

Lesson 9

Multimedia

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" />
```

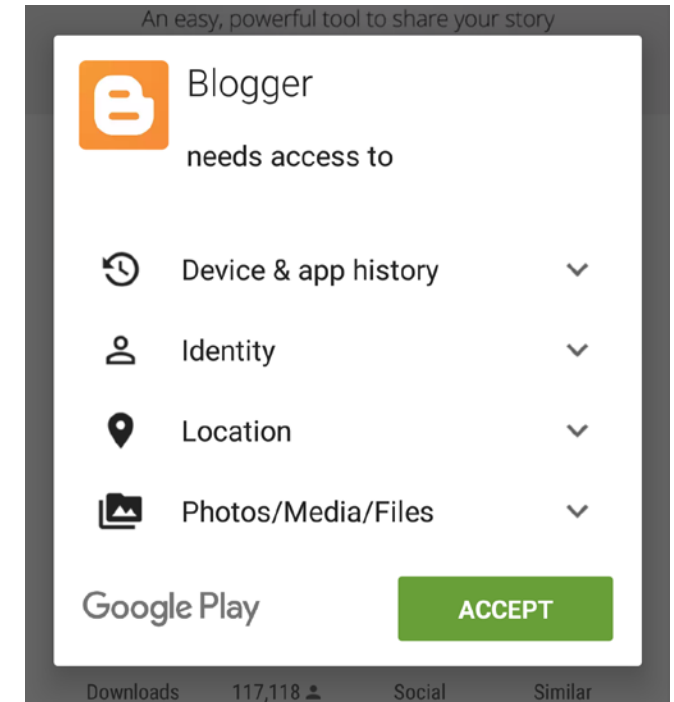
```
<uses-permission  
android:name="android.permission.READ_CALENDAR" />
```

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

How users grant permission

For apps created before Marshmallow(Android 6.0)

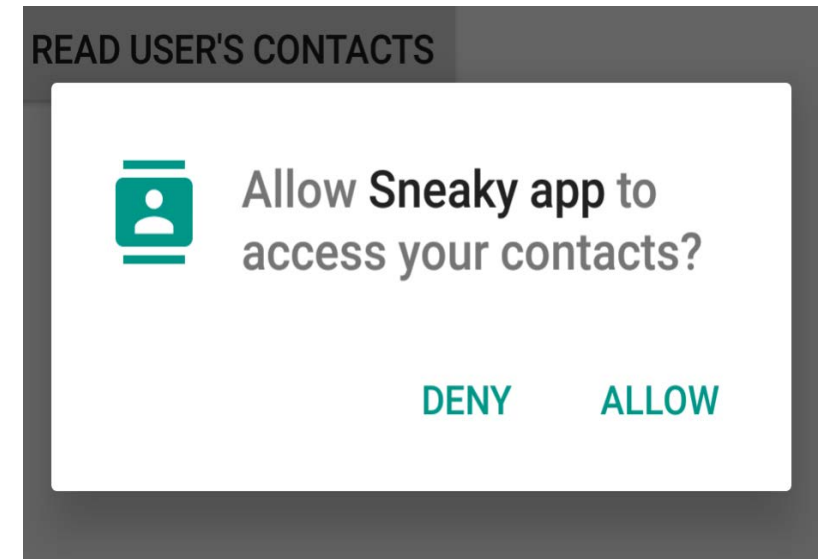
- Users grant permission before installing



How users grant permission

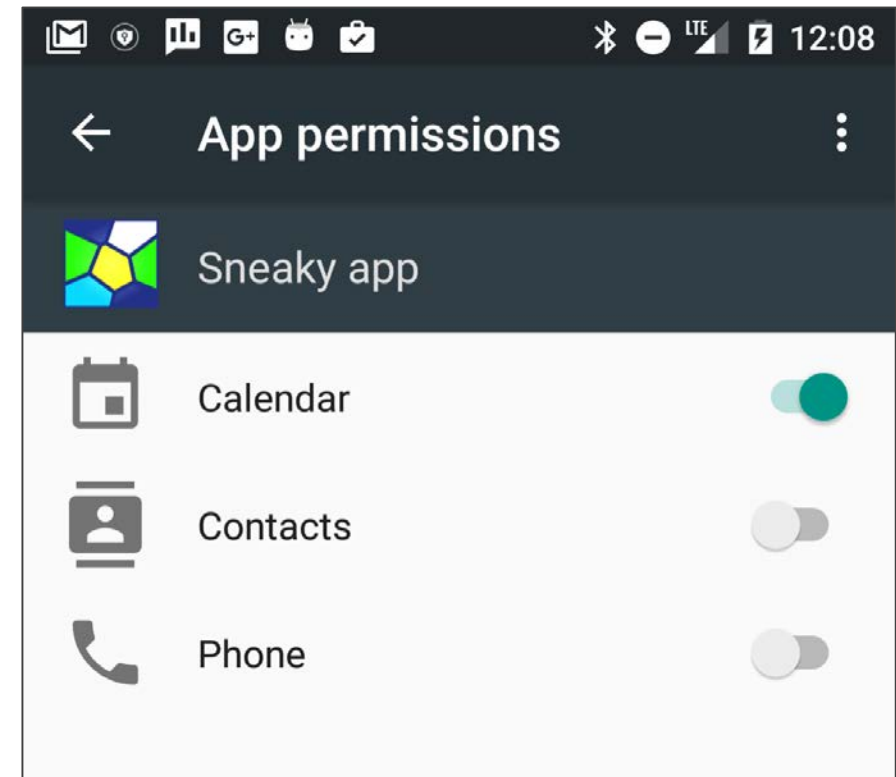
Marshmallow onwards

