|  |
| --- |
| Android Tutorial – Part 7 |

|  |
| --- |
| 6-29-2018 |



Table of Contents

[Introduction 2](#_Toc518394075)

[Android Camera API 3](#_Toc518394076)

[Things to consider before using Camera in the app 3](#_Toc518394077)

[Camera Requirement in app 3](#_Toc518394078)

[Quick Picture or Customized Camera 3](#_Toc518394079)

[Storage 3](#_Toc518394080)

[ConstraintLayout in Android 5](#_Toc518394081)

[Constraints overview 5](#_Toc518394082)

[The view inspector 8](#_Toc518394083)

[References 17](#_Toc518394084)

# Introduction

This is the part seven of the android tutorial series. It is a continuation from last week. In order to follow this successfully, it is required to have,

* A basic understanding given about android in last session.
* The environment set up.
* The project created during last tutorial, opened in Android Studio.
* AVD or an Actual device ready for app deployment.

To catch up, in the last session (Android Tutorial Part 6),

* Using Emulator to Test Sensors
* Android Motion Sensors
  + Rotational Vector Sensor
  + Gyroscope
* Android Position Sensor
  + Orientation Sensor
* Creating a Custom View in Android
  + Using paint object
  + Using canvas
* The proximity sensor
* Android environment sensors
  + Ambient Temperature
  + Light
  + Pressure
  + Relative Humidity
* Temperature

Source code for the previous tutorial:-<https://github.com/nadee158/android_tutorial_part_6.git>

With that knowledge in hand, in this session below areas will be covered,

* Android Camera API

# Android Camera API

Most Android devices have at least one camera. Some devices have a front and a back facing camera.

The Android framework includes support for various cameras and camera features available on devices, allowing developers to capture pictures and videos in the applications.

The Android framework supports capturing images and video through the **android.hardware.camera2 API** or **camera Intent**

## Things to consider before using Camera in the app

Before enabling the application to use cameras on Android devices, we should consider about how the app intends to use this hardware feature.

### Camera Requirement in app

* Is the app totally dependent on Camera – app will not work without it?
  + If yes, the camera requirement should be declared in the manifest.

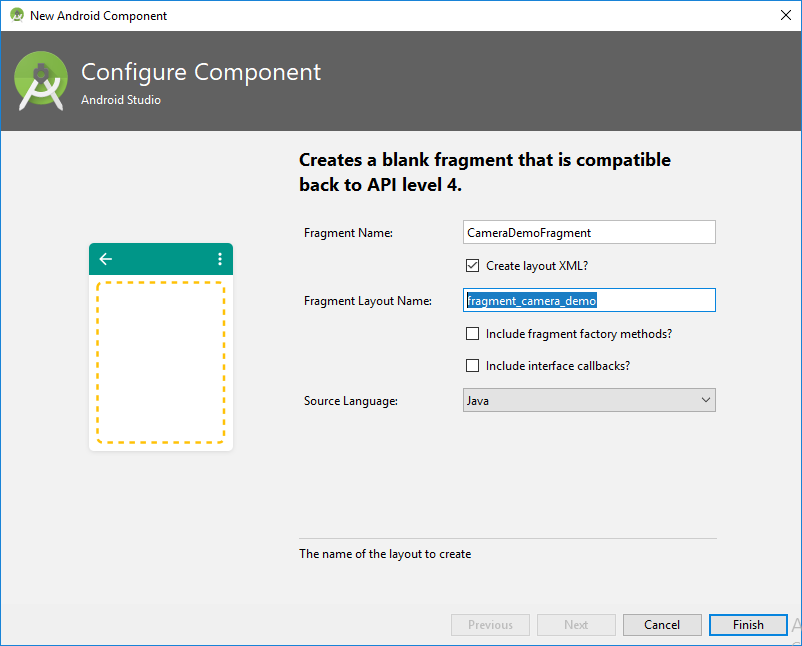
### Quick Picture or Customized Camera

* How will the application use the camera?
  + Is it for just snapping a quick picture or video clip?
    - If yes, consider Using Existing Camera Apps
  + Does the app provide a new way to use cameras?
    - If yes, build a camera using the Camera API

### Storage

The visibility, availability and security of the generated images or videos through the app

Lets now see how we could use an existing camera app to get an image to our app

1. Create a new “**Fragment**” to display the data retrieved from the Rotational Vector Sensor
   1. Right click on “**java/lk.uok.mit.fragment**”, select “**New**”🡪”**Fragment**”🡪”**Fragment (Blank)**”  
      
   2. Fill the details as shown below;  
      
      1. Fragment Name:- **CameraDemoFragment**
      2. Check create layout XML option
      3. Fragment Layout Name:- **fragment\_camera\_demo**
      4. **Uncheck both “Include fragment factory methods” and “include interface callback”**
      5. Source Language:-Java
2. Add a **menu item** to the **drawer** to access the new “**CameraDemoFragment**”
   1. Open the “**drawer\_view.xml**” inside “**res/menu**” folder and add a new menu item with id “**nav\_camera\_demo**” like below;  
      **<item  
       android:id="@+id/nav\_camera\_demo"  
       android:title="Camera Demo" />**
3. Add an entry to the switch case statement inside “**NavigationItemSelectedListener**” inside “**MainActivity**”
   1. Open “**MainActivity**” inside “**java**” folder
   2. Modify the “**setNavigationItemSelectedListene**r(” method of “**NavigationView**” inside “**onCreate**” method like shown below;  
        
      **case R.id.nav\_camera\_demo:  
       *//Open the CameraDemoFragment  
       // Begin the transaction* ft = getSupportFragmentManager().beginTransaction();  
       *// Replace the contents of the container with the new fragment* ft.replace(R.id.*fragment\_content*, new CameraDemoFragment());  
       *// Complete the changes added above* ft.commit();  
       break;**

# ConstraintLayout in Android

ConstraintLayout allows developers to create large and complex layouts with a **flat view hierarchy** (no nested view groups).

