

USJT - 2018 – Programação Multiplataforma e Paradigmas de Linguagens

Professores: Bonato e Rodrigo

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.

J

á 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 e 8.0/8.1 Oreo.

A distribuição atual do uso de versões, baseada nos acessos ao Google Play, medidos até 06/Fev/18 é:

- Oreo (API 26 e 27): 1,1%
- Nougat (API 24 e 25): 28,5%
- Marshmallow (API 23): 28,1%
- Lollipop (API 21 e 22): 24,6%
- KitKat (API 19): 12,0%
- JellyBean (API 16 a 18): 5,0%
- Ice Cream Sandwich (API 15): 0,4%
- Gingerbread (API 10): 0,3%

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

## A JDK

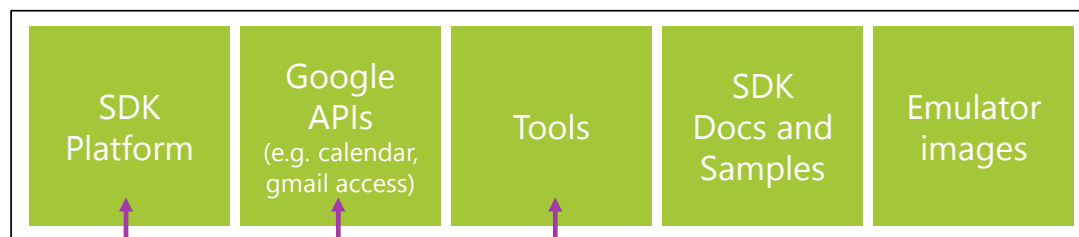
A JDK é a coleção de bibliotecas e ferramentas necessárias para construir e rodar aplicativos Java.



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.



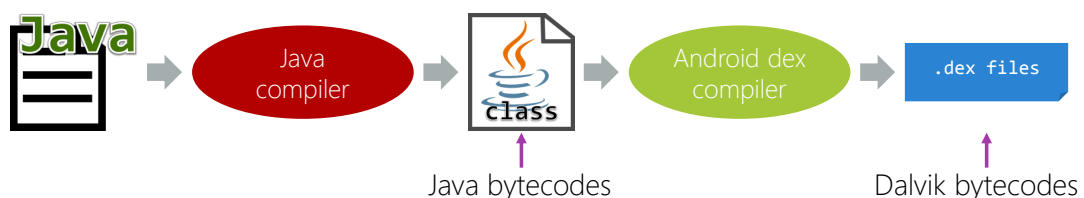
Bibliotecas core, como o java.\* e o android.\*

Bibliotecas opcionais

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

## Compilação

O código fonte Java é compilado em Dalvik bytecode para a implantação.



## Empacotamento

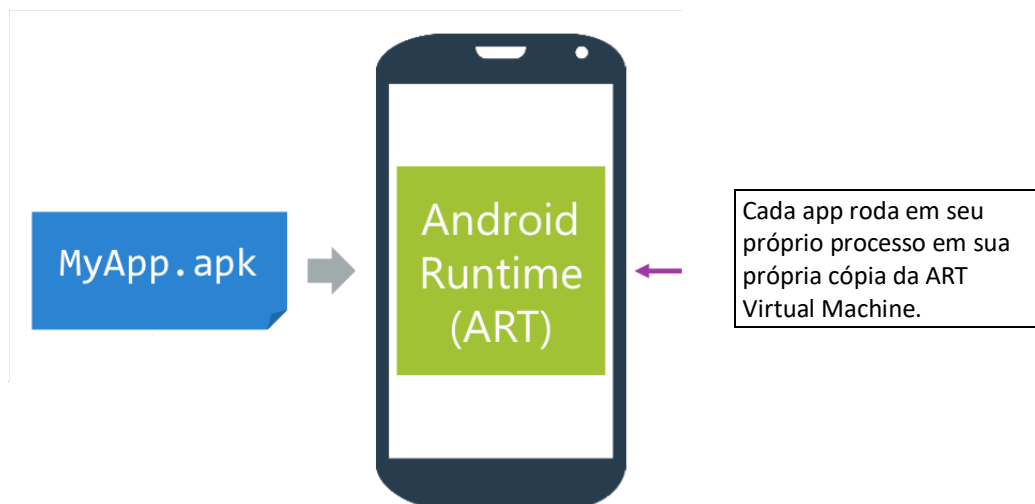
O bytecode do App e seus assets (suas imagens, ícones, arquivos de dados, etc.) são combinados em um Application Package (arquivo .apk) para implantação.

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



## Execução

O Android Runtime (ART) 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.



