CS 473 - MDP
Mobile Device Programming

Maharishi International University

CS 473 - MDP
Mobile Device Programming

MS.CS Program

Department of Computer Science

Renuka Mohanraj , Ph.D.

Maharishi International
University

CS 473 – MDP
Mobile Device Programming

Lesson 9
Multimedia

Maharishi International
University

Wholeness of the Lesson

Multimedia capabilities of mobile devices play a significant role for consumers. Android's open-source technology provides support for multimedia capabilities. It ensures that it offers multimedia API for playing and recording of wide range of image, audio, and video formats. Camera API also exposes the capabilities to build applications with comprehensive multimedia functionality. *In a similar way, when the mind transcends, awareness become saturated with the level from which all the laws of nature begin to operate, the unified field. Having the support of this powerful capability, desires can effortlessly meet with fulfillment.*

Agenda

- Android Run time permissions
- Video Playback using VideoView and MediaController Classes
- Video Recording using Camera Intents
- Access Galley
- Audio Recording

Android Runtime Permissions

- You have already used permissions in some of your apps:
  - Permission to connect to the Internet
- App must get permission to do anything that
  - Uses data or resources that the app did not create
  - Uses network, hardware, features that do not belong to it
  - Affects the behavior of the device
  - Affects the behavior of other apps
  - If it isn't yours, get permission!

How apps declare permissions they need

List permissions in the AndroidManifest.xml using **<uses-permission>**

Example :

<uses-permission
android:name="android.permission.READ_CONTACTS" />

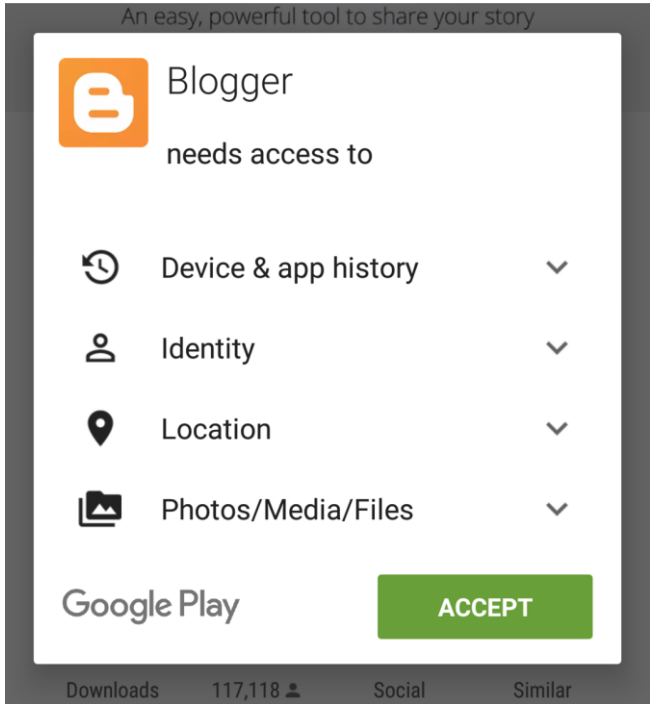
<uses-permission
android:name="android.permission.READ_CALENDAR" />

<uses-permission
android:name="android.permission.CALL_PHONE"/>

How users grant permission

For apps created before Marshmallow(Android 6.0 – API 23)

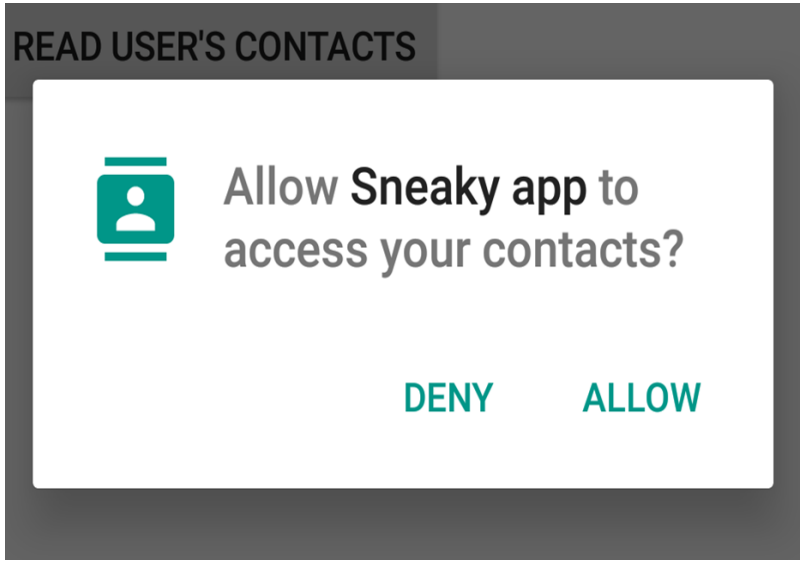
- Users grant permission before installing



