



# Aula 04

## Câmera, Galeria e Bibliotecas de Terceiros

Prof. Gabriele Dani

# Roteiro

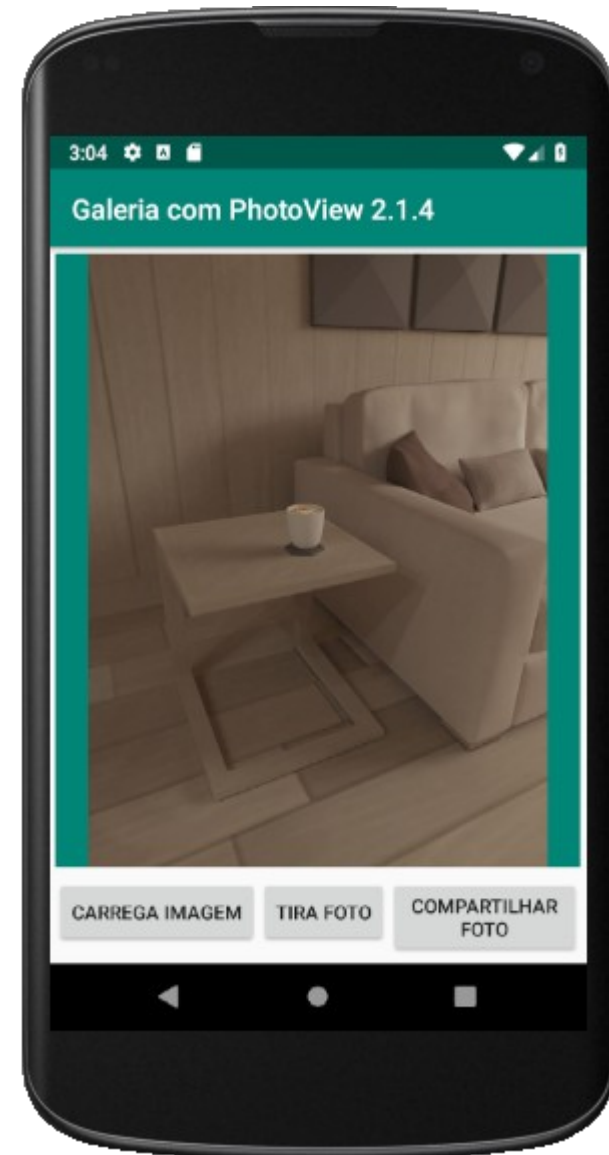
- Acesso à arquivos armazenados no dispositivo
- `startActivityResult(...)`
- Exibição de Imagens (ImageView)
- Densidades de tela do Android
- Uso de Bibliotecas de Terceiros
- Ativação e uso da Câmera
- Exercícios

# Aplicação Exemplo

- Para explicar os conceitos, vamos acompanhar o desenvolvimento de uma pequena aplicação
- O objetivo da mesma é localizar fotos no dispositivo e compartilhá-las, e permitir tirar a foto na hora se desejado

# Aplicação Exemplo

- A aplicação terá a seguinte aparência:



# Criação do projeto

- Crie um novo projeto, com uma Empty Activity
- Para esse projeto, usaremos uma combinação de ***LinearLayout*** e ***GridLayout***
- Substitua o conteúdo do arquivo ***activity\_mail.xml*** pelo conteúdo a seguir:

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:layout_weight="0"
    android:orientation="vertical">

    <GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:id="@+id/GridLayout1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:columnCount="3"
        android:orientation="horizontal"
        android:rowCount="8"
        android:useDefaultMargins="true"
        tools:context=".MainActivity">

