**Android Multi-media API :-**

Multimedia API is represented by android.media and android.widget packages and android Multimedia API contains all needed classes and interfaces that are required to deal with multi media content like audio, video files.

**Q. How to play and control an audio file in Android?**

**Ans.**  Android provides many ways to control playback of an audio file. One of the way is through a class called MediaPlayer.

**Media player:** This class has been defined in android.media pkg. An object of this class provides facilities to access built in Media player Services for playing & controlling audio files.

An object of MediaPlayer class is created by

Static MediaPlayer create(Context context, int id); method of this class.

Eg-

MediaPlayer mb= MediaPlayer.create(this,R.raw.songName);

We have to use this approach while we are playing an audio file which already has stored in application/ project. (eg. Game).

**Location of song file:-** projectName/res/raw/song.mp3

**OR**

Through “new” keyword

We can also create an object of this class by using of “new” keyword of default Constructor calling.

MediaPlayer mp= new MediaPlayer();

**Note:- This approach is used when we have the extract song from external storage.**

**METHODS OF MEDIAPLAYER CLASS**

1. **public void setDataSource(String Path) :-**  set the data source (file path or http url) to use.
2. **public void prepare() :-**  prepares the player for playback synchronously.
3. **public void start() :-**  it starts or resumes the playback.
4. **public void stop():-** it stops the play back.
5. **public void pause() :-** it pauses the play back.
6. **public boolean isPlaying() :-** checks if Media Player is playing.
7. **public void seekTo(int millis) :-**  seeks to specified time in milliseconds.
8. **public void setLooping(boolean looping) :-**  sets the player for looping or non-looping.
9. **public void selectTrack(int index) :-** it selects a track for specified index.
10. **public int getCurrentPosition() :-** returns the current playback position.
11. **public int getDuration():-** returns duration of the file.
12. **public void setVolume(float leftVolume, float rightVolume):-**  sets the volume of the player.

**Note:-** we can also set a view containing controls(play/pause, prev/next etc) for MediaPlayer by MediaController.

**MediaController Class:-** This class is define in android.widget pkg. An object of this class provides controls for media (MediaPlayer/VideoView) which typically contains the buttons like "Play/Pause", "Rewind", "Fast Forward" and a progress slider.

The way to use this class is to instantiate it programmatically. The MediaController will create a default set of controls and put them in a window floating above the view specified with **setAnchorView(viewObj)**. The window will disappear if it left idle for three seconds and reappear when the user touches the anchor view.

Functions like show() and hide() have no effect when MediaController is created in an xml layout. MediaController will hide and show the buttons according to these rules:

* The "previous" and "next" buttons are hidden until setPrevNextListeners() has been called
* The "previous" and "next" buttons are visible but disabled if setPrevNextListeners() was called with null listeners
* The "rewind" and "fastforward" buttons are shown unless requested otherwise by using the MediaController(Context, boolean) constructor with the boolean set to false

The way to use this class is to instantiate it programmatically.

**MediaController m= new MediaController(Context context);**

**Important Methods –**

|  |  |
| --- | --- |
| void | [setAnchorView](http://developer.android.com/reference/android/widget/MediaController.html#setAnchorView(android.view.View))([View](http://developer.android.com/reference/android/view/View.html) view)  Set the view that acts as the anchor for the control view. |
| void | [setEnabled](http://developer.android.com/reference/android/widget/MediaController.html#setEnabled(boolean))(boolean enabled)  Set the enabled state of this view. |
| void | [setMediaPlayer](http://developer.android.com/reference/android/widget/MediaController.html#setMediaPlayer(android.widget.MediaController.MediaPlayerControl))([MediaController.MediaPlayerControl](http://developer.android.com/reference/android/widget/MediaController.MediaPlayerControl.html) player) |
| void | [setPrevNextListeners](http://developer.android.com/reference/android/widget/MediaController.html#setPrevNextListeners(android.view.View.OnClickListener, android.view.View.OnClickListener))([View.OnClickListener](http://developer.android.com/reference/android/view/View.OnClickListener.html) next, [View.OnClickListener](http://developer.android.com/reference/android/view/View.OnClickListener.html) prev) |
| void | [show](http://developer.android.com/reference/android/widget/MediaController.html#show(int))(int timeout)  Show the controller on screen. |
| void | [show](http://developer.android.com/reference/android/widget/MediaController.html#show())()  Show the controller on screen. |
| boolean | [isShowing](http://developer.android.com/reference/android/widget/MediaController.html#isShowing())() |

**Q. How to play and control video file in Android?**

**Ans.** Android provides facilities to play and control any video file with the help of “VideoView” and “MediaController” classes.

