Lecture 6

Handling Media in Android Applications

- Mobile Mindset
- Mobile Platforms and Application Development fundamentals
- Introduction to Android Operating System
- Mobile Interface Design Concepts and UI/UX Design Fundamentals
- Main Components of Android Application
- Data Handling in Mobile Platforms
- Handling Media in Android Applications
- Sensors in Android
- Security Aspects of Mobile Application development
- Android Services

Objectives

- At the end of this Lecture, students should be able to
  - ✓Understanding of the Playing Audio / Visual Content
  - ✓Understand the Image Handling Process
  - ✓Understand the Mobile Camera Handling Process

Media Player overview

- The Android multimedia framework includes support for playing variety of common media types, so that you can easily integrate audio, video and images into your applications.

- You can play audio or video from media files stored in your application's resources from standalone files in the file system, or from a data stream arriving over a network connection

- The following classes are used to play sound and video in the Android framework:

MediaPlayer

This class is the primary API for playing sound and video.

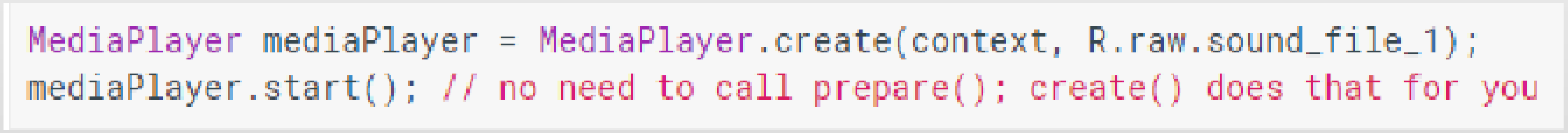
MediaPlayer class can be used to control playback of audio/video files and streams.

AudioManager

This class manages audio sources and audio output on a device.

Using MediaPlayer

- One of the most important components of the media framework is the MediaPlayer class.
- Android is providing MediaPlayer class to access built-in mediaplayer services like playing audio, video
- An object of this class can fetch, decode, and play both audio and video with minimal setup.
- In order to use MediaPlayer, we have to call a static Method **create()** of this class.



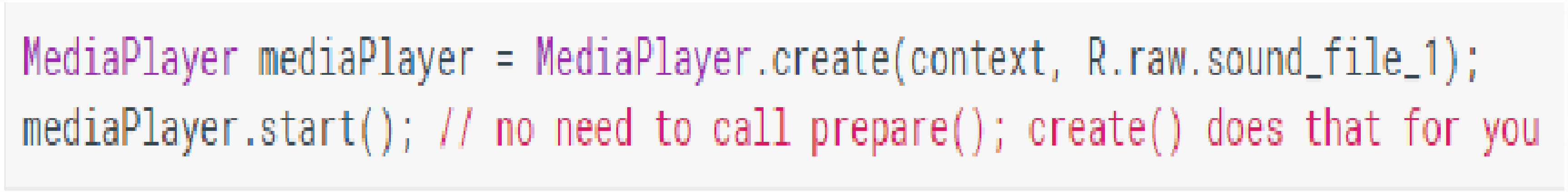
```
MediaPlayer mediaPlayer = MediaPlayer.create(context, R.raw.sound_file_1);
mediaPlayer.start(); // no need to call prepare(); create() does that for you
```

Methods provided by Mediaplayer class for better dealing with audio/video files

Method	Description
isPlaying()	Returns true/false indicating the song is playing or not
seekTo(position)	Move song to that particular position millisecond
getCurrentPosition()	Returns the current position of song in milliseconds
getDuration()	Returns the total time duration of song in milliseconds
reset()	Resets the media player
release()	Releases any resource attached with MediaPlayer object
setVolume(float leftVolume, float rightVolume)	Sets the up down volume for this player