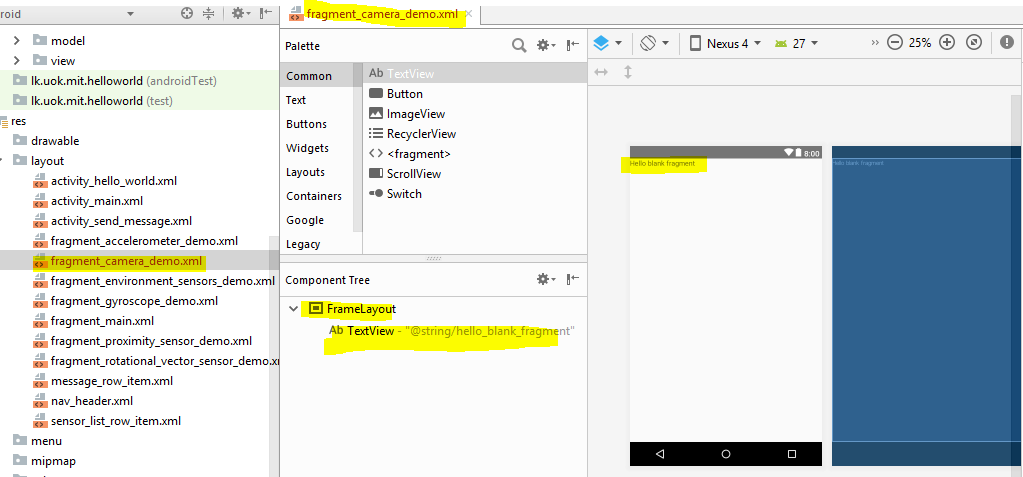
* It's **similar to RelativeLayout** in that all views are laid out according to relationships between sibling views and the parent layout,
* But it's **more flexible than RelativeLayout** and **easier to use with Android Studio's Layout Editor**.
  + All the power of ConstraintLayout is available directly from the Layout Editor's visual tools
  + The layout API and the Layout Editor were specially built for each other.
  + It is possible to build the layout with ConstraintLayout entirely by drag-and-dropping instead of editing the XML.