        <!-- Aqui vai a imagem -->

        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_columnSpan="3">

            <!-- Aqui vão os botões -->

        </LinearLayout>
    </GridLayout>

</LinearLayout>

```

# Elementos da Activity

- Em ***activity\_main.xml***, deverão ser inseridos um campo `ImageView` e 3 botões :
  - Um para carregar imagens;
  - Outro para tirar fotos
  - E outro para compartilhar imagens

# ImageView

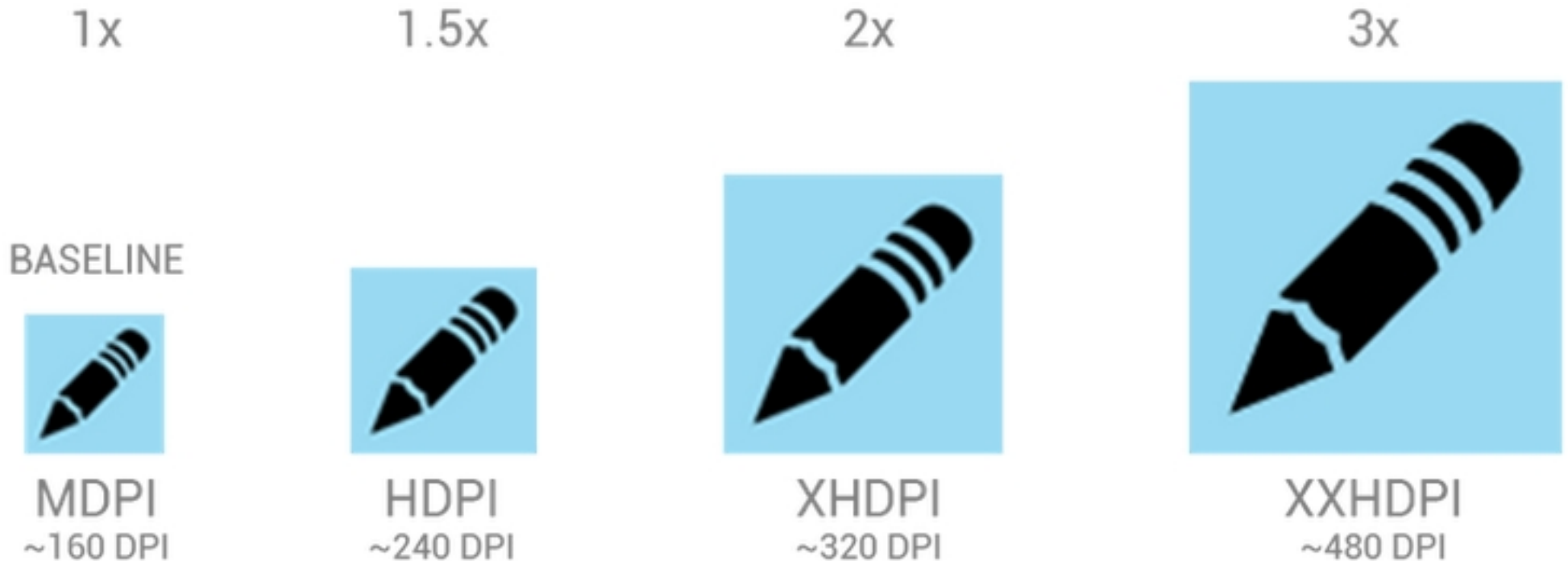
- O componente ***ImageView*** é usado para apresentar uma imagem em um aplicativo Android.
- Ao colocá-lo na ***Activity***, você deve definir o tamanho que o mesmo ocupará. Isso pode ser feito através das **unidades de medida** ou dos valores de propriedade como ***match\_parent*** e ***wrap\_content***



# Unidades de Medida do Android

<b>px</b>	Pixels	Corresponde aos pixels reais na tela
<b>In</b>	Polegadas (Inches)	Baseada no tamanho físico da tela (1 Polegada = 2,54 centímetros)
<b>mm</b>	Milímetros	Baseada no tamanho físico da tela
<b>pt</b>	Pontos (Points)	1/72 avos de uma polegada baseada no tamanho físico da tela
<b>dp</b> ou <b>dip</b>	Pixels independentes de densidade  (Density-independent Pixels)	Uma unidade abstrata que é baseada na densidade física da tela. Essas unidades são relativas a uma tela de 160 dpi, portanto 1 dp é igual a 1 px em uma tela de 160 dpi. A razão de dp para pixel muda com a densidade da tela, mas não necessariamente em proporção direta. O Android aceita os termos “dp” e “dip”, embora “dp” seja mais consistente com “sp”.
<b>sp</b>	Pixels independentes de escala  (Scale-independent Pixels)	É semelhante a unidade “dp”, mas pode ter a escala aplicada pelo tamanho de fonte selecionado pelo usuário. Seu uso é recomendado quando especifica-se tamanhos de fontes, de maneira que estes possam ser ajustados pelo tamanho de tela e pelas preferências do usuário

# Densidades de Tela



# match\_parent e wrap\_content

- Para garantir que o layout esteja flexível e adapte-se a diferentes tamanhos de tela, use ***"wrap\_content"*** e ***"match\_parent"*** para a largura e a altura de alguns componentes de exibição.

# match\_parent e wrap\_content

- ***match\_parent***: A View deve ocupar o espaço igual ao da sua Parent View (View Pai). Em qualquer dimensão(altura e/ou largura)
- ***fill\_parent***: Mesma função do match\_parent, porém era utilizada no android 2.1, quando foi feito o update para o 2.3, foi depreciada em uso do match\_parent.
- ***wrap\_content***: A View deve ocupar apenas o espaço que necessitar(altura e/ou largura) para exibir suas informações no layout.

# ImageView é limitado

- O componente ***ImageView*** realizará a exibição da imagem na tela e só.
- Se você quiser aplicar zoom sobre a imagem, este deve ser implementado manualmente.
- Para isso, usa-se bibliotecas de terceiros. Uma delas é a ***PhotoView***

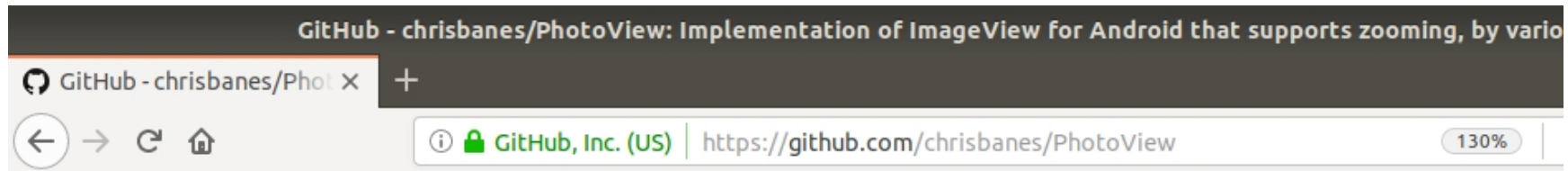
# Usando PhotoView

- ***PhotoView*** é uma subclasse de ***ImageView***, desenvolvida por Chris Banes, e disponível em <https://github.com/chrisbanes/PhotoView>
- Ela adiciona ao ***ImageView*** *zoom* controlado por gestos, além de *scroll* (rolagem) e notificação de alterações da proporção para a aplicação.

# Adicionando bibliotecas externas

- Para adicionar bibliotecas externas, temos que buscar, na documentação da biblioteca, os caminhos a partir de onde a mesma deve ser importada

# Adicionando bibliotecas externas

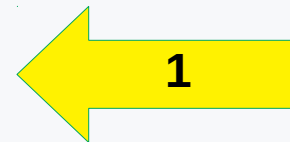


JitPack 2.3.0

## Dependency

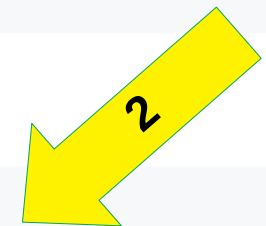
Add this in your root `build.gradle` file (**not** your module `build.gradle` file):