Both classes are define in android.widget packages.

**VideoView:-** An Object of this class provides facility to play and control video player with the help of Mediacontroller.

In order to use video view, we have to define <VideoView> tag in .xml file.

**METHODS OF VIDEO VIEW:-**

1. **public void setMediaController(MediaController Controller):-**  sets the MediaController on VideoView.
2. **public void setVideoURI(Uri uri):-**  set the URI of the video file.
3. **public void start():-**  start the video view.
4. **public void stopPlayBack :-**  stops the playback.
5. **public void pause():-**  pauses the playback.
6. **public void suspend():**  suspend the playback.
7. **public void resume():-** resumes the playback.
8. **public void seek(int mills):-** seek to specified time in milliseconds.

**Q. How to record audio/video in android?**

**Ans.** Every android device has a built in Micro-Phone through which we can record audio and store it, or play it over device/phone. There are many ways to do it but the most common way is through **“MediaRecorder”** class.

**MediaRecorder:-**  This class has been defined in android.media pkg. An object of this class provides facilities to record audio/video and create media file to play later etc.

This document shows you how to write an application that captures audio from a device microphone, save the audio and play it back.

**Note:** The Android Emulator does not have the ability to capture audio, but actual devices are likely to provide these capabilities.

**How to instantiate MediaRecorder?**

**MediaRecorder mr= new MediaRecorder();**

**NOTE:- After recording Media, we can create a sound files that can be played later.**

After creating an object of MediaRecorder class, we have to set the Audio receiving source, output format , encoding format and output file’s location.

**SYNTAX:-**

**mr.setAudioSource(MediaRecorder.AudioSource.MIC);**

**Set o/p format:-**

**mr.setOutputFormat(MediaRecorder.OutputFormat.DEFAULT);**

**Set audio encoder:-**

**mr.setAudioEncoder(MediaRecorder.OutputFormat.AMR\_NB);**

**Save the o/p Files:-**

**mr.setOutputFile(outputFile.getAbsolutePath());**

After specifying AudioSource, Format and its o/p files, we have to call two basic following methods on MediaRecorder class:-

1. mr.prepare();
2. mr.start();

**NOTE:- A part from these Methods , there are other methods in MediaRecorder Class that allows us to more control over Audio/Video Recording.**

**COMMONLY USED METHODS:-**

1. **setAudioSource():-**  this method specifies the source of audio to be recorded.
2. **setVideoSource():-**  this method specifies the source of video to be recorded.
3. **setOutputFormat():-**  this method specifies the audio format in which audio to be stored.
4. **setAudioEncoder():-**  this method specifies the Audio encoder to be used.
5. **setOutputFile():-**  this method configure the path to the file into which the recorded audio is to be stored.
6. **stop():-** this method stops the recording process.
7. **release():-** this method should be called when record instance is not needed.

**NOTE:- we have to take following permission from Android OS by defining in Android.Manifest.xml :-**

**<use permission name= “android.permission.WRITE\_EXTERNAL\_STORAGE >;**

**<use permission name= “android.permission.RECORD\_AUDIO”>;**

**Android camera API :-** Almost android devices have at least one camera. Some devices have front camera and back camera as well.

Using the cameraandroid devices can be done by following two ways by which we can use camera in our application :-

1. **Using existing android camera application in our application.**
2. **We can build our own camera application by using camera API (android.hardware.camera2 pkg added in API 21).**

**Using Existing Camera Application :-** In this case, we nothing to do much except sending/ broadcasting an implicit intent to inbuilt android camera application.

We must use following Action in implicit intent for using inbuilt camera app-

android.provider.MediaStore.ACTION\_IMAGE\_CAPTURE.

OR.

ACTION\_VIDEO\_CAPTURE to launch an existing camera application install on our device.