## Atualizações da SDK

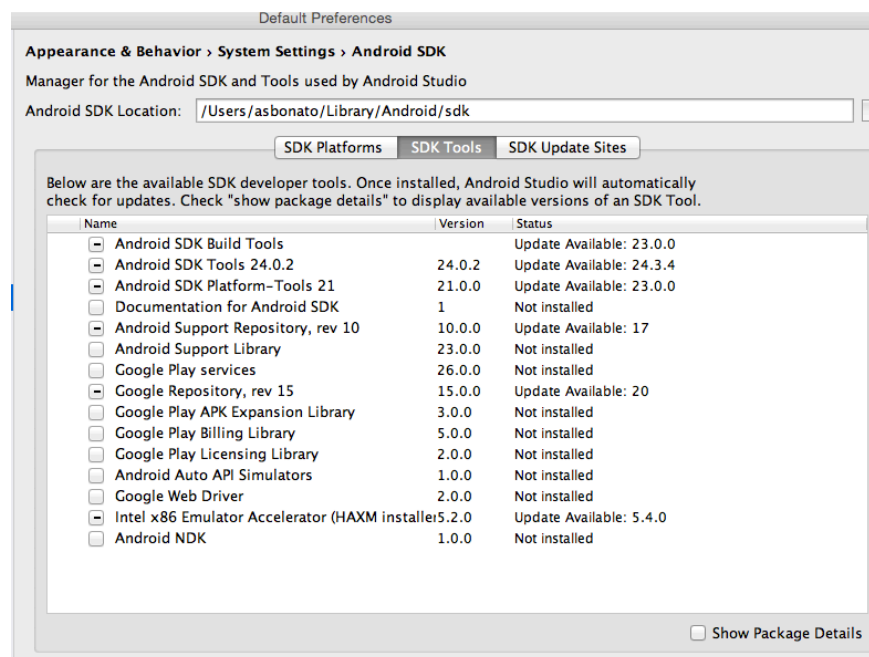
Você precisa atualizar manualmente sua Plataforma e Ferramentas Android SDK de modo a contruir apps nas últimas versões do Android. As versões, como vimos acima, são nomeadas por um nome código (sobremesa), um número e cada uma possui um nível de API. Este nível de API é usado para controlar as combinações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

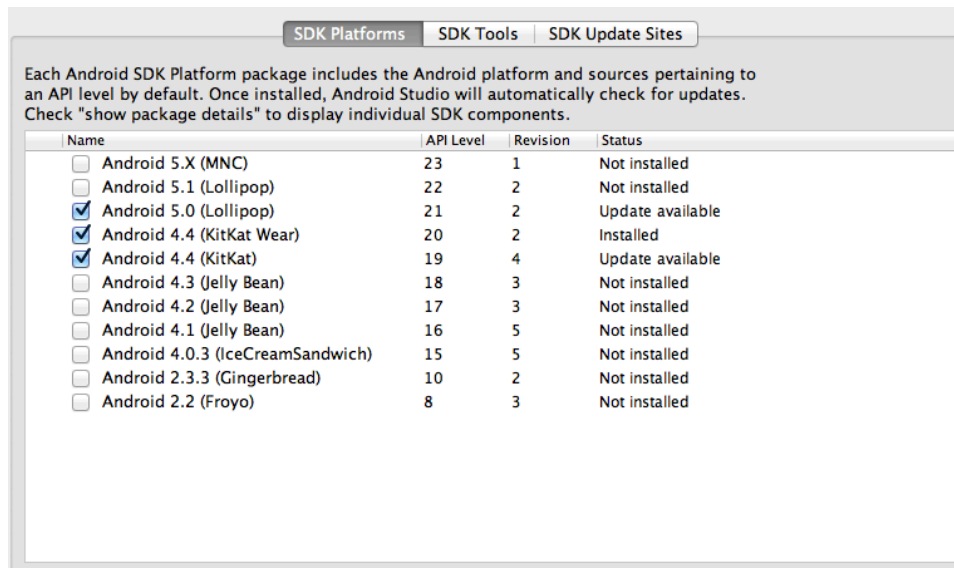
## O SDK Manager

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.