How users grant permission

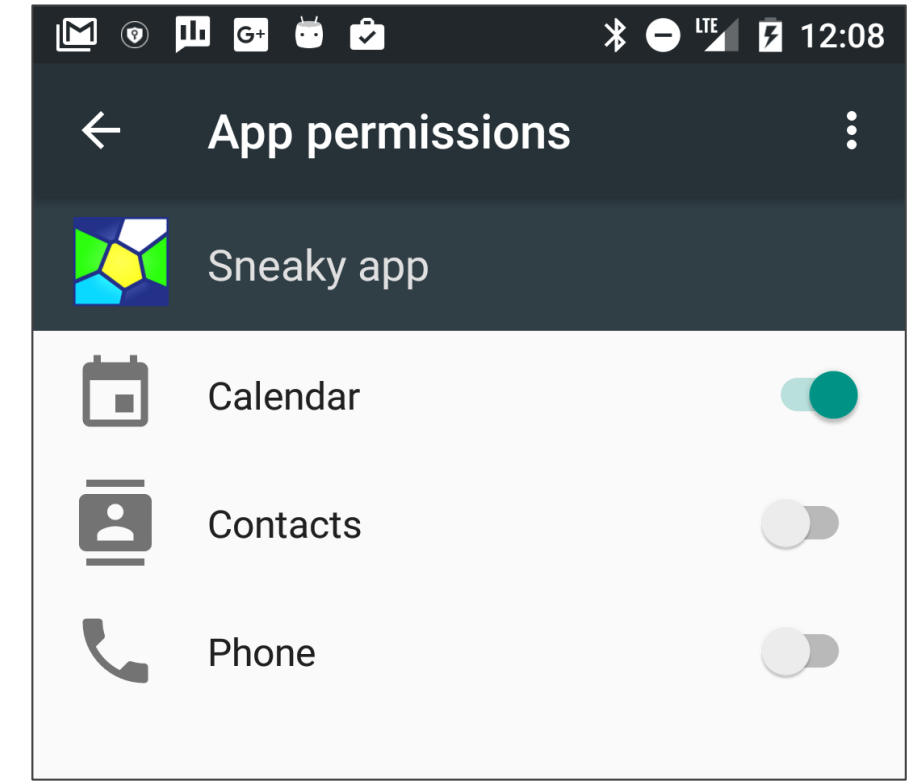
Marshmallow onwards

- Installation doesn't ask user to give permissions
- App must get runtime permission for accessing the features



How users revoke permission

- Before Marshmallow
  - Uninstall app!
- Marshmallow onwards
  - Revoke individual permissions
  - Settings > apps > permissions

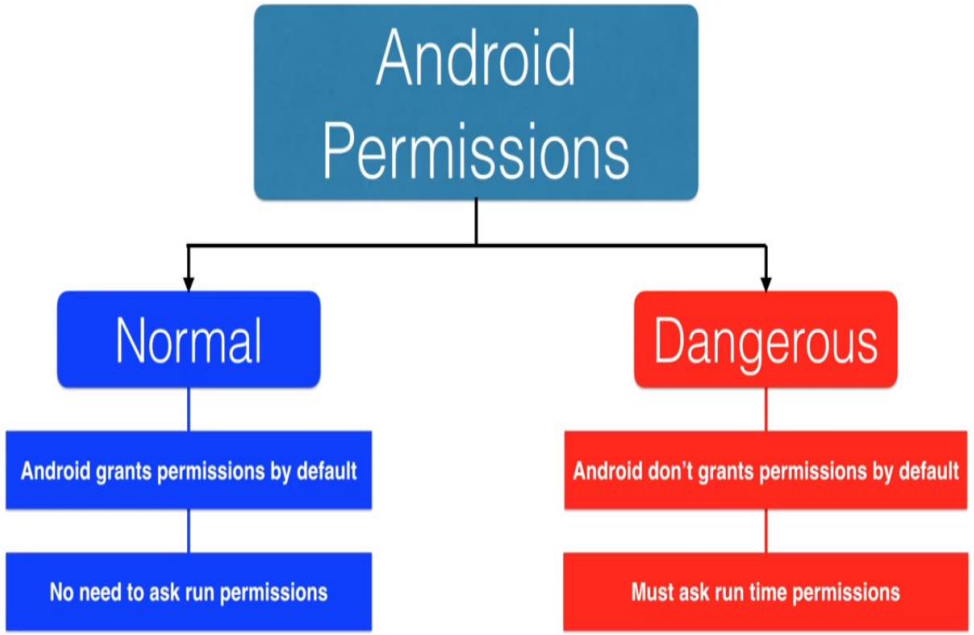


Differences in permission models

- Before Marshmallow
  - If app is running, it can assume that user granted permissions during installation
- After Marshmallow
  - App needs to get permission at runtime
  - Must check if it still has permission every time
  - User can revoke permissions at any time

Android permissions category

- All permission need to be included on Android Permissions
- **Normal/Installed** permissions do not directly risk the user's privacy
  - ***Example*:** Set the time zone, Internet
  - Android automatically grants normal permissions.
- **Dangerous/Runtime** permissions give access to user's private data
  - ***Example*:** Read the user's contacts.
  - Android asks user to explicitly grant dangerous permissions



