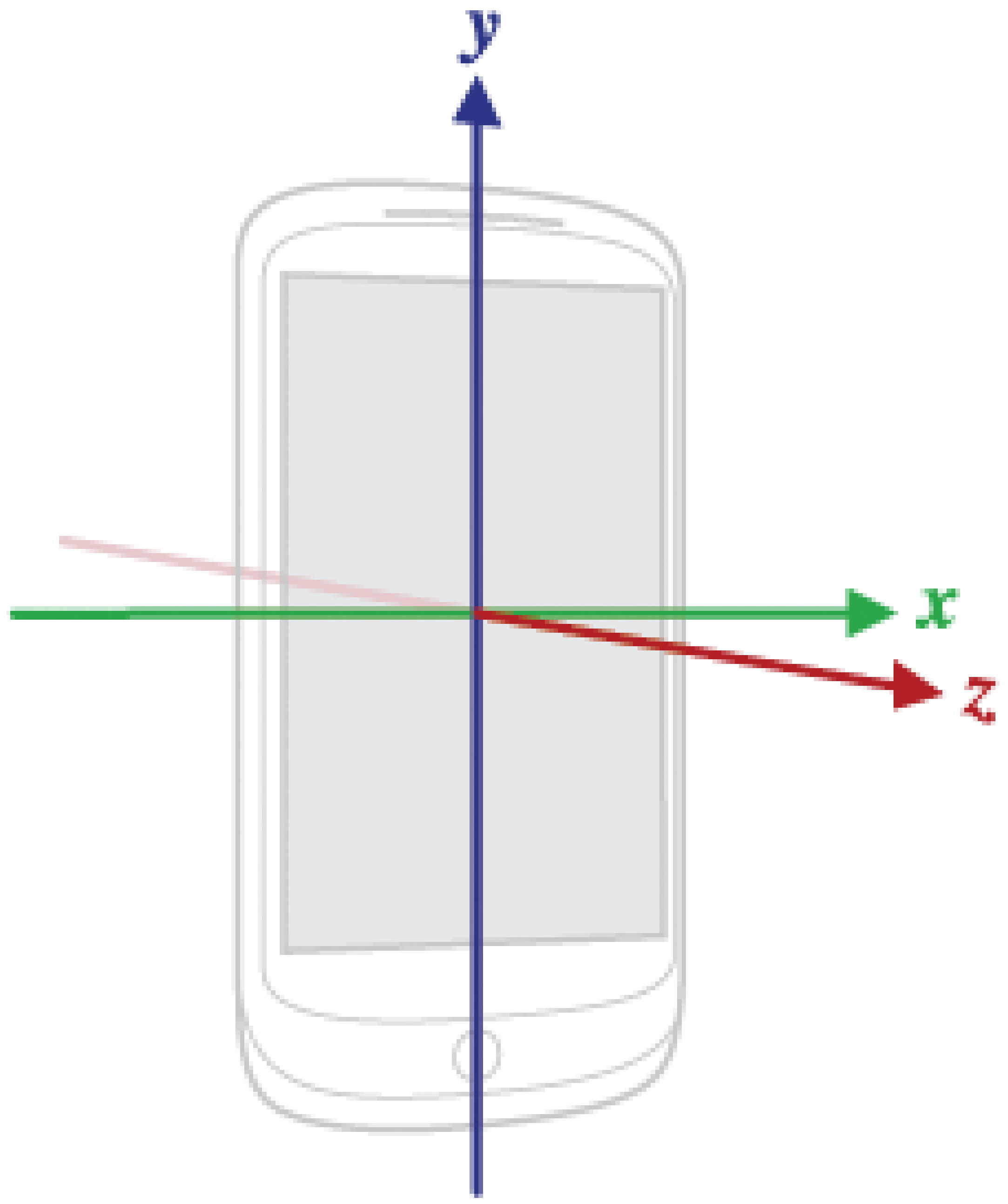
Mobile Application Development

Mobile Platforms and Application Development fundamentals

Lecture Plan

- Introduction to App Development
- Mobile Platforms and Application Development fundamentals
- Mobile Interface Design Concepts and UI/UX Design
- Introduction to Android Operating System
- Main Components of Android Application
- Data Handling in Android Applications
- Sensors and Media Handling in Android Applications
- Android Application Testing and security aspects



Sensors

- Most Android-powered devices have built-in sensors.
- These sensors measure motion, orientation, and various environmental conditions.
- The sensors can provide raw data with high precision and accuracy.
- The sensors are useful for monitoring three-dimensional device movement or positioning, or changes in the ambient environment near a device.