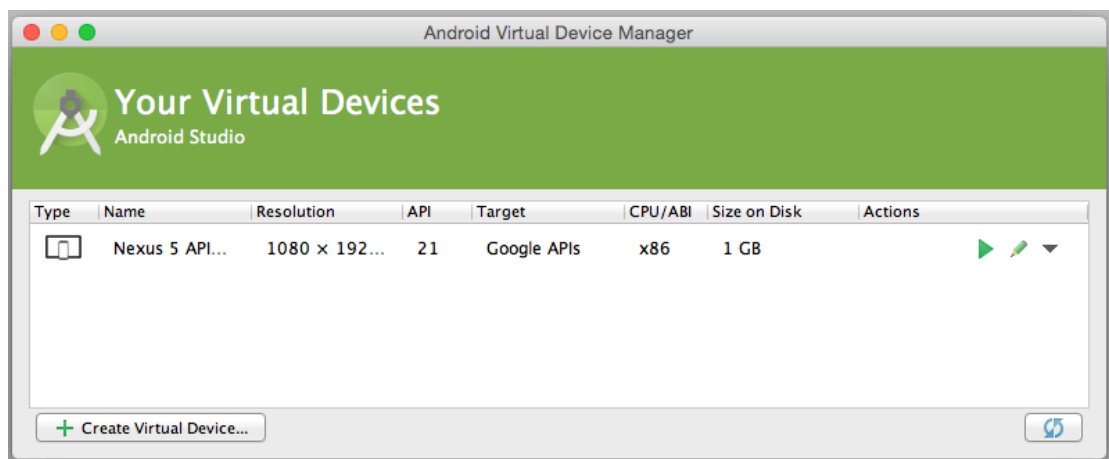
Use o SDK Manager também para atualizar as versões da plataforma para as quais você pretende desenvolver. **Instale somente as que precisar, pois consomem bastante espaço em disco.**



## Emuladores

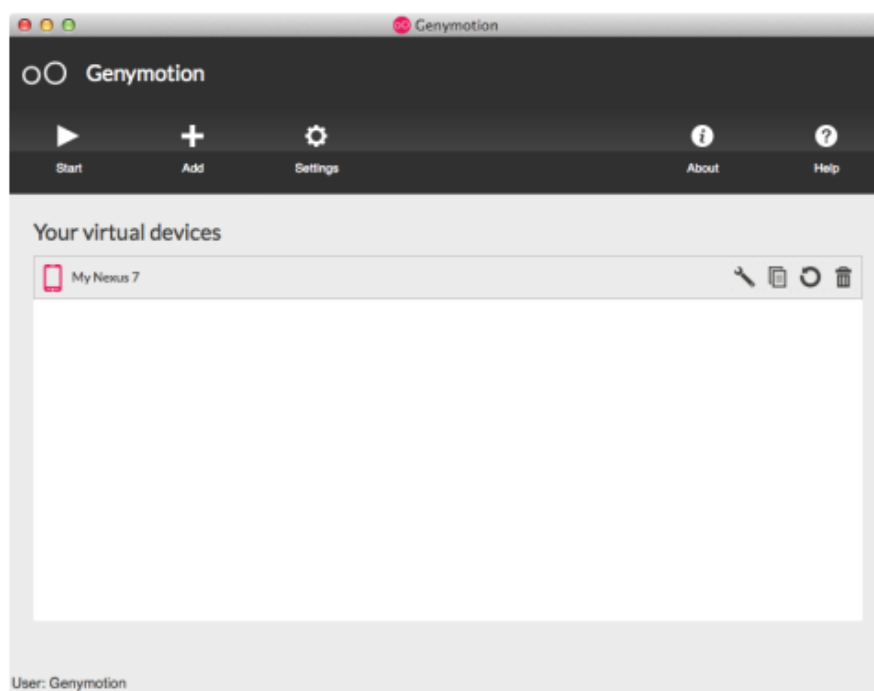
Para a execução do aplicativo durante o desenvolvimento há duas alternativas. A execução em um dispositivo real ou o uso de emuladores.