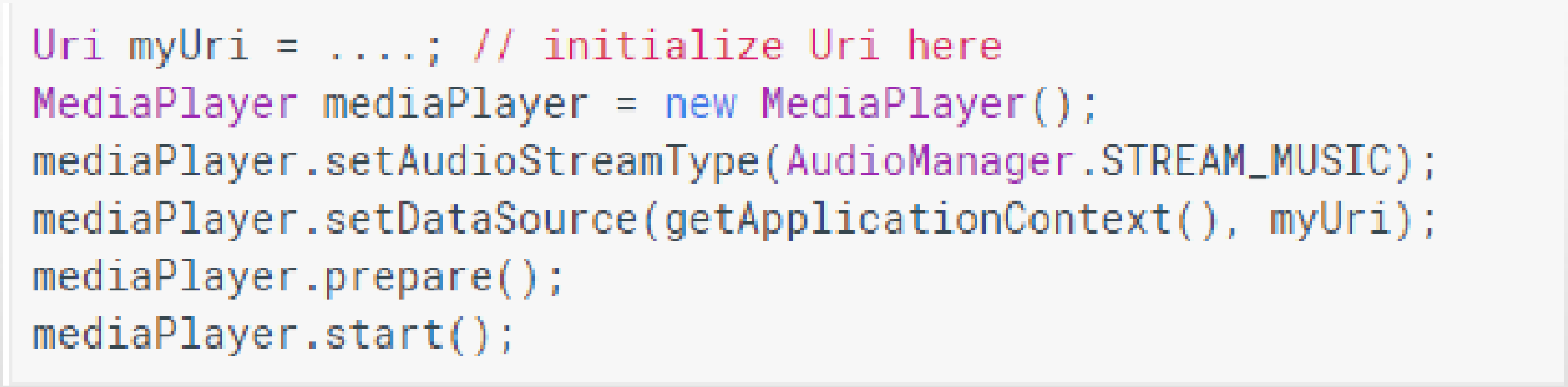
- MediaPlayer supports several different media sources such as:
  - ❖Local resources
  - ❖Internal URIs, such as one you might obtain from a Content Resolver
  - ❖External URLs (streaming)

play audio using local recourses



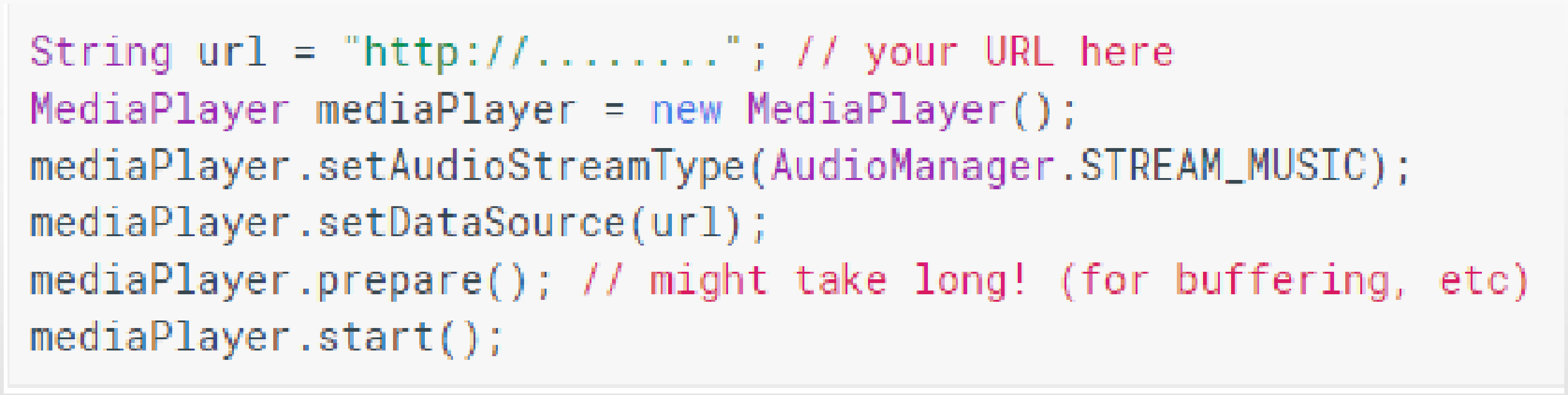
```
MediaPlayer mediaPlayer = MediaPlayer.create(context, R.raw.sound_file_1);
mediaPlayer.start(); // no need to call prepare(); create() does that for you
```

play from a URI available locally in the system



```
Uri myUri = ....; // initialize Uri here
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mediaPlayer.setDataSource(getApplicationContext(), myUri);
mediaPlayer.prepare();
mediaPlayer.start();
```

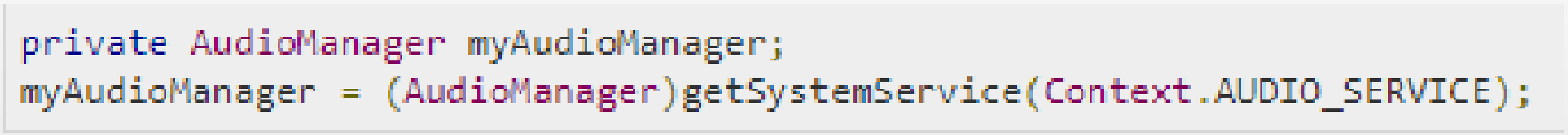
Playing from a remote URL via HTTP streaming



```
String url = "http://........"; // your URL here
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mediaPlayer.setDataSource(url);
mediaPlayer.prepare(); // might take long! (for buffering, etc)
mediaPlayer.start();
```

Using AudioManager

- Can easily control your ringer volume and ringer profile (silent,vibrate,loud) in android.
- Android provides AudioManager class that provides access to these controls.



```
private AudioManager myAudioManager;
myAudioManager = (AudioManager)getSystemService(Context.AUDIO_SERVICE);
```

Image Handling Process in Android

- Make your images look and perform their best on Android using various APIs for bitmaps, drawables, and other types of graphics.