Android permissions category

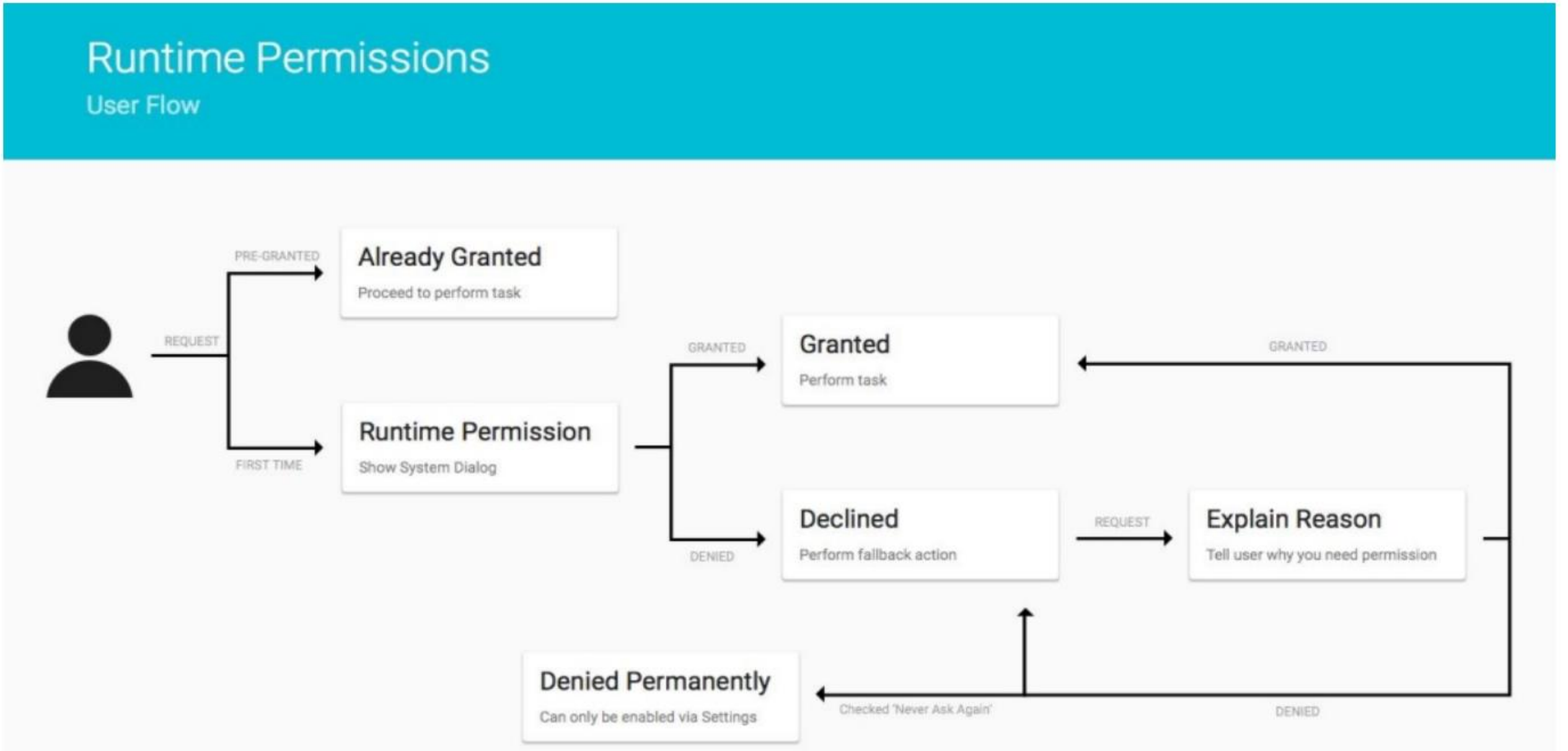
- **Special** permissions
  - There are a couple of permissions that don't behave like normal and dangerous permissions. SYSTEM_ALERT_WINDOW and WRITE_SETTINGS are particularly sensitive, so most apps should not use them. If an app needs one of these permissions, it must declare the permission in the manifest, and send an intent requesting the user's authorization. The system responds to the intent by showing a detailed management screen to the user.
- **Signature** permissions
  - The system grants these app permissions at install time, but only when the app that attempts to use a permission is signed by the same certificate as the app that defines the permission.(Eg: Access Linux user id to access system files)
    - Ref :https://developer.android.com/guide/topics/permissions/defining#signing

Dangerous/Runtime permissions

Permission Group	Permission
Calendar	READ_CALENDAR WRITE_CALENDAR
Camera	CAMERA
Contacts	READ_CONTACTS WRITE_CONTACTS GET_ACCOUNTS
Location	ACCESS_FINE_LOCATION ACCESS_COARSE_LOCATION
Microphone	RECORD_AUDIO
Phone	READ_PHONE_STATE CALL_PHONE READ_CALL_LOG WRITE_CALL_LOG ADD_VOICEMAIL USE_SIP PROCESS_OUTGOING_CALLS
Sensors	BODY_SENSORS
SMS	SEND_SMS RECEIVE_SMS READ_SMS RECEIVE_WAP_PUSH RECEIVE_MMS
Storage	READ_EXTERNAL_STORAGE WRITE_EXTERNAL_STORAGE

Run time permissions flow – Dangerous Permission

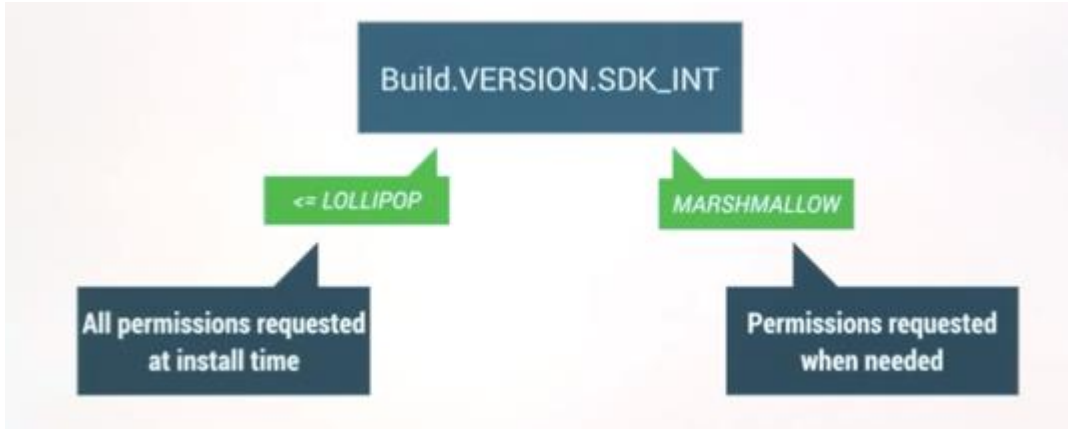
https://developer.android.com/training/permissions/requesting



Runtime Permission flow - RunTimePermissionCamera

Step 1 2 3 4 5

If you are using below Android 6.0 permissions granted at install time or else need to go with the step 2 – 5 mentioned in the flow. 98% of users are above Android 6 version worldwide.