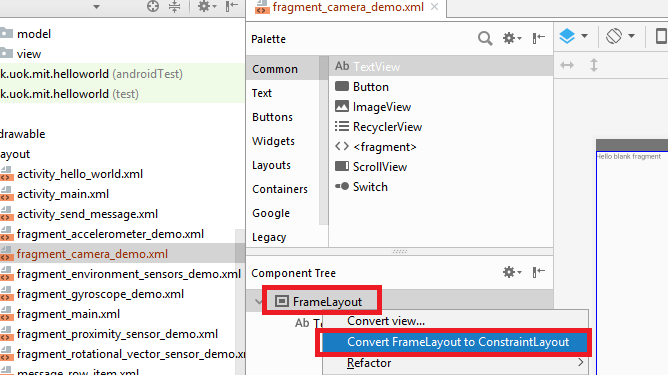
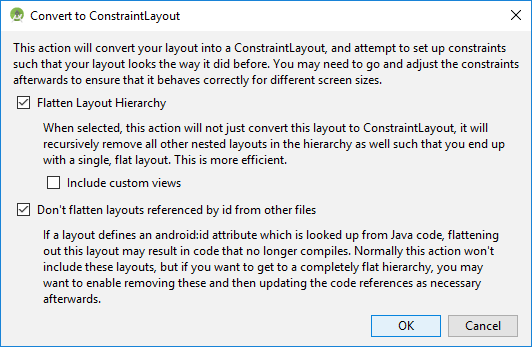
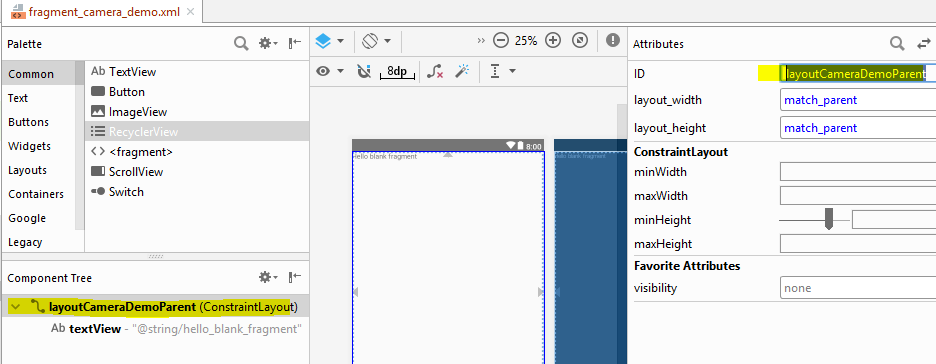
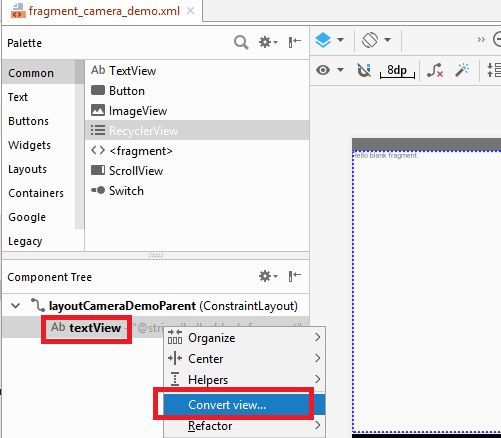
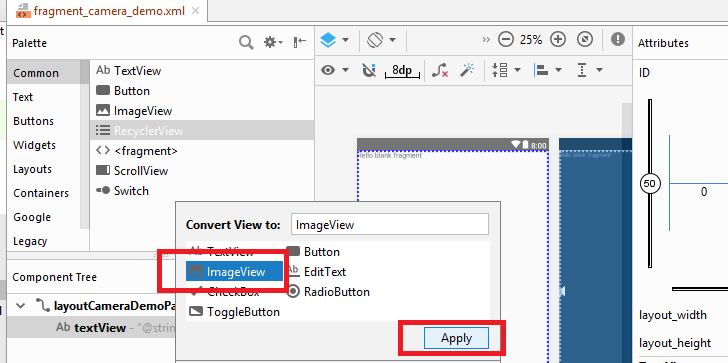
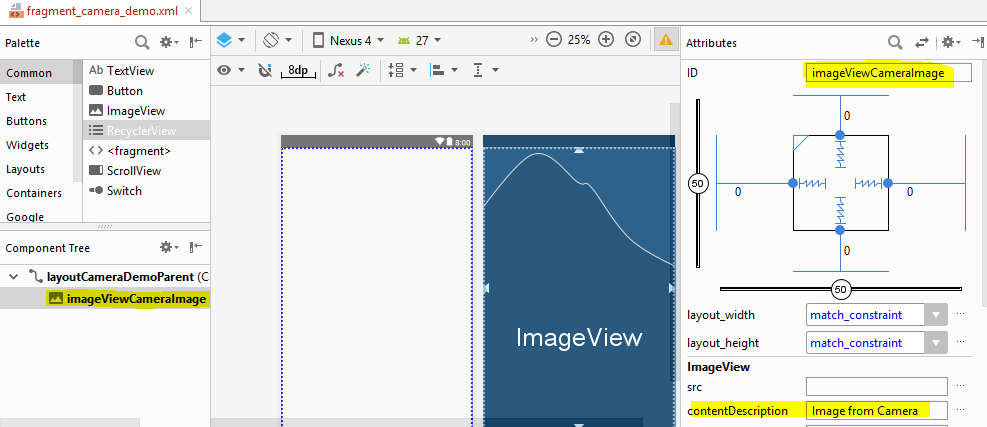
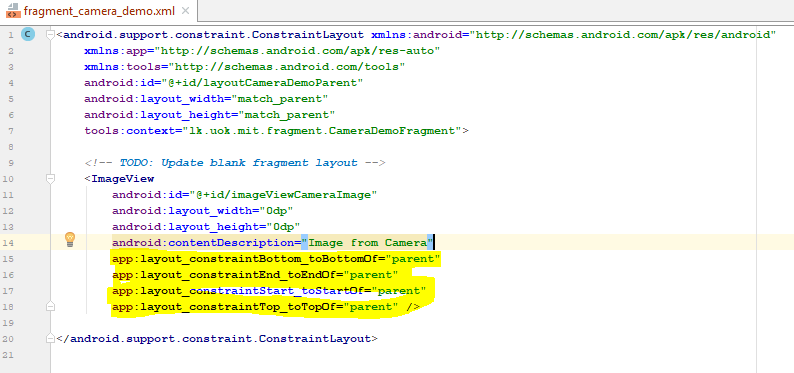
## Constraints overview

* To define a view's position in ConstraintLayout, you must add at least one horizontal and one vertical constraint for the view.
* Each constraint represents a connection or alignment to another view, the parent layout, or an invisible guideline.
* Each constraint defines the view's position along either the vertical or horizontal axis; so each view must have a minimum of one constraint for each axis, but often more are necessary.
* When you drop a view into the Layout Editor, it stays where you leave it even if it has no constraints.
* However, this is only to make editing easier; if a view has no constraints when you run your layout on a device, it is drawn at position [0,0] (the top-left corner).

Lets use a constraint layout in out fragment layout to see how to use a one in an app;