```
allprojects {  
    repositories {  
        maven { url "https://jitpack.io" }  
    }  
}
```



Then, add the library to your module `build.gradle`

```
dependencies {  
    implementation 'com.github.chrisbanes:PhotoView:latest.release.here'  
}
```

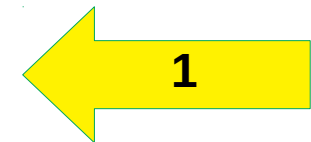




# Ajustando os arquivos do Gradle

- No arquivo ***build.gradle*** do ***projeto***, localize a seção ***allprojects*** e insira nela a linha indicada com a seta 1


```
allprojects {  
    repositories {  
        google()  
        jcenter()  
        maven {url "https://jitpack.io"}  
    }  
}
```



# Ajustando os arquivos do Gradle

- No arquivo ***build.gradle*** do módulo, localize a seção *dependencies* e insira nela a linha indicada com a seta 2

```
dependencies {  
    implementation fileTree(dir: 'libs', include: ['*.jar'])  
    implementation 'com.android.support:appcompat-v7:28.0.0'  
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'  
    testImplementation 'junit:junit:4.12'  
    androidTestImplementation 'com.android.support.test:runner:1.0.2'  
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'  
    implementation 'com.github.chrisbanes:PhotoView:2.1.4'  
}
```



- A versão atual da biblioteca é 2.3.0, mas usamos 2.1.4 por questão de compatibilidade

# Usando a biblioteca

- Se você chegou a inserir uma *ImageView*, substitua seu código em ***activity\_main.xml*** por:

```
<com.github.chrisbanes.photoview.PhotoView  
    android:id="@+id/pv_image"  
    android:layout_width="324dp"  
    android:layout_height="439dp"  
    android:layout_rowSpan="7"  
    android:layout_columnSpan="3"  
    android:layout_gravity="fill_horizontal"  
    android:background="@color/colorPrimary"  
    android:scaleType="centerInside" />
```

# Programando os botões

- Agora que temos o ***PhotoView*** e os botões na tela, chega a hora de programar os ***Intents*** dos botões
- O primeiro botão é o que busca imagens no dispositivo.
- Ele usará o método ***startActivityForResult(...)***

# Método (ação) buscar(...)

- Note aqui o acesso ao provider MediaStore:

```
public void buscar(View view) {  
    Intent intent = new Intent(Intent.ACTION_PICK,  
                                android.provider.MediaStore.Images.Media.EXTERNAL_CONTENT_URI);  
    startActivityForResult(intent, GALERIA_IMAGENS);  
}
```

# startActivityResult(...)

- O método ***startActivityResult()*** inicia uma atividade da qual se espera uma resposta.
- No nosso exemplo, ele solicitará a abertura da galeria para a seleção da imagem, e ficará aguardando que o usuário selecione uma imagem.
- Devemos programar a ação a ser tomada após o retorno da atividade. Isso é feito com o método ***onActivityResult(...)***

# onActivityResult(...)

@Override

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
    super.onActivityResult(requestCode, resultCode, data);  
    if (resultCode == RESULT_OK && requestCode == GALERIA_IMAGENS) {  
        Uri selectedImage = data.getData();  
        String[] filePath = {MediaStore.Images.Media.DATA};  
        Cursor c = getContentResolver().query(selectedImage, filePath, null,  
            null, null);  
        c.moveToFirst();  
        int columnIndex = c.getColumnIndex(filePath[0]);  
        String picturePath = c.getString(columnIndex);  
        c.close();  
        arquivoFoto = new File(picturePath);  
        mostraFoto(arquivoFoto.getAbsolutePath());  
    }  
}
```

- GALERIA\_IMAGENS é uma constante que definimos no código para identificar a Intent de origem da requisição

```
private final int GALERIA_IMAGENS = 1;
```

# Método buscar(...)

- O método **buscar** abre a atividade que permite a seleção da imagem.
- A imagem retornada é tratada no método **onActivityResult(...)**. É nele que é invocado broadcast que solicita a seleção da foto a partir da galeria e retorna sua URI (Uniform Resource Identifier).
- Através da URI manda-se mostrar a foto



# Acesso a arquivos

- Para acessar arquivos, deve ser solicitada a permissão no arquivo ***AndroidManifest.xml***:

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>  
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

# Carregando imagens

- Para buscar fotos no dispositivo, solicitou-se uma ação de **PICK**, para um **provider** (*MediaStore*)
  - **PICK?**
  - **Provider?**

# ACTION\_PICK

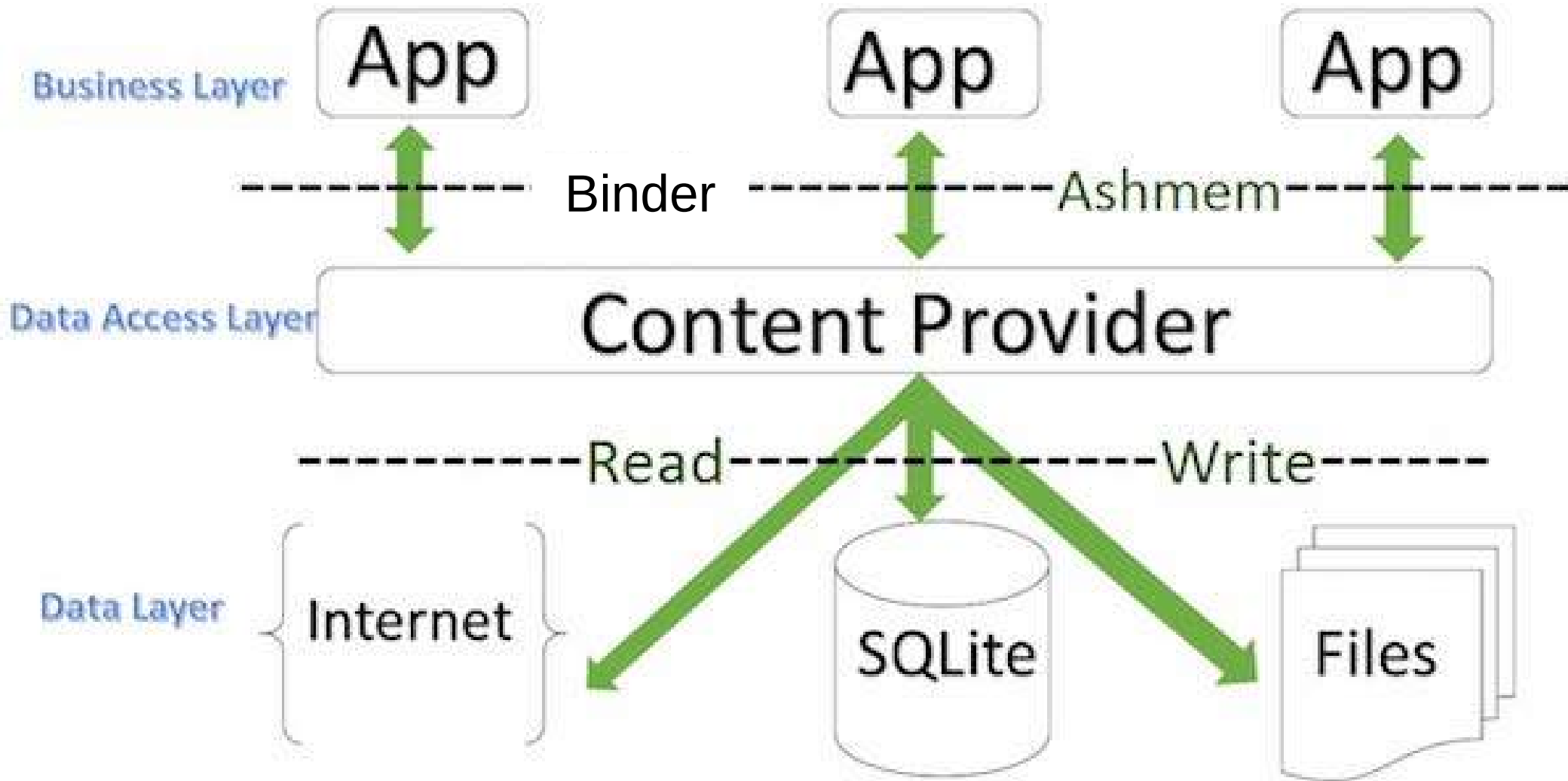
- ***Intent.ACTION\_PICK*** : A classe ***Intent*** tem uma vasta lista de Actions, cada uma com uma finalidade.
- A ação de selecionar algo no dispositivo é a ***ACTION\_PICK***
- ***Procure na documentação por todas as Actions disponíveis!***