- 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

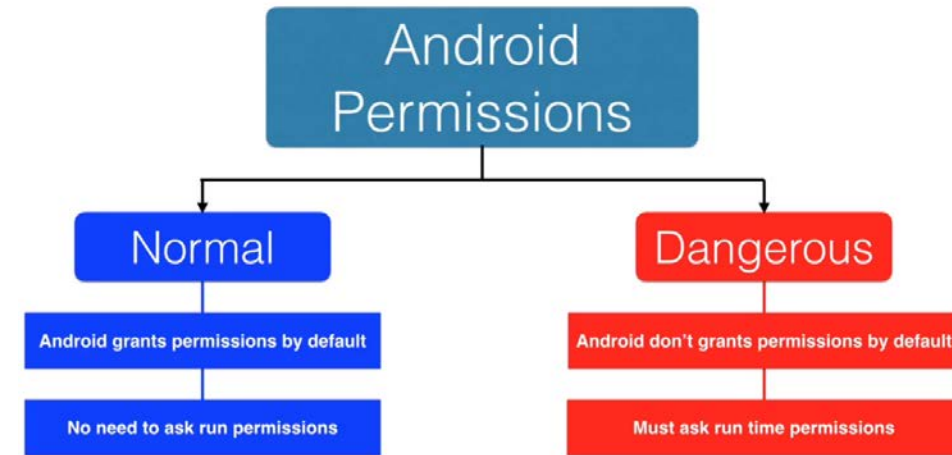


Differences in permission models

- 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** permissions do not directly risk the user's privacy
 - *Example:* Set the time zone, Internet
 - Android automatically grants normal permissions.
- **Dangerous** permissions give access to user's private data
 - *Example:* Read the user's contacts.
 - Android asks user to explicitly grant dangerous permissions



