Assim como inserimos o EditText no modo Design, podemos trabalhar com esse componente via código no arquivo XML da Activity. Por padrão, o EditText possui alguns complementos como mostra o código a seguir:

```
<EditText
    android:id="@+id/editText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="textPersonName"
    android:text="Name"
/>
```

O complemento **android:ems="10"** é responsável por definir o tamanho do EditText e o **android:inputType="textPersonName"** define o tipo do teclado utilizado para preencher o campo no dispositivo do usuário.

Vamos verificar alguns tipos de teclados durante o projeto, o **"textPersonName"** mostra o teclado completo. Diferente do **"phone"** que exibe apenas o teclado numérico, ele é destinado para campos que trabalham com telefone.

A imagem 9 mostra o resultado visual da Activity após a inserção dos componentes EditText.

Verifique o código XML utilizado para o desenvolvimento da tela:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.appcompat.widget.LinearLayoutCompat
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```

tools:context=".MainActivity"
android:orientation="vertical">

<TextView
    android:id="@+id/txtNome"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Nome:"
/>

<EditText
    android:id="@+id/edtNome"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="textPersonName"
    android:text=""
/>

<TextView
    android:id="@+id/txtTelefone"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Telefone:"
/>

<EditText
    android:id="@+id/edtTelefone"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="phone"
    android:text=""
/>

<TextView
    android:id="@+id/txtEmail"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="E-mail:"
/>

<EditText
    android:id="@+id/edtEmail"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ems="10"
    android:inputType="textEmailAddress"
    android:text=""
/>

</androidx.appcompat.widget.LinearLayoutCompat>

```

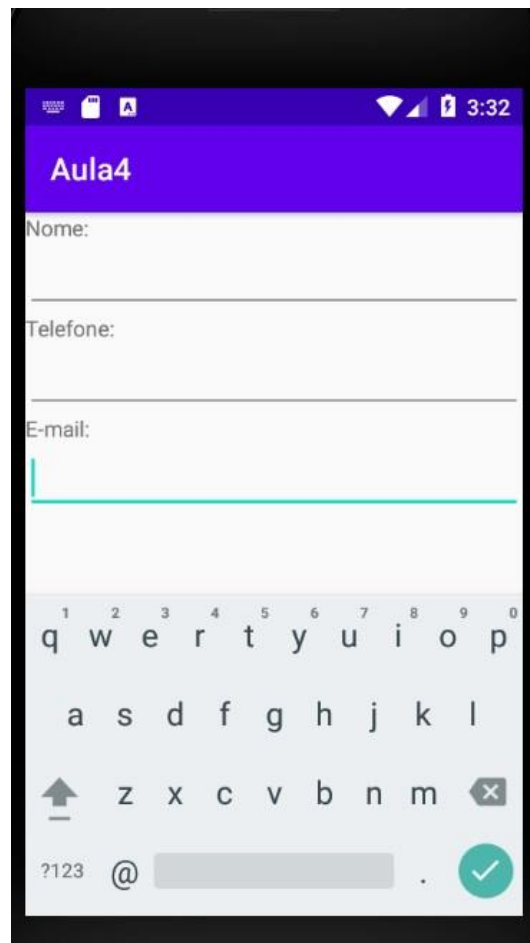


imagem 9 – Activity com LinearLayout com três componentes “EditText”.

Trabalhando com Button no LinearLayout