Sensor Categories

Motion sensors

- These sensors measure acceleration forces and rotational forces along three axes. This category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.

Environmental sensors

- These sensors measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. This category includes barometers, photometers, and thermometers.

Position sensors

- These sensors measure the physical position of a device. This category includes orientation sensors and magnetometers.

Introduction to sensors

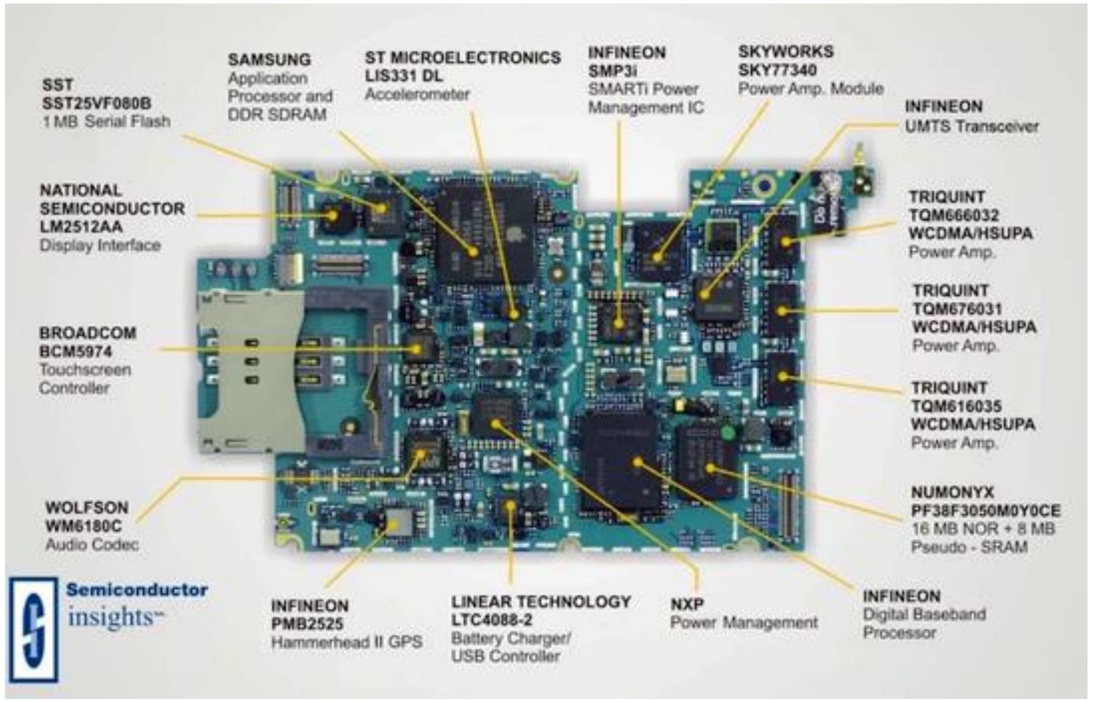
- Android sensor framework provides access to various sensors
- Two types of sensors: hardware-based and software-based
- Hardware-based sensors are physical components in a device
- They measure environmental properties directly
- Software-based sensors mimic hardware-based sensors
- They get their data from hardware-based sensors
- Software-based sensors are also known as virtual or synthetic sensors
- Examples of software-based sensors are linear acceleration and gravity sensors.