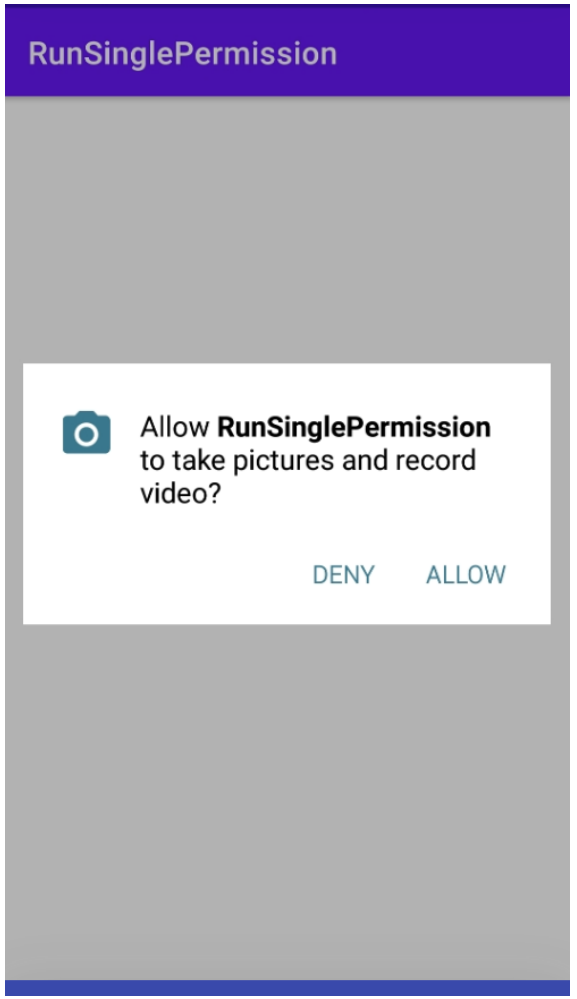

```
if (android.os.Build.VERSION.SDK_INT >=
android.os.Build.VERSION_CODES.M){     }
```

Run Time Permission Camera - Example

Step 1: Add the below permission in the Manifest file.

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

- If you first time installed app and click the Request Permission button you will get the given screen.
- If you click the **Allow** button, will get “Permission granted Screen”.
- If you click the **Deny** button, can explain the reason to the user.
- The Alert dialog message is come from the API depends on the permission you requested with your app name. Came with DENY and ALLOW buttons.
- **Refer Demo: RunSinPermission**



Run time permissions Camera

- Step – 2- Need to declare the object to get the Activity Result Callback

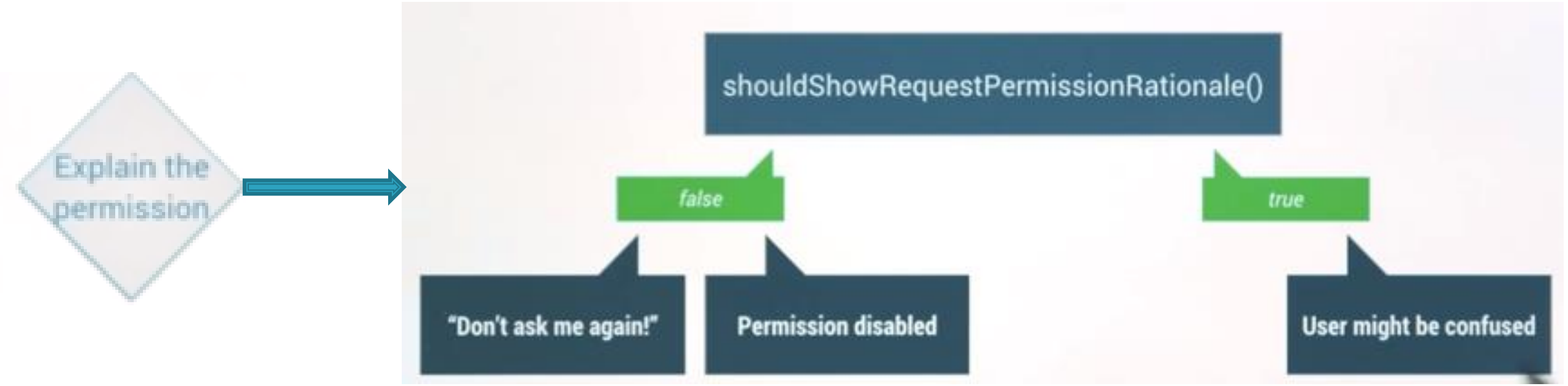
```
private val requestPermissionLauncher =
        registerForActivityResult(
            ActivityResultContracts.RequestPermission()
        ) { isGranted: Boolean -> // Lambda argument of boolean result
            if (isGranted) {
                Log.i("Permission: ", "Granted") // Your action
            } else {
                Log.i("Permission: ", "Denied") // Your action
            }
        }
```

Logic to Check Permission Granted or not

```
ContextCompat.checkSelfPermission( this,
            Manifest.permission.CAMERA) ==
            PackageManager.PERMISSION_GRANTED -> {
            Toast.makeText(this,"You are already granted this
permission", Toast.LENGTH_LONG).show()
        }
```

Explain to the User – Why Permission

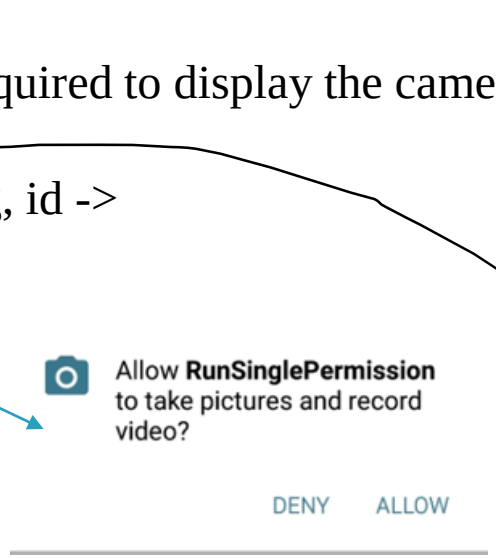
If you don’t already have permission, explain why the permission is necessary. shouldShowRequestPermissionRationale() return false if the user disable permission or enable Don’t ask me again option. If it returns true means user previously rejected due to confusion that app need a permission. Now again user is trying to access the feature and request permission.

