# Desenvolvimento Mobile

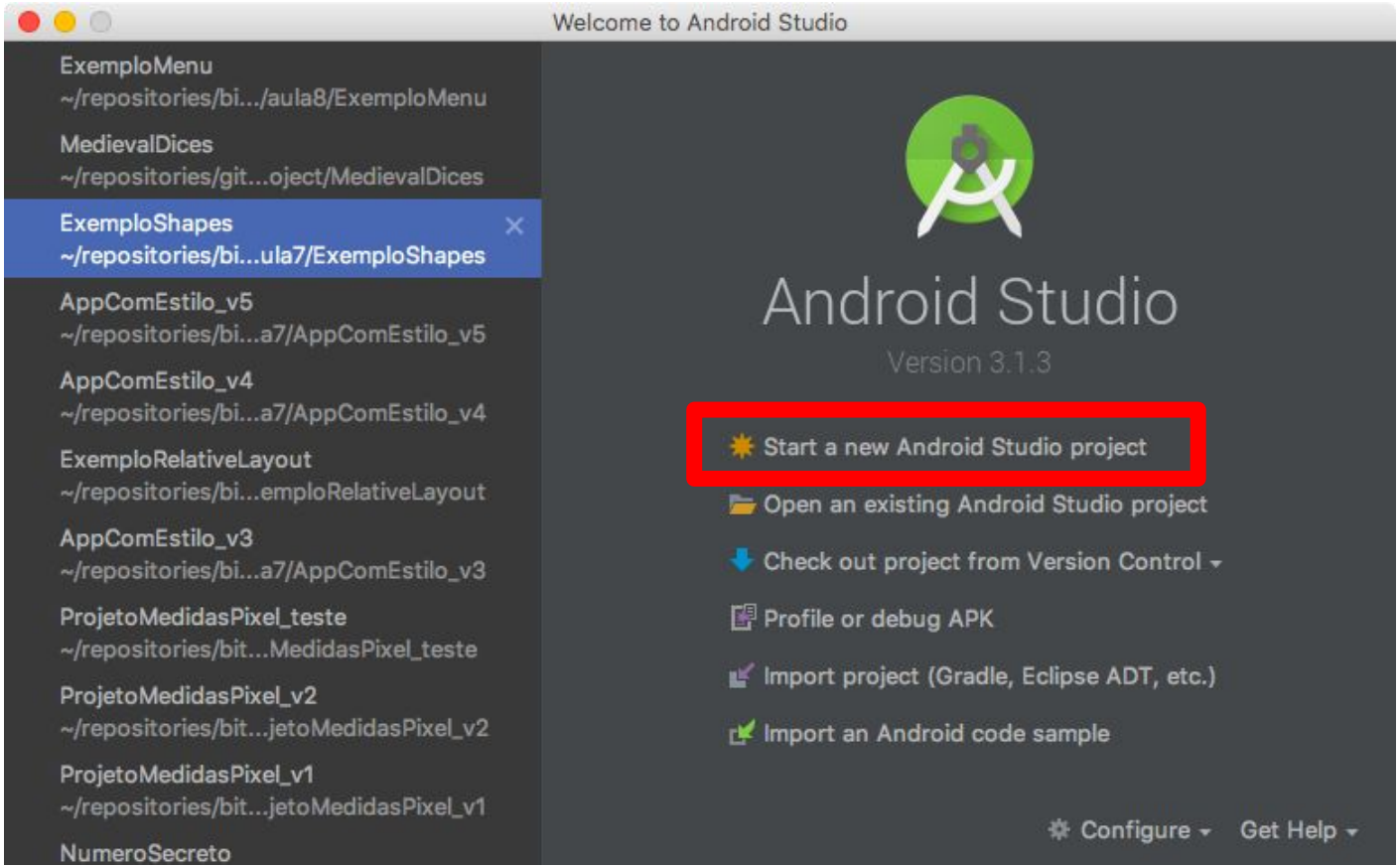
Aula 8

# Toast vs SnackBar

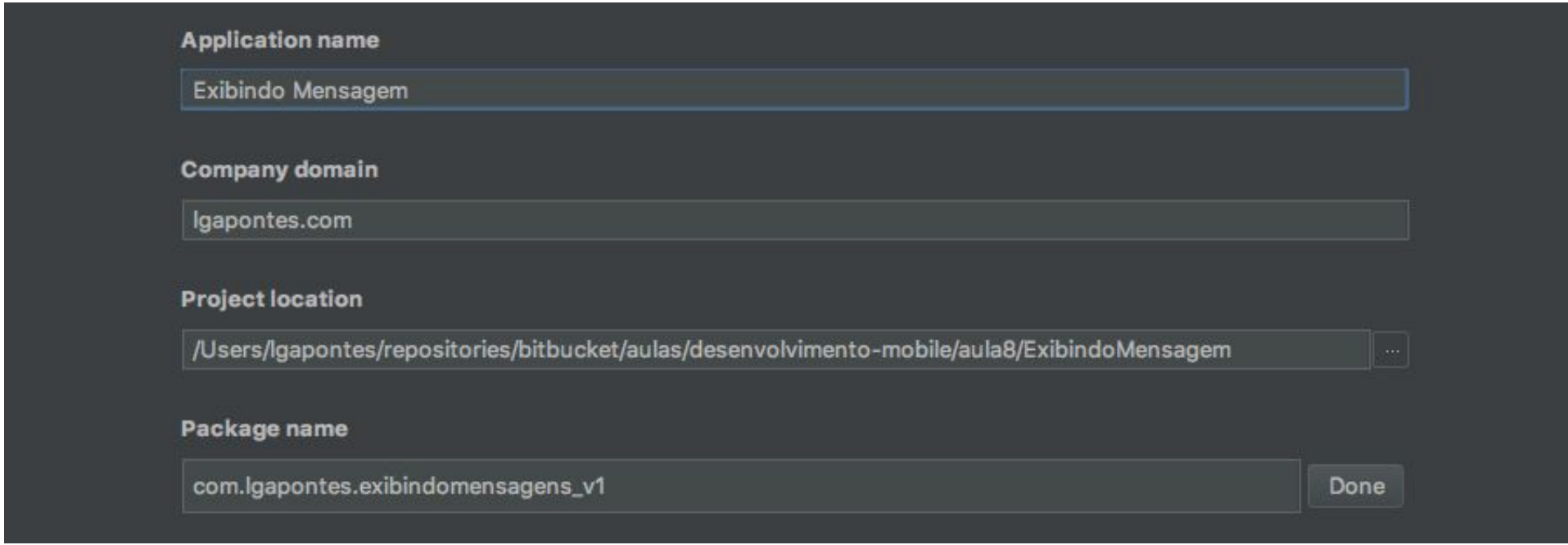
- Não pode fechar por *swiping*
- Pode ser exibido sem nenhuma Activity
- Não aceita ações do usuário


- Pode fechar por *swiping*
- Não pode ser exibido fora da Activity
- Aceita ações do usuário

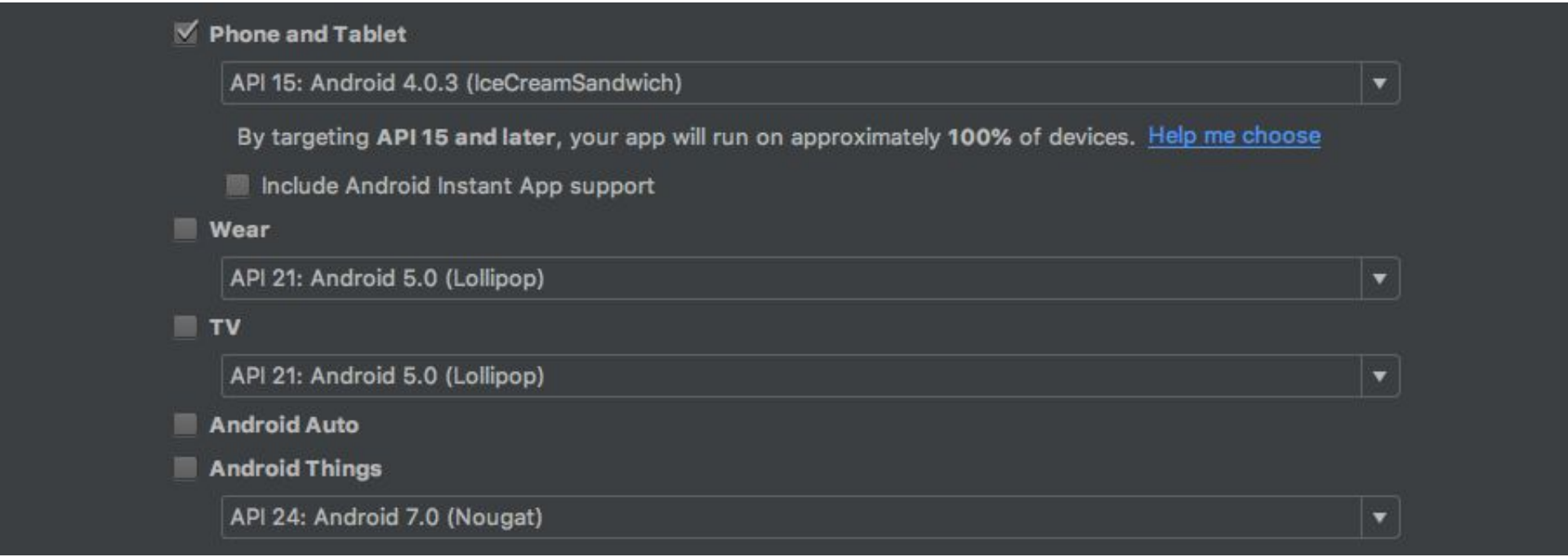
# Trabalhando com Toast (*ExibindoMensagem*)



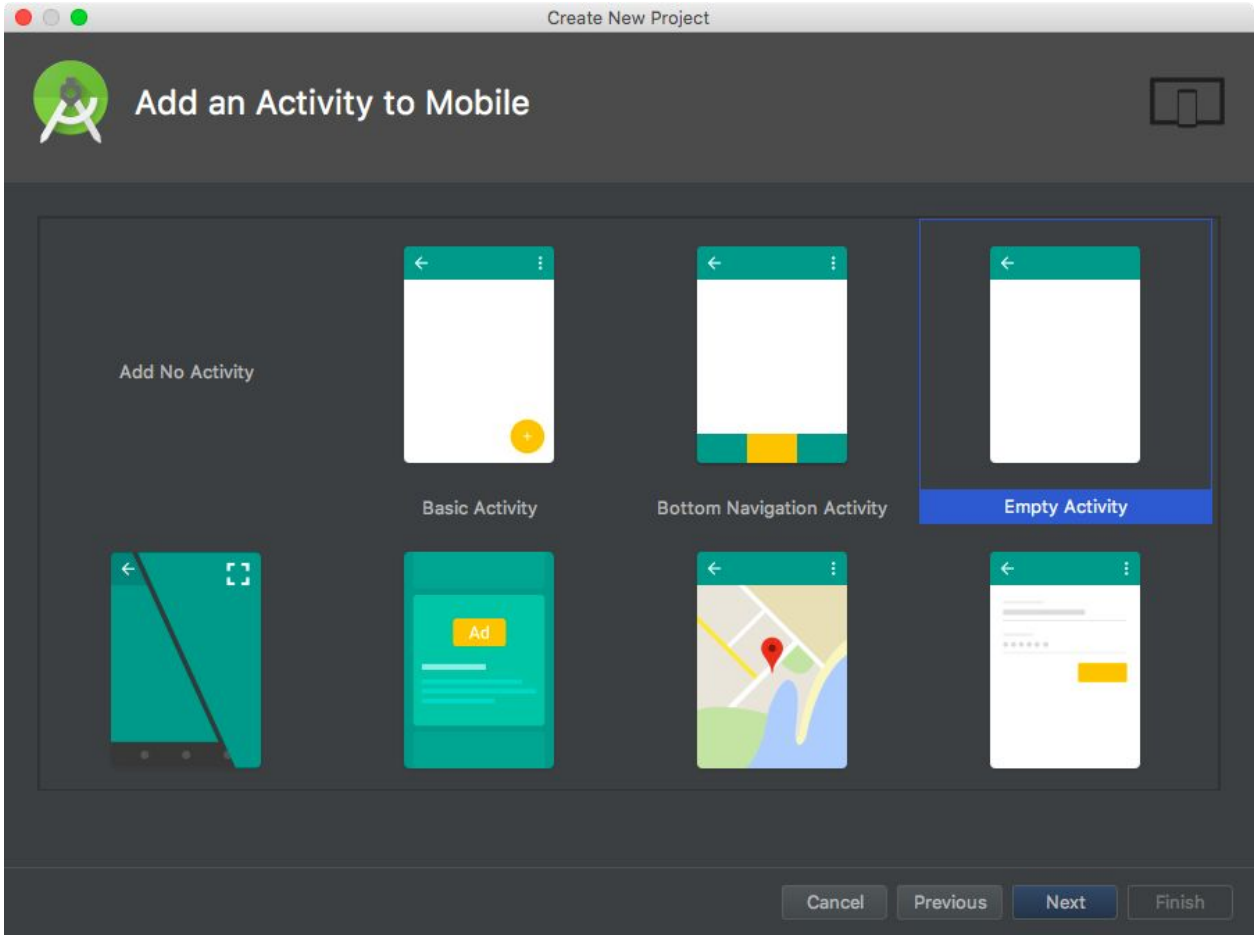
# Trabalhando com Toast (*ExibindoMensagem*)



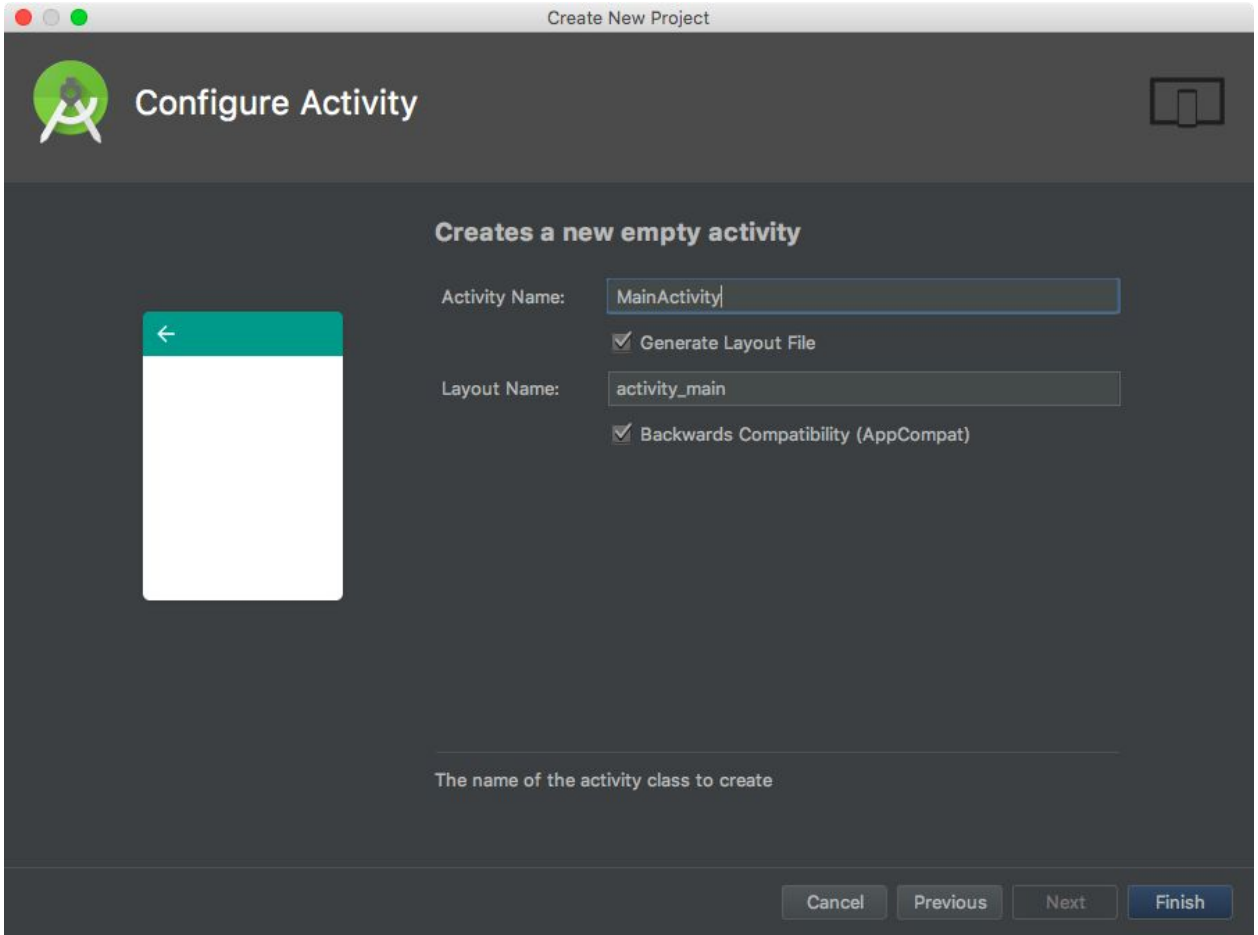
# Trabalhando com Toast (*ExibindoMensagem*)



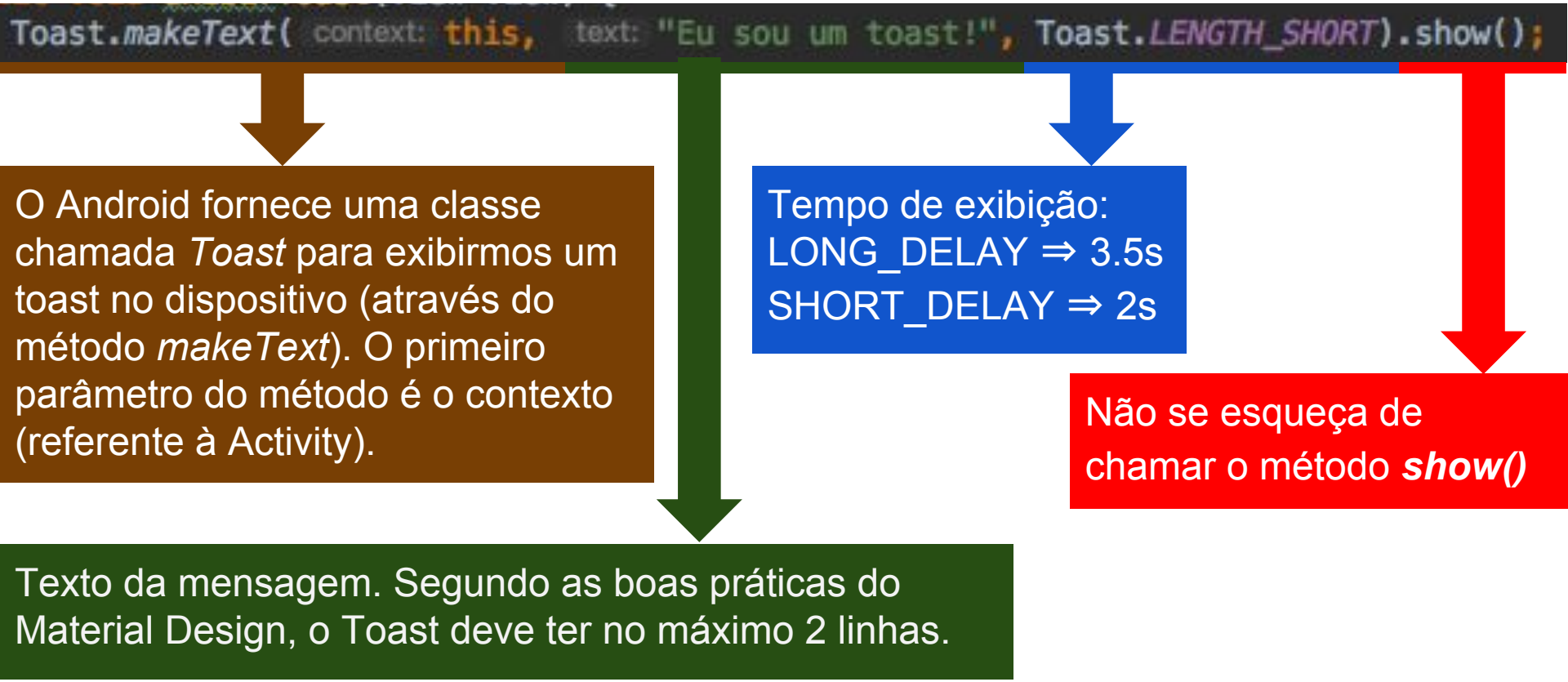
# Trabalhando com Toast (*ExibindoMensagem*)



# Trabalhando com Toast (*ExibindoMensagem*)



# Trabalhando com Toast (*ExibindoMensagem*)



# Trabalhando com Toast (*ExibindoMensagem*)

activity_main.xml | MainActivity.java

```java
package com.lgapontes.exibindomensagens_v1;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

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

    public void exibirToast(View view) {
        Toast.makeText( context: this,  text: "Eu sou um toast!", Toast.LENGTH_SHORT).show();
    }
}
```

