|  |  |
| --- | --- |
|  | FTCE  Faculdade de Tecnologia e Ciências Exatas |

USJT - 2019 – Programação para Web e Dispositivos Móveis

Professores: Anderson e Bossini

Aula: 01

Assunto: Introdução ao Android e ao Android Studio

**Conceitos básicos**

**Android**

É um sistema operacional móvel baseado no kernel do Linux atualmente desenvolvido pelo Google. O projeto se iniciou com a empresa Android Inc., fundada em 2003 e comprada em 2005 pelo Google. O sistema operacional foi revelado em 2007, quando o Google fundou a Open Handset Alliance, um consórcio de empresas de hardware, software e telecom cujo objetivo era o desenvolvimento de padrões abertos de telefonia. O primeiro smartphone com Android foi o HTC Dream, lançado em outubro de 2008.

Já houve várias versões do Android, todas elas nomeadas em ordem alfabética com nomes de sobremesas: 1.5 Cupcake, 1.6 Donut, 2.0 Eclair, 2.2 Froyo, 2.3 Gingerbread, 3.0 Honeycomb, 4.0 Ice Cream Sandwich, 4.1/4.2/4.3 Jelly Bean, 4.4 KitKat, 5.0/5.1 Lollipop, 6.0 Marshmallow, 7.0/7.1 Nougat, 8.0/8.1 Oreo e 9.0 Pie.

A distribuição atual do uso de versões, baseada nos acessos ao Google Play, medidos até 26/Out/18 é:

- Oreo (API 26 e 27): 21,5%

- Nougat (API 24 e 25): 28,2%

- Marshmallow (API 23): 21,3%

- Lollipop (API 21 e 22): 17,9%

- KitKat (API 19): 7,6%

- JellyBean (API 16 a 18): 3,0%

- Ice Cream Sandwich (API 15): 0,3%

- Gingerbread (API 10): 0,2%

Estes dados podem ser obtidos nos dashboards de desenvolvedores disponíveis em: https://developer.android.com/about/dashboards/index.html

**O JDK (Java Development Kit)**

A JDK é a coleção de bibliotecas e ferramentas necessárias para construir e rodar aplicativos Java.



Estas ferramentas são usadas no processo de build do Android

**Android SDK**

Contém as APIs e ferramentas necessárias para criar e rodar apps Android nativos.



Ferramentas de construção (ex. compilador de bytecode) e de execução (ferramentas de debug)

Bibliotecas core, como o java.\* e o android.\*

Bibliotecas opcionais

**Compilação**

Até a versão 4.4 do Android, o processo de build ocorria como ilustra a figura a seguir.



**Empacotamento**

O bytecode do App e seus assets (suas imagens, ícones, arquivos de dados, etc.) são combinados em um Application Package (arquivo .apk) para implantação.

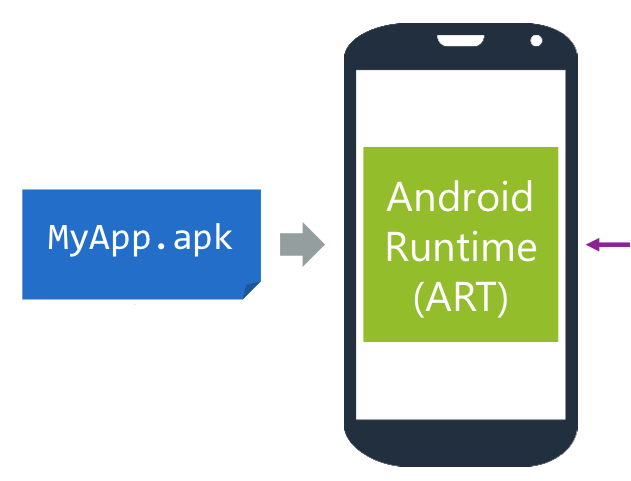
Antes de enviar para o Google Play, o app deve ser assinado digitalmente por meio do jarsigner (a IDE faz isso).



O arquivo apk contém todos os assets do App e está pronto para rodar

**Execução**

O Android Runtime (ART) é o sucessor da máquina Dalvik e é o responsável pela execução. Os bytecodes são compilados em código nativo durante a instalação, em um processo chamado Ahead-of-Time, ou AOT). Versões anteriores à 5.0 Lollipop usavam a Dalvik Virtual Machine que fazia a tradução do bytecode.



Cada app roda em seu próprio processo em sua própria cópia da ART Virtual Machine.

