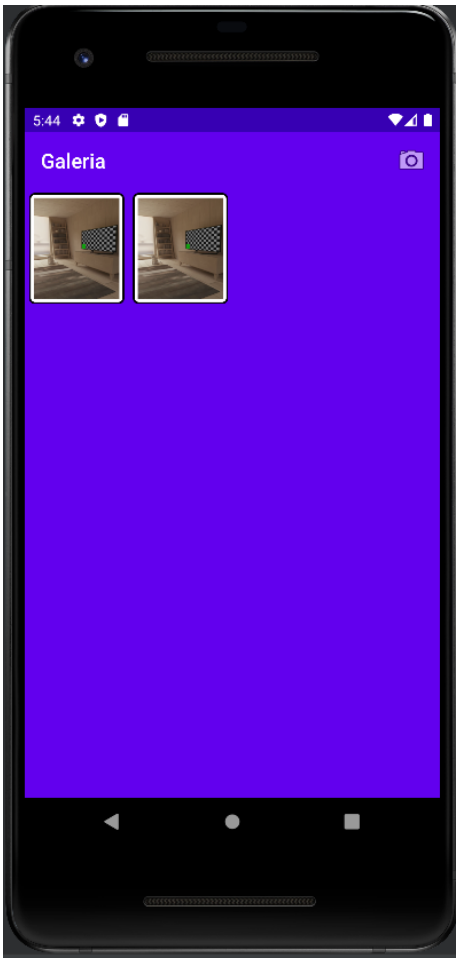
Dispositivos Móveis - Prof. Daniel Ribeiro Trindade

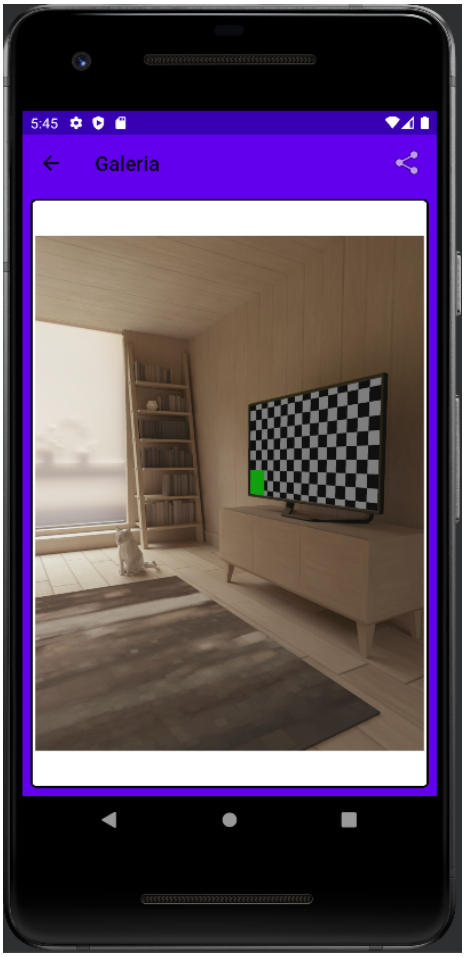
Roteiro para criação da APP Galeria

Neste roteiro iremos criar uma app para Android do tipo Galeria de Fotos.

A app será composta por duas telas:

1. **MainActivity**: a tela principal, que exibe as fotos tiradas em uma galeria no formato GRID. Nessa tela o usuário também conta com um botão na ActionBar da app para acionar a câmera e tirar a foto.
2. **PhotoActivity**: uma segunda tela, responsável por exibir em tamanho maior uma das fotos da galeria exibida em MainActivity. Nessa tela o usuário também conta com um botão na ActionBar da app para compartilhar a foto.





Passo 1 - Criação do Projeto

- Crie um novo projeto no Android Studio
  - O nome da APP deve ser Galeria (ou Galeria 2);
  - O package deve ser sobrenome.nome.galeria. Por exemplo: **trindade.daniel.galeria**;

    - A linguagem deve ser Java;
    - Escolha o template “Empty”;
- Depois de criar o projeto, compartilhe ele no GitHub;
- Dê o commit inicial no projeto e de um push para que ele suba para o GitHub.

Passo 2 - Criação da Interface

Neste projeto iremos criar um app que não usa a ActionBar padrão do Android. ActionBar é a barra que toda tela de app Android possui por padrão e que fica na parte superior da tela. Ela contém geralmente somente o nome da app.

Para que possamos criar nossa própria ActionBar, primeiramente nós temos que indicar no projeto que usaremos um tema que não possui uma ActionBar por padrão. Para isso:

- Abra o arquivo **“res -> values -> themes ->themes.xml”**
- Mude o tema parent de **DarkActionBar** para **NoActionBar**, como mostrado no código abaixo:

```
1. <resources xmlns:tools="http://schemas.android.com/tools">
2.     <!-- Base application theme. -->
3.     <style name="Theme.Galeria"
   parent="Theme.MaterialComponents.DayNight.NoActionBar">
4.
5.  ...
6.
7.     </style>
8. </resources>
```

O arquivo **themes.xml** indica, de uma forma geral, como será a aparência da sua app. No Android, o tema de uma aplicação sempre herda de um tema já pré-definido da biblioteca **MaterialComponents**. Assim, para você alterar alguma característica de estilo, você deve apenas acrescentar/modificar o item que você quer alterar em relação ao tema parent. Assim, ao alterar o parent do tema para **NoActionBar**, o tema da aplicação agora irá herdar de um tema já pré-definido onde as telas da aplicação não mais terão uma ActionBar padrão.

Ainda no arquivo themes.xml, há um item definido como colorPrimary. Esse item de estilo define a cor de fundo que a ActionBar e outro elementos de interface da sua app vão ter.

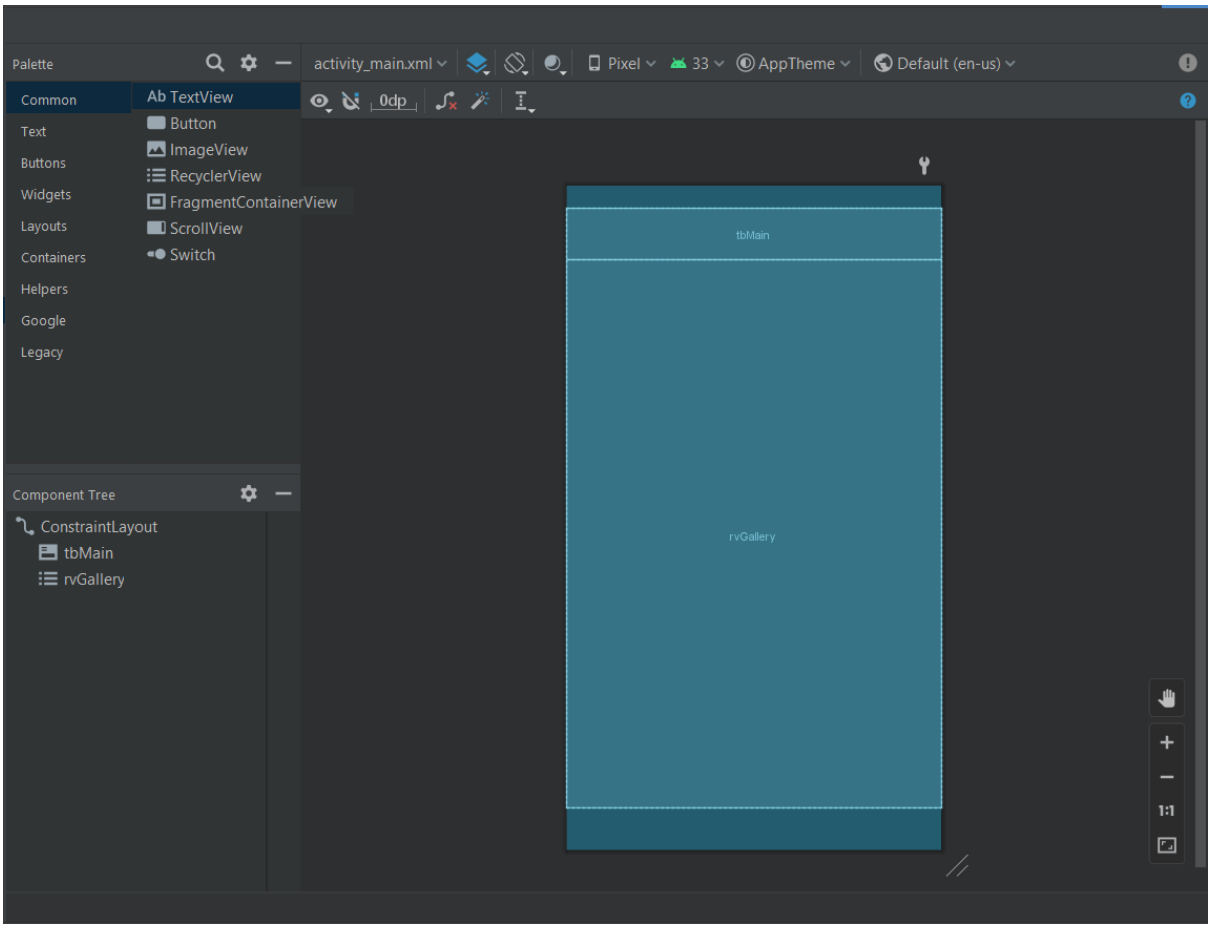
- **Escolha uma nova cor para colorPrimary!**

Para saber mais sobre estilos e temas no Android:

