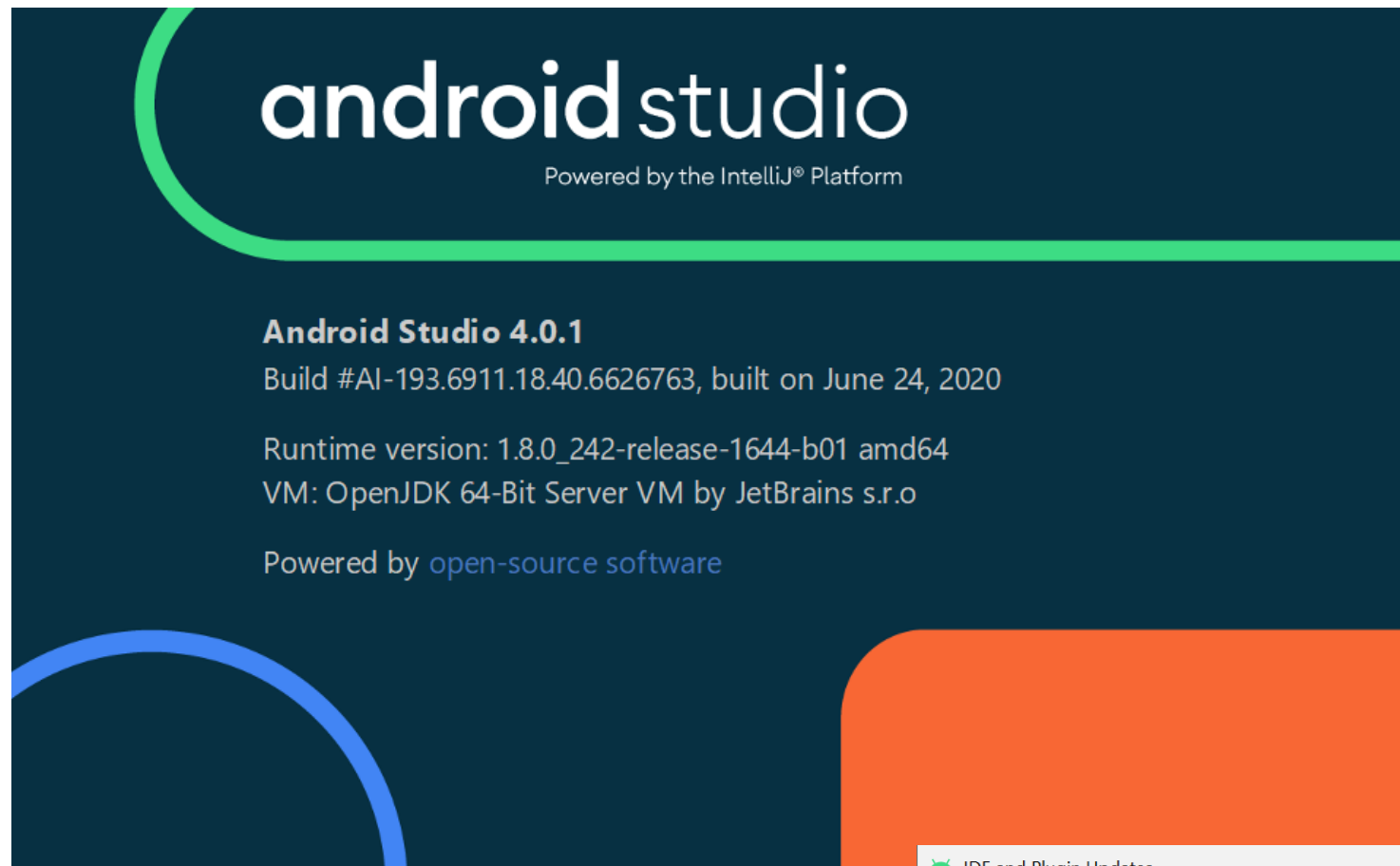


Programação para Dispositivos Móveis

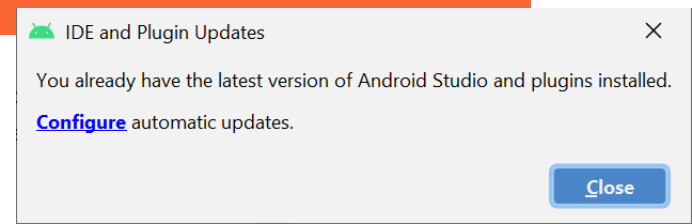
Conteúdo

- Preparando o Android Studio
- Componentes de um app
- Ciclo de vida de uma Activity
- Estrutura de arquivos
- Layouts, identificadores, retrato/paissagem
- Internacionalização
- Exemplos com ImageView / TextView / ScrollView
- Exportando um .zip do projeto
- Gerando um arquivo de instalação APK
- Exercícios

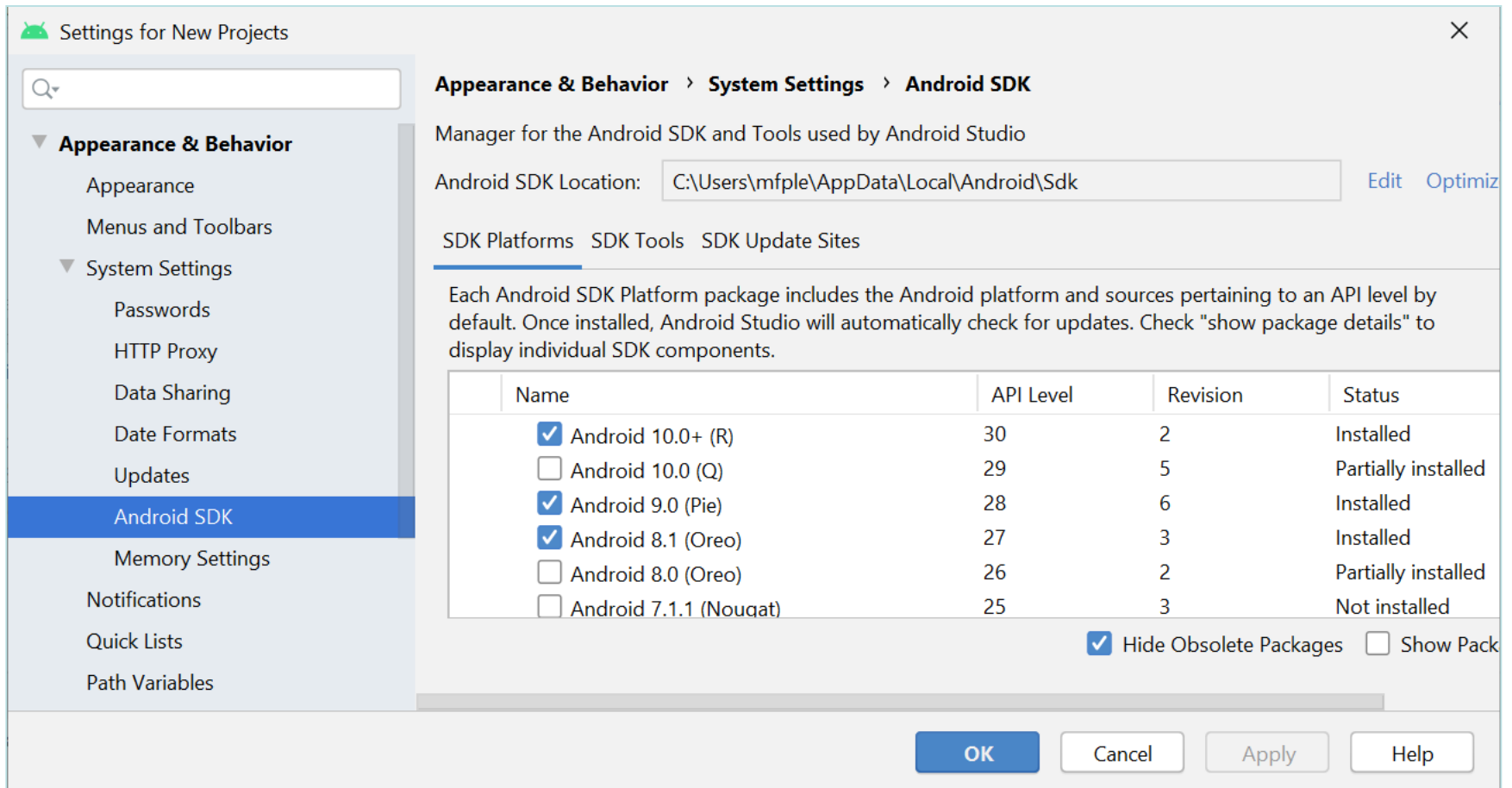
Android Studio - atualize para a última versão



Em 13/08/2020 a última versão é Android Studio **4.0.1**. Utilize a opção **Help / Check for Updates...** para atualizar plugins etc.



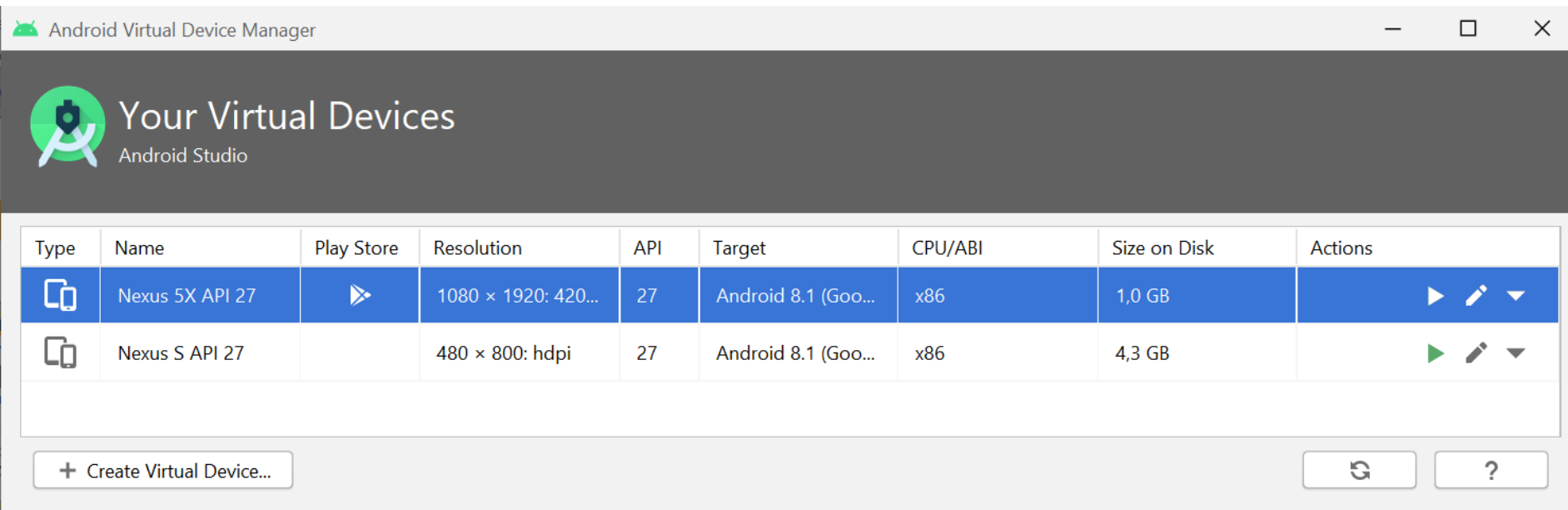
As versões de Android com o SDK Manager



Sugerimos instalar a última versão do SDK, mas também o **Android 8.1 (Oreo)**, que é a **API 27**, porque muitos exemplos da disciplina foram preparados para a API 27.

A sugestão e a figura acima são de 13/08/2020.

Emuladores: Android Virtual Device (AVD) Manager



Nas práticas, vamos usar vários emuladores para testar nossos apps, por exemplo:

- **Nexus 5X** **API 27** (o Nexus 5X tem uma resolução bastante padrão, 1080 x 1920)
- **Nexus S** **API 27** (o Nexus S tem uma resolução pequena, 480 x 800)

Sugerimos instalar estes dois emuladores.