**Atualizações da SDK**

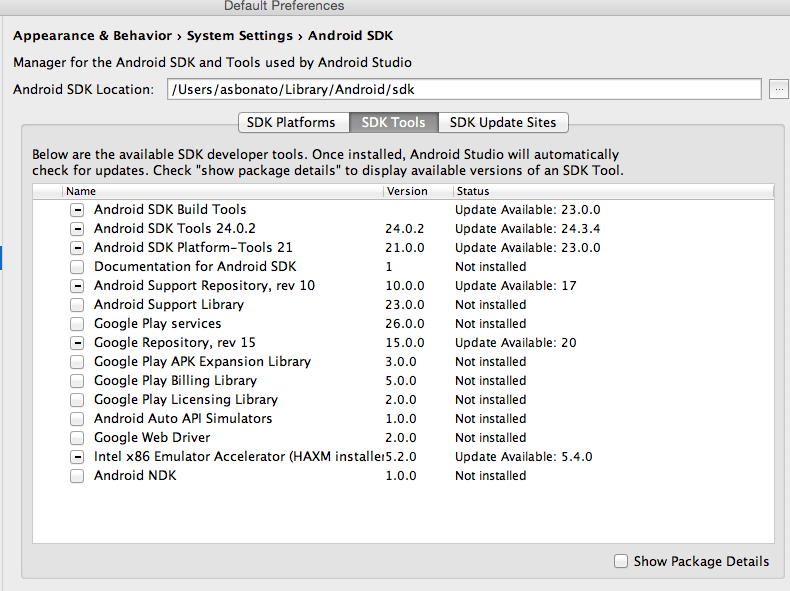
Você precisa atualizar manualmente sua Plataforma e Ferramentas Android SDK de modo a contruir apps nas últimas versões do Android. As versões, como vimos acima, são nomeadas por um nome código (sobremesa), um número e cada uma possui um nível de API. Este nível de API é usado para controlar as combinações de bibliotecas, arquivos de manifesto, permissões, etc.

De acordo com os percentuais de uso do Android vistos na página 1, atualmente pode-se desenvolver aplicativos voltados para a API 21 ou 22 com a API mínima 17, contemplando assim a maioria dos dispositivos Android.

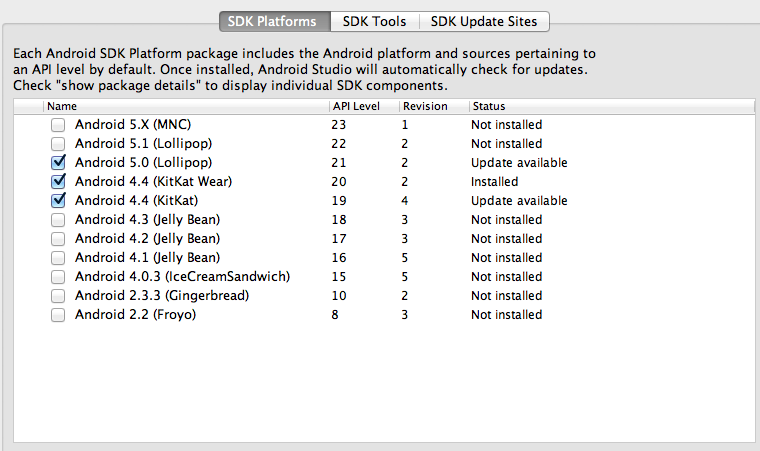
**O SDK Manager**

É a ferramenta do Google que permite que você instale versões novas (e antigas) das API do Android. Pode ser usada standalone ou integrada no Android Studio desde a versão 1.3.

O Android divide as ferramentas em três categorias: Build Tools, SDK Tools e Platform Tools . Você deve sempre manter as três atualizadas.



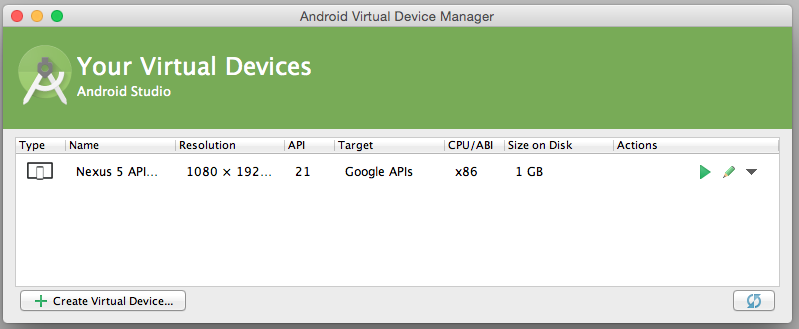
Use o SDK Manager também para atualizar as versões da plataforma para as quais você pretende desenvolver. **Instale somente as que precisar, pois consomem bastante espaço em disco**.