- When you need to display static images in your app, you can use the **Drawable class** and its subclasses to draw shapes and images.

Ways to define and instantiate a Images

- Inflate an image resource (a bitmap file) saved in your project.

- Inflate an XML resource that defines the drawable properties.

To use an image resource, add your file to the res/drawable/ directory of your project

Create drawables from resource images

- Android provides Bitmap class to handle images. This can be found under android.graphics.bitmap
- Supported file types are PNG (preferred), JPG (acceptable), and GIF (discouraged).
- App icons, logos, and other graphics, such as those used in games, are well suited for this technique.
- To use an image resource, add file to the res/drawable/ directory of your project

create a bitmap of image from the imageView.

```
private Bitmap bmp;
private ImageView img;
img = (ImageView)findViewById(R.id.imageView1);
BitmapDrawable  abmp = (BitmapDrawable)img.getDrawable();
```

create bitmap by calling getBitmap() function of BitmapDrawable class.

```
bmp = abmp.getBitmap();
```

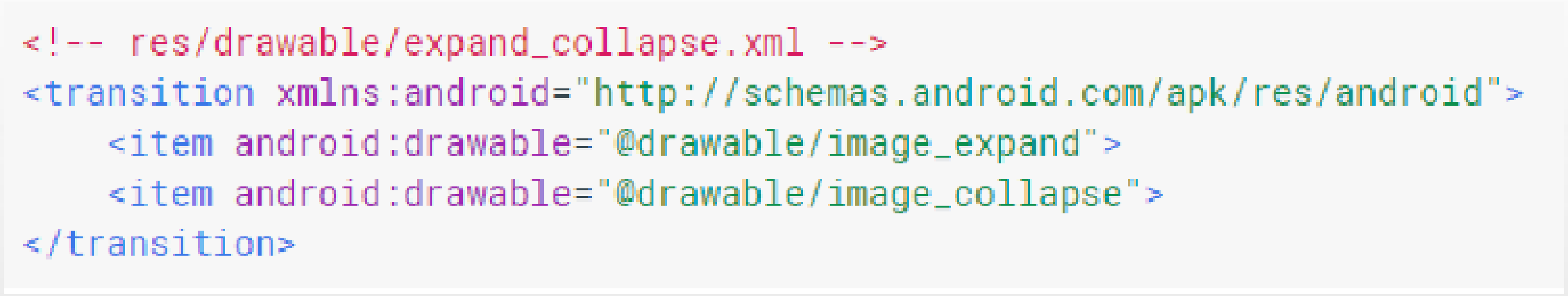
Get pixels from this bitmap and apply processing to it

```
for(int i=0; i<bmp.getWidth(); i++){
    for(int j=0; j<bmp.getHeight(); j++){
        int p = bmp.getPixel(i, j);
    }
}
```

Create drawables from XML resources

- If there is a Drawable object that you'd like to create, which isn't initially dependent on variables defined by your code or user interaction, then defining the Drawable in XML is a good option
- After you've defined your Drawable in XML, save the file in the res/drawable/ directory of your project.

- After you've defined your Drawable in XML, save the file in the res/drawable/ directory of your project.



```
<!-- res/drawable/expand_collapse.xml -->
<transition xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/image_expand">
    <item android:drawable="@drawable/image_collapse">
</transition>
```

Understand the Mobile Camera Handling Process

- The Android framework includes support for various cameras and camera features available on devices.
- Can use existing android camera application in your application
- Can directly using Camera API provided by android in application

Considerations

- Camera Requirement

Is the use of a camera so important to your application that you do not want your application installed on a device that does not have a camera? If so, you should declare the camera requirement in your manifest.

- Quick Picture or Customized Camera

How will your application use the camera? Are you just interested in snapping a quick picture or video clip, or will your application provide a new way to use cameras?

- Foreground Services Requirement

On Android 9 (API level 28) and later, apps running in the background cannot access the camera. Therefore, use the camera either when your app is in the foreground or as part of a foreground service.