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



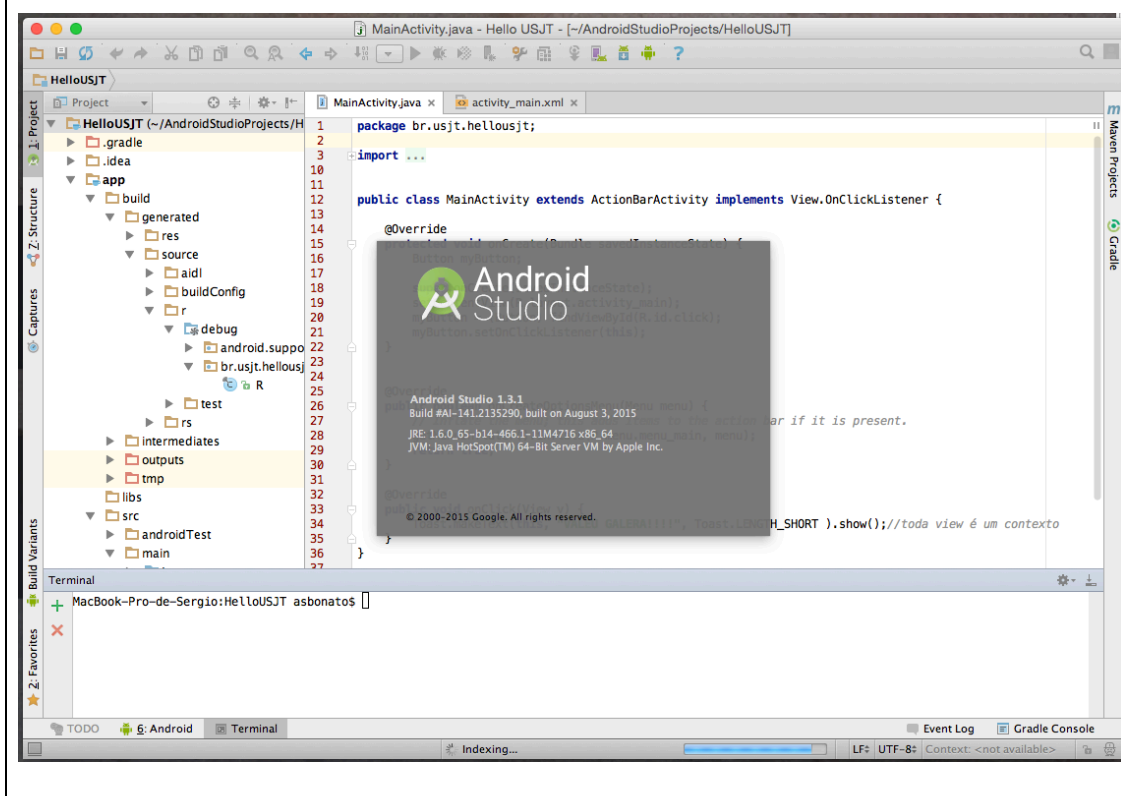
Para criar um novo emulador, é necessário clicar em Create Virtual Device e configurar os parâmetros. Não se esqueça que você deve ter um device configurado para o nível de API que estiver desenvolvendo o app.

Porém, o emulador fornecido pelo Google é excessivamente lento. Nos laboratórios não podemos usar por causa de licenciamento (grátis só para uso pessoal), mas na sua máquina sugerimos usar o Genymotion, que roda dentro de uma VM Virtual Box. O processo de criação de dispositivos é semelhante ao AVD visto acima.



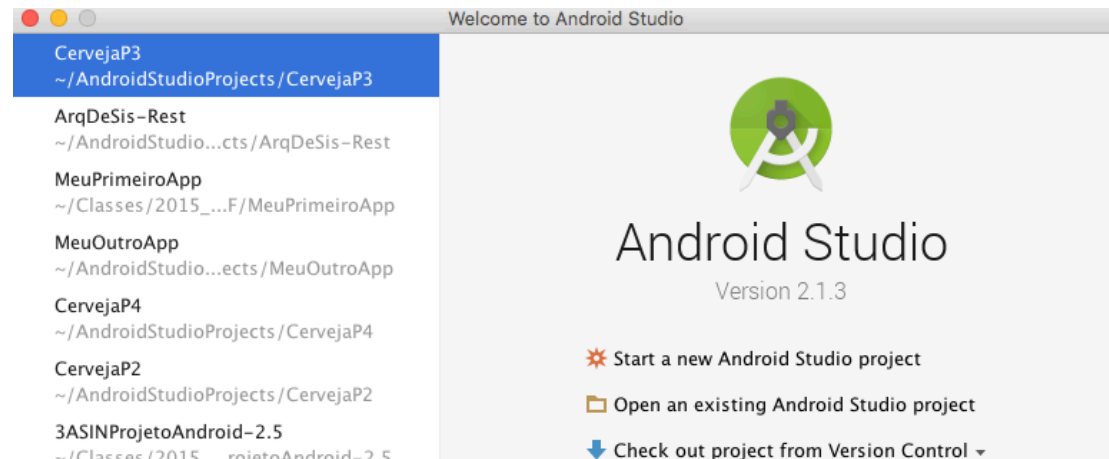
## A IDE de Desenvolvimento

A IDE que iremos usar é o Android Studio, criado pelo Google a partir da IDE IntelliJ. Há uns dois anos a ferramenta usada era o Eclipse com um plugin para Android. Porém, o Android Studio tem performance muito melhor. A Android Studio será melhor conhecido no tutorial a seguir.