1. Modify the “**fragment\_camera\_demo.xml**” to display the image retrieved from the Camera Intent,

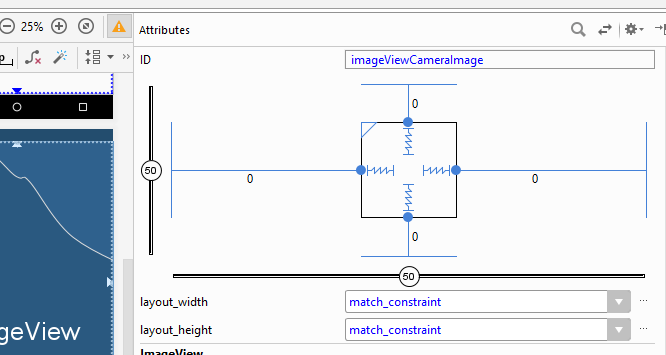
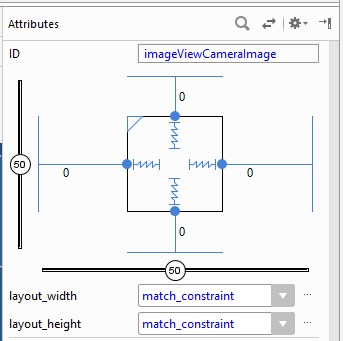
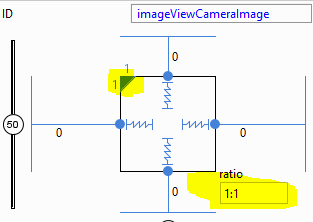
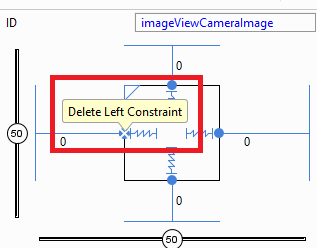
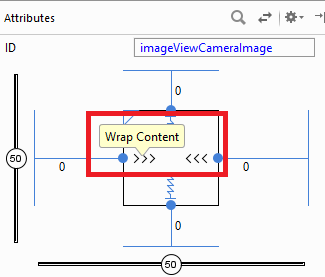
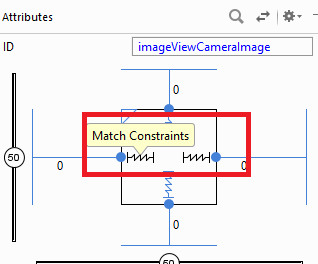
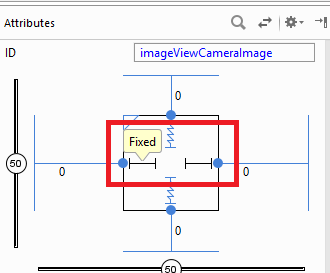
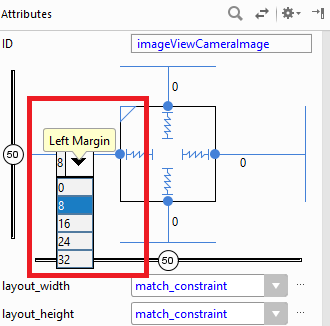
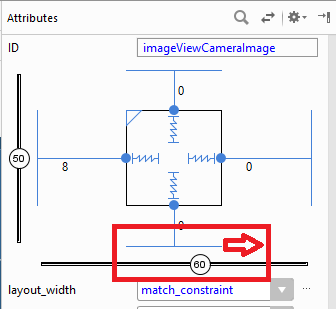
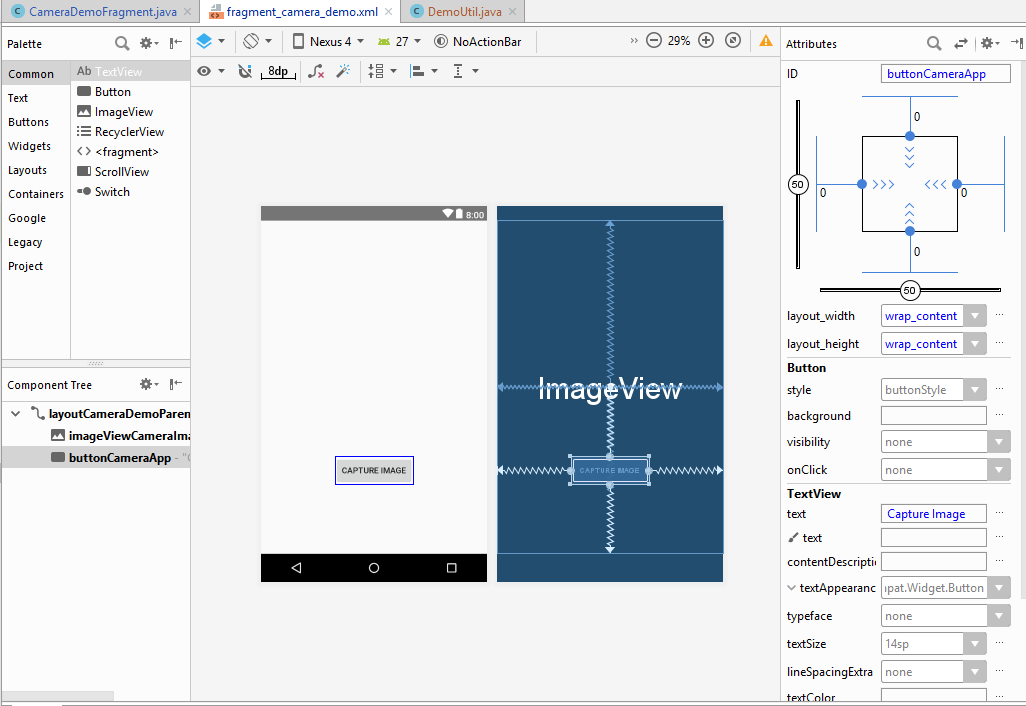
Open the **fragment\_camera\_demo.xml**  file inside “**res/layout**” folder and go to its design view, it should look like below now;  


1. Right click on the “**FrameLayout**” inside the “**Component Tree**” and select “**Convert FrameLayout to ConstraintLayout**”  
     
   1. Then the below confirm box will appear, check both checkboxes and click ok  
      
2. Change the id of the ConstraintLayout to “**layoutCameraDemoParent**” and finally, the component will look like below;  
   
3. Now to convert the “textView” to an ImageView to display the image captured from camera,
   1. Right click on “**textView**” in inside the “**Component Tree**” and select “**Convert View**”  
      
   2. Select “**ImageView**” from popup and change its id to “**imageViewCameraDemo**”  
      
   3. Change the id to “**imageViewCameraImage**” and content description to “**Image from Camera**”  
      
4. Now go to the “Text” view and check, the xml code looks like below;  
   
   1. Note the highlighted code segments, as it was mentioned above, to define a view's position in ConstraintLayout, you must add
      1. Horizontal constraint  
         LEFT:-**app:layout\_constraintStart\_toStartOf="parent"**RIGHT**:-app:layout\_constraintEnd\_toEndOf="parent"**
      2. Vertical constraint  
         TOP:- **app:layout\_constraintTop\_toTopOf="parent"**  
         BOTTOM:- **app:layout\_constraintBottom\_toBottomOf="parent"**  
           
         for the view, and each constraint represents a connection or alignment to **another view**, the **parent layout**, or **an invisible guideline**, and in our case its “PARENT LAYOUT”