# Trabalhando com Toast (*ExibindoMensagem*)

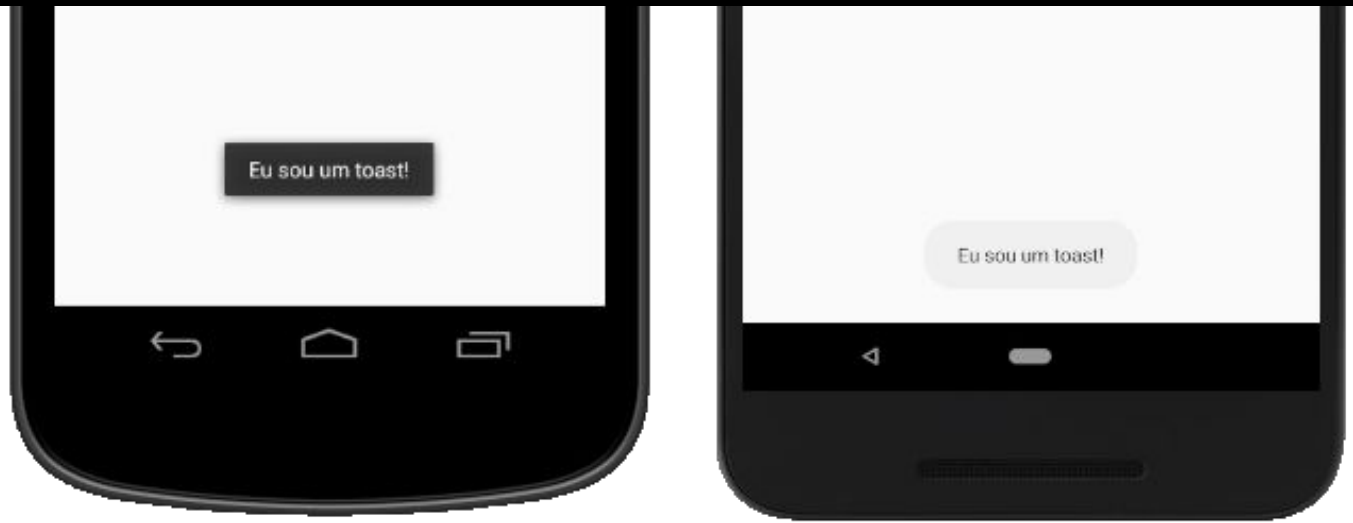
activity_main.xml | MainActivity.java

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3       android:layout_width="match_parent"
4       android:layout_height="match_parent"
5       android:orientation="vertical">
6
7       <Button
8           android:layout_width="wrap_content"
9           android:layout_height="wrap_content"
10          android:layout_gravity="center_horizontal"
11          android:text="Exibir Toast"
12          android:onClick="exibirToast" />
13
14  </LinearLayout>
```

# Trabalhando com Toast (*ExibindoMensagem*)

O Toast pode ter um visual particular nas diferentes versões do Android SDK. Apesar da recomendação do Material Design (que também é da Google) indicar cantos arredondados, em SDK's mais antigos às vezes eles não tem esse detalhe.

**É possível, entretanto, criar um Toast customizado!**

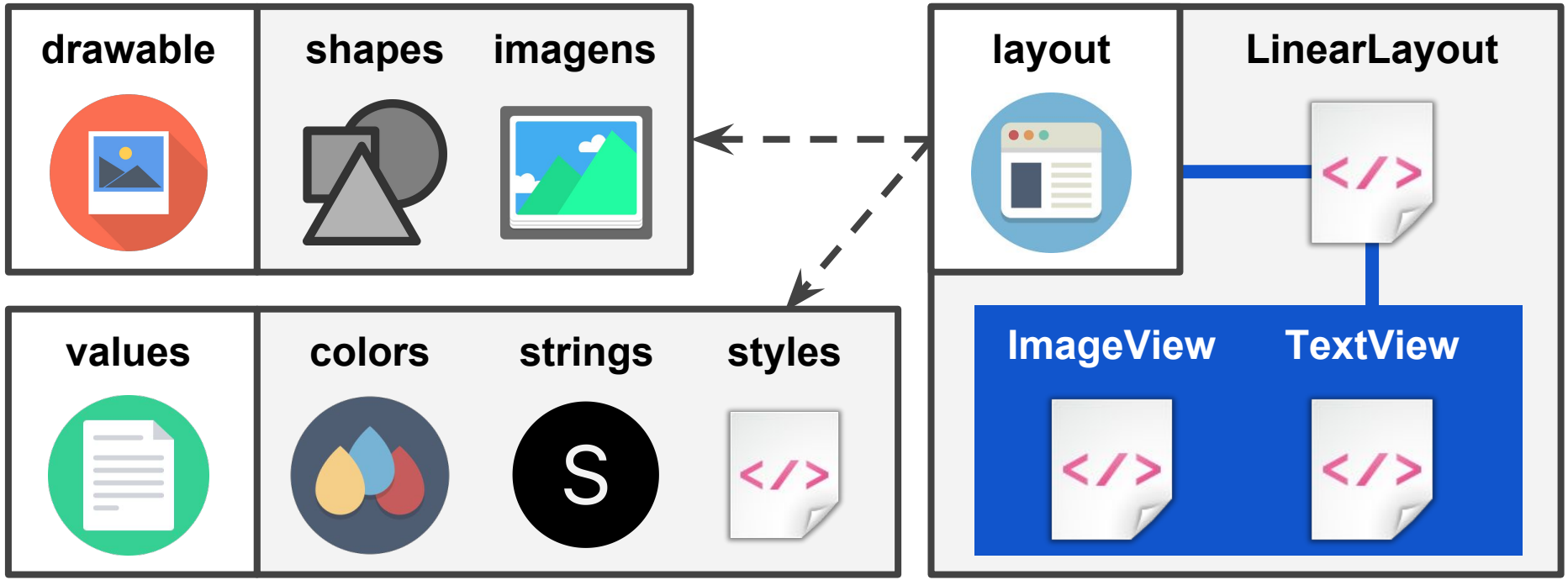


# Trabalhando com Toast Customizado (*ExibindoMensagem*)

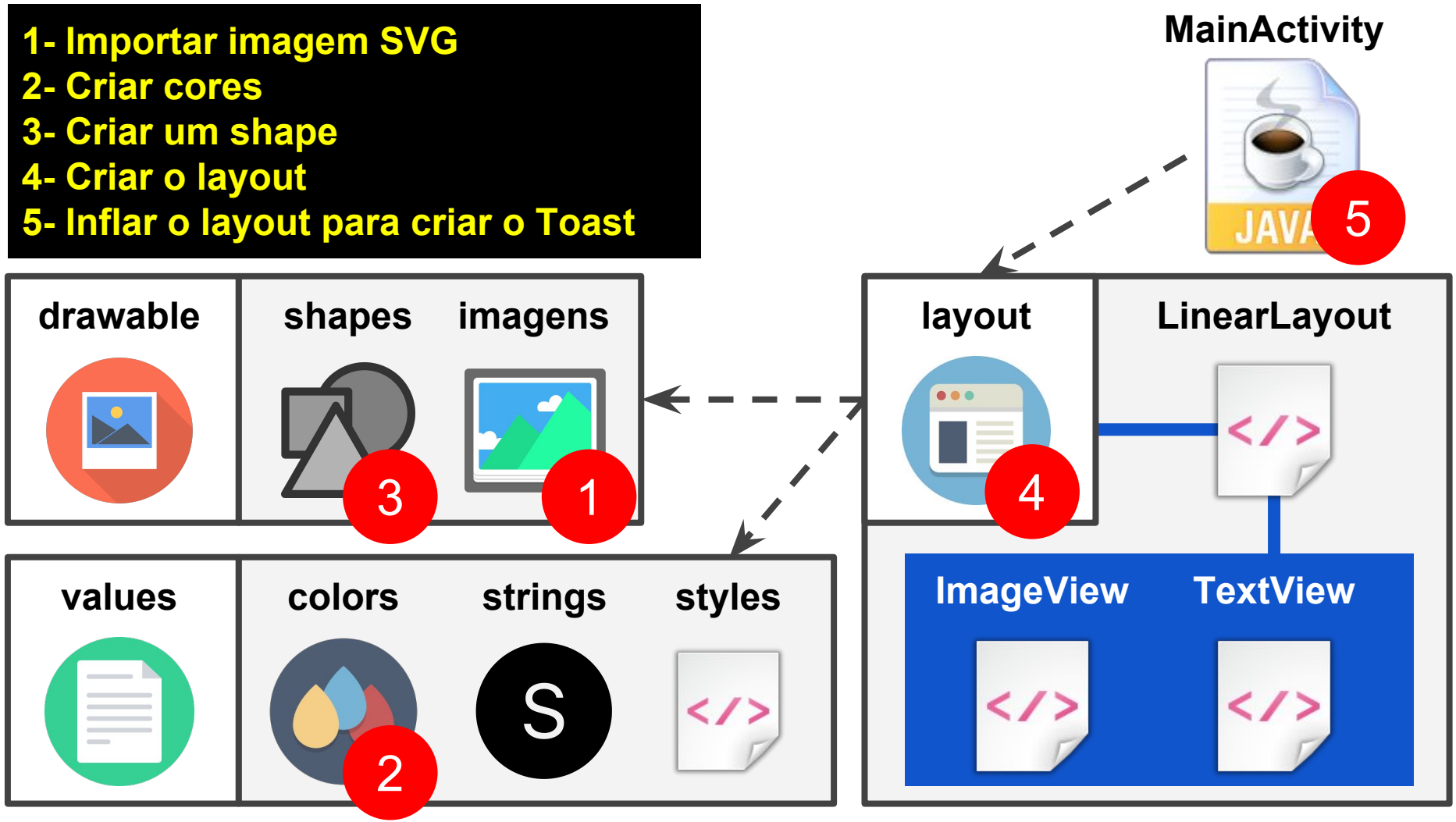
**Vamos criar o seguinte Toast...**

# Trabalhando com Toast Customizado (*ExibindoMensagem*)

**Um Toast customizado nada mais é do que um objeto *Toast* inflando um Layout criado na pasta *layout*.**



O termo **inflar** é utilizado quando criamos um **Layout** a partir de um arquivo XML


1- Importar imagem SVG
2- Criar cores
3- Criar um shape
4- Criar o layout
5- Inflar o layout para criar o Toast
MainActivity
JAVA
5
drawable
shapes
imagens
3
1
layout
LinearLayout
4
ImageView
TextView
values
colors
strings
styles
S
2

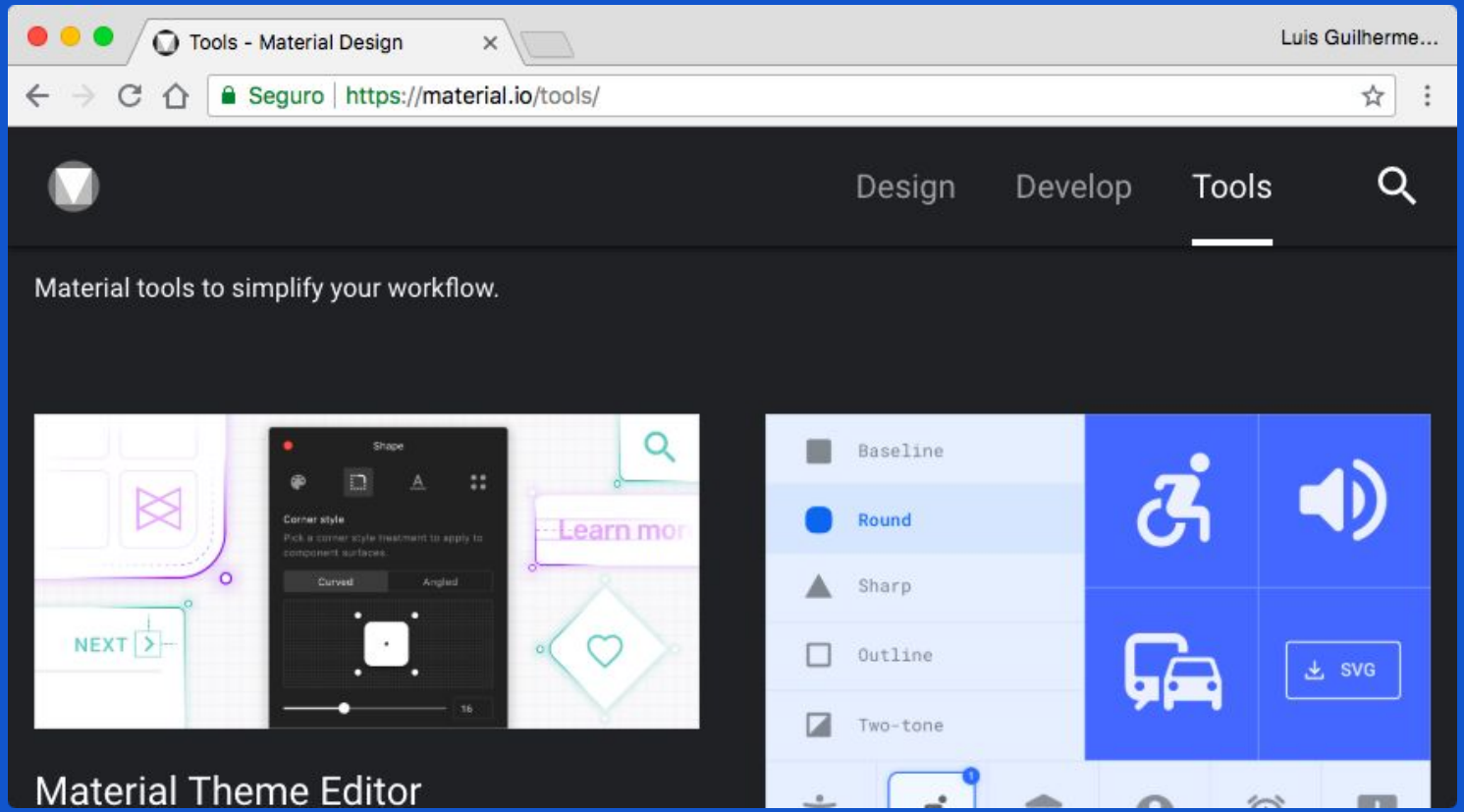
# Trabalhando com Toast Customizado (*ExibindoMensagem*)

**1** **Primeiramente veremos como acrescentar ícones escaláveis**


Também podemos utilizar imagens comuns (PNG, JPG, etc). Veremos, entretanto, que imagens vetoriais são muito poderosas!

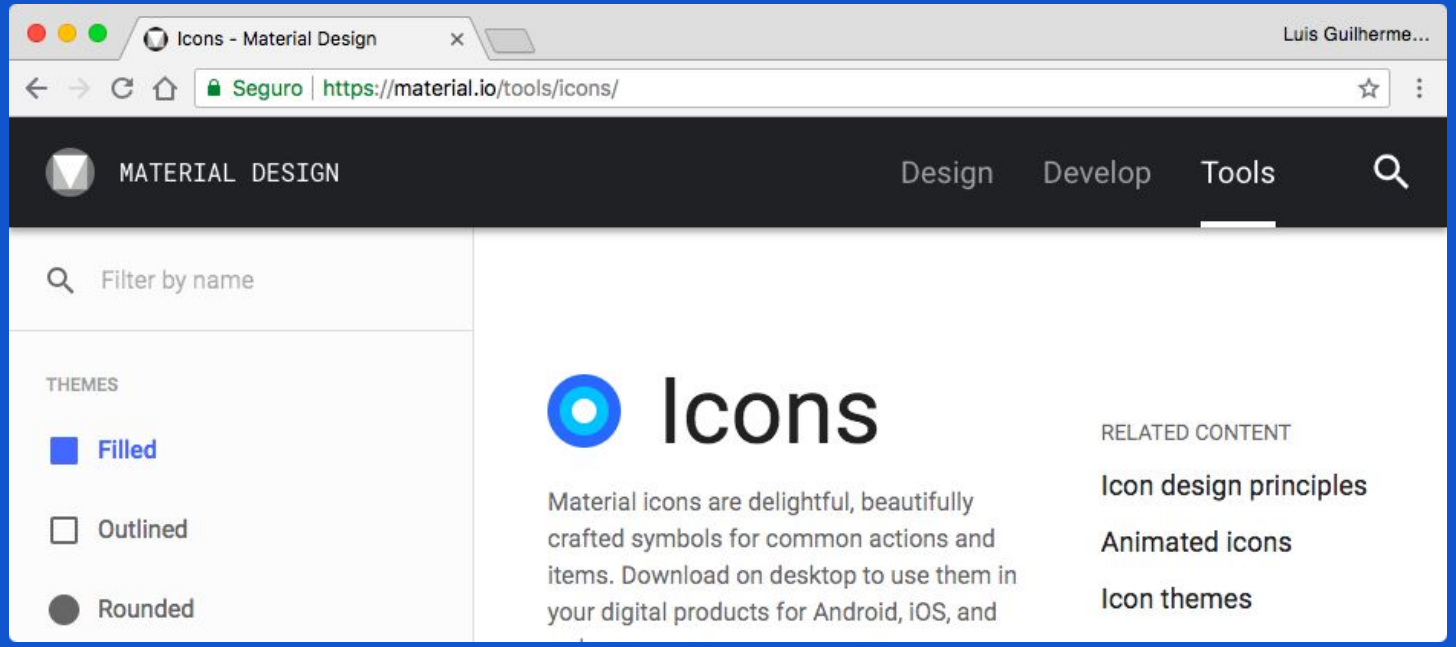
# Material Design Tools

https://material.io/tools/



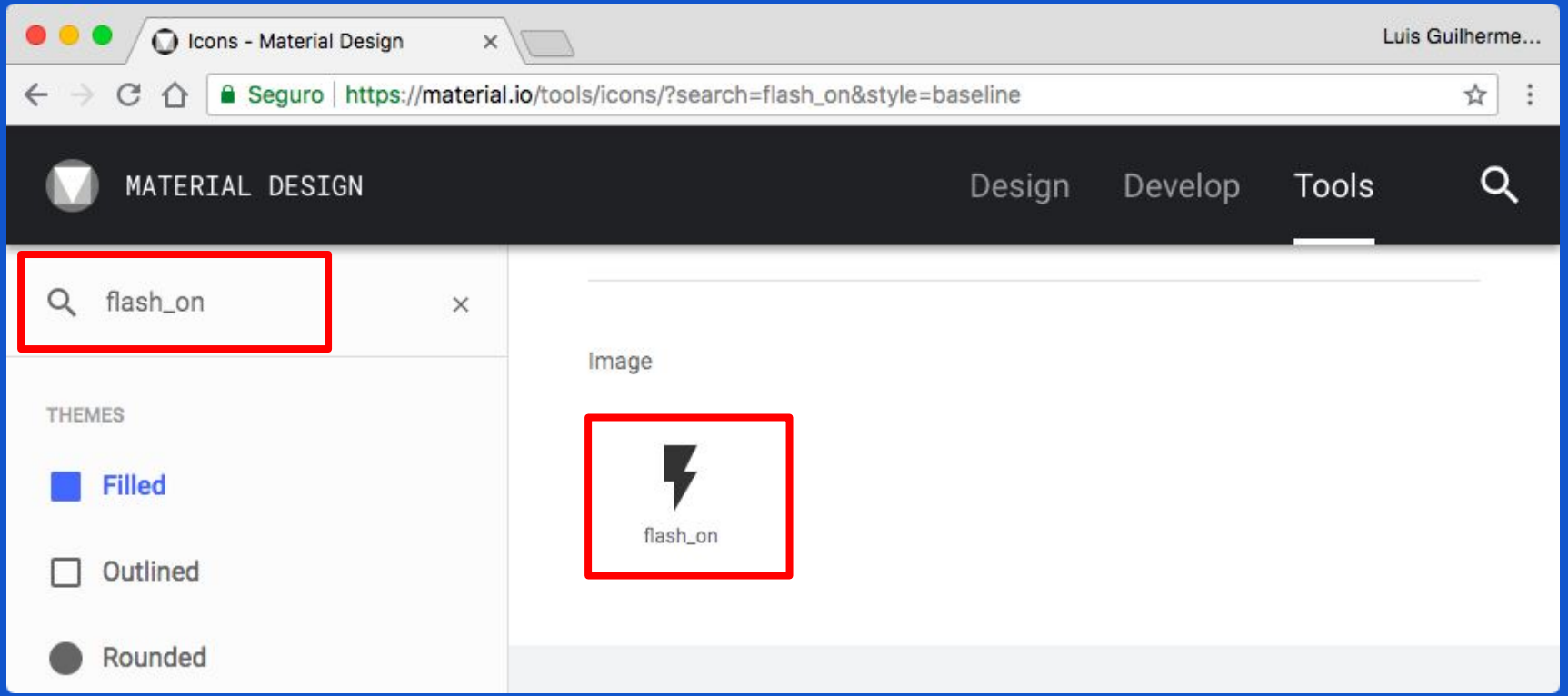
# Trabalhando com Scalable Vector Graphics (SVG)

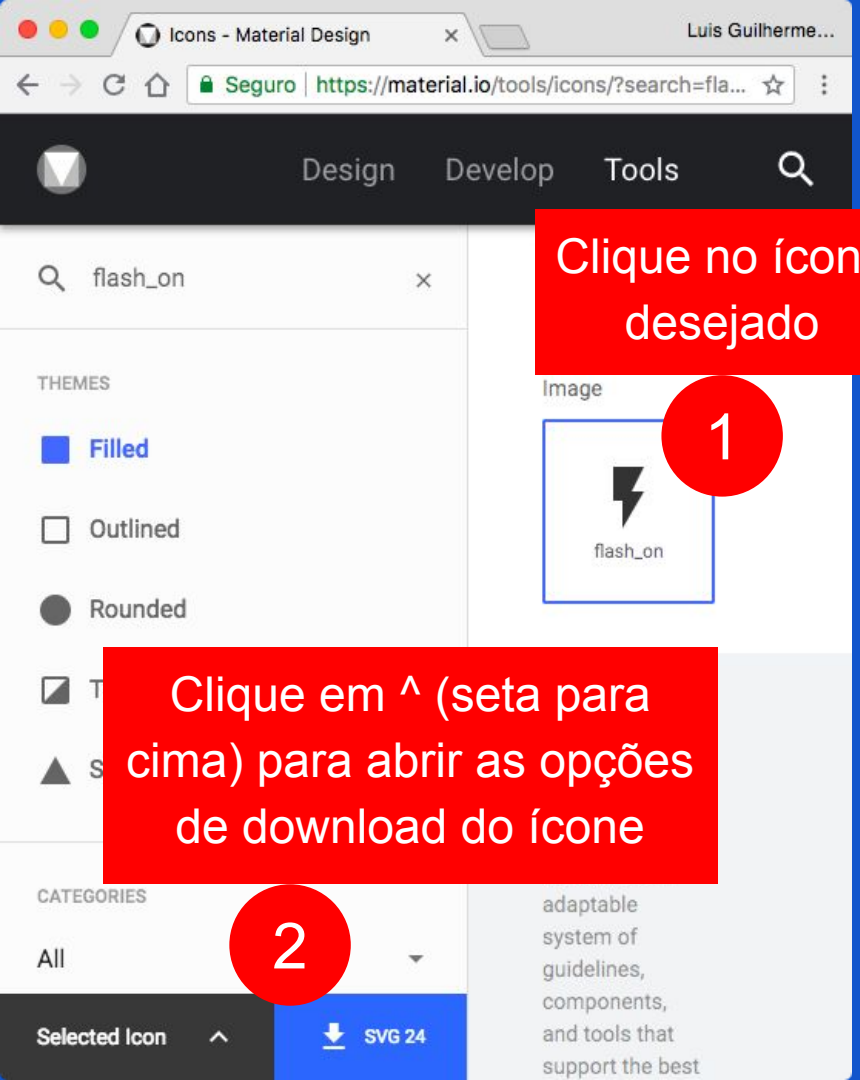
Imagens SVG são automaticamente ajustadas pelo Android de acordo com a densidade de tela do aparelho. O próprio Google oferece uma infinidade de ícones para utilização em sua App: https://material.io/tools/icons/

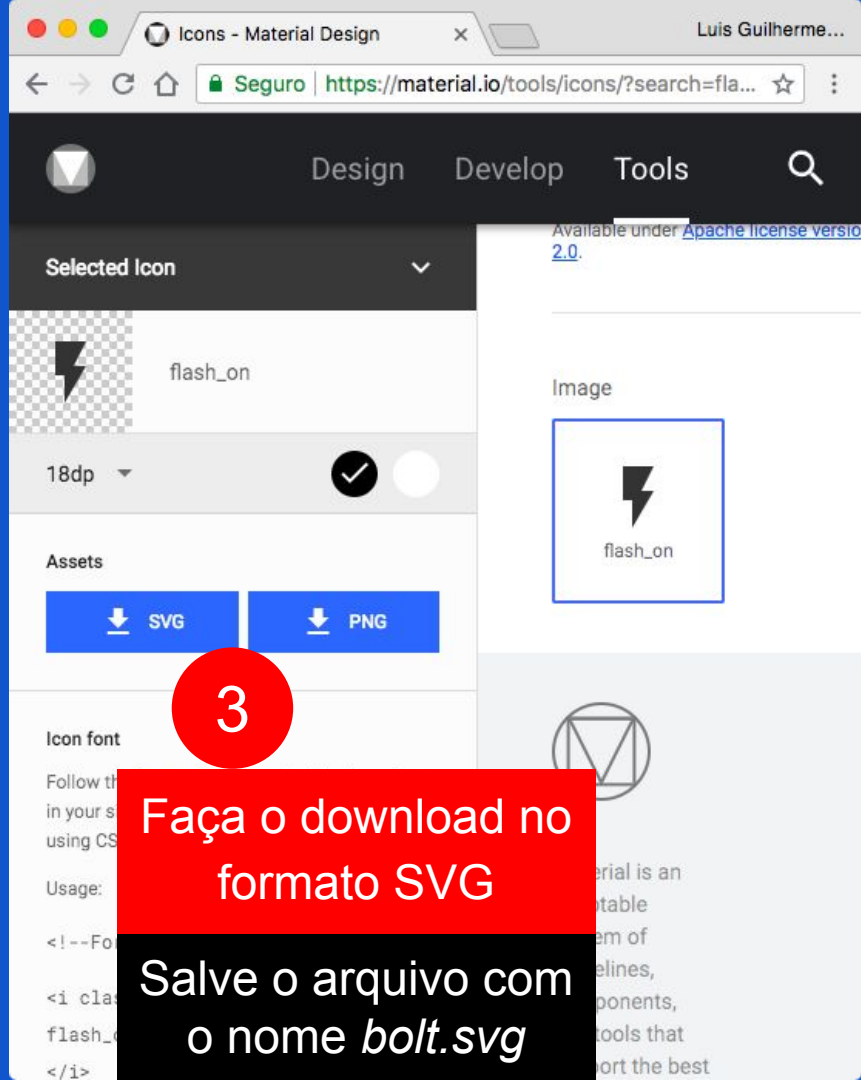


# Trabalhando com Scalable Vector Graphics (SVG)

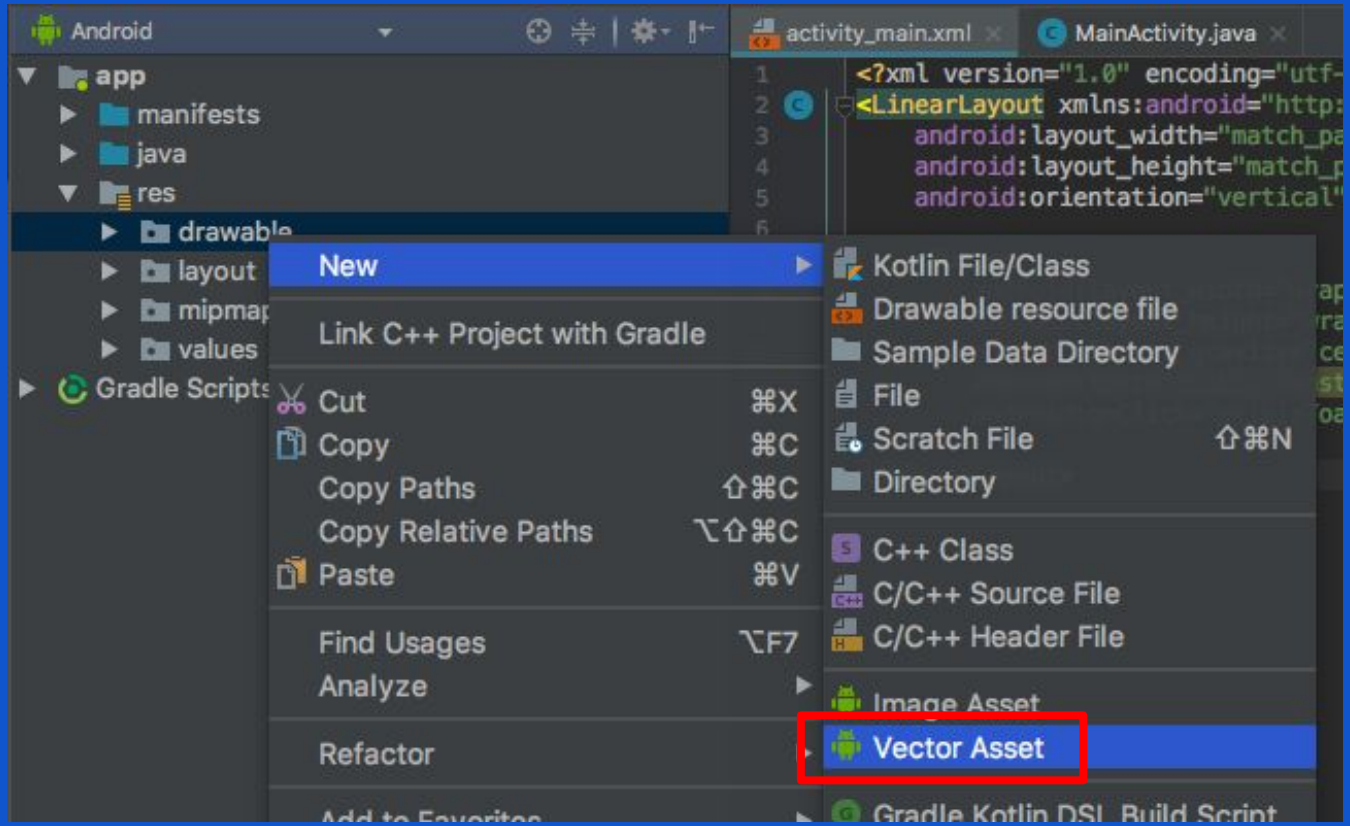
Procure pelo ícone *flash_on* no campo de busca.




Icons - Material Design
Luis Guilherme...
Seguro | https://material.io/tools/icons/?search=fla...
Design
Develop
Tools
flash_on
THEMES
Filled
Outlined
Rounded
CATEGORIES
All
Selected Icon
SVG 24
Clique no ícone desejado
1
Image
flash_on
Clique em ^ (seta para cima) para abrir as opções de download do ícone
2
adaptable system of guidelines, components, and tools that support the best

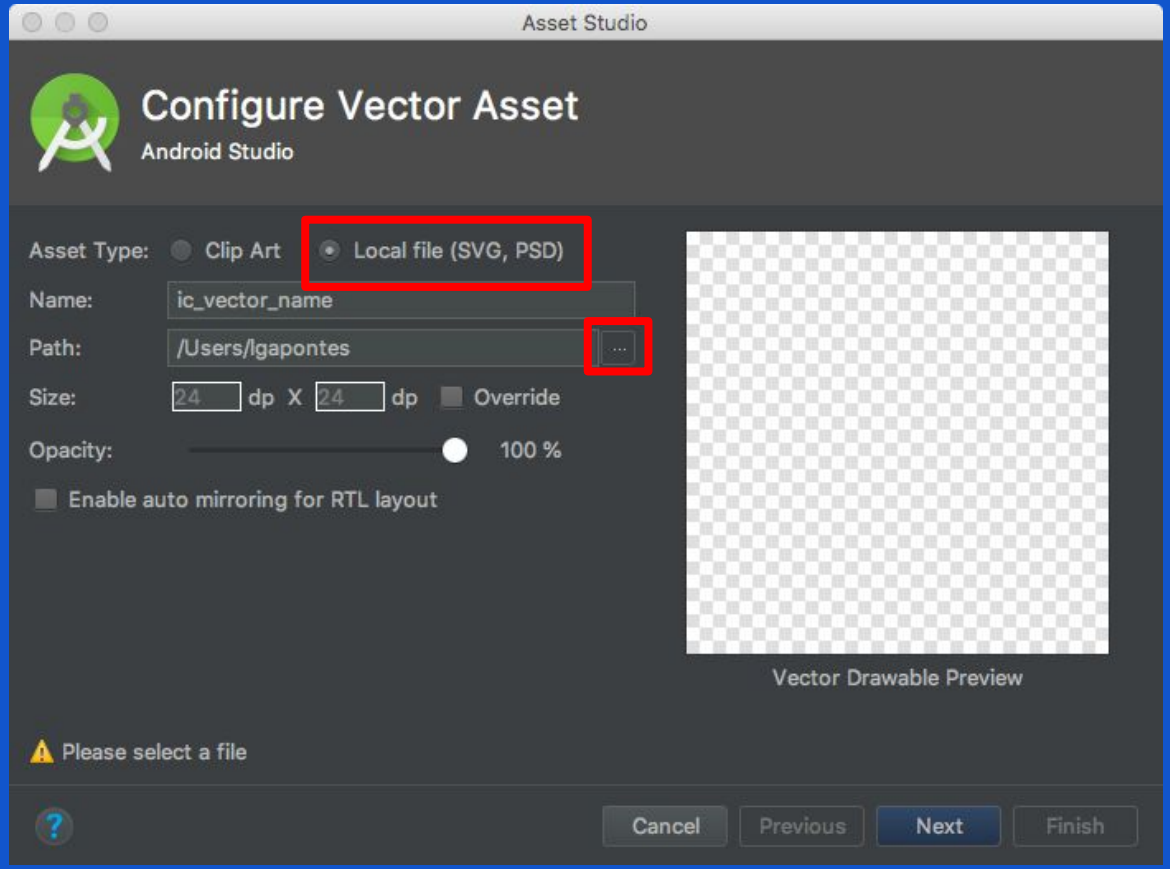

Icons - Material Design
Luis Guilherme...
Seguro | https://material.io/tools/icons/?search=fla...
Design
Develop
Tools
Selected Icon
flash_on
18dp
Assets
SVG
PNG
3
Faça o download no formato SVG
Icon font
Usage:
Salve o arquivo com o nome bolt.svg
Image
flash_on

# Trabalhando com Scalable Vector Graphics (SVG)



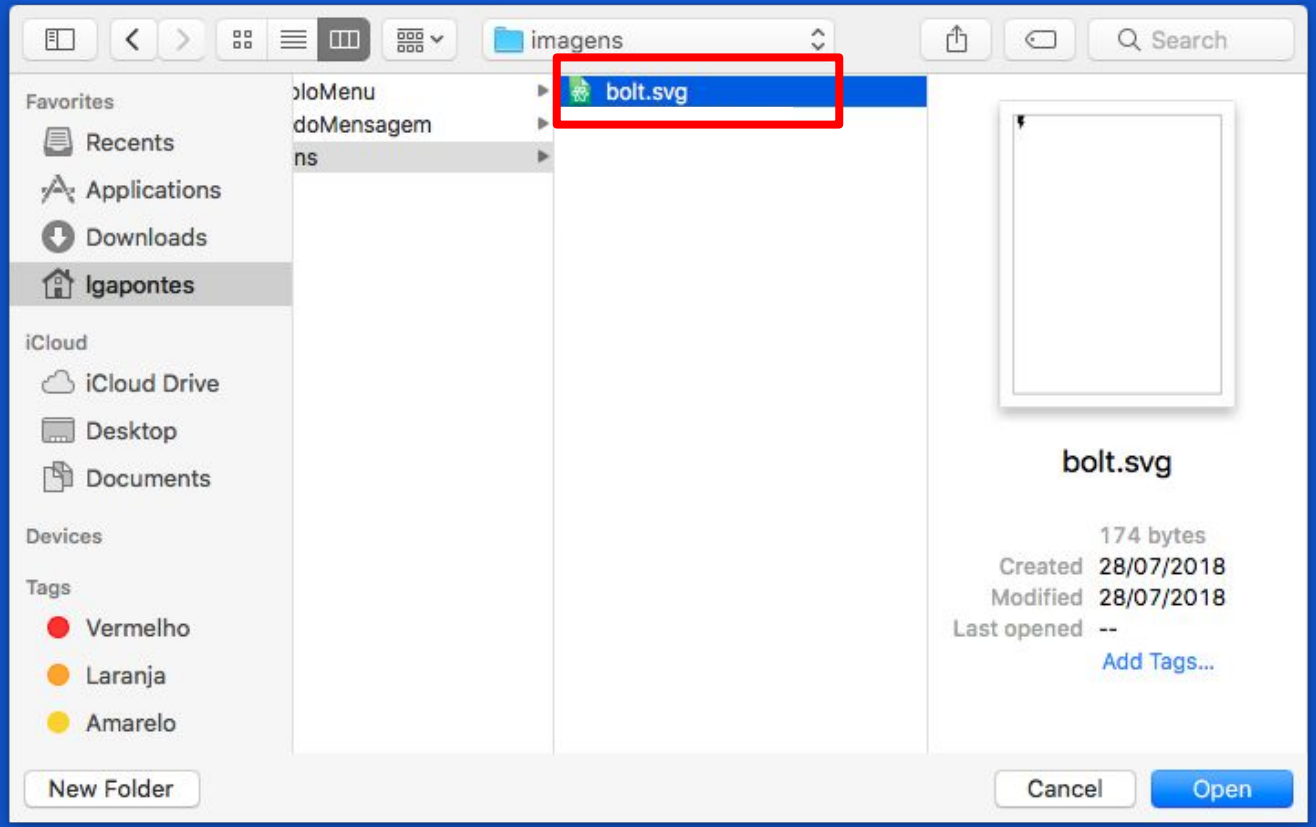

Clique com o botão direito na pasta *drawable*, selecione o menu *new*, submenu *Vector Asset*

# Trabalhando com Scalable Vector Graphics (SVG)



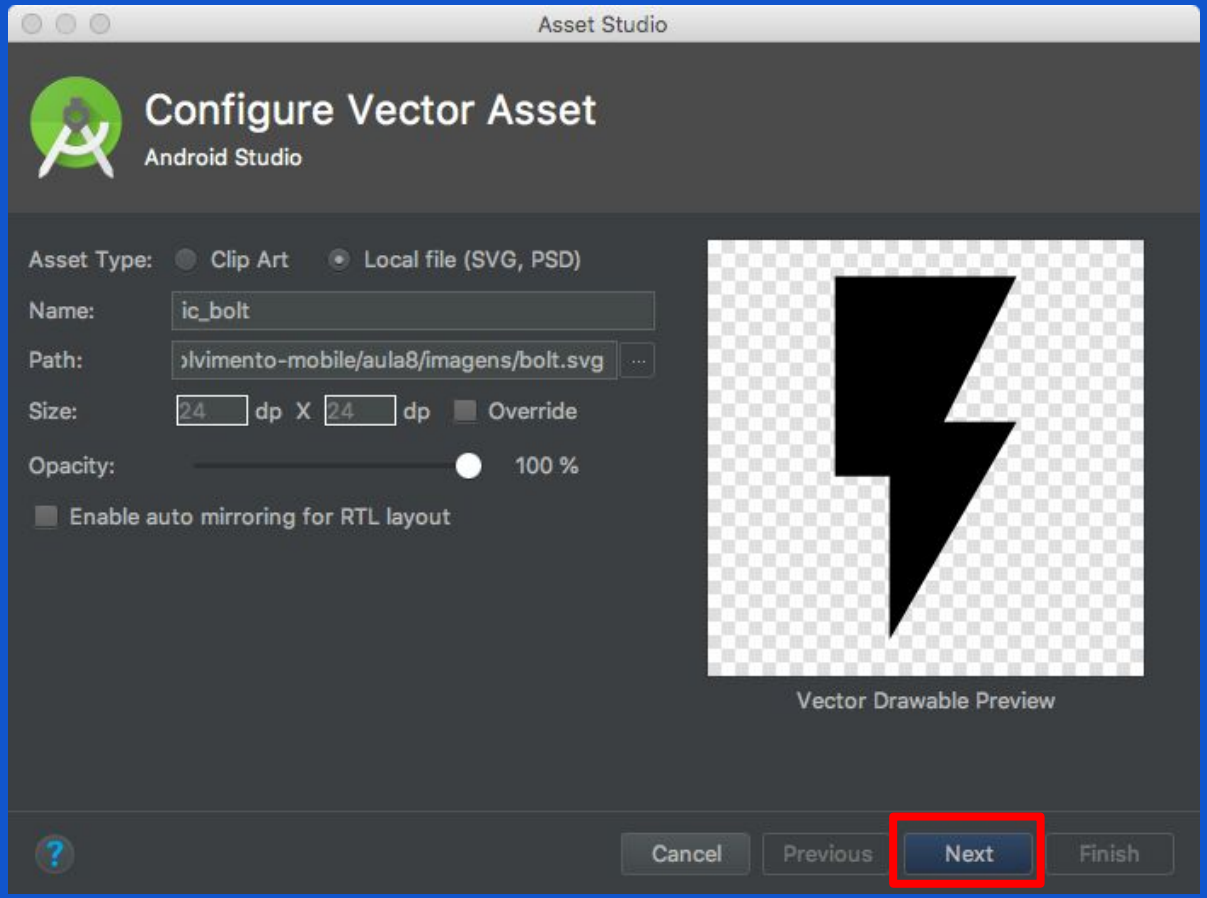

Clique na radiobutton *Local file* e em seguida, clique no botão … para selecionar o arquivo baixado.

# Trabalhando com Scalable Vector Graphics (SVG)

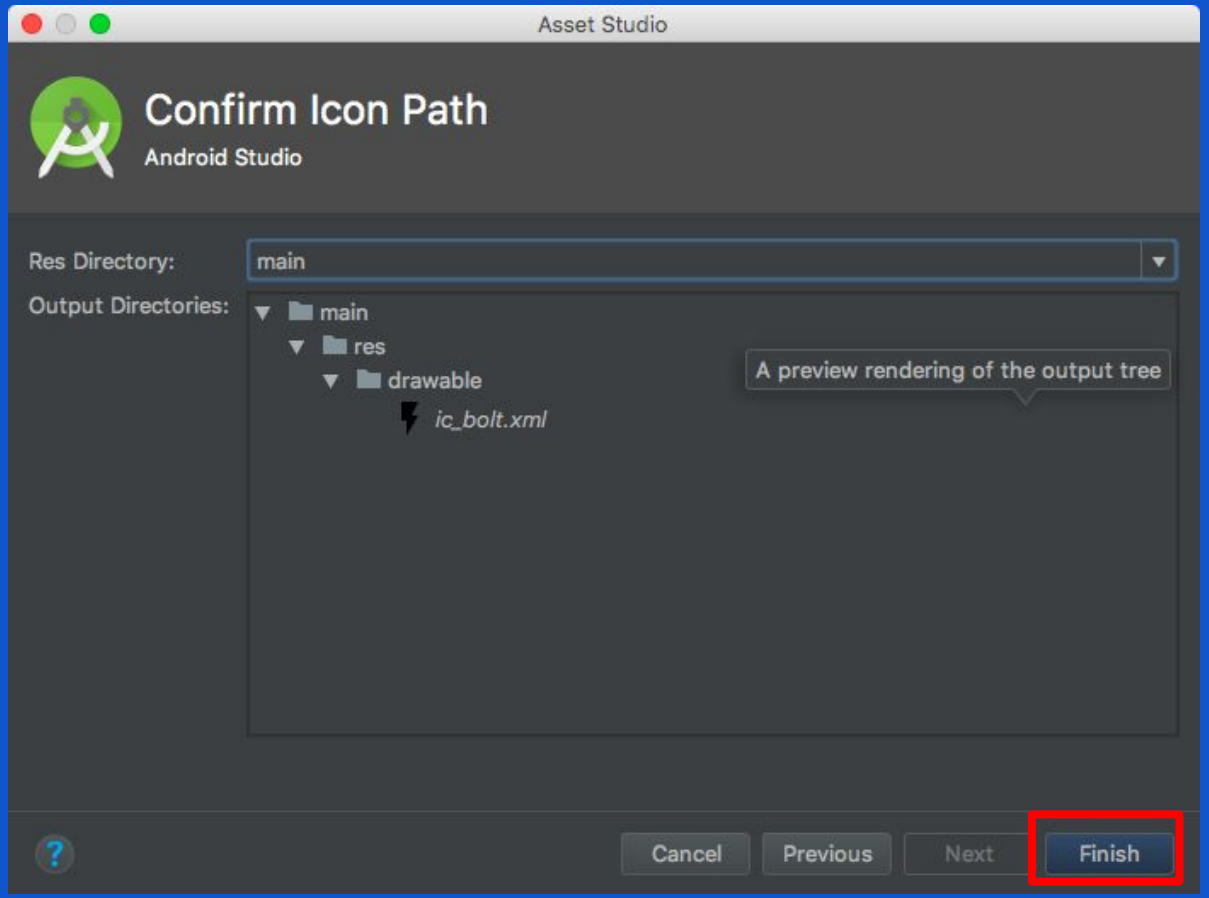



Selecione o arquivo *bolt.svg*

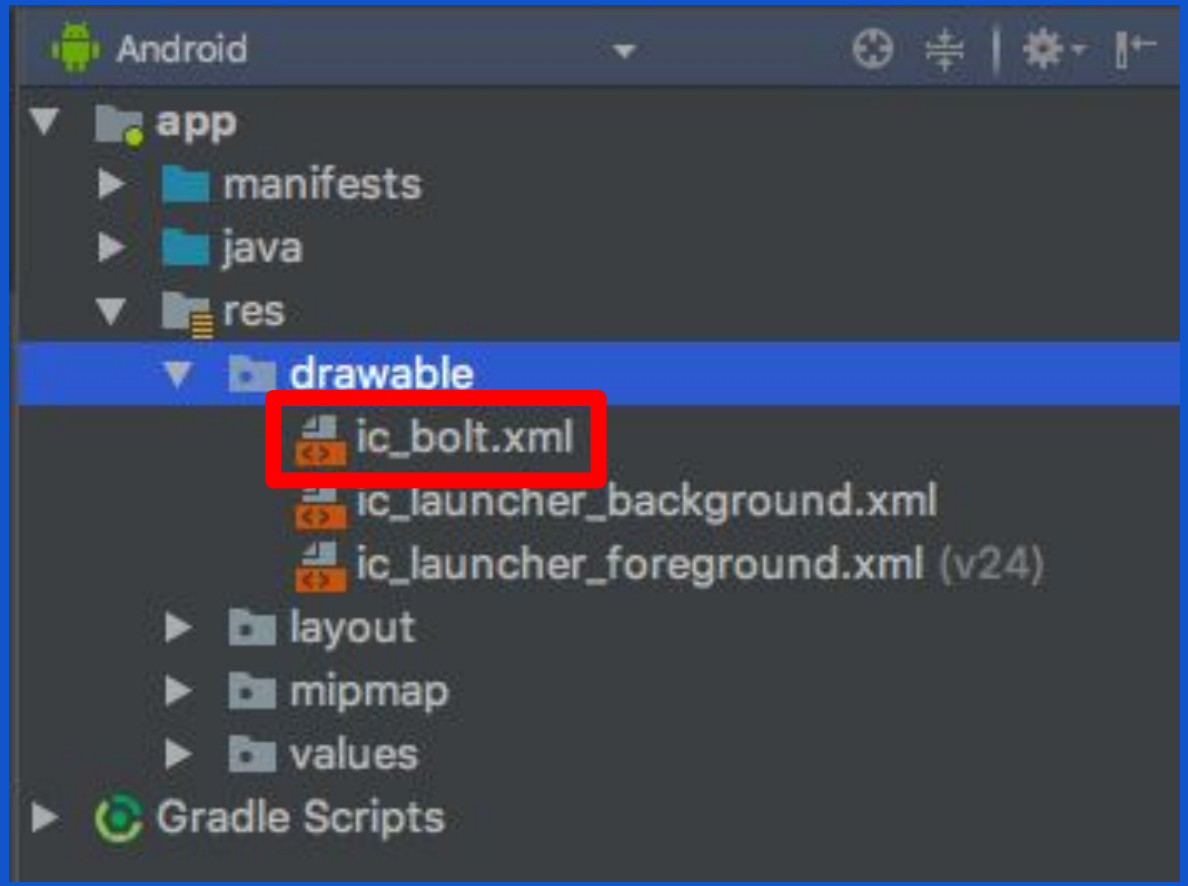
# Trabalhando com Scalable Vector Graphics (SVG)



# Trabalhando com Scalable Vector Graphics (SVG)



# Trabalhando com Scalable Vector Graphics (SVG)




O Android Studio vai criar um arquivo XML com os dados do SVG e colocar um prefixo *ic_* (ícone).

O guia do Android orienta colocarmos prefixos *ic_* para todos os ícones e prefixos *bg_* para imagens de fundo.

# Trabalhando com Toast Customizado (*ExibindoMensagem*)

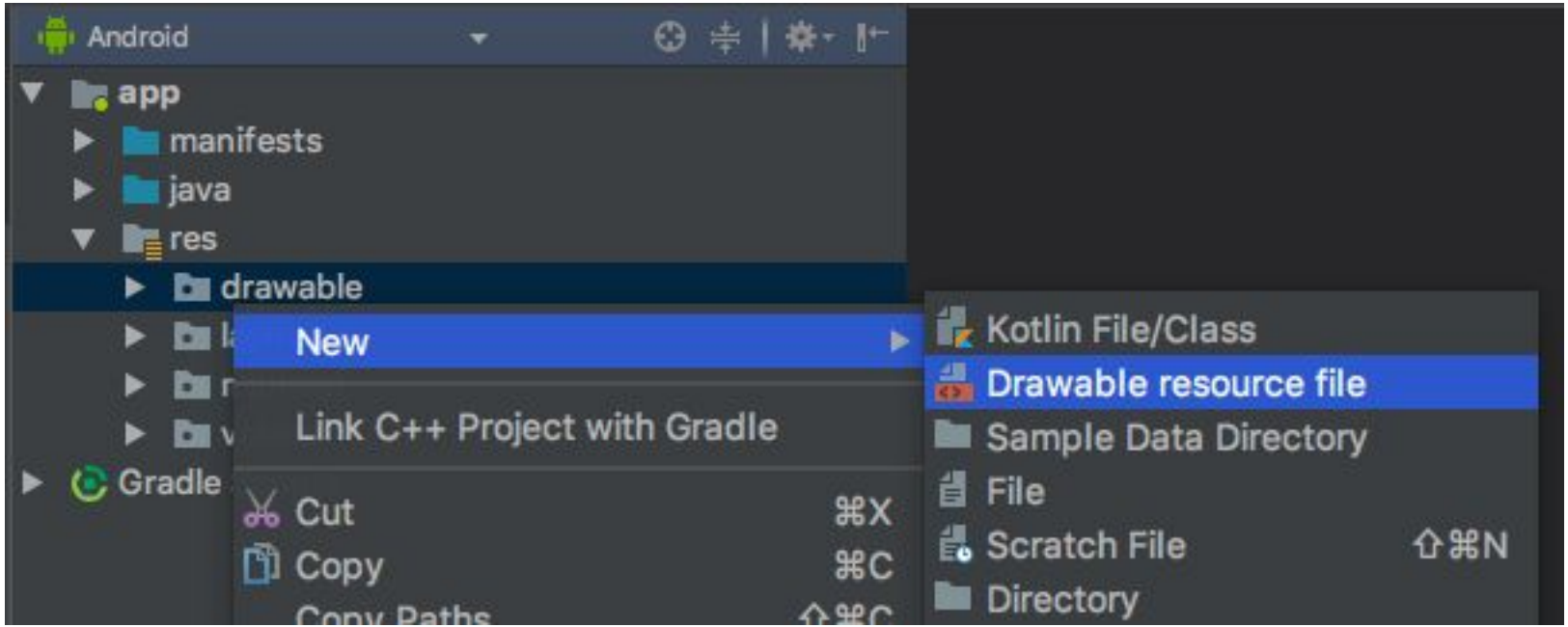
2 **Vamos definir as cores utilizadas no Toast no arquivo *colors.xml***

colors.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>

    <color name="amarelo">#ffc518</color>
    <color name="preto">#0d0d0d</color>
    <color name="branco">#ffffff</color>
</resources>
```

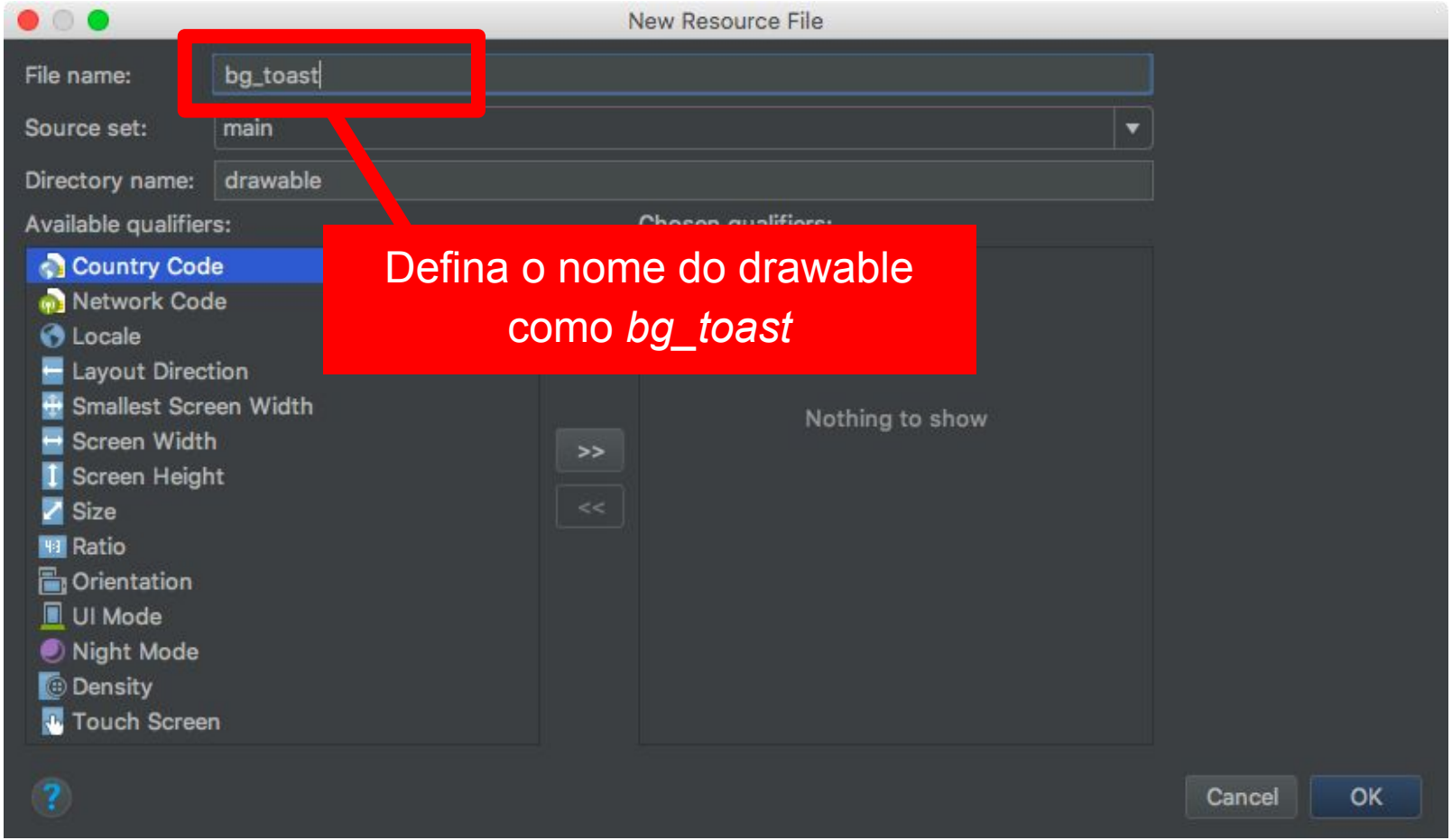
# Trabalhando com Toast Customizado (*ExibindoMensagem*)