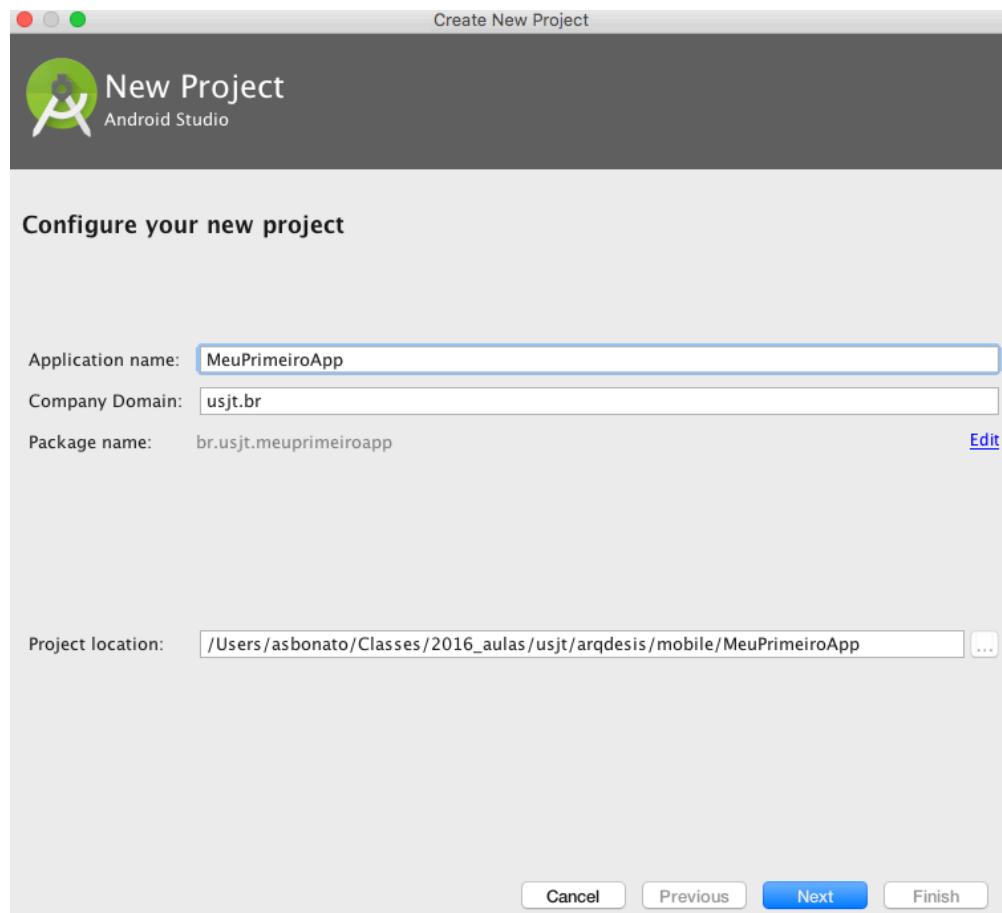


## Exercício: Meu Primeiro App Android

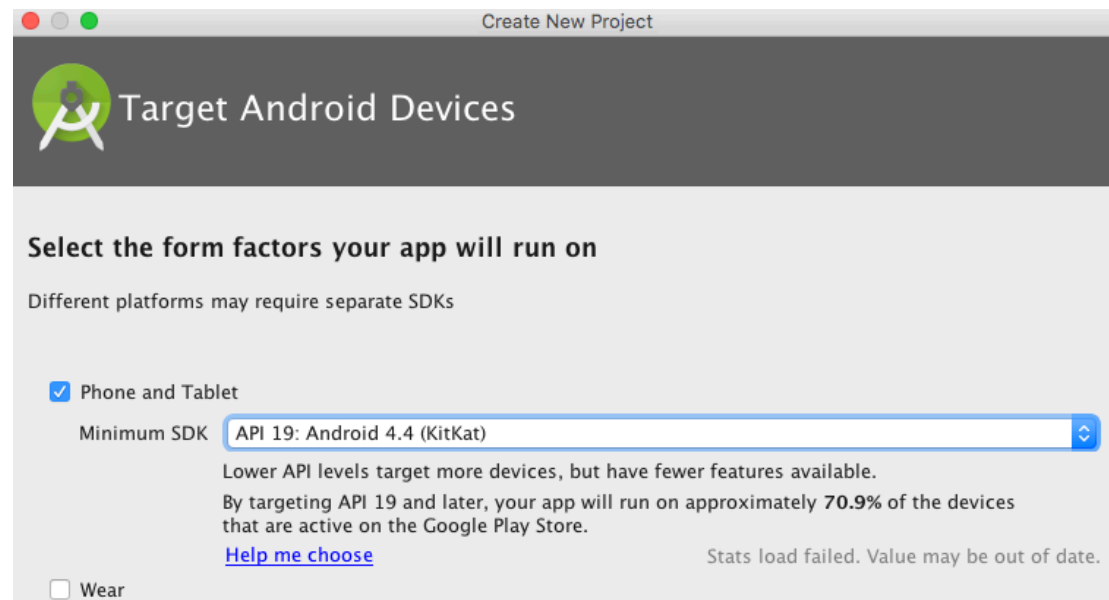
1. Abra o Android Studio e clique em Start a New Android Studio Project.