Sensors supported by Android Versions

Android Version	Supported Sensor Types
Android 1.5 (Cupcake)	Accelerometer, Magnetic field
Android 1.6 (Donut)	Accelerometer, Magnetic field, Orientation, Temperature
Android 2.1 (Eclair)	Accelerometer, Magnetic field, Orientation, Temperature, Proximity, Light
Android 2.2 (Froyo)	Accelerometer, Magnetic field, Orientation, Temperature, Proximity, Light, Pressure
Android 2.3 (Gingerbread)	Accelerometer, Magnetic field, Orientation, Temperature, Proximity, Light, Pressure
Android 4.0 (Ice Cream Sandwich)	Accelerometer, Magnetic field, Orientation, Gyroscope, Light, Proximity, Pressure
Android 4.1 - 4.3 (Jelly Bean)	Accelerometer, Magnetic field, Orientation, Gyroscope, Light, Proximity, Pressure, Humidity, Temperature
Android 4.4 (KitKat)	Accelerometer, Magnetic field, Orientation, Gyroscope, Light, Proximity, Pressure, Humidity, Temperature, Step counter, Step detector
Android 5.0 (Lollipop)	Accelerometer, Magnetic field, Orientation, Gyroscope, Light, Proximity, Pressure, Humidity, Temperature, Step counter, Step detector
Android 6.0 (Marshmallow)	Accelerometer, Magnetic field, Orientation, Gyroscope, Light, Proximity, Pressure, Humidity, Temperature, Step counter, Step detector
Android 7.0 – 12.0	Accelerometer, Magnetic field, Orientation, Gyroscope, Light, Proximity, Pressure, Humidity, Temperature, Step counter, Step detector, Heart rate

Sensor framework

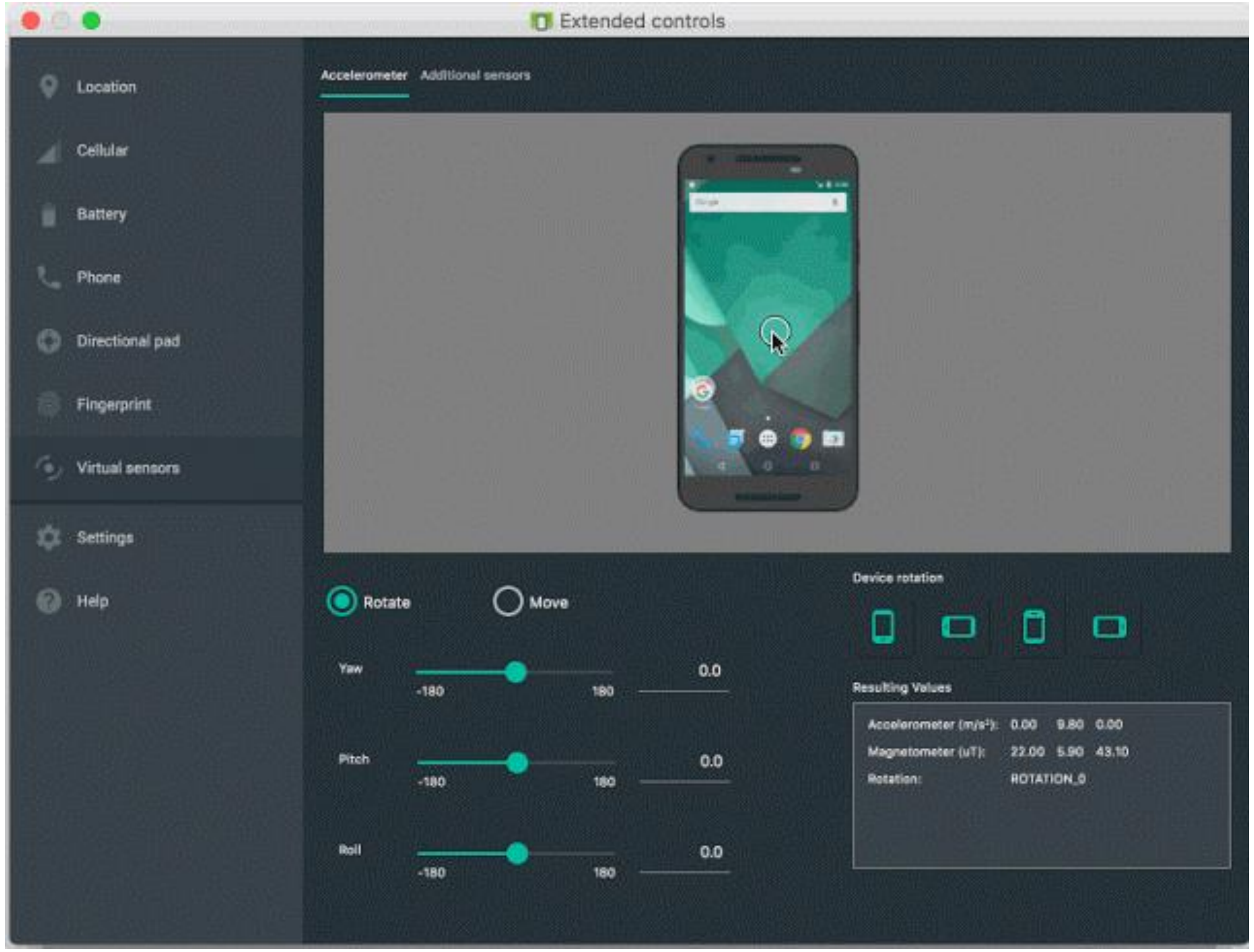
- You can access these sensors and acquire raw sensor data by using the Android sensor framework.
- The framework is part of the android.hardware package and includes classes like SensorManager, Sensor, SensorEvent, and SensorEventListener.
- SensorManager is used to create an instance of the sensor service and provides methods to access, list, and calibrate sensors.
- Sensor is used to create an instance of a specific sensor and provides methods to determine its capabilities.
- SensorEvent is used to create a sensor event object that includes information like raw sensor data, sensor type, accuracy, and timestamp.
- SensorEventListener is used to create callback methods that receive notifications when sensor values or accuracy change.
- The framework can be used to identify sensors and their capabilities at runtime and monitor sensor events to acquire raw sensor data.
- Identifying sensors and their capabilities is useful for features that rely on specific sensors or types, while monitoring sensor events is how raw sensor data is acquired.