3 **Vamos criar um shape para arredondar os cantos do Toast**



# Trabalhando com Toast Customizado (*ExibindoMensagem*)

3



# Trabalhando com Toast Customizado (*ExibindoMensagem*)

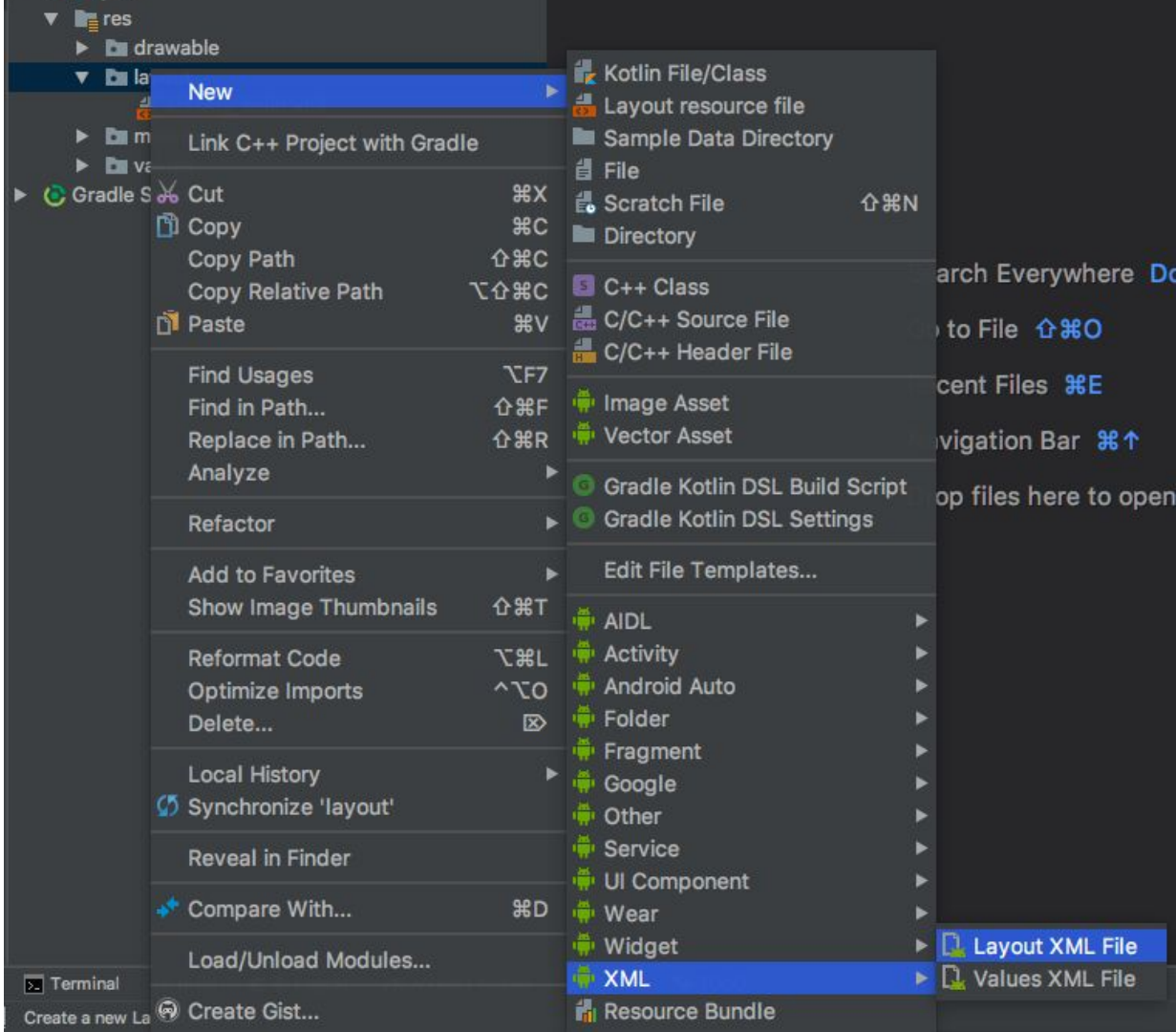
3 **O shape deve ser um *rectangle* com cor preta e *radius* de *5dp***

gb_toast.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <shape
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      android:shape="rectangle">
5      <solid android:color="@color/preto"/>
6      <corners android:radius="5dp"/>
7  </shape>
```

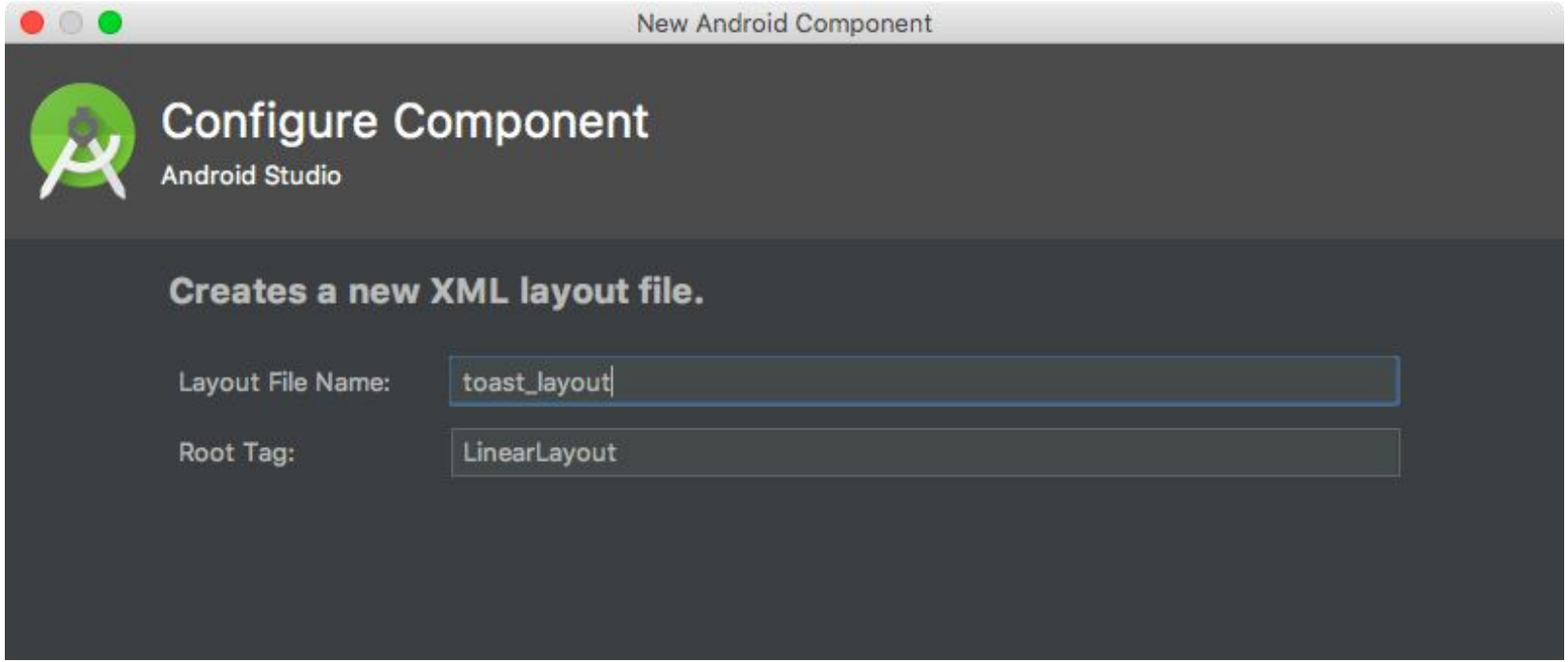
4

**Crie um novo arquivo de layout XML dentro da pasta *layout***



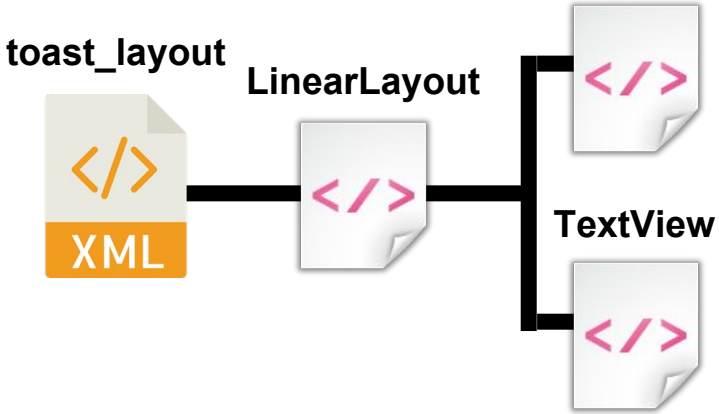
# Trabalhando com Toast Customizado (*ExibindoMensagem*)

4 **Nomeie o layout de *toast_layout* e matenha como *LinearLayout***



4

**A estrutura do Toast será composta pelos seguintes elementos:**




```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:background="@drawable/gb_toast"
    android:id="@+id/toast_container">
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="10dp"
        android:layout_marginLeft="10dp"
        android:layout_marginBottom="10dp"
        android:src="@drawable/ic_bolt"
        android:tint="@color/amarelo" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:textSize="18sp"
        android:textColor="@color/branco"
        android:text="toast"
        android:id="@+id/toast_message" />
</LinearLayout>
```

4

**Linear Layout:**

- Background apontando para *bg_toast*
- Defina um ID

**Image View:**

- Aponte para *ic_bolt*
- Altere a cor para amarelo através do atributo *tint*

**TextView:**

- Defina um ID
- Defina a cor do texto como branca

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:background="@drawable/gb_toast"
    android:id="@+id/toast_container">
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_marginTop="10dp"
        android:layout_marginLeft="10dp"
        android:layout_marginBottom="10dp"
        android:src="@drawable/ic_bolt"
        android:tint="@color/amarelo" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:textSize="18sp"
        android:textColor="@color/branco"
        android:text="toast"
        android:id="@+id/toast_message" />
</LinearLayout>
```

# 5 Crie um método *exibirToastCustomizado()* para exibir o Toast

```
public void exibirToastCustomizado(View view) {
    Toast toast = new Toast( context: this);
    toast.setGravity(Gravity.CENTER,  xOffset: 0,  yOffset: 0);
    toast.setDuration(Toast.LENGTH_LONG);

    LayoutInflater inflater = getLayoutInflater();
    View layout = inflater.inflate(
            R.layout.toast_layout,
            (LinearLayout) findViewById(R.id.toast_container)
    );

    TextView textView = (TextView) layout.findViewById(R.id.toast_message);
    textView.setText("Eu sou um toast customizado!");

    toast.setView(layout);
    toast.show();
}
```

# 5 Crie um método *exibirToastCustomizado()* para exibir o Toast

```
public void exibirToastCustomizado(View view) {
    Toast toast = new Toast( context: this);
    toast.setGravity(Gravity.CENTER,  xOffset: 0,  yOffset: 0);
    toast.setDuration(Toast.LENGTH_LONG);

    LayoutInflater inflater = getLayoutInflater();
    View layout = inflater.inflate(
            R.layout.toast_layout,
            (LinearLayout) findViewById(R.id
    );

    TextView textView = (TextView) layout.f
    textView.setText("Eu sou um toast custom

    toast.setView(layout);
    toast.show();
}
```

1

- Crie um Toast
- Centralize-o através da constante *Gravity.CENTER*. Os valores *xOffset* e *yOffset* afastam horizontal e verticalmente da esquerda (medindo em pixels).
- Defina a duração

# 5 Crie um método *exibirToastCustomizado()* para exibir o Toast

```
public void exibirToastCustomizado(View view) {
    Toast toast = new Toast( context: this);
    toast.setGravity(Gravity.CENTER,  xOffset: 0,  yOffset: 0);
    toast.setDuration(Toast.LENGTH_LONG);

    LayoutInflater inflater = getLayoutInflater();
    View layout = inflater.inflate(
            R.layout.toast_layout,
            (LinearLayout) findViewById(R.id.toast_container)
    );
```

2

**LayoutInflater:** o conceito de *inflater* é utilizado para indicar a "criação" de um novo layout a partir de um arquivo XML. Neste caso, estamos "inflando" o layout que será exibido no Toast. O método *inflate()* recebe dois parâmetros: o arquivo XML a partir do qual o *inflater* criará a nova estrutura e o elemento que vai receber essa estrutura (neste caso, o próprio *LinearLayout* exposto no XML).

# 5 Crie um método *exibirToastCustomizado()* para exibir o Toast

- Obtenha o *TextView* para alterar o texto que será exibido na tela.
  **Atenção:** note que neste caso o método *findViewById()* não é invocado a partir da Activity. Ele é invocado a partir do layout inflado.
- Defina o view do Toast e invoque o método *show()*

```
View layout = inflater.inflate(
        R.layout.toast_layout,
        (LinearLayout) findViewBy    .id.toast_container)
);

TextView textView = (TextView) layout.findViewById(R.id.toast_message);
textView.setText("Eu sou um toast customizado!");

toast.setView(layout);
toast.show();
}
```

3

# Trabalhando com Toast Customizado (*ExibindoMensagem*)

Por fim, acrescente um novo *Button* no layout principal do app e execute-o para ver o resultado!

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:text="Exibir Toast Customizado"
    android:onClick="exibirToastCustomizado" />
```

## Trabalhar [illegible] izado (*Exibi*

Por fim, [illegible]
principal d[illegible]

*Eu posso colocar um Button dentro do Toast customizado?*

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:text="Exibir Toast Customizado"
    android:onClick="exibirToastCustomizado" />
```

## Trabalhar [illegible] izado (*Exibi*

Por fim, [illegible]
principal [illegible]

*Eu posso colocar um Button dentro do Toast customizado?*

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:text="Exibir Toast Customizado"
    android:onClick="exibirToastCustomizado" />
```


Tecnicamente sim, porém caso haja necessidade do usuário interagir com a mensagem, a recomendação do Material Design é utilizar outro componente: **SnackBar**

# Trabalhando com SnackBar (*ExibindoMensagem*)

**Vamos agora criar um SnackBar...**

# Curiosidade

Reza a lenda que o componente *SnackBar* recebeu este nome porque permite ao usuário realizar uma ação a partir de uma mensagem rápida.

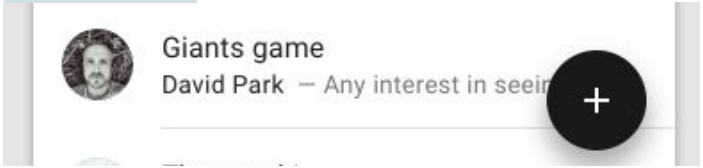
# Design Support Library

https://developer.android.com/topic/libraries/support-library/packages?hl=pt-br#design

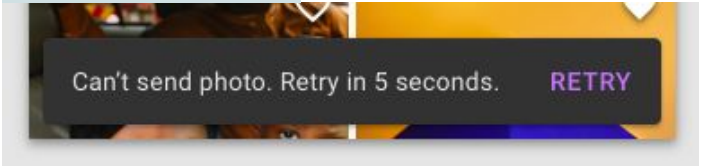
O Android possui uma biblioteca chamada *Design Support Library* que oferece vários recursos apresentados pela especificação do Material Design que as bibliotecas padrões de *Support Libraries* não oferecem por default.

Design Support Library

## FAB

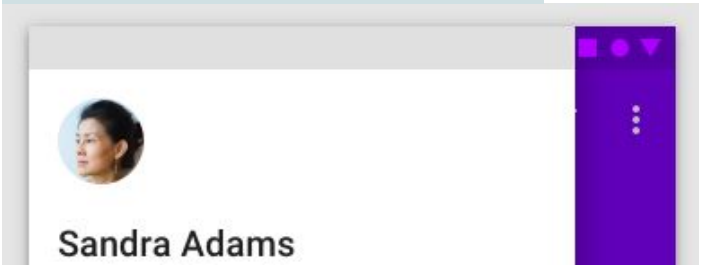



## SnackBar

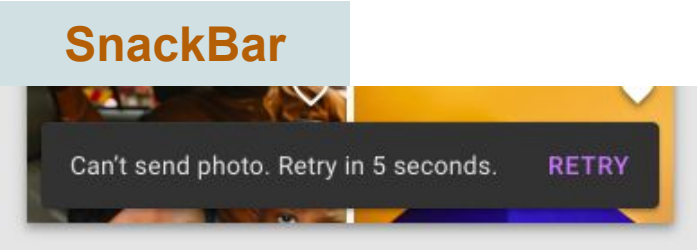



## Tabs


## Navigation Drawer

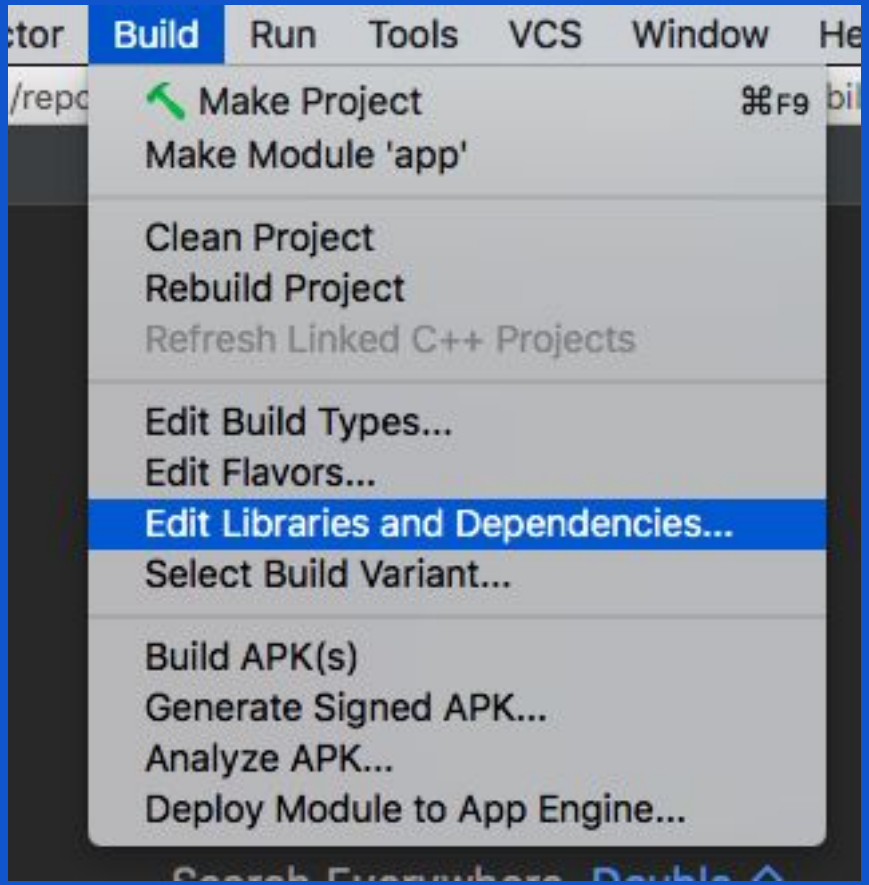


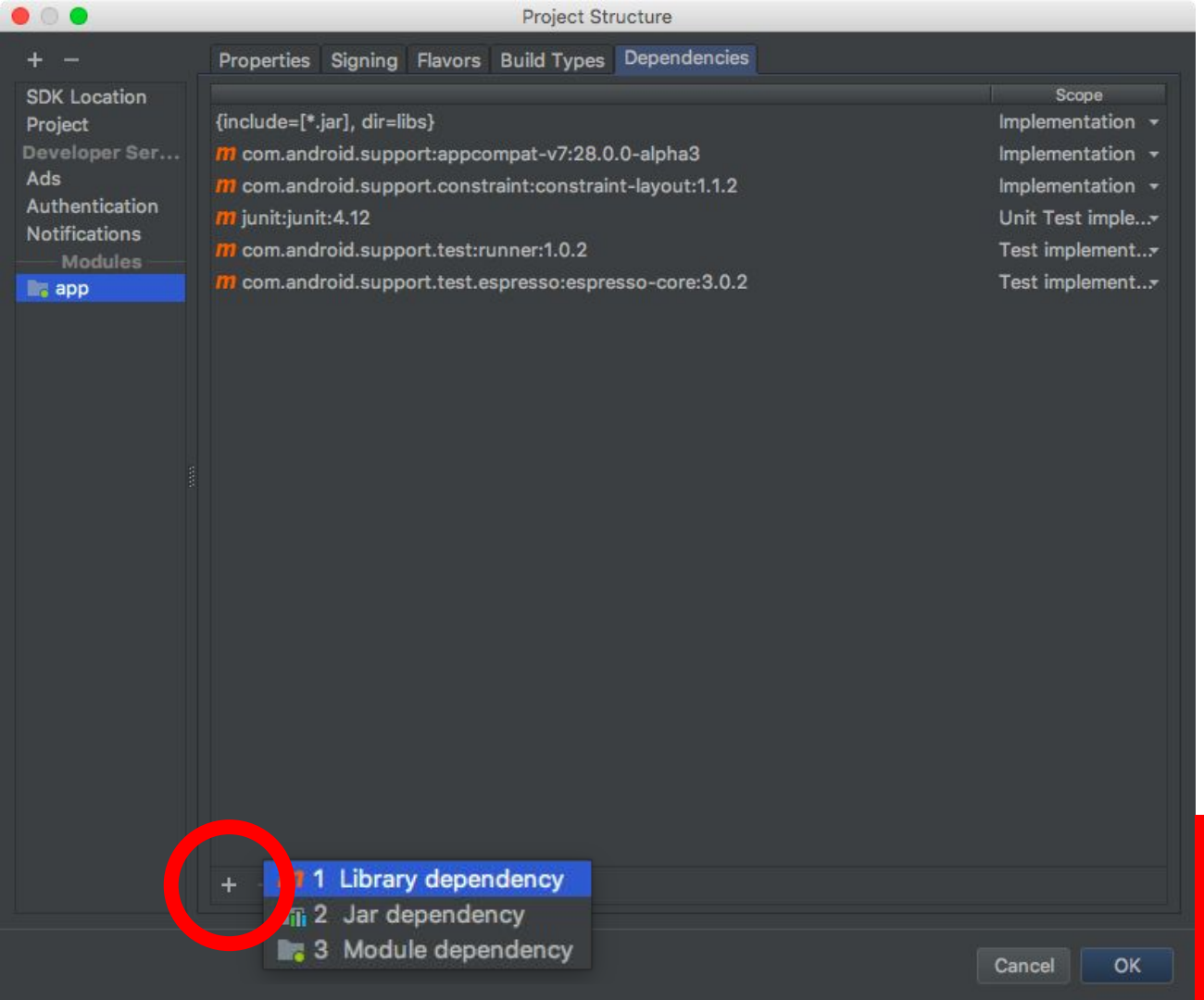
# Utilizando o SnackBar




## Design Support Library

Vamos importar a biblioteca *Design Support Library* através do menu *Edit Libraries and Dependencies...*






ou

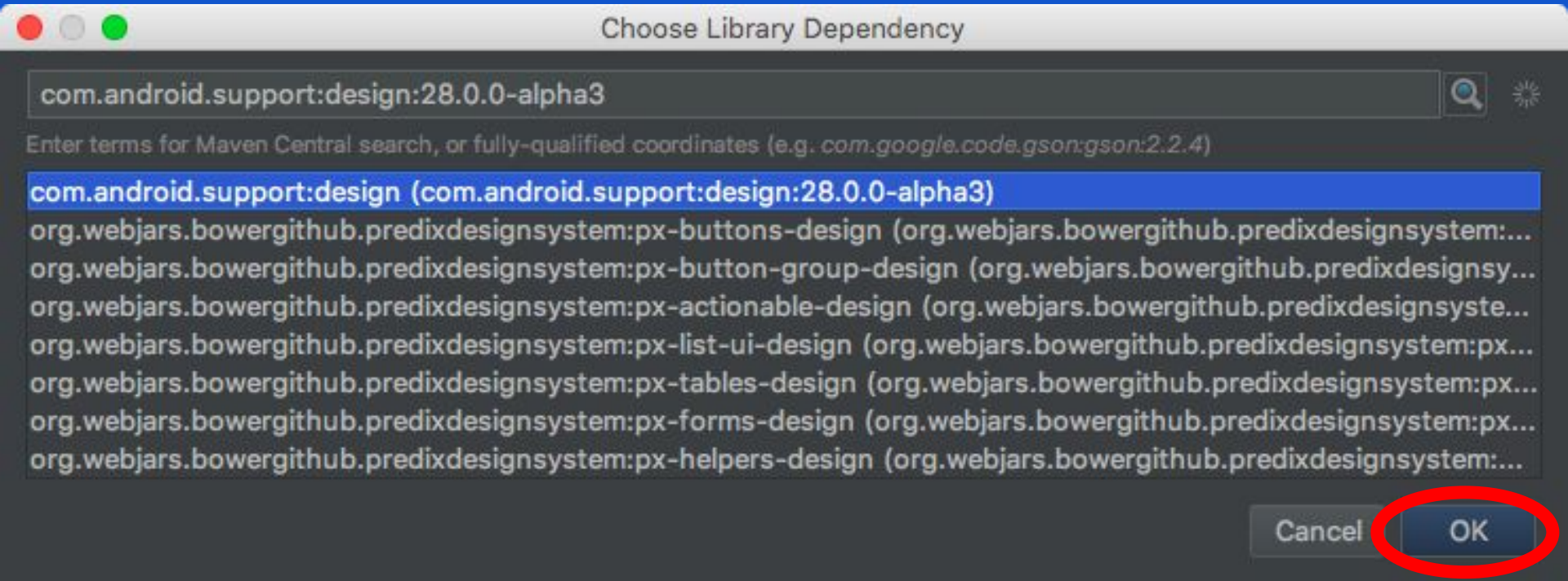
No **Windows** ou **Linux**, o botão de importar nova biblioteca fica no canto superior à direita.

---

No **Mac** fica na parte inferior (vide imagem)

Escolha a opção *Library dependency*

Digite "*design*" no campo de busca e selecione a biblioteca *com.android.support:design*. A versão exibida neste print certamente será diferente da exibida em seu Android Studio. Não se preocupe com isso, o Gradle vai resolver as dependências corretamente!



# Trabalhando com SnackBar (*ExibindoMensagem*)

Primeiramente vamos ajustar o layout para permitir o efeito de *swiping* que será aplicado ao SnackBar.

Para isso, basta envolver o layout que já temos com um *CoordinatorLayout*.
**Acrescente um ID a ele.**

Aproveite para acrescentar um *Button* chamado o método *exibirSnackbar()*

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/main_layout">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:text="Exibir Toast"
            android:onClick="exibirToast" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:text="Exibir Toast Customizado"
            android:onClick="exibirToastCustomizado" />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:text="Exibir Snackbar"
            android:onClick="exibirSnackbar" />

    </LinearLayout>
</android.support.design.widget.CoordinatorLayout>
```

# Trabalhando com SnackBar (*ExibindoMensagem*)

Vamos começar criando um simples SnackBar (sem ação ou tratamento do evento de *swiping*).

```
public void exibirSnackbar(View view) {
    CoordinatorLayout main = (CoordinatorLayout) findViewById(R.id.main_layout);
    Snackbar snackbar = Snackbar.make(main,  text: "Eu sou um Snackbar", Snackbar.LENGTH_LONG);
    snackbar.show();
}
```


Neste exemplo, o *swiping* já funciona, porém não há Listener escutando a ação.

# Trabalhando com SnackBar (*ExibindoMensagem*)

Vamos agora acrescentar uma ação "DESFAZER".

```
public void exibirSnackbar(View view) {
    CoordinatorLayout main = (CoordinatorLayout) findViewById(R.id.main_layout);
    Snackbar snackbar = Snackbar.make(main,  text: "Eu sou um Snackbar", Snackbar.LENGTH_LONG);
    snackbar.setAction( text: "DESFAZER", new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Toast.makeText( context: MainActivity.this,  text: "Parabéns, ação desfeita!", Toast.LENGTH_SHORT).show();
        }
    });
    snackbar.show();
}
```

# Trabalhando com SnackBar (*ExibindoMensagem*)

Vamos agora escutar os eventos do SnackBar, em especial o que é comumente chamado de **swipe-to-dismiss**.

```
snackbar.addCallback(new Snackbar.Callback() {
    @Override
    public void onDismissed(Snackbar snackbar, int event) {}
    @Override
    public void onShown(Snackbar snackbar) {}
});
```

Primeiramente vamos adicionar um objeto de callback no SnackBar através do método *addCallback()*. A classe anônima *Snackbar.Callback* deve implementar dois métodos:

- onDismissed(): invocado após o fechamento do SnackBar
- onShown(): invocado após a exibição do SnackBar

# Trabalhando com SnackBar (*ExibindoMensagem*)

```
snackbar.addCallback(new Snackbar.Callback() {
    @Override
    public void onDismissed(Snackbar snackbar, int event) {}
    @Override
    public void onShown(Snack
});
```

Entretanto, existem várias ações que podem fechar o SnackBar!
https://developer.android.com/reference/android/support/design/widget/Snackbar.Callback

Vamos usar:
DISMISS_EVENT_SWIPE

**DISMISS_EVENT_ACTION**

Indicates that the Snackbar was dismissed via an action click.

**DISMISS_EVENT_CONSECUTIVE**

Indicates that the Snackbar was dismissed from a new Snackbar being shown.

**DISMISS_EVENT_MANUAL**

Indicates that the Snackbar was dismissed via a call to `dismiss()`.

**DISMISS_EVENT_SWIPE**

Indicates that the Snackbar was dismissed via a swipe.

**DISMISS_EVENT_TIMEOUT**

Indicates that the Snackbar was dismissed via a timeout.

# Trabalhando com SnackBar (*ExibindoMensagem*)

No método *onDismissed()*, verifique se o código do evento é *DISMISS_EVENT_SWIPE* e, se for o caso, exiba uma mensagem.

```
snackbar.addCallback(new Snackbar.Callback() {
    @Override
    public void onDismissed(Snackbar snackbar, int event) {
        if (event == Snackbar.Callback.DISMISS_EVENT_SWIPE) {
            Toast.makeText( context: MainActivity.this,  text: "Ok, você cancelou!", Toast.LENGTH_SHORT).show();
        }
    }
    @Override
    public void onShown(Snackbar snackbar) {}
});
```

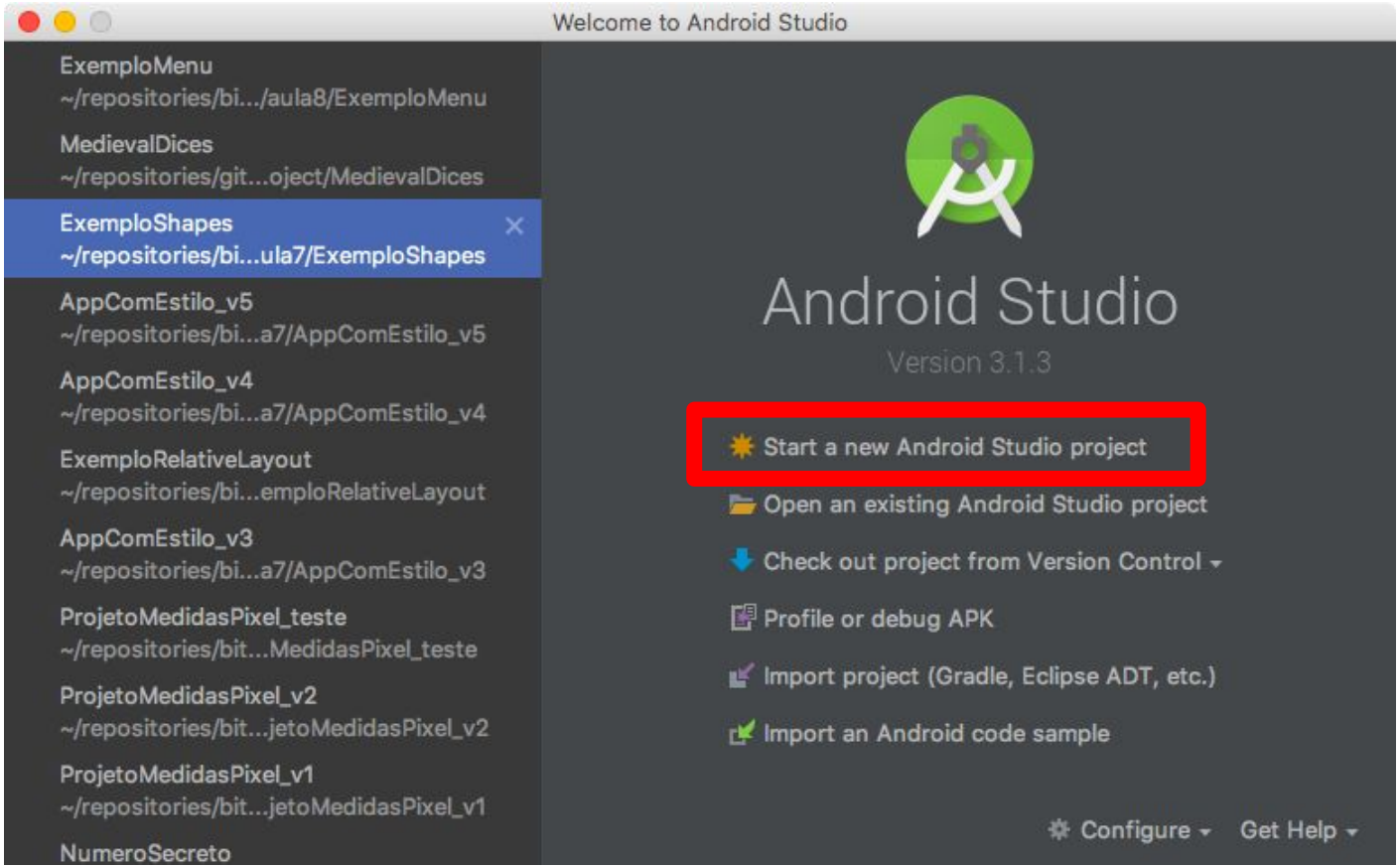
# Exercício em Sala

Crie um novo projeto chamado *Notepad* (*Empty Activity, API 15*). Ao clicar no botão "Limpar", o app deve exibir um *SnackBar* perguntando se o usuário realmente deseja apagar o texto. Ao clicar em APAGAR, deve-se limpar o texto.