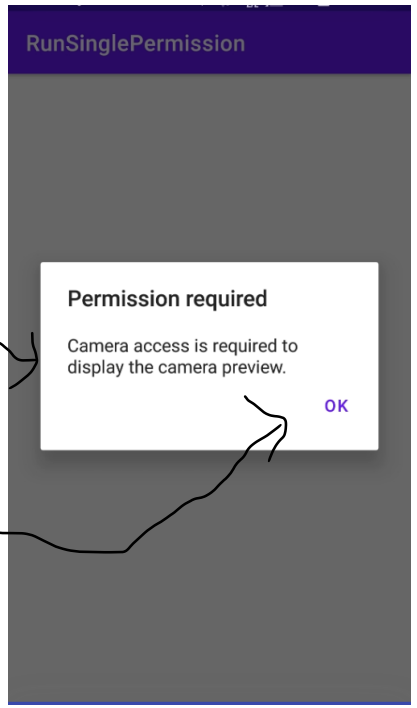


Logic to explain the reason showing Alert Dialog

Step 3 Code – If the user click the Deny button, explain the reason why permission needed using Alert Dialogs.

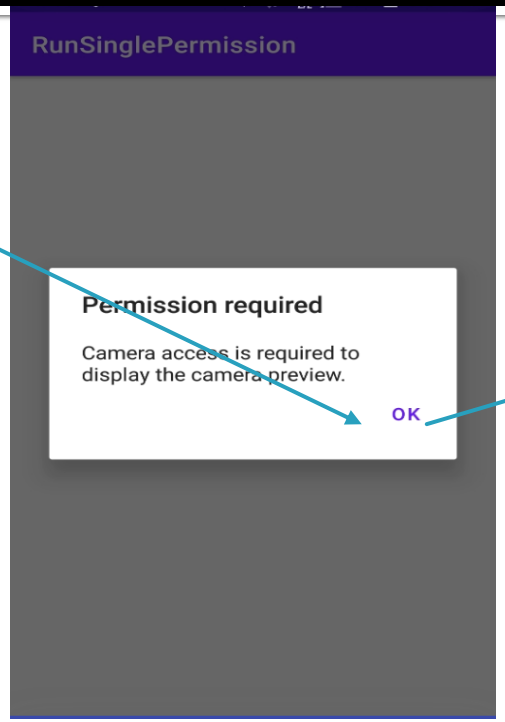
```
ActivityCompat.shouldShowRequestPermissionRationale( this,  Manifest.permission.CAMERA ) -> {
        val builder = AlertDialog.Builder(this)
         builder.setMessage("Camera access is required to display the camera preview.")
          .setTitle("Permission required")
         builder.setPositiveButton("OK") { dialog, id ->
            requestPermissionLauncher.launch(
                Manifest.permission.CAMERA
            )
         }
         val dialog = builder.create()
         dialog.show()
    }
```

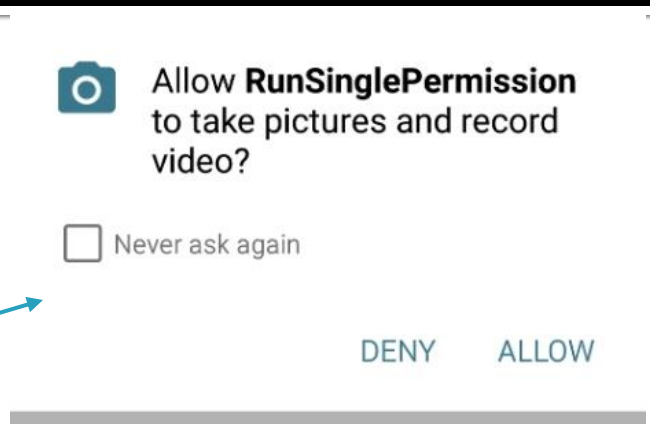


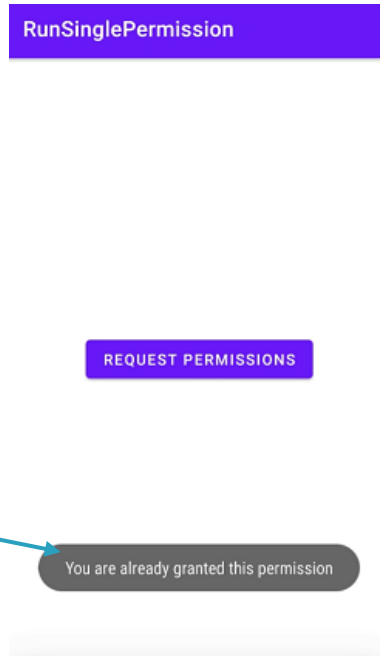


Action Depends on the User Response

- If the user press the OK button, will get another alert to accept or User understand the runtime permission requirements and can click allow.
- If the Permission already granted user will get the Toast Message when press the REQUEST PERMISSION button.