Curiosidade: as parcerias da empresa Google

- 2010 - Nexus One - **HTC** (Taiwan)
- 2010 - Nexus S - **Samsung** (Coreia do Sul)
- 2011 - Galaxy Nexus - **Samsung** (Coreia do Sul)
- 2012 - Nexus 4 - **LG** (Coreia do Sul)
- 2014 - Nexus 5 - **LG** (Coreia do Sul)
- 2014 - Nexus 6 - **Motorola** (USA, Lenovo, China)
- 2015 - Nexus 5X e Nexus 6P - **Huawei** (China)
- 2016-2020 - **Google** (USA) - Google Pixel, Pixel XL, Pixel 2, Pixel 2 XL, Pixel 3, Pixel XL 3, Pixel 4, Pixel 4 XL...



Nexus One, HTC, 2010

Obs: a partir de 2016 a Google projetou e desenvolveu seus próprios aparelhos. Veja mais dados sobre estes aparelhos em: <https://www.techtudo.com.br/noticias/2019/03/do-nexus-ao-pixel-relembre-todos-os-celulares-criados-pelo-google.ghtml>

Componentes de um app Android

Activities	Activity: telas do app	}	Geralmente presentes nos apps
View/UI	Componentes da UI		
Intent	Ações/interação no app		
Services	Execução de tarefas em segundo plano	}	Podem estar ou não presentes nos apps, depende do projeto
Content Provider	Disponibilidade dos dados entre app		
Broadcast Receiver	Comunicação entre SO e apps		

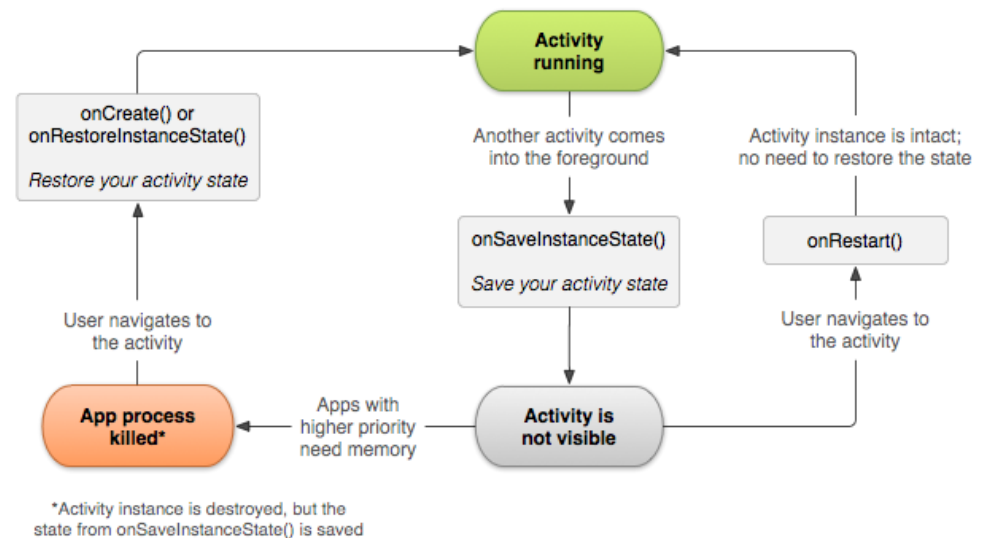
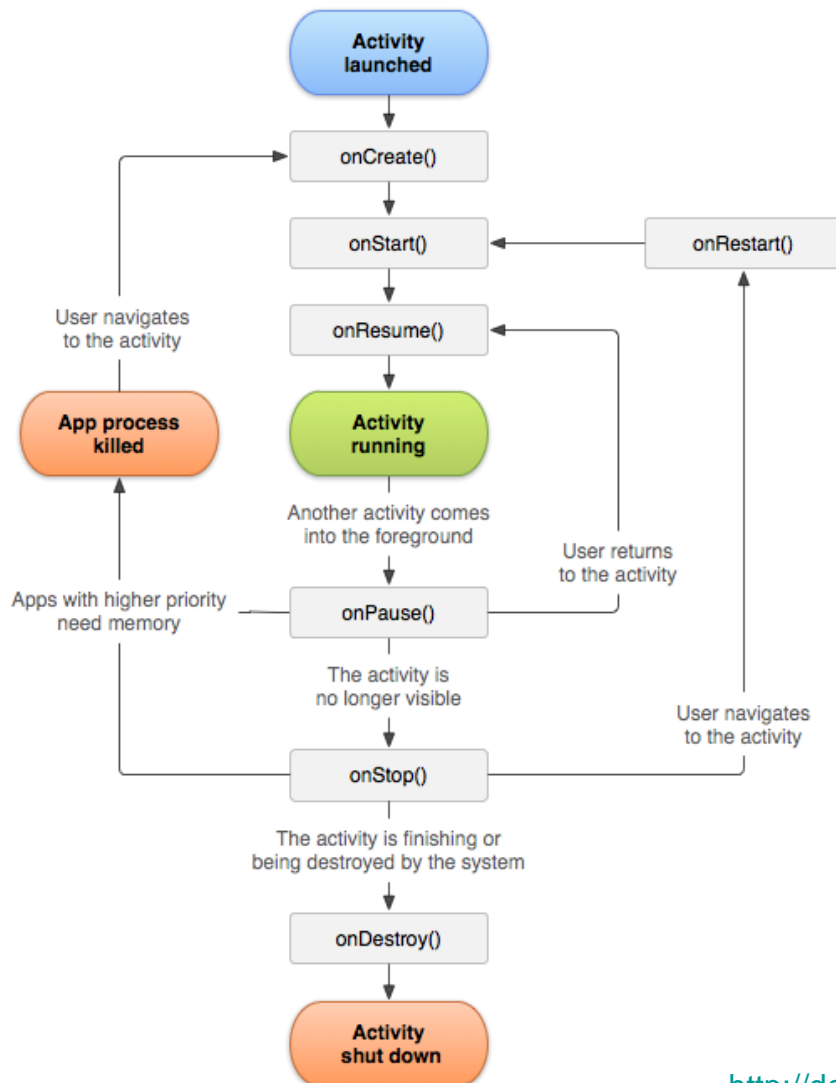
Ciclo de vida de uma Activity (eventos)

onCreate ()	Método chamado quando a Activity é criada.
onStart ()	Método chamado quando a Activity é iniciada e se torna visível.
onPause ()	Método breve chamado quando a Activity entra em pausa, não está mais no primeiro plano (mas poderia estar visível), ao ser abandonada pelo usuário ou para o sistema atender outro aplicativo. A Activity perde o foco. Uma Activity em pausa pode ser retomada (onResume) ou parada (onStop).
onResume ()	Método chamado quando a Activity é retomada, recebe o foco.
onStop ()	Método chamado quando a Activity fica em background e não está mais visível. O aplicativo poderá pausar animações, salvar dados em um banco de dados etc. neste evento. Depois de um onStop a Activity poderá ser destruída ou reiniciada (veja os próximos dois eventos).
onDestroy ()	Método chamado antes da Activity ser destruída.
onRestart ()	Método chamado quando a Activity é reiniciada após um stop.
onSaveInstanceState()	O sistema chama este método "à medida que a atividade começa a parar". Salve aqui estados de IU leves, simples.

Ciclo de vida de uma Activity

Uma Activity pode estar:

- 1 – Ativa
- 2 – Pausada (ainda visível)
- 4 – Parada
- 5 – Encerrada



Salvando o estado da Activity

Exemplo - testando o ciclo de vida de uma Activity

```
public class MainActivity extends AppCompatActivity {
```

```
    private String msg="";

    @Override
    protected void onCreate(Bundle savedInstanceState) {

    }

    @Override
    protected void onStart() {

    }

    @Override
    protected void onResume() {

    }

    @Override
    protected void onPause() {

    }

    @Override
    protected void onStop() {

    }

    @Override
    protected void onDestroy() {

    }

    @Override
    public void onSaveInstanceState(Bundle outState) {

    }

    @Override
    protected void onRestoreInstanceState(Bundle savedInstanceState) {

    }

    public void enviarSMS(View v) {

    }
}
```

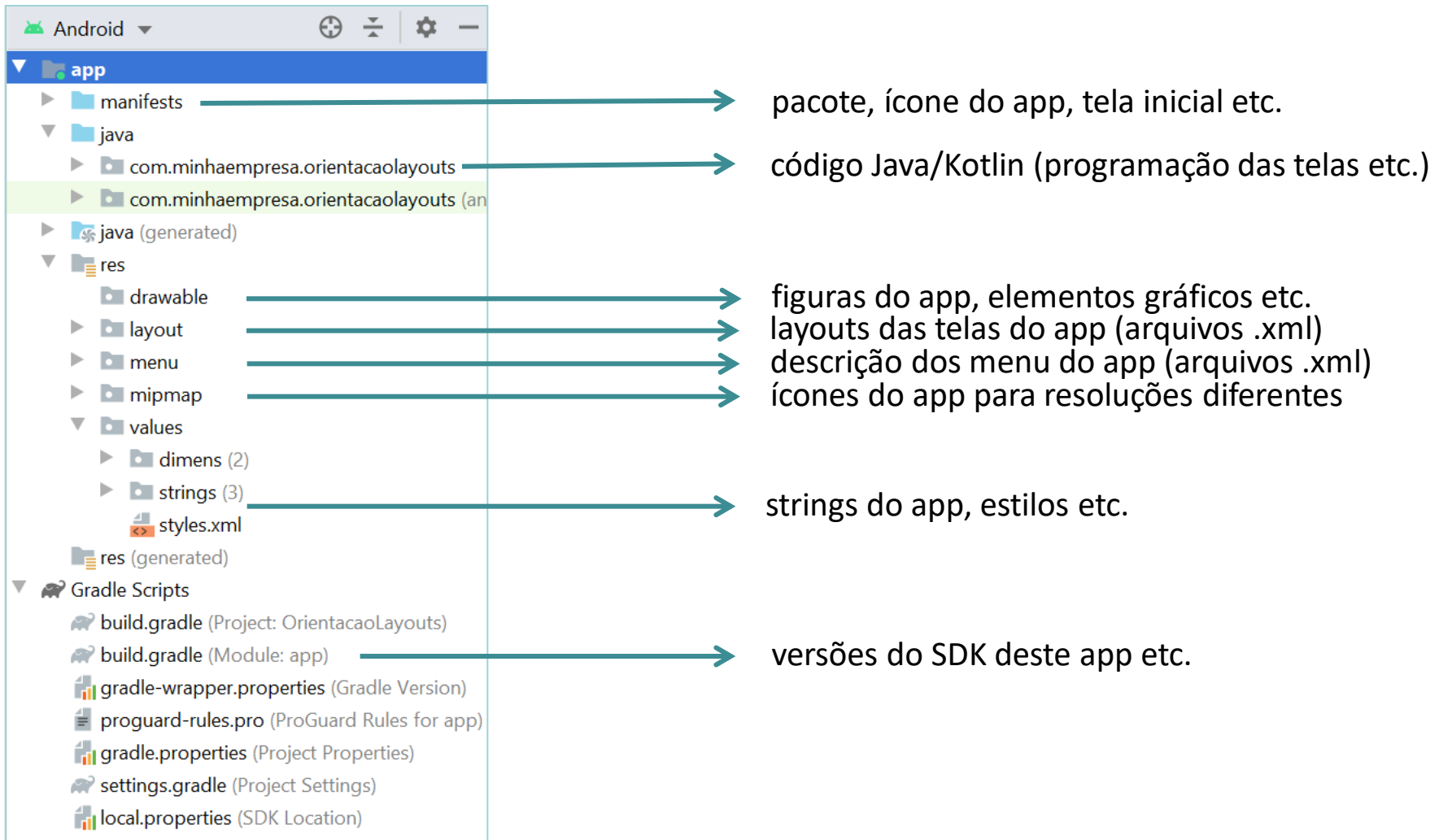
Visualizemos as informações no **painel LogCat**.

Sintaxe: `Log.e("tag name", "mensagem");`



Entrou no método onCreate()
 Entrou no método onStart()
 Entrou no método onResume()
 Entrou no método onPause()
 Entrou no método onSaveInstanceState()
 Entrou no método onStop()
 Entrou no método onDestroy()
 Entrou no método onCreate()
 Entrou no método onStart()
 Entrou no método onRestoreInstanceState()
 Entrou no método onResume()

Estrutura de arquivos de um projeto



Android Studio

O arquivo AndroidManifest.xml

- Arquivo **AndroidManifest.xml**
 - Descreve o que nossa aplicação é, contém a lista de telas e itens como segurança e permissões

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.alcides.myapplication" >
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="My Application"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="My Application" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

- Arquivo **R.Java**

Possui os IDs dos recursos utilizados no projeto. Você não precisa mexer neste arquivo, ele é criado automaticamente pela ferramenta. Caso mostre algum erro neste arquivo, apague ele que a ferramenta o criará novamente.