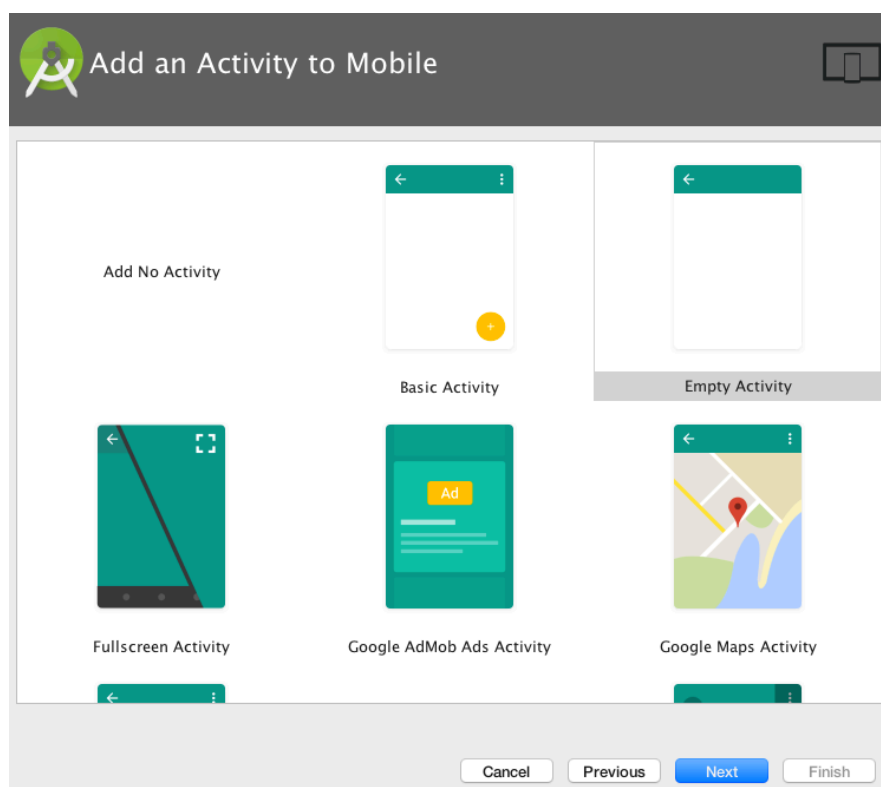
2. Dê um nome para seu projeto. Coloque a url da universidade ao contrário como company domain. Escolha a localização do projeto no seu file system (um diretório onde você tenha direito de gravação). Clique em Next.



3. Escolha a API alvo para Phone e Tablet (deixe as outras sem clicar). Geralmente, você deve usar uma que abranja o maior número de dispositivos possível, mas também deve levar em consideração as novas funcionalidades que irá perder ao fazer isso. Entretanto, para este tutorial, escolha uma versão cuja API esteja instalada no seu SDK. Clique em Next.