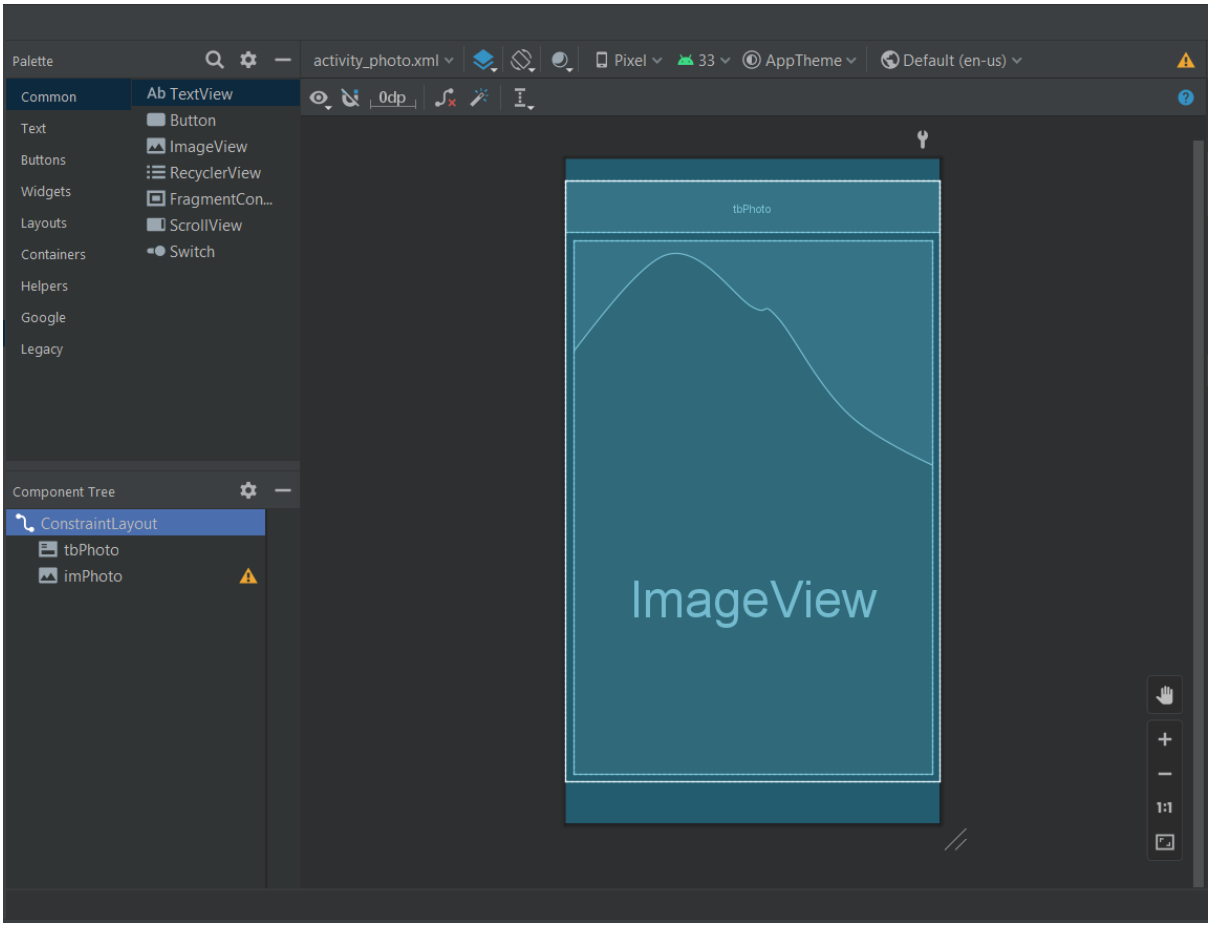
https://developer.android.com/guide/topics/ui/look-and-feel/themes?hl=pt-br

Depois de modificar o tema da app, iremos criar as telas da app. São duas telas: MainActivity e PhotoActivity:

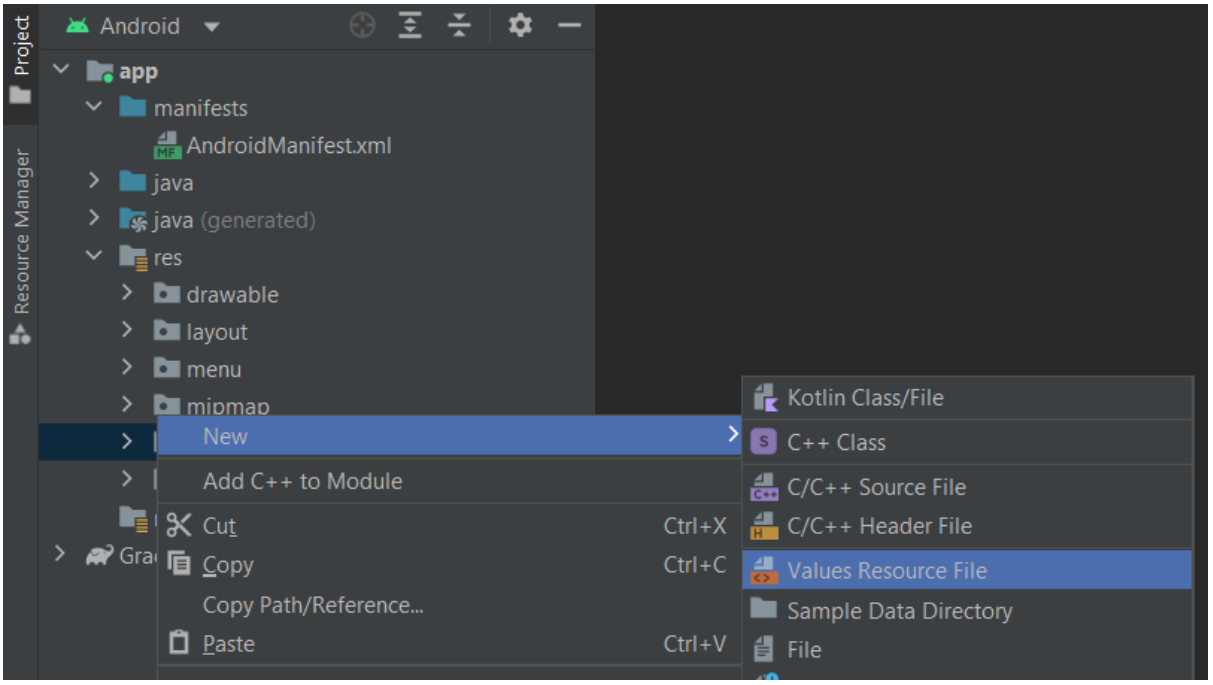
- Modifique o layout de **MainActivity** para ficar como mostrado na figura abaixo. Esse layout possui dois elementos de UI:
    - **ToolBar** (Palette -> Containers -> ToolBar) -> Deve ficar na parte superior da tela, ocupando o tamanho máximo na horizontal (match_constraint) e tamanho mínimo na altura (wrap_content). O id desse elemento de interface deve ser **tbMain**.
    - **RecycleView** (Palette -> Containers -> RecycleView) -> deve ocupar o restante da tela, sempre abaixo da ToolBar. O id desse elemento de interface deve ser **rvGallery**.

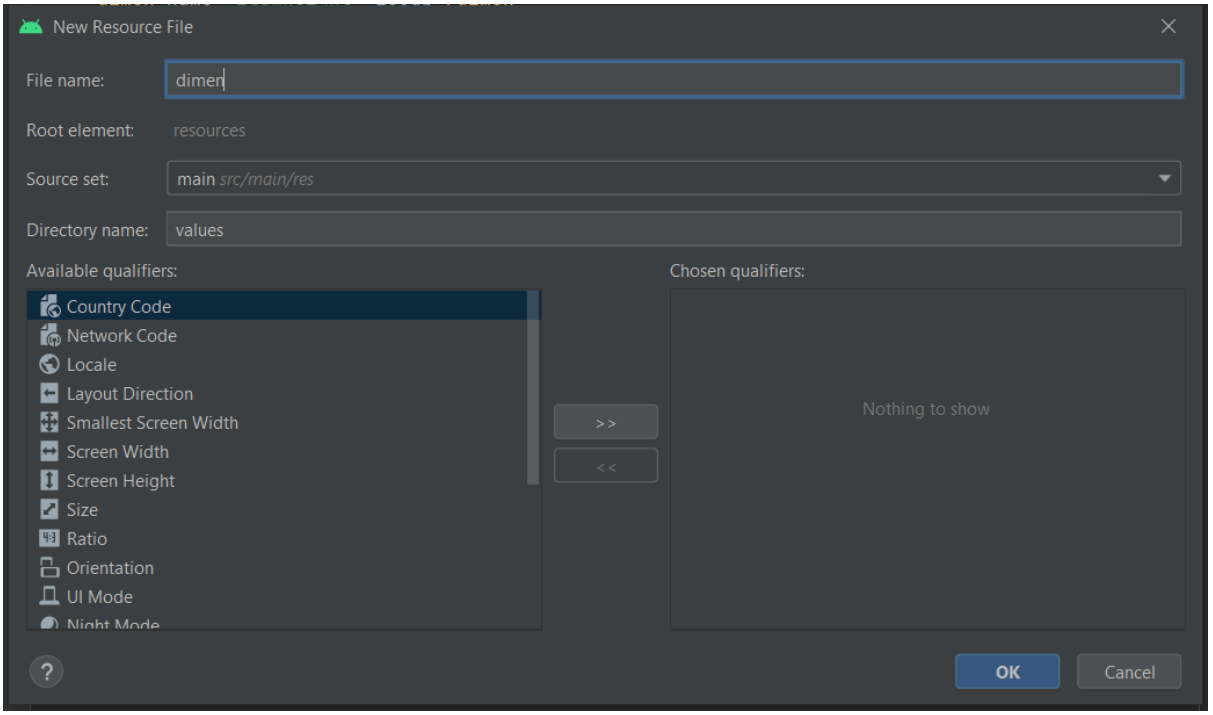


- Crie uma nova Activity chamada **PhotoActivity**;
- Modifique o layout de **PhotoActivity** para ficar como mostrado na figura abaixo. Esse layout possui dois elementos de UI:
    - **ToolBar** (Palette -> Containers -> ToolBar) -> Deve ficar na parte superior da tela, ocupando o tamanho máximo na horizontal (match_constraint) e tamanho mínimo na altura (wrap_content). O id desse elemento de interface deve ser **tbPhoto**.
    - **ImageView** (Palette -> Widgets-> ImageView) -> deve ocupar o restante da tela, sempre abaixo da ToolBar. O id desse elemento de interface deve ser **imPhoto**.



- Crie um novo arquivo dentro da pasta res -> value, como nas figuras abaixo. Esse arquivo irá guardar os valores de dimensões para o tamanho da imagem que será exibida na galeria de fotos em MainActivity.