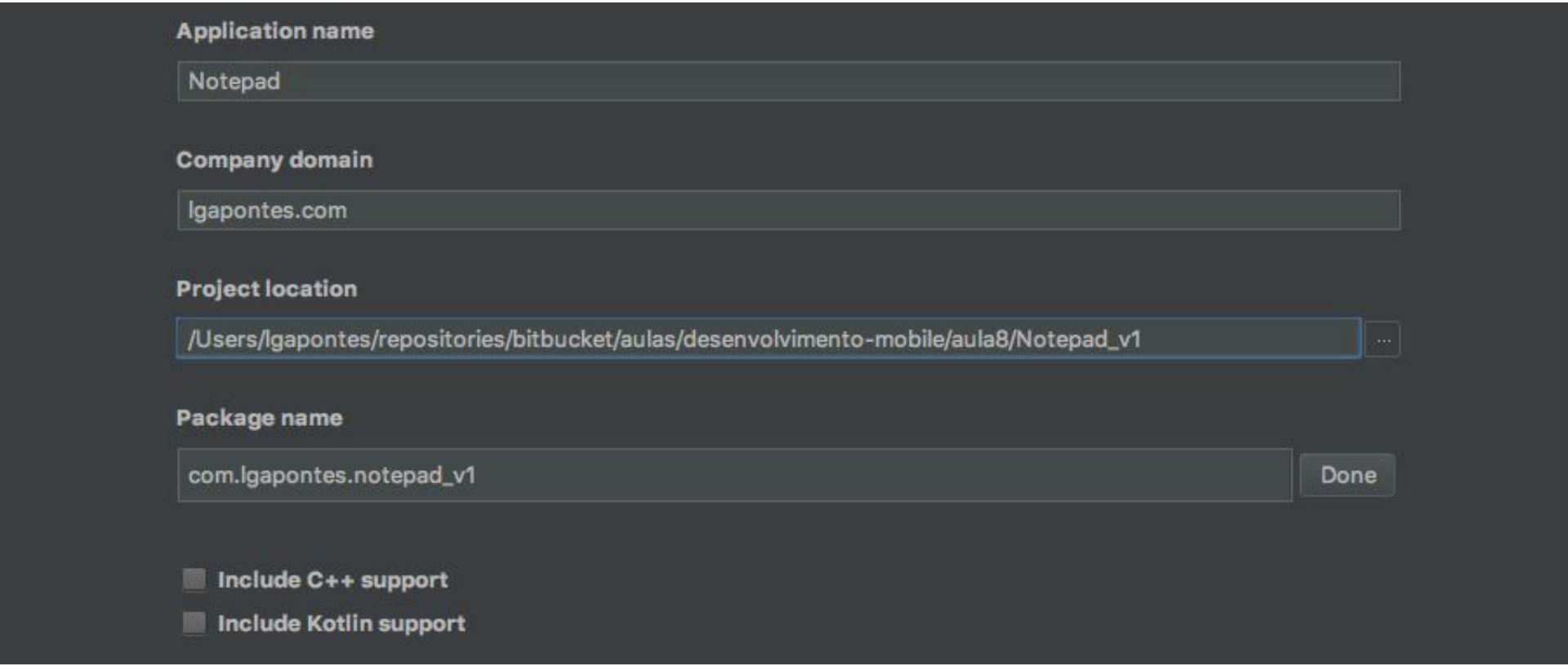
Veja ao lado como criar um EditText com muitas linhas!

```
<EditText
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="5dp"
    android:background="@color/cinza"
    android:textSize="20sp"
    android:fontFamily="monospace"
    android:gravity="top|left"
    android:scrollbars="vertical" />
```

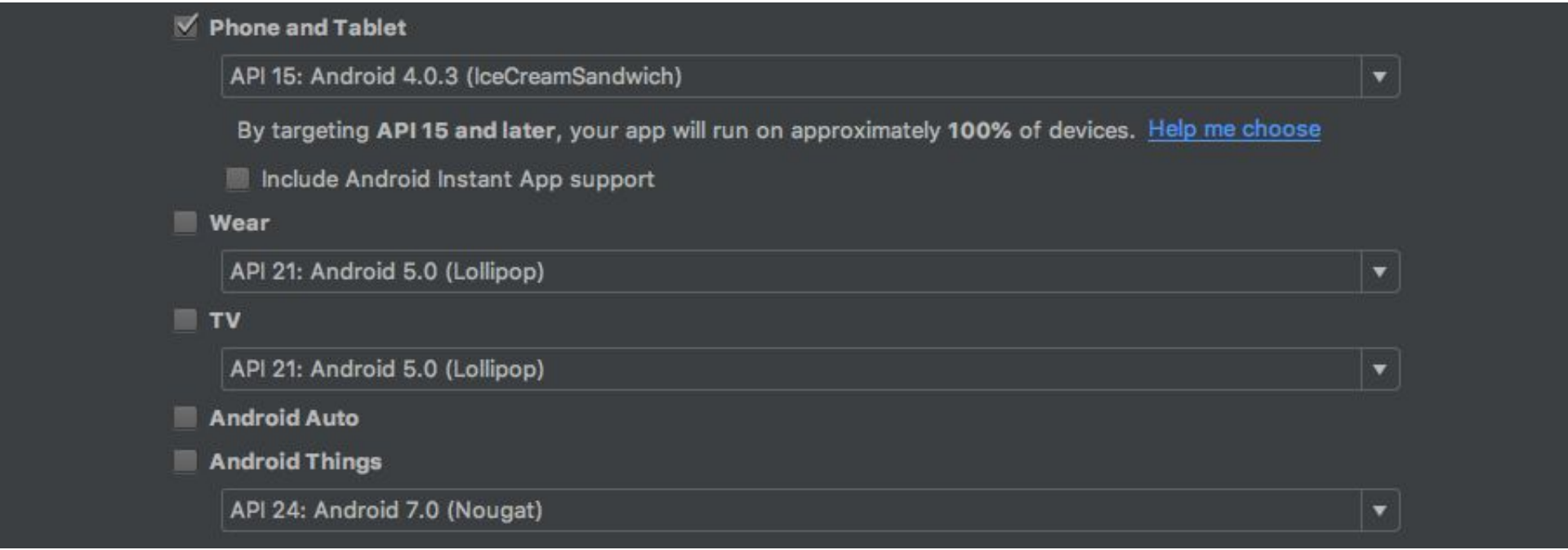
# Resolvendo o Exercício



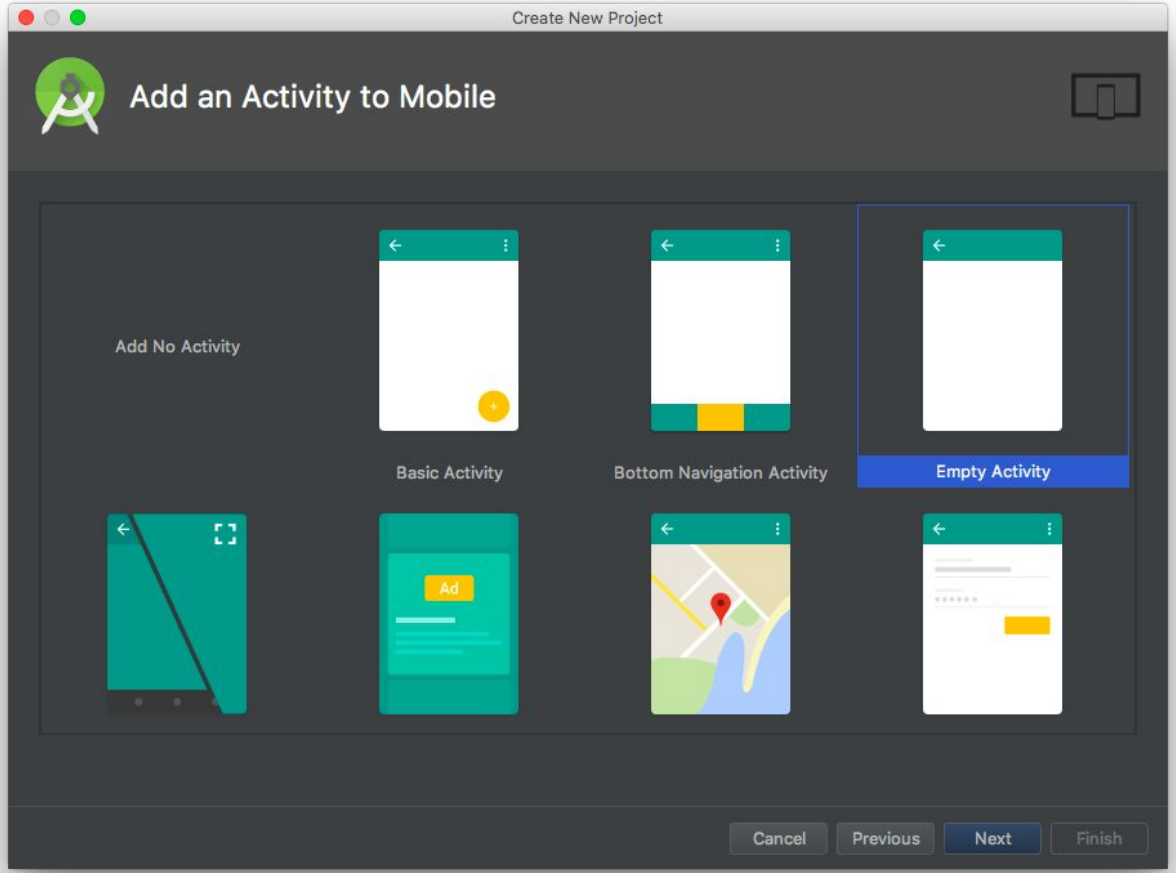
# Resolvendo o Exercício



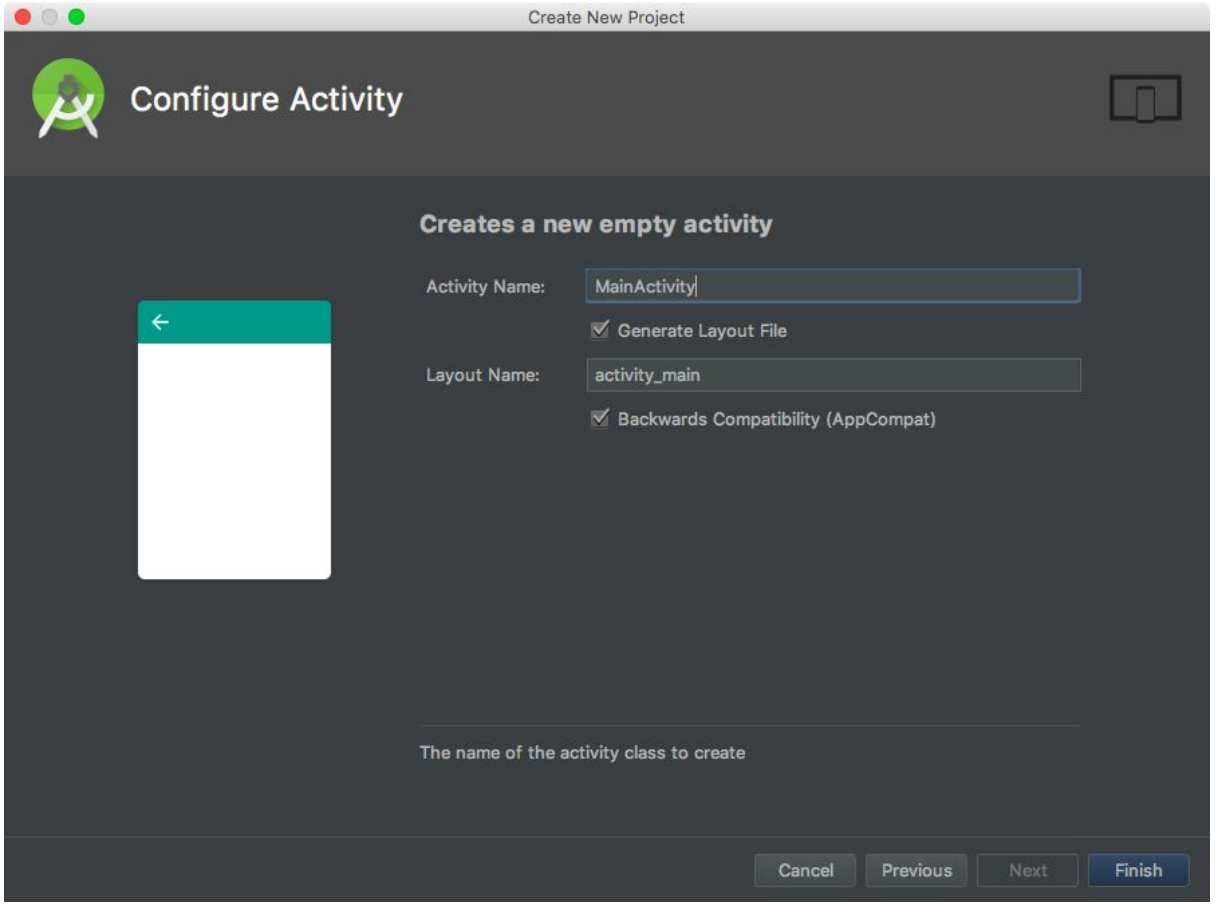
# Resolvendo o Exercício



# Resolvendo o Exercício



# Resolvendo o Exercício



# Resolvendo o Exercício

colors.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
    <color name="cinza">#efefef</color>
</resources>
```

strings.xml

Edit translations for all locales in the translations editor.

```xml
<resources>
    <string name="app_name">Notepad</string>
    <string name="texto_botao_limpar">Limpar</string>
    <string name="mensagem_confirmacao">Deseja realmente apagar o texto?</string>
    <string name="snackbar_action">APAGAR</string>
</resources>
```

# Resolvendo o Exercício

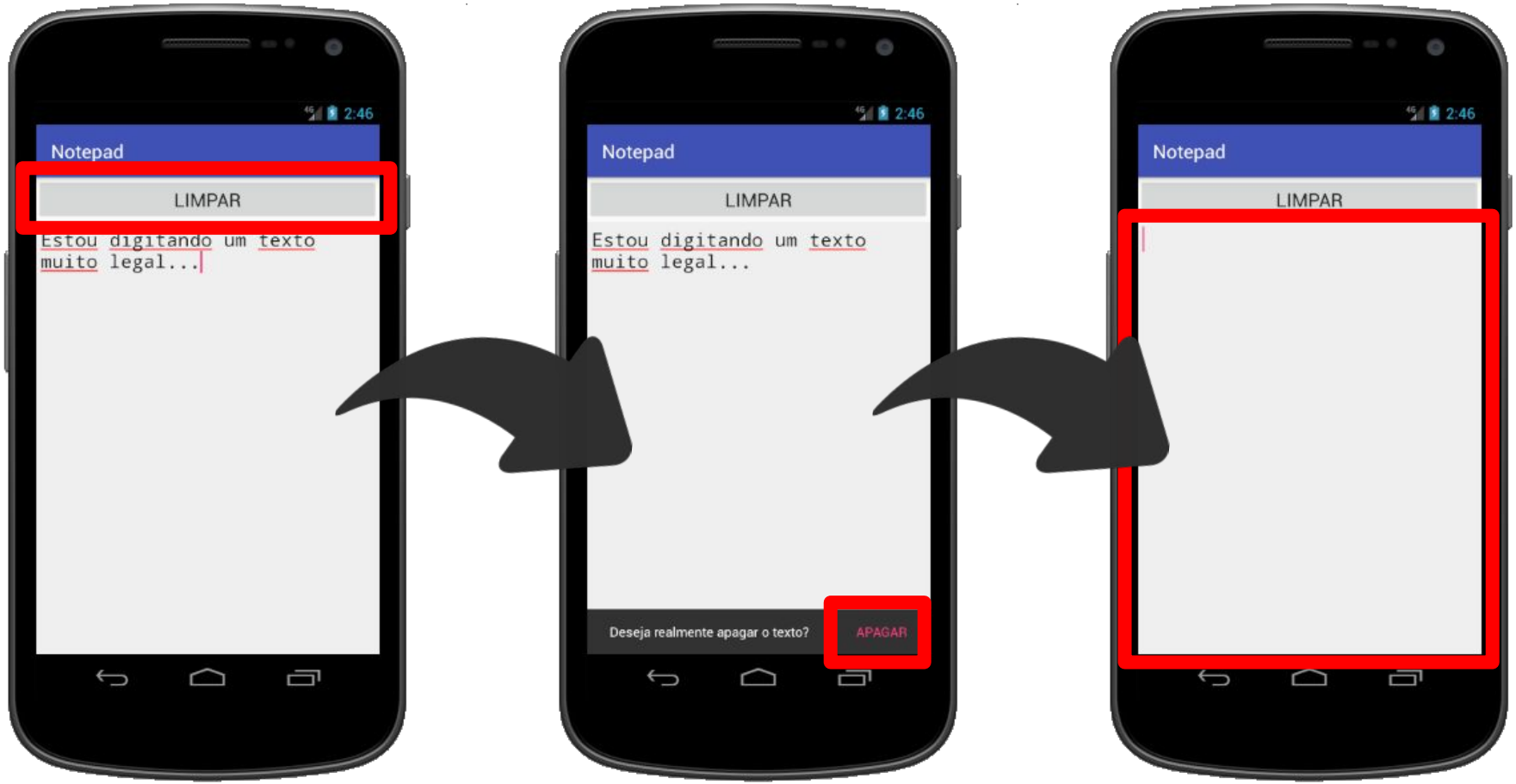
activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/coordinator_layout">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">
        <Button
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/texto_botao_limpar"
            android:textSize="20sp"
            android:id="@+id/botao_limpar"/>
        <EditText
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:padding="5dp"
            android:background="@color/cinza"
            android:textSize="20sp"
            android:fontFamily="monospace"
            android:gravity="top|left"
            android:scrollbars="vertical"
            android:id="@+id/edit_text" />
    </LinearLayout>
</android.support.design.widget.CoordinatorLayout>
```

# Resolvendo o Exercício

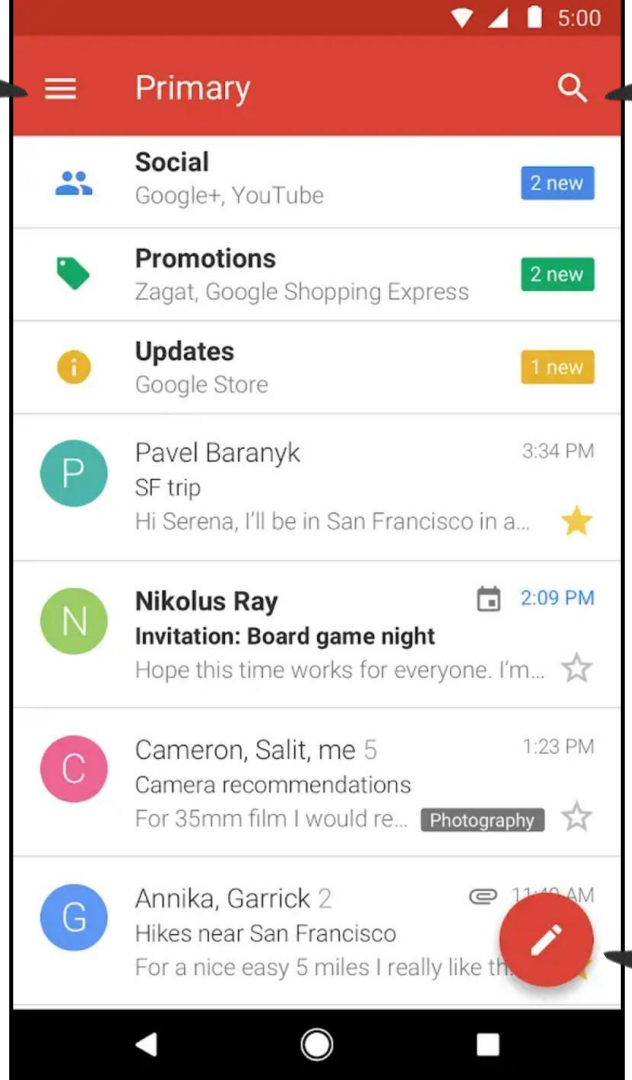
```java
public class MainActivity extends AppCompatActivity {

    private CoordinatorLayout layout;
    private Button botaoLimpar;
    private EditText editText;

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

        layout = (CoordinatorLayout) findViewById(R.id.coordinator_layout);
        editText = (EditText) findViewById(R.id.edit_text);

        botaoLimpar = (Button) findViewById(R.id.botao_limpar);
        botaoLimpar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar snackbar = Snackbar.make(layout, "Deseja realmente apagar o texto?", Snackbar.LENGTH_LONG);
                snackbar.setAction("APAGAR", new View.OnClickListener() {
                    @Override
                    public void onClick(View view) {
                        editText.setText("");
                    }
                });
                snackbar.show();
            }
        });
    }
}
```

# Resolvendo o Exercício



Mas esse botão "Limpar" em layout mobile é uma boa prática?


**Discussão em sala:**
Abram algum aplicativo mobile em seu celular Android e veja onde ficam posicionados os botões de ação...

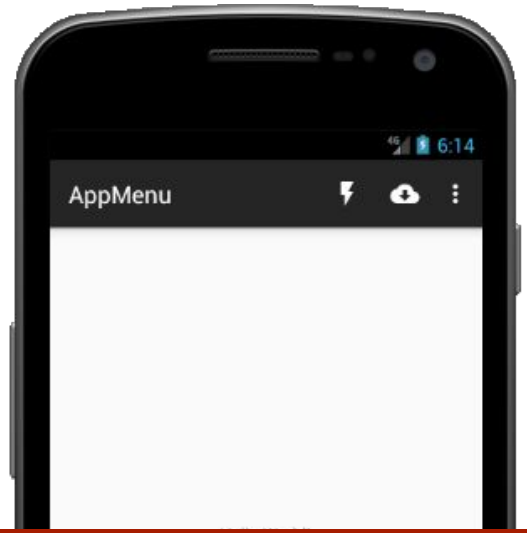


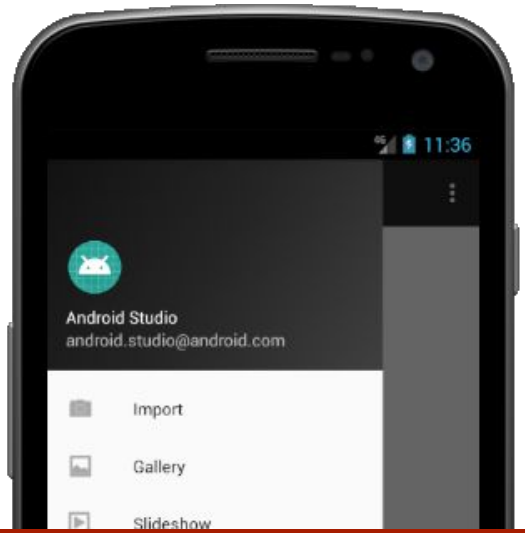

**Navigation Drawer**

**Este componente será estudado no futuro pois antes precisamos abordar os conceitos de *Fragment***

**ActionBar Menu**

**Floating Action Button**




# ActionBar Menu VS Navigation Drawer

Recomendado para casos mais simples, com poucas Views, onde as principais ações podem ficar expostas diretamente sem poluir o ActionBar Menu.

Recomendado para Apps mais complexas, com muitas Views, cuja navegação seria facilitada quando houver uma Navigation Drawer.

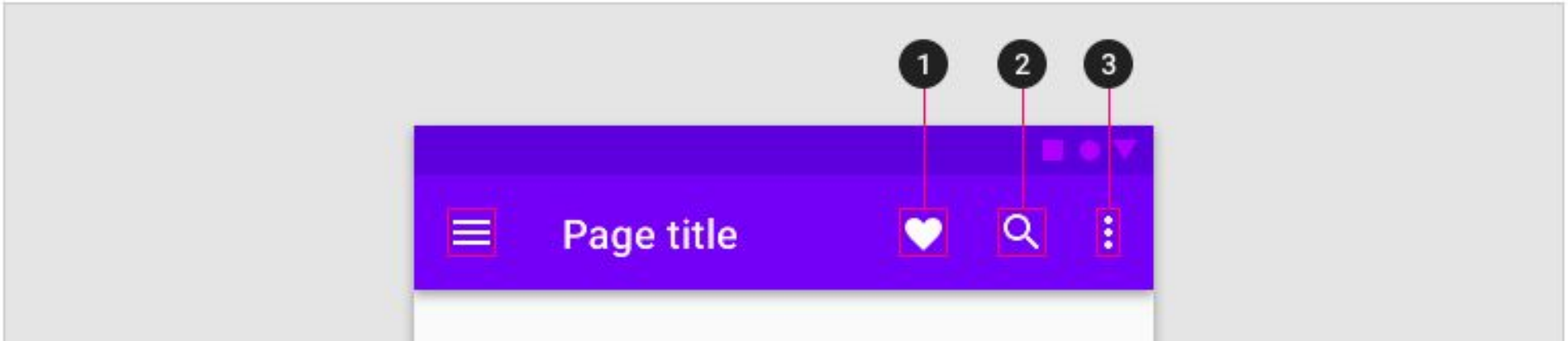
# ActionBar Menu

Material Design:
https://material.io/design/components/app-bars-top.html#anatomy

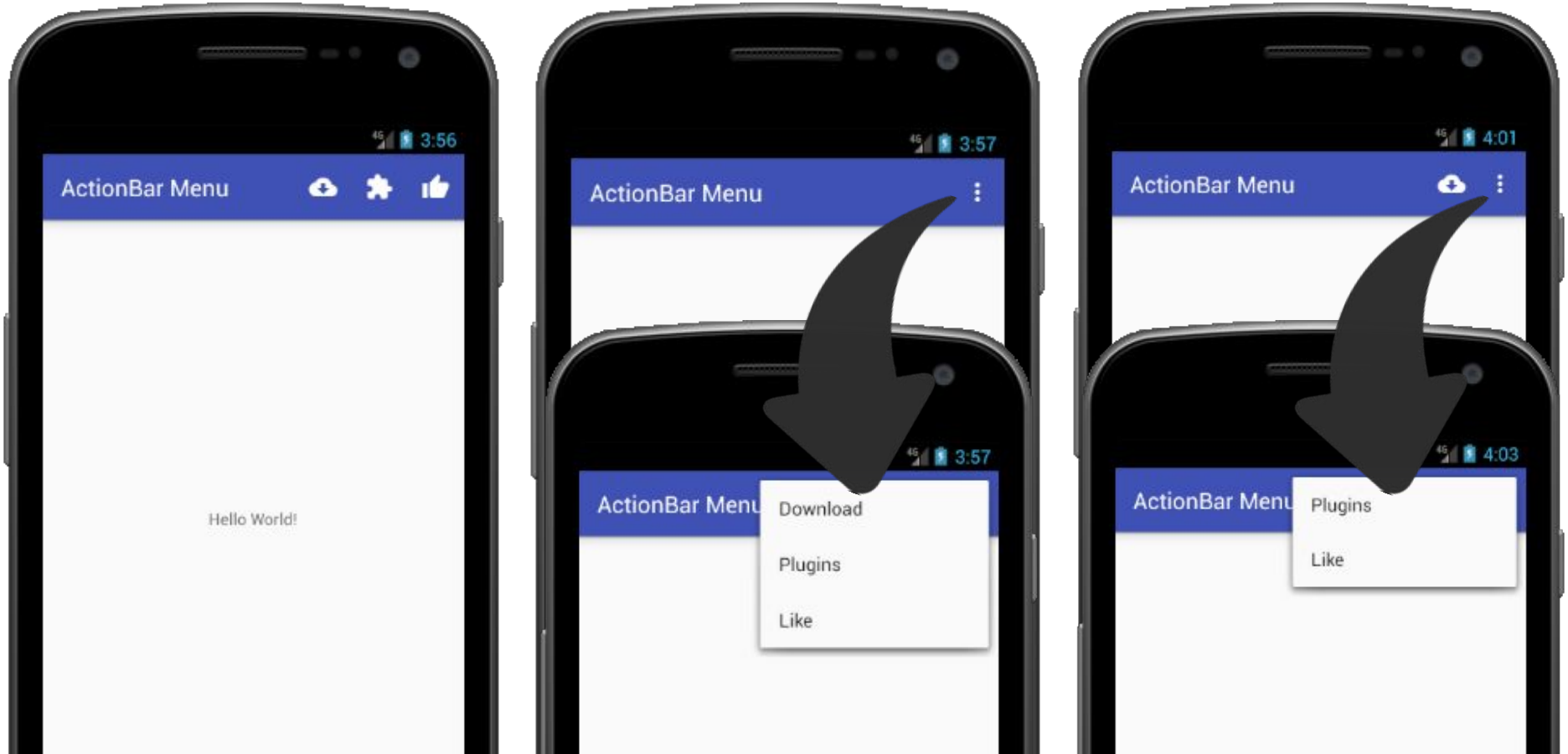
## Icon placement

Place most-used actions on the left, progressing towards the least-used actions on the far right. Any remaining actions that don't fit on the app bar can go into an overflow menu.

Actions move into and out of the overflow menu as the app bar width changes, such as if a device is rotated from landscape to portrait orientation. More guidance on this is available in **top app bar behavior.**

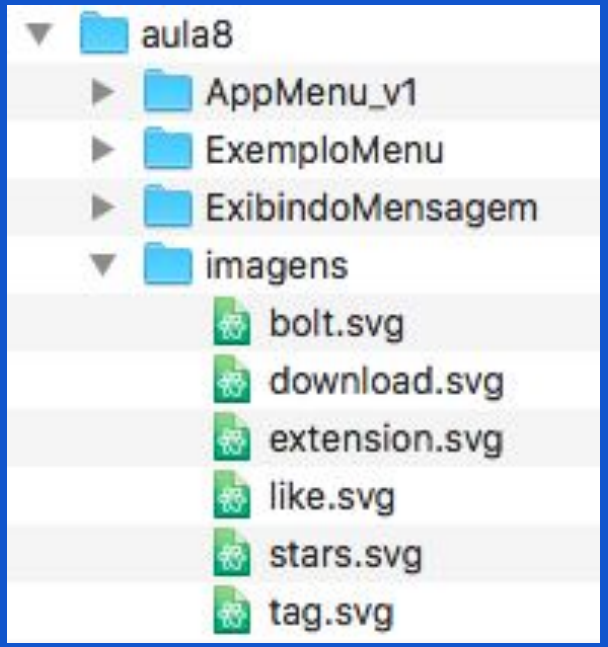


# ActionBar Menu



# Imagens para os menus (em formato SVG)

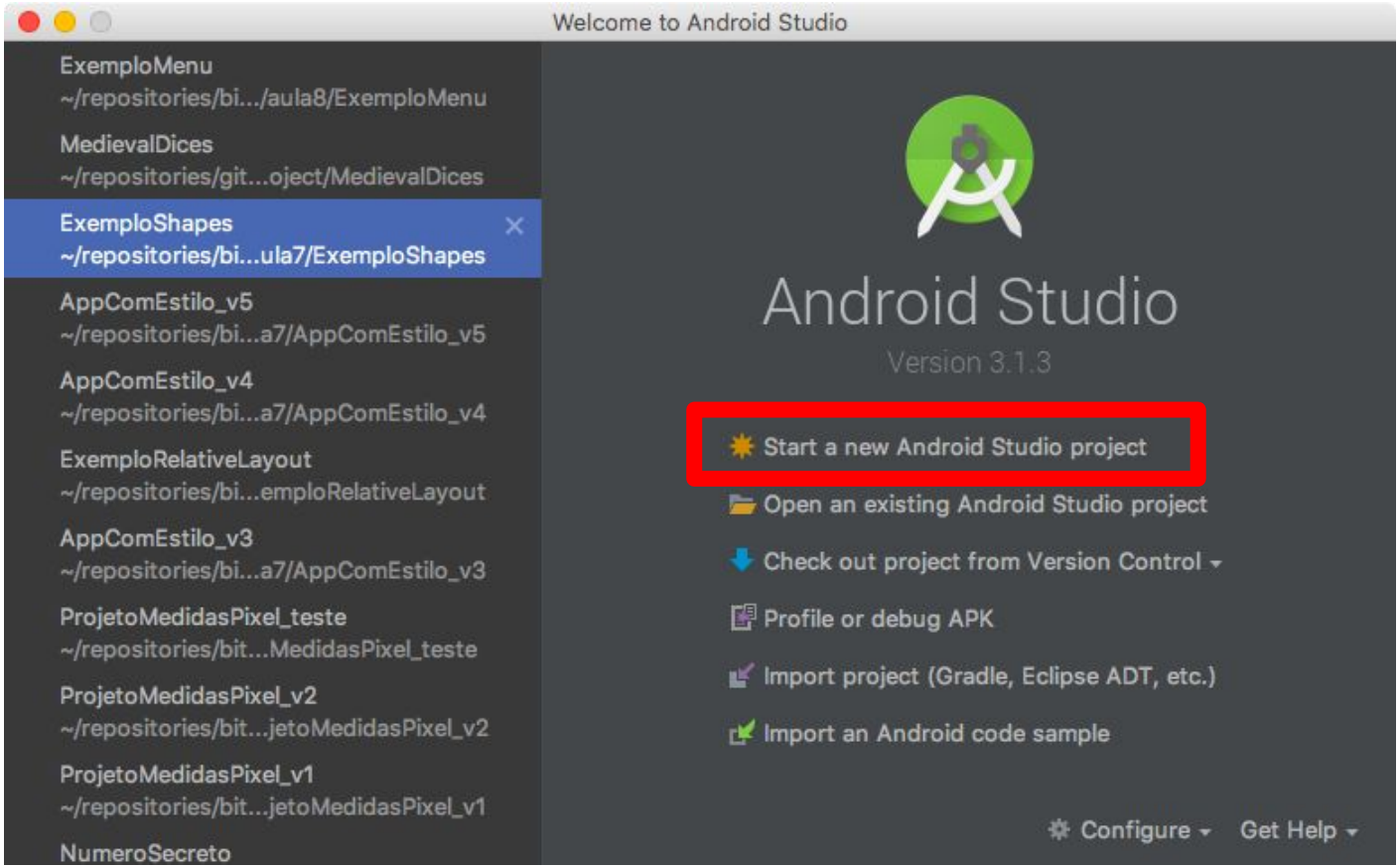
Baixe 6 imagens no Material Design Icons (https://material.io/tools/icons/), com formato SVG. Caso prefira, na pasta *imagens* da Aula 8 do EAD já existem 6 imagens.



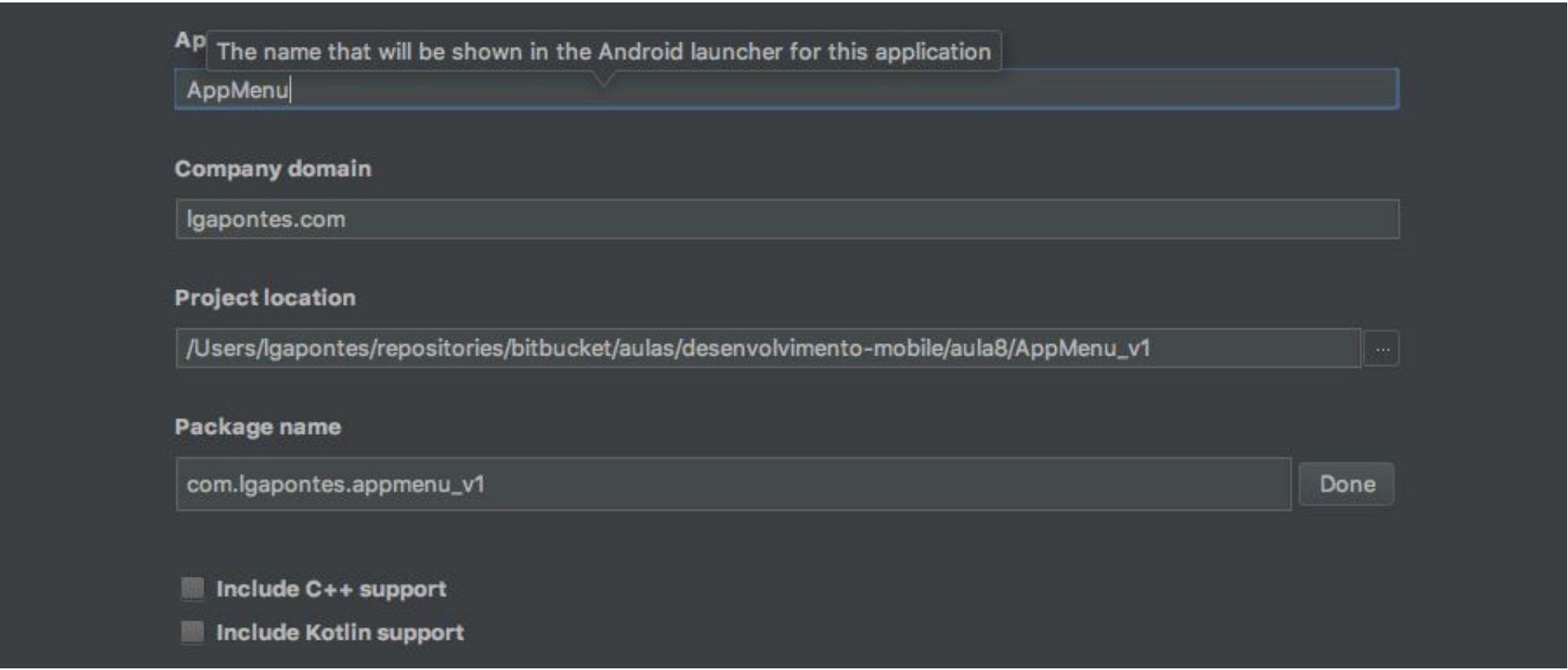

As imagens devem se chamar:

- `bolt.svg`
- `download.svg`
- `extension.svg`
- `like.svg`
- `stars.svg`
- `tag.cvg`

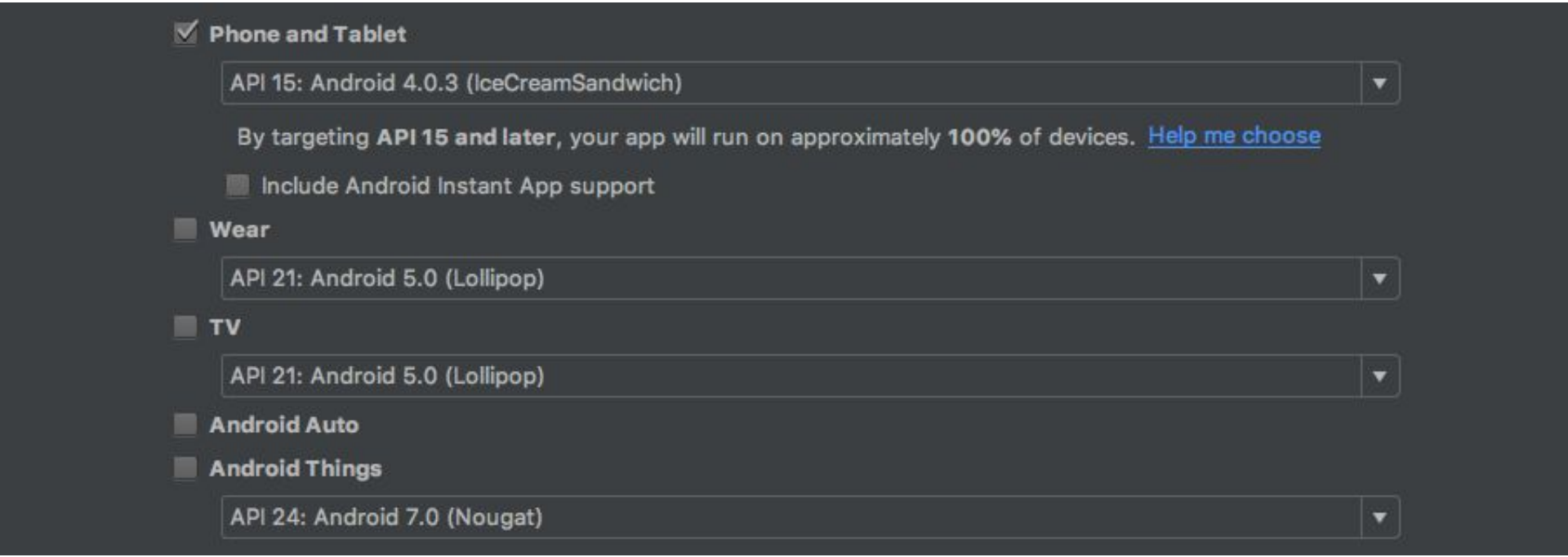
# Trabalhando com ActionBar Menu (*AppMenu*)