- **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.

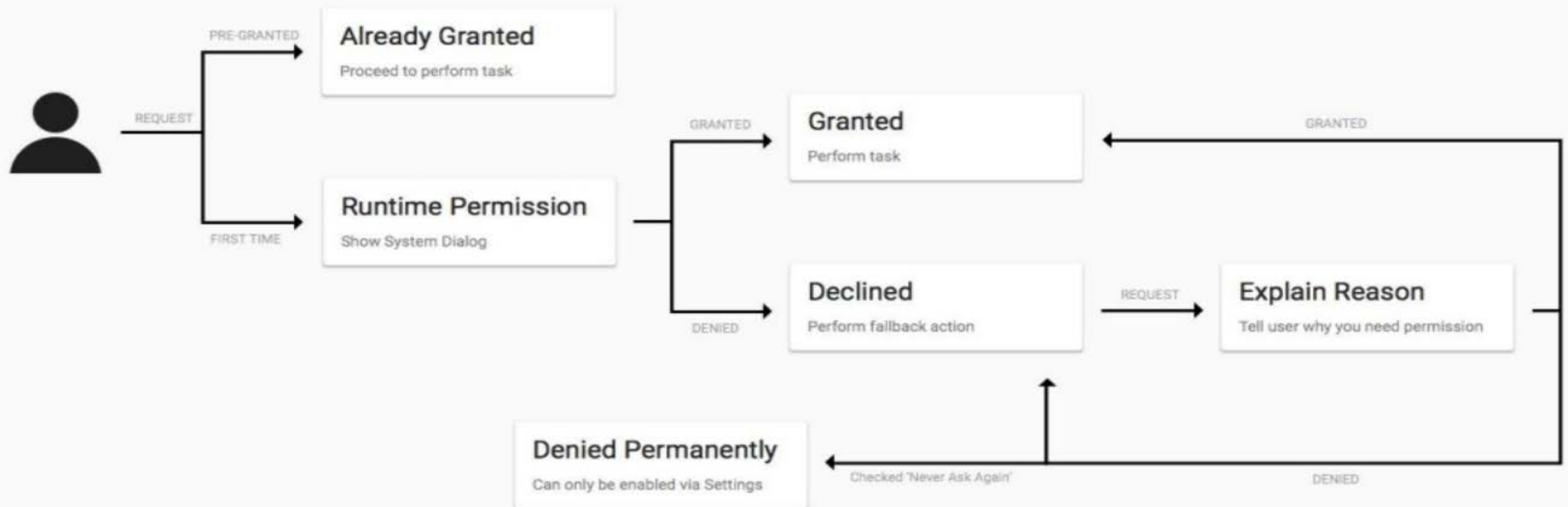
Dangerous permissions

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

Runtime Permissions

User Flow

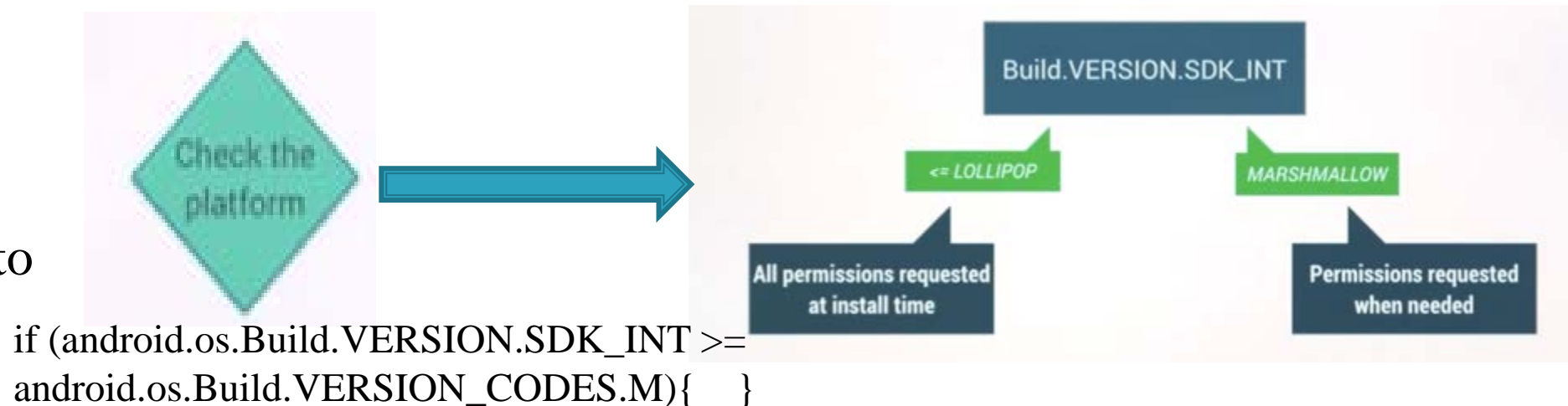


Runtime Permission flow and template code



Steps 1 2 3 4 5

Step 1: 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



Runtime Permission flow and template code

Step 2

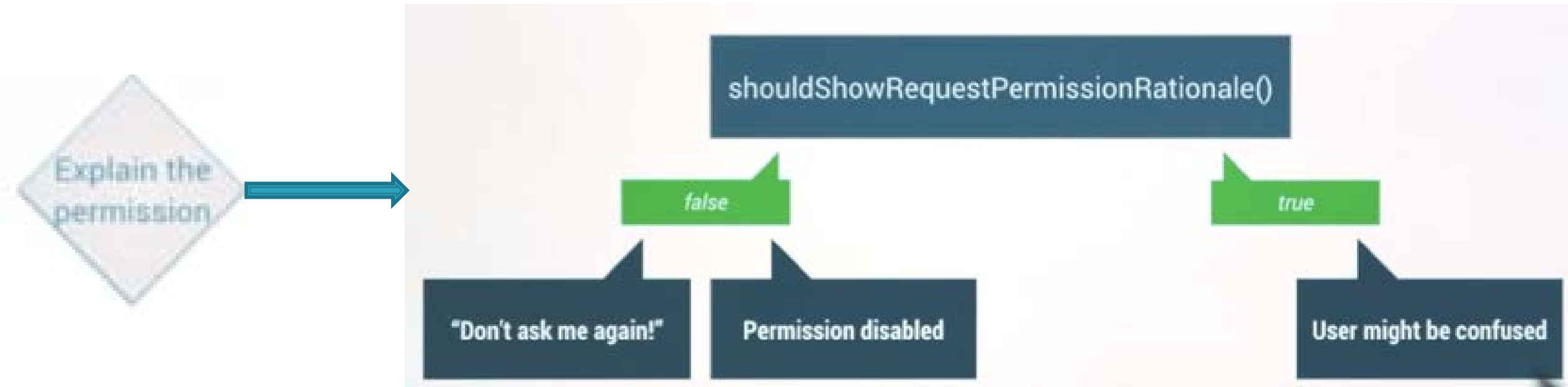
Check the permission