Main Point 1

Permission requests protect sensitive information available from a device and should only be used when access to information is necessary for the functioning of your app. Android 6.0 Marshmallow introduced a new permissions model that lets apps request permissions from the user at runtime, rather than prior to installation. ***Science of Consciousness:*** *In Transcendental consciousness, as a field of COMPLETE SELF-SUFFICIENCY, functioning from within itself, not dependent on any influence from outside to create - and always creating a totally evolutionary influence, enriching life.*

Introduction – VideoView and MediaController

- The Android SDK includes two classes that make the implementation of video playback on Android devices extremely easy to implement when developing applications.
- This lesson will provide an overview of these two classes, VideoView and MediaController
- Introducing the Android VideoView Class
  - simplest way to display video within an Android application is to use the VideoView class.
  - This is a visual component which, when added to the layout of an activity, provides a surface onto which a video may be played.
- Android currently supports the following video formats:
  - H.263
  - H.264 AVC
  - H.265 HEVC
  - MPEG-4 SP
  - VP8
  - VP9

Methods from VideoView Class

It has a wide range of methods that may be called in order to manage the playback of video. Some of the more commonly used methods are as follows:

- **setVideoPath(String path)** – Specifies the path (as a string) of the video media to be played. This can be either the URL of a remote video file or a video file local to the device.
- **setVideoUri(Uri uri)** – Performs the same task as the setVideoPath() method but takes a Uri object as an argument instead of a string.
- **start()** – Starts video playback.
- **stopPlayback()** – Stops the video playback.
- **pause()** – Pauses video playback.
- **isPlaying()** – Returns a Boolean value indicating whether a video is currently playing.
- **getDuration()** – Returns the duration of the video. Will typically return **-**1 unless called from within the OnPreparedListener() callback method.

Methods from VideoView Class

•**getCurrentPosition()** – Returns an integer value indicating the current position of playback.

• **setMediaController(MediaController)** – Designates a MediaController instance allowing playback controls to be displayed to the user.

setOnPreparedListener(MediaPlayer.OnPreparedListener) – Allows a callback method to be called when the video is ready to play.

• **setOnErrorListener(MediaPlayer.OnErrorListener) -** Allows a callback method to be called when an error occurs during the video playback.

• **setOnCompletionListener(MediaPlayer.OnCompletionListener) -** Allows a callback method to be called when the end of the video is reached.

Android MediaController Class

- If a video is simply played using the VideoView class, the user will not be given any control over the playback, which will run until the end of the video is reached.
- This issue can be addressed by attaching an instance of the MediaController class to the VideoView instance.
- The MediaController will then provide a set of controls allowing the user to manage the playback (such as pausing and seeking backwards/forwards in the video time-line).
- The position of the controls is designated by anchoring the controller instance to a specific view in the user interface layout.
- Once attached and anchored, the controls will appear briefly when playback starts and may subsequently be restored at any point by the user tapping on the view to which the instance is anchored.

Methods from MediaController Class

- **setAnchorView(View view)** – Designates the view to which the controller is to be anchored. This controls the location of the controls on the screen.
- **show()** – Displays the controls.
- **show(int timeout)** – Controls are displayed for the designated duration (in milliseconds).
- **hide()** – Hides the controller from the user.
- **isShowing()** – Returns a Boolean value indicating whether the controls are currently visible to the user.

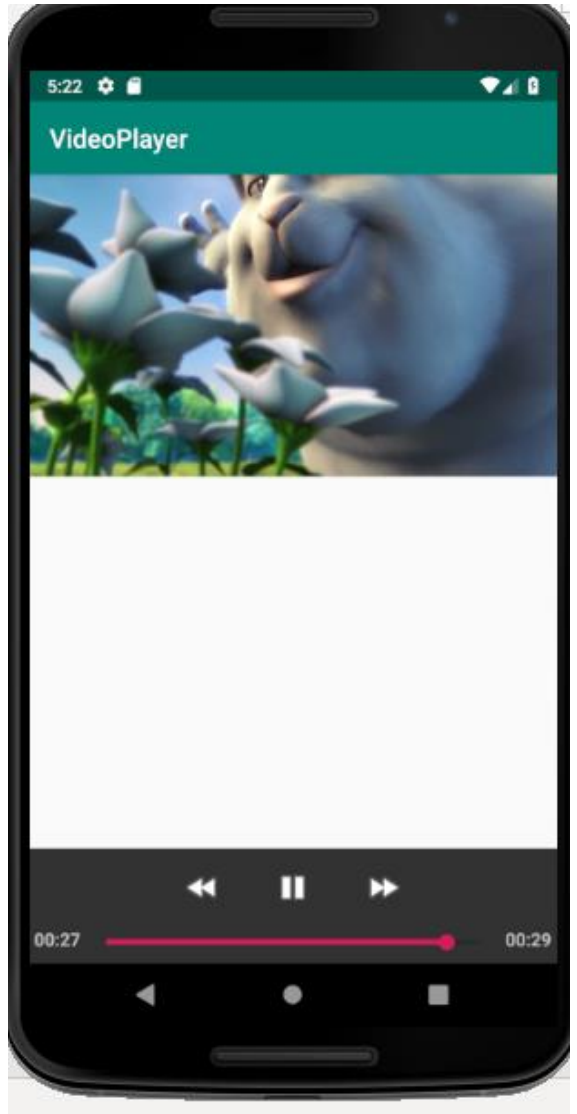
Hands on Example – 1 - VideoPlayer

- To play a web-based MPEG-4 video file using VideoView and MediaController classes is done by doing the following steps.
  - Design your Layout with VideoView Component.
  - The next step is to configure the VideoView with the path of the video to be played and then start the playback using your Kotlin code.
  - Adding below Internet Permission line at AndroidManifest.xml before the application tag.