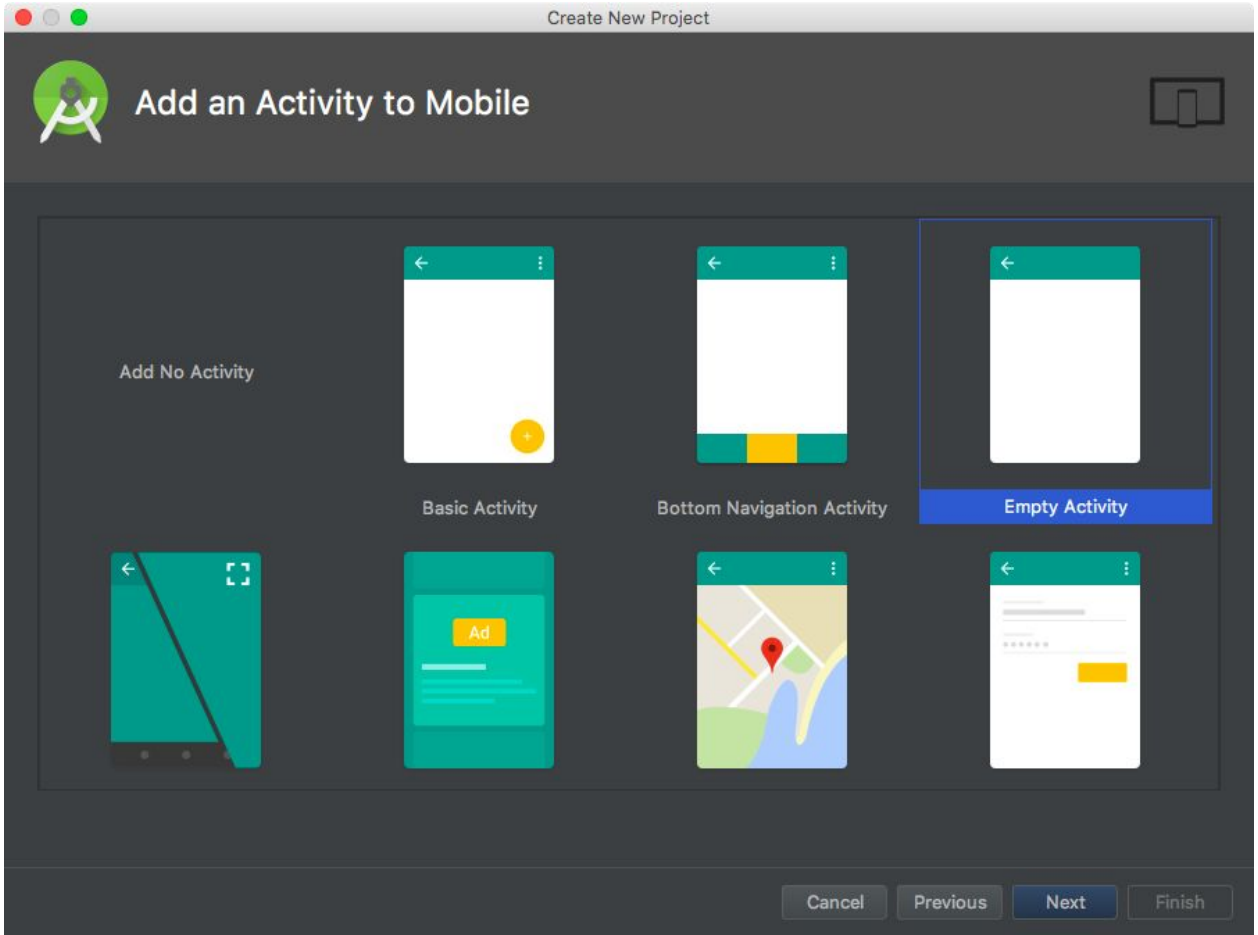
# Trabalhando com ActionBar Menu (*AppMenu*)



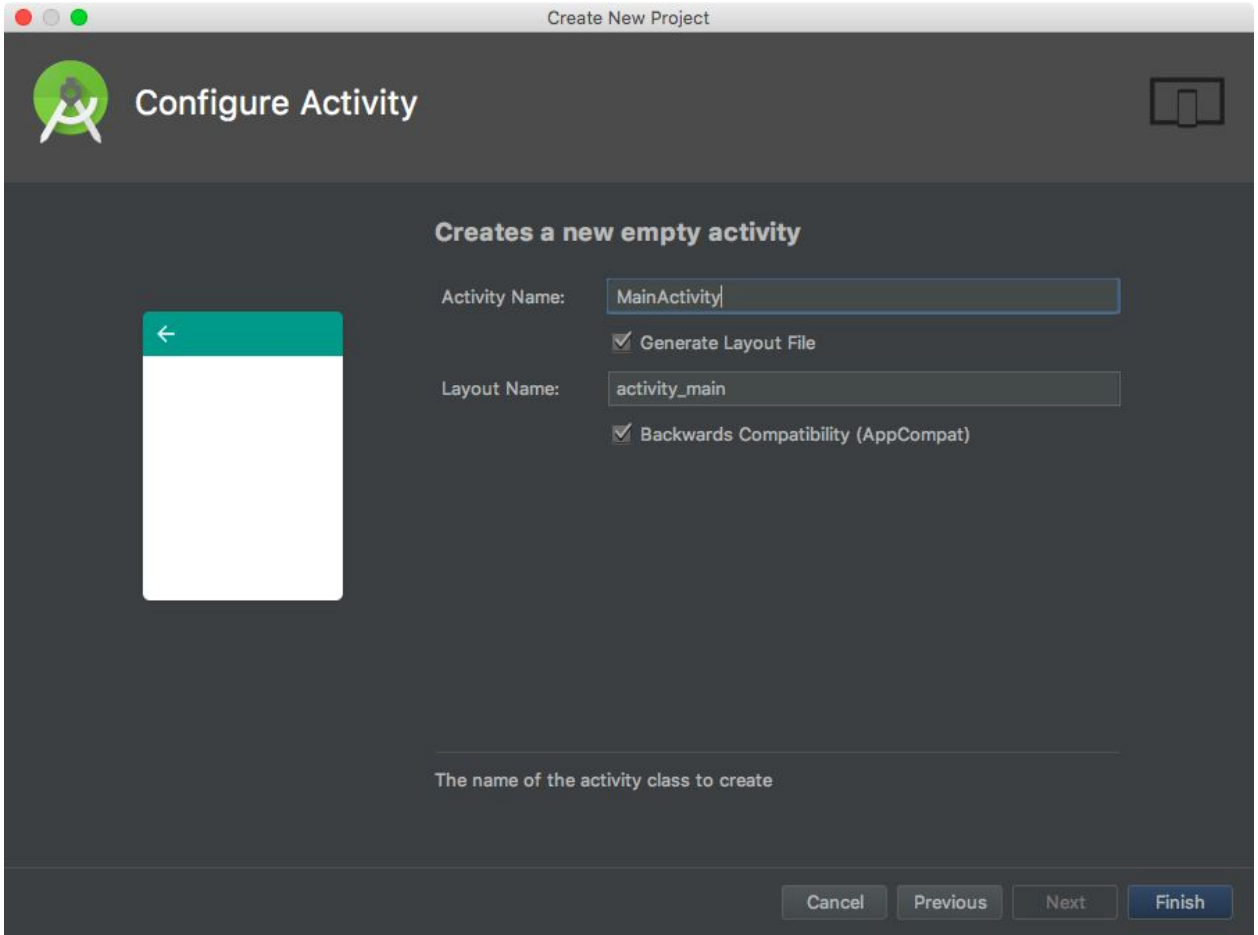
# Trabalhando com ActionBar Menu (*AppMenu*)



# Trabalhando com ActionBar Menu (*AppMenu*)

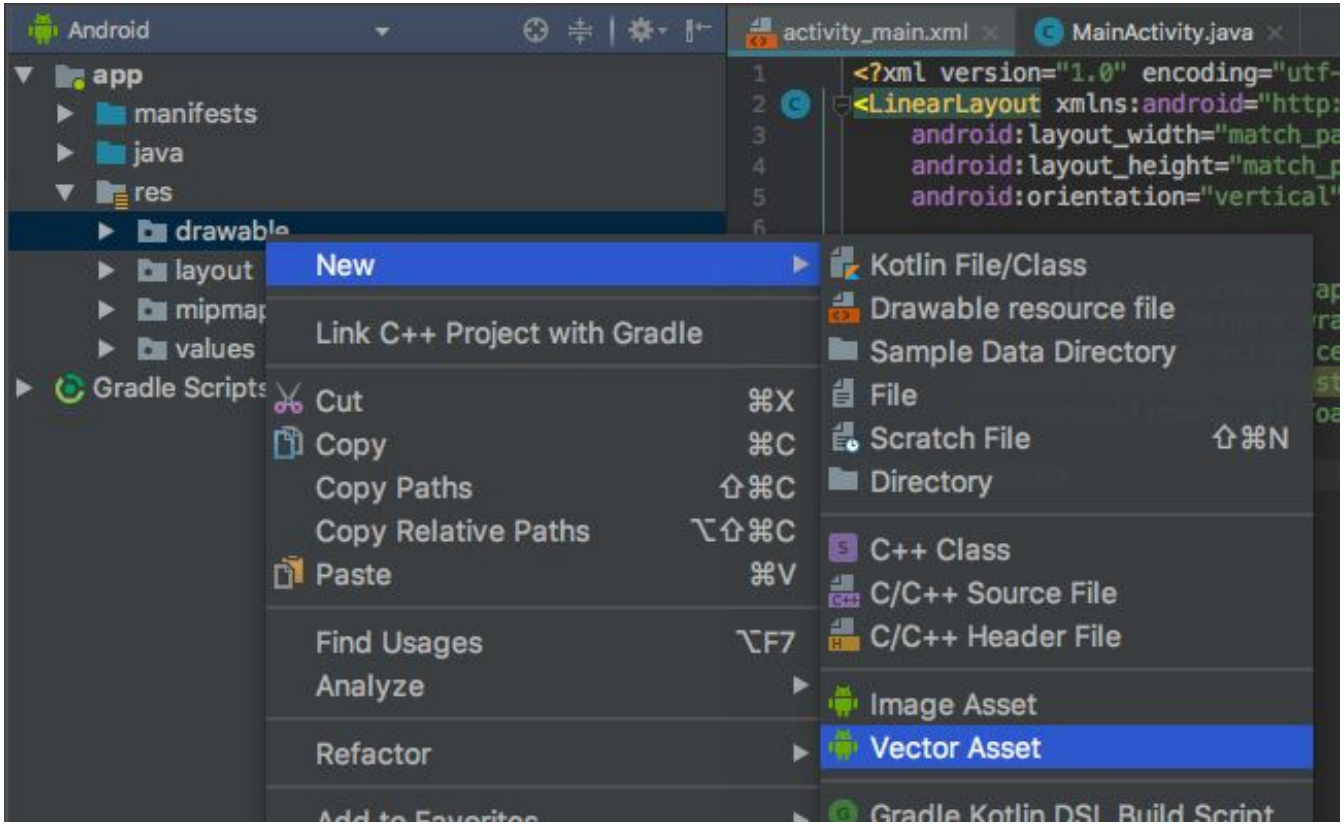


# Trabalhando com ActionBar Menu (*AppMenu*)



# Trabalhando com ActionBar Menu (*AppMenu*)

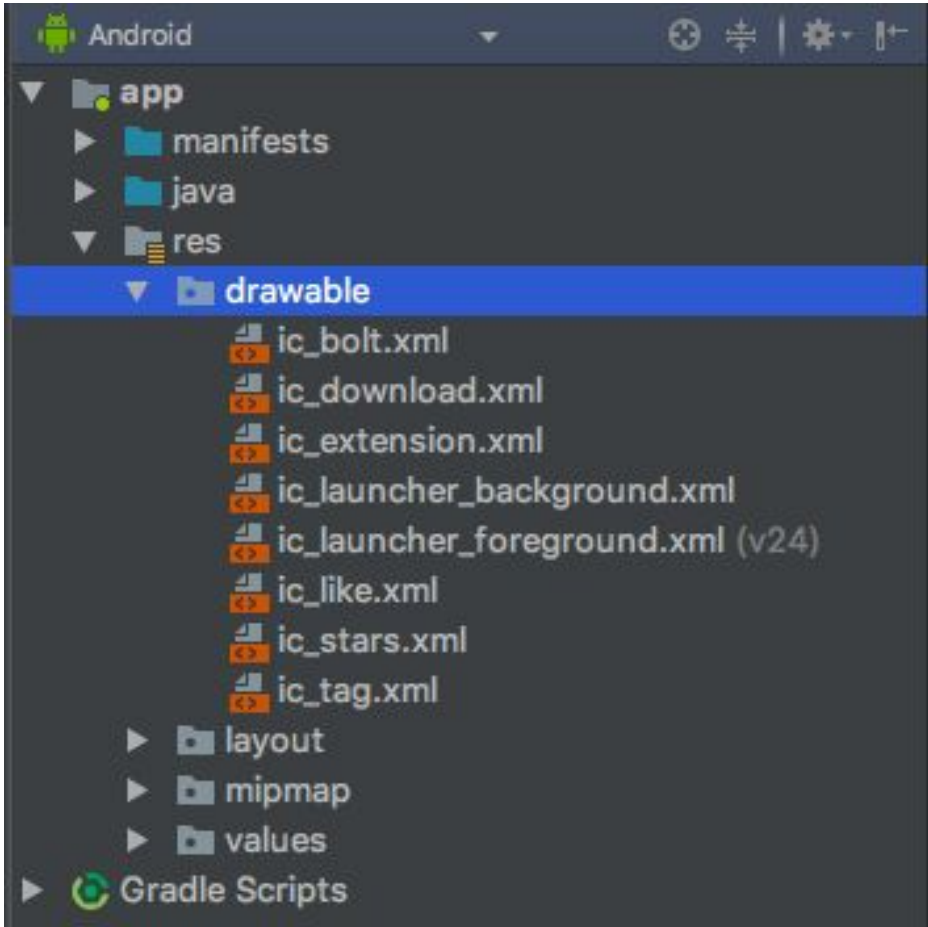
Importe as imagens como *Vector Asset* para o projeto AppMenu.



# Trabalhando com ActionBar Menu (*AppMenu*)

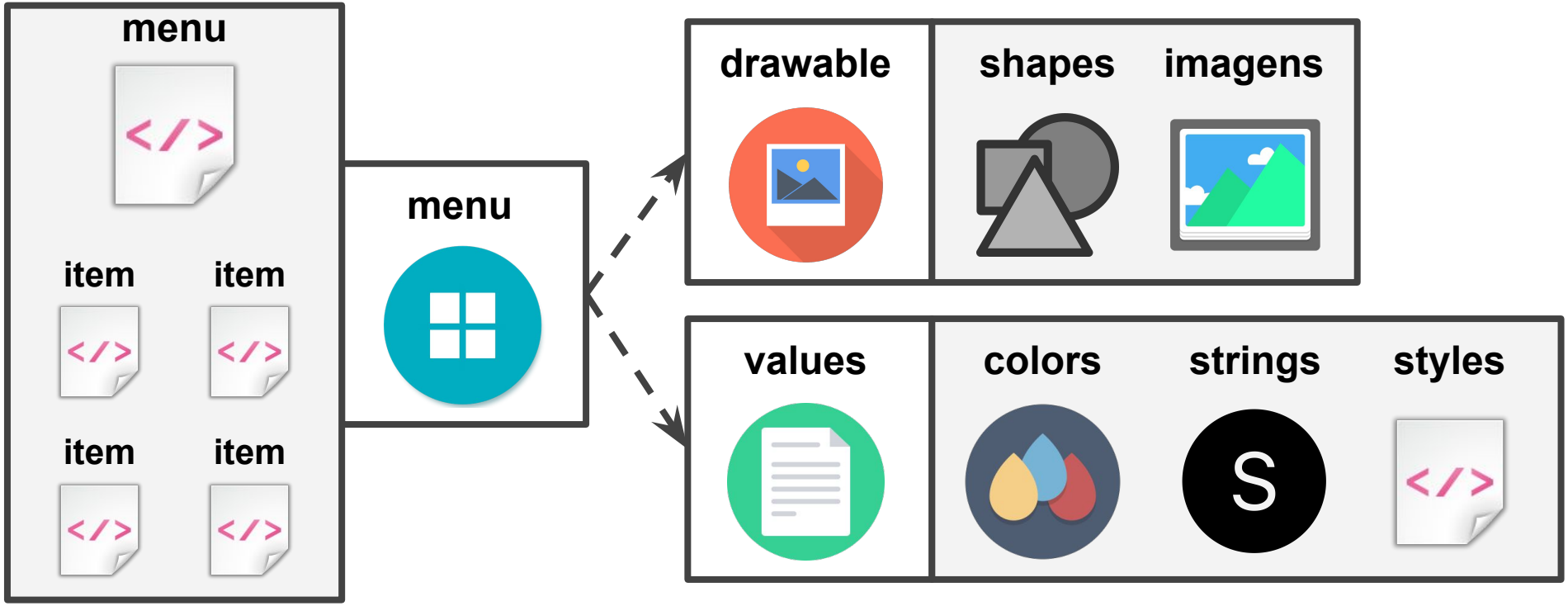
Ao final do processo, você deve ter os seguintes drawables:

- `ic_bolt.xml`
- `ic_download.xml`
- `ic_extension.xml`
- `ic_like.xml`
- `ic_starts.xml`
- `ic_tag.xml`



# Trabalhando com ActionBar Menu (*AppMenu*)

Um menu é um arquivo de Layout disposto na pasta `res/menu/` que contém umas série de tags `<item>`



# Trabalhando com ActionBar Menu (*AppMenu*)

Vamos começar definindo a string dos menus. Menus exibidos como ícones na ActionBar não exibem as labels. Por outro lado, menus exibidos no *Overflow Menu* não exibem as imagens. Vamos também definir as cores

strings.xml

Edit translations for all locales in the translations editor.

```
1   <resources>
2       <string name="app_name">AppMenu</string>
3
4       <string name="bolt">Flash</string>
5       <string name="download">Download</string>
6       <string name="plugin">Plugin</string>
7       <string name="like">Like</string>
8       <string name="stars">Importante</string>
9       <string name="tag">Marcador</string>
10  </resources>
```

colors.xml

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <resources>
3       <color name="colorPrimary">#252525</color>
4       <color name="colorPrimaryDark">#121212</color>
5       <color name="colorAccent">#4967ff</color>
6       <color name="branco">#FFFFFF</color>
7   </resources>
```

O Android que controla as cores do *ActionBar*. Para sobrescrever este comportamento, devemos criar um estilo e aplicá-lo no tema geral.

styles.xml

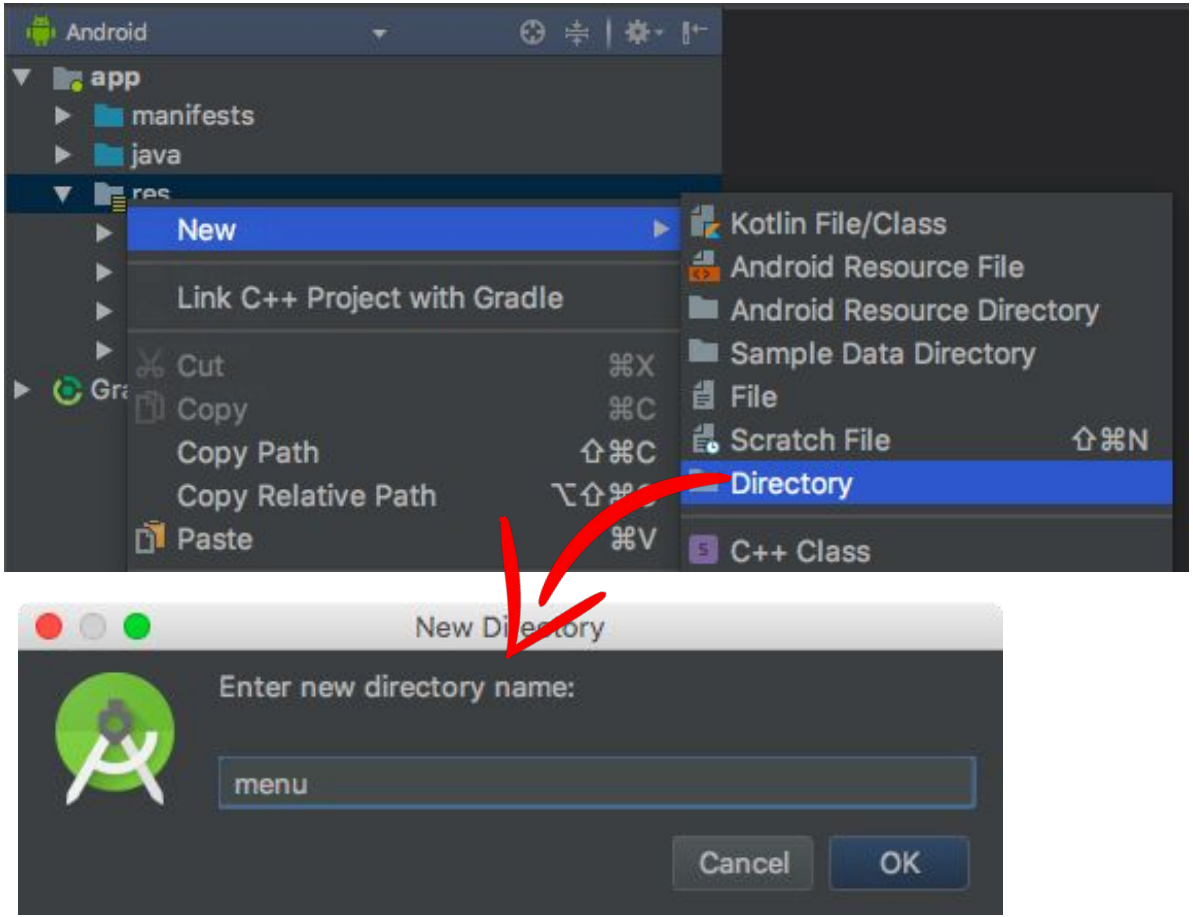
Edit all themes in the project in the theme editor.

```
1  <resources>
2      <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
3          <item name="colorPrimary">@color/colorPrimary</item>
4          <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
5          <item name="colorAccent">@color/colorAccent</item>
6
7          <item name="actionBarTheme">@style/ActionbarTheme</item>
8      </style>
9      <style name="ActionbarTheme">
10         <item name="tint">@color/branco</item>
11         <item name="iconTint">@color/branco</item>
12         <item name="titleTextColor">@color/branco</item>
13         <item name="actionMenuTextColor">@color/branco</item>
14     </style>
15 </resources>
```


- `tint` ⇒ cor do ícone (três pontos) do *Overflow Menu*
- `iconTint` ⇒ cor do ícone que é exibido no ActionBar
- `titleTextColor` ⇒ cor do título da ActionBar
- `actionMenuTextColor` ⇒ para exibição do app em orientação *landscape* e configuração para exibição do título e do ícone (ao mesmo tempo), define a cor do nome do item que aparece sobre a ActionBar

```
10  <item name="tint">@color/branco</item>
11  <item name="iconTint">@color/branco</item>
12  <item name="titleTextColor">@color/branco</item>
13  <item name="actionMenuTextColor">@color/branco</item>
14      </style>
15  </resources>
```

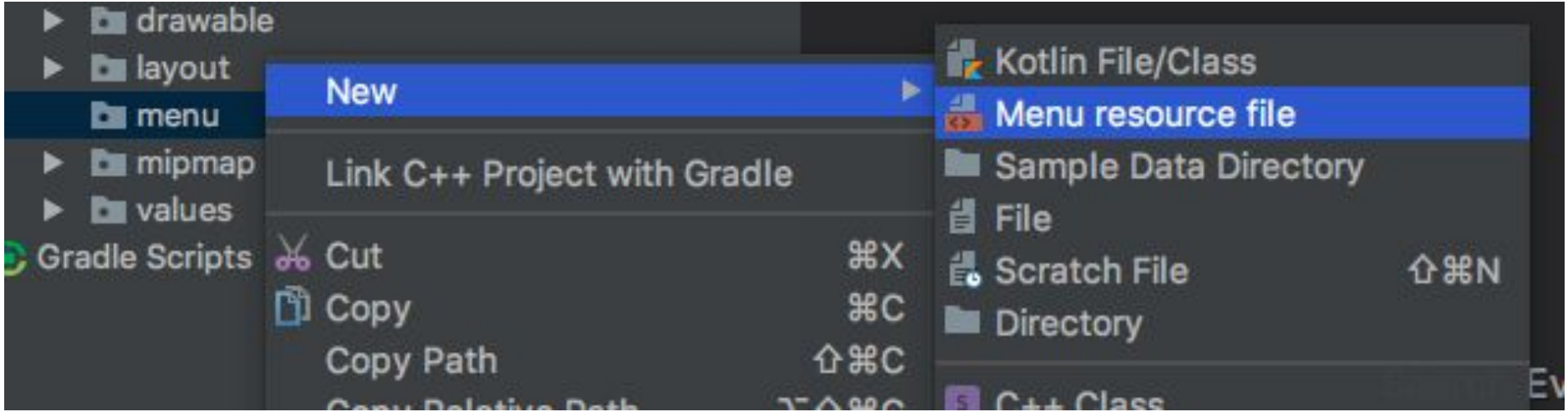
# Trabalhando com ActionBar Menu (*AppMenu*)



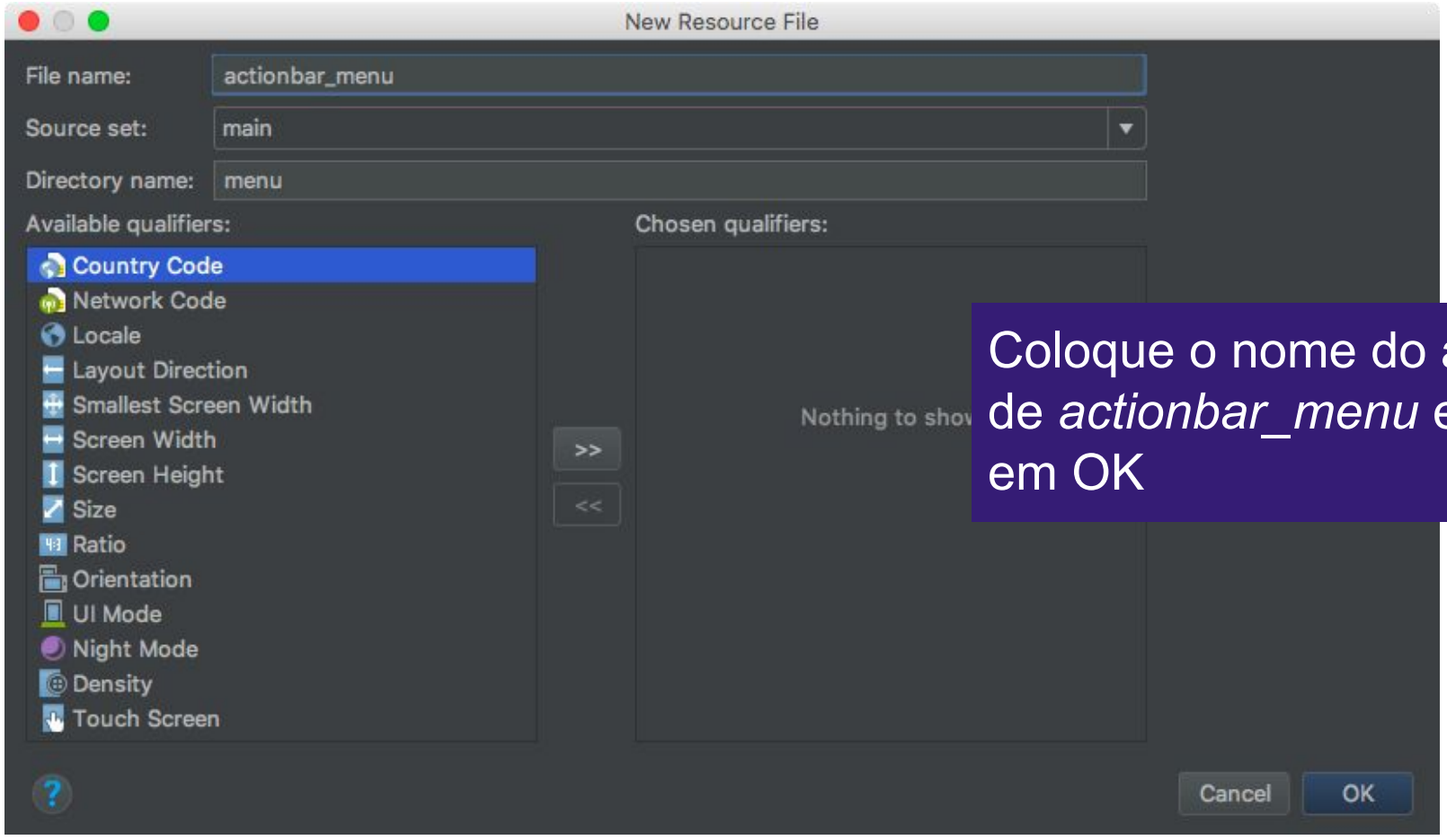

Algumas versões do Android Studio já criam o projeto com a pasta `res/menu/` por padrão. Caso esta pasta não exista, crie-a clicando com o botão direito na pasta `res`, menu *New*, *Directory*

# Trabalhando com ActionBar Menu (*AppMenu*)

Agora clique com o botão direito na pasta `res/menu/`, selecione a opção *New*, submenu *Menu resource file*



# Trabalhando com ActionBar Menu (*AppMenu*)




Coloque o nome do arquivo de *actionbar_menu* e clique em OK

# Trabalhando com ActionBar Menu (*AppMenu*)

Vejamos um código com um único item de menu

actionbar_menu.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item
        android:icon="@drawable/ic_bolt"
        android:title="Flash"
        app:showAsAction="ifRoom"
        android:id="@+id/menu_bolt" />

</menu>
```

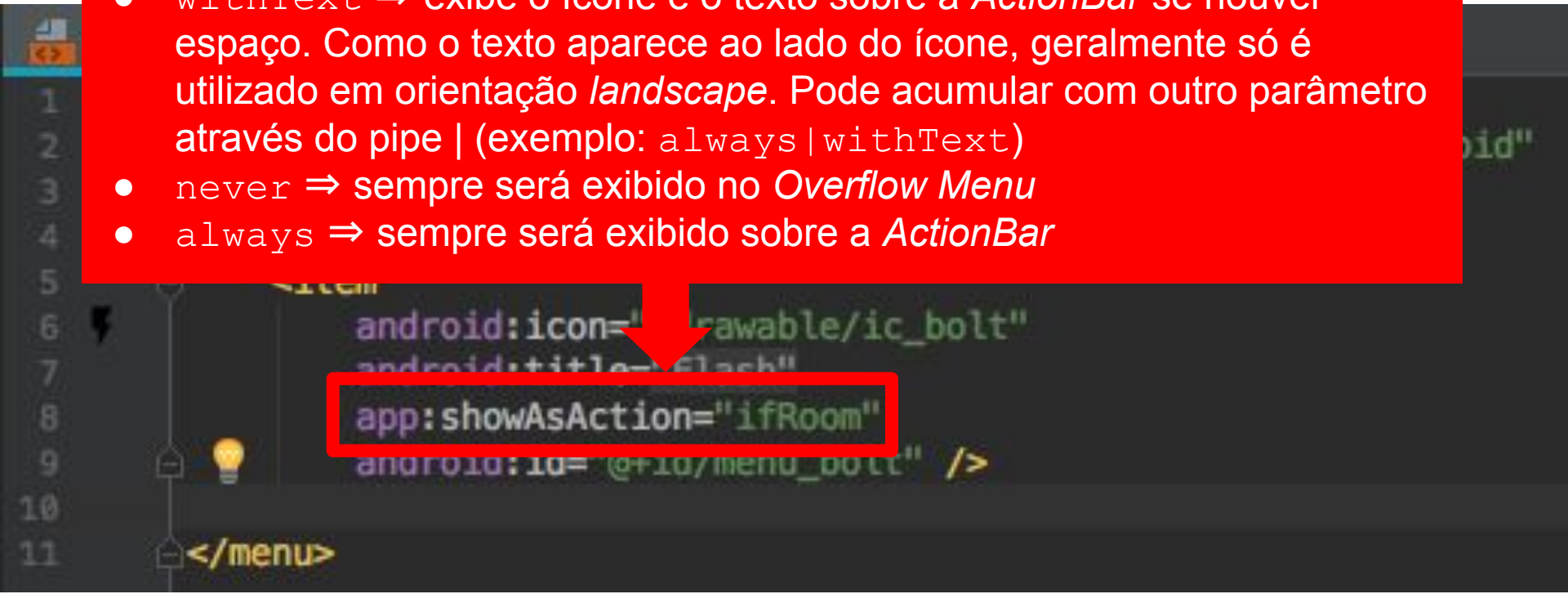
# Trabalhando com ActionBar Menu (*AppMenu*)

- `android:icon` ⇒ Aponta para o drawable
- `android:title` ⇒ Define o nome do menu (para o caso de exibição dentro do *Overflow Menu*)
- `android:id` ⇒ ID do item. Será necessário para verificar qual menu foi clicado.

```
1
2                                                    .com/apk/res/android"
3       xmlns:app="http://schemas.android.com/apk/res-auto">
4
5       <item
6           android:icon="@drawable/ic_bolt"
7           android:title="Flash"
8           app:showAsAction="ifRoom"
9           android:id="@+id/menu_bolt" />
10
11  </menu>
```

# Trabalhando com ActionBar Menu (*AppMenu*)

- `ifRoom` ⇒ exibe sobre a *ActionBar* se houver espaço
- `withText` ⇒ exibe o ícone e o texto sobre a *ActionBar* se houver espaço. Como o texto aparece ao lado do ícone, geralmente só é utilizado em orientação *landscape*. Pode acumular com outro parâmetro através do pipe | (exemplo: `always|withText`)
- `never` ⇒ sempre será exibido no *Overflow Menu*
- `always` ⇒ sempre será exibido sobre a *ActionBar*



actionbar_menu.xml

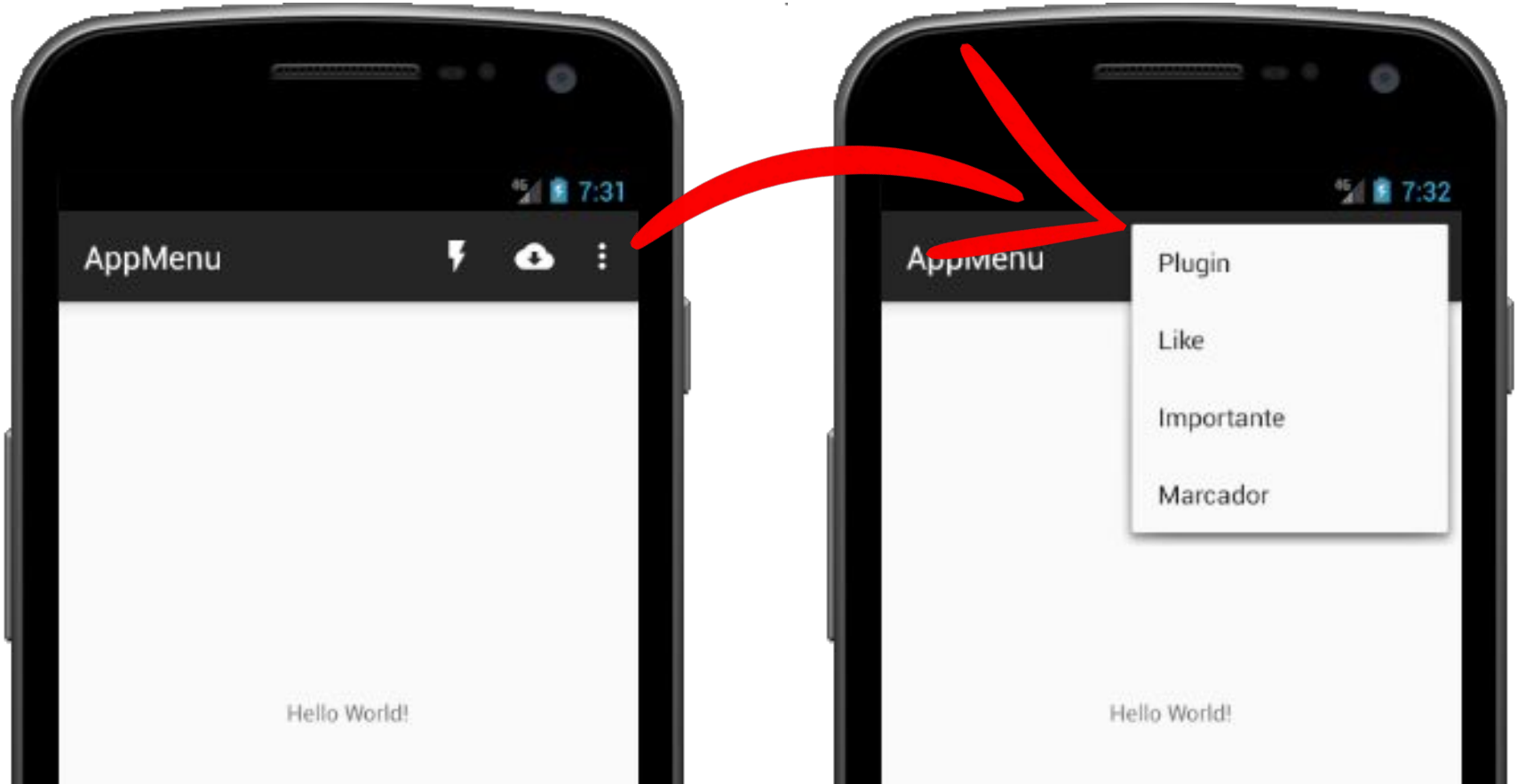
```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:icon="@drawable/ic_bolt" android:title="@string/bolt"
        app:showAsAction="ifRoom" android:id="@+id/menu_bolt" />
    <item
        android:icon="@drawable/ic_download" android:title="@string/download"
        app:showAsAction="ifRoom" android:id="@+id/menu_download" />
    <item
        android:icon="@drawable/ic_extension" android:title="@string/plugin"
        app:showAsAction="ifRoom" android:id="@+id/menu_plugin" />
    <item
        android:icon="@drawable/ic_like" android:title="@string/like"
        app:showAsAction="ifRoom" android:id="@+id/menu_like" />
    <item
        android:icon="@drawable/ic_stars" android:title="@string/stars"
        app:showAsAction="ifRoom" android:id="@+id/menu_stars" />
    <item
        android:icon="@drawable/ic_tag" android:title="@string/tag"
        app:showAsAction="ifRoom" android:id="@+id/menu_tag" />
</menu>
```

# Trabalhando com ActionBar Menu (*AppMenu*)

Agora que já temos o menu pronto, vamos inflá-lo na *Activity* através do método *onCreateOptionsMenu()*

```
public class MainActivity extends AppCompatActivity {

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.actionbar_menu,menu);
        return super.onCreateOptionsMenu(menu);
    }

}
```