**Emuladores**

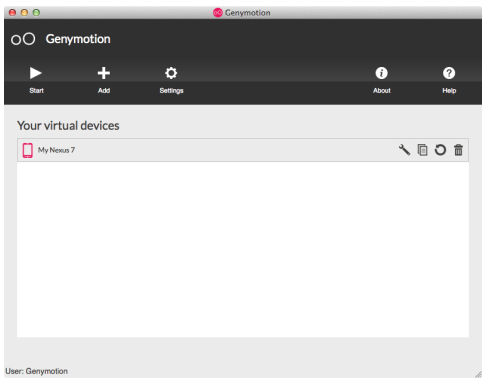
Para a execução do aplicativo durante o desenvolvimento há duas alternativas. A execução em um dispositivo real ou o uso de emuladores.

O Android SDK fornece um emulador no qual é possível emular uma série de dispositivos.



Para criar um novo emulador, é necessário clicar em Create Virtual Device e configurar os parâmetros. Não se esqueça que você deve ter um device configurado para o nível de API que estiver desenvolvendo o app.

O emulador fornecido pelo Google já foi excessivamente lento. Hoje em dia seu desempenho é razoável. Nos laboratórios não podemos usar por causa de licenciamento (grátis só para uso pessoal), mas na sua máquina também há a possibilidade de usar o Genymotion, que roda dentro de uma VM Virtual Box. O processo de criação de dispositivos é semelhante ao AVD visto acima.



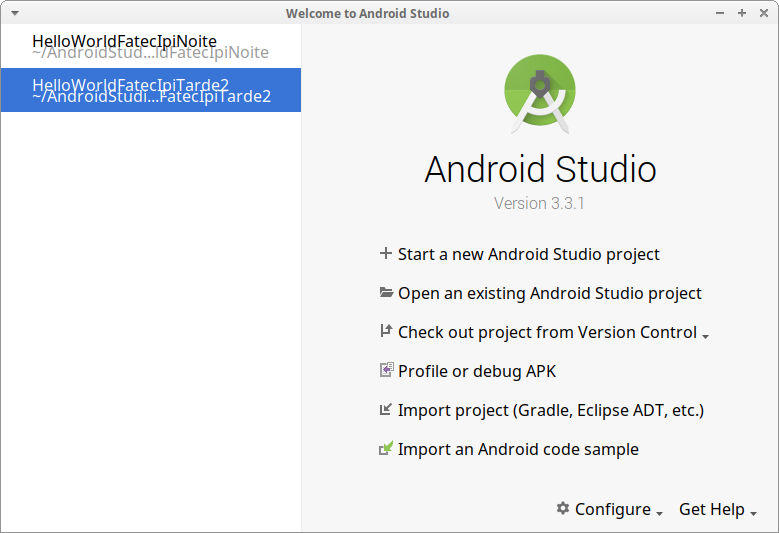
**O IDE de Desenvolvimento**

O IDE que iremos usar é o Android Studio, criado pelo Google a partir da IDE IntelliJ. Até 2014 a ferramenta usada era o Eclipse com um plugin para Android, chamado ADT. Desde então o Android Studio tornou-se o IDE oficial.

**Exercício: Meu Primeiro App Android**

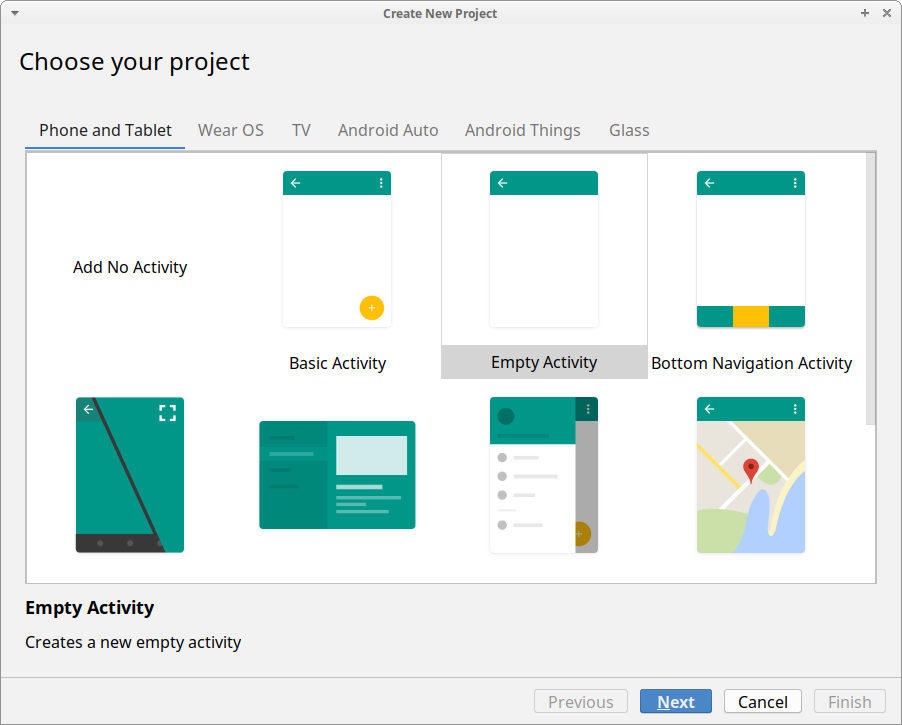
**Passo 1**. Abra o Android Studio e clique em Start a New Android Studio Project, como mostra a Figura 1.