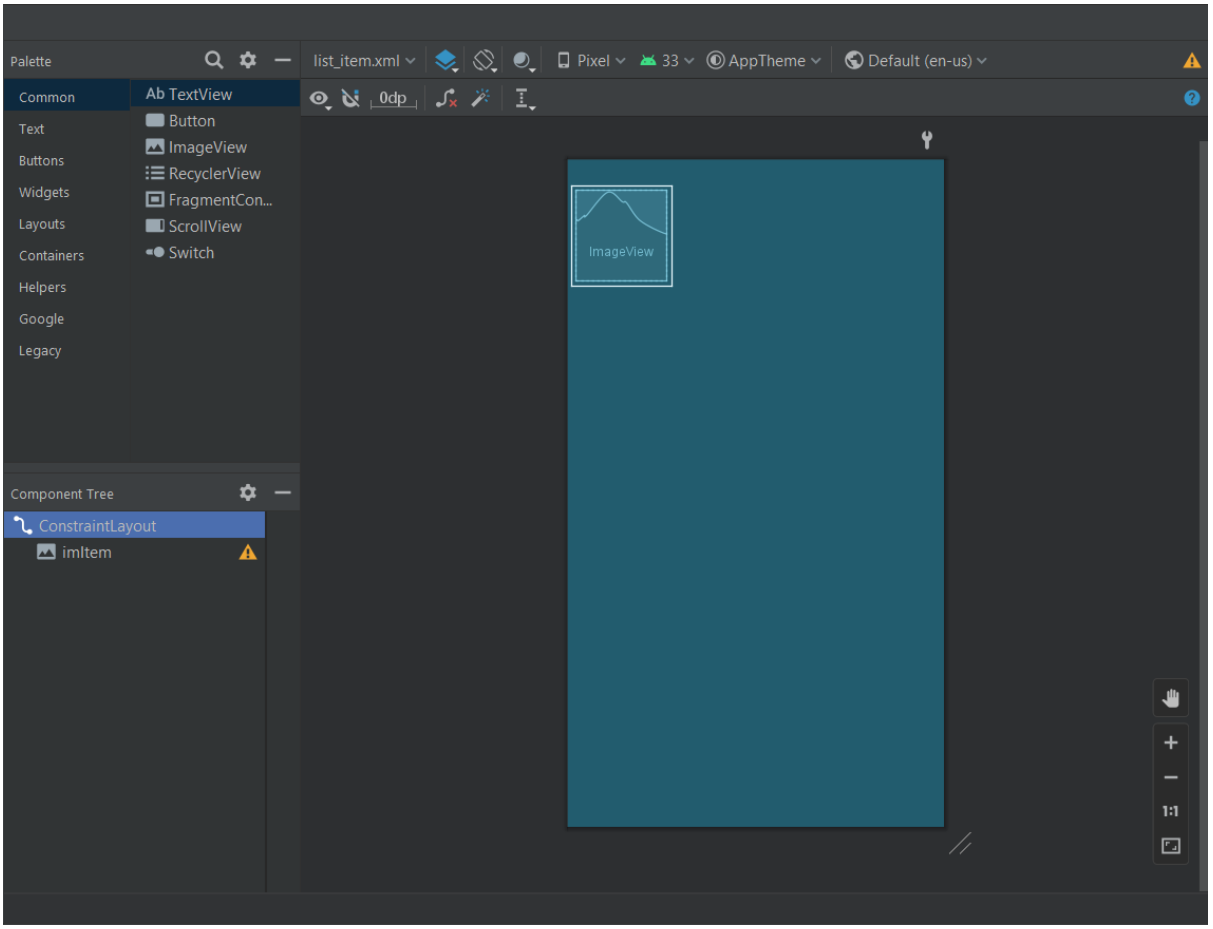


- No novo arquivo, coloque o seguinte conteúdo:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <resources>
3.     <dimen name="itemWidth">100dp</dimen>
4.     <dimen name="itemHeight">100dp</dimen>
5. </resources>
```

No código acima nós definimos dois novos recursos que poderão ser usados na nossa aplicação: itemWidth e itemHeight, ambos com valor de 100dp. Esses recursos são do tipo dimen e serão usados para especificar o tamanho da imagem da tela de galeria.
A galeria presente em MainActivity é formada por uma lista (RecycleView) cujos os item são formados por imagens em miniatura. Essas imagens são exibidas na galeria na forma de um Grid de imagens. O próximo passo é criar o layout de um item do RecycleView.

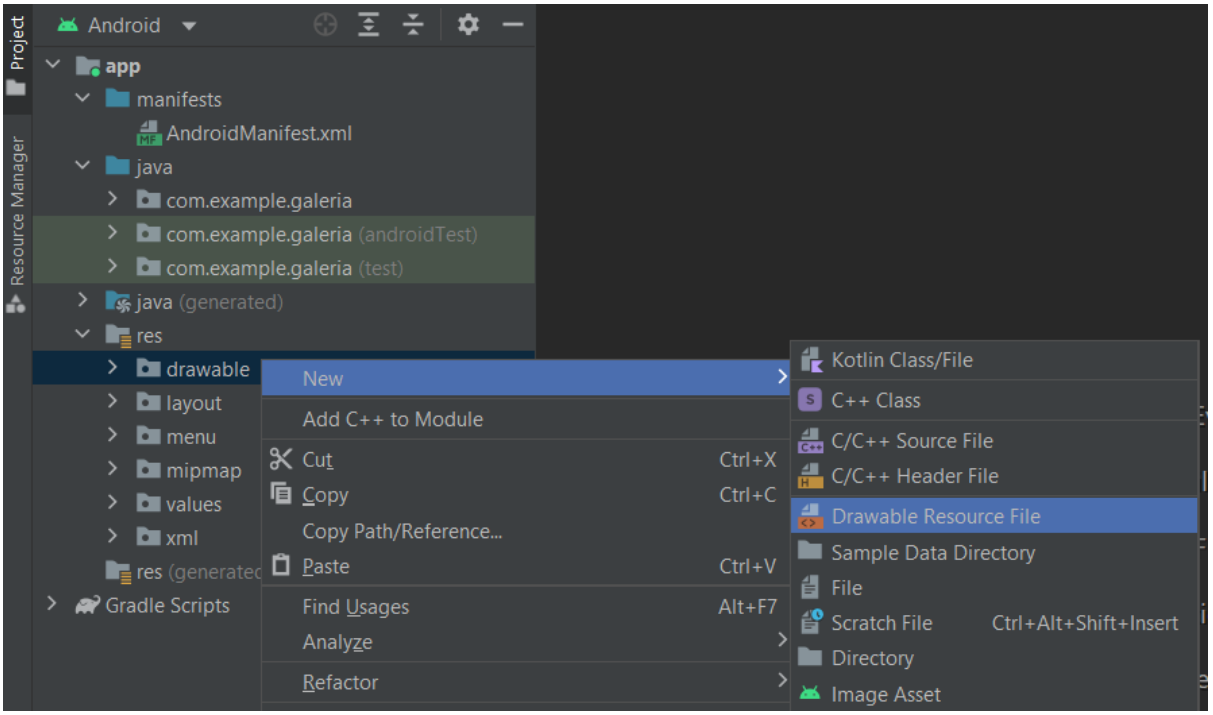
- Crie um novo arquivo de layout com o nome **list_item.xml** (não é pra criar uma nova Activity).
    - O layout deve conter apenas um elemento de interface do tipo ImageView, como mostrado na figura abaixo. O id desse elemento deve ser **imItem**.



Passo 3 - Modificando a Aparência da APP: Cores, Background

Uma outra forma de modificar o layout/aparência de um elemento de interface é através da criação de um drawable que irá ser setado como plano de fundo (background) do elemento. Um drawable é um arquivo que define formas geométricas através de código.

- Crie um novo arquivo de drawable como mostrado na figura abaixo. O nome do arquivo deve ser **image_bg**



No novo arquivo criado, coloque o código abaixo:

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <shape
   xmlns:android="http://schemas.android.com/apk/res/android">
3.     <solid android:color="#FFFFFF"/>
4.     <corners android:radius="5dp"/>
5.     <padding android:bottom="5dp" android:left="5dp"
   android:right="5dp" android:top="5dp"/>
6.     <stroke android:width="2dp"/>
7. </shape>
```

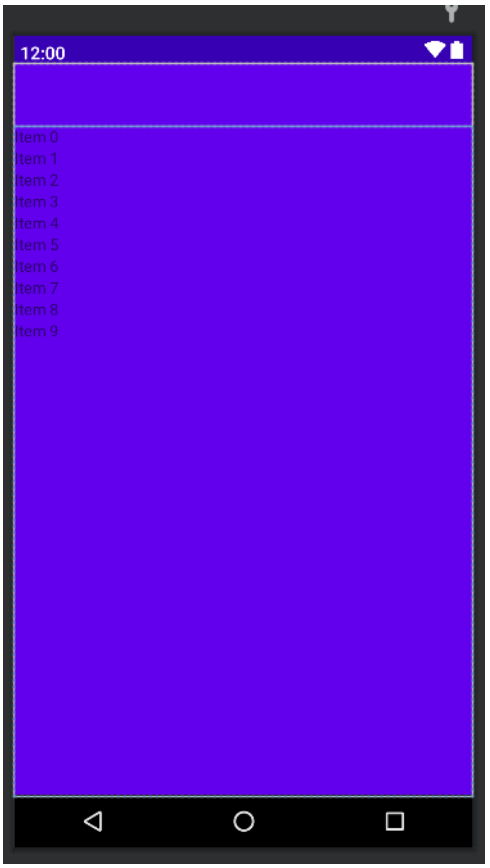
O código acima define um shape (forma) na linha 2. Essa “forma” possui as seguintes caracteristicas:

- Cor solida branca (linha 3);
- Cantos arredondados com raio de 5dp (linha 4);
- Margem interna de 5dp em todas as direções (linha 5);
- Linha de contorno com grossura de 2dp (linha 6);

Depois de definir um drawable, nós podemos usá-lo como background para alguns elementos de interface.

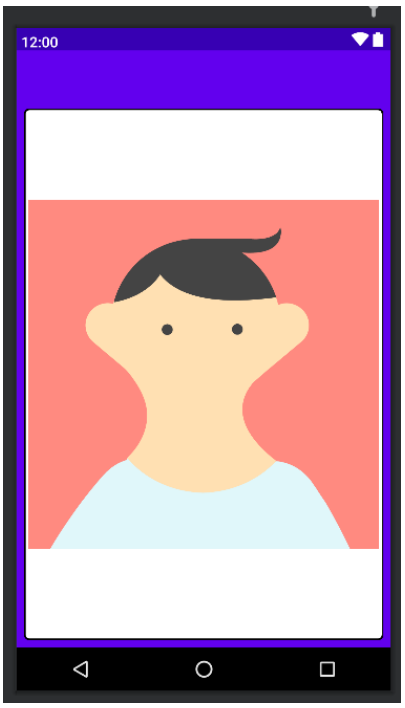
- Modifique o layout de **MainActivity** para ficar como mostrado na figura abaixo. Esse layout possui dois elementos de UI:
    - Selecione o elemento de UI **tbMain**. Nas propriedades do elemento, ache o atributo **background** e o sete para **colorPrimary**. Isso tem o efeito de fazer com que a ToolBar fique na cor definida em colorPrimary.

- Selecione o elemento de UI **rvGallery**. Nas propriedades do elemento, ache o atributo **background** e o sete para **colorPrimary**. Isso tem o efeito de fazer com que o fundo do RecycleView fique na cor definida em colorPrimary.