<uses-permission android:name="android.permission.INTERNET" />

Hands on Example - VideoPlayer

- Once the app is loaded, it will play the Video from the given URL with MediaController.
- Refer: VideoPlayer



Example – VideoPlayer – activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <VideoView
        android:id="@+id/videoView1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Example – VideoPlayer – MainActivity.kt

```
class MainActivity : AppCompatActivity() {
    private var TAG = "VideoPlayer"
    var mediaController: MediaController? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        configureVideoView()
    }
```

Example – VideoPlayer – MainActivity.kt

```
private fun configureVideoView() {
        // To read from the given URL needs Internet permission in Manifest
       videoView1.setVideoPath("https://www.demonuts.com/Demonuts/smallvideo.mp4")
        // To read from raw folder
      //  videoView1.setVideoPath("android.resource://" + packageName + "/" + R.raw.samplevideo )
        /*VideoView canvas will cause the media controls will appear over the video playback by tapping.
         These controls should include a seekbar together with fast forward, rewind and play/pause buttons.*/
        mediaController = MediaController(this)
        mediaController?.setAnchorView(videoView1)

        // configure video playback to loop continuously and display the video duration on logs.
        videoView1.setOnPreparedListener { mp ->
            mp.isLooping = true
            Log.i(TAG, "Duration = " + videoView1.duration) }
        // Start Playing
        videoView1.start()
    }
}
```

Summary

- Android devices make excellent platforms for the delivery of content to users, particularly in the form of video media.
- Android SDK provides two classes, namely VideoView and MediaController, which combine to make the integration of video playback into Android applications quick and easy, often involving just a few lines of Kotlin code.

Main Point 2

- VideoView is used to play a video file. The VideoView class has a wide range of methods that may be called in order to manage the playback of video. MediaPlayer is responsible for playing audio and video and is the primary class. ***Science of Consciousness:*** *Transcendental Meditation is primary tool provides a wide range of benefits in order to manage stress, balancing the nervous system and increase the brain coherence.*

Video Recording using Camera Intents

- Most of the Android devices are equipped with at least one camera.
- The Android framework provides support for various cameras and camera features available on devices, allowing you to capture pictures and videos in your application.
- The Android framework supports capturing images and video through the Camera API or Camera Intent.
- We will discuss to make use of CameraIntent in this lesson
- Refer : VideoRecordingDemo

Hands on Example – Video Recording

Checking for Camera Support

- Before attempting to access the camera on an Android device, it is essential that defensive code be implemented to verify the presence of camera hardware.
- Camera can be identified via a call to the *PackageManager.hasSystemFeature()* method.

```
private fun hasCamera(): Boolean {
return packageManager.hasSystemFeature(
PackageManager.FEATURE_CAMERA_ANY)
}
```

- In order to check for the presence of a front-facing camera, the code needs to check for the presence of the PackageManager.FEATURE_CAMERA_FRONT feature.

Hands on Example - Video Recording

Register to get back the Activity Result – new Kotlin Way

```
val getVideo = registerForActivityResult(ActivityResultContracts.StartActivityForResult()){
    if(it.resultCode== RESULT_OK){
        val videoUrl = it?.data?.data
        videoView.setVideoURI(videoUrl)
        videoView.start()
    }
}
```

it: ActivityResult!

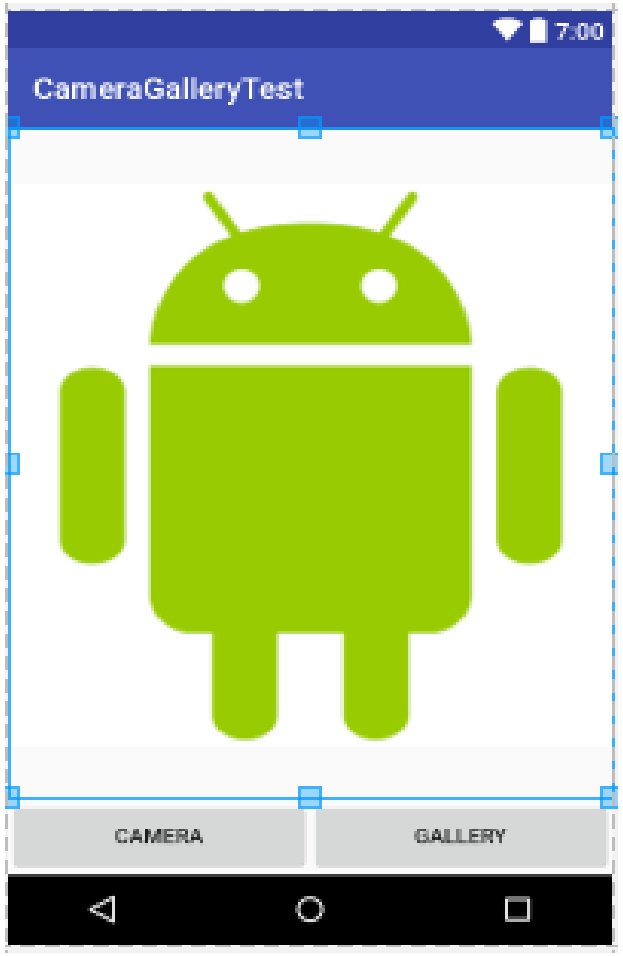
Calling the Video

getVideo.launch(Intent(MediaStore.ACTION_VIDEO_CAPTURE) Capture Intent

Refer : VideoRecordingDemo

Hands on Example – Camera & Gallery

- Problem : Click the Camera button to take a picture using device Camera and set the captures image to the ImageView Component.
- Click the Gallery button choose the image from your device Photo Gallery and the selected image will be set in the ImageView Component.
- **Refer : AccessCameraGallery**



Camera button click code

lateinit var startforResultCamera : ActivityResultLauncher<Intent>

```
// Get captured image from the Camera Intent to set on the ImageView UI
    startforResultCamera  = registerForActivityResult(ActivityResultContracts.StartActivityForResult()){ it ->
        if(it.resultCode == Activity.RESULT_OK && it.data!=null){
            val extras = it.data!!.extras
            val imageBitmap = extras?.get("data") as Bitmap
            iv.setImageBitmap(imageBitmap)
        }
        else{
            Toast.makeText(this,"Fail to retrieve", Toast.LENGTH_LONG).show()
        }
    }
// Start taking picture by clicking Camera button from your activity
    camera.setOnClickListener {
            val takePictureIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
            if (takePictureIntent.resolveActivity(packageManager) != null) {
                // Call launch by passing
                startforResultCamera.launch(takePictureIntent)
            }
        }
```