**Intent i= new Intent(android.provider.MediaStore.ACTION\_IMAGE\_CAPTURE);**

**Working with inbuilt CAMERA application we need to set permission and features in manifest.xml file. So that we have to add following features and permission in android manifest file.**

<use\_permission name=”android.permission.WRITE\_EXTERNAL\_STORAGE”/>

< use\_permission name=” android.permission.CAMERA”/>

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

Adding camera features to your manifest causes Google Play to prevent your application from being installed to devices that do not include a camera or do not support the camera features you specify.

**And We should check whether CAMERA is available or not on device as following-**

## 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. This is of particular importance since not all Android devices include a camera.

The presence or otherwise of a camera can be identified via a call to the PackageManager.hasSystemFeature() method. 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. This can be encapsulated into the following convenience method:

private boolean hasCamera() {

if (getPackageManager().hasSystemFeature(

PackageManager.FEATURE\_CAMERA\_FRONT)){

return true;

} else {

return false;

}

}

## Calling the Video Capture Intent

Use of the video capture intent involves, at a minimum, the implementation of code to call the intent activity and a method (onActivityResult) to handle the return from the activity. The Android built-in video recording intent is represented by MediaStore.ACTION\_VIDEO\_CAPTURE and may be launched as follows:

private static final int VIDEO\_CAPTURE = 101;

Intent intent = new 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. A specific location for the media file may be specified using the putExtra() method of the intent, referencing the MediaStore.EXTRA\_OUTPUT key constant to pass through the target URI value. The following code, for example, specifies that the video should be stored on the SD card in a file named myvideo.mp4:

File mediaFile =

new File(Environment.getExternalStorageDirectory().getAbsolutePath()

+ "/myvideo.mp4");

Intent intent = new Intent(MediaStore.ACTION\_VIDEO\_CAPTURE);

Uri videoUri = Uri.fromFile(mediaFile);

intent.putExtra(MediaStore.EXTRA\_OUTPUT, videoUri);

startActivityForResult(intent, VIDEO\_CAPTURE);

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:

protected void onActivityResult(int requestCode, int resultCode, Intent data) {

if (requestCode == VIDEO\_CAPTURE) {

if (resultCode == RESULT\_OK) {

Toast.makeText(this, "Video saved to:\n" +

data.getData(), Toast.LENGTH\_LONG).show();

} else if (resultCode == RESULT\_CANCELED) {

Toast.makeText(this, "Video recording cancelled.",

Toast.LENGTH\_LONG).show();

} else {

Toast.makeText(this, "Failed to record video",

Toast.LENGTH\_LONG).show();

}

}

}

## Building a Camera App

Some developers may require a camera user interface that is customized to the look of their application or provides special features. Creating a customized camera activity requires more code than [using an intent](http://developer.android.com/guide/topics/media/camera.html#intents), but it can provide a more compelling experience for your users.

**Note: The following guide is for the older, deprecated**[Camera](http://developer.android.com/reference/android/hardware/Camera.html)**API. For new or advanced camera applications, the newer**[android.hardware.camera2](http://developer.android.com/reference/android/hardware/camera2/package-summary.html)**API is recommended.**

The general steps for creating a custom camera interface for your application are as follows:

* **Detect and Access Camera** - Create code to check for the existence of cameras and request access.
* **Create a Preview Class** - Create a camera preview class that extends [SurfaceView](http://developer.android.com/reference/android/view/SurfaceView.html) and implements the [SurfaceHolder](http://developer.android.com/reference/android/view/SurfaceHolder.html) interface. This class previews the live images from the camera.
* **Build a Preview Layout** - Once you have the camera preview class, create a view layout that incorporates the preview and the user interface controls you want.
* **Setup Listeners for Capture** - Connect listeners for your interface controls to start image or video capture in response to user actions, such as pressing a button.
* **Capture and Save Files** - Setup the code for capturing pictures or videos and saving the output.
* **Release the Camera** - After using the camera, your application must properly release it for use by other applications.

|  |  |
| --- | --- |
| **Class** | **Description** |
| **Camera** | It is used to control the camera and take images or capture video from the camera |
| **SurfaceView** | This class is used to present a live camera preview to the user. |

### **Detecting camera hardware**

If your application does not specifically require a camera using a manifest declaration, you should check to see if a camera is available at runtime. To perform this check, use the [PackageManager.hasSystemFeature()](http://developer.android.com/reference/android/content/pm/PackageManager.html" \l "hasSystemFeature(java.lang.String)) method, as shown in the example code below:

/\*\* Check if this device has a camera \*/  
private boolean checkCameraHardware(Context context) {  
    if (context.getPackageManager().hasSystemFeature(PackageManager.FEATURE\_CAMERA)){  
        // this device has a camera  
        return true;  
    } else {  
        // no camera on this device  
        return false;  
    }  
}

Android devices can have multiple cameras, for example a back-facing camera for photography and a front-facing camera for video calls. Android 2.3 (API Level 9) and later allows you to check the number of cameras available on a device using the [Camera.getNumberOfCameras()](http://developer.android.com/reference/android/hardware/Camera.html" \l "getNumberOfCameras()) method.

### **Accessing cameras**

If you have determined that the device on which your application is running has a camera, you must request to access it by getting an instance of [Camera](http://developer.android.com/reference/android/hardware/Camera.html) .

To access the primary camera, use the [Camera.open()](http://developer.android.com/reference/android/hardware/Camera.html" \l "open()) method and be sure to catch any exceptions, as shown in the code below:

/\*\* A safe way to get an instance of the Camera object. \*/  
public static Camera getCameraInstance(){  
    Camera c = null;  
    try {  
        c = Camera.open(); // attempt to get a Camera instance  
    }  
    catch (Exception e){  
        // Camera is not available (in use or does not exist)  
    }  
    return c; // returns null if camera is unavailable  
}

**Caution:** Always check for exceptions when using [Camera.open()](http://developer.android.com/reference/android/hardware/Camera.html" \l "open()). Failing to check for exceptions if the camera is in use or does not exist will cause your application to be shut down by the system.

On devices running Android 2.3 (API Level 9) or higher, you can access specific cameras using [Camera.open(int)](http://developer.android.com/reference/android/hardware/Camera.html#open(int)). The example code above will access the first, back-facing camera on a device with more than one camera.

### **Creating a preview class**

For users to effectively take pictures or video, they must be able to see what the device camera sees. A camera preview class is a [SurfaceView](http://developer.android.com/reference/android/view/SurfaceView.html) that can display the live image data coming from a camera, so users can frame and capture a picture or video.

The following example code demonstrates how to create a basic camera preview class that can be included in a [View](http://developer.android.com/reference/android/view/View.html) layout. This class implements [SurfaceHolder.Callback](http://developer.android.com/reference/android/view/SurfaceHolder.Callback.html) in order to capture the callback events for creating and destroying the view, which are needed for assigning the camera preview input.

/\*\* A basic Camera preview class \*/  
public class CameraPreview extends SurfaceView implements SurfaceHolder.Callback {  
    private SurfaceHolder mHolder;  
    private Camera mCamera;  
  
    public CameraPreview(Context context, Camera camera) {  
        super(context);  
        mCamera = camera;  
  
        // Install a SurfaceHolder.Callback so we get notified when the  
        // underlying surface is created and destroyed.  
        mHolder = getHolder();  
        mHolder.addCallback(this);  
        // deprecated setting, but required on Android versions prior to 3.0  
        mHolder.setType(SurfaceHolder.SURFACE\_TYPE\_PUSH\_BUFFERS);  
    }  
  
    public void surfaceCreated(SurfaceHolder holder) {  
        // The Surface has been created, now tell the camera where to draw the preview.  
        try {  
            mCamera.setPreviewDisplay(holder);  
            mCamera.startPreview();  
        } catch (IOException e) {  
            Log.d(TAG, "Error setting camera preview: " + e.getMessage());  
        }  
    }  
  
    public void surfaceDestroyed(SurfaceHolder holder) {  
        // empty. Take care of releasing the Camera preview in your activity.  
    }  
  
    public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {  
        // If your preview can change or rotate, take care of those events here.  
        // Make sure to stop the preview before resizing or reformatting it.  
  
        if (mHolder.getSurface() == null){  
          // preview surface does not exist  
          return;  
        }  
  
        // stop preview before making changes  
        try {  
            mCamera.stopPreview();  
        } catch (Exception e){  
          // ignore: tried to stop a non-existent preview  
        }  
  
        // set preview size and make any resize, rotate or  
        // reformatting changes here  
  