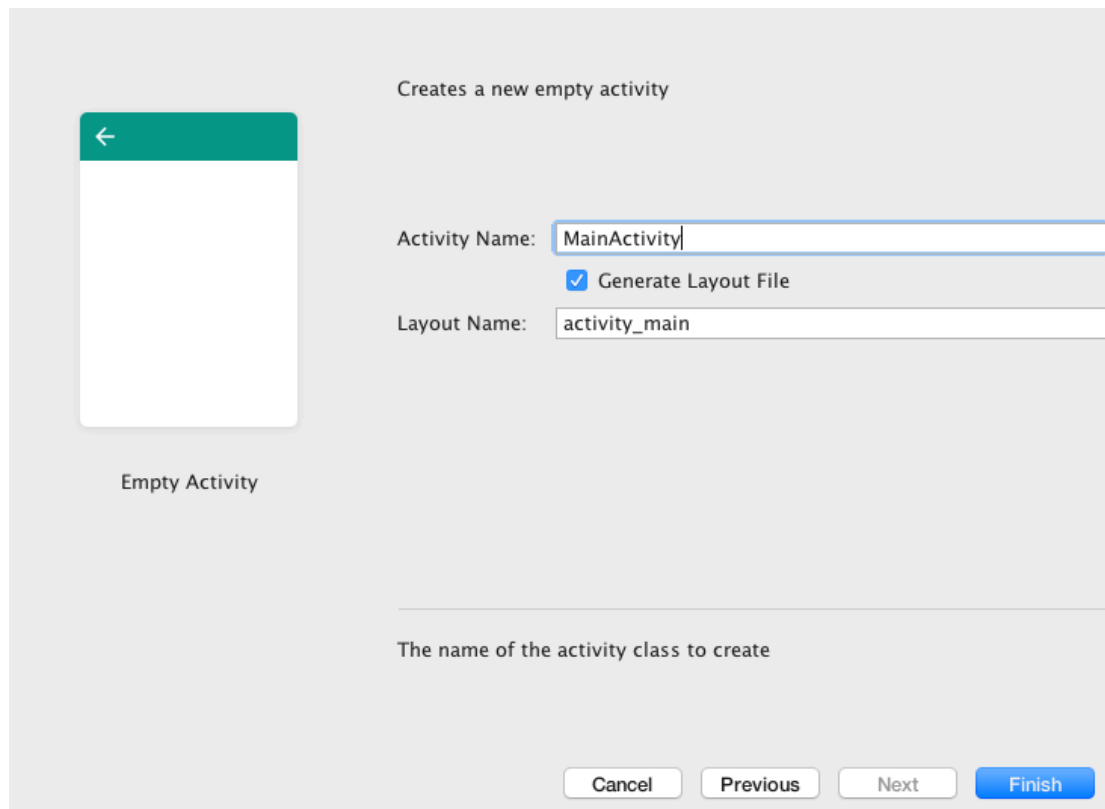


4. Escolha uma Empty Activity (veremos mais sobre Activities nas próximas aulas) e clique Next.

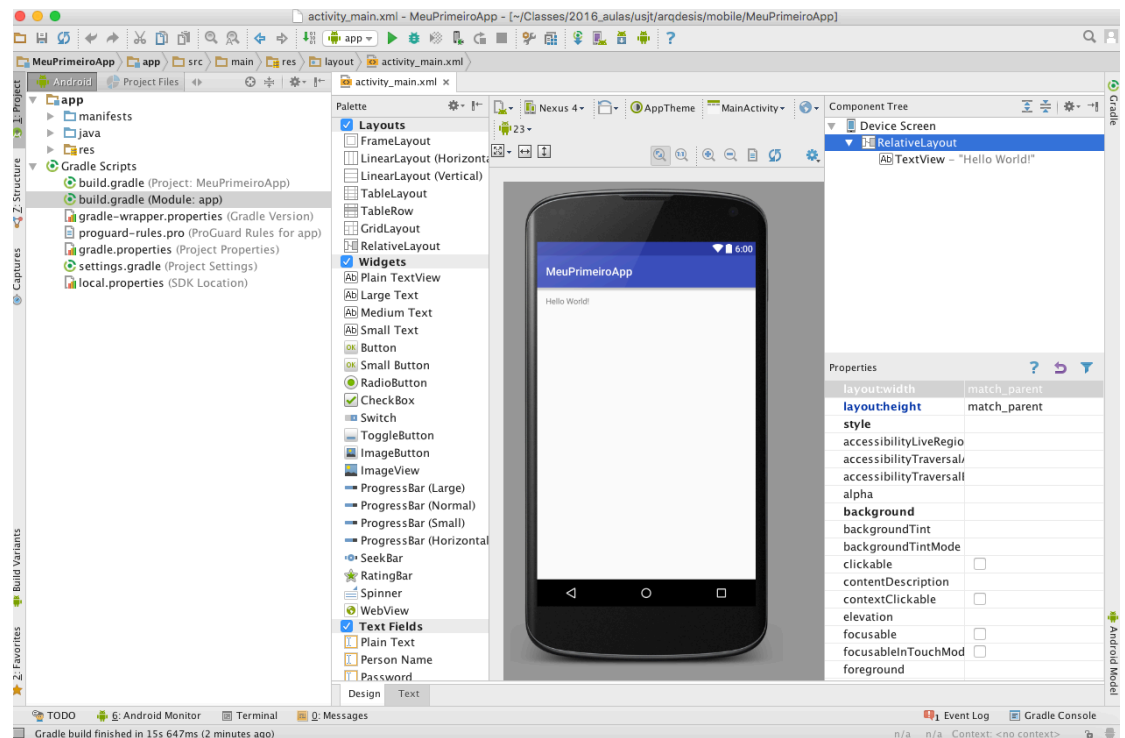




5. Pode deixar os nomes sugeridos pelo Studio para a Activity e clique em Finish.



6. O Studio irá criar seu projeto. Sua tela deverá parecer com a tela abaixo. 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.