SST SST25VF080B 1 MB Serial Flash
SAMSUNG Application Processor and DDR SDRAM
ST MICROELECTRONICS LIS331 DL Accelerometer
INFINEON SMP3i SMARTi Power Management IC
SKYWORKS SKY77340 Power Amp. Module
INFINEON UMTS Transceiver
NATIONAL SEMICONDUCTOR LM2512AA Display Interface
TRIQUINT TQM666032 WCDMA/HSUPA Power Amp.
BROADCOM BCM5974 Touchscreen Controller
TRIQUINT TQM676031 WCDMA/HSUPA Power Amp.
TRIQUINT TQM616035 WCDMA/HSUPA Power Amp.
WOLFSON WM6180C Audio Codec
NUMONYX PF38F3050M0Y0CE 16 MB NOR + 8 MB Pseudo - SRAM
Semiconductor insights
INFINEON PMB2525 Hammerhead II GPS
LINEAR TECHNOLOGY LTC4088-2 Battery Charger/ USB Controller
NXP Power Management
INFINEON Digital Baseband Processor

Best Practices when using Sensors

- Use the appropriate sensor for the job
- Minimize sensor usage
- Handle sensor events efficiently
- Consider sensor accuracy
- Test on multiple devices
- Handle sensor errors gracefully
- Respect user privacy

Test with the Android Emulator



[✓ Rotate] Move

Yaw 0.0 (-180 to 180)

Pitch 0.0 (-180 to 180)

Roll 0.0 (-180 to 180)

Resulting Values

```
Accelerometer (m/s²):  0.00   9.80   0.00
Magnetometer (uT):     22.00  5.90   43.10
Rotation:              ROTATION_0
```

Media Handling

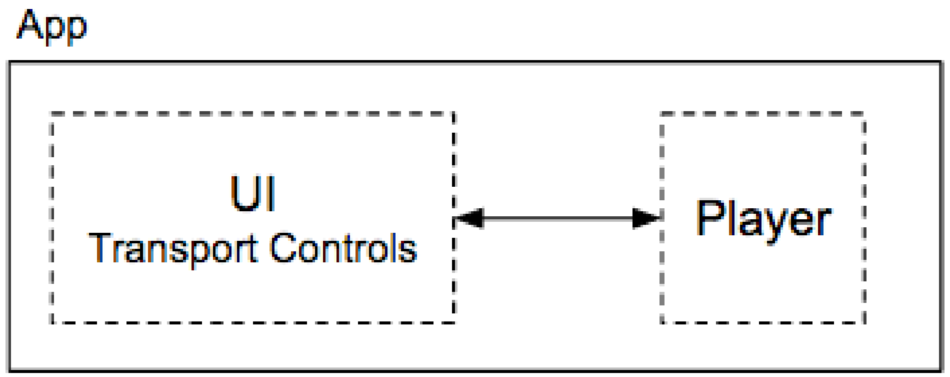
- Media, such as images, videos, and audio, is an integral part of many Android apps, from social media and entertainment apps to news and productivity apps.
- Effective media handling can significantly enhance the user experience of an app, making it more engaging, informative, and enjoyable to use.
- Poorly optimized media handling can lead to slow loading times, stuttering or freezing during playback, and excessive battery drain, all of which can negatively impact user experience.
- Media handling in Android can be complex, with multiple file formats and codecs to support, various APIs to choose from, and performance optimization techniques to implement, making it an area where developers need to have expertise to create high-quality apps.
- Ensuring proper media handling is also important for app security, as media can potentially contain malicious code or vulnerabilities that can be exploited by attackers.