        // start preview with new settings  
        try {  
            mCamera.setPreviewDisplay(mHolder);  
            mCamera.startPreview();  
  
        } catch (Exception e){  
            Log.d(TAG, "Error starting camera preview: " + e.getMessage());  
        }  
    }  
}

If you want to set a specific size for your camera preview, set this in the surfaceChanged() method as noted in the comments above. When setting preview size, you must use values from [getSupportedPreviewSizes()](http://developer.android.com/reference/android/hardware/Camera.Parameters.html" \l "getSupportedPreviewSizes()). Do not set arbitrary values in the [setPreviewSize()](http://developer.android.com/reference/android/hardware/Camera.Parameters.html" \l "setPreviewSize(int, int)) method.

### **Placing preview in a layout**

A camera preview class, such as the example shown in the previous section, must be placed in the layout of an activity along with other user interface controls for taking a picture or video. This section shows you how to build a basic layout and activity for the preview.

The following layout code provides a very basic view that can be used to display a camera preview. In this example, the [FrameLayout](http://developer.android.com/reference/android/widget/FrameLayout.html) element is meant to be the container for the camera preview class. This layout type is used so that additional picture information or controls can be overlayed on the live camera preview images.

<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="horizontal"  
    android:layout\_width="fill\_parent"  
    android:layout\_height="fill\_parent"  
    >  
  <FrameLayout  
    android:id="@+id/camera\_preview"  
    android:layout\_width="fill\_parent"  
    android:layout\_height="fill\_parent"  
    android:layout\_weight="1"  
    />  
  
  <Button  
    android:id="@+id/button\_capture"  
    android:text="Capture"  
    android:layout\_width="wrap\_content"  
    android:layout\_height="wrap\_content"  
    android:layout\_gravity="center"  
    />  
</LinearLayout>

On most devices, the default orientation of the camera preview is landscape. This example layout specifies a horizontal (landscape) layout and the code below fixes the orientation of the application to landscape. For simplicity in rendering a camera preview, you should change your application's preview activity orientation to landscape by adding the following to your manifest.

<activity android:name=".CameraActivity"  
          android:label="@string/app\_name"  
  
          android:screenOrientation="landscape">  
          <!-- configure this activity to use landscape orientation -->  
  
          <intent-filter>  
        <action android:name="android.intent.action.MAIN" />  
        <category android:name="android.intent.category.LAUNCHER" />  
    </intent-filter>  
</activity>

**Note:** A camera preview does not have to be in landscape mode. Starting in Android 2.2 (API Level 8), you can use the [setDisplayOrientation()](http://developer.android.com/reference/android/hardware/Camera.html" \l "setDisplayOrientation(int)) method to set the rotation of the preview image. In order to change preview orientation as the user re-orients the phone, within the [surfaceChanged()](http://developer.android.com/reference/android/view/SurfaceHolder.Callback.html" \l "surfaceChanged(android.view.SurfaceHolder, int, int, int)) method of your preview class, first stop the preview with [Camera.stopPreview()](http://developer.android.com/reference/android/hardware/Camera.html" \l "stopPreview()) change the orientation and then start the preview again with [Camera.startPreview()](http://developer.android.com/reference/android/hardware/Camera.html" \l "startPreview()).

In the activity for your camera view, add your preview class to the [FrameLayout](http://developer.android.com/reference/android/widget/FrameLayout.html) element shown in the example above. Your camera activity must also ensure that it releases the camera when it is paused or shut down. The following example shows how to modify a camera activity to attach the preview class shown in [Creating a preview class](http://developer.android.com/guide/topics/media/camera.html#camera-preview).

public class CameraActivity extends Activity {  
  
    private Camera mCamera;  
    private CameraPreview mPreview;  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
  
        // Create an instance of Camera  
        mCamera = getCameraInstance();  
  
        // Create our Preview view and set it as the content of our activity.  
        mPreview = new CameraPreview(this, mCamera);  
        FrameLayout preview = (FrameLayout) findViewById(R.id.camera\_preview);  
        preview.addView(mPreview);  
    }  
}

**Note:** The getCameraInstance() method in the example above refers to the example method shown in [Accessing cameras](http://developer.android.com/guide/topics/media/camera.html#access-camera).