Gallery button click code

lateinit var startforResultGalley : ActivityResultLauncher<Intent>

```
// Get the image from the Gallery using GetConent() from ActivityResultContract
    startforResultGalley =  registerForActivityResult(ActivityResultContracts.StartActivityForResult()) {
        if(it!=null)
          iv.setImageURI(it.data?.data)
        else
           Toast.makeText(this,"Fail to retrieve", Toast.LENGTH_LONG).show()
    }
// Start Pick picture by clicking Camera button from Gallery app
    gallery.setOnClickListener {
        val i = Intent()
        // Activity Action for the intent : Pick an item from the data, returning what was selected.
        i.action = Intent.ACTION_PICK
        i.type = "image/*"
        startforResultGalley.launch(i)
    }
```

Audio Recording using MediaRecorder

- In terms of audio playback, most implementations of Android support AAC LC/LTP, HE-AACv1 (AAC+), HE-AACv2 (enhanced AAC+), AMR-NB, AMR-WB, MP3, MIDI, Ogg Vorbis, and PCM/WAVE formats.
- **MediaRecorder** is used to record audio and video.
- Audio playback can be performed using either the **MediaPlayer** or the **AudioTrack** classes.
- AudioTrack is a more advanced option that uses streaming audio buffers and provides greater control over the audio.
- The MediaPlayer class, on the other hand, provides an easier programming interface for implementing audio playback and will meet the needs of most audio requirements. We are discussing MediaPlayer in this course.

Audio Recording using MediaRecorder

Steps need to follow

- Initialize a new instance of MediaRecorder with the following calls:
  - Set the audio source using setAudioSource(). You'll probably use MIC.
  - Set the output file format using setOutputFormat().
  - Set the output file name using setOutputFile().
  - Set the audio encoder using setAudioEncoder().
  - Complete the initialization by calling prepare().
- Start and stop the recorder by calling start() and stop() respectively.
- When you are done with the MediaRecorder instance free its resources as soon as possible by calling release().

AndroidManifest.xml

- Need to add the following permissions on AndroidManifest.xml

<uses-permission

android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

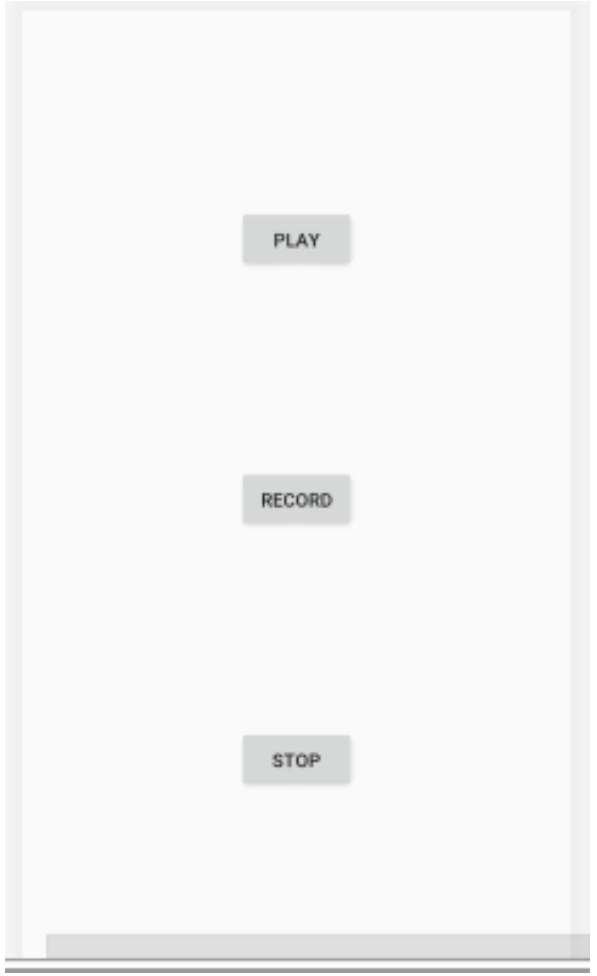
<uses-permission

android:name="android.permission.RECORD_AUDIO" />

- Need to integrate code for runtime permission check

Hands on Example – Audio Recording

- Click PLAY button to play the recorded audio
- Click RECORD button to record audio using your device micro phone.
- Click STOP button to stop recording.
- Refer : AudioRecording



Main point 3

The Android framework provides greater support for various cameras and camera features available on devices, allowing you to capture pictures and videos in your application. Android framework supports capturing images and video through the Camera API or camera Intent. ***Science of Consciousness:*** Support of nature is the most important factor in fulfillment of desires. So, with the support of nature one can enjoy greater efficiency and accomplish more.

UNITY CHART

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

All knowledge contained in point

1. Android multimedia framework includes support for playing variety of common media types, so that one can easily integrate audio, video and images into your applications.
2. Android offers support for several multimedia formats, such as, MPEG4, MIDI, 3GP, and so on. Enjoy greater efficiency and accomplish more.

---

3. **Transcendental Consciousness**: TC is the home of all knowledge. All knowledge has its basis in the unbounded field of pure consciousness.
4. ***Impulses within the Transcendental field:*** *These infinitely diverse impulses which create the whole universe, flow within the one holistic field of pure consciousness, transcendental consciousness.*
5. ***Wholeness moving within Itself****: In Unity Consciousness, when the home of all knowledge has become fully integrated in all phases of life, it is possible to know anything, any particular thing, instantly.*