Figura 1



**Passo 2**. Como na Figura 2, escolha um template para sua aplicação. Há diferentes opções que testaremos ao longo do curso. Neste exemplo, escolha Empty Activity.

Figura 2



**Passo 3** Na tela a seguir, exibida pela Figura 3, preencha os campos com os seguintes valores:

Application name: Meu Primeiro App

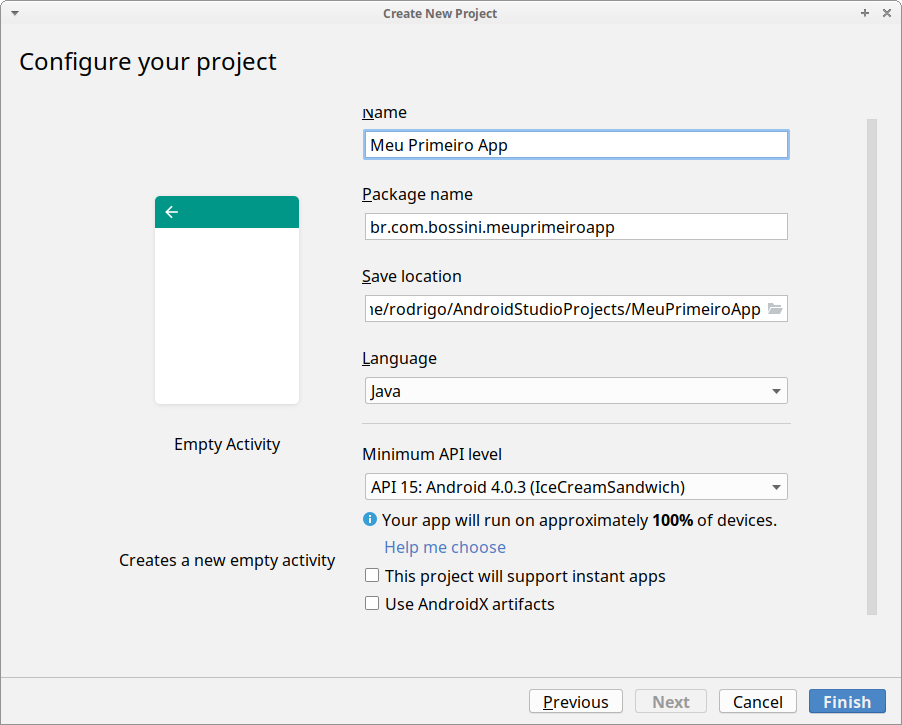
Package Name: br.com.bossini.meuprimeiroapp