# Providers

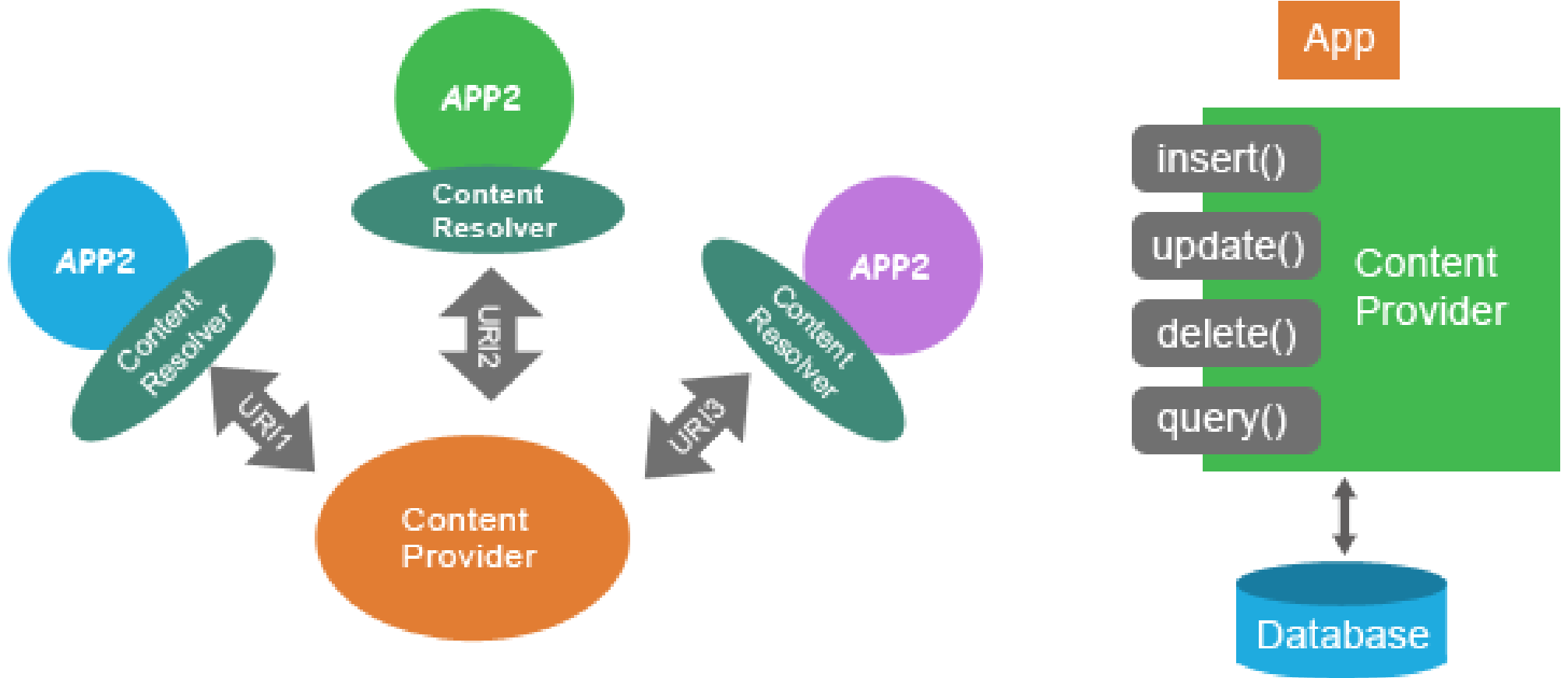
- Provedores de conteúdo gerenciam o acesso a um conjunto estruturado de dados. Eles encapsulam os dados e fornecem mecanismos para definir a segurança dos dados.
- Provedores de conteúdo são a interface padrão que conecta dados em um processo com código em execução em outro processo.
- É através de provedores(providers) que se pode acessar a agenda, lista de contatos, relação de chamadas, alarmes, mídias, configurações, dicionário e outros serviços do Android.

