# CSIT242 Mobile Application Development

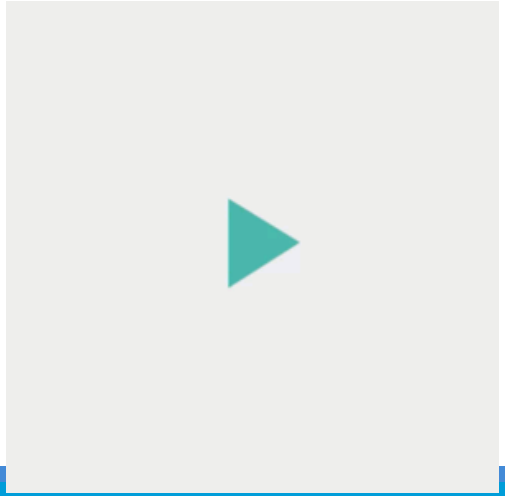
## LECTURE 5 – ANIMATIONS AND MEDIA

# Outline

- Android – Animations, Transitions
- Android – Multimedia Framework
- Android - Camera
- iOS – Animations
- iOS – Core Animation
- iOS – Media Layer – Core Graphics, Core Image, AVFoundation
- iOS - Camera

# Android – Animations

- Animation is actually illusion of motion
  - It is a technique in which successive images which differs very lightly creates effect of motion
- Animations can add visual cues that notify users about what's going on in the app
  - such as when new content loads or new actions become available
- Animations add a polished look to the app - a higher quality look and feel



Source: https://developer.android.com/training/animation/overview

# Android – Animations

- Animate bitmaps - animating a bitmap graphic such as an icon or illustration
  - These animations are defined statically with a drawable resource, but the animation behaviour can be defined at runtime as well
  - For example, animating a play button transforming into a pause button
- Animate UI visibility and motion - changing the visibility or position of views in the layout
  - including subtle animations to help the user understand how the UI is changing
- Physics-based motion - the animations should apply real-world physics so they are natural-looking
  - For example, they should maintain momentum when their target changes, and make smooth transitions during any changes

# Android – Animations

- Animate layout changes - on Android 4.4 (API level 19) and higher, the transition framework is used to create animations for swapping the layout within the current activity or fragment
  - For example, when the user taps an item to see more information, the layout is replaced with the item details
- Animate between activities - on Android 5.0 (API level 21) and higher, can be created animations for transition between activities
  - This is based on the same transition framework as animate layout changes, but it allows creating animations between layouts in separate activities

# Android – Animations

- Generally, an Animation resource can define two types of Android animation:
  - View animations
    - Tween animations - these animations are applied to View objects
    - Frame animations - these display a sequence of drawable images
  - Property animations – these enable you to change the properties of an object, like text size or colour

# Android – View Animations

- View animation system can be used to perform tween animation on Views
  - Tween animation calculates the animation with information such as the start point, end point, size, rotation, and other common aspects of an animation
- A tween animation can perform a series of simple transformations on the contents of a View object
  - For example, a TextView object can move, rotate, grow, or shrink the text
- The `animation package` provides all the classes used in a tween animation
  - Packages that need to be included are: `android.view.animation.Animation` and `android.view.animation.AnimationUtils`
- A sequence of animation instructions that defines the tween animation can be defined by XML or Android code

# Android – View Animations

- The animation instructions define the transformations to occur, when they will occur, and how long they should take to apply
- Transformations can be sequential or simultaneous
  - For example, you can have the contents of a TextView move from left to right, and then rotate 180 degrees, or you can have the text move and rotate simultaneously.
- Each transformation takes a set of common parameters (start time and duration) and a set of parameters specific for that transformation
  - starting size and ending size for size change,
  - starting angle and ending angle for rotation, etc.,
- To make several transformations happen simultaneously
  - they should have the same start time
- To make several transformations happen sequential
  - the start time plus the duration of the preceding transformation need to be calculated

# Android – View Animations

- The animation XML files belongs in the `res/anim/` directory of the Android project
- The file must have a single root element:
  - `<alpha>`
    - attributes: `android:fromAlpha, android:toAlpha`
  - `<scale>`
    - attributes: `android:fromXScale, android:toXScale, android:fromYScale, android:toYScale, android:pivotX, android:pivotY`
  - `<translate>`
    - attributes: `android:fromXDelta, android:toXDelta, android:fromYDelta, android:toYDelta`
  - `<rotate>`
    - attributes: `android:fromDegrees, android:toDegrees, android:pivotX, android:pivotY`
  - interpolator element or
  - `<set>` element that holds groups of these elements (which may include another `<set>`)
- By default, all animation instructions are applied simultaneously; To make them occur sequentially, you must specify the `startOffset` attribute

# Android – View Animations

- Example:

```
/* file spaceship_anim.xml */
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/decelerate_interpolator">
        <scale
            android:fromXScale="1.4"
            android:toXScale="0.5"
            android:fromYScale="1.4"
            android:toYScale="0.5"
            android:pivotX="50%"
            android:pivotY="50%"
            android:startOffset="700"
            android:duration="1000"
            android:fillBefore="false" />
        <rotate
            android:fromDegrees="0"
            android:toDegrees="-360"
            android:pivotX="50%"
            android:pivotY="50%"
            android:startOffset="1700"
            android:duration="4000" />
</set>
```

```
/*in MainActivity.java*/
    ImageView spaceshipImage = (ImageView) findViewById(R.id.imageView);
    Animation animation = AnimationUtils.loadAnimation(getApplicationContext(), R.anim.spaceship_anim);
    spaceshipImage.startAnimation(animation);
```

# Android – Property Animation

- The property animation system is a robust framework that allows animating almost anything
  - It define an animation to change any object property over time, regardless of whether it draws to the screen or not
- A property animation changes a property's (a field in an object) value over a specified length of time

# Android – Property Animation

- The property animation system allows defining the following characteristics of an animation:
  - Duration of an animation
  - Time interpolation - how the values for the property are calculated as a function of the animation's current elapsed time
  - Repeat count and behavior - whether or not to have an animation repeat when it reaches the end of a duration and how many times to repeat the animation - specify whether the animation to play back in reverse
  - Animator sets - group animations into logical sets that play together or sequentially or after specified delays
  - Frame refresh delay - how often to refresh frames of the animation
- The animation XML file belongs in the `res/animator/` directory of the Android project

# Android – Property Animation

- Most of the property animation system's APIs are in `android.animation`
- Some important components (class/interface) of the property animation system are:
  - `ValueAnimator` - keeps track of the animation's timing, such as how long the animation has been running, and computes the value of the property that it is animating
  - `ObjectAnimator` – (subclass of `ValueAnimator`) allows setting a target object and object property to animate - updates the property accordingly when it computes a new value for the animation
  - `AnimatorSet` - provides a mechanism to group animations together so that they run in relation to one another