```
<Button
    android:id="@+id/btnSalvar"
    android:layout_margin="16dp"
    android:layout_gravity="center_horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Salvar" />
```

O componente Button segue o mesmo padrão dos componentes anteriores. Vamos utilizar esse exemplo para apresentar dois novos complementos, que podem ser utilizados nos outros componentes da Activity.

O complemento **android:layout_margin="16dp"** é responsável por gerar uma margem nas quatro orientações do componente. Essa margem serve para gerar a espécie de um espaçamento entre os componentes de uma tela e até mesmo entre o componente e o próprio LinearLayout.

Podemos aplicar uma margem em um componente em apenas uma de suas orientações, como por exemplo, aplicar uma margem apenas do lado esquerdo de um botão.

Para isso utilizamos outros complementos semelhantes ao **android:layout_margin**.

Veja na imagem 10 a relação de estilos de margem:

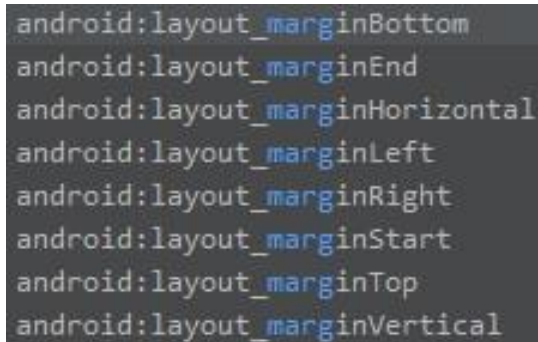
A screenshot of a code editor showing a list of Android layout margin attributes. The attributes are listed vertically: android:layout_marginBottom, android:layout_marginEnd, android:layout_marginHorizontal, android:layout_marginLeft, android:layout_marginRight, android:layout_marginStart, android:layout_marginTop, and android:layout_marginVertical. The text is in a monospaced font with syntax highlighting, where the 'margin' part of each attribute is highlighted in blue.

imagem 10 – Estilos de margem para componentes.

O complemento **android:layout_gravity="center_horizontal"** é utilizado para alinhar o componente ao centro da Activity. Encontramos também outras formas de alinhamento como a esquerda, a direita, entre outros tipos.

Verifique o código a seguir com a conclusão do desenvolvimento da Activity, utilizando o LinearLayout:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.appcompat.widget.LinearLayoutCompat
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical">

    <TextView
        android:id="@+id/txtNome"
        android:layout_marginStart="16dp"
        android:layout_marginTop="16dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Nome:"
    />

    <EditText
        android:id="@+id/edtNome"
```

```

        android:layout_marginHorizontal="16dp"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textPersonName"
        android:text=""
    />

    <TextView
        android:id="@+id/txtTelefone"
        android:layout_marginStart="16dp"
        android:layout_marginTop="16dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Telefone:"
    />

    <EditText
        android:id="@+id/edtTelefone"
        android:layout_marginHorizontal="16dp"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="phone"
        android:text=""
    />

    <TextView
        android:id="@+id/txtEmail"
        android:layout_marginStart="16dp"
        android:layout_marginTop="16dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="E-mail:"
    />

    <EditText
        android:id="@+id/edtEmail"
        android:layout_marginHorizontal="16dp"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:inputType="textEmailAddress"
        android:text=""
    />

    <Button
        android:id="@+id/btnSalvar"
        android:layout_margin="16dp"
        android:layout_gravity="center_horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Salvar" />

</androidx.appcompat.widget.LinearLayoutCompat>

```

Trabalhando com AlertDialog (Caixa de Mensagem)

No projeto, vamos utilizar uma caixa de mensagem para exibir os dados digitados nos campos EditText simulando que os dados foram salvos. Para isso vamos utilizar o comando **AlertDialog**.

É necessário preparar todos os dados do **AlertDialog** antes da sua utilização. Veja a seguir o código desenvolvido na classe Java da Main Activity, disponível em **app > java > com.example.aula4 > MainActivity**.