# Providers

Binder



# Providers – Interface Padrão



# Providers

- Não é preciso desenvolver o próprio provedor se você não pretende compartilhar seus dados com outros aplicativos.
- No entanto, precisará do próprio provedor para fornecer sugestões de pesquisa personalizada em seu aplicativo.
- Também precisará do próprio provedor se quiser copiar e colar dados complexos ou arquivos de seu aplicativo em outros aplicativos.
- A lista dos providers disponíveis no Android pode ser visualizada em :

<https://developer.android.com/reference/android/provider/package-summary.html>

# Provider no Aplicativo

- No nosso caso, o provider que será utilizado é o **nativo** do Android (**MediaStore.Images.Media.DATA**)
- O provider de mídia contém metadados para todas mídias de qualquer tipo disponíveis tanto no armazenamento interno quanto externo do dispositivo.
- Através dele acessamos as imagens armazenadas



# Provider de Arquivos

- Para ter acesso ao sistema de arquivos, cria-se, dentro da pasta **res**, uma pasta **xml**, e dentro da mesma um arquivo chamado **providers\_path.xml**, com o seguinte conteúdo:

```
<?xml version="1.0" encoding="utf-8" ?>
<paths xmlns:android="http://schemas.android.com/apk/res/android">
    <external-path name="external_files" path="."/>
</paths>
```

# Provider de Arquivos

- Esse arquivo deve ser referenciado dentro do **AndroidManifest.xml**. Copie o conteúdo abaixo para depois da declaração das activities no arquivo de manifesto:

```
<provider
    android:name="android.support.v4.content.FileProvider"
    android:authorities="${applicationId}.provider"
    android:exported="false"
    android:grantUriPermissions="true">
    <meta-data
        android:name="android.support.FILE_PROVIDER_PATHS"
        android:resource="@xml/providers_path"/>
</provider>
```

# Método mostraFoto(String caminho)

- Chamado pelos outros métodos para exibir a foto tirada ou selecionada

```
private void mostraFoto(String caminho) {  
    Bitmap bitmap =  
        BitmapFactory.decodeFile(caminho);  
    imagem.setImageBitmap(bitmap);  
}
```

# Acionando a câmera



# Como invocar a câmera?

- A ***câmera*** do dispositivo também é acionada através de uma ***Intent***.
- Mas antes de vermos o código, é necessário solicitar permissão para usar a câmera no arquivo ***AndroidManifest.xml***. Adicione:

```
<uses-feature android:name="android.hardware.camera"  
android:required="true" />
```

# Método (ação) tirarFoto(...)

```
public void tirarFoto(View view) {
    Intent takePictureIntent = new
        Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    if (takePictureIntent.resolveActivity(getPackageManager()) != null) {
        try {
            arquivoFoto = criaArquivo();
        } catch (IOException ex) {
            mostraAlerta(getString(R.string.erro), getString(
                R.string.erro_salvando_foto));
        }
        if (arquivoFoto != null) {
            Uri photoURI = FileProvider.getUriForFile(getBaseContext(),
                getBaseContext().getApplicationContext().getPackageName() +
                    ".provider", arquivoFoto);
            takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, photoURI);
            startActivityResult(takePictureIntent, CÂMERA);
        }
    }
}
```

# Método tirarFoto(...)

- A ação de tirar foto também requer uma Activity que retorne algo, no caso, uma foto. Por isso, usa-se startActivityForResult(...)
- Cria-se mais uma constante no início do código:

```
private final int CAMERA = 3;
```

- E é preciso criar um método para nomear os novos arquivos de imagens, evitando colisões:

```
criaArquivo()
```

# Método criaArquivo(...)

```
private File criaArquivo() throws IOException {  
    String timeStamp = new  
        SimpleDateFormat("yyyyMMdd_Hhmmss").format(  
            new Date());  
    File pasta = Environment.getExternalStoragePublicDirectory(  
        Environment.DIRECTORY_PICTURES);  
    File imagem = new File(pasta.getPath() + File.separator  
        + "JPG_" + timeStamp + ".jpg");  
    return imagem;  
}
```



# Retorno da Activity

- Dentro do método `startActivityForResult(...)`, que você já definiu, acrescente o seguinte código para tratar o retorno da câmera:

```
if (resultCode == RESULT_OK && requestCode == CAMERA) {  
    sendBroadcast(new Intent(  
        Intent.ACTION_MEDIA_SCANNER_SCAN_FILE,  
        Uri.fromFile(arquivoFoto))  
    );  
    mostraFoto(arquivoFoto.getAbsolutePath());  
}
```

# Compartilhando a foto tirada

- O último botão irá compartilhar a foto tirada. Para isso precisaremos de mais uma ação e um método.
- O compartilhamento será feito através de uma Intent implícita