# Android – Property Animation

- Evaluators - tell the property animation system how to calculate values for a given property - it take the timing data that is provided by an Animator class, the animation's start and end value, and calculate the animated values of the property based on this data
- Time interpolator - defines how specific values in an animation are calculated as a function of time
  - For example, you can specify animations to happen linearly across the whole animation, meaning the animation moves evenly the entire time, or you can specify animations to use non-linear time, for example, accelerating at the beginning and decelerating at the end of the animation
  - Because the view animation system already defines many interpolators in `android.view.animation`, those interpolators can be used in the property animation system as well

# Android – Property Animation

- Animation Listeners can be implemented to listen for important events during an animation's duration:
  - `Animator.AnimatorListener`
    - `onAnimationStart()` - Called when the animation starts
    - `onAnimationEnd()` - Called when the animation ends
    - `onAnimationRepeat()` - Called when the animation repeats itself
    - `onAnimationCancel()` - Called when the animation is canceled
  - `ValueAnimator.AnimatorUpdateListener`
    - `onAnimationUpdate()` - called on every frame of the animation
- Instead of implementing the `Animator.AnimatorListener` interface `AnimatorListenerAdapter` class can be extended
  - The `AnimatorListenerAdapter` class provides empty implementations of the methods that can be overriden

# Android – Property Animations

- Example:

```
/* file sun_swing.xml */
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/accelerate_decelerate_interpolator"
    android:ordering="sequentially" >
    <objectAnimator
        android:duration="3000"
        android:propertyName="x"
        android:repeatCount="infinite"
        android:repeatMode="reverse"
        android:valueTo="-400"
        android:valueType="floatType" />
</set>
```

```
/* file wheel_spin.xml */
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
    android:interpolator="@android:anim/accelerate_decelerate_interpolator"
    android:ordering="sequentially" >
    <objectAnimator
        android:duration="3000"
        android:propertyName="rotation"
        android:repeatCount="infinite"
        android:repeatMode="reverse"
        android:valueTo="180"
        android:valueType="floatType" />
</set>
```

# Android – Property Animations

- Example:

```
/*in MainActivity.java*/
    //wheel animation
ImageView wheel = (ImageView)findViewById(R.id.wheel);
//load the wheel animation
AnimatorSet wheelSet = (AnimatorSet)
AnimatorInflater.loadAnimator(this, R.animator.wheel_spin);
//set the view as target
wheelSet.setTarget(wheel);
//start the animation
wheelSet.start();

//get the sun view
ImageView sun = (ImageView)findViewById(R.id.sun);
//load the sun movement animation
AnimatorSet sunSet = (AnimatorSet)
AnimatorInflater.loadAnimator(this, R.animator.sun_swing);
//set the view as target
sunSet.setTarget(sun);
//start the animation
sunSet.start();
```

```
//darken sky
ObjectAnimator skyAnim =
ObjectAnimator.ofInt(findViewById(R.id.car_layout),
     "backgroundColor", Color.rgb(0x66, 0xcc, 0xff),
                        Color.rgb(0x00, 0x66, 0x99));
skyAnim.setDuration(3000);
skyAnim.setRepeatCount(ValueAnimator.INFINITE);
skyAnim.setRepeatMode(ValueAnimator.REVERSE);
skyAnim.setEvaluator(new ArgbEvaluator());
skyAnim.start();

//move clouds
ObjectAnimator cloudAnim =ObjectAnimator.ofFloat(
             findViewById(R.id.cloud1), "x",   350);
cloudAnim.setDuration(3000);
cloudAnim.setRepeatCount(ValueAnimator.INFINITE);
cloudAnim.setRepeatMode(ValueAnimator.REVERSE);
cloudAnim.start();

//other cloud
ObjectAnimator cloudAnim2 = ObjectAnimator.ofFloat(
               findViewById(R.id.cloud2), "x", 300);
cloudAnim2.setDuration(3000);
cloudAnim2.setRepeatCount(ValueAnimator.INFINITE);
cloudAnim2.setRepeatMode(ValueAnimator.REVERSE);
cloudAnim2.start();
```

# Android – Property Animation vs View Animation

- The view animation system provides the capability to only animate View objects, so to animate non-View objects, additional own code is needed
- The view animation system only exposes a few aspects of a View object to animate, such as the scaling and rotation of a View
- View animation system only modifies where the View was drawn, and not the actual View itself

- The property animation system can animate any property of any object (Views and non-Views) and the object itself is actually modified
- The property animation system is more robust in the way it carries out animation
  - Animators can be assign to the properties to be animated (colour, position, size) and can be defined aspects of the animation as interpolation and synchronization of multiple animators

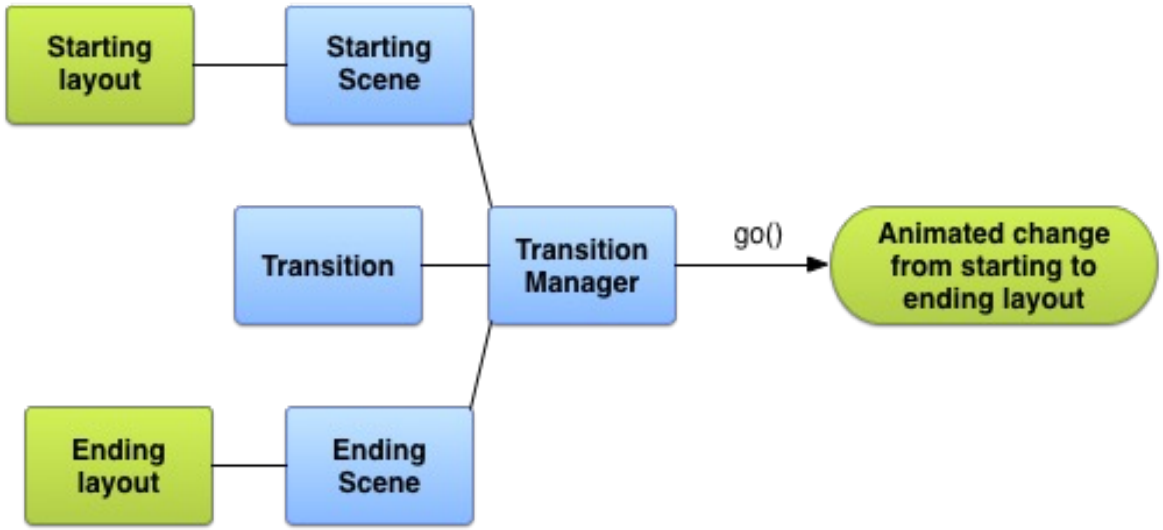
- The view animation system takes less time to setup and requires less code to write
- Both animation systems can be used for different situations