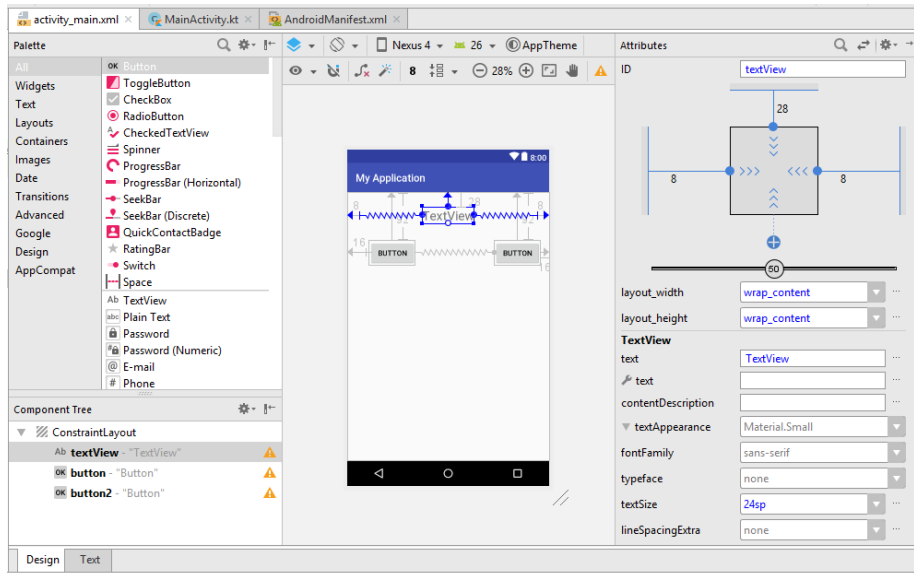
Layouts para uma tela

- Para desenhar as telas de um aplicativo utilizaremos determinadas estratégias para colocação de componentes, conhecidas como **layouts**.
- Uma tela poderá utilizar diferentes combinações de layouts, uns dentro de outros, consecutivos etc.
- Em Android existem diferentes tipos de layouts, como:
 - LinearLayout
 - RelativeLayout
 - TableLayout
 - ConstraintLayout (é o padrão das novas activities criadas no Android)

<https://developer.android.com/guide/topics/ui/declaring-layout?hl=pt-BR>

Layout de uma tela

Android guarda o projeto do visual da tela em um arquivo .xml. Cada componente da interface será uma tag no .xml. Observe no exemplo a seguir que **TextView** e **Button** são classes do Android.



Android Studio

XML

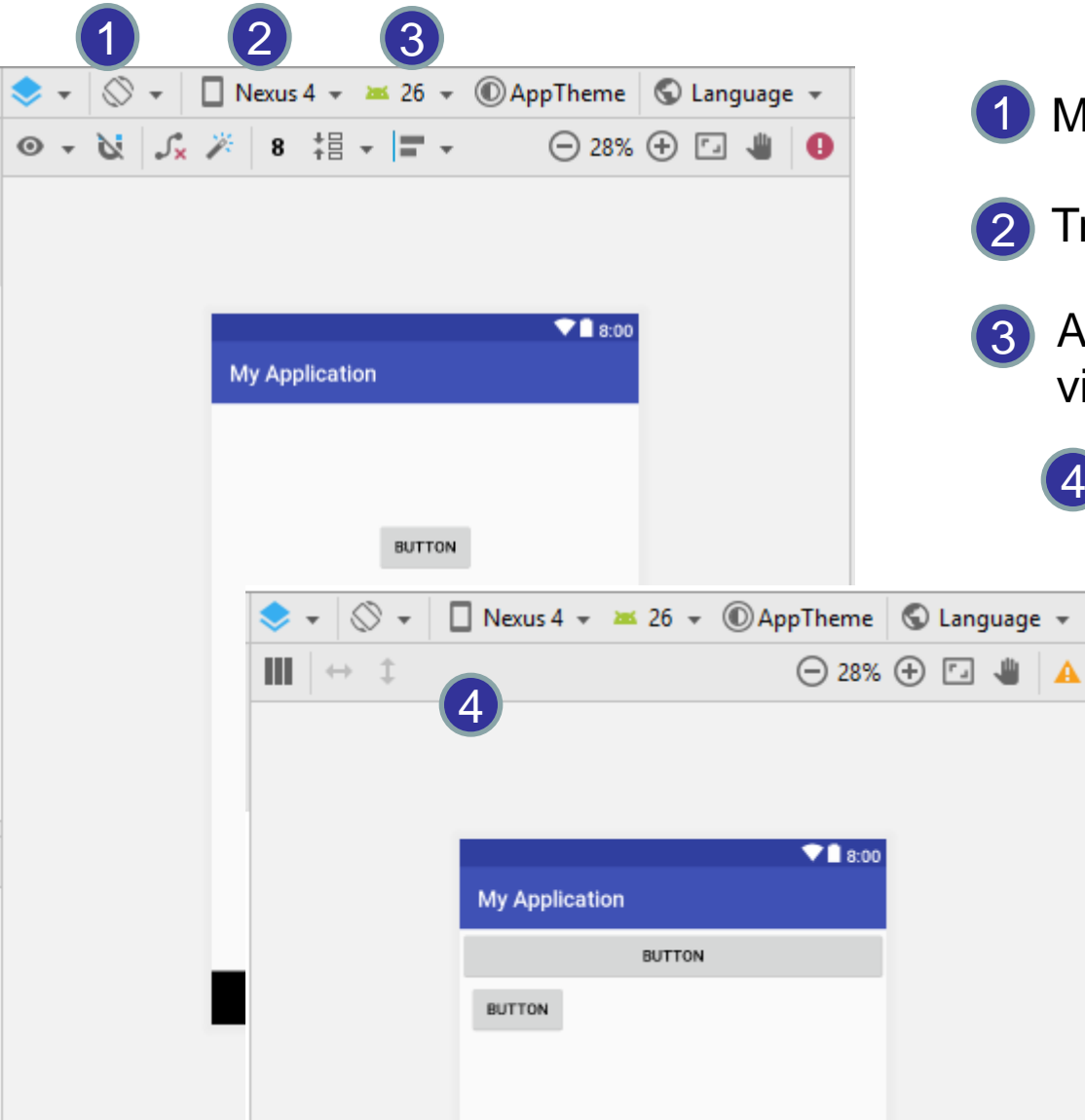
```
<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello_world" />

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView1"
    android:layout_below="@+id/textView1"
    android:layout_marginLeft="42dp"
    android:layout_marginTop="41dp"
    android:text="Button" />

<Button
    android:id="@+id/button2"
    style="?android:attr/buttonStyleSmall"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@+id/button1"
    android:layout_marginLeft="46dp"
    android:layout_toRightOf="@+id/button1"
    android:text="Button" />
```

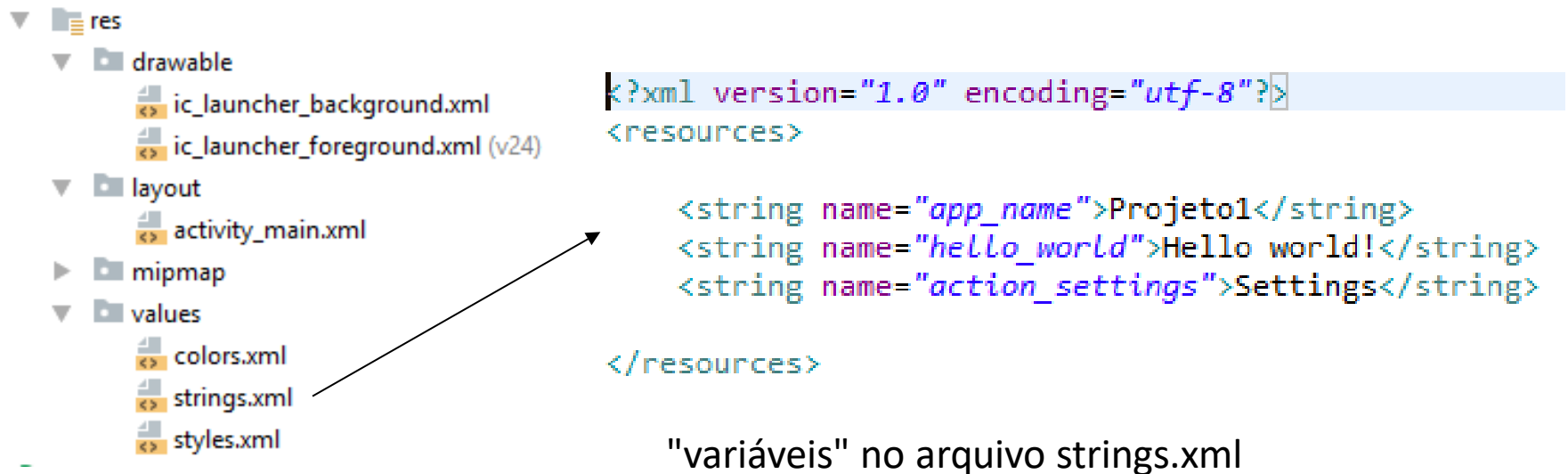
Alguns testes com a tela (antes de executar)

- 1 Muda a orientação da tela
- 2 Troca o dispositivo para visualização
- 3 Altera a versão de Android na visualização
- 4 Ferramentas que auxiliam a manipulação do layout conforme o modelo escolhido



Layout do projeto - identificadores vs. constantes

- Quando criamos nosso Layout, será melhor trabalhar com "variáveis" (identificadores) para os elementos de rótulo, botões etc., ou seja, criamos "variáveis" nos arquivos .xml para cada informação textual, porque isso irá facilitar a manutenção da interface e posteriormente a **internacionalização do app**.
- As variáveis e seus valores ficam no arquivo chamado **strings.xml** na pasta **values**. Na verdade, podemos ter diferentes arquivos strings.xml, um para cada idioma suportado.



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

    <string name="app_name">Projeto1</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>