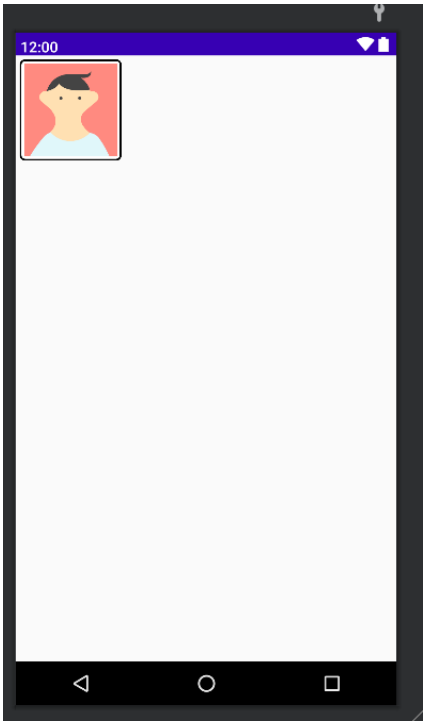


- Modifique o layout de **PhotoActivity** para ficar como mostrado na figura abaixo:

    - Selecione o elemento de UI **tbPhoto**. Nas propriedades do elemento, ache o atributo **background** e o sete para **colorPrimary**. Isso tem o efeito de fazer com que a ToolBar fique na cor definida em colorPrimary.
    - Selecione o elemento de UI **ConstraintLayout**. Nas propriedades do elemento, ache o atributo **background** e o sete para **colorPrimary**. Isso tem o efeito de fazer com que o fundo do elemento fique na cor definida em colorPrimary.
    - Selecione o elemento de UI **imPhoto**. Nas propriedades do elemento, ache o atributo **background** e o sete para **image_bg**. Isso tem o efeito de fazer com que o fundo do elemento fique com a forma definida em no arquivo image_bg.xml.



- Modifique o layout **list_item.xml** para ficar como mostrado na figura abaixo:
    - Selecione o elemento de UI **imlItem**. Nas propriedades do elemento, ache o atributo **background** e o sete para **image_bg**. Isso tem o efeito de fazer com que o fundo do elemento fique com a forma definida em no arquivo image_bg.xml.



Passo 4 - Adição da ToolBar

Após a criação dos elementos de interface, é hora de configurá-los.

- Sete o elemento de interface tbMain (ToolBar) como a ActionBar padrão de MainActivity.
    - Dentro do método onCreate de MainActivity.java adicione os códigos das linhas 6 e 7 abaixo:

```
1. @Override
2. protected void onCreate(Bundle savedInstanceState) {
3.     super.onCreate(savedInstanceState);
4.     setContentView(R.layout.activity_main);
5.
6.     Toolbar toolbar = findViewById(R.id.tbMain);
7.     setSupportActionBar(toolbar);
8.
9.  ...
10.}
```

No código acima, a linha 6 obtem o elemento tbMain, enquanto a linha 7 indica para MainActivity que tbMain deve ser considerado como a ActionBar padrão da tela.

- Faça a mesma coisa para configura a ToolBar de PhotoAcitivity.

Passo 5 - Como Habilitar o Botão “Voltar” da ToolBar

Na tela PhotoActivity é possível habilitar um botão “Voltar” na ActionBar. Esse botão tem o ícone de uma seta apontando para a esquerda e vai aparecer logo no começo da ActionBar, antes do nome da app. Ao apertar esse botão, o usuário é redirecionado para uma tela “pai” que é definida pelo desenvolvedor. No nosso caso, apertar o botão “Voltar” nos levará de volta para MainActivity.

Para habilitar o botão de voltar:

- No método onCreate de PhotoActivity.java acrescente as linhas 9 e 10 do código abaixo.

```
1. @Override
2. protected void onCreate(Bundle savedInstanceState) {
3.     super.onCreate(savedInstanceState);
4.     setContentView(R.layout.activity_photo);
5.
6.     Toolbar toolbar = findViewById(R.id.tbPhoto);
7.     setSupportActionBar(toolbar);
8.
9.     ActionBar actionBar = getSupportActionBar();
10.        actionBar.setDisplayHomeAsUpEnabled(true);
11. ...
```

No código acima, a linha 9 obtém da Activity a ActionBar padrão (a qual foi setada pelas linhas 6 e 7). A linha 10 habilita o botão de voltar na ActionBar.

O próximo passo é indicar para qual Activity o usuário será redirecionado quando ele clicar no botão “Voltar”.

- No arquivo AndroidManifest.xml, ache a declaração de PhotoActivity. Modifique a declaração de forma que ela fique como no código abaixo.

```
1. <activity android:name=".PhotoActivity"
2.     android:parentActivityName=".MainActivity">
3.     <meta-data
4.         android:name="android.support.PARENT_ACTIVITY"
5.         android:value=".MainActivity"/>
6. </activity>
```

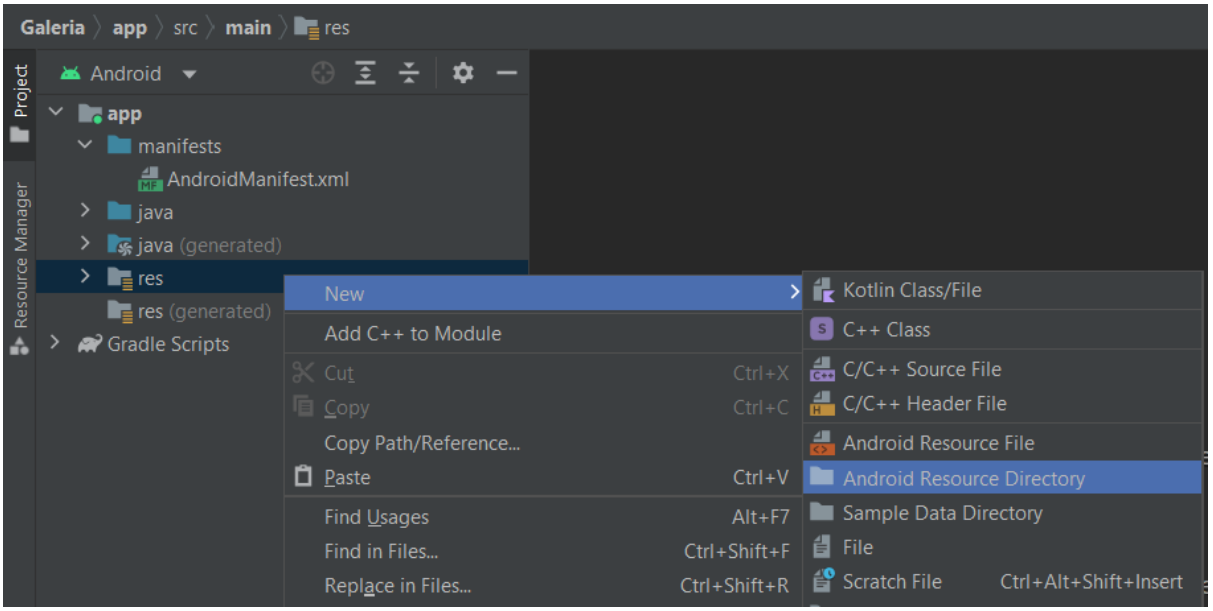
No código acima, as 2, 4 e 5 indicam que o usuário será levado para MainActivity quando clicar no botão “Voltar” de PhotoActivity.

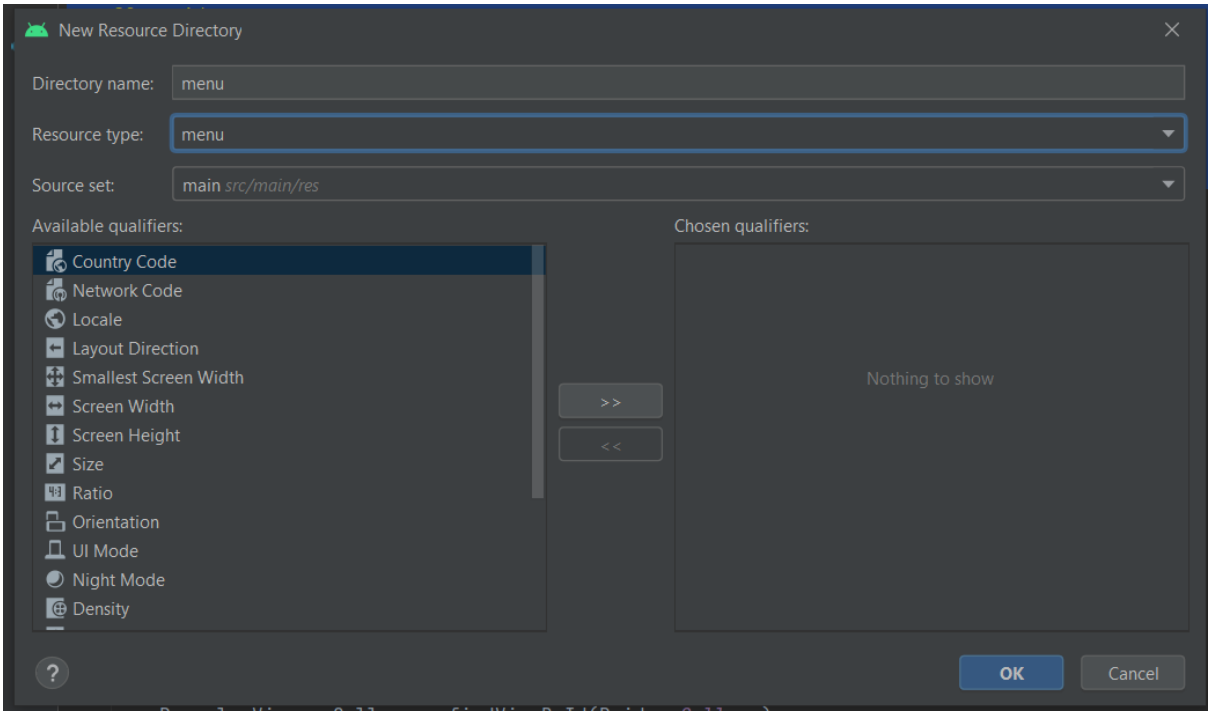
Passo 6 - Como Adicionar Itens na ToolBar

Os itens da ToolBar são definidos na forma de opções em um menu. Cada ToolBar da app deve ter seu próprio arquivo de menu onde são definidos os itens que aparecerão na ToolBar.