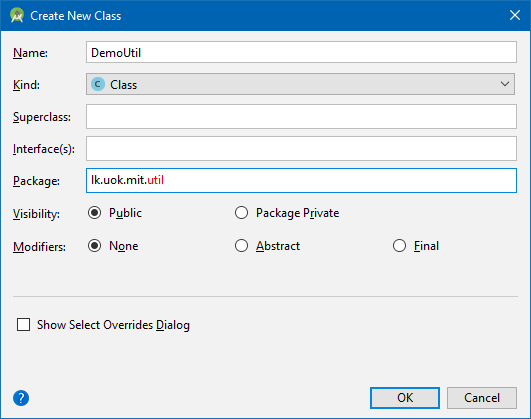
## The view inspector

Near the top of the Attributes window there is a special section called the view inspector.

* It is available only for views in a constraint layout
* It includes controls for several layout attributes, as shown in figure below.

1. Now go again to the “**Design**” view of the “**fragment\_camera\_demo.xml** ” and select “**imageViewCameraImage**” from “**Component Tree**” view and check the “**Attribute window”**;  
   
2. The Attributes window includes controls for
3. size ratio – This is to set the view size to a ratio such as 16:9, if at least one of the view dimensions is set to "match constraints" (0dp).
4. To enable the ratio, click here and then enter the width:height ratio in the input that appears.  
   
5. delete constraint – to remove the constrain  
   
6. height/width mode – to change the way the height and width are calculated, click the symbol to toggle between these settings
7. https://developer.android.com/studio/images/buttons/layout-width-fixed.png - **Fixed**: to specify a specific dimension in the text box below or by resizing the view in the editor
8. https://developer.android.com/studio/images/buttons/layout-width-wrap.png - **Wrap Content**: The view expands only as much as needed to fit its contents.
9. https://developer.android.com/studio/images/buttons/layout-width-match.png - **Match Constraints**: The view expands as much as possible to meet the constraints on each side (after accounting for the view's margins)  
      
   
10. margins – to control the margin for each view  
    
11. constraint bias - the view becomes centered between the two constraints with a bias of 50% by default
12. can adjust the bias by dragging the bias slider in the Attributes window or by dragging the view  
    
13. Add two more button to the layout, and the design view and text view should look like below;
    1. “Design” view of “**fragment\_camera\_demo.xml** ”  
       
    2. Text view of “**fragment\_camera\_demo.xml** ”  
       <**android.support.constraint.ConstraintLayout 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/layoutCameraDemoParent"  
        android:layout\_width="match\_parent"  
        android:layout\_height="match\_parent"  
        tools:context="lk.uok.mit.fragment.CameraDemoFragment"**>  
         
        <**ImageView  
        android:id="@+id/imageViewCameraImage"  
        android:layout\_width="0dp"  
        android:layout\_height="wrap\_content"  
        android:contentDescription="Image from Camera"  
        android:scaleType="fitXY"  
        app:layout\_constraintBottom\_toBottomOf="parent"  
        app:layout\_constraintEnd\_toEndOf="parent"  
        app:layout\_constraintStart\_toStartOf="parent"  
        app:layout\_constraintTop\_toTopOf="parent"** />  
         
        <**Button  
        android:id="@+id/buttonCameraApp"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:text="Capture Image"  
        app:layout\_constraintBottom\_toBottomOf="parent"  
        app:layout\_constraintEnd\_toEndOf="parent"  
        app:layout\_constraintStart\_toStartOf="parent"  
        app:layout\_constraintTop\_toBottomOf="@+id/imageViewCameraImage"** />  
         
       </**android.support.constraint.ConstraintLayout**>

After getting familiar with the constrain layout and its specific view inspector, lets get back to our main objective, to display an image captured from camera through the default camera app;

1. Add permission to write the captured image file to external storage, and to access camera by adding below lines to “**AndroidManifest.xml**” file inside “manifests” folder  
   *<!-- Permission to access memory -->*<**uses-permission android:name="android.permission.WRITE\_EXTERNAL\_STORAGE"** />  
   *<!-- Permission to access camera -->*<**uses-permission android:name="android.permission.CAMERA"** />
2. Before using the camera, we have to check if a camera is available, if the required permissions are given. In order to write these methods it is possible to use the same “**CameraDemoFragment**” class.   
   But it’s more useful to use a separate utility class to write these methods, since that can be used across many fragments/activities.  
   Let’s create a class called “**DemoUtil**” inside a package named, “**lk.uok.mit.util**”  
   1. Create a new Class, right click on “java” got to “New”-->”Java Class”.
   2. Fill the details as shown below; this class will contain utility methods required for the project  
      
      1. Name:- **DemoUtil**
      2. Kind:- **Class**
      3. Package:- **lk.uok.mit.util**
   3. Next, lets write methods required for the Camera integration, (which can be reused) inside this new class as shown below;

1. Write a common method to check if an app has a set of given user permissions;  
   Open the “**DemoUtil”** class inside “**java**” write below method
   1. Method name:- **hasUserPermissions**
   2. Return Type:- **boolean** - if at least one permission is not given, this will return false
   3. Access modifiers :- **public static**
   4. Parameters:-
      1. **Context context** – the context of which permissions are required
      2. **String[] permissions** – the list of permission to be checked
   5. Logic:-  
      *// Check whether user has all the passed permissions or not.***public static boolean** hasUserPermissions(Context context, String[] permissions) {  
       **boolean** ret = **true**;  
       *//iterate through the permissions passed* **for** (**int** i = 0; i < permissions.**length**; i++) {  
       *// check if the permission is available for each permission* **int** hasPermission = ContextCompat.*checkSelfPermission*(context, permissions[i]);  
       *//if the permission status is not equal to granted, return false* **if** (hasPermission != PackageManager.***PERMISSION\_GRANTED***) {  
       ret = **false**;  
       }  
       }  
       **return** ret;  
      }