# Android – Transition

- `android.transition` package enables "scenes & transitions" functionality for view hierarchies
- Android's transition framework allows animation of all kinds of motion in the UI by simply providing the starting layout and the ending layout
  - the transition framework figures out how to animate from the starting layout to the ending layout based on the selected type of animation (e.g. fade the views in/out or change the view sizes)
- The transition framework includes the following features:
  - Group-level animations - apply one or more animation effects to all of the views in a view hierarchy
  - Built-in animations - use predefined animations for common effects such as fade out or movement
  - Resource file support - load view hierarchies and built-in animations from layout resource files
  - Lifecycle callbacks - receive callbacks that provide control over the animation and hierarchy change process

# Android – Transition

- The basic process to animate between two layouts is as follows:
  - Create a `Scene` object - for the starting layout and the ending layout
    - The starting layout's scene is often determined automatically from the current layout
  - Create a `Transition` object - define what type of animation should be applied
  - Call `TransitionManager.go()` - the system runs the animation to swap the layouts




Source: https://developer.android.com/training/transitions

# Android – Multimedia framework

- The Android multimedia framework includes support for playing variety of common media types, allowing easy integration of audio, video and images into the applications
- By using Android APIs, the app can offer playing audio or video from media files stored in:
  - the application's resources (raw resources)
  - from standalone files in the file system
  - or from a data stream arriving over a network connection
- Classes that are used to play sound and video in the Android framework are:
  - `MediaPlayer` - this class is the primary API for playing sound and video
  - `AudioManager` - this class manages audio sources and audio output on a device

# Android – Media Player

- `MediaPlayer` class - one of the most important components of the media framework
- An object of this class can fetch, decode, and play both audio and video with minimal setup
- It supports different media sources:
  - Local resources
  - Internal URIs (e.g. obtained from a Content Resolver)
  - External URLs (streaming)

# Android – Media Player

- Depending on the files that will be used and the related features used by the MediaPlayer, some permissions need to be requested
- The manifest file must have the appropriate declarations for:
  - Internet Permission (network access) - if the app is using MediaPlayer to stream network-based content

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

  - Wake Lock Permission - If the app media player needs to keep the screen from dimming or the processor from sleeping - uses the methods `MediaPlayer.setScreenOnWhilePlaying()` or `MediaPlayer.setWakeMode()` methods

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

# Android – Media Player

- Example:
  - play audio that is available as a local raw resource (saved in the application's `res/raw/` directory)

```
MediaPlayer mediaPlayer = MediaPlayer.create(context,
                                             R.raw.my_song);
mediaPlayer.start(); // Starts or resumes playback

// no need to call prepare() - create(..) is doing that
//mediaPlayer.pause() - pause the player
//mediaPlayer.stop() - stop the player
//mediaPlayer.seekTo(int) - position in msec
```

*A "raw" resource is a file that the system does not try to parse in any particular way. It should be a properly encoded and formatted media file in one of the supported formats.*

# Android – Media Player

- Example:
  - play audio from a URI available locally in the system (can be obtained through a Content Resolver or other way)

```
Uri myUri = ....; // initialize Uri here
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setAudioAttributes(
    new AudioAttributes.Builder()
        .setContentType(AudioAttributes.CONTENT_TYPE_MUSIC)
        .setUsage(AudioAttributes.USAGE_MEDIA)
        .build()
);
mediaPlayer.setDataSource(getApplicationContext(), myUri);
mediaPlayer.prepare(); // Prepares the player for
                       //playback, synchronously
mediaPlayer.start();
```

# Android – Media Player

- Example:
  - play audio from a remote URL via HTTP streaming

```
String url = "http://........";
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setAudioAttributes(
    new AudioAttributes.Builder()
        .setContentType(AudioAttributes.CONTENT_TYPE_MUSIC)
        .setUsage(AudioAttributes.USAGE_MEDIA)
        .build()
);mediaPlayer.setDataSource(url);
mediaPlayer.prepare();
// might take long! (for fetching, buffering)
mediaPlayer.start();
```

*If a URL is passed for streaming an online media file, the file must be capable of progressive download.*

# Android – Media Player

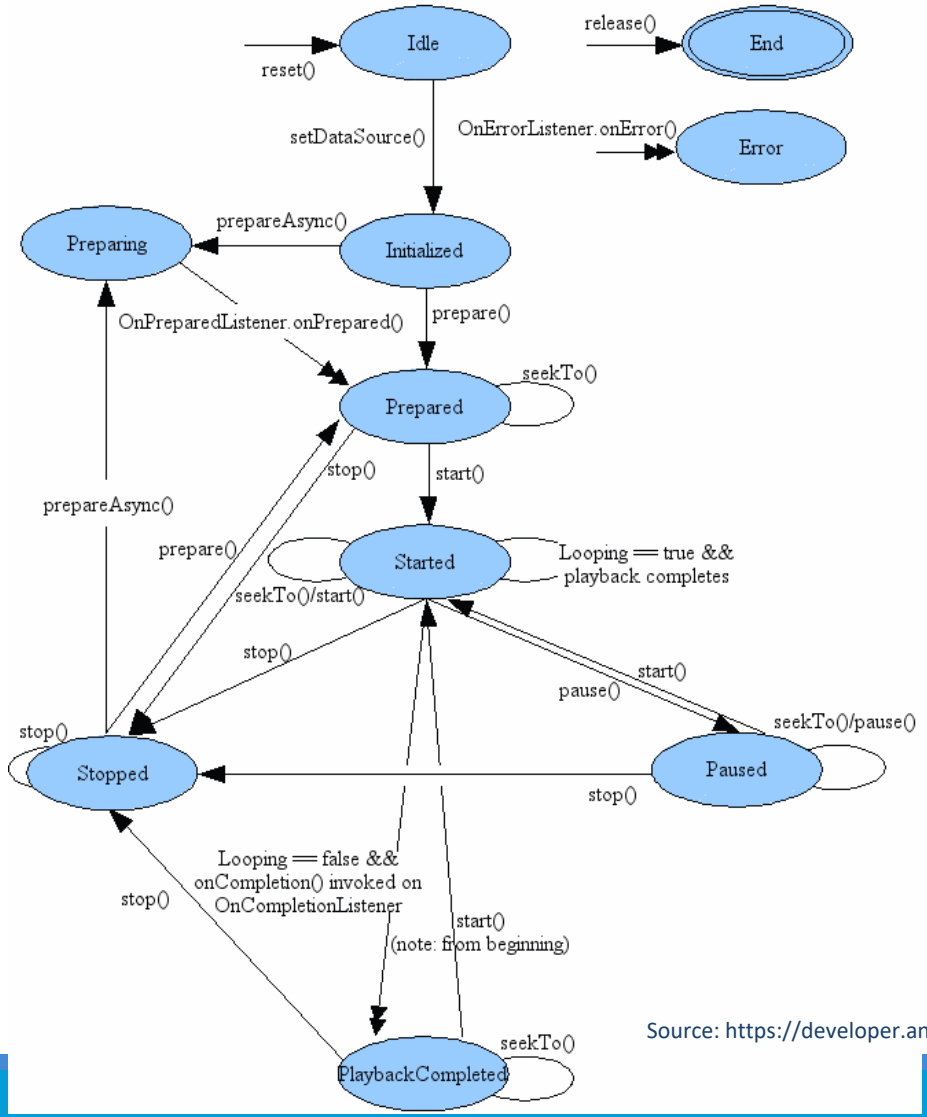
- The call to `prepare()` can take a long time to execute, because it involve fetching and decoding media data
  - Should never be called from the application's UI thread
- Another way to overcome this is using the `prepareAsync()` method that starts preparing the media in the background and returns immediately
  - `MediaPlayer.OnPreparedListener, setOnPreparedListener(), onPrepared()`
- MediaPlayer is a state-based - has an internal state and certain operations are only valid when the player is in specific states
  - Calling MediaPlayer methods from the wrong state is a common cause of bugs
- MediaPlayer `release()` method should be called (when the instance is no longer needed) to make sure that any system resources allocated to it are properly released:

```
mediaPlayer.release();
mediaPlayer = null;
```