# Trabalhando com ActionBar Menu (*AppMenu*)

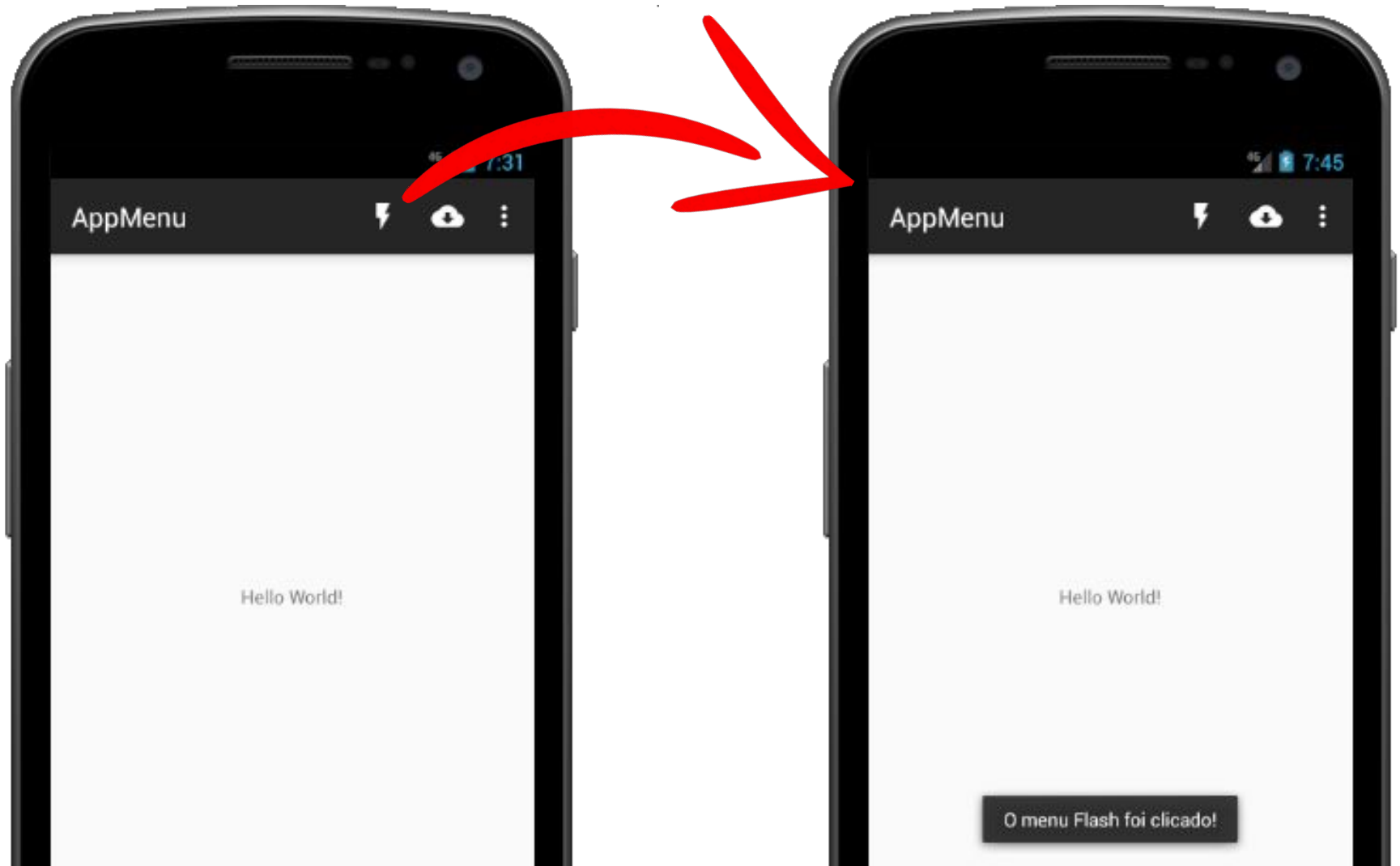


# Trabalhando com ActionBar Menu (*AppMenu*)

Para escutar o click nos itens de menu, precisamos apenas implementar o método *onOptionsItemSelected()*. Como este método recebe o *item* clicado, podemos simplesmente comparar o ID do item com o ID das tags <*item*> do XML e executar a ação desejada.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_bolt:
            Toast.makeText( context: this,  text: "Flash clicado!", Toast.LENGTH_SHORT).show();
            break;
    }
    return super.onOptionsItemSelected(item);
}
```

```java
public class MainActivity extends AppCompatActivity {

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.actionbar_menu,menu);
        return super.onCreateOptionsMenu(menu);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.menu_bolt: exibirToast( menu: "Flash"); break;
            case R.id.menu_download: exibirToast( menu: "Download"); break;
            case R.id.menu_plugin: exibirToast( menu: "Plugin"); break;
            case R.id.menu_like: exibirToast( menu: "Like"); break;
            case R.id.menu_stars: exibirToast( menu: "Importante"); break;
            case R.id.menu_tag: exibirToast( menu: "Marcador"); break;
        }
        return super.onOptionsItemSelected(item);
    }

    private void exibirToast(String menu) {
        Toast.makeText( context: this,  text: "O menu " + menu + " foi clicado!", Toast.LENGTH_SHORT).show();
    }
}
```

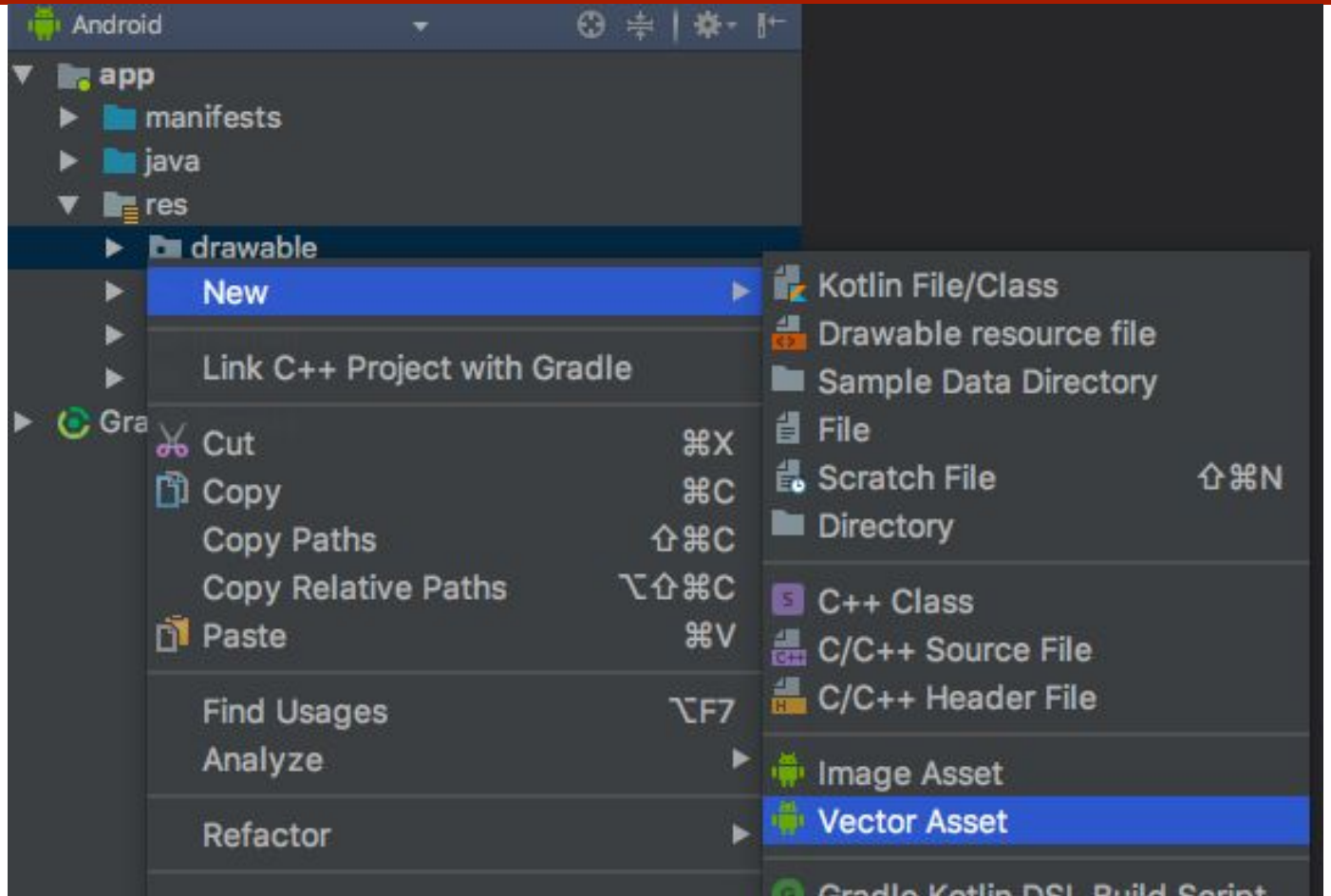

7:31
AppMenu
Hello World!
7:45
AppMenu
Hello World!
O menu Flash foi clicado!

# Exercício em Sala

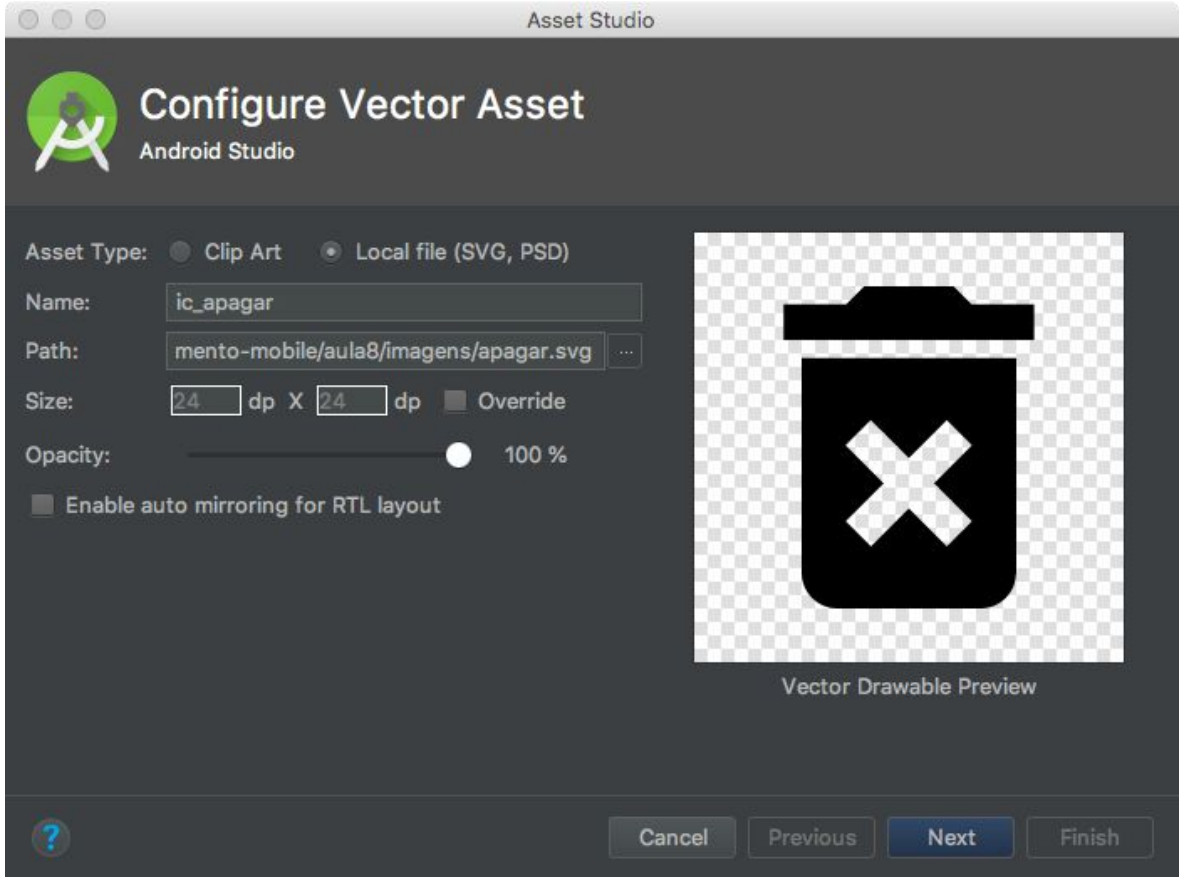
Altere o app *Notepad* observando os seguintes detalhes:

- Remova o botão "LIMPAR"
- Adicione um *ActionBar Menu* com um ícone semelhante ao da figura.
- Quando o usuário clicar neste ícone, o app deve exibir o *SnackBar* para confirmar a exclusão do texto.

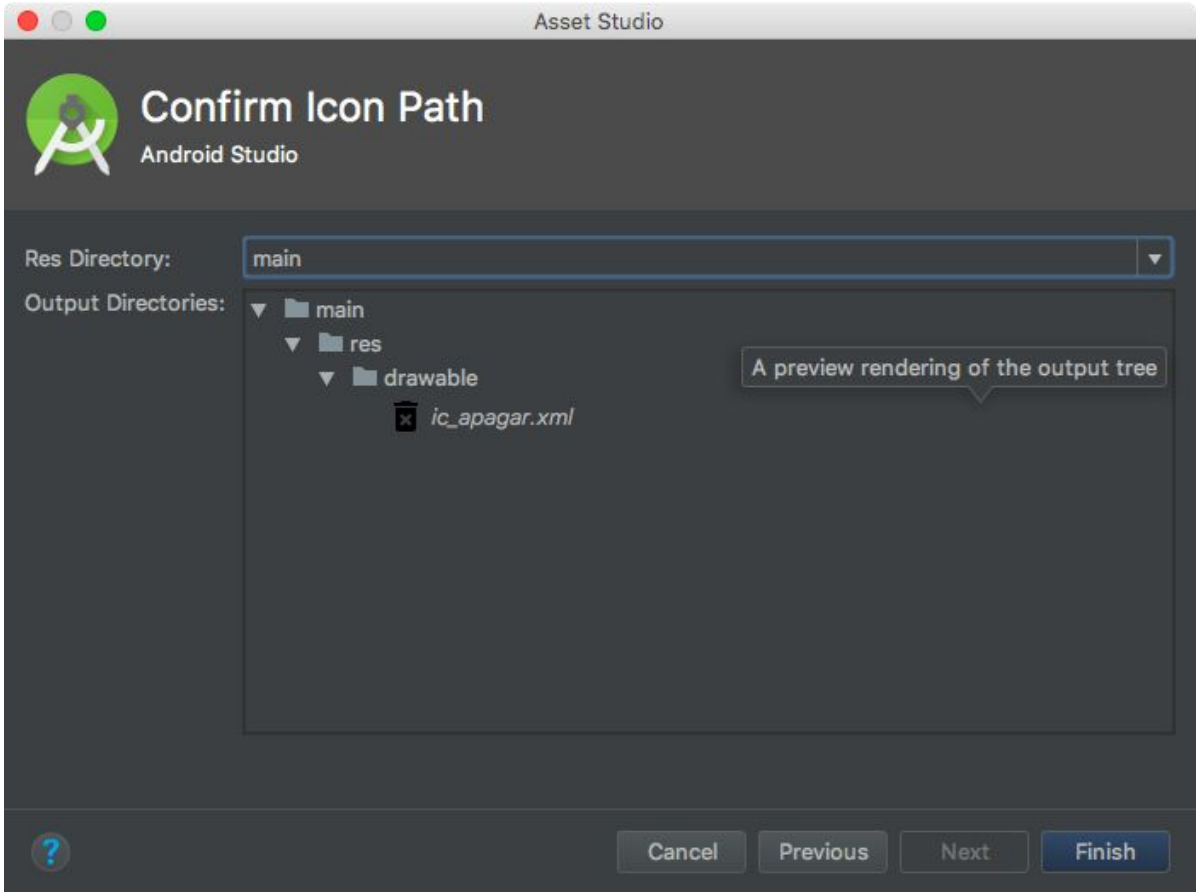
# Resolvendo o Exercício



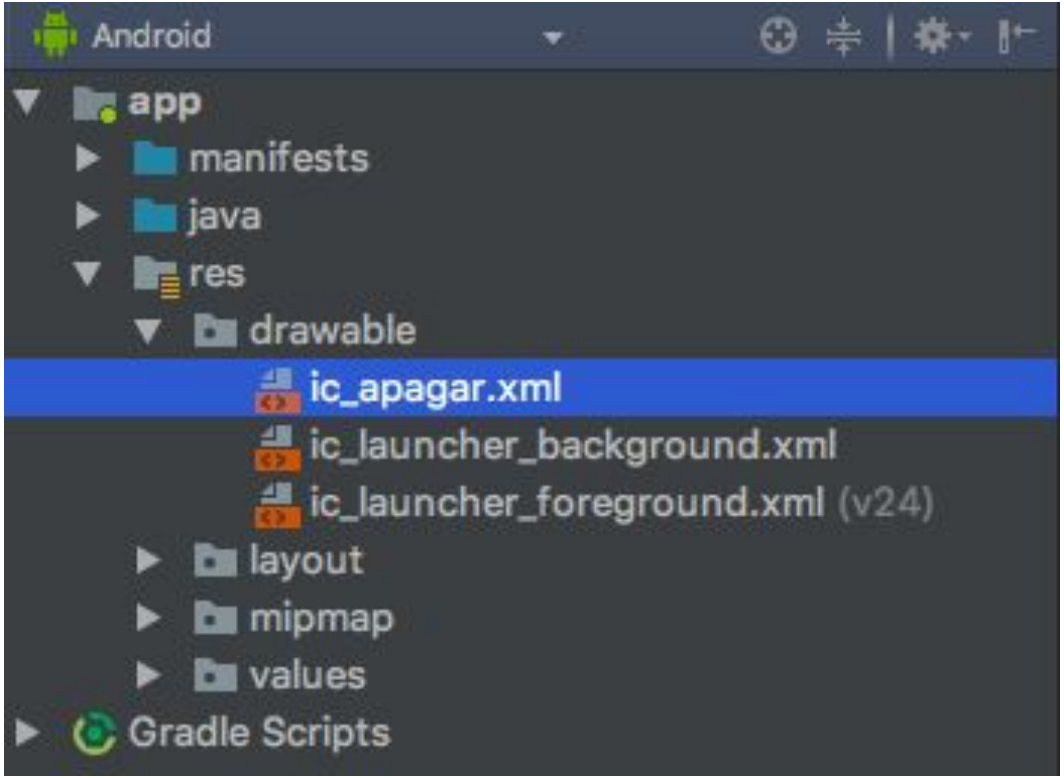
# Resolvendo o Exercício



# Resolvendo o Exercício



# Resolvendo o Exercício



# Resolvendo o Exercício

colors.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>
    <color name="cinza">#efefef</color>
    <color name="branco">#ffffff</color>
</resources>
```

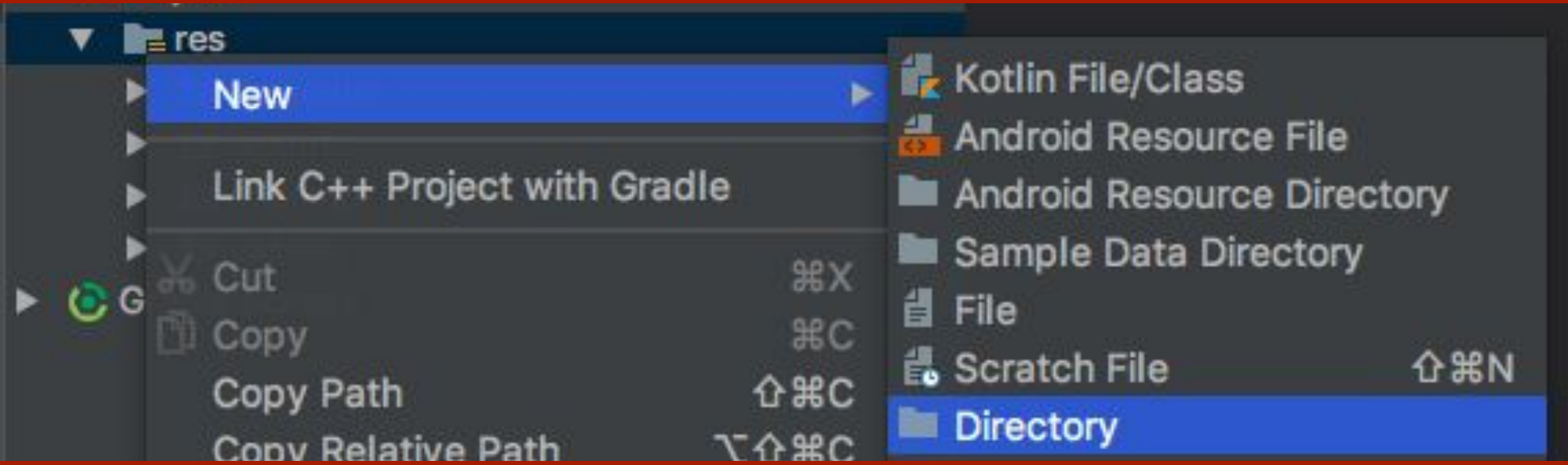
# Resolvendo o Exercício

styles.xml

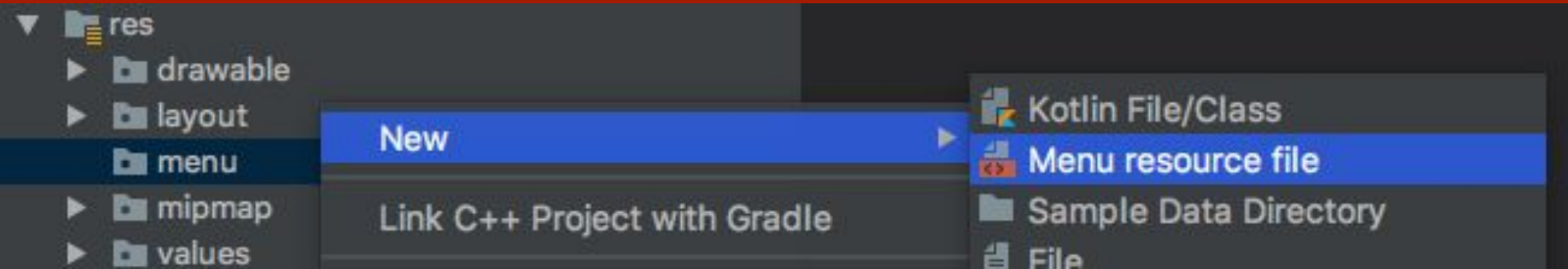
Edit all themes in the project in the theme editor.

```
1  <resources>
2      <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
3          <item name="colorPrimary">@color/colorPrimary</item>
4          <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
5          <item name="colorAccent">@color/colorAccent</item>
6          <item name="actionBarTheme">@style/cores_menu</item>
7      </style>
8      <style name="cores_menu">
9          <item name="tint">#ffffff</item>
10         <item name="iconTint">#ffffff</item>
11         <item name="titleTextColor">#ffffff</item>
12         <item name="actionMenuTextColor">#ffffff</item>
13     </style>
14 </resources>
15
```

## Criando o diretório menu




## Criando o arquivo de menu



# Resolvendo o Exercício

layout_menu.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:icon="@drawable/ic_apagar"
        android:title="Apagar"
        app:showAsAction="always"
        android:id="@+id/menu_apagar" />
</menu>
```

# Ajustando a *MainActivity*

Podemos criar um método *apagarTexto()* para centralizar a chamada ao *SnackBar*. O método *onCreate()* servirá apenas para obter os *Views* do XML.

```
private CoordinatorLayout layout;
private EditText editText;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    layout = (CoordinatorLayout) findViewById(R.id.coordinator_layout);
    editText = (EditText) findViewById(R.id.edit_text);
}

private void apagarTexto() {
    Snackbar snackbar = Snackbar.make(layout, R.string.mensagem_confirmacao, Snackbar.LENGTH_LONG);
    snackbar.setAction(R.string.snackbar_action, new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            editText.setText("");
        }
    });
    snackbar.show();
}
```

# Ajustando a *MainActivity*

Devemos também implementar o método *onCreateOptionsMenu()* para inflar o XML do menu, e o método *onOptionsItemSelected()* para escutar o clique do usuário no item de menu.

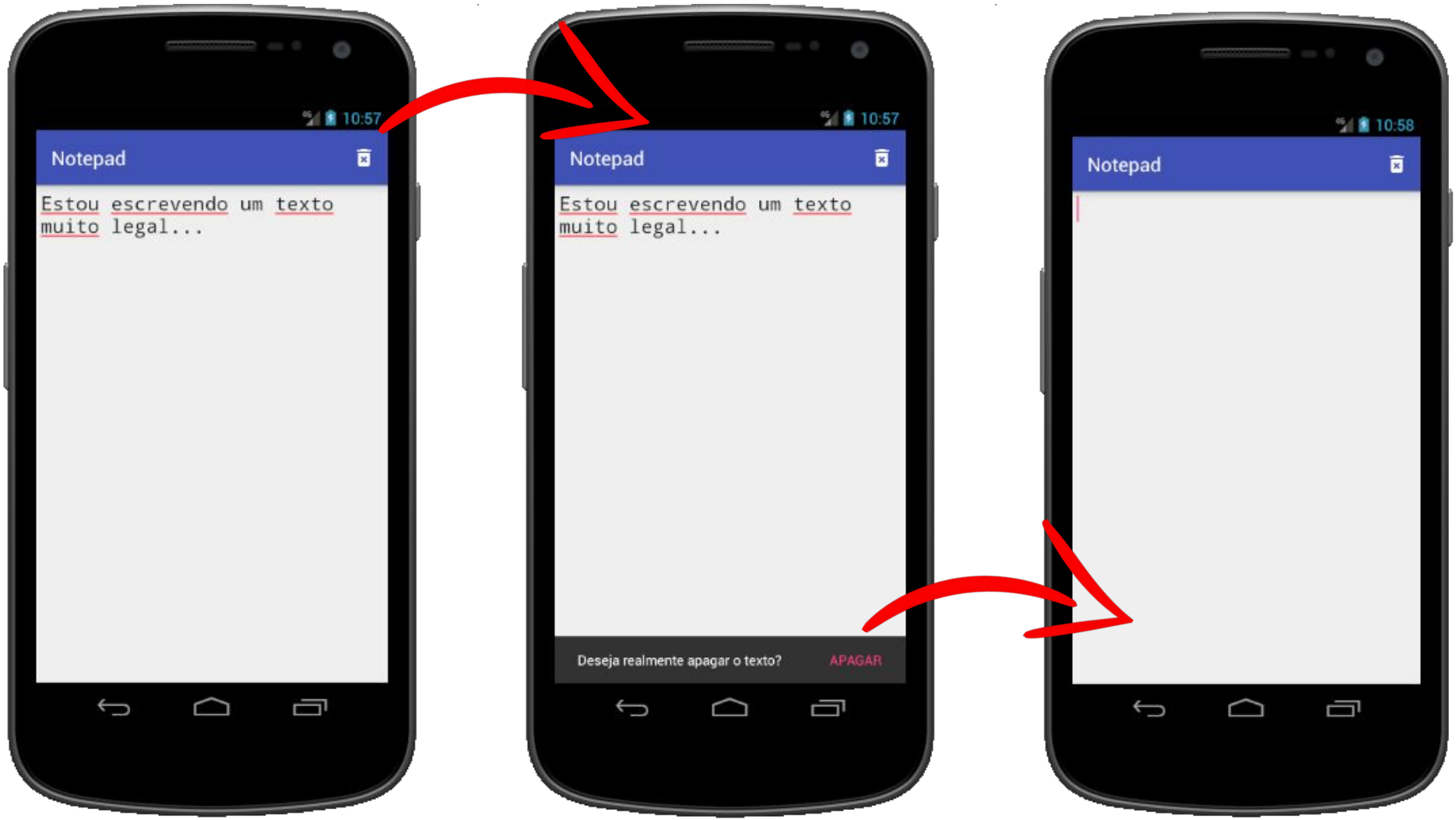
```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_apagar:
            apagarTexto();
            break;
    }
    return super.onOptionsItemSelected(item);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.layout_menu,menu);
    return super.onCreateOptionsMenu(menu);
}
```

# Ajustando a *activity_main.xml*

Por fim, vamos remover o botão do XML principal.

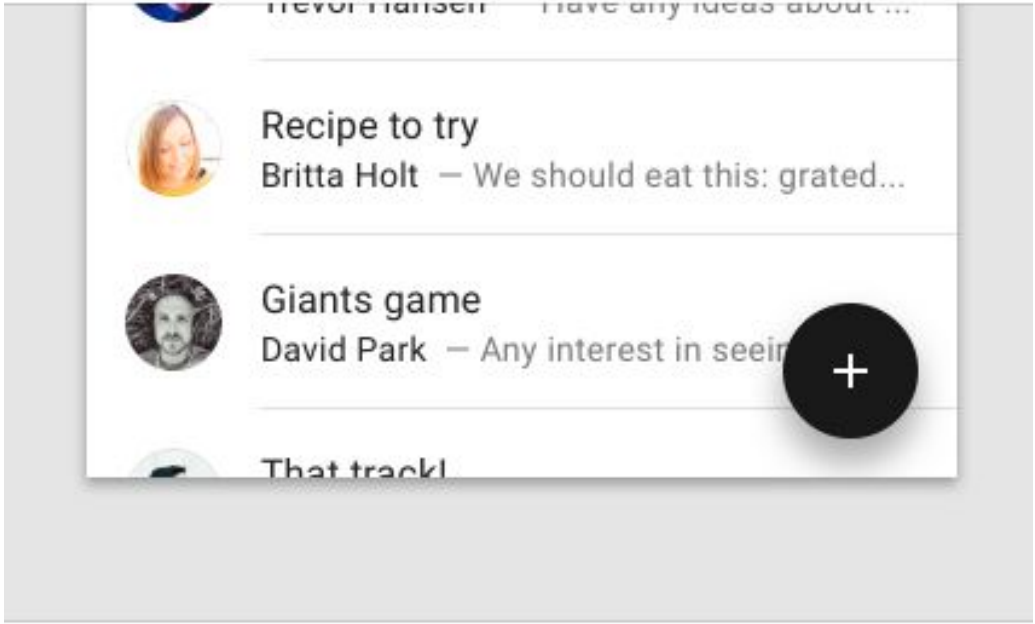
activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/coordinator_layout">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">
        <EditText
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:padding="5dp"
            android:background="@color/cinza"
            android:textSize="20sp"
            android:fontFamily="monospace"
            android:gravity="top|left"
            android:scrollbars="vertical"
            android:id="@+id/edit_text" />
    </LinearLayout>
</android.support.design.widget.CoordinatorLayout>
```


10:57
Notepad
Estou escrevendo um texto muito legal...
10:57
Notepad
Estou escrevendo um texto muito legal...
Deseja realmente apagar o texto?
APAGAR
10:58
Notepad

# Floating Action Button

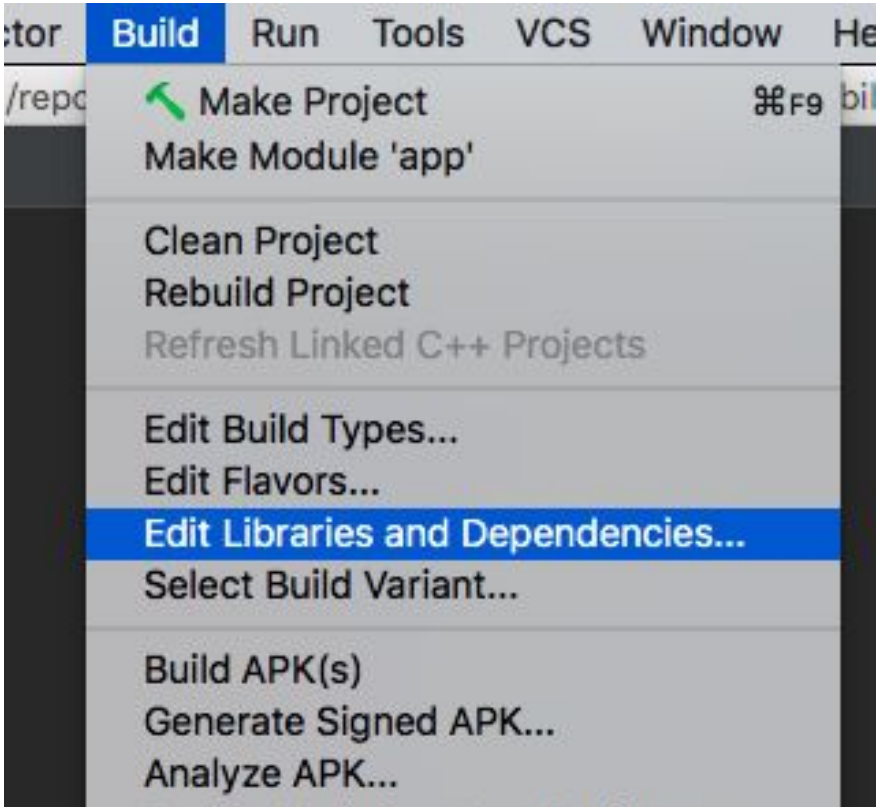
Vamos agora criar o Floating Action Button (FAB). Basicamente, um FAB é um botão totalmente arredondado, geralmente posicionado no canto inferior à direita, e que fica sobre todos os demais componentes da *Activity* (com exceção do *Toast* e *SnackBar*).
https://material.io/design/components/buttons-floating-action-button.html




Antigamente (antes de 2015), para criar um FAB precisávamos criar um *shape* arredondado para um botão com `elevation` de `6dp`. Ele provocava este mesmo efeito (sombreado) que hoje está disponível pela ***Design Support Library***.

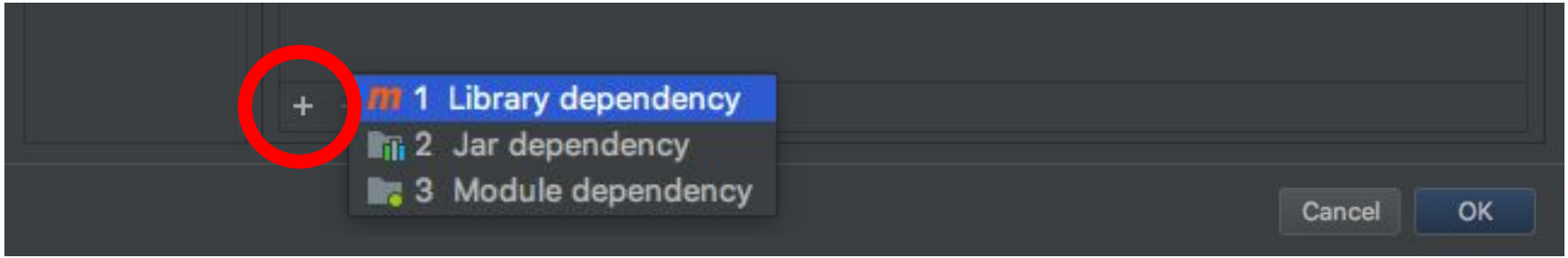
# Floating Action Button

Ainda no projeto *AppMenu*, vamos importar a ***Design Support Library***.