1. Write a common method to check if an app has a set of given user permissions;  
   Open the “**DemoUtil”** class inside “**java**” write below method;
   1. Method name:- **isSystemFeatureAvailable**
   2. Return Type:- **boolean** - if the feature is not available, this will return false
   3. Access modifiers :- **public static**
   4. Parameters:-
      1. **Context context** – the context of which permissions are required
      2. **String[] permissions** – the list of permission to be checked
   5. Logic:-  
      *//a method to check if a camera is available in device***public static boolean** isSystemFeatureAvailable(Context context, String featureName) {  
       **if** (context.getPackageManager().hasSystemFeature(featureName)) {  
       *// this device has a camera* **return true**;  
       } **else** {  
       *// no camera on this device* **return false**;  
       }  
      }
2. Write a common method to get a unique file name (to save the image file)  
   Open the “**DemoUtil”** class inside “**java**” write below method;
   1. Method name:- getUniqueFileName
   2. Return Type:- **String** – the generated file name
   3. Access modifiers :- **public static**
   4. Parameters:-
      1. **String prefix** – the prefix of the file, can be used to easily recognize the files
      2. **String fileExtension** – the extension of the file

Logic:-  
**public static** String getUniqueFileName(String prefix, String fileExtension){  
 *//create a simple date format* SimpleDateFormat dateFormat = **new** SimpleDateFormat(**"yyyyMMdd\_HHmmss"**);  
 *// Preparing media file naming convention  
 //get current time* Date currentTime= Calendar.*getInstance*().getTime();  
 *// get the current time using the dateformat object* String timeStamp=dateFormat.format(currentTime);  
 *//construct a unique file name* String fileName = prefix + timeStamp + **"."** + fileExtension;  
 **return** fileName;  
}

1. Next lets a write a method which will create a file by given name in the given directory, this is to be used to store the captured image data through camera;
   1. Add the below class variables to the “**DemoUtil”** class  
      *//the unique permission request code***public static int** *UNIQUE\_APP\_REQUEST\_CODE* = 1234;  
        
      *//extension of the image file saved***public static final** String ***IMAGE\_EXTENSION*** = **"jpg"**;  
        
      *//the prefix of the image file saved***public static final** String ***IMAGE\_PREFIX*** = **"IMG\_"**;  
        
      *//the physical folder name in which the captured images are saved***public static final** String ***GALLERY\_DIRECTORY\_NAME*** = **"CameraDemo"**;
   2. Method name:- **constructOutputMediaFile**
   3. Return Type:- **File** – the generated file
   4. Access modifiers :- **public static**
   5. Parameters:-
      1. **String fileName** – the name of the generated file
   6. Logic:-  
      */\*\*  
       \* Creates and returns the image or video file before opening the camera  
       \*/***public static** File constructOutputMediaFile(String fileName) {  
       *// External sdcard location* File mediaStorageDir = **new** File(  
       Environment.*getExternalStoragePublicDirectory*(Environment.*DIRECTORY\_PICTURES*),  
       ***GALLERY\_DIRECTORY\_NAME***);  
       *// Create the storage directory if it does not exist* **if** (!mediaStorageDir.exists()) {  
       **if** (!mediaStorageDir.mkdirs()) {  
       Log.*e*(***GALLERY\_DIRECTORY\_NAME***, **"Failed to CREATE create "** + ***GALLERY\_DIRECTORY\_NAME*** + **" directory"**);  
       **return null**;  
       }  
       }  
       *//create a new file* File mediaFile = **new** File(mediaStorageDir.getPath() + File.***separator*** + fileName);  
       Log.*e*(**"GALLERY\_DIRECTORY"**, **"Failed to CREATE create "** + mediaFile);  
       **return** mediaFile;  
      }
2. In order to render the captured image, we need a method to get the URI of the generated file, this can also be written in the “**DemoUtil”** class
   1. Method name:- **getOutputMediaFileUri**
   2. Return Type:- **Uri** – the Uniform Resource Identifier of generated file
   3. Access modifiers :- **public static**
   4. Parameters:-
      1. Context context – the context
      2. File file – The file of which URI is required
   5. Logic:-  
      *//get uri of the media file***public static** Uri getOutputMediaFileUri(Context context, File file) {  
       **return** FileProvider.*getUriForFile*(context, context.getPackageName() + **".provider"**, file);  
      }
3. After capturing image, before displaying in the ImageView, it has to be downsized in order to avoid OutOfMemory exceptions, this is also written in “**DemoUtil”** class
   1. Method name:- **optimizeBitmap**
   2. Return Type:- **Bitmap** – the bitmap of the captured image
   3. Access modifiers :- **public static**
   4. Parameters:-
      1. int sampleSize – the number of pixels in either dimension that correspond to a single pixel in the decoded bitmap
      2. String filePath – The file path of the saved image
   5. Logic:-  
      */\*\*  
       \* Downsizing the bitmap to avoid OutOfMemory exceptions  
       \*/***public static** Bitmap optimizeBitmap(**int** sampleSize, String filePath) {  
       *// bitmap factory* BitmapFactory.Options options = **new** BitmapFactory.Options();  
        
       *// downsizing image as it throws OutOfMemory Exception for larger  
       // images* options.**inSampleSize** = sampleSize;  
        