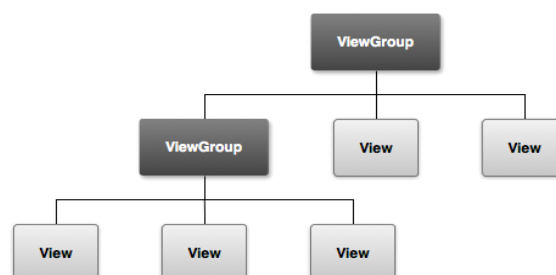


## 7. Uma breve explicação da estrutura do projeto:

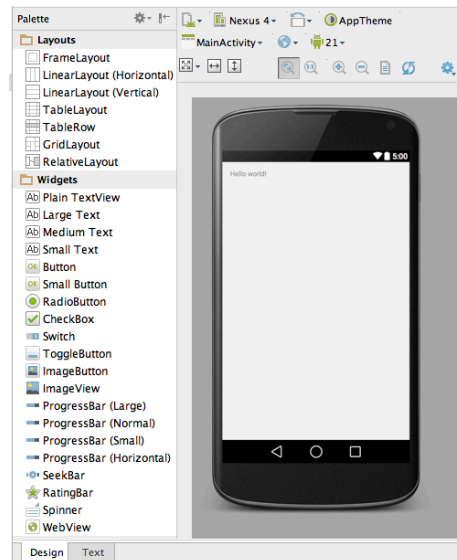
- **AndroidManifest.xml** define cada um dos componentes do app; mexa nele com cuidado
- **src/** contém o código fonte; tem uma Activity que roda quando o app é iniciado
- **bin/** contém os .class gerados
- **gen/** arquivos gerados pelo ADT; nunca mexa neles!
- **res/** contém vários subdiretórios 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

## 8. 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 contêineres invisíveis definem o layout dos views



9. Clique no diretório res/layout. O 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).



```
<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:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp" tools:context=".MainActivity">

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

</RelativeLayout>
```

10. 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 orientation.
- 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.

```
<?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">

</LinearLayout>
```

11. 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 se 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>
```

## 12. 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>
```

## 13. 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">

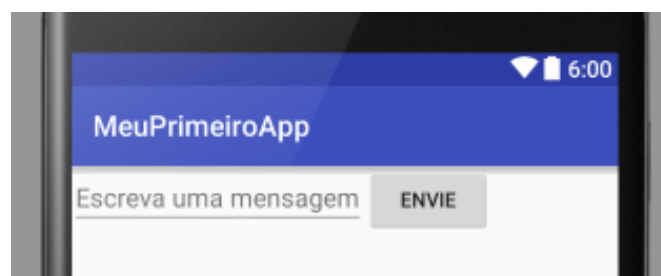
    <EditText
        android:id="@+id/edit_message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="@string/edit_message"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_send"/>

</LinearLayout>

```

14. 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 `Odp`. 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" />

```

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



16. 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"/>
```

## 17. 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){
}
```

## 18. 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);
}
```

## 19. 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 startActivity passando o intent como parâmetro

```
//constante static para identificar a mensagem
public final static String EXTRA_MESSAGE =
```

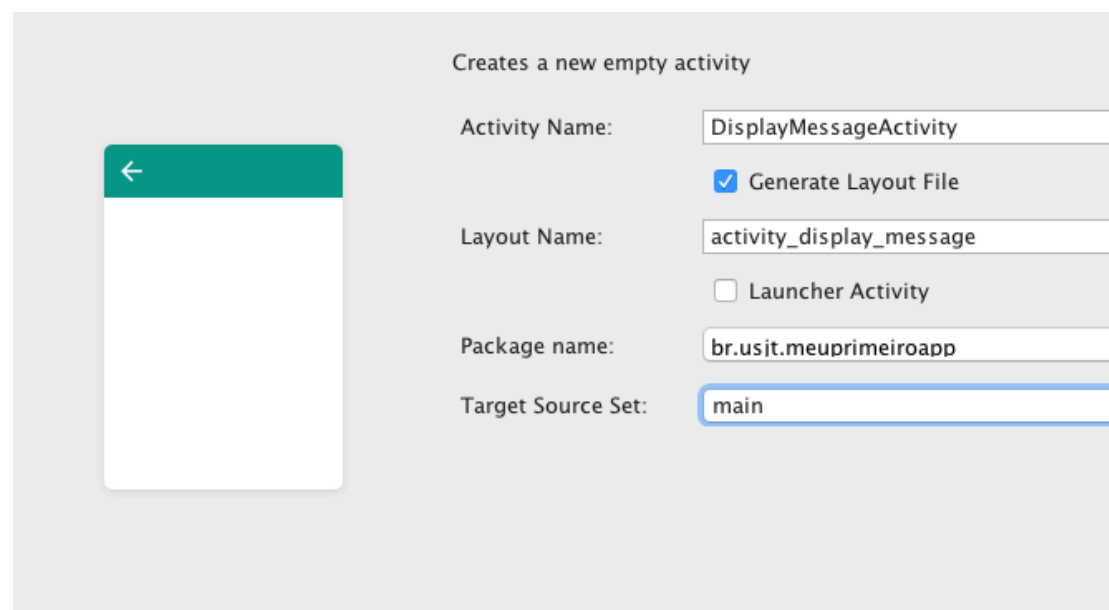
```

"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);
}

```

## 20. 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>

```



## 21. 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()`.

## 22. Valide se foi adicionada a nova Activity no manifesto

- Abra o AndroidManifest.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>

```

23. 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 release

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.