Save Location: Valor padrão

Language: Java

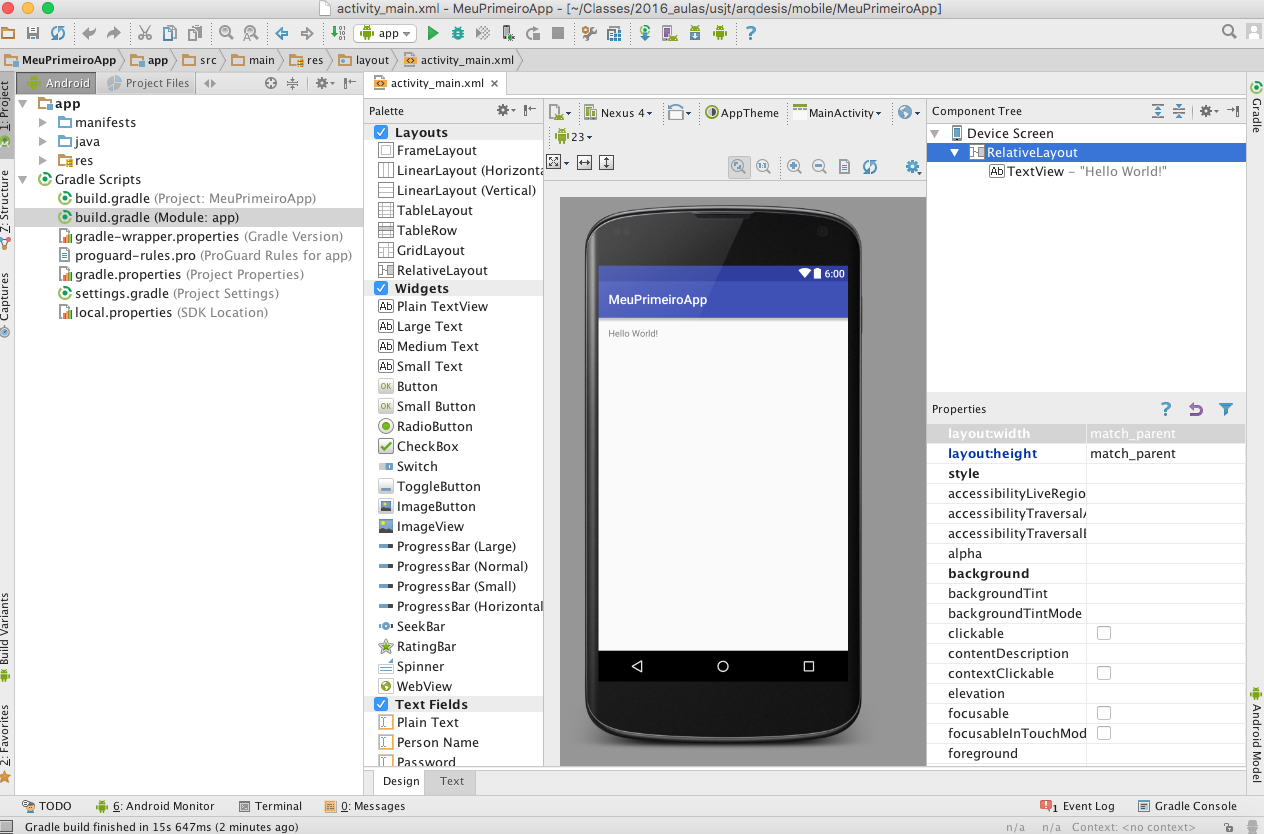
Minimum API Level: API 15

Figura 3



**Passo 4** O Android Studio irá criar seu projeto. Sua tela deverá parecer com a tela exibida pela Figura 4 Na porção esquerda da tela você vê a estrutura do projeto. Na porção central está aberta a classe MainActivity.java e o activity\_main.xml, que é o layout da tela. Na porção direita, há um preview de como uma aplicação ficaria em um dispositivo Nexus 4. Veja que, ao criar um projeto, o Hello World já vem pronto. Explore o preview em outros dispositivos, dê uma olhada no código da MainActivity e explore as pastinhas do projeto.

Figura 4



**Passo 5** Uma breve explicação da estrutura do projeto:

**- AndroidManifest.xml** define cada um dos componentes do app; mexa nele com cuidado

**- src/** contém o código fonte; tem uma Activity que roda quando o app é iniciado

**- bin/** contém os .class gerados

**- gen/** arquivos gerados pelo ADT; nunca mexa neles!

**- res/** contém vários subdiretórios com os recursos do app, como:

**- drawable** onde você coloca os bitmaps; você pode criar subdiretórios como o **drawable-hdpi** para imagens em alta resolução, ldpi para baixa e um mdpi para média, etc.

**- layout** contém os arquivos que definem a interface do app

**- values** contém XMLs com definições de strings e cores

**Passo 6** Criando a interface do usuário

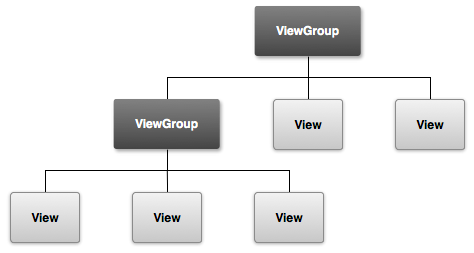
- A interface do app é criada usando-se uma hierarquia de objetos View e ViewGroup

- Os objetos View são as UI que contém os botões, campos, etc.