       **return** BitmapFactory.*decodeFile*(filePath, options);  
      }
4. After saving the image, it should be visible in the gallery of the phone, in order for that to happen in older devices as well, we need to refresh the gallery. Below method is for that purpose and its also written in the “**DemoUtil”** class;
   1. Method name:- **refreshGallery**
   2. Return Type:- **void**
   3. Access modifiers :- **public static**
   4. Parameters:-
      1. **Context context –** the context
      2. **String filePath** – The file path of the saved image
   5. Logic:-  
      */\*\*  
       \* Refreshes gallery on adding new image/video. Gallery won't be refreshed  
       \* on older devices until device is rebooted  
       \*/***public static void** refreshGallery(Context context, String filePath) {  
       *// ScanFile so it will be appeared on Gallery* MediaScannerConnection.*scanFile*(context,  
       **new** String[]{filePath}, **null**,  
       **new** MediaScannerConnection.OnScanCompletedListener() {  
       @Override  
       **public void** onScanCompleted(String path, Uri uri) {  
       }  
       });  
      }
5. In case required to refer to, below are the complete set of imports inside “**DemoUtil”** class  
   **import** android.content.Context;  
   **import** android.content.pm.PackageManager;  
   **import** android.graphics.Bitmap;  
   **import** android.graphics.BitmapFactory;  
   **import** android.media.MediaScannerConnection;  
   **import** android.net.Uri;  
   **import** android.os.Environment;  
   **import** android.support.v4.content.ContextCompat;  
   **import** android.support.v4.content.FileProvider;  
   **import** android.util.Log;  
     
   **import** java.io.File;  
   **import** java.text.SimpleDateFormat;  
   **import** java.util.Calendar;  
   **import** java.util.Date;
6. Now the common methods are completed and lets write code to capture images in “**CameraDemoFragment**”, modify it as shown below
   1. Open the “**CameraDemoFragment**” inside “java”
   2. Modify the “**onCreateView**” method inside “**CameraDemoFragment**” and add the text to display on title bar as shown below;  
      @Override  
      **public** View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {  
       *//set the text appear in title bar* getActivity().setTitle(**"Camera Demo"**);  
       **return** inflater.inflate(R.layout.***fragment\_camera\_demo***, container, **false**);  
      }
   3. Add below class variables to “**CameraDemoFragment**” class  
      *//a code to identify the requested context of the capera app***private static final int *CAMERA\_CAPTURE\_IMAGE\_REQUEST\_CODE*** = 1;  
        
      *// Bitmap sampling size***public static final int *BITMAP\_SAMPLE\_SIZE*** = 8;  
        
      *//to hold the bitmap of the captured image***private** Bitmap **bitmap**;  
        
      *//the image view***private** ImageView **imageView**;  
        
      *//the button to capture image***private** Button **buttonCaptureImage**;  
        
      *//the constructed storage path of the captured image***private** String **imageStoragePath**;  
        
      *//the context***private** Context **context**;  
        
      *//permission array required execute the code here***private** String[] **requiredPermissions** = {Manifest.permission.***WRITE\_EXTERNAL\_STORAGE***, Manifest.permission.***CAMERA***};
7. Write a method inside “**CameraDemoFragment**” class to capture image, that is to start the camera app by passing required data to capture the image and return in to our app;
   1. Method name:- **captureImage**
   2. Return Type:- **void**
   3. Access modifiers :- **private**
   4. Parameters:- None
   5. Logic:-

*/\*\*  
 \* Capturing Camera Image will launch camera app requested image capture  
 \*/***private void** captureImage() {  
 Intent intent = **new** Intent(MediaStore.***ACTION\_IMAGE\_CAPTURE***);  
 *//get a unique name for the image file* String fileName =DemoUtil.*getUniqueFileName*(DemoUtil.***IMAGE\_PREFIX***,DemoUtil.***IMAGE\_EXTENSION***);  
 *//construct a file to store the captured image* File file = DemoUtil.*constructOutputMediaFile*(fileName);  
 **if** (file != **null**) {  
 *//if the file is created and returned, get its path* **imageStoragePath** = file.getAbsolutePath();  
 }  
 *//get the uniform resource locator of the file* Uri fileUri = DemoUtil.*getOutputMediaFileUri*(**context**, file);  
 *//put the URI in to camera intent as data, it will populate this file* intent.putExtra(MediaStore.***EXTRA\_OUTPUT***, fileUri);  
 *// start the image capture Intent* startActivityForResult(intent, ***CAMERA\_CAPTURE\_IMAGE\_REQUEST\_CODE***);  
}

1. Write a method inside “**CameraDemoFragment**” class to generate a bitmap from the captured image file and set to the ImageView
   1. Method name:- **previewCapturedImage**
   2. Return Type:- **void**
   3. Access modifiers :- **private**
   4. Parameters:- None
   5. Logic:-

*/\*\*  
 \* Display image from gallery  
 \*/***private void** previewCapturedImage() {  
 *//construct the bit map ater reducing the file seize* Bitmap bitmap = DemoUtil.*optimizeBitmap*(***BITMAP\_SAMPLE\_SIZE***, **imageStoragePath**);  
 *//set the generated bitmap to the image view* **imageView**.setImageBitmap(bitmap);  
}