```
public void showCamera(View view) {  
    // Check if the Camera permission is already available.  
    if (checkSelfPermission(Manifest.permission.CAMERA) == PackageManager.PERMISSION_GRANTED) {  
        // Camera permissions is already available, show the camera preview.  
        showCameraPreview();  
    } else {  
        // Camera permission has not been granted.  
  
        // Provide an additional rationale to the user if the permission was not granted  
        // and the user would benefit from additional context for the use of the permission.  
        if (shouldShowRequestPermissionRationale(Manifest.permission.CAMERA)) {  
            Toast.makeText(this, "Camera permission is needed to show the camera preview.",  
                Toast.LENGTH_SHORT).show();  
        }  
  
        // Request Camera permission  
        requestPermissions(new String[]{Manifest.permission.CAMERA}, REQUEST_CAMERA);  
    }  
}
```

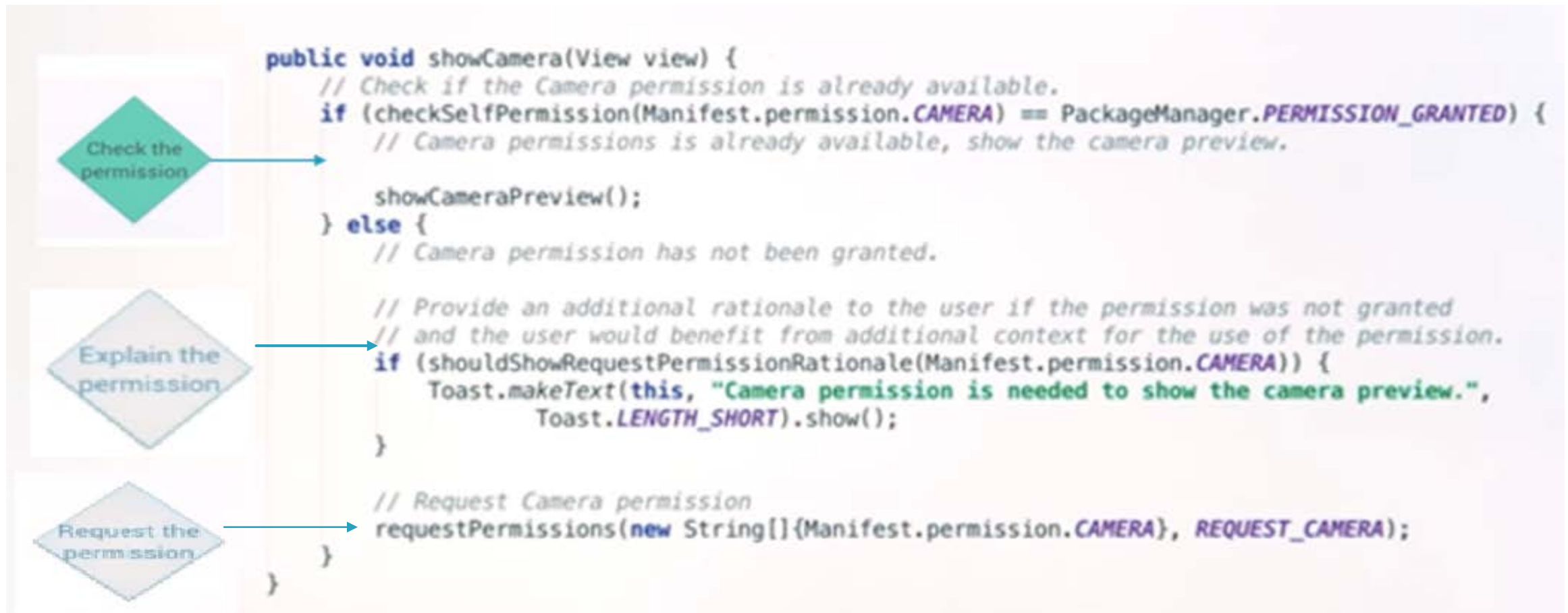
Runtime Permission flow and template code

Step 3: 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. [Revoke permission by the user]



Runtime Permission flow and template code

Step 4: Call `requestPermissions()` method by passing requested permission and the request code and will be used to identify which request has triggered the call to the `onRequestPermissionsResult()`




Runtime Permission flow and template code

Step 5 : Handle the request permission on Marshmallow

```
@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions,
    int[] grantResults) {

    if (requestCode == REQUEST_CAMERA) {
        // Received permission result for camera permission.

        // Check if the only required permission has been granted
        if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            // Camera permission has been granted, preview can be displayed
            showCameraPreview();
        } else {
            // Camera permission was denied, so we cannot use this feature.
            Toast.makeText(this, "Permission was not granted", Toast.LENGTH_SHORT).show();
        }
    } else {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    }
}
```



Handle the response

Example - Creating the Permissions –RECORD_AUDIO

- The Android Support Library contains a number of methods that can be used to seek and manage dangerous permissions within the code of an Android app. These API calls can be made safely regardless of the version of Android on which the app is running, but will only perform meaningful tasks when executed on Android 6.0 or later.
- Add the below permission on your AndroidManifest.xml
**<uses-permission
android:name='\"android.permission.RECORD_AUDIO\"' />**

Example - Creating the Permissions

- A permission request is made via a call to the *requestPermissions()* method of the ActivityCompat class. When this method is called, the permission request is handled asynchronously and a method named *onRequestPermissionsResult()* is called when the task is completed.
- The *requestPermissions()* method takes as arguments a reference to the current activity, together with the identifier of the permission being requested and a request code. The request code can be any integer value and will be used to identify which request has triggered the call to the *onRequestPermissionsResult()* method.
- Example :