# Android – Media Player

- The media can be played in the background (even when the application is not onscreen)
- For this purpose the app needs to start Service that will control the MediaPlayer instance
- All work in a Service is done in a single thread by default - if an activity and a service are running from the same application, they use the same thread (the "main thread")
  - When using a MediaPlayer from the main thread, should be called `prepareAsync()` rather than `prepare()`, and implement a `MediaPlayer.OnPreparedListener` (`onPrepared()` method) in order get notification when the preparation is complete and the playing can start

# Android – Media Player




Source: https://developer.android.com/reference/android/media/MediaPlayer

# Android – Media Player

- When designing applications that play media in the background, the device may go to sleep while your service is running
  - In order to ensure that the service continues to run under those conditions, "wake locks“ should be used
- A wake lock is a way to signal to the system that the application is using some feature that should stay available even if the phone is idle
  - Possible solution is to call the `setWakeMode()` method when initializing your MediaPlayer

```
mMediaPlayer = new MediaPlayer();
// ... other initialization here ...
mMediaPlayer.setWakeMode(getApplicationContext(),
                         PowerManager.PARTIAL_WAKE_LOCK);
```

*If you are streaming media over the network and you are using Wi-Fi, you probably want to hold a WifiLock as well!*

# Android – Media Player

- When a MediaPlayer is used as the main activity where user would like to have full control of it, should be used as Foreground service
- A foreground service holds a higher level of importance within the system - the system will almost never kill the service, because it is of immediate importance to the user
- When running in the foreground, the service also must provide a status bar notification to ensure that users are aware of the running service and allow them to open an activity that can interact with the service

*More readings: https://developer.android.com/guide/components/services*

# Android – Supported media format

- Audio

| Format/Codec | File Type |
|---|---|
| AAC LC | 3GPP (.3gp)<br>.mp4, .m4a<br>.aac |
| HE-AACv1, HE-AACv2, xHE-AAC | |
| AAC ELD | MPEG-TS(.ts) |
| AMR-NB and AMR-WB | 3GPP (.3gp) |
| FLAC | .flac only |
| MP3 | .mp3 |
| MIDI | .mid, .xmf, .mxmf, .rtttl, .rtx, .ota, .imy |
| Vorbis | .ogg and .mkv |
| PCM/WAVE | .wav |
| Opus | .mkv |

Source: https://developer.android.com/guide/topics/media/media-formats

# Android – Supported media format

- Video

| Format/Codec | File Type |
|---|---|
| H.263 | .3gp, mp4 |
| H.264 AVC | .3gp, .mp4, .ts |
| H.265 HEVC | .mp4 |
| MPEG-4 SP | .3gp |
| VP8 | .webm, .mkv |
| VP9 | |
| AV1 | .mp4, .mkv |

Source: https://developer.android.com/guide/topics/media/media-formats

# Android – Camera

- The Android framework includes support for various cameras and camera features available on devices, allowing to capture pictures and videos in the applications
- For performing basic camera actions (capturing a photo or video) by using the device's default camera application, camera intents should be used
- For the implementation of camera functionality in an Android app, there are three main options:
    - CameraX (recommended)
    - Camera2
    - Camera (deprecated)

# Android – Camera

- Use of an Intent to invoke an existing Android camera app - a quick way to enable taking pictures or videos in the application without writing a lot of code
- A camera intent makes a request to capture a picture or video clip through an existing camera app and then returns control back to your application
  - Create the Intent
  - Start the external Activity
  - Handle the image data/video when focus returns to the activity

# Android – Camera

- Camera2 is the low-level Android camera package (it replaces the deprecated Camera class)
- Camera2 provides in-depth controls for complex use cases, but requires managing device-specific configurations
  - e.g. use multiple camera streams simultaneously

# Android – Camera

- CameraX provides a consistent, easy-to-use API that works across the vast majority of Android devices
- Advantages:
  - Broad device compatibility (backward-compatibility to Android 5.0)
  - Ease of use, by supporting
    - Preview
    - Image analysis
    - Image capture
    - Video capture
  - Consistency across devices (consistent API experience across different device types and manufacturers)
  - Camera extensions (optional Extensions API)

# Android – Camera

- Camera Permission - request permission to use a device camera
  - This permission do not need to be included if the camera is used by invoking an existing camera app

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

- Storage Permission – declare use of device's external storage for devices running Android 9 and below

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

- CameraX has minimum version requirements for *Android API level 21* and *Android Architecture Components 1.1.1*

# iOS – Animations

- Animation is the visible change of an attribute over time
  - The changing attribute might be positional: something moves or changes size; But other kinds of attribute can animate as well
- Animation is crucial to the character of the iOS interface

# iOS – Animations

- `UIImageView` provides a form of very simple animation
- **`UIImageView`** can be supplied with an array of **`UIImages`**, as the value of its **`animationImages`** or **`highlightedAnimation`** - images property
  - This array represents the "frames" of a simple cartoon so when the **`startAnimating`** message is sent, the images are displayed at a frame rate determined by the **`animationDuration`** property and repeating as many times as specified by the **`animationRepeatCount`** property (the default is 0, meaning to repeat forever) or until the **`stopAnimating`** message is received
- Before and after the animation, the image view continues displaying its `image` (or `highlightedImage`)
- An image itself can be an animated image - the difference is that an animated image is always animating, repeating its sequence as long as it appears in the interface