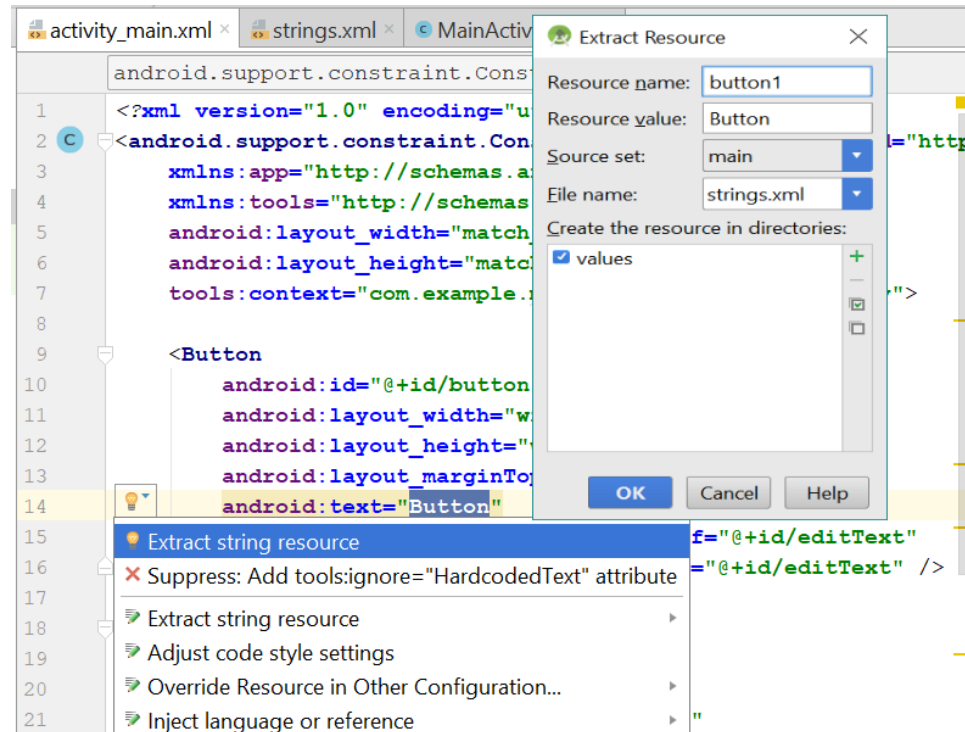
</resources>
```

"variáveis" no arquivo strings.xml

Layout do projeto - identificadores

No arquivo .xml da tela:

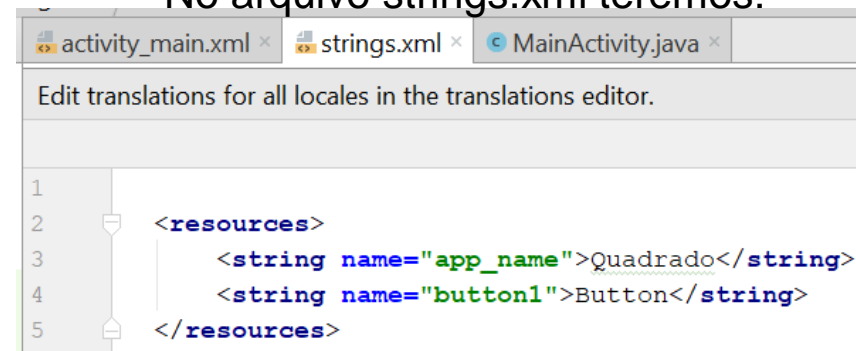
- selecione o texto
- pressione **Alt+Enter**
- escolha **Extract String Resource**



No arquivo .xml da tela em questão:

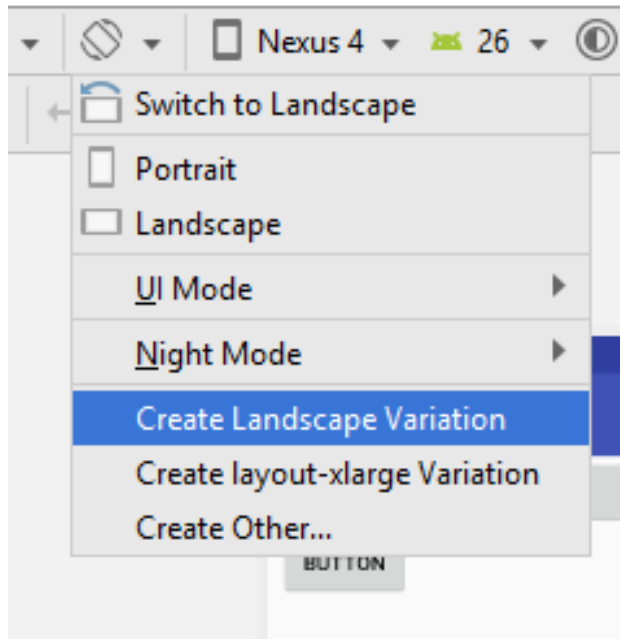
```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="36dp"
    android:text="@string/button1"
    app:layout_constraintStart_toStartOf="@+id/editText"
    app:layout_constraintTop_toBottomOf="@+id/editText" />
```

No arquivo strings.xml teremos:



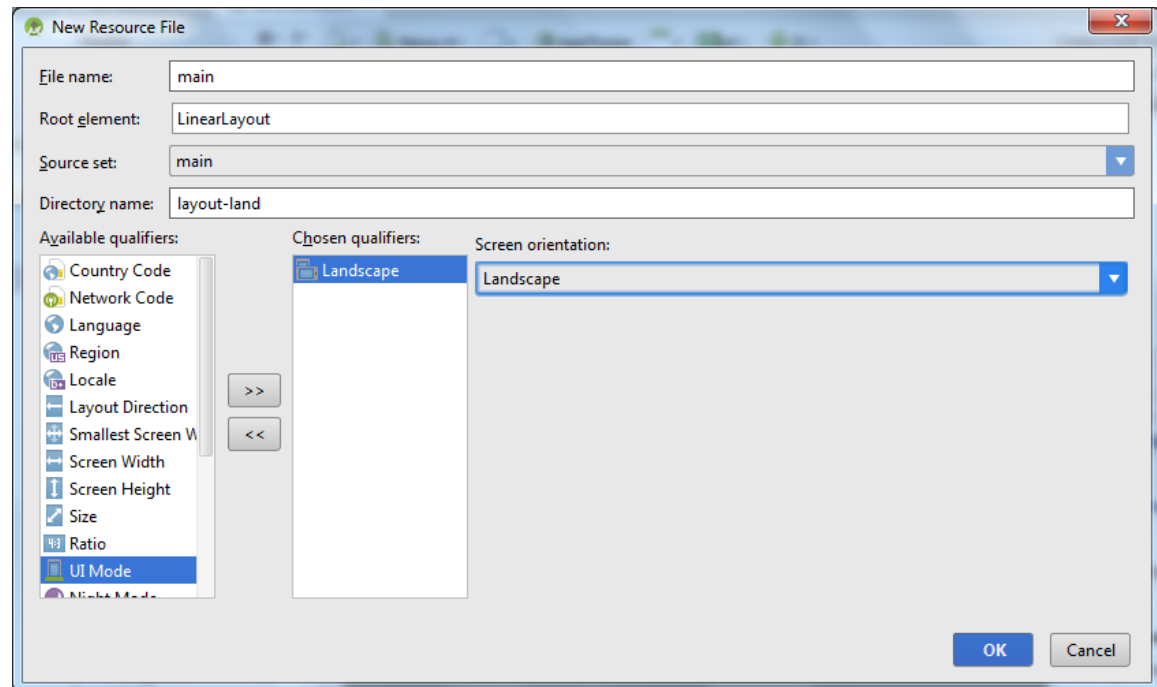
Criando um layout landscape (paisagem)

Opção 1. Melhor, porque faz uma cópia das tags XML.



É importante testar o app nas orientações landscape e portrait (paisagem e retrato).

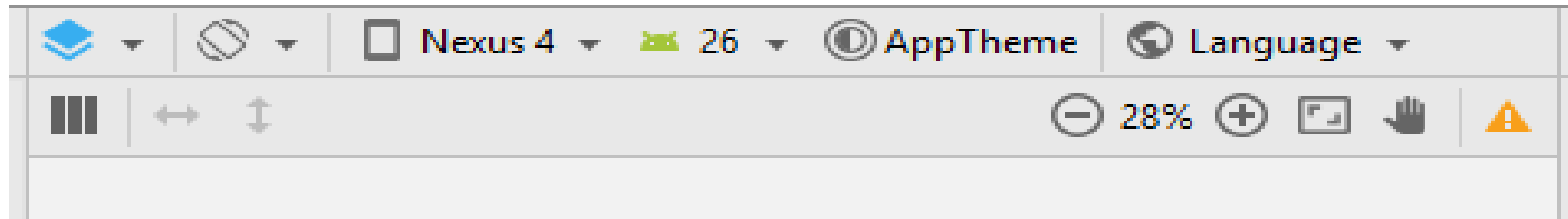
Opção 2. Clique com o botão direito sobre a pasta layout, escolha New >> Layout resource file. Adicione o item Orientation e depois escolha Landscape.



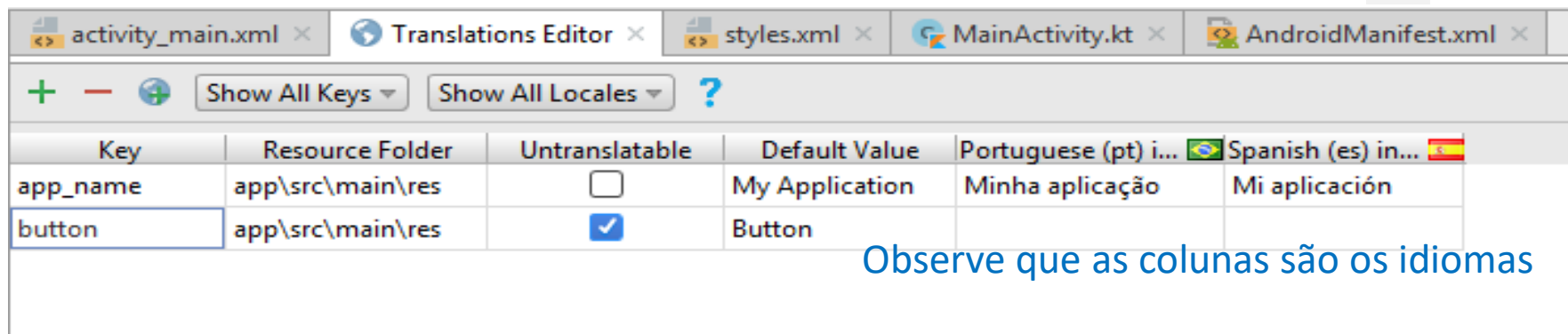
Internacionalização

- Frequentemente necessitamos elaborar um aplicativo com versões em vários idiomas (multilíngue) ou até para países, regiões ou costumes específicos, para que o universo de usuários potenciais seja maior e nosso programa atenda determinadas especificações.
- A tecnologia Android fornece alguns recursos especiais para garantir esse objetivo em forma automática.
- Lembre-se de colocar todas as informações textuais da interface através de variáveis (identificadores) no arquivo strings.xml (veja em um slide anterior).
- Existem várias formas de gerar os textos para outros idiomas. Após finalizar a interface em um idioma (com o uso do arquivo strings.xml), você poderá clicar no Globo (na área de layout) e configurar outros idiomas.
- Dependendo da configuração de idioma do aparelho, os dados serão apresentados no idioma correspondente definido pelo app!

Internacionalização - no Android Studio

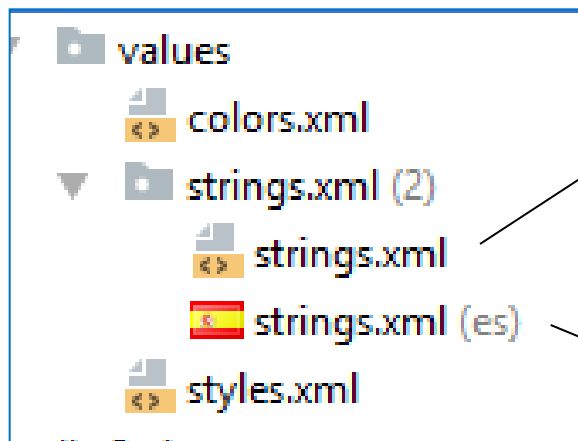


Para inserir outros idiomas: clique em Language / Edit Translations... / botão 



Key	Resource Folder	Untranslatable	Default Value	Portuguese (pt) i...	Spanish (es) in...
app_name	app\src\main\res	<input type="checkbox"/>	My Application	Minha aplicação	Mi aplicación
button	app\src\main\res	<input checked="" type="checkbox"/>	Button		

Observe que as colunas são os idiomas



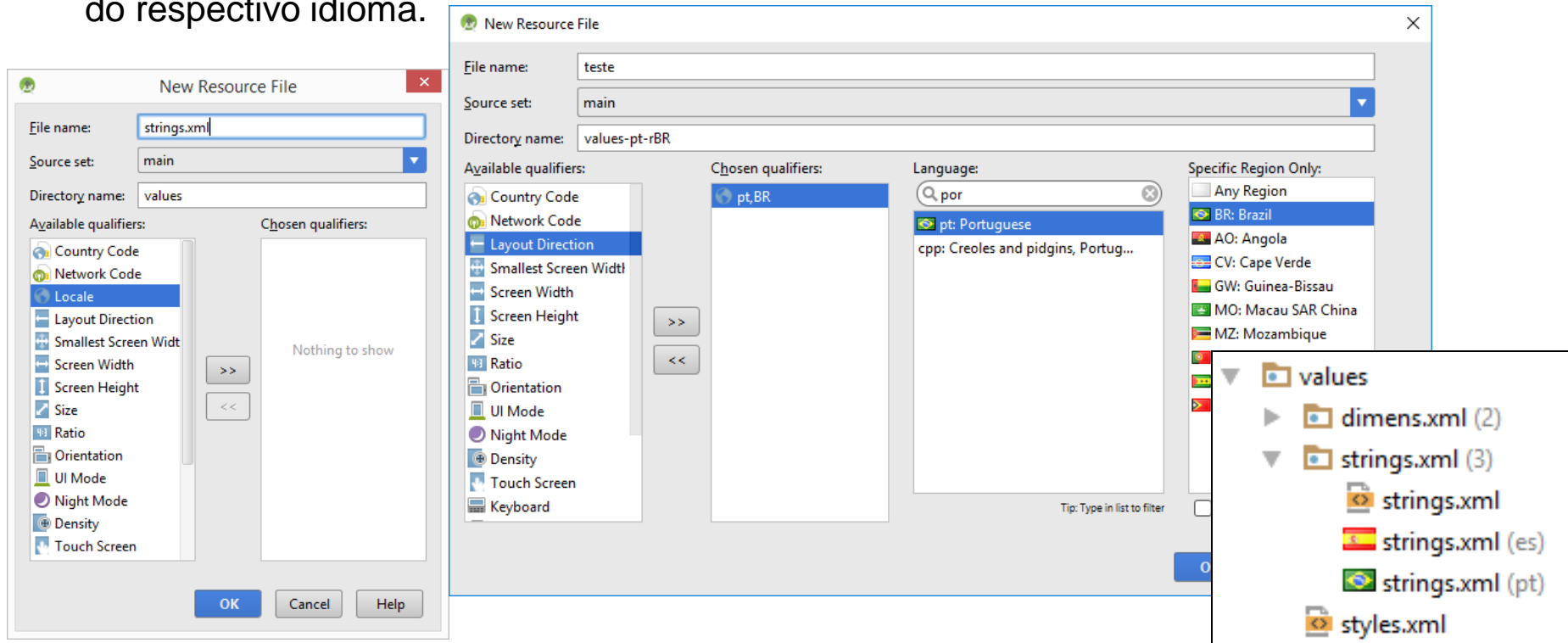
Finalmente, teremos arquivos de textos para cada idioma suportado:

```
<resources>
    <string name="app_name">My Application</string>
</resources>
```

```
<resources>
    <string name="app_name">"Mi aplicación "</string>
</resources>
```

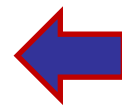
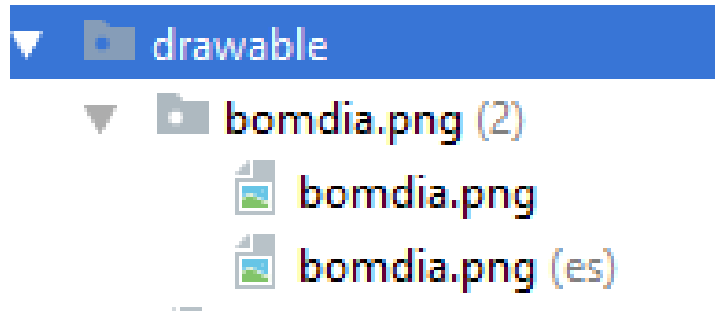
Internacionalização - no Android Studio

- **Segunda forma:** Clique com o botão direito do mouse na pasta values e escolha New>>Values Resource file (Android Studio)
- Insira o nome do arquivo como strings, em seguida escolha a opção Locale e a linguagem do seu interesse.
- Após criar o novo idioma, copie, cole e traduza o conteúdo do arquivo strings.xml da pasta do respectivo idioma.

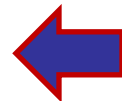


Internacionalização

- A internacionalização também funciona para **imagens**.
- Neste caso **deve ser criada uma pasta para cada idioma no recurso drawable**, as imagens devem ter o mesmo nome e devem ser copiadas para a respectiva pasta drawable do idioma.
- O Android Studio facilita a criação e a cópia de arquivos para diferentes idiomas.
- Devemos criar as pastas dos idiomas e quando for copiar as imagens, selecionar a pasta que se deseja salvar a mesma, lembre-se que a imagem (o arquivo da imagem) deve ter o mesmo nome para todos os idiomas.



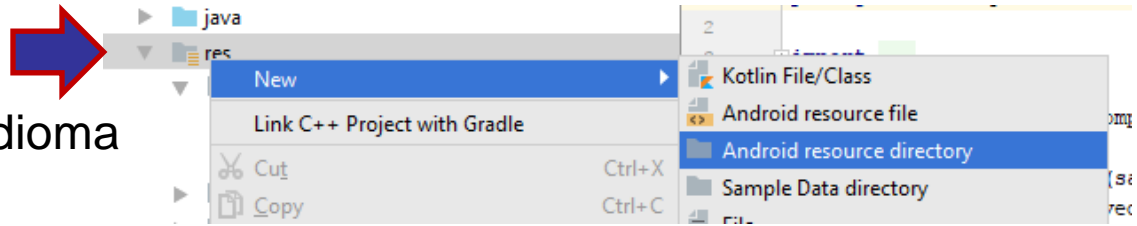
Idioma padrão, neste caso BR



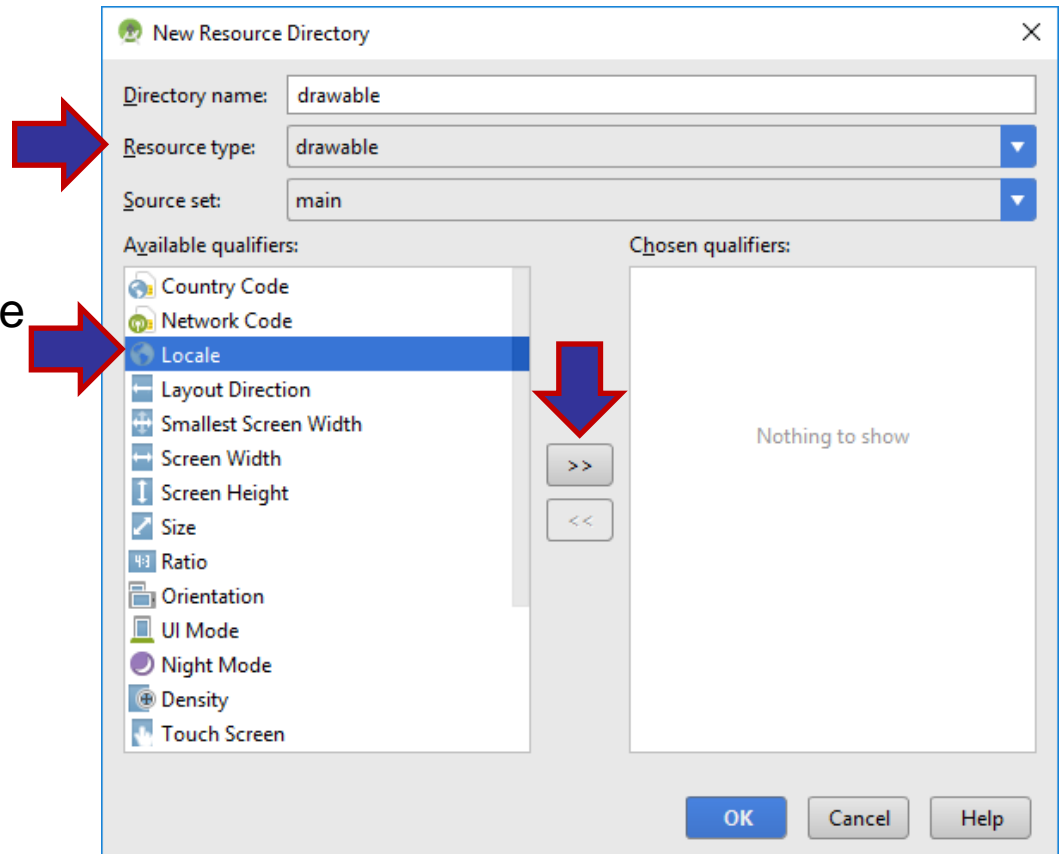
Idioma espanhol, ES

Internacionalização

Passo 1: Criar a pasta para o idioma

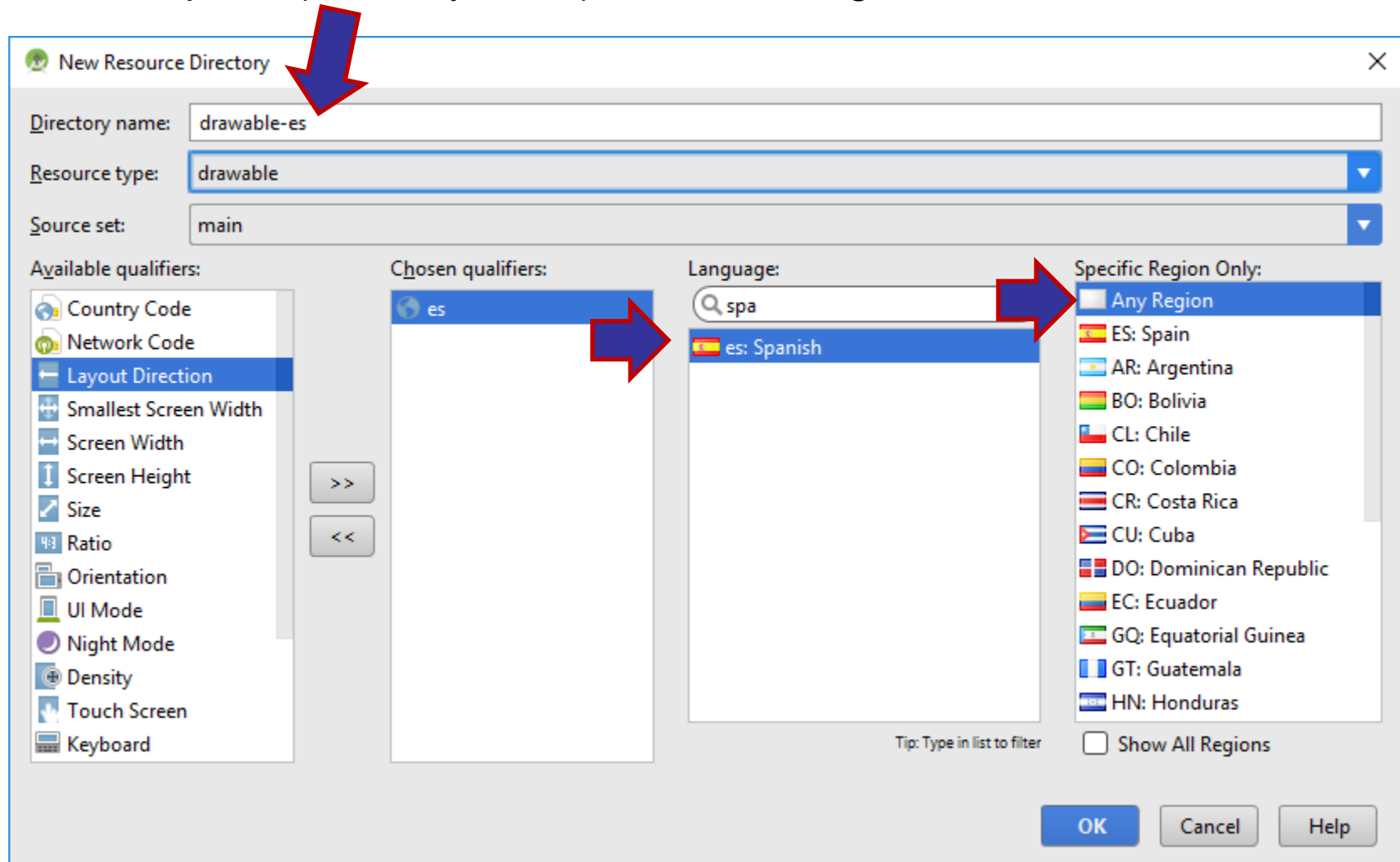


Passo 2: Altere a opção Resource type para drawable e adicione a opção Locale



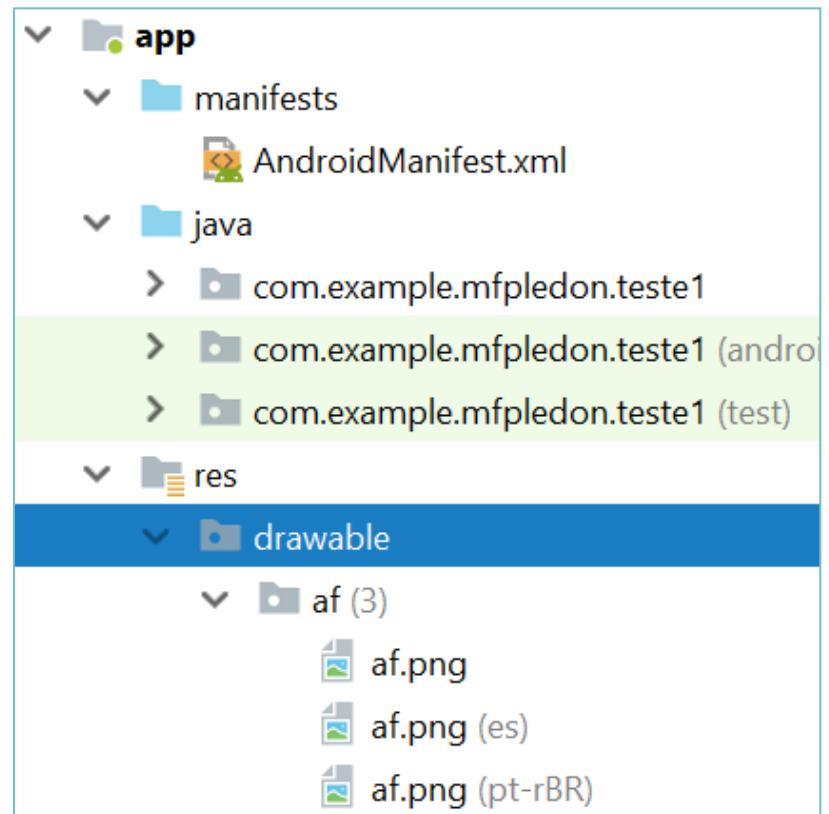
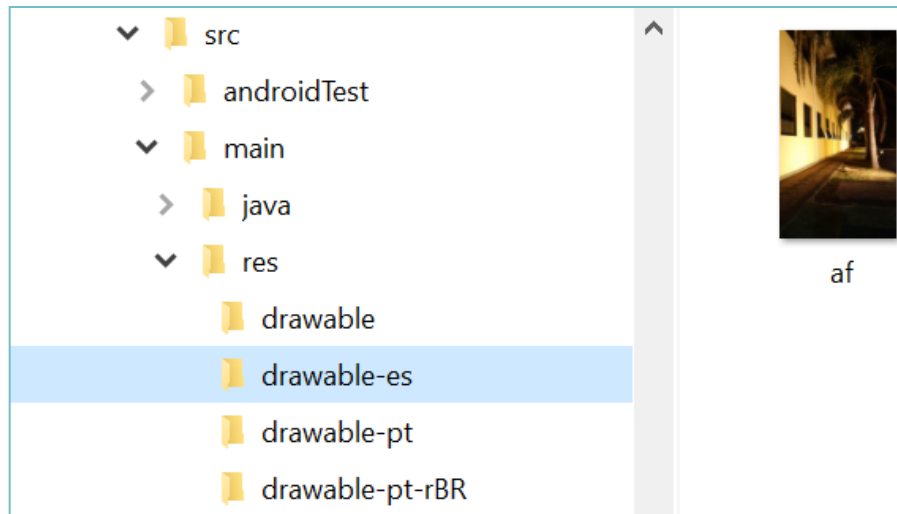
Internacionalização

O nome da pasta (Directory name) é diferente segundo o idioma selecionado.



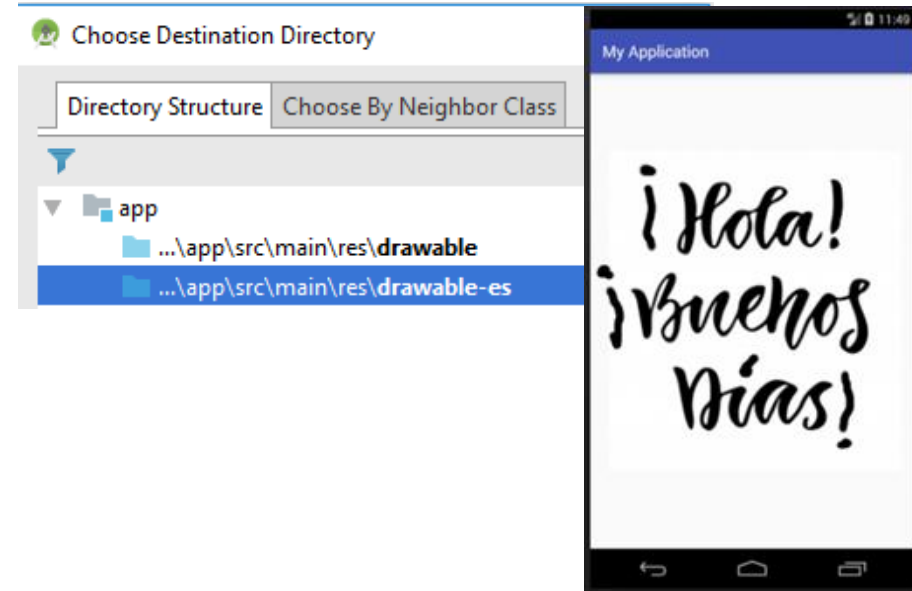
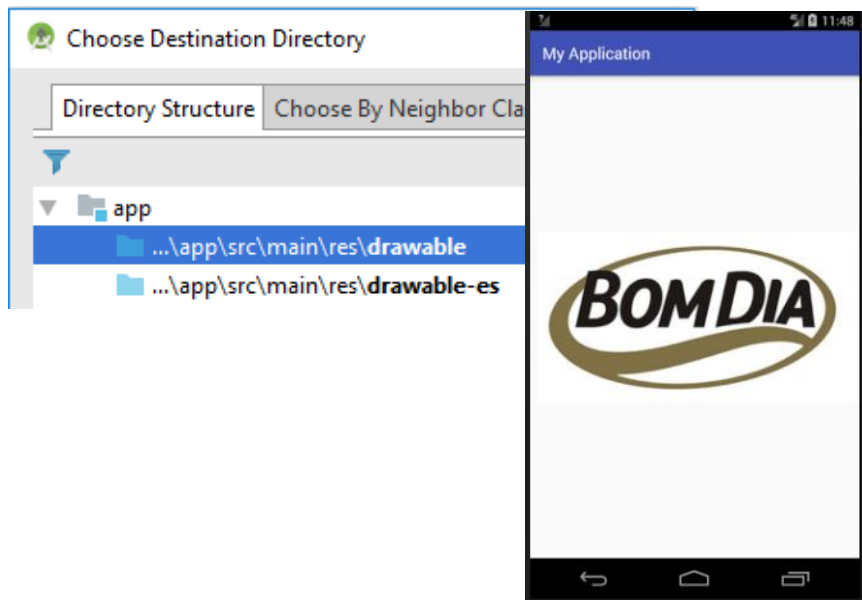
Internacionalização

O objetivo do processo anterior seria ter uma imagem diferente (af.png no exemplo a seguir) específica para cada idioma/região:



Internacionalização

- Ao colar uma imagem na pasta drawable padrão (copiando do Explorador de Windows, por exemplo), o Android Studio irá solicitar a subpasta do idioma.
- Escolha o idioma correspondente e cole a imagem.
- Quando o app é executado, a imagem é selecionada conforme o idioma.



Internacionalização - normas ISO para idiomas

- Será necessário saber o idioma e o país selecionados no aparelho Android para que o programa efetue algumas ações adicionais. Duas normas ISO estabelecem códigos de duas letras para identificar cada idioma e cada país:

"The language codes are two-letter lowercase ISO language codes (such as "en") as defined by ISO 639-1. The country codes are two-letter uppercase ISO country codes (such as "US") as defined by ISO 3166-1".

<http://developer.android.com/reference/java/util/Locale.html>

- Assim, por exemplo, as duas letras minúsculas "en" especificam idioma inglês, "es" idioma espanhol e "pt" idioma português.
- Os códigos de letras de países (duas letras maiúsculas) permitem identificar, por exemplo, Inglaterra como "GB", Espanha como "ES" e Portugal e Brasil pelas siglas "PT" e "BR".

Detectando o país e o idioma no programa Android