Para criar um arquivo de menu:

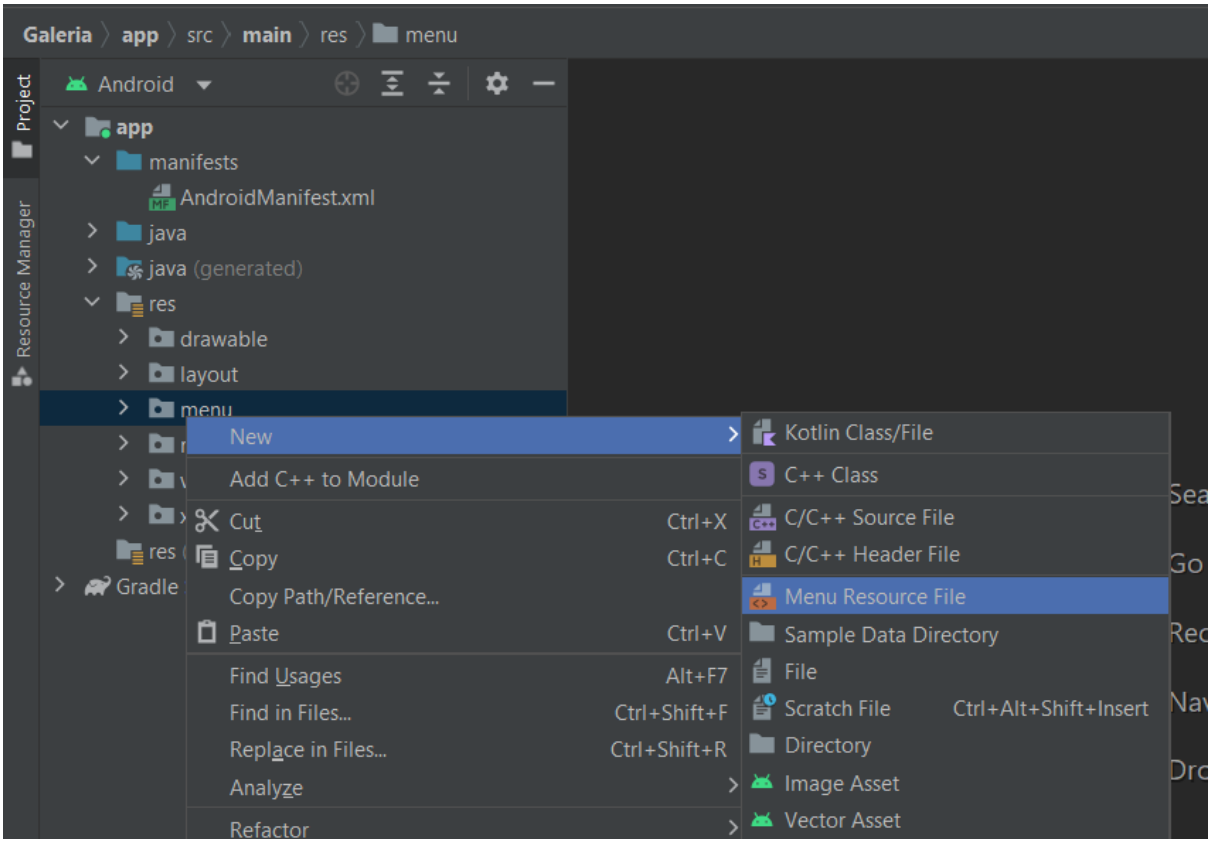
- Crie uma pasta dentro do diretório **res**. Essa pasta irá conter os arquivos de menus. As figuras abaixo mostram como fazer isso.

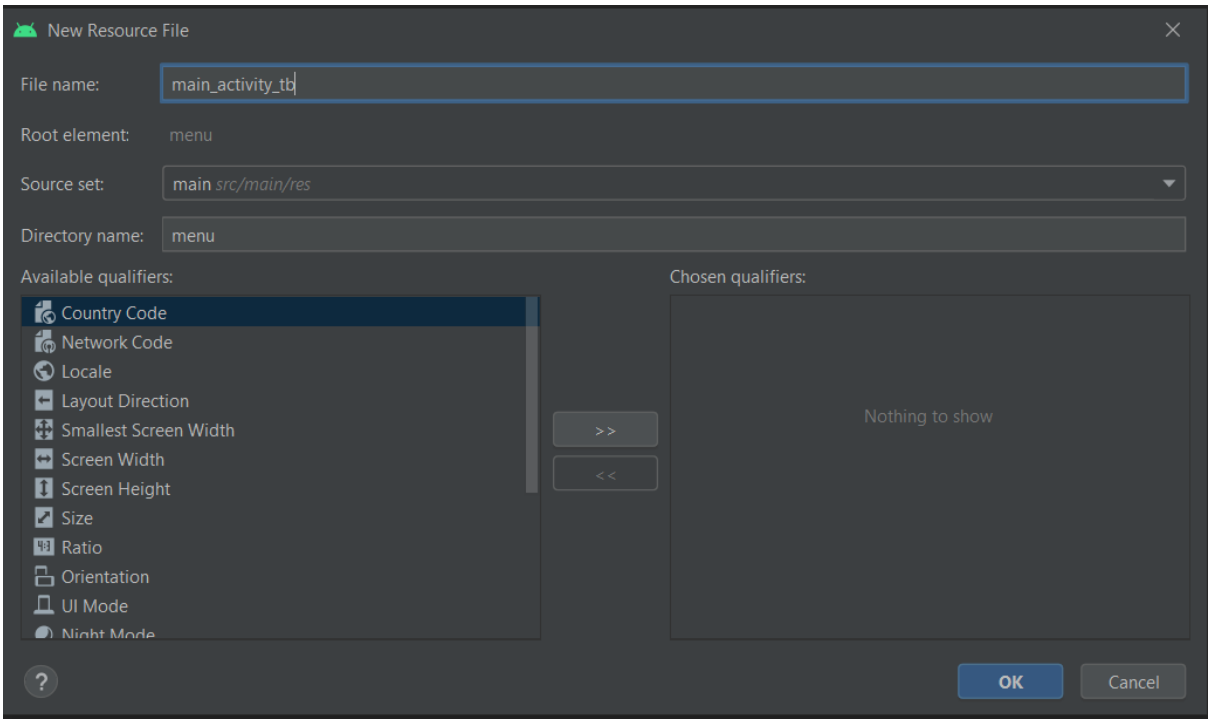




Depois de criar a pasta menu, é hora de criar os arquivos de menu em si:

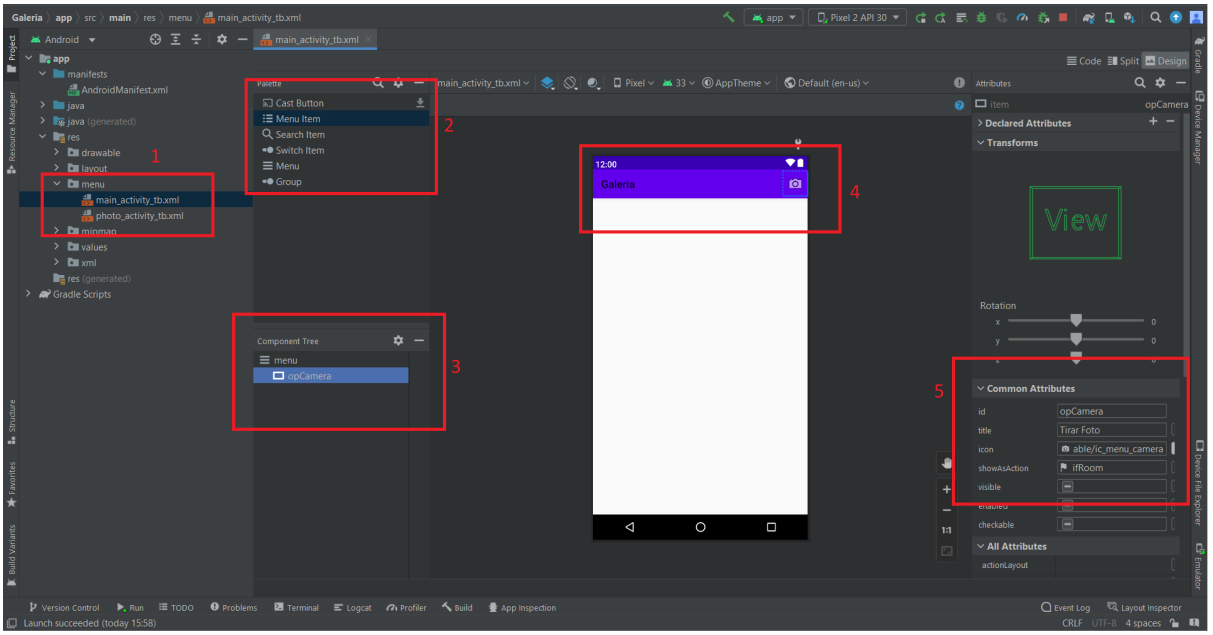
- Crie dois arquivos de menus: **main_activity_tb**.xml, que vai conter as opções de menu para a toolbar de MainActivity; **photo_activity_tb**.xml, que vai conter as opções de menu para a toolbar de PhotoActivity (ver figuras abaixo).





A figura abaixo mostra a visualização dos dois arquivos de menus criados (destaque 1). Para adicionar um item de menu no arquivo, temos que selecionar o item “Menu Item” na paleta de UI (destaque 2) e arrastá-lo para a árvore de menu (destaque 3). Depois de adicionado, o item aparece na pré-visualização da toolbar (destaque 4)

- Adicione um item de menu em main_activity_tb. No item adicionado sete as seguintes propriedades (destaque 5):
  - id -> opCamera;
  - title -> Tirar Foto;
  - icon -> Escolha um ícone de câmera;
  - showAsAction -> ifRoom;
- Adicione um item de menu em photo_activity_tb. No item adicionado sete as seguintes propriedades (destaque 5):
  - id -> opShare;
  - title -> Compartilhar;
  - icon -> Escolha um ícone de compartilhamento;
  - showAsAction -> ifRoom;



- Em MainActivity.java, adicione o método abaixo:

```
1. @Override
2. public boolean onCreateOptionsMenu(Menu menu) {
3.     super.onCreateOptionsMenu(menu);
4.     MenuInflater inflater = getMenuInflater();
5.     inflater.inflate(R.menu.main_activity_tb, menu);
6.     return true;
7. }
```

O código acima cria um “inflador de menu” (linha 4). Esse inflador cria as opções de menu definidas no arquivo de menu passado e as adiciona no menu da Activity.

- Faça o mesmo para PhotoActivity.

Passo 7 - Como Tratar os Eventos de Click dos Itens da ToolBar

Uma vez definido e criado o menu, temos que definir o que será feito quando o usuário clicar em uma determinada ação.

- Em MainActivity.java adicione o método abaixo.

```
1. @Override
2. public boolean onOptionsItemSelected(@NonNull MenuItem item) {
3.     switch (item.getItemId()) {
4.         case R.id.opCamera:
5.             dispatchTakePictureIntent();
6.             return true;
7.         default:
```

```
8.              return super.onOptionsItemSelected(item);
9.      }
10.     }
```

O método acima será chamado sempre que um item da ToolBar for selecionado. Caso o ícone de câmera tenha sido clicado, então será executado código que dispara a câmera do celular (linha 5).

- Em PhotoActivity.java adicione o método abaixo.

```
1. @Override
2. public boolean onOptionsItemSelected(@NonNull MenuItem item) {
3.     switch (item.getItemId()) {
4.         case R.id.opShare:
5.             sharePhoto();
6.             return true;
7.         default:
8.             return super.onOptionsItemSelected(item);
9.     }
10.    }
```

O método acima será chamado sempre que um item da ToolBar for selecionado. Caso o ícone de câmera tenha sido clicado, então será executado código que compartilha a foto (linha 5).