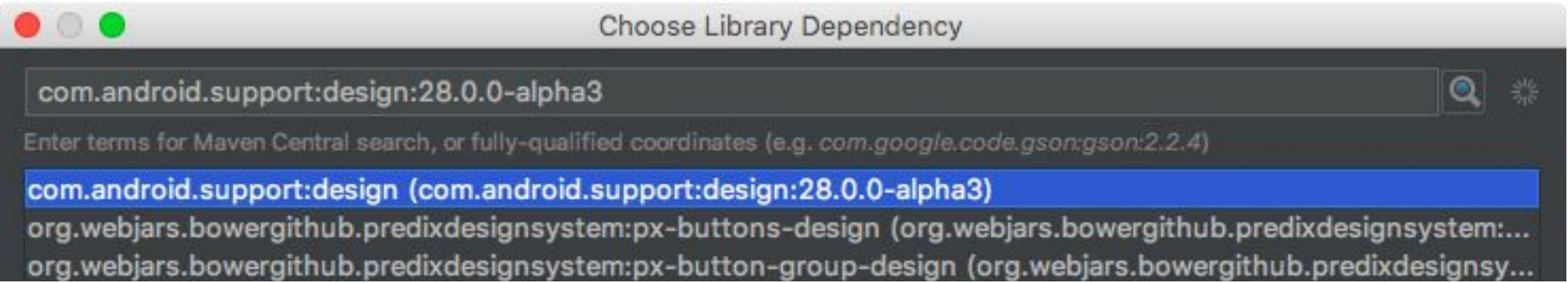


# Floating Action Button

Clique na opção *Library dependency*

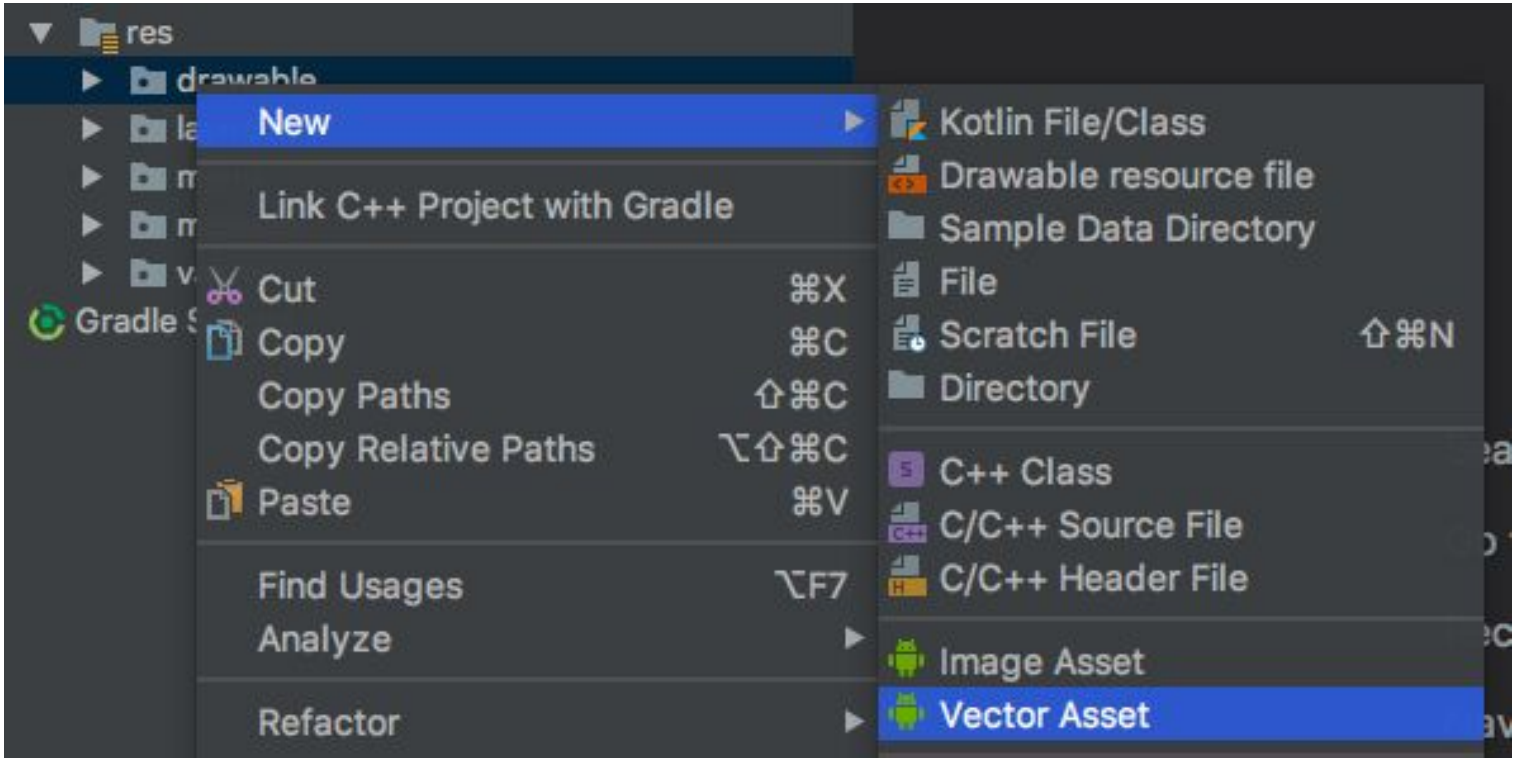



Faça uma busca pela palavra *Design* e selecione o item a seguir:



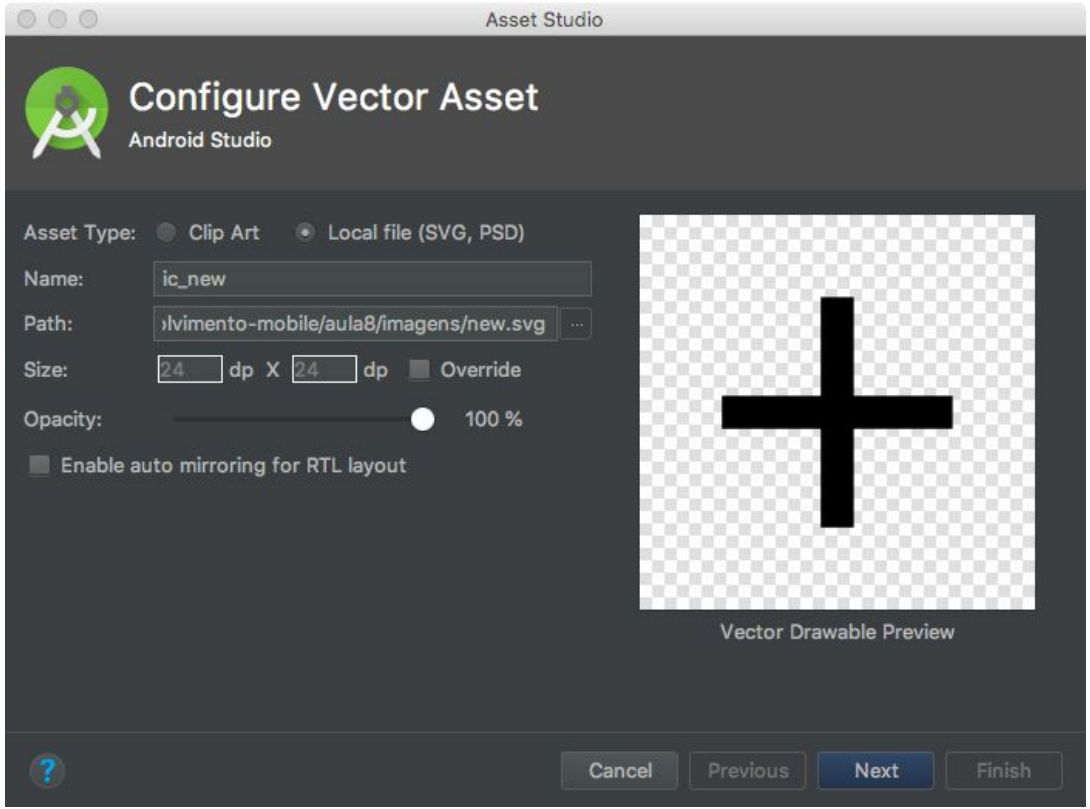
# Floating Action Button

Vamos agora importar um ícone para colocar dentro do FAB



# Floating Action Button

Vamos agora importar um ícone para colocar dentro do FAB



# Floating Action Button

A inclusão do FAB no Layout é muito simples! Altere o layout para *RelativeLayout* para posicionar o FAB corretamente.

activity_main.xml

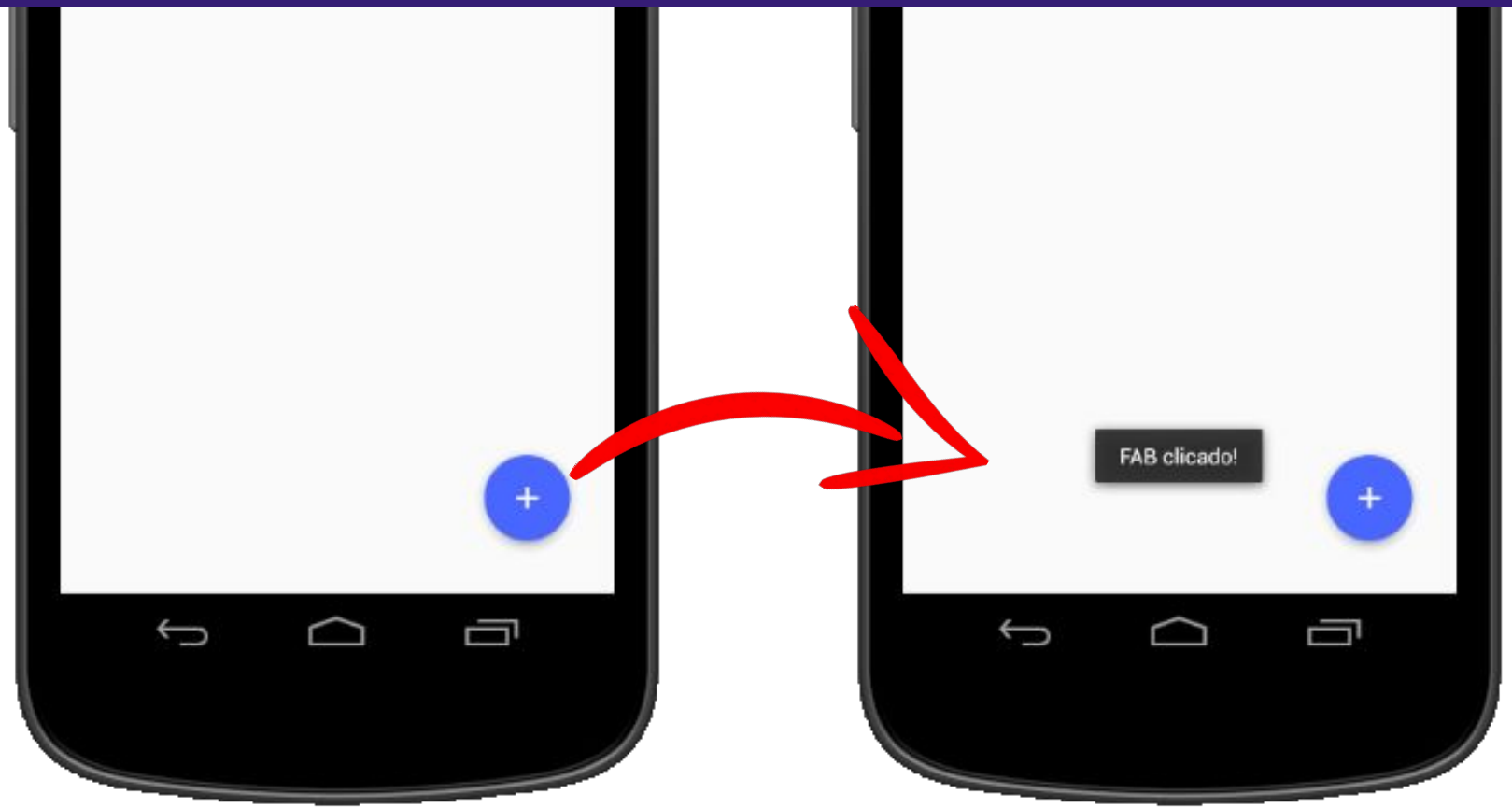
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.design.widget.FloatingActionButton
        android:id="@+id/fab_novo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:src="@drawable/ic_new"
        android:tint="@color/branco"
        android:layout_margin="16dp" />

</RelativeLayout>
```

# Floating Action Button

Por fim, no método *onCreate()* podemos obtê-lo através do *findViewById()* e adicionar um *OnClickListener* apropriado.

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

    FloatingActionButton fab = findViewById(R.id.fab_novo);
    fab.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Toast.makeText( context: MainActivity.this,  text: "FAB clicado!", Toast.LENGTH_LONG).show();
        }
    });
}
```

# Floating Action Button



# Icon Launcher

**Orientação oficial (Android):**

- 512 x 512 pixels. Na prática, eles serão menores no Android, porém ao publicar no Google Play, você precisará de um ícone deste tamanho.
- Separar entre *background* e *foreground*. Isso é último para auxiliar os efeitos dos diferentes dispositivos.

**Dicas Extras:**

- 1024 x 1024 pixels, para aproveitar uma possível publicação na web
- Mantenha um ícone completo (*background* e *foreground*) para facilitar a publicação no Google Play
- Evite ícones arredondados ou com formatos diferenciados. Isso pode dificultar a adaptação do ícone em vários dispositivos. **O ideal é usar ícones quadrados.**

# Icon Launcher

Vamos começar excluindo os ícones que já existem no projeto. Na prática, os ícones são salvos nas pastas `mipmap`, organizadas por densidade.

**mipmap-mdpi**

**mipmap-hdpi**

**mipmap-xhdpi**

**mipmap-xxhdpi**

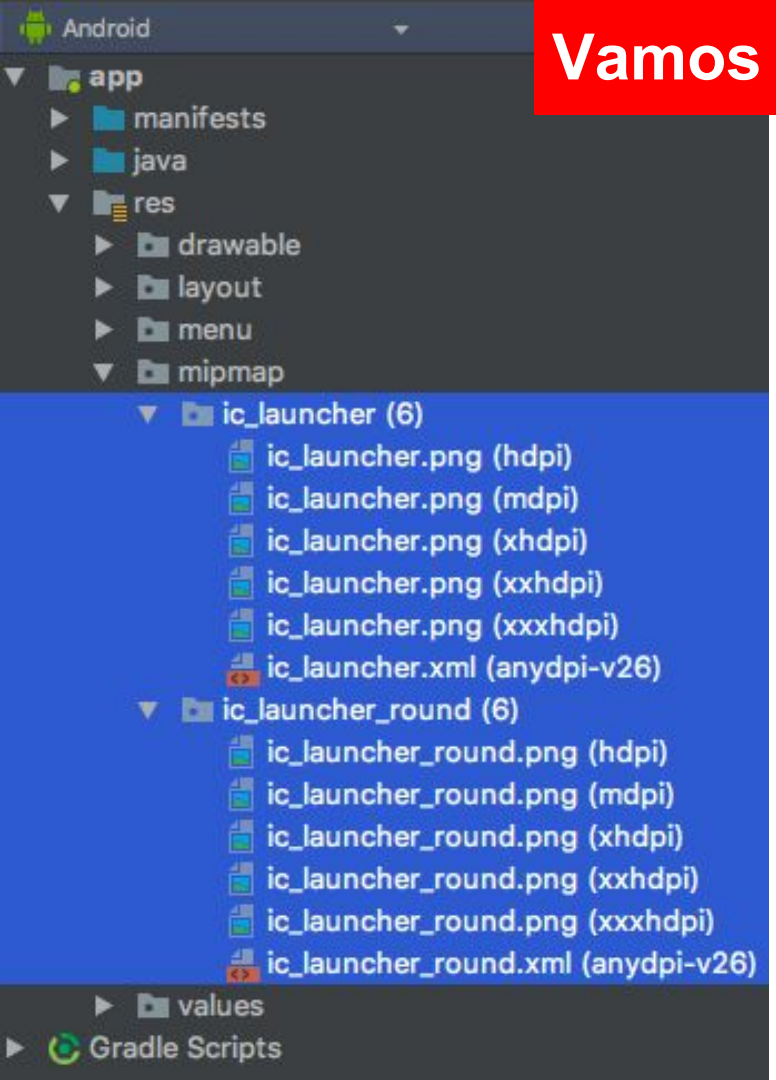
**mipmap-xxxhdpi**

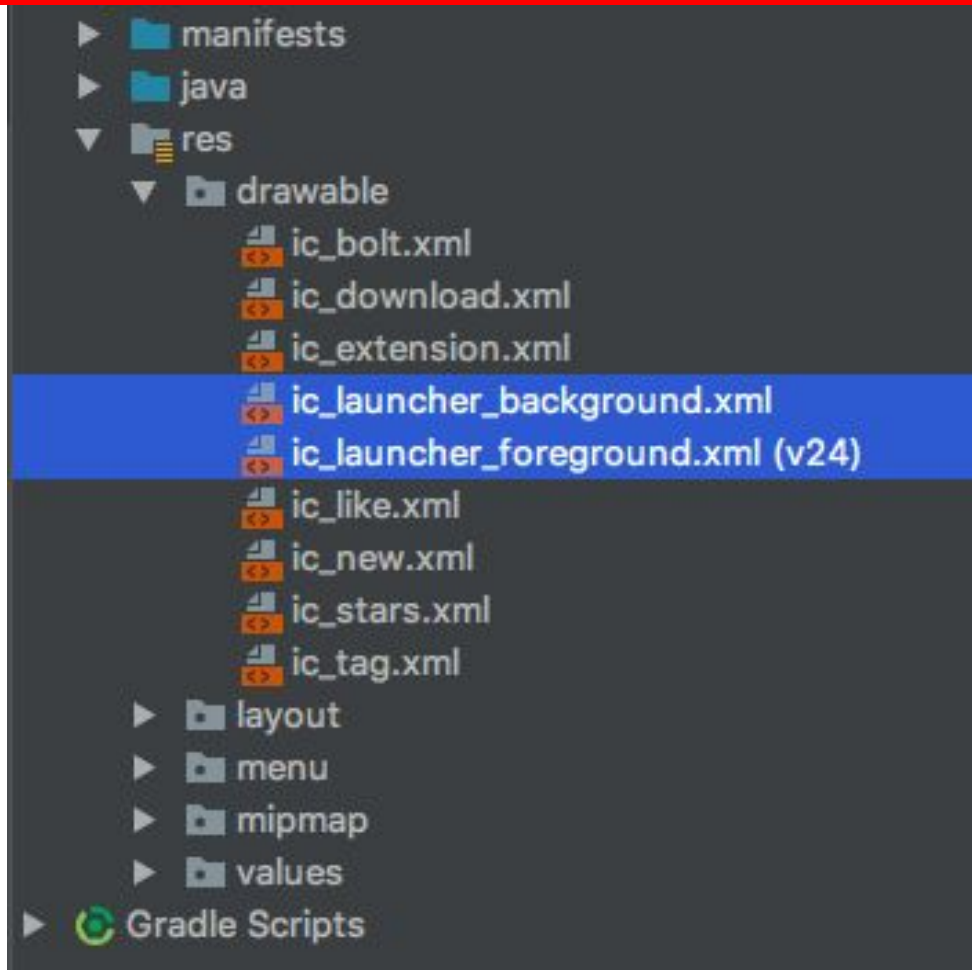
Além da densidade, os ícones também são organizados por ícones normais e arredondados (para do Android 7.1).

**ic_launcher**

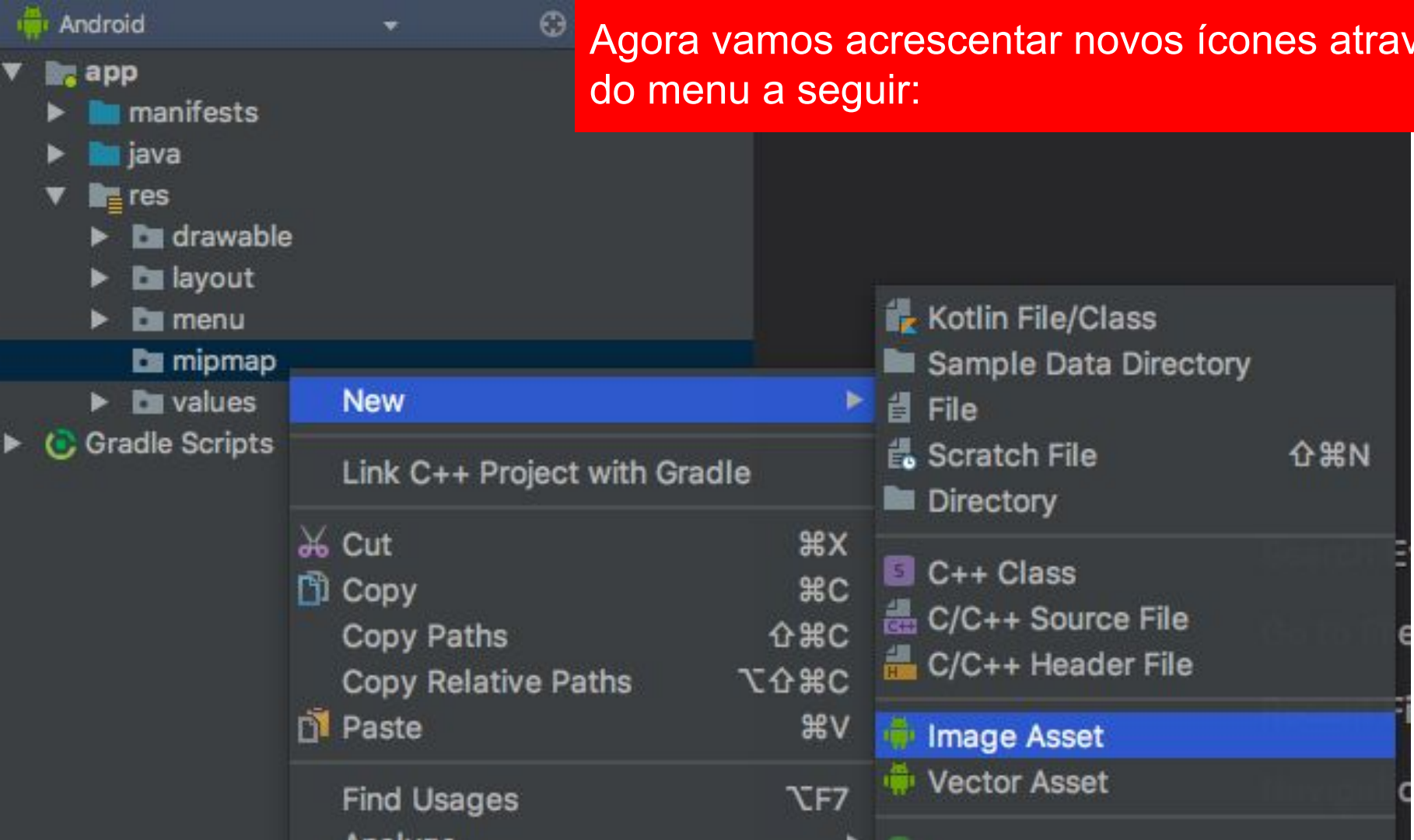
**ic_launcher_round**

# Vamos apagar os seguintes arquivos:



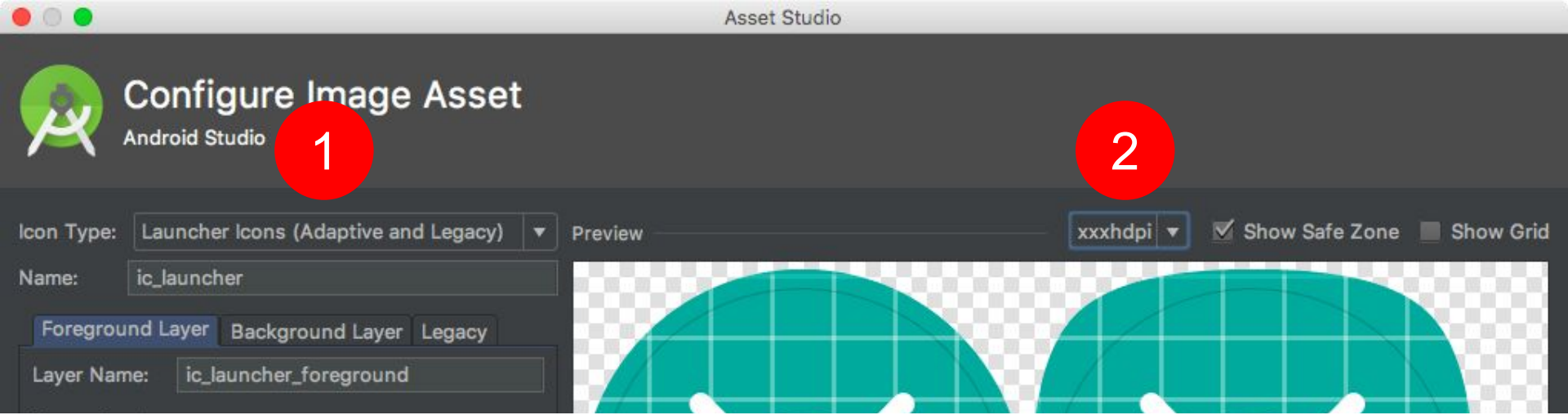
Agora vamos acrescentar novos ícones através do menu a seguir:



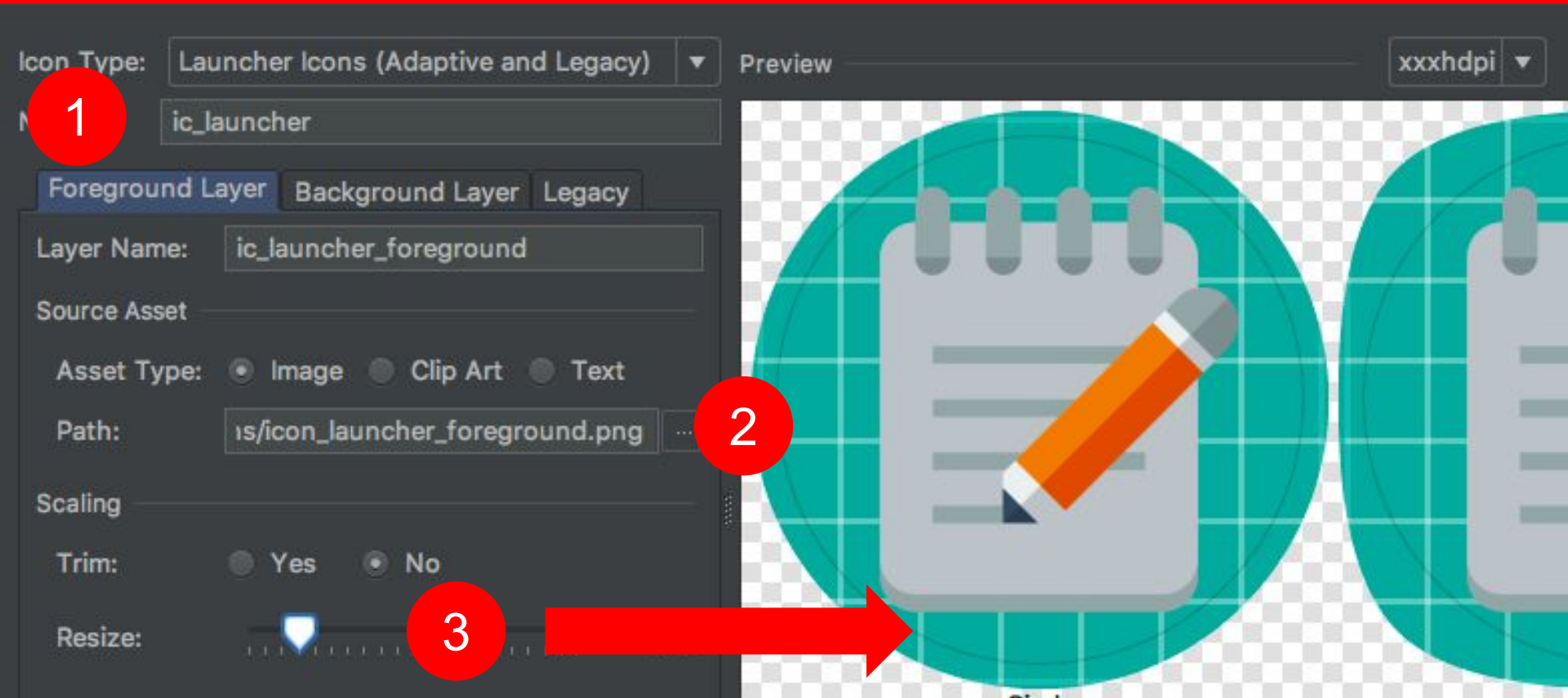
Selecione as seguintes opções:

1- **Icon Type:** Launcher Icons (Adaptive and Legacy)

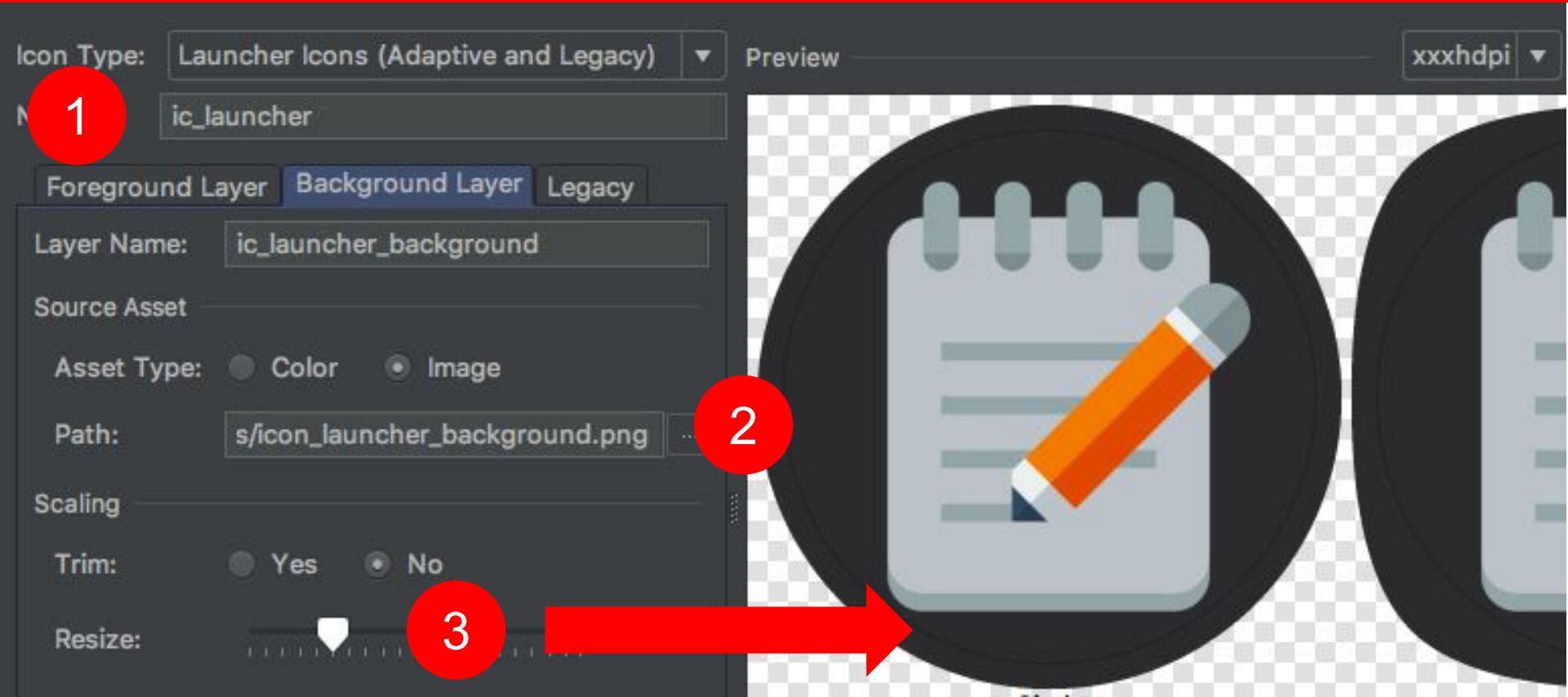
2- **Densidade:** xxxhdpi



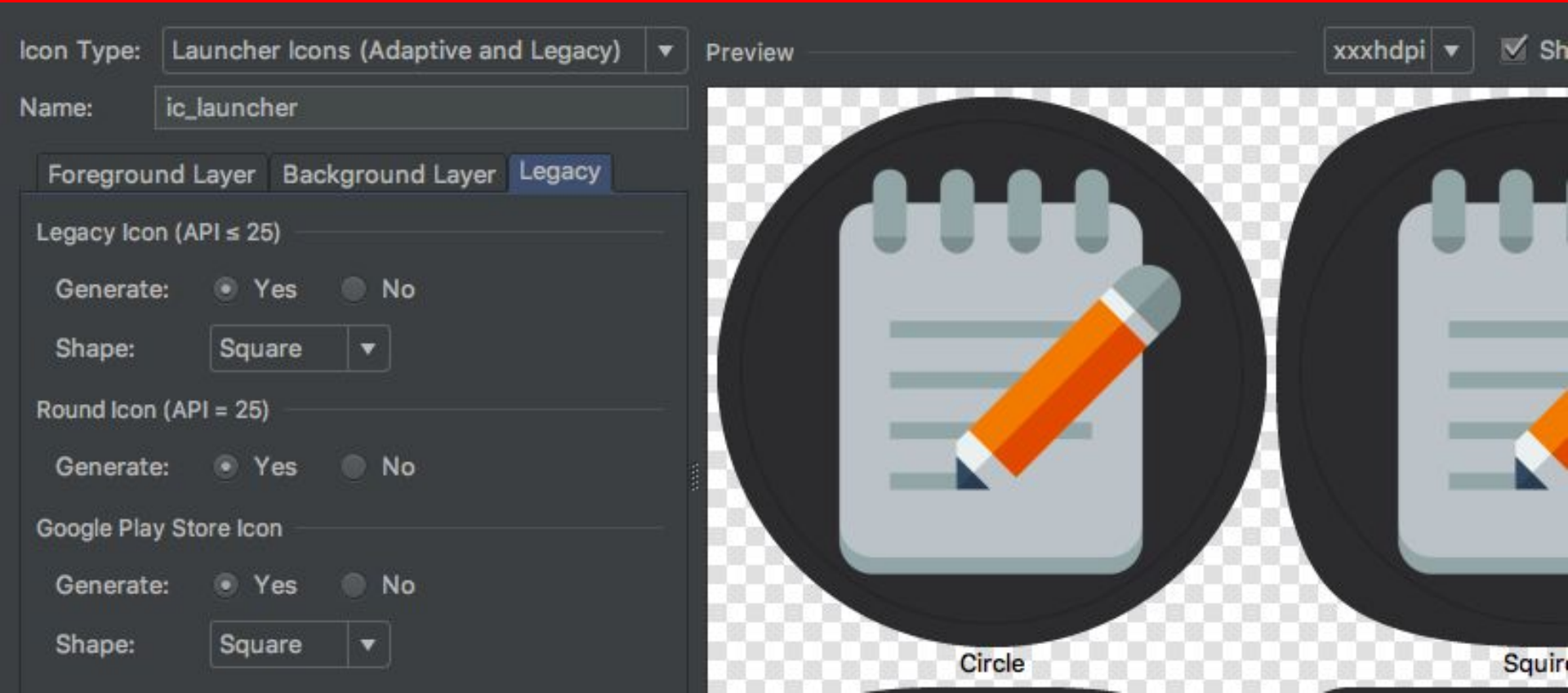
1- Aba Foreground Layer
2- Arquivo `icon_launcher_foreground.png` disponível na pasta imagem do EAD
3- Ajuste para encaixar corretamente em todos os layouts apresentados

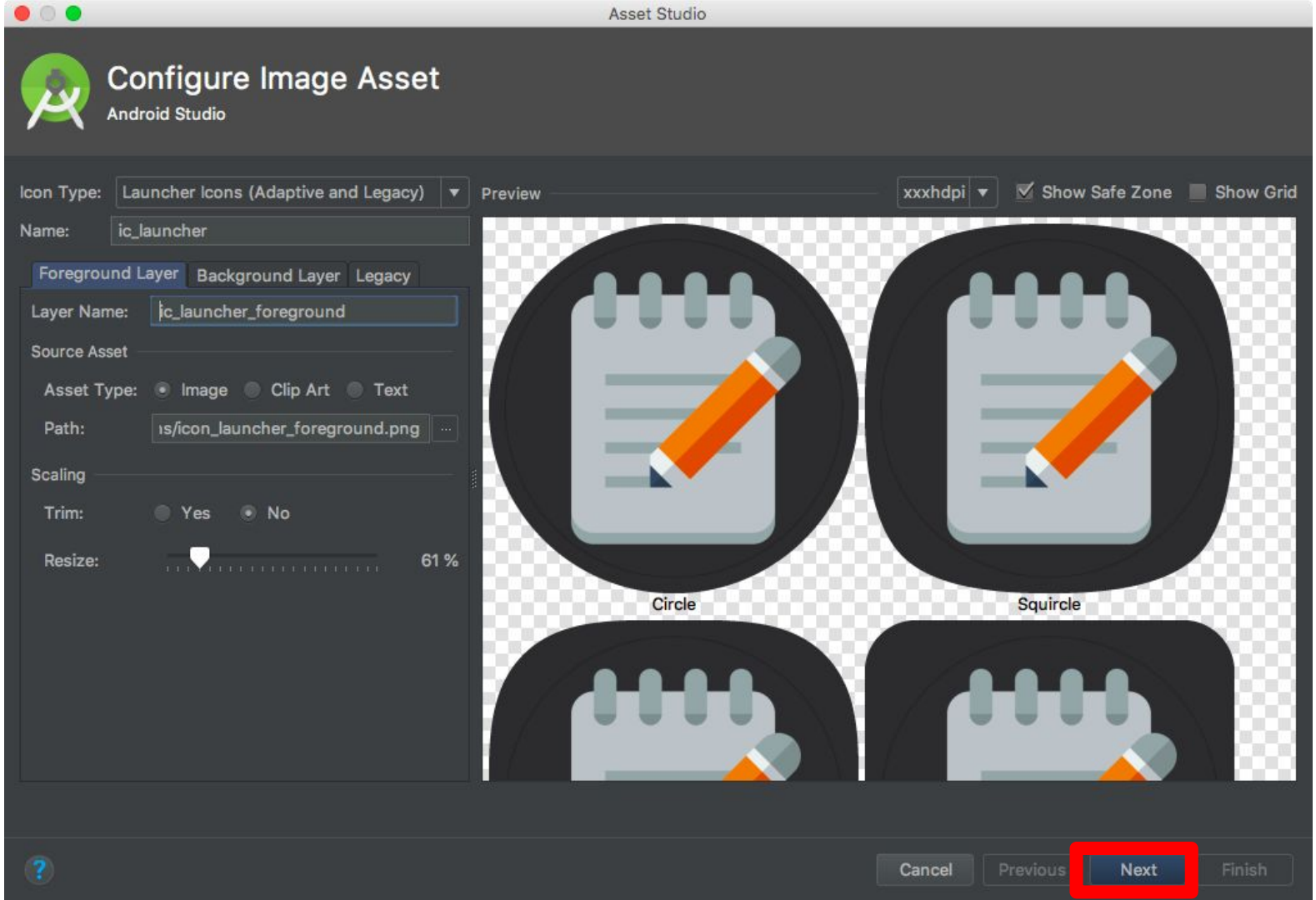


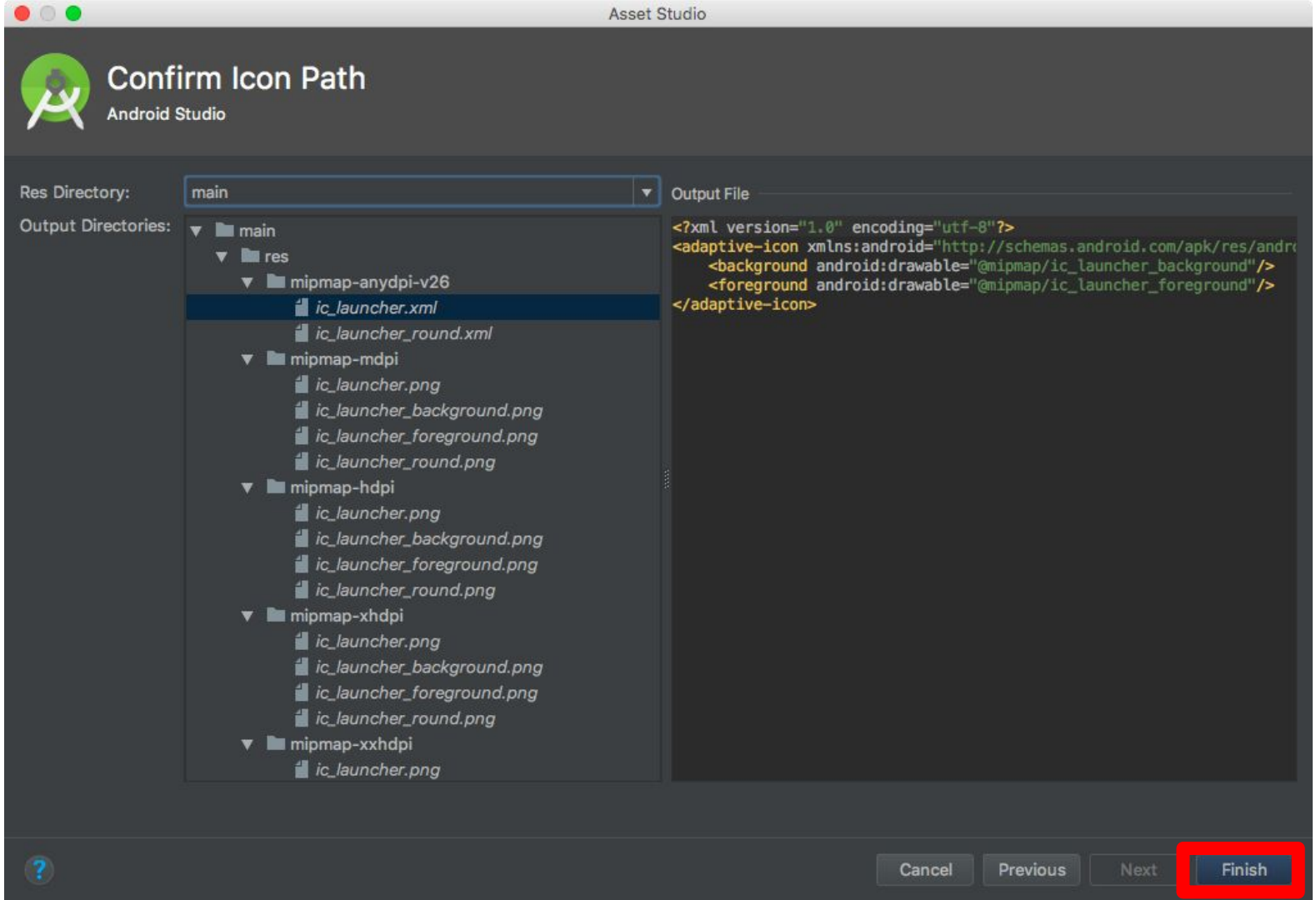
1- Aba Background Layer
2- Arquivo `icon_launcher_background.png` disponível na pasta imagem do EAD
3- Se necessário, ajuste para encaixar corretamente nos layouts