# iOS – Animations

- `UIView` can be animated directly for a limited range of properties (alpha, bounds, center, frame, transform, and in some cases backgroundColor)
- The view animation API has evolved historically in three distinct major stages

- *1. Begin and commit*
  - A view animation constructed imperatively using a sequence of UIView class methods

    For example:

```
UIView.beginAnimations(nil, context: nil)
UIView.setAnimationDuration(1)
self.v.backgroundColor = .red
UIView.commitAnimations()
```

# iOS – Animations

- *2. Block-based animation*
  - Starting with iOS 4, the entire operation of configuring a view animation was reduced to a single UIView class (static) method, to which we need to pass a block in which is set an animatable property

  **`animate(withDuration:delay:options:animations:completitions)`**

  For example:

```
UIView.animate(withDuration:1) {
   self.v.backgroundColor = .red
}
```

- Block-based animation completely supersedes begin-and-commit animation

*In a `UIView.animate` class method, the options: argument combines additional options: timing curve, repeat, autoreverse.*

# iOS – Animations

- New solution in iOS 13, is to omit `.autoreverse` and `.repeat` from the animation options, and instead call another UIView class method

```
modifyAnimations(withRepeatCount:autoreverses:)
```

inside the animations function containing the actual animations:

```
let xorig = self.v.center.x
UIView.animate(withDuration:1, delay: 0, options: [], animations: {
        UIView.modifyAnimations(withRepeatCount: 3, autoreverses: true) {
                                                    self.v.center.x += 100} },
        completion: { _ in self.v.center.x = xorig })
-----
let xorig = self.v.center.x
 UIView.animate(withDuration:1){
        UIView.modifyAnimations(withRepeatCount: 3, autoreverses: true) {
                self.v.center.x += 100}
        } completion: { _ in self.v.center.x = xorig }
```

# iOS – Animations

- *3. Property animator*
  - In iOS 10 was introduced the property animator (UIViewPropertyAnimator)
  - It receives an animations function in which an animatable property/ies is set

    For example:

```
let anim = UIViewPropertyAnimator(duration: 1, curve: .linear)
{
    self.v.backgroundColor = .red
}
anim.startAnimation()
```

  - Instead of the standard initializers, If we want to start the animations immediately after the creation of the animator, the following method can be used

**runningPropertyAnimator(withDuration:delay:options:animations:completion:)**

    - This method creates the property animator object, configures it, and calls its startAnimation() method

# iOS – Animations

- The property animator does not supersede block-based animation - it supplements and expands it
- For certain kinds of animation (repeating animation, autoreversing animation, transition animation) the block-based animation is more applicable
- The property animator advantages
  - a full range of timing curves : .curveEaseInOut, .curveEaseIn, .curveEaseOut, .curveLinear
  - multiple completion functions that contains commands to be executed when the animation finishes
  - state of the animation: .inactive, .active, .stopped
  - ability to pause, resume, reverse (`isReversed` property), and interact by touch with a view animation

*If there is need for the delay of the animation can be used:* `func addAnimations(() -> Void, delayFactor: CGFloat) or startAnimation(afterDelay: TimeInterval)`

# iOS – Animations

- Example

```
var scaleFactor: CGFloat = 2
var angle: Double = 180
let timing = UICubicTimingParameters(animationCurve: .easeIn)
let animator = UIViewPropertyAnimator(duration:2.5,
                                      timingParameters:timing)
animator.addAnimations {
      let scaleTrans = CGAffineTransform(scaleX: self.scaleFactor,
                                         y: self.scaleFactor)
      let rotateTrans = CGAffineTransform( rotationAngle:
                                CGFloat(self.angle * Double.pi/180))
      self.smileImg.transform = scaleTrans.concatenating(rotateTrans)
      self.angle =  ( self.angle==180 ? 360 : 180)
      self.scaleFactor = (self.scaleFactor == 2 ? 1 : 2)
      self.label?.backgroundColor = UIColor.purple
}
animator.addCompletion {_ in
               self.label?.backgroundColor = UIColor.green }
animator.startAnimation()
```

# iOS – Transitions

- A transition is an animation that emphasizes a view's change of content
- Transitions are ordered using one of two `UIView` class methods:
  - `transition(with:duration:options:animations:completion:)`
  - `transition(from:to:duration:options:completion:)`
  - The transition animation types are expressed as part of the options: bitmask:
    - `.transitionFlipFromLeft, .transitionFlipFromRight`
    - `.transitionCurlUp, .transitionCurlDown`
    - `.transitionFlipFromBottom, .transitionFlipFromTop`
    - `.transitionCrossDissolve`

# iOS – Transitions

- For example

a `UIImageView` containing an image of smiley curl up / flips over as its image changes to a other smiley face

```
let opts: UIView.AnimationOptions = .transitionCurlUp

UIView.transition(with: self.smileImg, duration:0.8, options: opts,
            animations: {
                if self.smileImg.tag == 1 {
                    self.smileImg.image = UIImage(named: "smile2")
                    self.smileImg.tag = 2}
                else{
                    self.smileImg.image = UIImage(named: "smile1")
                    self.smileImg.tag = 1
                }
            })
```

# iOS – Core Animation

- Core Animation is the fundamental underlying iOS animation technology
- Core Animation is explicit layer animation, and revolves primarily around the **`CAAnimation`** class and its subclasses, which allows creating far more elaborate specifications of an animation than other animations
- Core Animation works on a view's underlying layer - it is the only way to apply full-on layer property animation to a view
  - permits fine control over the intermediate values and timing of an animation
  - allows animations to be grouped into complex combinations
  - provides transition animation effects that aren't available otherwise, such as new content "pushing" the previous content out of a layer

*Animating a view's underlying layer with Core Animation is layer animation, not view animation — so you don't get any automatic layout of that view's subviews. This can be a reason for preferring view animation.*