- Storage

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

Class	Description
android.hardware.camera2	the primary API for controlling device cameras. It can be used to take pictures or videos
Camera	This class is the older deprecated API for controlling device cameras.
SurfaceView	This class is used to present a live camera preview to the user.
MediaRecorder	This class is used to record video from the camera.
Intent	can be used to capture images or videos without directly using the Camera object.

Manifest declarations

- Camera Permission

Your application must request permission to use a device camera.

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

- Camera Features

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

- Storage Permission

If your application saves images or videos to the device's external storage (SD Card), you must also specify this in the manifest.

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

- Audio Recording Permission

For recording audio with video capture, your application must request the audio capture permission.

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

- Location Permission

If your application tags images with GPS location information, you must request the ACCESS_FINE_LOCATION permission

Use existing android camera application in application

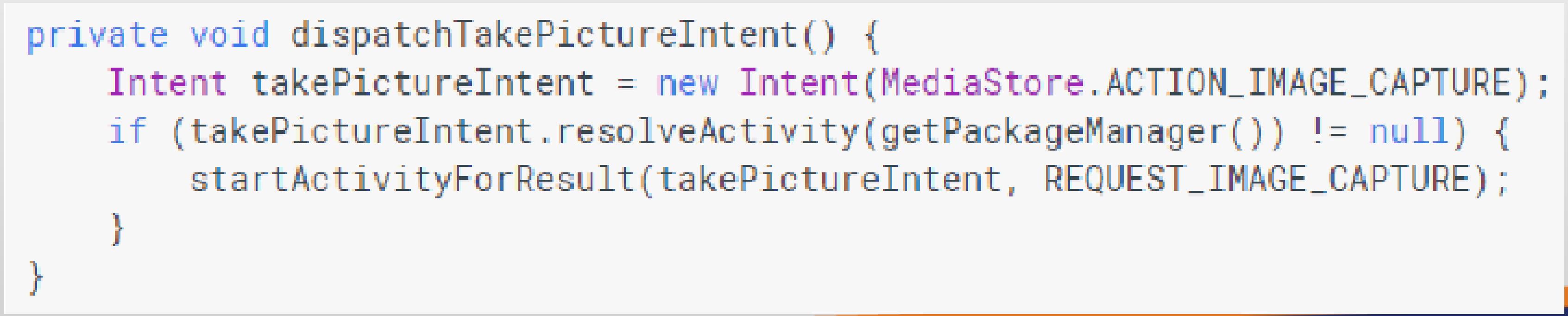
- A quick way to enable taking pictures or videos in your application without a lot of extra code is to use an Intent to invoke an existing Android camera application.
- most Android-powered devices already have at least one camera application installed.

Take a photo with a camera app

This process involves three pieces:

- The Intent
- A call to start the external Activity
- Some code to handle the image data when focus returns to your activity.

use MediaStore.ACTION_IMAGE_CAPTURE to launch an existing camera application installed on your phone.



```java
private void dispatchTakePictureIntent() {
    Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    if (takePictureIntent.resolveActivity(getPackageManager()) != null) {
        startActivityForResult(takePictureIntent, REQUEST_IMAGE_CAPTURE);
    }
}
```

Other useful Intents provided by MediaStore

Name	Description
ACTION_IMAGE_CAPTURE_SECURE	Returns the image captured from the camera , when the device is secured
ACTION_VIDEO_CAPTURE	Calls the existing video application in android to capture video
EXTRA_SCREEN_ORIENTATION	Use to set the orientation of the screen to vertical or landscape
EXTRA_FULL_SCREEN	Use to control the user interface of the ViewImage
INTENT_ACTION_VIDEO_CAMERA	Use to launch the camera in the video mode
EXTRA_SIZE_LIMIT	Use to specify the size limit of video or image capture size

Building a camera app using Camera API

- Some developers may require a camera user interface that is customized to the look of their application or provides special features.

- For new or advanced camera applications, the newer android.hardware.camera2 API is recommended.

steps for creating a custom camera interface for your application

- 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 and implements the SurfaceHolder interface. This class previews the live images from the camera.

- Build a Preview Layout

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 application

Save the full-size photo

- Media files created by users such as pictures and videos should be saved to a device's external storage directory (SD Card) to conserve system space and to allow users to access these files without their device.

```
Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES)
```

- This method returns the standard, shared and recommended location for saving pictures and videos
- If your application is uninstalled by the user, media files saved to this location will not be removed.

```
Context.getExternalFilesDir(Environment.DIRECTORY_PICTURES)
```

- This method returns a standard location for saving pictures and videos which are associated with your application.
- If your application is uninstalled, any files saved in this location are removed

Camera features

- Android supports a wide array of camera features you can control with your camera application, such as picture format, flash mode, focus settings, and many more.
- Metering and focus areas
- Face detection
- Time lapse video