Na aba Legacy, verifique apenas se todas as opções de geração dos ícones legados encontram-se marcados com **Yes**

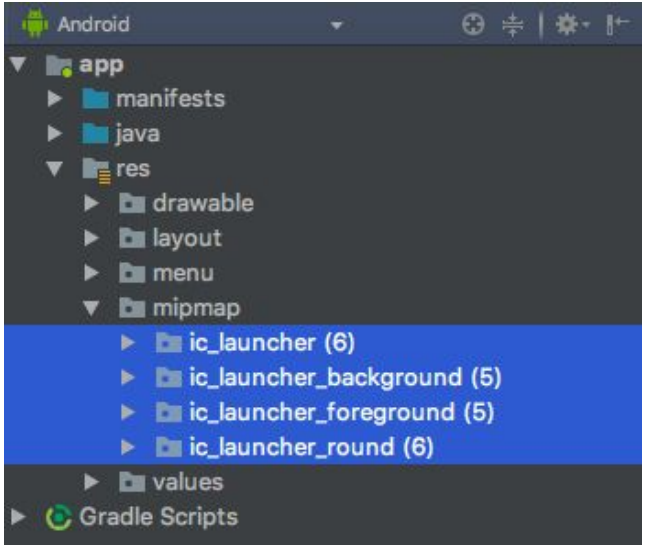


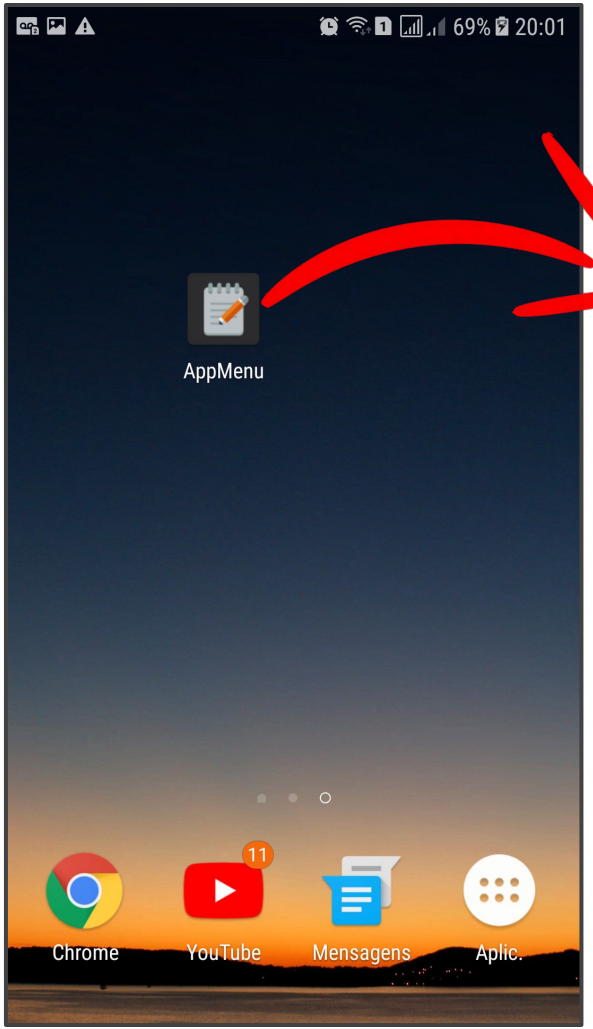

Asset Studio
Configure Image Asset
Android Studio
Icon Type:
Launcher Icons (Adaptive and Legacy)
Name:
ic_launcher
Foreground Layer
Background Layer
Legacy
Layer Name:
ic_launcher_foreground
Source Asset
Asset Type:
Image
Clip Art
Text
Path:
ıs/icon_launcher_foreground.png
Scaling
Trim:
Yes
No
Resize:
61 %
Preview
xxxhdpi
Show Safe Zone
Show Grid
Circle
Squircle
Cancel
Previous
Next
Finish

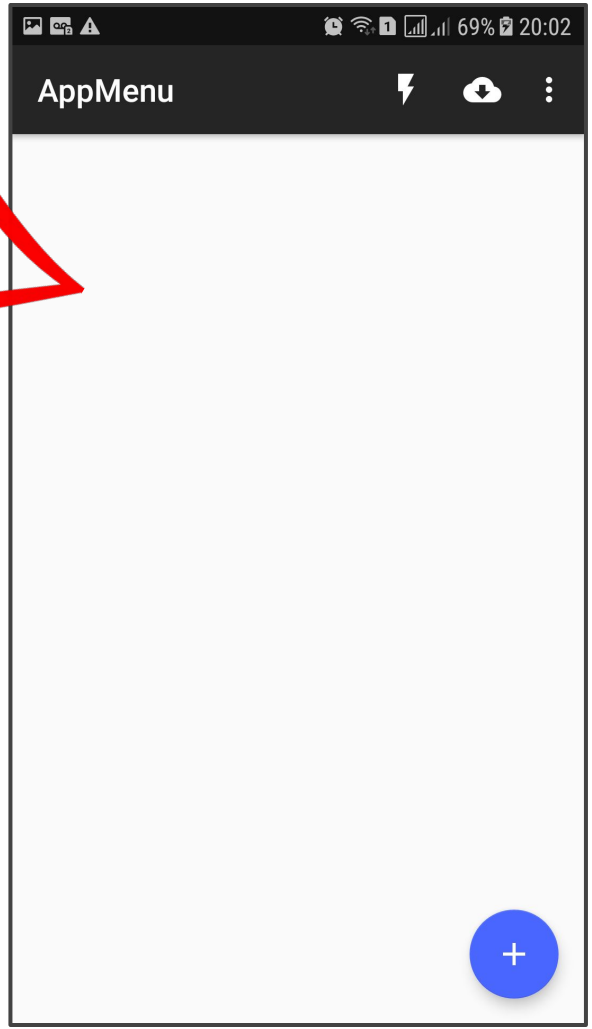

Asset Studio
Confirm Icon Path
Android Studio
Res Directory:
main
Output Directories:
main
res
mipmap-anydpi-v26
ic_launcher.xml
ic_launcher_round.xml
mipmap-mdpi
ic_launcher.png
ic_launcher_background.png
ic_launcher_foreground.png
ic_launcher_round.png
mipmap-hdpi
ic_launcher.png
ic_launcher_background.png
ic_launcher_foreground.png
ic_launcher_round.png
mipmap-xhdpi
ic_launcher.png
ic_launcher_background.png
ic_launcher_foreground.png
ic_launcher_round.png
mipmap-xxhdpi
ic_launcher.png
Output File
<?xml version="1.0" encoding="utf-8"?>
<adaptive-icon xmlns:android="http://schemas.android.com/apk/res/andr
<background android:drawable="@mipmap/ic_launcher_background"/>
<foreground android:drawable="@mipmap/ic_launcher_foreground"/>
</adaptive-icon>
Cancel
Previous
Next
Finish

# Icon Launcher

Após este procedimento os ícones serão criados e publicados junto com seu aplicativo!



# Exercício em Sala

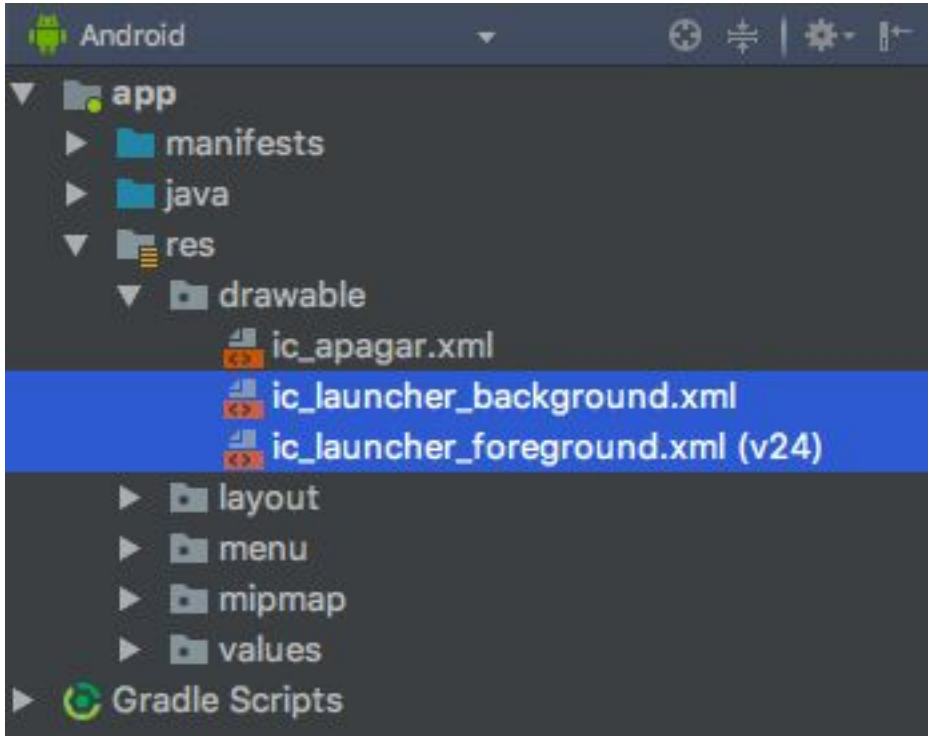
Altere o app *Notepad* observando os seguintes detalhes:

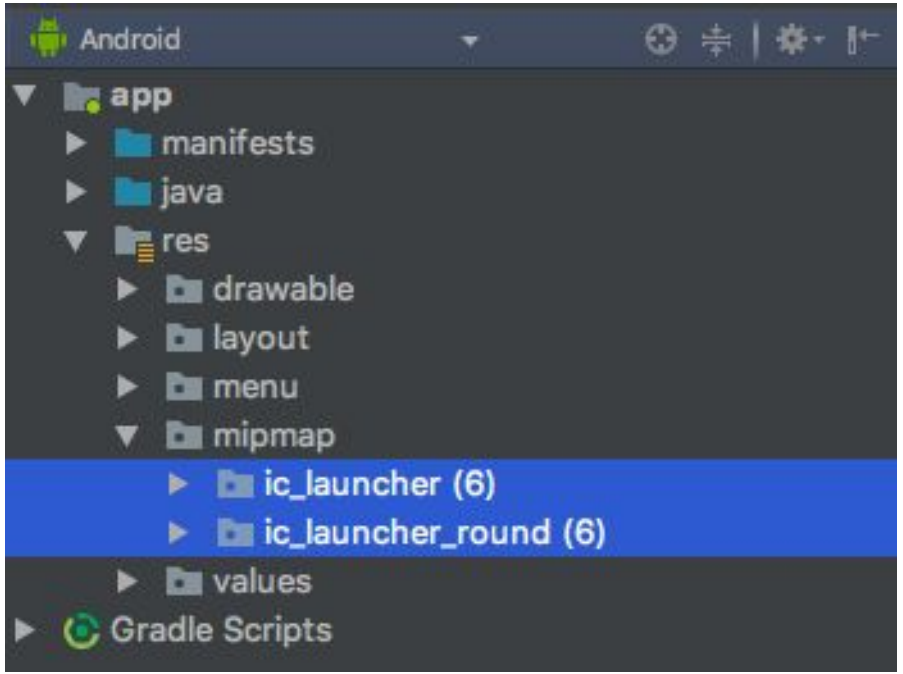
- Altere as cores do App para:
  - `colorPrimary: #2C2C2E`
  - `colorPrimaryDark #000000`
  - `colorAccent: #4967ff`
- Altere o *Icon Launcher* para os arquivos `icon_launcher_background.png` e `icon_launcher_foreground.png` disponíveis na pasta imagens do EAD

# Resolvendo o Exercício

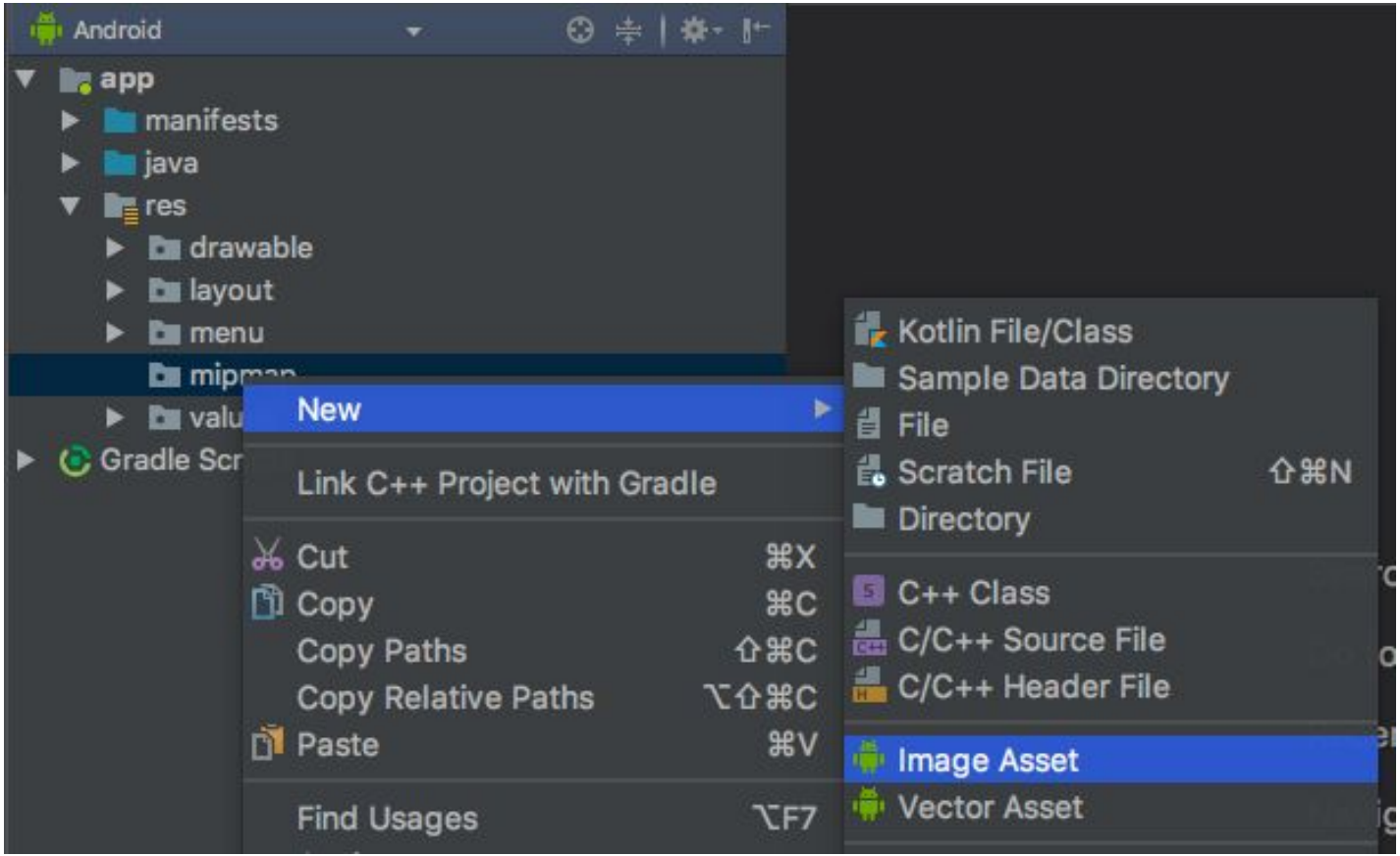
# Resolvendo o Exercício

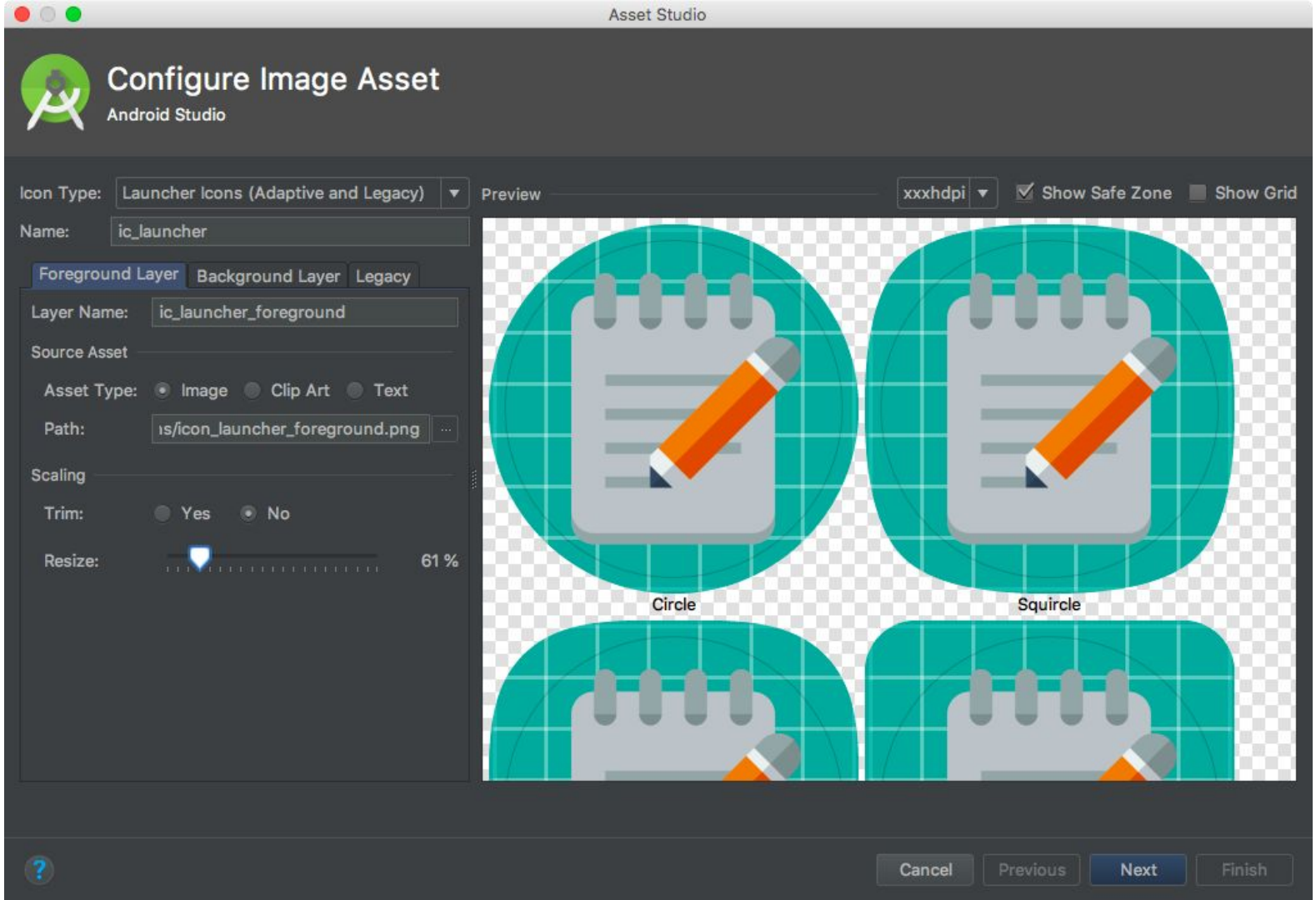
## Apague os seguintes arquivos...

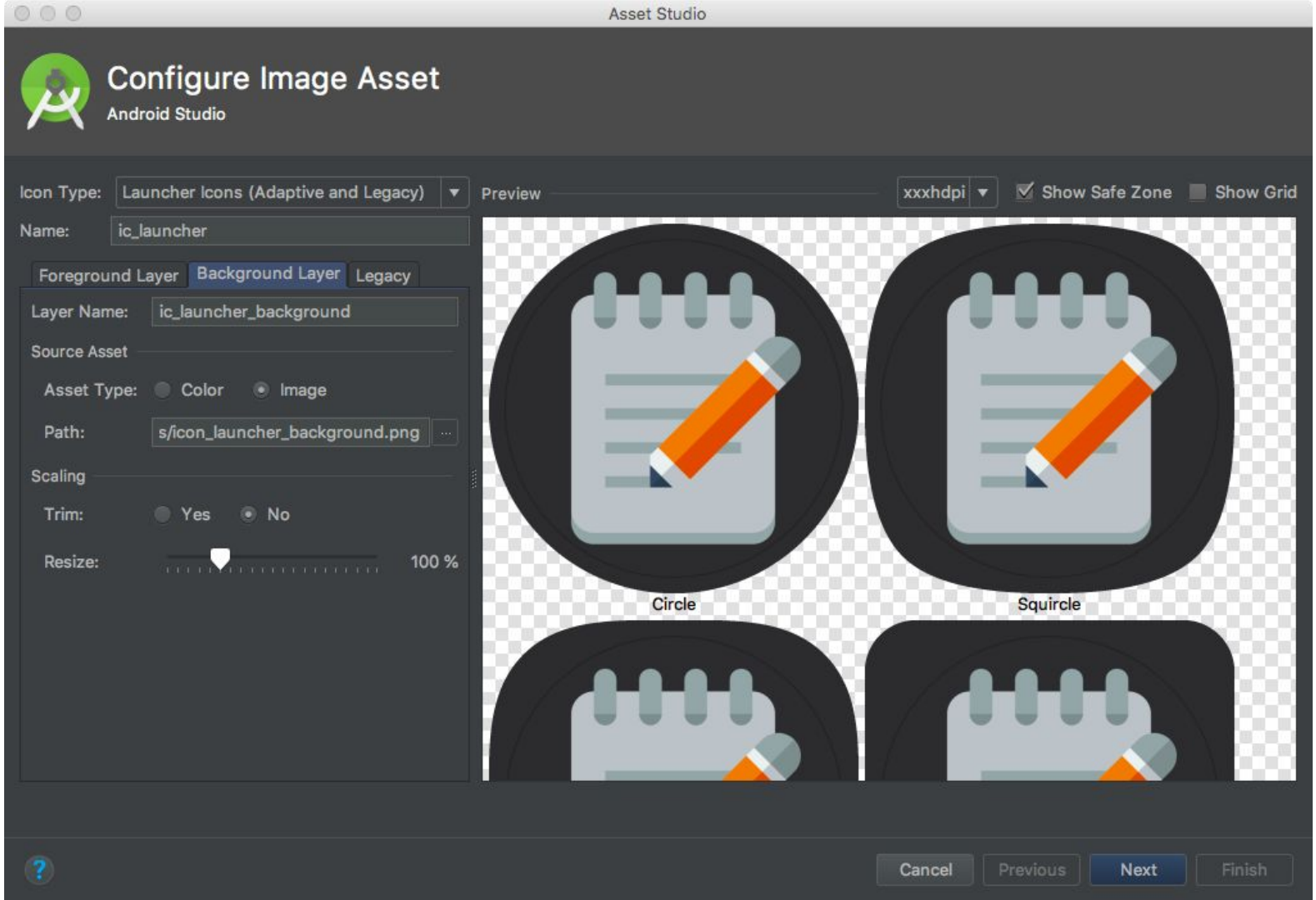


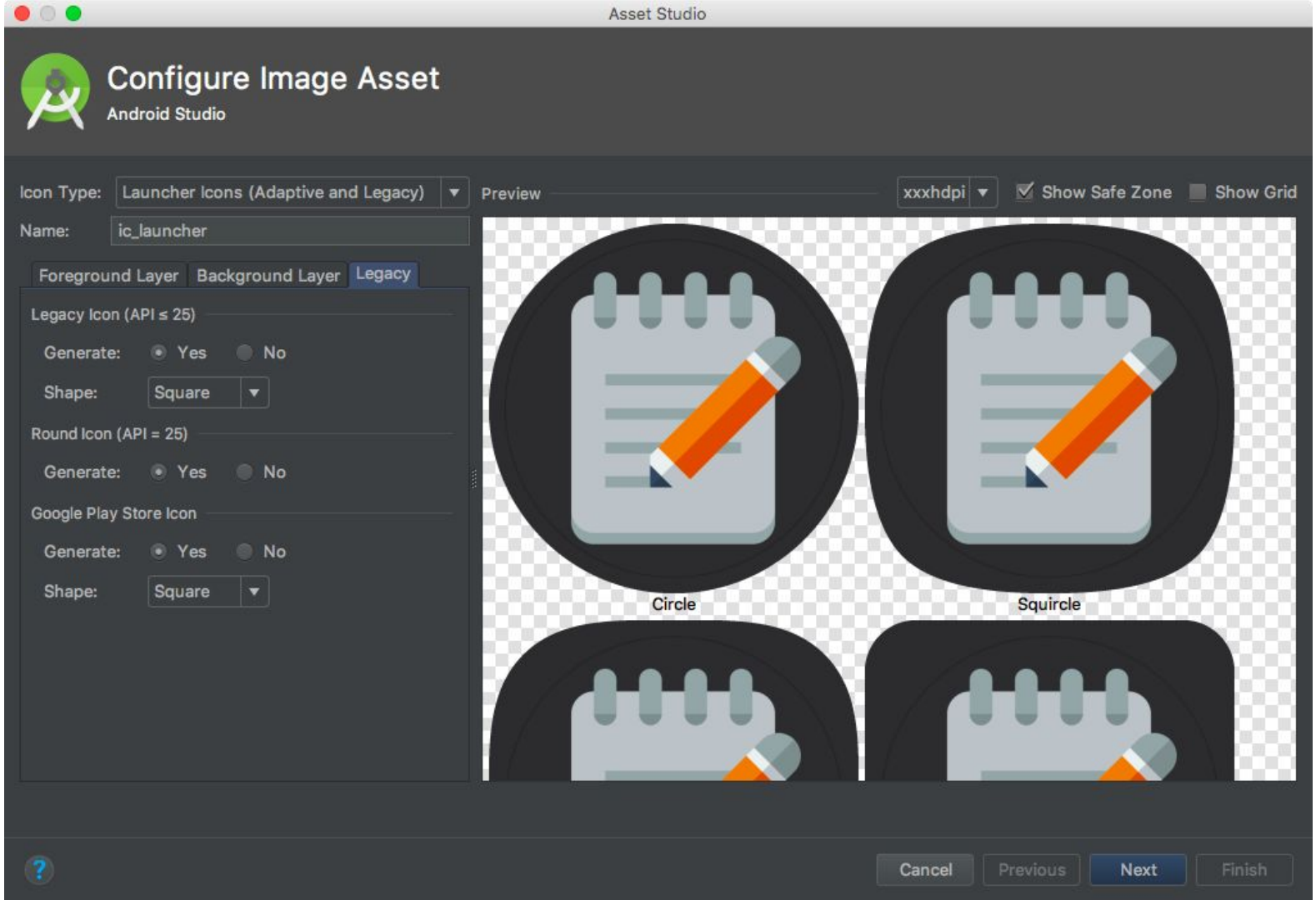



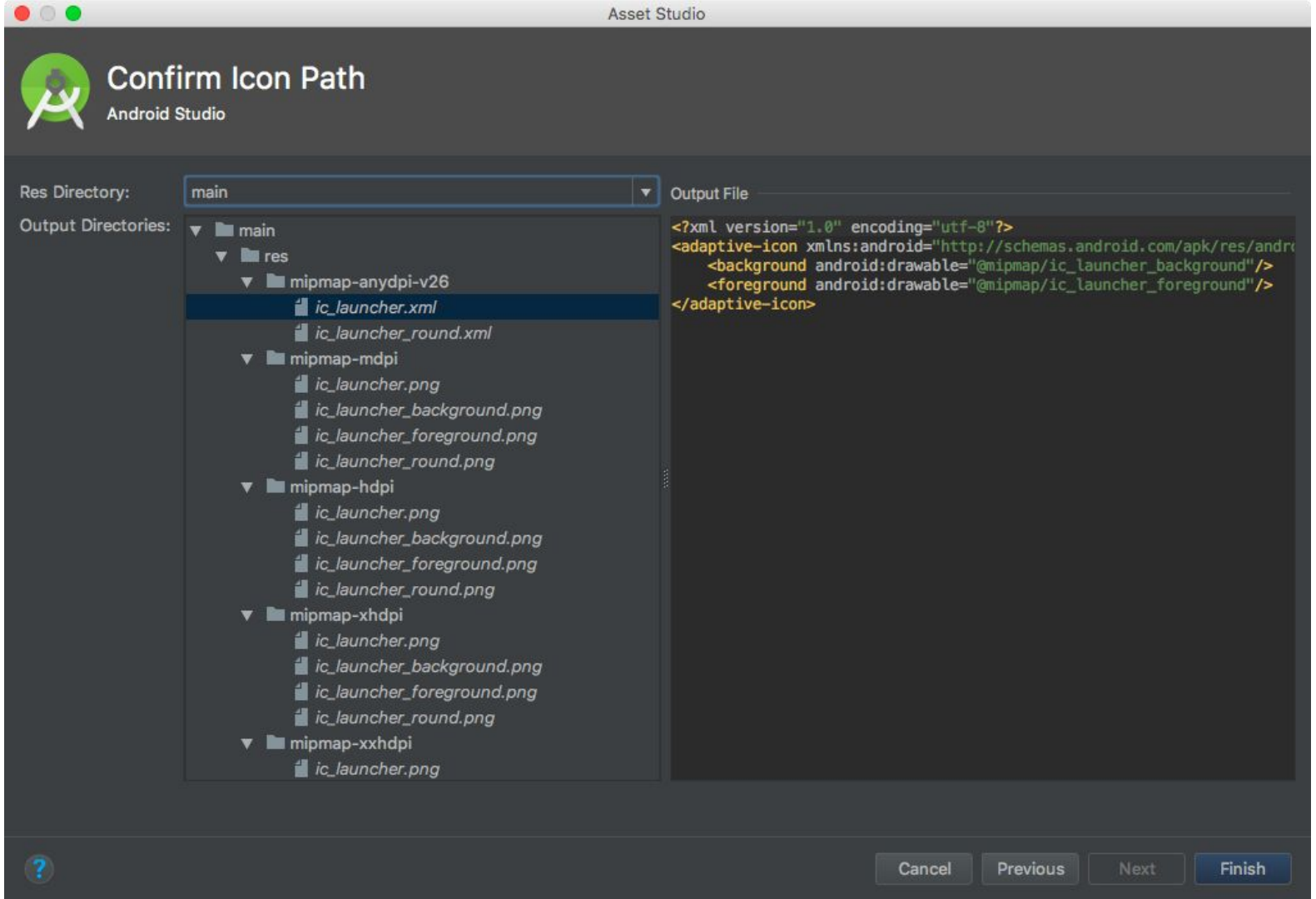
# Resolvendo o Exercício

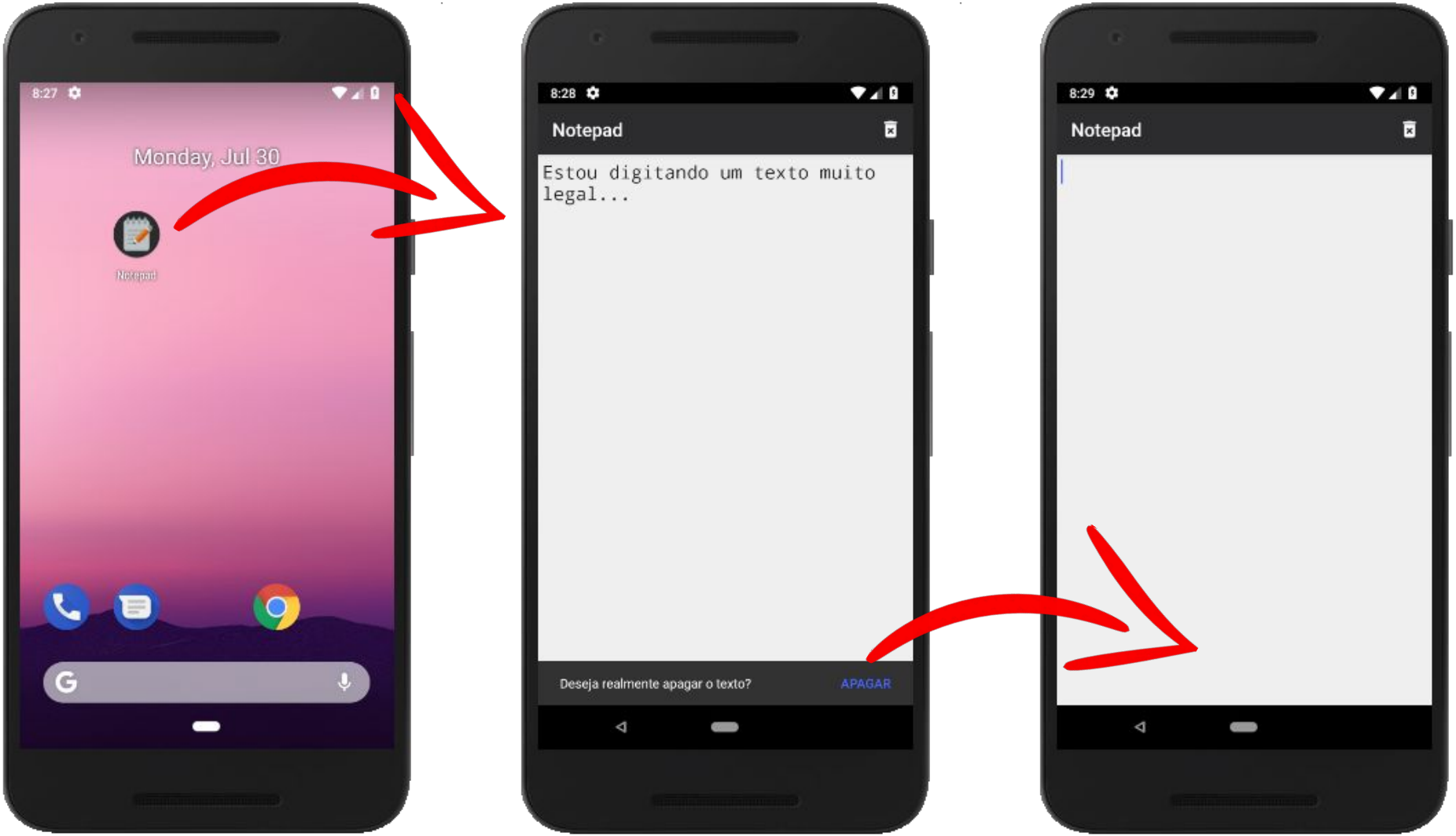



Asset Studio
Configure Image Asset
Android Studio
Icon Type:
Launcher Icons (Adaptive and Legacy)
Name:
ic_launcher
Foreground Layer
Background Layer
Legacy
Layer Name:
ic_launcher_foreground
Source Asset
Asset Type:
Image
Clip Art
Text
Path:
ıs/icon_launcher_foreground.png
Scaling
Trim:
Yes
No
Resize:
61 %
Preview
xxxhdpi
Show Safe Zone
Show Grid
Circle
Squircle
Cancel
Previous
Next
Finish


Asset Studio
Configure Image Asset
Android Studio
Icon Type:
Launcher Icons (Adaptive and Legacy)
Name:
ic_launcher
Foreground Layer
Background Layer
Legacy
Layer Name:
ic_launcher_background
Source Asset
Asset Type:
Color
Image
Path:
s/icon_launcher_background.png
Scaling
Trim:
Yes
No
Resize:
100 %
Preview
xxxhdpi
Show Safe Zone
Show Grid
Circle
Squircle
Cancel
Previous
Next
Finish


Asset Studio
Configure Image Asset
Android Studio
Icon Type:
Launcher Icons (Adaptive and Legacy)
Name:
ic_launcher
Foreground Layer
Background Layer
Legacy
Legacy Icon (API ≤ 25)
Generate:
Yes
No
Shape:
Square
Round Icon (API = 25)
Generate:
Yes
No
Google Play Store Icon
Generate:
Yes
No
Shape:
Square
Preview
xxxhdpi
Show Safe Zone
Show Grid
Circle
Squircle
Cancel
Previous
Next
Finish


Asset Studio
Confirm Icon Path
Android Studio
Res Directory:
main
Output Directories:
main
res
mipmap-anydpi-v26
ic_launcher.xml
ic_launcher_round.xml
mipmap-mdpi
ic_launcher.png
ic_launcher_background.png
ic_launcher_foreground.png
ic_launcher_round.png
mipmap-hdpi
ic_launcher.png
ic_launcher_background.png
ic_launcher_foreground.png
ic_launcher_round.png
mipmap-xhdpi
ic_launcher.png
ic_launcher_background.png
ic_launcher_foreground.png
ic_launcher_round.png
mipmap-xxhdpi
ic_launcher.png
Output File
<?xml version="1.0" encoding="utf-8"?>
<adaptive-icon xmlns:android="http://schemas.android.com/apk/res/andr
<background android:drawable="@mipmap/ic_launcher_background"/>
<foreground android:drawable="@mipmap/ic_launcher_foreground"/>
</adaptive-icon>
Cancel
Previous
Next
Finish


8:27
Monday, Jul 30
Notepad
8:28
Notepad
Estou digitando um texto muito legal...
Deseja realmente apagar o texto?
APAGAR
8:29
Notepad

# Obrigado!