1. Inside **CameraDemoFragment** class, override the **onViewCreated** method to initialize the views and add onclick listener to the button  
   @Override  
   **public void** onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {  
    **super**.onViewCreated(view, savedInstanceState);  
    *//initialize the image view* **imageView** = view.findViewById(R.id.***imageViewCameraImage***);  
    *//initialize the button* **buttonCaptureImage** = view.findViewById(R.id.***buttonCameraApp***);  
    *//initalize the context* **context** = getContext();  
    *//set the onclick listener of the button* **buttonCaptureImage**.setOnClickListener(**new** View.OnClickListener() {  
    @Override  
    **public void** onClick(View v) {  
    *//check if user has given the required permissions* **boolean** hasUserPermissions = DemoUtil.*hasUserPermissions*(**context**, **requiredPermissions**);  
    **if** (!(hasUserPermissions)) {  
    *//if read contact permission is not already granted, request permission* requestPermissions(**requiredPermissions**, DemoUtil.*UNIQUE\_APP\_REQUEST\_CODE*);  
    } **else** {  
    *//check if the camera is available in this device* **boolean** cameraAvailable = DemoUtil.*isSystemFeatureAvailable*(**context**, PackageManager.***FEATURE\_CAMERA***);  
    **if** (cameraAvailable) {  
    *//if camera is available, capture the image* captureImage();  
    }**else**{  
    *//else notify the user* Toast.*makeText*(**context**,  
    **"Camera is not Available!"**, Toast.***LENGTH\_SHORT***)  
    .show();  
    }  
    }  
    }  
    });  
   }
   1. As you could notice, when user clicks on “Capture Image” button, user will be redirected to the camera app, we wrote it in “captureImage()” method above
   2. After user captures the image, he will be redirected back to our intent
   3. In order to capture that, method shown below is required.
2. Inside **CameraDemoFragment,** override the **“onActivityResult”** method to get the captured image from camera intent  
   */\*\*  
    \* Activity result method will be called after closing the camera  
    \*/*@Override  
   **public void** onActivityResult(**int** requestCode, **int** resultCode, Intent data) {  
    *// if the result is capturing Image* **if** (requestCode == ***CAMERA\_CAPTURE\_IMAGE\_REQUEST\_CODE***) {  
    **if** (resultCode == Activity.***RESULT\_OK***) {  
    *// Refreshing the gallery* DemoUtil.*refreshGallery*(**context**, **imageStoragePath**);  
    *// successfully captured the image  
    // display it in image view* previewCapturedImage();  
    } **else if** (resultCode == Activity.***RESULT\_CANCELED***) {  
    *// user cancelled Image capture* Toast.*makeText*(**context**,  
    **"User cancelled image capture"**, Toast.***LENGTH\_SHORT***)  
    .show();  
    } **else** {  
    *// failed to capture image* Toast.*makeText*(**context**,  
    **"Sorry! Failed to capture image"**, Toast.***LENGTH\_SHORT***)  
    .show();  
    }  
    }  
   }
3. In case required to refer to, below are the complete set of imports inside **CameraDemoFragment** class  
   **import** android.Manifest;  
   **import** android.app.Activity;  
   **import** android.content.Context;  
   **import** android.content.Intent;  
   **import** android.content.pm.PackageManager;  
   **import** android.graphics.Bitmap;  
   **import** android.net.Uri;  
   **import** android.os.Bundle;  
   **import** android.provider.MediaStore;  
   **import** android.support.annotation.NonNull;  
   **import** android.support.annotation.Nullable;  
   **import** android.support.v4.app.Fragment;  
   **import** android.view.LayoutInflater;  
   **import** android.view.View;  
   **import** android.view.ViewGroup;  
   **import** android.widget.Button;  
   **import** android.widget.ImageView;  
   **import** android.widget.Toast;  
     
   **import** java.io.File;  
     
   **import** lk.uok.mit.helloworld.R;  
   **import** lk.uok.mit.util.DemoUtil;
4. Now save everything and run the app and check, the output should look like below;

Source code for this tutorial part can be found in Git Repository given below: - <https://github.com/nadee158/android_tutorial_part_7.git>

# References

Ableson, F. (2009, June 16). *Tapping into Android's sensors*. Retrieved from www.ibm.com: https://www.ibm.com/developerworks/library/os-android-sensor/index.html

*Android AutoCompleteTextView Example*. (2018, January 1). Retrieved from JavaTPoint: https://www.javatpoint.com/android-autocompletetextview-example

*Application Fundamentals*. (2018, May 2018). Retrieved from Android Developers: https://developer.android.com

CHUGH, A. (2018, April 2). *Android AutoCompleteTextView Example Tutorial*. Retrieved from JournalDev: https://www.journaldev.com/9574/android-autocompletetextview-example-tutorial

*Creating and Using Fragments*. (2015, 01 01). Retrieved from guides.codepath.com: https://guides.codepath.com/android/creating-and-using-fragments

google-developer-training. (2018, June 18). *Working with sensor data*. Retrieved from google-developer-training.gitbooks.io: https://google-developer-training.gitbooks.io/android-developer-advanced-course-practicals/unit-1-expand-the-user-experience/lesson-3-sensors/3-1-p-working-with-sensor-data/3-1-p-working-with-sensor-data.html

Green, P. (2015, April 03). *Using Android Sensors in Your App*. Retrieved from www.sitepoint.com: https://www.sitepoint.com/using-android-sensors-application/

Hathibelagal, A. (2017, January 27). *Android Sensors in Depth: Proximity and Gyroscope*. Retrieved from Envato.com: https://code.tutsplus.com/tutorials/android-sensors-in-depth-proximity-and-gyroscope--cms-28084

Kim, M. (2015, Jul 13). *Create a Bluetooth Scanner With Android's Bluetooth API*. Retrieved from code.tutsplus.com: https://code.tutsplus.com/tutorials/create-a-bluetooth-scanner-with-androids-bluetooth-api--cms-24084

*Sensors Overview*. (2018, June 19). Retrieved from Android Developers: https://developer.android.com/guide/topics/sensors/sensors\_overview

Tutorialspoint. (2018, June 18). *Android - Sensors*. Retrieved from Tutorialspoint: https://www.tutorialspoint.com/android/android\_sensors.htm