# Ação de Compartilhar

```
public void compartilhar(View view) {  
    Uri uri = null;  
    if(arquivoFoto!=null) {  
        uri = FileProvider.getUriForFile(getBaseContext(),  
            getBaseContext().getApplicationContext()  
                .getPackageName() +  
                    ".provider", arquivoFoto);  
        if(uri!=null) {  
            compartilharImagem(uri,"image/jpg");  
        }  
    }  
}
```

# Método compartilhar(...)

```
private void compartilharImagem(Uri uri, String tipo){  
    Intent intent = new Intent(Intent.ACTION_SEND);  
    intent.putExtra(Intent.EXTRA_STREAM, uri);  
    intent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);  
    intent.setType(tipo);  
    startActivity(intent);  
}
```

# Quase lá

- A única coisa que faltou foi solicitar pelas permissões de escrita e leitura de arquivos na primeira execução do programa.
- Para isso, será necessário adicionar os dois blocos seguintes no método ***onCreate()*** da aplicação

# Permissão de Leitura

```
// Pede permissão para acessar as mídias gravadas no dispositivo
if (ContextCompat.checkSelfPermission(this,
    Manifest.permission.READ_EXTERNAL_STORAGE)
    != PackageManager.PERMISSION_GRANTED) {
    if (ActivityCompat.shouldShowRequestPermissionRationale(this,
        Manifest.permission.READ_EXTERNAL_STORAGE)) {
    } else {
        ActivityCompat.requestPermissions(this,
            new String[]{Manifest.permission.READ_EXTERNAL_STORAGE},
            PERMISSAO_REQUEST);
    }
}
```

# Permissão de Escrita

```
// Pede permissão para escrever arquivos no dispositivo
if (ContextCompat.checkSelfPermission(this,
    Manifest.permission.WRITE_EXTERNAL_STORAGE)
    != PackageManager.PERMISSION_GRANTED) {
    if (ActivityCompat.shouldShowRequestPermissionRationale(this,
        Manifest.permission.WRITE_EXTERNAL_STORAGE)) {
    } else {
        ActivityCompat.requestPermissions(this,
            new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE},
            PERMISSAO_REQUEST);
    }
}
```

# Para finalizar

- Verifique se você declarou os seguintes atributos na sua classe ***MainActivity.java***:

```
private final int GALERIA_IMAGENS = 1;  
private final int PERMISSÃO_REQUEST = 2;  
private final int CAMERA = 3;
```

```
private File arquivoFoto = null;  
private ImageView imagem;
```