```
import java.util.Locale; // importamos a public final class Locale
```

```
...
```

```
int idioma=1;
```

```
String moeda="$";
```

```
Locale loc = Locale.getDefault();
```

```
String country = loc.getCountry();
```

```
String language = loc.getLanguage();
```

```
if(language.equalsIgnoreCase("en")) { // English
```

```
    idioma = 1;
```

```
    if(country.equalsIgnoreCase("GB")) moeda = "£";
```

```
}
```

```
if(language.equalsIgnoreCase("es")) { // Español
```

```
    idioma = 2;
```

```
    if(country.equalsIgnoreCase("ES")) moeda = "€";
```

```
}
```

```
if(language.equalsIgnoreCase("pt")) { // Português
```

```
    idioma = 3;
```

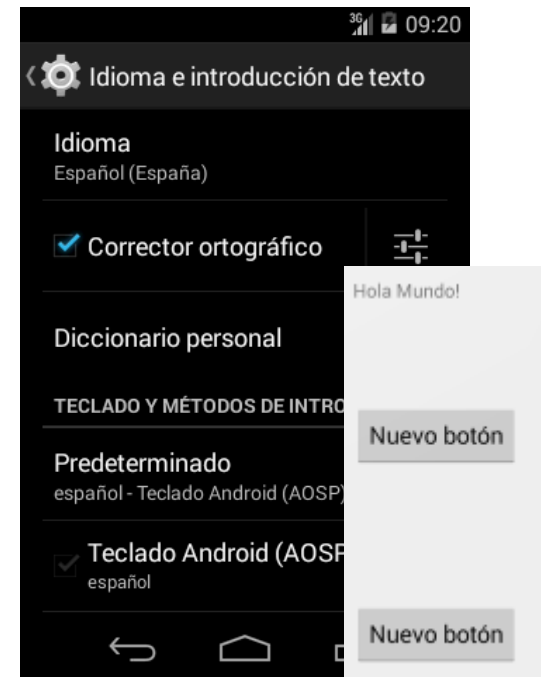
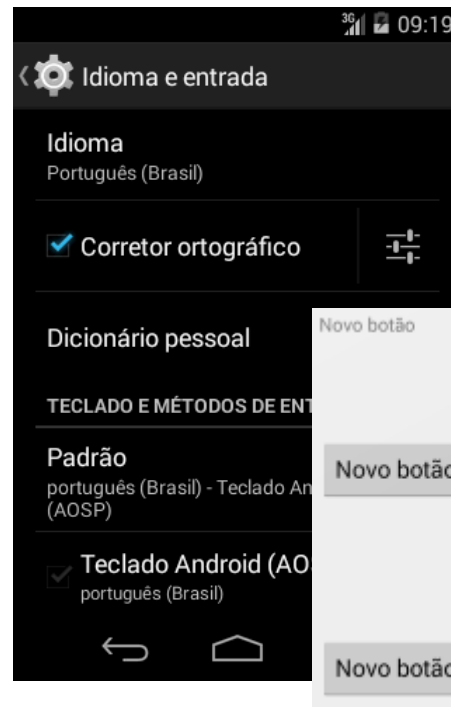
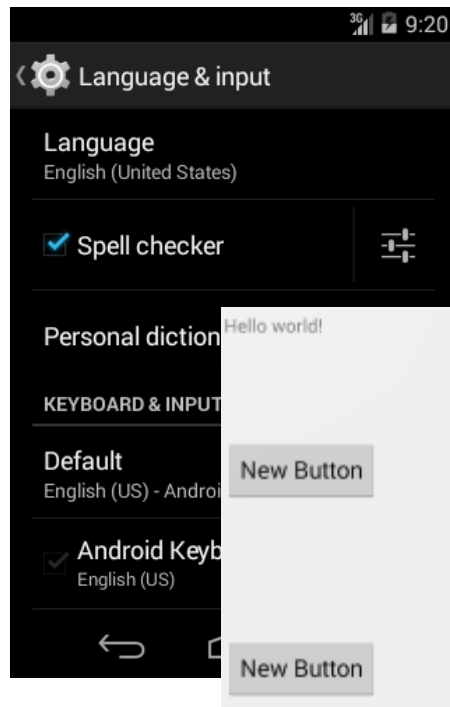
```
    if(country.equalsIgnoreCase("BR")) moeda = "R$";
```