```
val RECORD_REQUEST_CODE = 101
```

```
ActivityCompat.requestPermissions(this,  
    arrayOf(Manifest.permission.RECORD_AUDIO),  
    RECORD_REQUEST_CODE)
```

Refer : RunTimePermissionAudio

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.

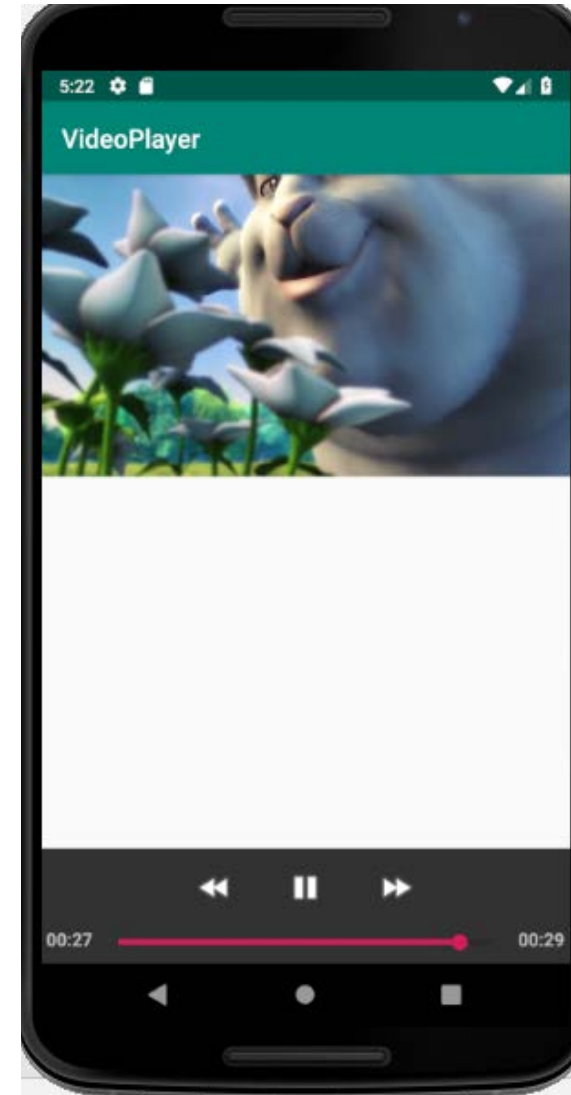
Hands on Example – 1 - VideoPlayer

- 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 the Video from the given URL with MediaController.
- Refer: VideoPlayer



Example – MediaPlayer – 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)  
    videoView1.setMediaController(mediaController)  
  
    // 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.

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 : CameraIntentsApp

Hands on Example – Video Recording

1. 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

2. Calling the Video Capture Intent

- The Android built-in video recording intent is represented by *MediaStore.ACTION_VIDEO_CAPTURE* and may be launched as follows:

```
private val VIDEO_CAPTURE = 101
```

```
val intent = Intent(MediaStore.ACTION_VIDEO_CAPTURE)
```

```
startActivityForResult(intent, VIDEO_CAPTURE)
```

When invoked in this way, the intent will place the recorded video into a file using a default location and file name.