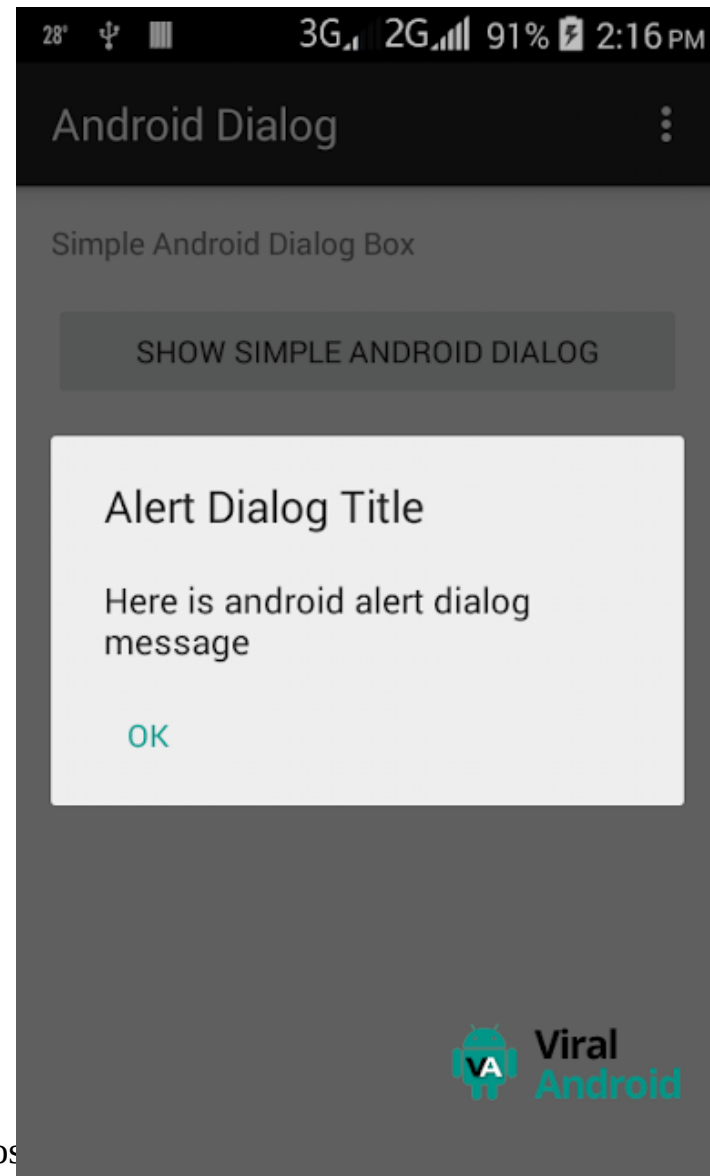
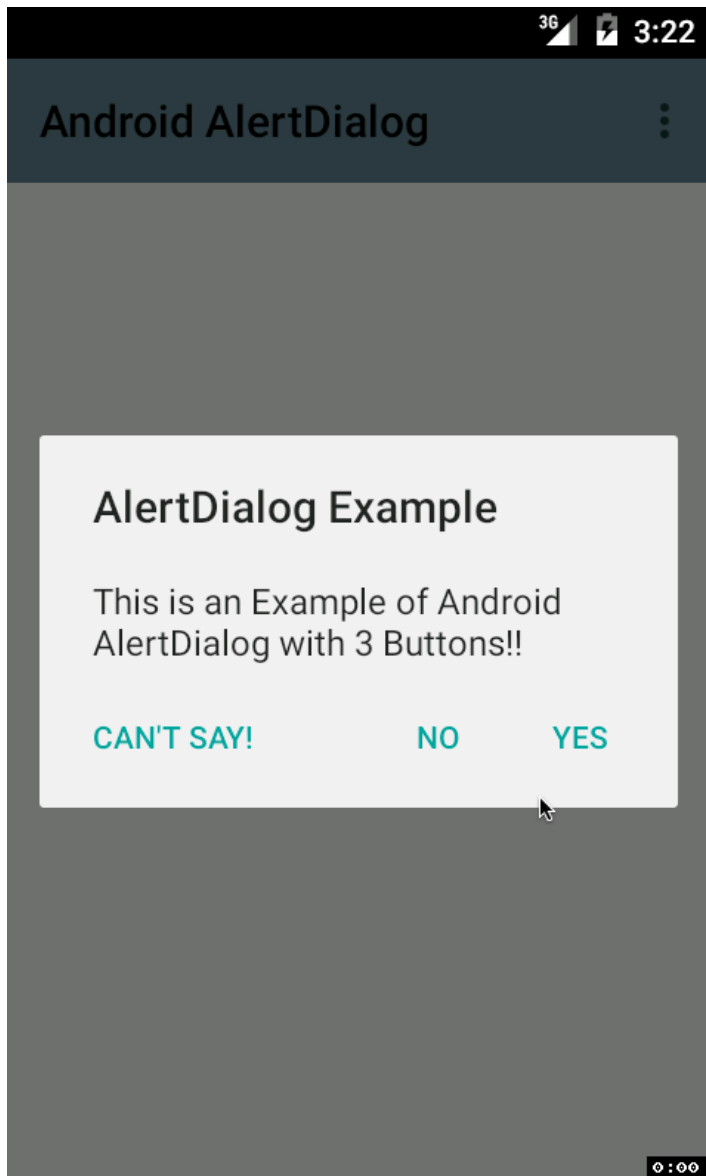


# Para finalizar

- E obtenha a referência para o componente de imagem dentro do método ***onCreate()***:

```
this.imagem = (ImageView) findViewById(R.id.pv_image);
```

# Dica : AlertDialog



# Dica : AlertDialog

- Você pode abrir uma janela modal sobreposta a atividade para mostrar alguma informação usando o componente AlertDialog. Veja o exemplo a seguir:

```
private void mostraAlerta(String titulo, String mensagem) {  
    AlertDialog alertDialog = new  
AlertDialog.Builder(MainActivity.this).create();  
    alertDialog.setTitle(titulo);  
    alertDialog.setMessage(mensagem);  
    alertDialog.setButton(AlertDialog.BUTTON_NEUTRAL,  
getString(R.string.ok),  
        new DialogInterface.OnClickListener() {  
            public void onClick(DialogInterface dialog, int which) {  
                dialog.dismiss();  
            }  
        });  
    alertDialog.show();  
}
```

# Dúvidas?



# Atividade 1

- Utilizando os códigos disponíveis nestes slides, construa a aplicação que acessa a câmera e a galeria

# Referências

- Reto Meier, Ian Lake - **Professional Android** (2018, Wrox)
- <https://developer.android.com/guide/components/services#java>
- <https://medium.com/@oieduardorabelo/desmistificando-a-densidade-de-pixel-b8ee8f7538a6>
- <https://www.youtube.com/watch?v=zhszwwkay2A&hl=pt-br>
- <http://developer.android.com/guide/topics/resources/more-resources.html#Dimension>
- <https://inducesmile.com/android-libraries/android-photoview-third-party-android-custom-view-library/>
- <https://developer.android.com/training/multiscreen/screensizes.html?hl=pt-br>