```
    if(country.equalsIgnoreCase("PT")) moeda = "€";
```

```
}
```

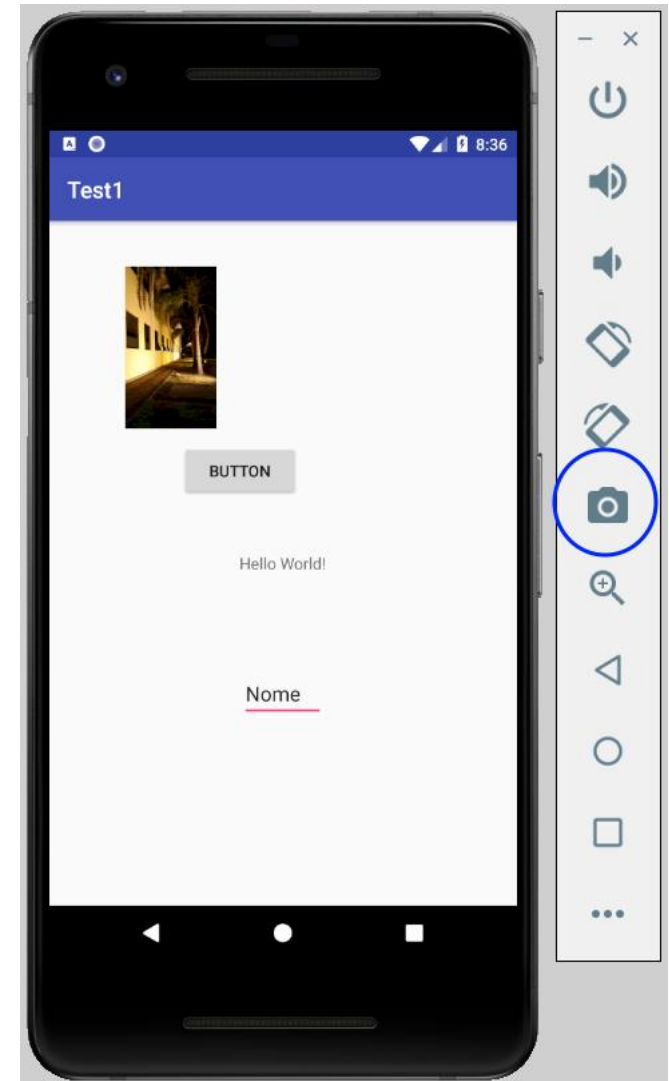
Prática: testar um app nos diferentes idiomas

- Para testar seu app em diversos idiomas você poderá visualizar na área de layout ou configurando o emulador em Configurações, Sistema, opção Idiomas (Settings, System, Languages):



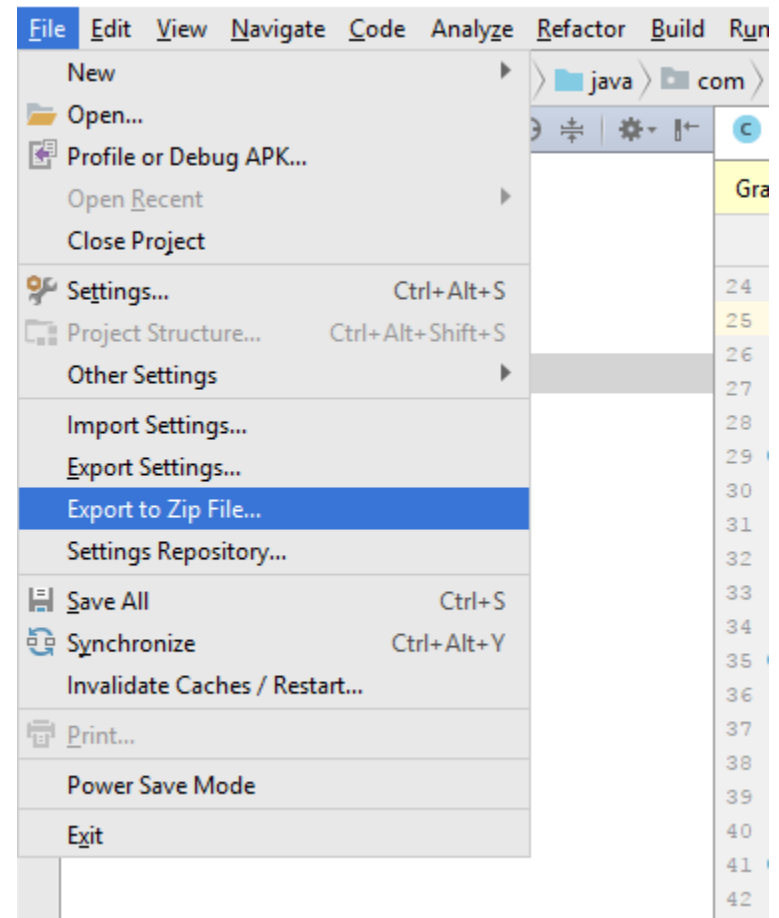
Screenshot (captura da tela) e rotar o emulador

- Após abrir a aplicação, podemos tirar print das telas através do painel dos emuladores do Android Studio.
- No emulador, selecione a tela da qual deseja tirar o print e clique no botão da câmera.
- Isso é interessante para publicar o app na loja Google Play, para entregar trabalhos, efetuar publicações em eventos etc.
- Também podemos testar o app nas posições portrait e landscape.



Exportar o projeto (ZIP)

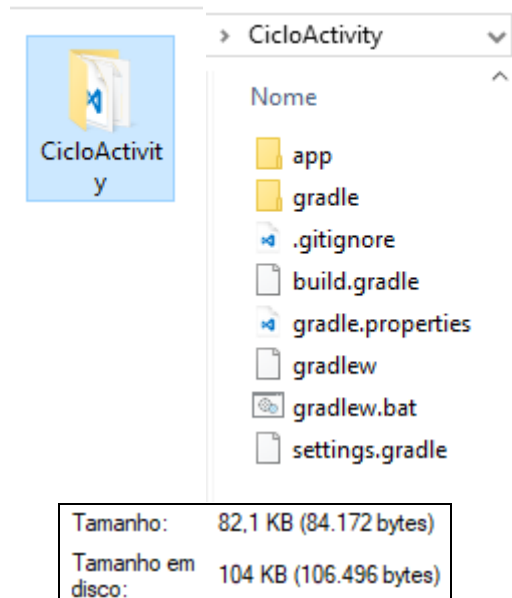
- Para exportar seu projeto Android para um arquivo ZIP, acesse o menu **File >> Export to Zip File...**
- Escolha o local para gravar o zip e clique em OK.
- **Muito importante:** este processo cria um ZIP somente com os arquivos do projeto, o que reduz consideravelmente o tamanho do mesmo! Entregue sempre seus trabalhos e provas utilizando este mecanismo, em lugar de copiar a pasta do projeto.



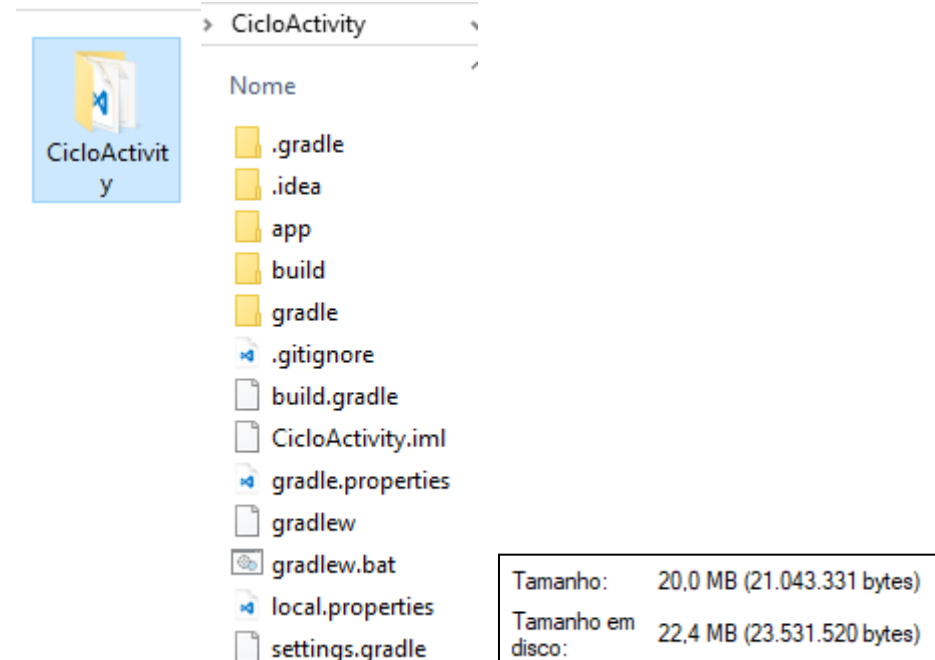
Importar o projeto (ZIP)

- Para importar o projeto do arquivo ZIP, primeiro descompacte o arquivo.
- No Android Studio escolha Open an existing Android Studio Project ou Import Project.
- A ferramenta abrirá o projeto e vai inserir na pasta os demais arquivos para seu projeto rodar normalmente.

Antes



Depois



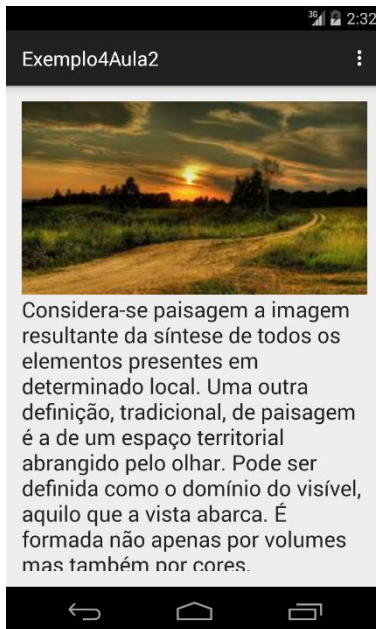
ImageView

- Primeiro crie um projeto Android.
- A classe ImageView permite colocar figuras dentro das telas de nossos aplicativos.
- Antes de usar esse objeto, vamos **copiar imagens para a pasta drawable**, basta selecionar os arquivos de imagens e arrastar para a respectiva pasta (ou copiar e colar).
- Em seguida insira um componente ImageView.
- Selecione uma das imagens para inserir no ImageView, você pode fazer isso através do código xml ou no painel de propriedade. Nos dois casos a propriedade é **src** ou **srcCompat**.
- Opcionalmente, você poderá configurar a propriedade `scaleType` para ajustar a imagem e ativar a propriedade `adjustViewBounds` para deixar o componente rente a imagem.
- Crie um arquivo de layout portrait e outro landscape.



TextView

- Componente tipo label (rótulo que o usuário não modifica).
- Insira um componente TextView abaixo da imagem no layout portrait e o mesmo no lado direito da imagem no layout landscape. Você deverá associar uma variável a esse componente, lembre-se de inserir essa variável no arquivo de strings. A mesma variável deverá ser utilizada em ambos os layouts.



↙ Variável

`android:text="@string/texto"`

TextView dentro de um ScrollView

- O ScrollView é um componente que permite rolar o conteúdo.
- No exemplo anterior o texto ficou cortado e não conseguimos rolar o mesmo, para resolver esse problema, vamos inserir um componente ScrollView com o componente TextView do texto dentro.
- Isso será feito no modo XML nos dois layouts.

portrait

```
<ScrollView
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_below="@+id/imageView">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/texto"
        android:textAppearance="?android:attr/textAppearanceLarge" />

</ScrollView>
```

```
<ScrollView
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_toEndOf="@+id/imageView"
    android:layout_alignParentTop="true">

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/texto"
        android:textAppearance="?android:attr/textAppearanceLarge" />

</ScrollView>
```

landscape

```
</ScrollView>
```

TextView com ScrollView dentro de um ConstraintLayout

```
<ScrollView
    android:id="@+id/scrolltext"
    android:layout_width="115dp"
    android:layout_height="158dp"
    android:layout_marginStart="24dp"
    android:layout_marginTop="112dp"
    app:layout_constraintHeight_default="wrap"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent">

    <TextView
        android:id="@+id/txtscroll"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/id_hello"
        app:layout_constraintTop_toTopOf="@+id/scrolltext"
        app:layout_constraintHeight_default="wrap" />