- Os ViewGroups são contâiners invisíveis que definem o layout dos views

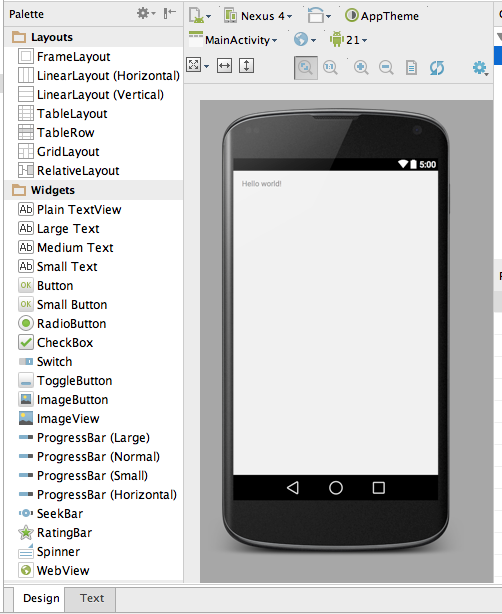
Veja a Figura 5.

Figura 5



**Passo 7** Clique no diretório res/layout. O Android Studio terá criado automaticamente um arquivo activity\_main.xml que contém a interface inicial do app. Esta interface pode ser alterada tanto graficamente - clicando em Design - como no modo xml, clicando em Text (preferível). Veja a Figura 6.

Figura 6



Dependendo da versão do Android Studio sendo utilizada, o código do arquivo activity\_main.xml será parecido com o que exibe a Listagem 1.

Listagem 1

|  |
| --- |
| *<?***xml version="1.0" encoding="utf-8"***?>*  <**android.support.constraint.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"** />  </**android.support.constraint.ConstraintLayout**> |

**Passo 8** Apague o TextView original e altere o layout para LinearLayout. O XML deve ficar como abaixo. LinearLayout é uma subclasse do ViewGroup que define um layout vertical ou horizontal para o app na propriedade orientantion.

- O match\_parent (ou fill\_parent até a versão 2.3) nas configurações indica que a janela do app deve ter as mesmas dimensões da janela pai.

**Passo 9** Adicionando um text field no layout.

- o EditText é um campo texto editável.

- o id é um identificador único para o campo; o @ é sempre requerido quando você vai ser referir a um recurso em Android; o + só é necessário na criação do id;

- para cada id será criado um recurso no diretório /gen

- o wrap\_content em width e height diz que o campo deve ser um tamanho suficiente para que o conteúdo caiba nele

- o hint é a string default que aparece no campo antes dele ser preenchido

O xml deve ficar como abaixo. Vai haver um erro na hint, mas já arrumaremos isso.

*<?***xml version="1.0" encoding="utf-8"***?>*<**LinearLayout  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context="br.usjt.meuprimeiroapp.MainActivity"**>  
  
 <**EditText  
 android:id="@+id/edit\_message"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:hint="@string/edit\_message"**/>  
  
</**LinearLayout**>

**Passo 10** Adicionando strings no seu app

- Vá em res/values e abra o arquivo strings.xml

- Ele também pode ser editado graficamente ou direto no xml.

- Faremos no XML

- Atenção: **Nunca chumbe strings direto no seu código; use sempre string resources!**

Apague tudo e deixe conforme abaixo:

<**resources**>  
 <**string name="app\_name"**>MeuPrimeiroApp</**string**>  
 <**string name="edit\_message"**>Escreva uma mensagem</**string**>  
 <**string name="button\_send"**>Envie</**string**>  
</**resources**>

**Passo 11** Crie um botão no seu app

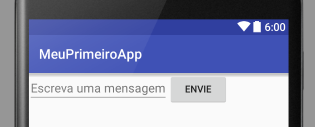
- Clique logo abaixo do EditText

- Novamente, o wrap\_content diz que o tamanho do botão é do tamanho do conteúdo

- Veja que o texto do botão é uma string que você acabou de definir no strings.xml

*<?***xml version="1.0" encoding="utf-8"***?>*<**LinearLayout  
 xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context="br.usjt.meuprimeiroapp.MainActivity"**>  
  
 <**EditText  
 android:id="@+id/edit\_message"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:hint="@string/edit\_message"**/>  
  
 <**Button  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="@string/button\_send"**/>  
  
</**LinearLayout**>

**Passo 12** Veja como o app está ficando clicando no Design.



Note que o campo texto e o botão não ocupam toda a largura da tela.

- Aumente o tamanho do campo texto no próprio Design e volte para o Text para ver o efeito.

- Foi criado o atributo android:layout\_weight para aumentar.

- Ele recebe números 1, 2, 3, etc que dizem quantas partes da view o campo deve ocupar. Claro que isso é proporcional aos outros campos. Ajuste o peso para 1.