# iOS – Core Animation

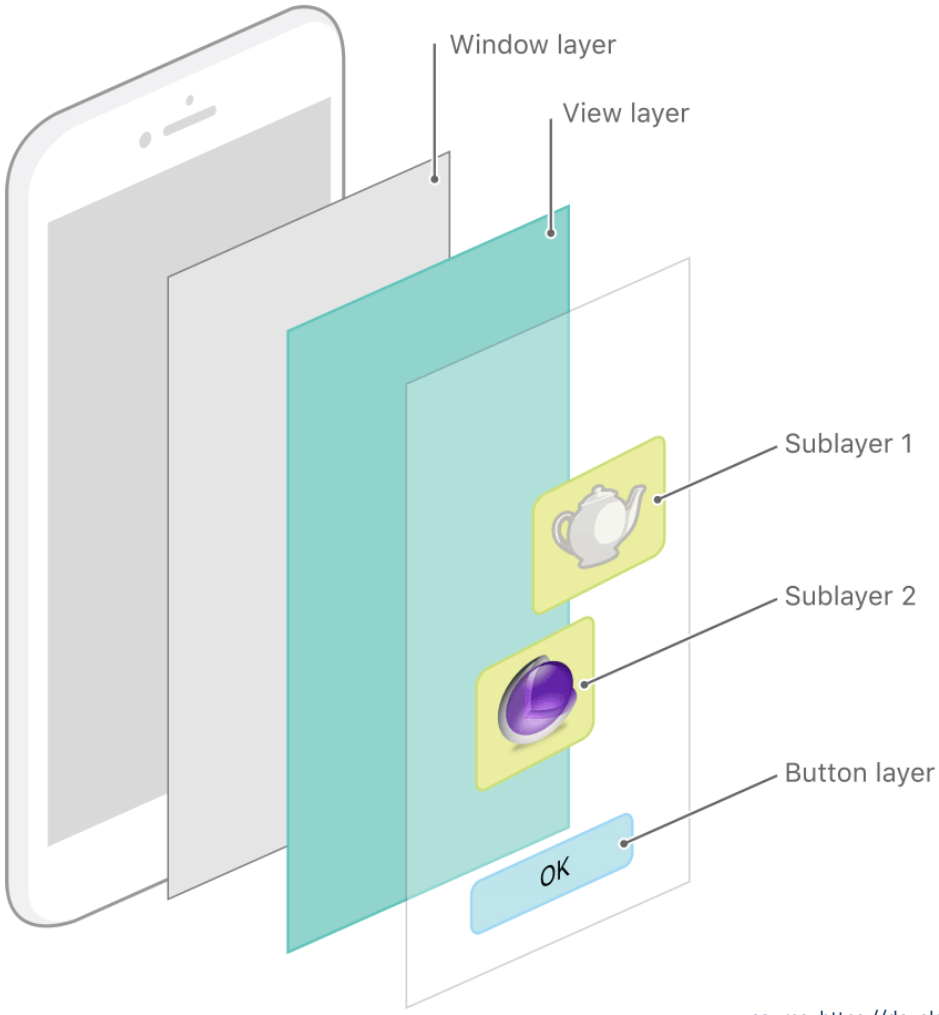
- Core Animation is a graphics rendering and animation infrastructure that can be used for animation of the views and other visual elements in the app
  - It is not a drawing system itself and is not replacement for the app's views
- Core Animation is doing most of the work required to draw each frame of an animation
  - only few animation parameters (such as the start and end points) needs to be configured and tell Core Animation to start
- Core Animation is handing most of the actual drawing work
  - It achieves this behavior by caching the contents of views into bitmaps that can be manipulated directly by the graphics hardware
  - This automatic graphics acceleration results in high frame rates and smooth animations without burdening the CPU and slowing down the app

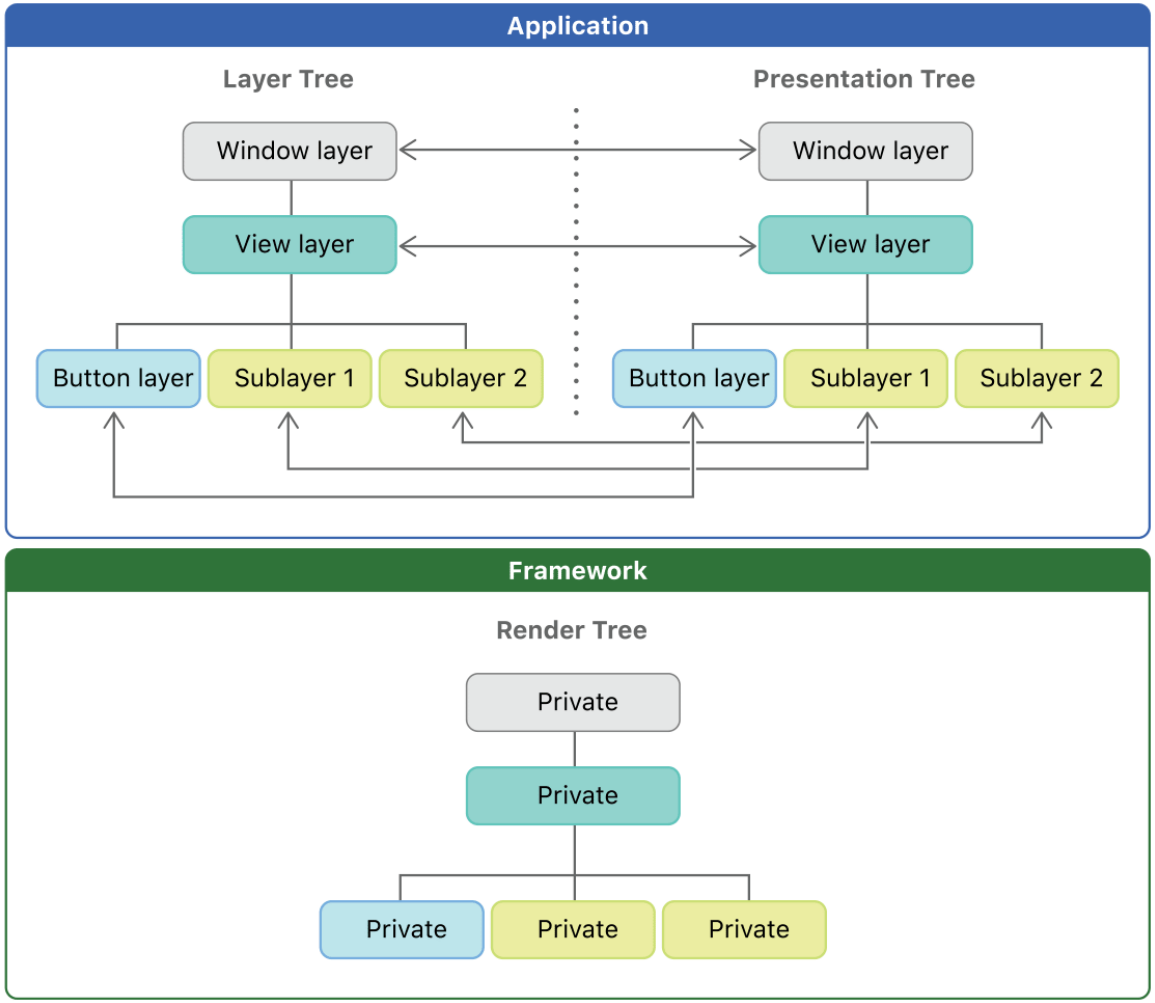
# iOS – Core Animation

- An app using Core Animation has three sets of layer objects
- Each set of layer objects has a different role in making the content of the app appear onscreen:
  - Objects in the model layer tree (or simply “layer tree”) are the ones the app interacts with the most; The objects in this tree are the model objects that store the target values for any animations; Whenever you change the property of a layer, you use one of these objects
  - Objects in the presentation tree contain the in-flight values for any running animations; The objects in the presentation tree reflect the current values as they appear onscreen
  - Objects in the render tree perform the actual animations and are private to Core Animation