Hands on Example - Video Recording

3. Overrie onActivityResult() method

- When the user either completes or cancels the video recording session, the *onActivityResult()* method of the calling activity will be called.
- This method needs to check that the request code passed through as an argument matches that specified when the intent was launched, verify that the recording session was successful and extract the path of the video media file.
- The corresponding *onActivityResult()* method for the above intent launch code might, therefore, be implemented as follows:

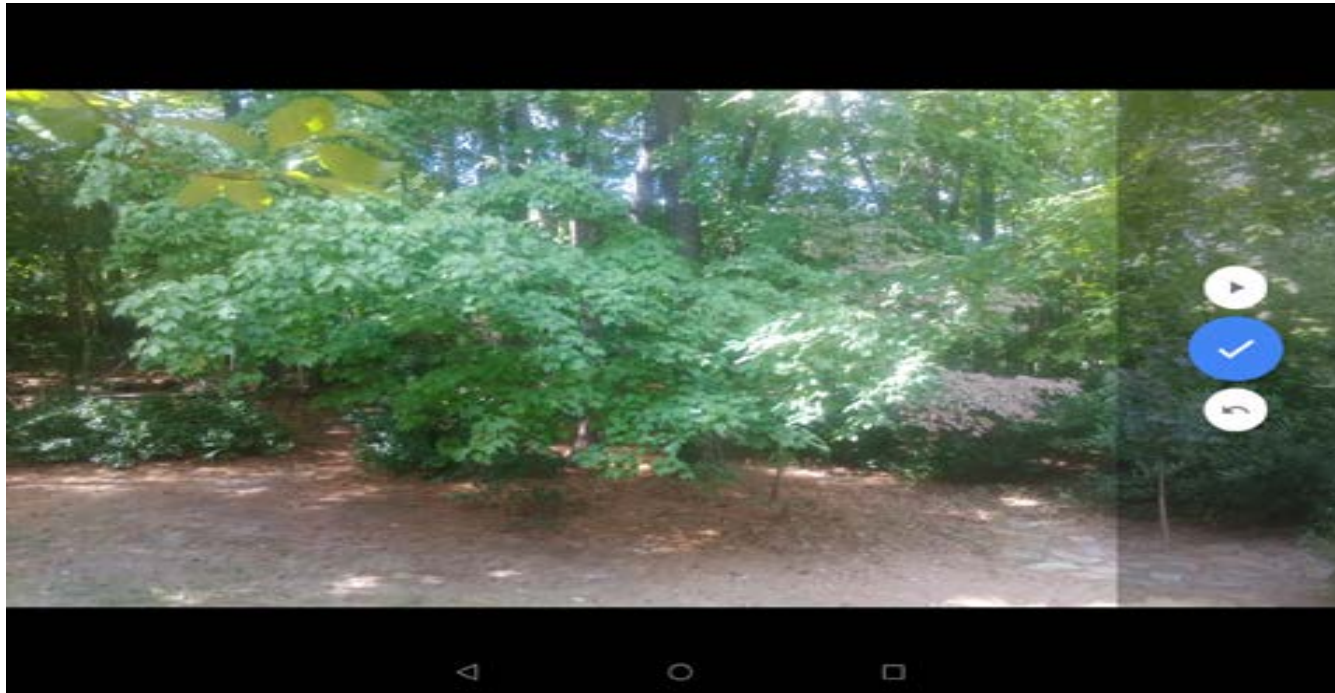
Hands on Example - Video Recording

3. Overrie onActivityResult() method

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent) {  
    val videoUri = data.data  
    if (requestCode == VIDEO_CAPTURE) {  
        if (resultCode == Activity.RESULT_OK) {  
            Toast.makeText(this, "Video saved to:\n" + videoUri, Toast.LENGTH_LONG).show()  
        }  
        else if (resultCode == Activity.RESULT_CANCELED) {  
            Toast.makeText(this, "Video recording cancelled.",  
                Toast.LENGTH_LONG).show() }  
        else {  
            Toast.makeText(this, "Failed to record video",  
                Toast.LENGTH_LONG).show() }  
        }  
    }  
}
```