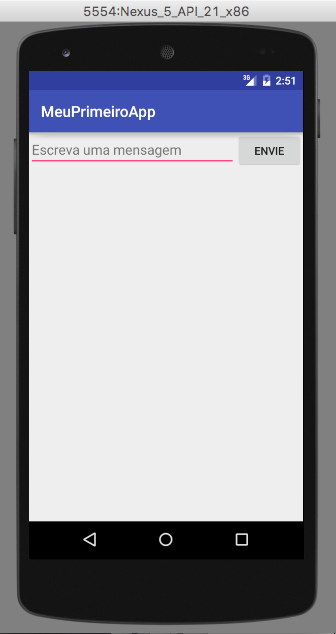
- Por exemplo, se você der a um view o peso 1 a a outro o peso 2, a soma é 3.

- Então o primeiro view irá ocupar 1/3 do layout e o outro, 2/3.

- Agora mude o atributo width (largura) para 0dp. Isso é mais eficiente pois o wrap\_content obriga o layout a ficar calculando o tamanho

<**EditText  
 android:id="@+id/edit\_message"  
 android:layout\_width="0dp"  
 android:layout\_height="wrap\_content"  
 android:hint="@string/edit\_message"  
 android:layout\_weight="1"** />

**Passo 13** Rode o app pela primeira vez para ver como está ficando. Escolha um emulador no AVD. Depois de rodar, deixe o emulador aberto, assim será mais rápido na próxima vez.



**Passo 14** Respondendo ao botão enviar chamando outra Activity

- Inclua a propriedade android:OnClick=”send Message” no xml do botão em main xml

- Isso vai fazer com que um método sendMessage seja chamado quando o botão for clicado

- Agora precisa definir o método

<**Button  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="@string/button\_send"  
 android:onClick="sendMessage"**/>

**Passo 15** Criando métodos

- Abra a classe MainActivity.java no diretório /src, dentro da package (se não estiver aberta.

- Adicione o método no final da classe, conforme abaixo.

- Retornará void, deve ser public e receber uma View como parâmetro

- O Android Studio automaticamente importa o que estiver faltando

*//será chamado quando o usuário clicar em Enviar***public void** sendMessage(View view){  
  
}

**Passo 16** Criando um Intent

- Os Intents ligam componentes separados, como duas Activity

- Insira uma linha de código dentro do sendMessage (ignore o erro por enquanto)

- O primeiro parâmetro do Intent é o contexto; no caso, this.

- O class é a classe para onde o sistema deve entregar o Intent.

*//será chamado quando o usuário clicar em Enviar***public void** sendMessage(View view){  
 Intent intent = **new** Intent(**this**, DisplayMessageActivity.**class**);  
}

**Passo 17** Configure mais dados e chame a outra Activity

- Estes dados serão carregados pelo Intent para a outra Activity

- Crie uma referência ao EditText do app; use o findViewById para encontrá-lo;

- Pegue o texto digitado e adicione ao Intent; ele carrega vários pares de dados chamados Extras

- Use o putExtra para adicionar um Extra; crie a constante EXTRA\_MESSAGE como referência para que a próxima Activity possa pegar a mensagem

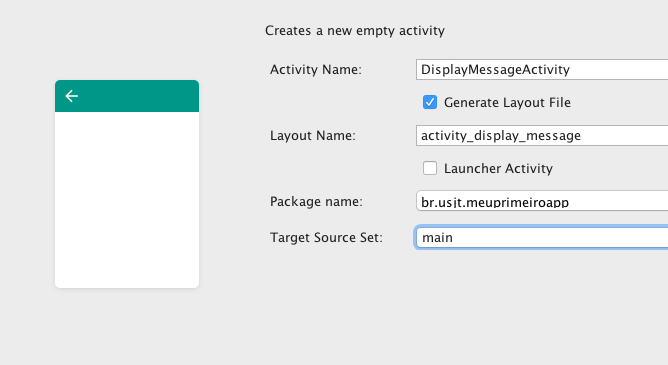
- Chame a outra Activity via starActivity passando o intent como parâmetro

*//constante static para identificar a mensagem***public final static** String ***EXTRA\_MESSAGE*** = **"br.usjt.meuprimeiroapp.MESSAGE"**;  
*//será chamado quando o usuário clicar em Enviar***public void** sendMessage(View view){  
 Intent intent = **new** Intent(**this**, DisplayMessageActivity.**class**);  
 EditText editText = (EditText) findViewById(R.id.***edit\_message***);  
 String message = editText.getText().toString();  
 intent.putExtra(***EXTRA\_MESSAGE***, message);  
 startActivity(intent);  
}

**Passo 18** Crie uma nova Activity

- Na janela de Projeto, clique com o botão direito sobre App > New > Activity, escolha uma nova Empty Activity

- Altere os dados conforme abaixo e clique em Finish.