*You should never modify the objects in the presentation tree. Instead, you use these objects to read current animation values, perhaps to create a new animation starting at those values.*

# iOS – Core Animation




source: https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CoreAnimation_guide/CoreAnimationBasics/CoreAnimationBasics.html

# iOS – Core Animation

- Layers are not a replacement for the app's views, so a visual interface cannot be created based solely on layer objects
- Layers provide infrastructure for the views
  - Layers make it easier and more efficient to draw and animate the contents of views and maintain high frame rates while doing so
  - Layers do not handle events, draw content, participate in the responder chain, or do many other things
  - Every app must still have one or more views to handle those kinds of interactions
- In iOS, every view is backed by a corresponding layer object

# iOS – Media Layer

- Technologies of the Media layer allow 2D and 3D graphics, animations, image effects, and professional-grade audio and video functionality into the app
- This layer has the many frameworks like:
  - Core Graphics – provides support for managing the graphic context and for drawing (primitive shapes, images, text, …)
  - Core Animation - provides support for animating views and other content
  - Core Image - provides support for manipulating video and still images
  - OpenGL - provide support for 2D and 3D rendering using hardware-accelerated interfaces
  - Core Text - provides a text layout and rendering engine
  - AVFoundation - provides services for audiovisual media
  - Core Audio - provides sophisticated services for manipulating multichannel audio
  - …

# iOS – Core Graphics

- The Core Graphics framework is based on the Quartz advanced drawing engine
- It provides low-level, lightweight 2D rendering with high-fidelity output
- Core graphics framework can be used to handle
  - path-based drawing
  - transformations
  - color management
  - offscreen rendering
  - patterns
  - gradients and shadings
  - image data management
  - image creation, and image masking
  - PDF document creation, display, and parsing

# iOS – Core Image

- The Core Image is an image processing and analysis technology that provides high-performance (near real-time) processing for still and video images
- It allows use of many built-in image filters to process images and build complex effects by chaining filters
- Core Image hides the details of low-level graphics processing by providing an easy-to-use application programming interface (API)
- Core Image operates on image data types from the Core Graphics, Core Video, and Image I/O frameworks, using either a GPU or CPU rendering path

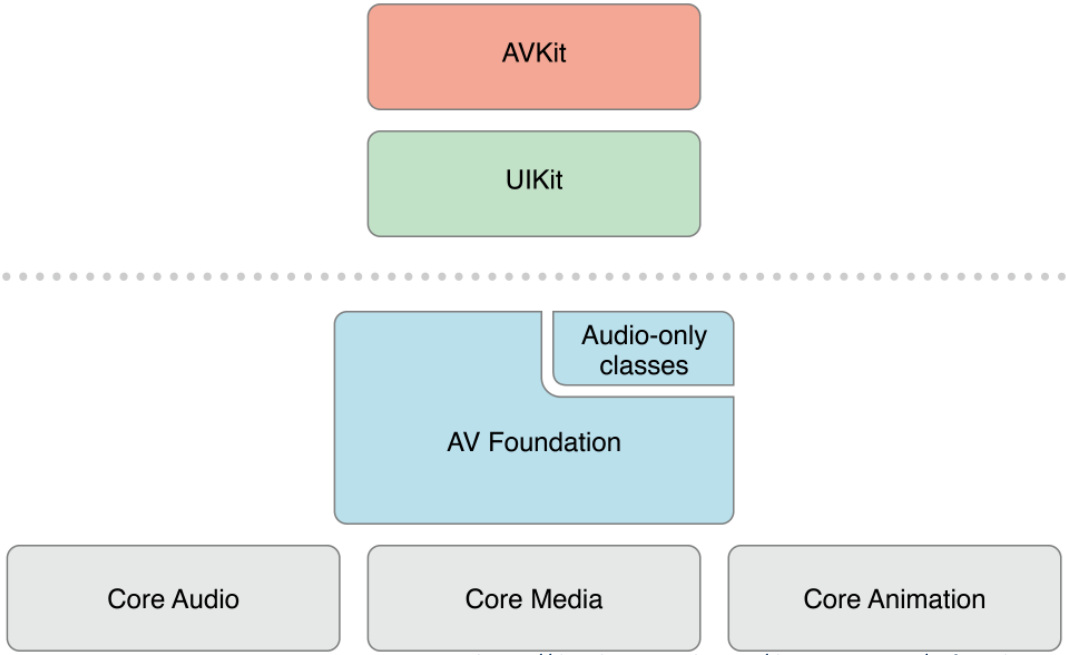
*You can use both the built-in and custom image processing filters in your app to implement special effects and perform other types of image manipulations.*

# iOS – Core Image

- The Core Image framework provides:
  - Crop, composite, blur, and sharpen images
  - Correct colour (including white-point adjustments)
  - Apply colour effects (e.g. sepia tone)
  - Generate colour, checkerboard patterns, Gaussian gradients, other pattern images
  - Warp the geometry of an image - applying an affine transform or a displacement effect
  - Add transition effects to images or video
  - Provide real-time control (e.g. colour adjustment and support for sports mode, vivid mode, and other video modes)
  - Apply linear lighting effects (e.g. spotlights)
  - Face detection including mouth and eye bounds

# iOS – AVFoundation

- AVFoundation is one of several frameworks that can be used to play and create time-based audiovisual media
- The AV Foundation framework (AVFoundation.framework) provides services for capturing, playing, inspecting, editing, and reencoding time-based audiovisual media




Source: https://developer.apple.com/documentation/avfoundation

# iOS – AVFoundation

- The services offered by this AVFoundation framework include:
  - Movie or audio capture
  - Movie or audio playback, including precise synchronization and audio panning
  - Media editing and track management
  - Media asset and metadata management
  - Audio file inspection (e.g. data format, sample rate, and number of channels)
- For most audio recording and playback requirements, can be used the following classes:
  - `AVAudioPlayer` - to play sound files,
  - `AVAudioRecorder` – to record audio.