Passo 8 - Como Permitir que outras Apps possam Ler e Escrever em Pastas que Pertencem à APP

No Android, cada app possui seu próprio espaço privado de armazenamento. Via de regra, somente a própria app pode ler e escrever nas pastas privadas. Entretanto, existem ocasiões em que se torna necessário acessar dados que pertencem a outras apps ou deixar que outras apps acessem os dados da nossa app. Esse é o caso quando nós usamos a câmera do celular ou queremos compartilhar arquivos com outras apps.

No caso da câmera, a nossa app requisita que a câmera tire uma foto e a salve em um arquivo que está localizado no espaço privado da nossa aplicação. Como a câmera usa uma app que não é a nossa, então devemos dar acesso temporário para que a app da câmera seja capaz de salvar a foto no arquivo que pertence a nossa app.

Para permitir que outras apps consigam ler/escrever no espaço interno da nossa app, nós iremos usar um componente Android chamado FileProvider. Ele funciona de forma similar ao compartilhamento de pastas e arquivos que existe no Windows/Linux. Entretanto, no caso do FileProvider o compartilhamento funciona apenas temporariamente.

Uma vez configurado, o FileProvider gera um endereço temporário para que outra app possar acessar um recurso da nossa aplicação. O nome desse endereço é URI. Uma URI tem o seguinte formato:

content://trindade.daniel.galeria.fileprovider/Pictures/foto.jpg

Na URI acima:

- **content://** -> significa que o recurso é um conteúdo;
- **trindade.daniel.galeria** -> o nome/package da app que é dona do recurso;
- **fileprovider** -> indica que essa URI é um compartilhamento de pastas/arquivos da app trindade.daniel.galeria;
- **Pictures** -> pasta localizada no espaço interno da app trindade.daniel.galeria, na qual por sua vez está localizado o arquivo que queremos compartilhar com outras apps;
- **foto.jpg** - > o arquivo em si que será compartilhado com outras apps;

O FileProvider gera uma URI temporária para cada arquivo que precisa ser compartilhado. A app externa recebe uma URI e, de posse dela, se torna capaz de ler/escrever no arquivo foto.jpg temporariamente.

Para configurar um FileProvider:

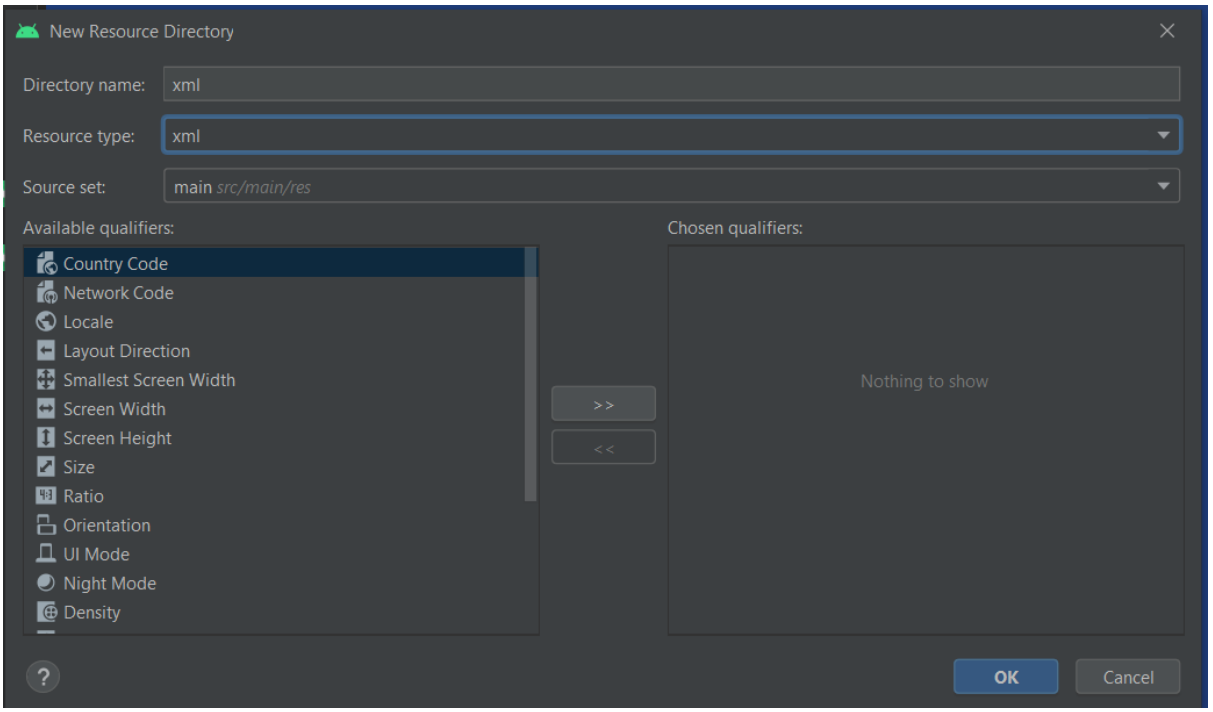
- Abra o arquivo **AndroidManifest.xml** e acrescente as linhas 16 a 25 do código abaixo. Esse código declara para o Android que app **trindade.daniel.galeria** (linha 17) irá compartilhar (linha 18) as pastas listadas no arquivo **res/xml/file_paths.xml** (linha 23).

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <manifest
   xmlns:android="http://schemas.android.com/apk/res/android"
3.     package="com.example.galeria">
4. ...
5.
6.     <application
7.         android:allowBackup="true"
8.         android:icon="@mipmap/ic_launcher"
9.         android:label="@string/app_name"
10.            android:roundIcon="@mipmap/ic_launcher_round"
11.            android:supportsRtl="true"
12.            android:theme="@style/AppTheme">
13.
14. ...
15.
16.            <provider
17.
   android:authorities="trindade.daniel.galeria.fileprovider"
18.                android:name="androidx.core.content.FileProvider"
19.                android:exported="false"
20.                android:grantUriPermissions="true">
21.                <meta-data
22.
   android:name="android.support.FILE_PROVIDER_PATHS"
23.                    android:resource="@xml/file_paths">
```

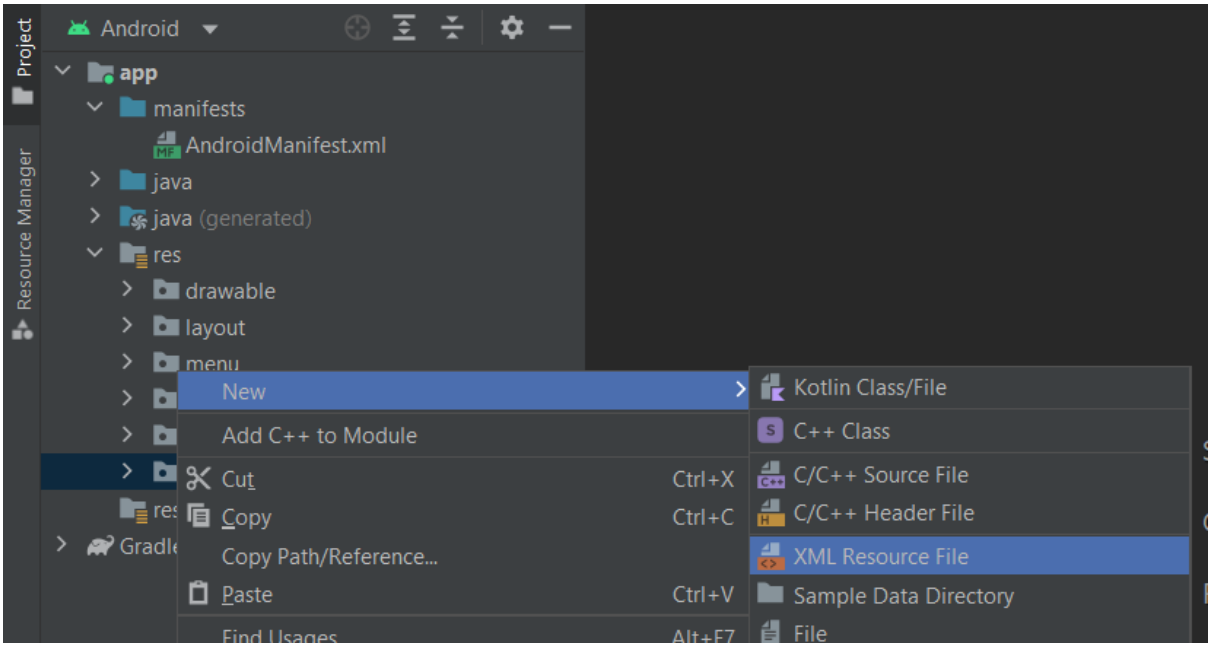
```
            </meta-data>
        </provider>

...

    </application>

</manifest>
```

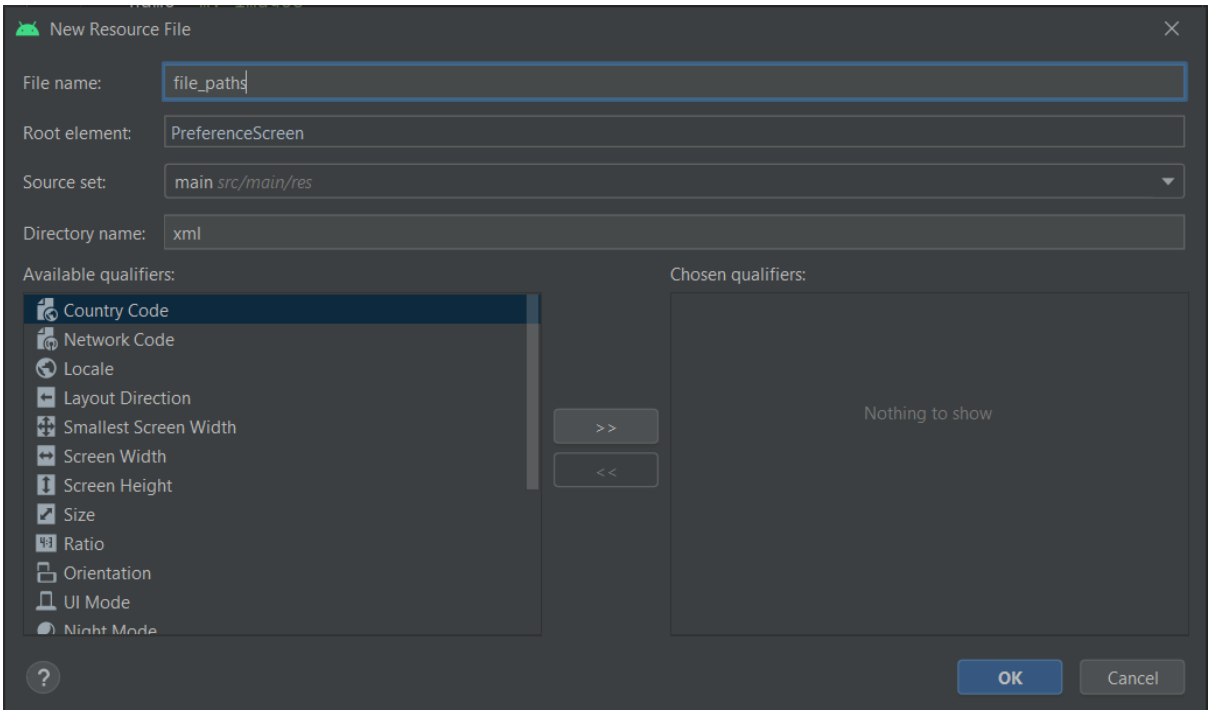
O próximo passo para concluir a configuração do nosso compartilhamento é indicar para o FileProvider quais pastas da nossa app serão efetivamente compartilhadas. Para isso, criaremos um novo arquivo chamado **file_paths.xml**:

- Crie um novo diretório de recursos do tipo xml dentro de res (figura abaixo).



- Dentro do novo diretório, crie um arquivo com o nome **file_paths** (figuras abaixo).





- Dentro do novo arquivo **file_paths**, coloque o código abaixo.

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <paths
   xmlns:android="http://schemas.android.com/apk/res/android">
3.     <external-files-path
4.         name="my_images"
5.         path="Pictures"/>
6. </paths>
```