</ScrollView>
```

Quando utilizamos o layout ConstraintLayout (padrão do Android Studio atualmente) os atributos ficarão diferentes do slide anterior, porque devemos considerar as restrições (constraints) dos componentes. Por exemplo, o ScrollView ao lado foi definido com largura de 115dp e altura de 158dp, com atributos de constraint relacionados com o parent (que seria o layout ConstraintLayout principal) e ainda com determinadas margens de 24dp e 112dp.

O TextView mostrado terá mecanismo de rolagem por causa de estar dentro do ScrollView. Observe que, também, ele terá algumas restrições:

```
app:layout_constraintTop_toTopOf
app:layout_constraintHeight_default
```

Rolagem para o conteúdo completo da tela

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

```
<ScrollView
```

```
    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">
```

```
<android.support.constraint.ConstraintLayout  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    tools:context=".MainActivity">
```

```
    ... os componentes da tela ficam aqui ...
```

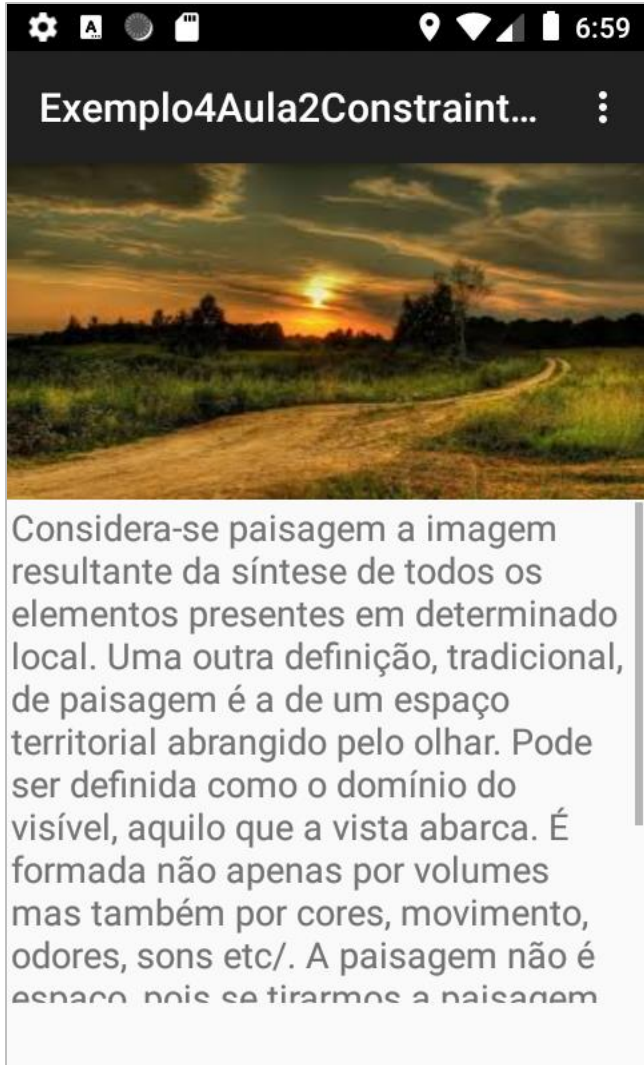
```
</android.support.constraint.ConstraintLayout>
```

```
</ScrollView>
```

Como foi mencionado, para a criação das telas (Activity) nas versões mais recentes do Android Studio é utilizado como padrão um layout que facilita bastante o projeto visual, chamado de ConstraintLayout.

Se queremos que a tela completa tenha mecanismo de rolagem, podemos colocar esse ConstraintLayout que foi gerado dentro de uma ScrollView, como mostrado no exemplo ao lado.

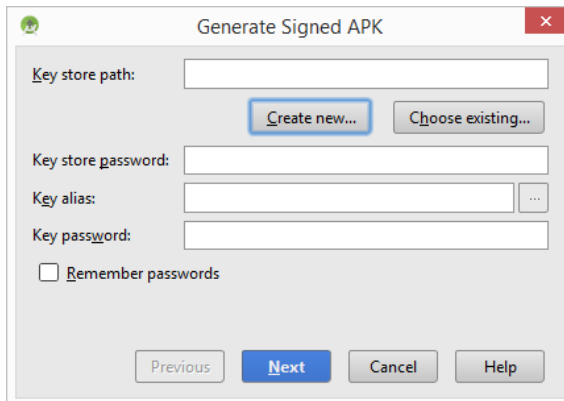
ScrollView - executando o exemplo



Observe as barras de rolagem do objeto ScrollView

Exportando seu projeto para apk

- Agora vamos gerar um arquivo de instalação da nossa aplicação. Esse arquivo possui a extensão .apk. Após gerar o arquivo você poderá enviar para seu dispositivo para instalar e testar. Este .apk pode ser enviado para a Google Play.
- Clique em Build >> Generate Signed APK
- Será necessário criar uma Keystore (chave criptografada do proprietário da app). Guarde com sigilo. Será obrigatória para futuras atualizações do app.



Generate Signed APK

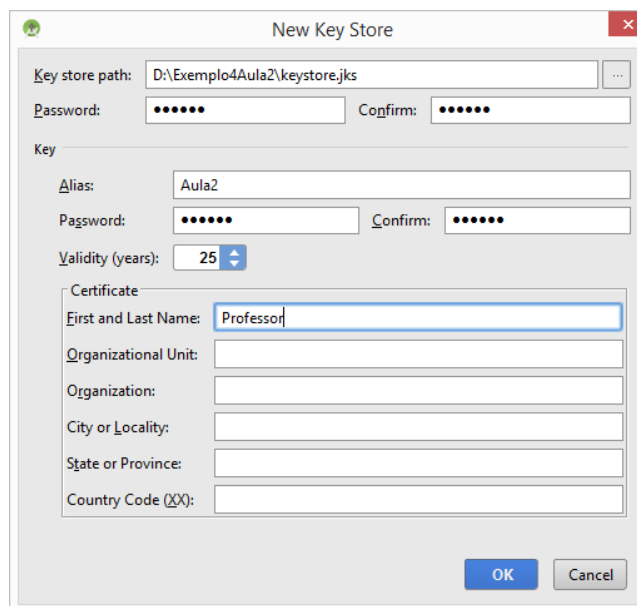
Key store path:

Key store password:

Key alias:

Key password:

☐ Remember passwords



New Key Store

Key store path:

Password: Confirm:

Key

Alias:

Password: Confirm:

Validity (years):

Certificate

First and Last Name:

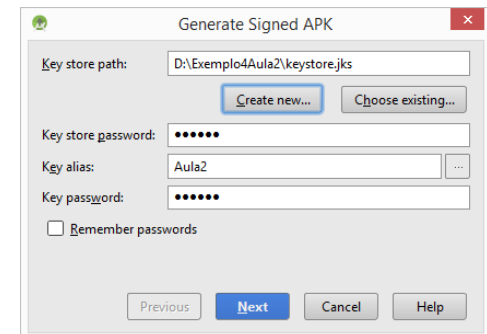
Organizational Unit:

Organization:

City or Locality:

State or Province:

Country Code (XX):



Generate Signed APK

Key store path:

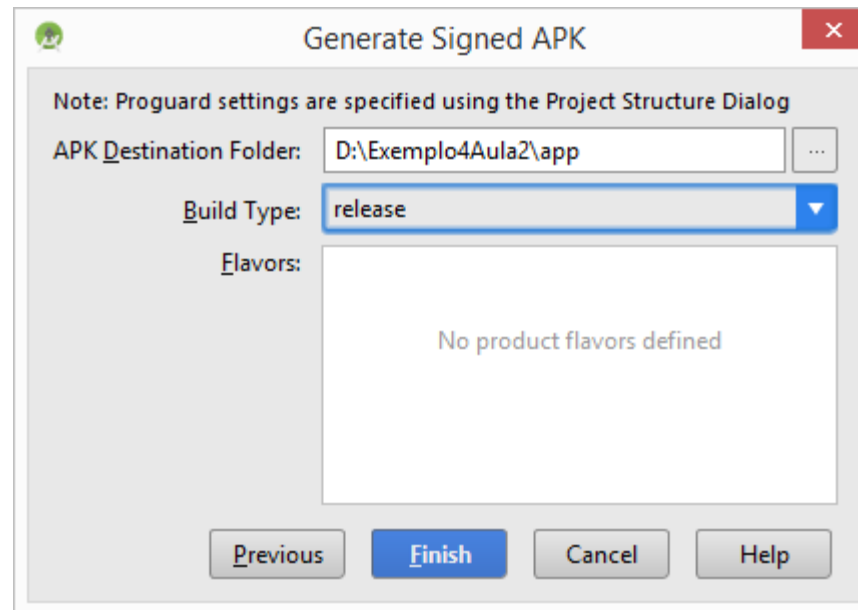
Key store password:

Key alias:

Key password:

☐ Remember passwords

Exportando seu projeto para apk



- **CUIDADO!** A Keystore será utilizado por todas as versões da sua aplicação. Se publicar um app com uma e depois perder e gerar outra, não poderá associar as aplicações, portanto, após gerar o keystore **faça backup** do arquivo e lembre sua senha, para poder efetuar futuras publicações da aplicação.

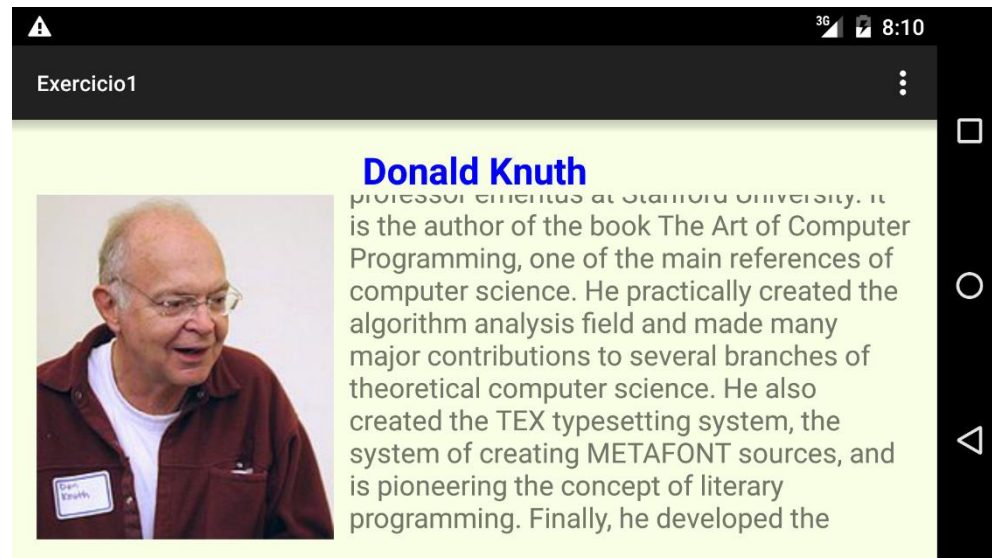
<http://developer.android.com/tools/publishing/app-signing.html>

Exercício

- Procure duas pessoas famosas (cientistas da computação) para fazer sua aplicação. Para cada pessoa você deverá incluir uma imagem e um texto sobre a mesma. O texto deve estar disponível em pelo menos dois idiomas. Algumas sugestões de pessoas para pesquisar:
 - Ada Lovelace
 - Alan Turing
 - Grace Hopper
 - Charles Babbage
 - John von Neumann
 - Donald Knuth
- Você deverá implementar dois layouts (portrait e landscape) para essa tarefa, veja modelo no próximo slide. Para cada layout utilize uma pessoa diferente. Utilize também scrollview no texto.

Exercício

- Continuação do exercício



Modifique:

- 1 – a cor de fundo (uma para cada layout)
- 2 – a cor do nome do cientista
- 3 – deve existir uma versão dos textos em PT-BR e outra em EN.

Bibliografia sugerida sobre Android

- ANDROID. Android Developers. Disponível em <http://developer.android.com>.
- LECHETA, RICARDO R. Google Android, Aprenda a criar aplicações para dispositivos móveis com o Android SDK. São Paulo: Novatec, 2010.
- MEDNIEKS, Z. et. al. Desenvolvimento de Aplicações Android. São Paulo: Novatec, 2009.
- LEE, Wei-Meng. Introdução ao Desenvolvimento de Aplicativos para o Android. Rio de Janeiro: Editora Ciência Moderna, 2011