Observe os comentários realizados no código explicando as etapas de desenvolvimento do **AlertDialog**:

```
package com.example.aula4;

import androidx.appcompat.app.AppCompatActivity;

import android.app.AlertDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button btnSalvarProg = (Button) findViewById(R.id.btnSalvar);
        final EditText edtNomeProg = (EditText) findViewById(R.id.edtNome);
        final EditText edtTelefoneProg = (EditText) findViewById(R.id.edtTelefone);
        final EditText edtEmailProg = (EditText) findViewById(R.id.edtEmail);

        btnSalvarProg.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                //Desenvolvimento do modelo do AlertDialog
                AlertDialog.Builder modelo = new
AlertDialog.Builder(MainActivity.this);
                //Define o título para o AlertDialog
                modelo.setTitle("Cadastro:");
                //Define a mensagem para o AlertDialog
                modelo.setMessage("Nome: " + edtNomeProg.getText()
                    + "\nTelefone: " + edtTelefoneProg.getText()
                    + "\nE-mail: " + edtEmailProg.getText());
                //Define um botão para o AlertDialog
                modelo.setPositiveButton("Ok", new DialogInterface.OnClickListener()
{
                    public void onClick(DialogInterface arg0, int arg1) {
                        //Define o que é feito quando o usuário clicar no botão
```

```
desenvolvido
    });
    //Cria um AlertDialog com base no modelo gerado anteriormente
    AlertDialog alerta = modelo.create();
    //Mostra o AlertDialog na tela
    alerta.show();
}
});
}
```

Observe as imagens 11 e 12 que demonstram o resultado do aplicativo desenvolvido:

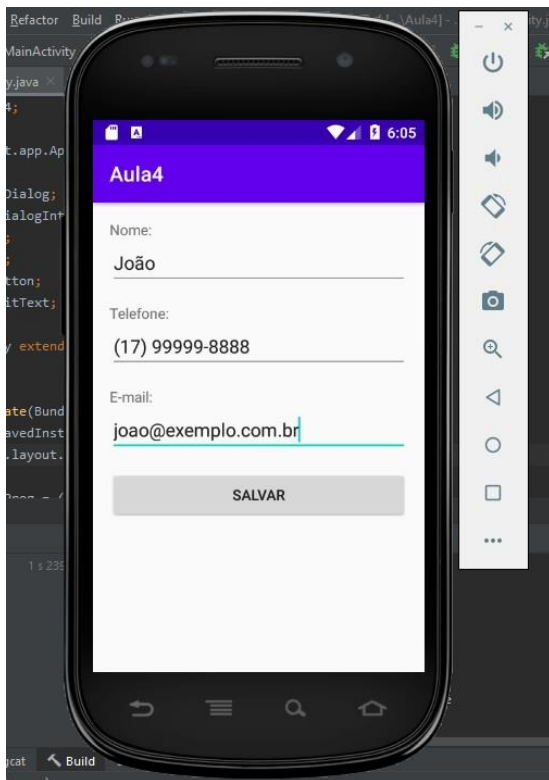


Imagem 11 – Aplicativo aula4.

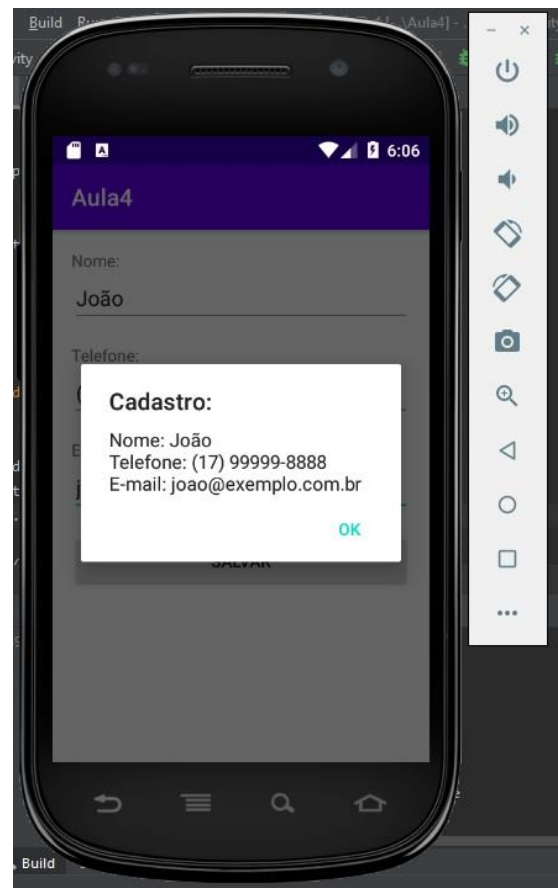


Imagem 11 – Aplicativo aula4.