No código acima indica que estamos compartilhando a pasta Pictures (linha 5) que pertence à estrutura de diretórios privada do tipo external-files-path (linha 3).

Passo 9 - Como Exibir os Elementos de uma Lista na Forma de uma Galeria (GRID)

Em nossa app as fotos serão tiradas pela app de câmera, que por sua vez as salva dentro da pasta privada “Pictures” da nossa app. Dessa forma, todas as fotos ficam sempre salvas nesta pasta.

Em MainActivity nós iremos exibir essas fotos na forma de uma lista organizada em GRID. Lembrando da atividade “Minha Lista”, para criar listas no Android usamos sempre o componente de UI RecycleView. Esse componente já faz parte do layout de MainActivity. O que precisamos fazer agora é configurar e preencher o RecycleView. Isso é sempre feito seguindo 3 etapas:

1. Criação de uma classe Adapter, a qual será responsável por construir e preencher cada item da lista do RecycleView. Essa classe deve implementar 3 métodos:
    a. **onCreateViewHolder** -> responsável por criar os elementos de interface para um item. Esses elementos são guardado em uma classe container do tipo **ViewHolder**.
    b. **onBindViewHolder** -> recebe o ViewHolder criado por onCreateViewHolder e preencher os elemento de UI com os dados do item;
    c. **getItemCount** -> informa quantos elementos a lista possui;
2. Criação de uma classe ViewHolder, a qual serve unicamente para guardar momentaneamente os elementos de interface;
3. Configuração do RecycleView, indicando quem será o Adapter que vai preencher a lista e como a lista será disposta.

- Crie uma classe MainAdapter que herde de RecycleView.Adapter. O construtor da classe deve aceitar dois parâmetros (linha 6):
    - Instância para a classe MainActivity;
    - Lista de Strings, onde cada String representa um caminho para uma foto salva na pasta Pictures.

```
1. public class MainAdapter extends RecyclerView.Adapter {
2.
3.     MainActivity mainActivity;
4.     List<String> photos;
5.
6.     public MainAdapter(MainActivity mainActivity, List<String>
   photos) {
7.         this.mainActivity = mainActivity;
8.         this.photos = photos;
9.     }
10.
11. ...
```

- Implemente os métodos:
    - onCreateViewHolder
    - getItemCount

- onBindViewHolder (código abaixo)

```
1. @Override
2. public void onBindViewHolder(@NonNull RecyclerView.ViewHolder
   holder, final int position) {
3.     ImageView imPhoto =
   holder.itemView.findViewById(R.id.imItem);
4.     int w = (int)
   mainActivity.getResources().getDimension(R.dimen.itemWidth);
5.     int h = (int)
   mainActivity.getResources().getDimension(R.dimen.itemHeight);
6.     Bitmap bitmap = Utils.getBitmap(photos.get(position), w,
   h);
7.     imPhoto.setImageBitmap(bitmap);
8.     imPhoto.setOnClickListener(new View.OnClickListener() {
9.         @Override
10.           public void onClick(View v) {
11.
   mainActivity.startPhotoActivity(photos.get(position));
12.           }
13.       });
14.   }
```

O código acima mostra a implementação de onBindViewHolder. A função preenche o ImageView com a foto correspondente. Nas linhas 4 e 5 são obtidos as dimensões que a imagem vai ter na lista (valores salvos em res->values->dimen.xml). A linha 6 carrega a imagem em um Bitmap ao mesmo tempo em que a foto é escalada para casar com os tamanhos definidos para o ImageView. Na linha 7 o Bitmap é setado no ImageView. Nas Linhas de 8 a 13 é definido o que acontece quando o usuário clica em cima de uma imagem: a app navega para PhotoActivity, cuja função é exibir a foto e tamanho ampliado.

- Configure o RecycleView (código abaixo).

```
1. public class MainActivity extends AppCompatActivity {
2.
3.     List<String> photos = new ArrayList<>();
4. ...
5.
6.     MainAdapter mainAdapter;
7.
8.     @Override
9.     protected void onCreate(Bundle savedInstanceState) {
10.          super.onCreate(savedInstanceState);
11.          setContentView(R.layout.activity_main);
12.
13. ...
14.
```

```
15.          File dir =
   getExternalFilesDir(Environment.DIRECTORY_PICTURES);
16.          File[] files = dir.listFiles();
17.          for(int i = 0; i < files.length; i++) {
18.              photos.add(files[i].getAbsolutePath());
19.          }
20.
21.          mainAdapter = new MainAdapter(MainActivity.this,
   photos);
22.
23.          RecyclerView rvGallery =
   findViewById(R.id.rvGallery);
24.          rvGallery.setAdapter(mainAdapter);
25.
26.          float w =
   getResources().getDimension(R.dimen.itemWidth);
27.          int numberOfColumns =
   Utils.calculateNoOfColumns(MainActivity.this, w);
28.          GridLayoutManager gridLayoutManager = new
   GridLayoutManager(MainActivity.this, numberOfColumns);
29.          rvGallery.setLayoutManager(gridLayoutManager);
30.
31.      }
```

No código acima, as linhas de 15 a 19 acessam o diretório “**Pictures**” (**Enviroment.PICTURES**) (linha15), leem a lista de fotos já salvas (linha 16) e as adicionam na lista de fotos (linhas 17 a 19). Nas linhas 21 a 24 é criado o MainAdapter e ele é setado no RecycleView. As linhas 26 e 27 calculam quantas colunas de fotos cabem na tela do celular. As linhas 28 a 29 configuram o RecycleView para exibir as fotos em GRID, respeitando o número máximo de colunas calculado nas linhas 26 e 27.

Passo 10 - Navegação entre Telas

Quando o usuário clica em uma das fotos da lista em MainActivity, ele deve ser redirecionado para PhotoActivity. Neste momento, MainActivity envia também para PhotoActivity qual foto ele deve exibir. PhotoActivity, por sua vez, recebe via Intent o caminho da foto e a carrega no ImageView.

- Adicione o método abaixo em **MainActivity**. Ele recebe como parâmetro qual foto deverá ser aberto por **PhotoActivity** (linha 1). Esse método é chamado dentro do método **onBindViewHolder** (**MainAdapter**) quando o usuário clica em uma foto. O caminho para a foto é passado para **PhotoActivity** via Intent (linha 3).

```
1. public void startPhotoActivity(String photoPath) {
2.     Intent i = new Intent(MainActivity.this,
   PhotoActivity.class);
3.     i.putExtra("photo_path", photoPath);
```

```
4.     startActivity(i);
5. }
```

- Em **PhotoActivity**, obtenha o caminho da foto que foi envia via o **Intent** de criação (linha 11 e 12), carregue a foto em um Bitmap (linha 14) e sete o Bitmap no ImageView (linhas 15 e 16):

```
1. public class PhotoActivity extends AppCompatActivity {
2.
3.     String photoPath;
4.
5.     @Override
6.     protected void onCreate(Bundle savedInstanceState) {
7.         super.onCreate(savedInstanceState);
8.         setContentView(R.layout.activity_photo);
9.             . . .
10.
11.            Intent i = getIntent();
12.            photoPath = i.getStringExtra("photo_path");
13.
14.            Bitmap bitmap = Utils.getBitmap(photoPath);
15.            ImageView imPhoto = findViewById(R.id.imPhoto);
16.            imPhoto.setImageBitmap(bitmap);
17.
18.     }
```

Passo 11 - Como Usar a Câmera do Celular

A câmera do Android usa uma app própria. Para tirar uma foto, nossa app realiza as seguintes etapas:

1. Cria um arquivo vazio dentro da pasta “Pictures”. Esse arquivo vai guardar a foto que será tirada pela câmera;
2. Obtenha um endereço URI para o arquivo criado;
3. Inicie a app de câmera e envia para ela, via Intent, o URI gerado no passo anterior;
4. A app de câmera irá tirar a foto;
5. Caso o usuário confirme a foto tirada, a app de câmera salva a foto no arquivo que foi criado na etapa 1. A app de câmera consegue escrever neste arquivo pois a nossa app deu permissão via FileProvider;
6. Caso o usuário não confirme a foto tirada, o arquivo criado na etapa 1 é excluído.