- Será criada uma nova classe, DisplayMessageActivity.java, na mesma package da MainActivity.java.

- Será criado também um novo layout para esta activity na pasta layout em res. Abra-o e acrescente android:id="@+id/activity\_display\_message" se já não estiver lá.

*<?***xml version="1.0" encoding="utf-8"***?>*<**RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:paddingBottom="@dimen/activity\_vertical\_margin"  
 android:paddingLeft="@dimen/activity\_horizontal\_margin"  
 android:paddingRight="@dimen/activity\_horizontal\_margin"  
 android:paddingTop="@dimen/activity\_vertical\_margin"  
 tools:context="br.usjt.meuprimeiroapp.DisplayMessageActivity"  
 android:id="@+id/activity\_display\_message"**>  
</**RelativeLayout**>

**Passo 19** Clique em DisplayMessageActivity.java

- A nova classe deve ficar conforme abaixo.

- O método onCreate é sempre obrigatório em uma Activity e é chamado sempre que uma Activity é criada.

@Override  
**protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_display\_message***);  
  
 Intent intent = getIntent();  
 String message =

intent.getStringExtra(MainActivity.***EXTRA\_MESSAGE***);  
 TextView textView = **new** TextView(**this**);  
 textView.setTextSize(40);  
 textView.setText(message);  
  
 ViewGroup layout = (ViewGroup)

findViewById(R.id.***activity\_display\_message***);  
 layout.addView(textView);  
}

Tem um monte de coisas acontecendo aqui:

- a chamada a getIntent() pega o Intent que chamou a Activity. Toda Activity é chamada por um Intent.

- o método intent.getStringExtra() recupera os dados passados pela primeira Activity.

- foi criado, por meio de código, um TextView e configurados seu tamanho e a texto.

- O TextView foi adicionado no layout identificado por R.id.activity\_display\_message***.*** Foi feito casting para ViewGroup pois é esta classe que tem o método addView().

**Passo 20** Valide se foi adicionada a nova Activity no manifesto

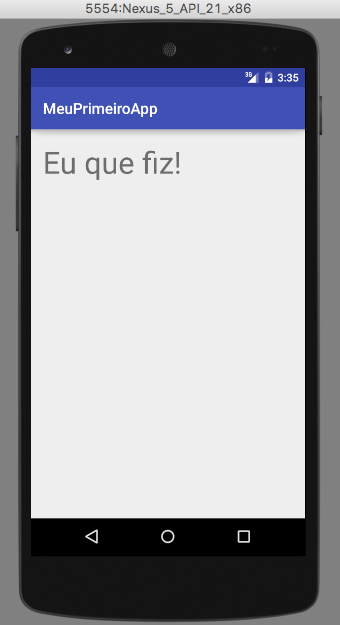
- Abra o AndroidManifes.xml na pastinha Manifests e vá na edição do XML

- O meta-data configura a hierarquia de activities; primeiro a MainActivity e depois a DisplayMessageActivity

*<?***xml version="1.0" encoding="utf-8"***?>*<**manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 package="br.usjt.meuprimeiroapp"**>  
  
 <**application  
 android:allowBackup="true"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/app\_name"  
 android:supportsRtl="true"  
 android:theme="@style/AppTheme"**>  
 <**activity android:name=".MainActivity"**>  
 <**intent-filter**>  
 <**action android:name="android.intent.action.MAIN"** />  
  
 <**category android:name="android.intent.category.LAUNCHER"** />  
 </**intent-filter**>  
 </**activity**>  
 <**activity android:name=".DisplayMessageActivity"**></**activity**>  
 </**application**>  
  
</**manifest**>

**Passo 21** Rode novamente o app, que está pronto!





**Bibliografia**

<http://developer.android.com/training/index.html>

**Exercício Prático 1: Preparar o github e subir o código da primeira entrega.**

Forma de entrega: suba o projeto para o GitHub e gere um **release**.

Envie um e-mail para seu professor com os seguintes dados:

RA

Nome

link do seu **repositório.**

**Somente é necessário o envio de um e-mail. Inclua o endereço do seu repositório. A partir daí o professor terá acesso a todas as releases. Note que o Github controla a data em que as releases são geradas e o prazo para gerar cada uma delas é sempre de uma semana após a aula corrente.**

NOTA: Todos os métodos e classes devem possuir comentários do tipo Javadoc com o nome e o RA do autor. A falta destes comentários, ou o comentário em nome de outro autor, acarretará na anulação da entrega.