Hands on Example - Video Recording

- The code example simply displays a toast message indicating the success of the recording intent session. In the event of a successful recording, the path to the stored video file is displayed.
- When executed, the video capture intent will launch and provide the user the opportunity to record video as per the screen below. Refer : CameraIntentsApp



Take Photo and access Gallery

- This example will discuss Camera Intent and make use of Gallery.
- Need to add the following uses-permission and uses-feature in your AndroidManifest.xml

1. Need permission to access your device camera

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

2. If you want to store the image on your mobile device or read an images from Gallery

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

3. Declares a hardware feature that is used by the application.

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

Refer : CameraGalleryApp

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.
- Need to check Runtime Permissions for
 - Manifest.permission.CAMERA
 - Manifest.permission.WRITE_EXTERNAL_STORAGE



Camera button click code

- Camera button click code

```
val REQUEST_IMAGE_CAPTURE :Int = 1
fun camera(view: View){
    val takePictureIntent =
        Intent(MediaStore.ACTION_IMAGE_CAPTURE)
    if (takePictureIntent.resolveActivity(packageManager) != null) {
        startActivityForResult(takePictureIntent,
                               REQUEST_IMAGE_CAPTURE)
    }
}
```

MediaStore.ACTION_IMAGE_CAPTURE : Standard Intent action that can be sent to have the camera application capture an image and return it.

Get the captured image using onActivityResult()

```
override fun onActivityResult(requestCode: Int, resultCode: Int,
                             data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    // Logic to get from Bundle
    if (requestCode == REQUEST_IMAGE_CAPTURE && resultCode == RESULT_OK) {
        /*The Android Camera application encodes the photo in the
        return Intent delivered to onActivityResult()
        as a small Bitmap in the extras, under the key "data".*/
        val extras = data!!.extras
        val imageBitmap = extras.get("data") as Bitmap
        iv.setImageBitmap(imageBitmap)
    } else if (requestCode == 2) { // For Clicking Gallery button
        // Set the selected image from the device image gallery to the ImageView
        // component
        iv.setImageURI(data!!.data)
    }
}
```


Audio Recording

- 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.
- 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 microphone.
- Click STOP button to stop recording.
- Refer : `AudioRecording`