- Acrescente os códigos abaixo em MainActivity:

```
1. public class MainActivity extends AppCompatActivity {
2.
3.     static int RESULT_TAKE_PICTURE = 1;
4.
5.     String currentPhotoPath;
```

```
6.
7. ...
8.
9.    private void dispatchTakePictureIntent() {
10.           File f = null;
11.           try {
12.               f = createImageFile();
13.           } catch (IOException e) {
14.               Toast.makeText(MainActivity.this, "Não foi
   possível criar o arquivo", Toast.LENGTH_LONG).show();
15.               return;
16.           }
17.
18.           currentPhotoPath = f.getAbsolutePath();
19.
20.           if(f != null) {
21.               Uri fUri =
   FileProvider.getUriForFile(MainActivity.this,
   "trindade.daniel.galeria.fileprovider", f);
22.               Intent i = new
   Intent(MediaStore.ACTION_IMAGE_CAPTURE);
23.               i.putExtra(MediaStore.EXTRA_OUTPUT, fUri);
24.               startActivityForResult(i, RESULT_TAKE_PICTURE);
25.           }
26.       }
27.
28.       private File createImageFile() throws IOException {
29.           String timeStamp = new
   SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
30.           String imageFileName = "JPEG_" + timeStamp;
31.           File storageDir =
   getExternalFilesDir(Environment.DIRECTORY_PICTURES);
32.           File f = File.createTempFile(imageFileName, ".jpg",
   storageDir);
33.           return f;
34.       }
35.
36.       @Override
37.       protected void onActivityResult(int requestCode, int
   resultCode, @Nullable Intent data) {
38.           super.onActivityResult(requestCode, resultCode,
   data);
39.           if(requestCode == RESULT_TAKE_PICTURE) {
40.               if(resultCode == Activity.RESULT_OK) {
41.                   photos.add(currentPhotoPath);
42.
   mainAdapter.notifyItemInserted(photos.size()-1);
43.               }
```

```
44.                 else {
45.                     File f = new File(currentPhotoPath);
46.                     f.delete();
47.                 }
48.             }
49.         }
50.
51. ...
52.
53.     }
```

No código acima:

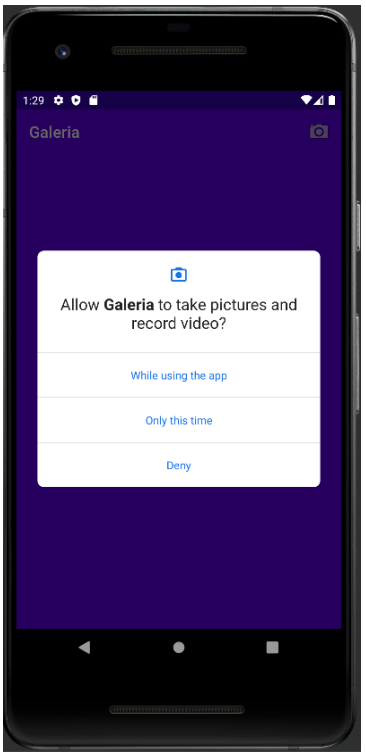
- As linhas de 9 a 26 contém o método que dispara a app de câmera;
    - Primeiramente, é criado um arquivo vazio dentro da pasta Pictures (linhas 10 a 16). Caso o arquivo não possa ser criado, é exibida uma mensagem para o usuário e o método retorna;
    - A criação em si do arquivo que vai guardar a imagem é feita pelo método createImageFile (linha 28). Esse método utiliza a data e hora para criar um nome de arquivo diferente para cada foto tirada. Isso garante que fotos tiradas no futuro sejam salvas em cima de fotos já tirada anteriormente;
    - Uma vez criado o arquivo, o local do mesmo é salvo no atributo de classe currentPhotoPath (linha 18. Esse atributo guarda somente o local do arquivo de foto que está sendo manipulado no momento;
    - Em seguida, é gerado um endereço URI para o arquivo de foto (linha21). Um Intent para disparar a app de câmera é criado (linha 22) e o URI é passado para a app de câmera via Intent (linha 23). Por último, a app de câmera é efetivamente iniciada e a nossa app fica a espera do resultado, no caso a foto (linha 24);
- Depois que a app de câmera retorna para a nossa aplicação, o método onActivityResult é chamado (linha 37);
    - Caso a foto tenha sido tirada (linha 40), o local dela é adicionado na lista de fotos (linha 41) e o MainAdapter é avisado (linha 42) de que uma nova foto foi inserida na lista e, portanto, o RecycleView deve ser atualizado também;
    - Caso a foto não tenha sido tirada (linha 44), o arquivo criado para conter a foto é excluído (linhas 45 e 46)

Passo 12 - Configurando as Permissões Necessárias

Quando uma app precisa usar recursos que são compartilhados com as outras apps do celular, o desenvolvedor precisa pedir explicitamente que o usuário permita que a app use tal recurso.

O pedido de permissão é feito através de um um dialog onde o usuário pode confirmar ou não a permissão de uso para um determinado recurso (figura abaixo). Exemplos de permissão que exigem tal procedimentos são aqueles ligados a hardware ou dados sensíveis do usuário. A seguir podemos citar alguns exemplos:

- Câmera;
- Localização/GPS
- Lista de contatos



Na nossa app temos que pedir permissão para usar o recurso de câmera do celular. O código envolvido nisso é quase sempre igual e pode, portanto, ser reutilizado sempre que for necessário pedir por alguma permissão.

Existem dois momentos possíveis em que a app pode pedir permissão para o usuário:

1. No momento em que o recurso precisa ser utilizado pela primeira vez;
2. No momento em que a app abre pela primeira vez e todas as permissões necessárias são pedidas de uma só vez.

Em nossa app optamos por usar a segunda opção.

- Em MainActivity adicione os métodos abaixo.

```
1.  public class MainActivity extends AppCompatActivity {
2.
3.      . . .
4.
5.      static int RESULT_REQUEST_PERMISSION = 2;
6.
7.      . . .
8.
9.      private void checkForPermissions(List<String> permissions) {
10.         List<String> permissionsNotGranted = new ArrayList<>();
11.
12.         for(String permission : permissions) {
13.             if( !hasPermission(permission)) {
14.                 permissionsNotGranted.add(permission);
```

```
15.             }
16.         }
17.
18.         if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
19.             if(permissionsNotGranted.size() > 0) {
20.                 requestPermissions(permissionsNotGranted.toArray(new
    String[permissionsNotGranted.size()]),RESULT_REQUEST_PERMISSION);
21.             }
22.         }
23.     }
24.
25.     private boolean hasPermission(String permission) {
26.         if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
27.             return
    ActivityCompat.checkSelfPermission(MainActivity.this, permission) ==
    PackageManager.PERMISSION_GRANTED;
28.         }
29.         return false;
30.     }
31.
32.     @Override
33.     public void onRequestPermissionsResult(int requestCode, @NonNull
    String[] permissions, @NonNull int[] grantResults) {
34.         super.onRequestPermissionsResult(requestCode, permissions,
    grantResults);
35.
36.         final List<String> permissionsRejected = new ArrayList<>();
37.         if(requestCode == RESULT_REQUEST_PERMISSION) {
38.
39.             for(String permission : permissions) {
40.                 if(!hasPermission(permission)) {
41.                     permissionsRejected.add(permission);
42.                 }
43.             }
44.         }
45.
46.         if(permissionsRejected.size() > 0) {
47.             if(Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
48.
    if(shouldShowRequestPermissionRationale(permissionsRejected.get(0)))
    {
49.                     new AlertDialog.Builder(MainActivity.this).
50.                             setMessage("Para usar essa app é preciso
    conceder essas permissões").
51.                             setPositiveButton("OK", new
    DialogInterface.OnClickListener() {
52.                                 @Override
53.                                 public void onClick(DialogInterface
    dialog, int which) {
54.
    requestPermissions(permissionsRejected.toArray(new
    String[permissionsRejected.size()]), RESULT_REQUEST_PERMISSION);
55.                                 }
```

```
56.                                 }).create().show();
57.                     }
58.                 }
59.             }
60.     }
61.}
```

No código acima há 3 métodos:

- **checkForPermissions** (linha 9) -> aceita como entrada uma lista de permissões. Nas linhas 12 a 16 cada permissão é verificada. Caso o usuário não tenha ainda confirmado uma permissão (linha 13), esta é posta em uma lista de permissões não confirmadas ainda (linha 14). A seguir, as permissões não concedidas são requisitadas ao usuário (linha 20) através da chamada do método checkSelfPermission. É este método que exibe o dialog ao usuário pedindo que ele confirme a permissão de uso do recurso.
- **hasPermission** (linha 25) -> verifica se uma determinada permissão já foi concedida ou não.
- **onRequestPermissionResult** (linha 33) -> esse método é chamado após o usuário conceder ou não as permissões requisitadas. Para cada permissão é verificado se a mesma foi concedida ou não (linhas 39 a 44). Caso ainda tenha alguma permissão que não foi concedida (linha 46) e ela é necessária para o funcionamento correto da app, então é exibida uma mensagem ao usuário informando que a permissão é realmente necessária (linhas 48 a 53) e novamente são requisitadas as permissões (linhas 54 a 56).

Uma vez adicionados os métodos, tudo o que temos que fazer é chamá-los pedindo pelas permissões necessárias. No nosso caso, a permissão para o uso da câmera.

- Adicione as linhas 8 a 11 do código abaixo no método onCreate de MainActivity.

```
1. @Override
2. protected void onCreate(Bundle savedInstanceState) {
3.     super.onCreate(savedInstanceState);
4.     setContentView(R.layout.activity_main);
5.
6.  ...
7.
8.     List<String> permissions = new ArrayList<>();
9.     permissions.add(Manifest.permission.CAMERA);
10.
11.        checkForPermissions(permissions);
12.
13. ...
14.
15.    }
```

Por último, é necessário declarar no arquivo AndroidManifest quais permissões e recursos nossa app usa.

- No arquivo AndroidManifest.xml adicione as linhas 5 e 6 do código abaixo.

```
1. <?xml version="1.0" encoding="utf-8"?>
2. <manifest
   xmlns:android="http://schemas.android.com/apk/res/android"
3.     package="com.example.galeria">
4.
5.     <uses-feature android:name="android.hardware.camera"
   android:required="true"/>
6.     <uses-permission android:name="android.permission.CAMERA"
   />
7.
8.  ...
9.
10.   </manifest>
```

Passo 13 - Compartilhar Foto com Outras Apps

Em PhotoActivity uma foto é exibida de forma ampliada. O usuário também pode compartilhar essa foto com outras apps do celular através do botão de compartilhar, localizado na ToolBar.

Para compartilhar uma foto, temos que antes gerar um URI para a foto. Esse URI será usado pelas outras apps para conseguir acessar a foto que pertence a nossa app (linha 3). Em seguida, criamos um Intent implícito do tipo ACTION_SEND, indicando que queremos enviar algo para qualquer app que seja capaz de aceitar o envio (linha 4). No Intent dizemos qual arquivo estamos querendo compartilhar (linha 5) e que tipo de dado o arquivo é (linha 6). Por último executamos o Intent (linha 7). O efeito disso é que o usuário recebe uma lista de apps com as quais ele pode compartilhar a foto.

- Adicione o método abaixo em PhotoActivity.

```
1. void sharePhoto() {
2.     // Codigo para cpmpartiilhar a foto
3.     Uri photoUri = FileProvider.getUriForFile(PhotoActivity.this,
   "trindade.daniel.galeria.fileprovider", new File(photoPath));
4.     Intent i = new Intent(Intent.ACTION_SEND);
5.     i.putExtra(Intent.EXTRA_STREAM, photoUri);
6.     i.setType("image/jpeg");
7.     startActivity(i);
8. }
```

Passo 14 - Teste

Teste sua aplicação, veja se ela possui problemas e os corrija antes de enviar para o professor.

Passo 15 - Envio da tarefa

Realize o commit e o push da sua app para o GitHub. **Envie no AVA somente o link para o GitHub!**