### **Capturing pictures**

Once you have built a preview class and a view layout in which to display it, you are ready to start capturing images with your application. In your application code, you must set up listeners for your user interface controls to respond to a user action by taking a picture.

In order to retrieve a picture, use the [Camera.takePicture()](http://developer.android.com/reference/android/hardware/Camera.html" \l "takePicture(android.hardware.Camera.ShutterCallback, android.hardware.Camera.PictureCallback, android.hardware.Camera.PictureCallback)) method. This method takes three parameters which receive data from the camera. In order to receive data in a JPEG format, you must implement an [Camera.PictureCallback](http://developer.android.com/reference/android/hardware/Camera.PictureCallback.html) interface to receive the image data and write it to a file. The following code shows a basic implementation of the [Camera.PictureCallback](http://developer.android.com/reference/android/hardware/Camera.PictureCallback.html) interface to save an image received from the camera.

private PictureCallback mPicture = new PictureCallback() {  
  
    @Override  
    public void onPictureTaken(byte[] data, Camera camera) {  
  
        File pictureFile = getOutputMediaFile(MEDIA\_TYPE\_IMAGE);  
        if (pictureFile == null){  
            Log.d(TAG, "Error creating media file, check storage permissions: " +  
                e.getMessage());  
            return;  
        }  
  
        try {  
            FileOutputStream fos = new FileOutputStream(pictureFile);  
            fos.write(data);  
            fos.close();  
        } catch (FileNotFoundException e) {  
            Log.d(TAG, "File not found: " + e.getMessage());  
        } catch (IOException e) {  
            Log.d(TAG, "Error accessing file: " + e.getMessage());  
        }  
    }  
};

Trigger capturing an image by calling the [Camera.takePicture()](http://developer.android.com/reference/android/hardware/Camera.html" \l "takePicture(android.hardware.Camera.ShutterCallback, android.hardware.Camera.PictureCallback, android.hardware.Camera.PictureCallback)) method. The following example code shows how to call this method from a button [View.OnClickListener](http://developer.android.com/reference/android/view/View.OnClickListener.html).

// Add a listener to the Capture button  
Button captureButton = (Button) findViewById(id.button\_capture);  
captureButton.setOnClickListener(  
    new View.OnClickListener() {  
        @Override  
        public void onClick(View v) {  
            // get an image from the camera  
            mCamera.takePicture(null, null, mPicture);  
        }  
    }  
);

**Note:** The mPicture member in the following example refers to the example code above.

**Caution:** Remember to release the [Camera](http://developer.android.com/reference/android/hardware/Camera.html) object by calling the [Camera.release()](http://developer.android.com/reference/android/hardware/Camera.html" \l "release()) when your application is done using it! For information about how to release the camera, see [Releasing the camera](http://developer.android.com/guide/topics/media/camera.html#release-camera).

|  |  |
| --- | --- |
| **Sr.No** | **Method & Description** |
| 1 | **getCameraInfo(int cameraId, Camera.CameraInfo cameraInfo)**  It returns the information about a particular camera |
| 2 | **getNumberOfCameras()**  It returns an integer number defining of cameras available on device |
| 3 | **lock()**  It is used to lock the camera , so no other application can access it |
| 4 | **release()**  It is used to release the lock on camera , so other applications can access it |
| 5 | **open(int cameraId)**  It is used to open particular camera when multiple cameras are supported |
| 6 | **enableShutterSound(boolean enabled)**  It is used to enable/disable default shutter sound of image capture |

Apart from the preview there are other options of the camera that can be set using the other functions provided by the Camera API

|  |  |
| --- | --- |
| **Sr.No** | **Method & Description** |
| 1 | **startFaceDetection()**  This function starts the face detection in the camera |
| 2 | **stopFaceDetection()**  It is used to stop the face detection which is enabled by the above function |
| 3 | **startSmoothZoom(int value)**  It takes an integer value and zoom the camera very smoothly to that value |
| 4 | **stopSmoothZoom()**  It is used to stop the zoom of the camera |
| 5 | **stopPreview()**  It is used to stop the preview of the camera to the user |
| 6 | **takePicture(Camera.ShutterCallback shutter, Camera.PictureCallback raw, Camera.PictureCallback jpeg)**  It is used to enable/disable default shutter sound of image capture |