# iOS – AVFoundation

- An audio player (`AVAudioPlayer`) is part of the AVFoundation framework and is the easiest way to play sounds with any degree of sophistication
  - Acceptable sound types include: MP3, AAC, ALAC, AIFF and WAV - starting in iOS 11 FLAC and Opus
  - Advantages: setting a sound's volume and stereo pan features, loop a sound, change the playing rate, set playback to begin somewhere in the middle of a sound
  - An audio player should always be used in conjunction with an audio session

*A single audio player can possess and play only one sound; but you can have multiple audio players, they can play separately or simultaneously, and you can even synchronize them.*

# iOS – AVFoundation

- Example (MyAudioPlayer)

```
    var myAudioPlayer : AVAudioPlayer?
- - -
    let soundPath = Bundle.main.path(forResource: "MySong",
                                                    ofType: "mp3")
    let fileData = URL.init(fileURLWithPath: soundPath!)
    do{
      try myAudioPlayer = AVAudioPlayer(contentsOf: fileData)
      myAudioPlayer?.prepareToPlay() //load the buffers + initialize hardware
    } catch{
       print("error playing sound file")
    }
- - - -
    myAudioPlayer?.play()

   /* if let player = myAudioPlayer{. player.play()    } */
```

# iOS – AVFoundation

- Video playback can be performed using classes such as `AVPlayer` provided by the AVFoundation framework
  - It supports a standard movie format like .mov or .mp4, as well as a sound files
- An `AVPlayer` is not a view!
  - An `AVPlayer`'s content is made visible through a `AVPlayerLayer` (subclass of `CALayer`) which can be added to the app's interface
- An AVFoundation video playback interface can be wrapped in a simple view controller
  - `AVPlayerViewController` (provided by the `AVKit` framework) automatically hosts an associated `AVPlayerLayer` in its own main view, providing standard playback transport controls - start and stop play or seek to a different frame
- If the app being sent into the background - `AVPlayer` will pause and resume when the app returns to the foreground

*A mobile device does not have unlimited power for decoding and presenting video in real time. A video that plays on your computer might not play at all on an iOS device.*

# iOS – AVFoundation

- Example: MyAVPlayer

```
let av = AVPlayerViewController()
let url = Bundle.main.url(forResource: "myVideo", withExtension:
"mov")!
let player = AVPlayer (url: url)
av.player = player
self.addChild(av)
self.view.addSubview(av.view)
av.view.frame = self.view.frame
//player.play()

- - -
let av = segue.destination as! AVPlayerViewController
let videoPath = Bundle.main.path(forResource: "myVideo", ofType: "mov")
let fileData = URL.init(fileURLWithPath: videoPath!)
let player = AVPlayer (url: fileData)
av.player = player
//av.player?.play()
self.present(av, animated: true)
```

# iOS – Supported media format

- Some of the media formats supported by MacOS (OS X)

| Format/Codec | File Type |
|---|---|
| Image formats | PICT, BMP, GIF, JPEG, TIFF, PNG, DIB, ICO, EPS, PDF |
| Audio file and data formats | AAC, AIFF, WAVE, uLaw, AC3, MPEG-3, MPEG-4 (.mp4, .m4a), .snd, .au, .caf, Adaptive multi-rate (.amr) |
| Video file formats | AVI, AVR, DV, M-JPEG, MPEG-1, MPEG-2, MPEG-4, AAC, OpenDML, 3GPP, 3GPP2, AMC, H.264, iTunes (.m4v), QuickTime (.mov, .qt) |
| Web streaming protocols | HTTP, RTP, RTSP |

Source: https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/OSX_Technology_Overview/MediaLayer/MediaLayer.html

*Apple lists compatible media formats for each iOS device under the "Tech Specs" page (on apple website)*

# iOS – Camera

- Photos framework (Photo Kit) provides support for manipulating photo albums and editing user's photos

- The `UIImagePickerController` is a class that:
  - can provide an interface in which the user can choose an item from the photo library
  - be used to give the user an interface similar to the Camera app, letting the user take photos and videos on devices with the necessary hardware
- introduced in iOS 14 - the `PHPickerViewController` class
  - can give the user an interface for exploring the photo library and choosing a photo (need to import PhotosUI)

- To use the photo library or camera, first need to be checked if it is available using `UIImagePickerController.isSourceTypeAvailable(_:)` method with:
  - .photoLibrary - parameter for the photo library
  - .camera - paremeter for camera.

# iOS – Camera

- Access to the camera and photo library requires user authorization
  - There is need for explicit authorization of the user for camera and photo library use
  - `AVCaptureDevice.authorizationStatus(forMediaType: .video)` – returns current authorization status for camera use.
  - `PHPhotoLibrary.authorizationStatus()` – returns information about app's authorization for accessing the user's Photos library

# iOS – Camera

- App's `Info.plist` file must provide a value for the *Photo Library Usage Description*, *Photo Library Additions Usage* and *Camera Usage Description* key
  - The first time any objects for a media type that requires permission is created, the system automatically displays an alert to request recording permission
  - Alternatively, can be called other methods to prompt the user at a time of your choosing
    - `AVCaptureDevice.requestAccess(…)` method should be used for requesting the user's permission for recording a specified media type
    - `PHPhotoLibrary.requestAuthorization(…)` method should be used for requesting the user's permission to access the photo library

*If your app will let the user record actual video (as opposed to stills), you will also need to obtain permission from the user to access the microphone.*

# iOS – Camera

- Camera can be controlled and images can be captured directly by using the `AVFoundation` framework
- This approach allows more detailed control
  - For example, you can control focus and exposure directly and independently, and for video, you can determine the quality, size, and frame rate of the resulting movie
- With this approach `AVCaptureSession` object should be used

# Resources

- Android Studio 3.5 Essentials - Java Edition, Neil Smyth, Payload Media Inc., 2019.
- Android Cookbook, Ian Darwin, O'Reilly Press, 2017.
- https://developer.android.com/training/animation/index.html
- https://developer.android.com/guide/topics/resources/animation-resource.html#View
- https://developer.android.com/develop/ui/views/animations/transitions/start-activity
- https://developer.android.com/guide/topics/media/mediaplayer
- https://developer.android.com/training/camera/choose-camera-library

# Resources

- Programming iOS 14 - Dive Deep into Views, View Controllers, and Frameworks, Matt Neuburg, O'Reilly Media Inc., 2021
- iOS 12 App Development Essentials, Neil Smyth, Payload Media, Inc., 2018.
- https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CoreAnimation_guide/Introduction/Introduction.html
- https://developer.apple.com/library/content/documentation/MacOSX/Conceptual/OSX_Technology_Overview/MediaLayer/MediaLayer.html
- https://developer.apple.com/library/archive/documentation/AudioVideo/Conceptual/AVFoundationPG/Articles/00_Introduction.html