Media Types

- Android can handle wide range of media types of images, videos, and audio.
- For Images:
  - JPEG: widely used for photographs and other images with many colors
  - PNG: used for images with transparency or where lossless compression is required
  - GIF: used for simple animations or small file size images with limited colors
  - WebP: developed by Google for web use, provides both lossy and lossless compression
- For Videos:
  - MP4: commonly used for video on the web and mobile devices, supports a variety of codecs
  - AVI: widely used for storing video on PCs, supports multiple codecs
  - MOV: developed by Apple for QuickTime, widely used for video on Macs and iPhones
  - MKV: open-source format, supports multiple audio and subtitle tracks, often used for high-definition video
- For Audio:
  - MP3: widely used for music and other audio files, provides good quality with reasonable file size
  - AAC: developed as a successor to MP3, provides better quality at lower bitrates
  - WAV: uncompressed format used for high-quality audio, often used for recording and editing
  - Ogg Vorbis: open-source format, provides good quality at lower bitrates, often used for streaming and gaming
- Refer the following link for more details
  - Supported media formats | Android Developers

Media app Architecture

- A multimedia application that plays audio or video usually has two parts
- A player that takes digital media in and renders it as video and/or audio
- A UI with transport controls to run the player and optionally display the player's state
- In Android you can build your own player from the ground up, or you can choose from these options:
  - The MediaPlayer class provides the basic functionality for a bare-bones player that supports the most common audio/video formats and data sources.
  - **ExoPlayer** is an open source library that's built on top of lower-level media framework components like MediaCodec and AudioTrack. ExoPlayer supports high-performance features like DASH which are not available in MediaPlayer. You can customize the ExoPlayer code, making it easy to add new components. ExoPlayer can only be used with Android version 4.1 and higher.

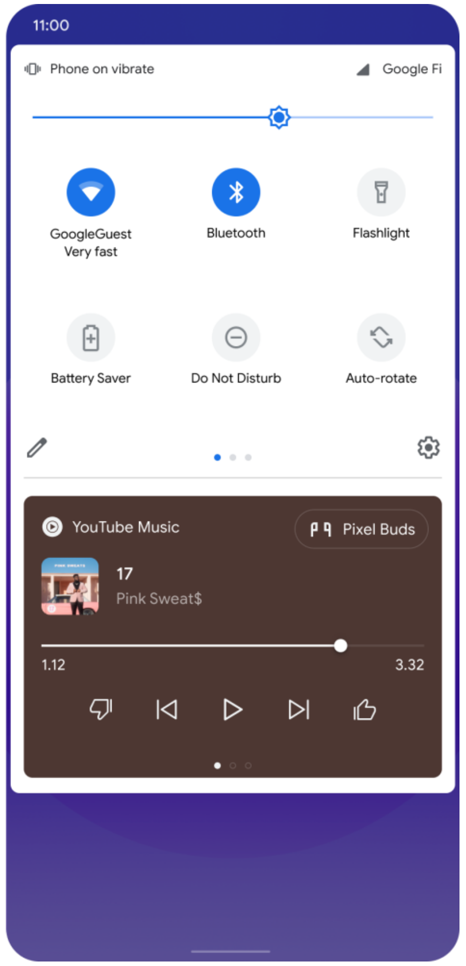


Media APIs

Media Player

Exo Player

Media Store

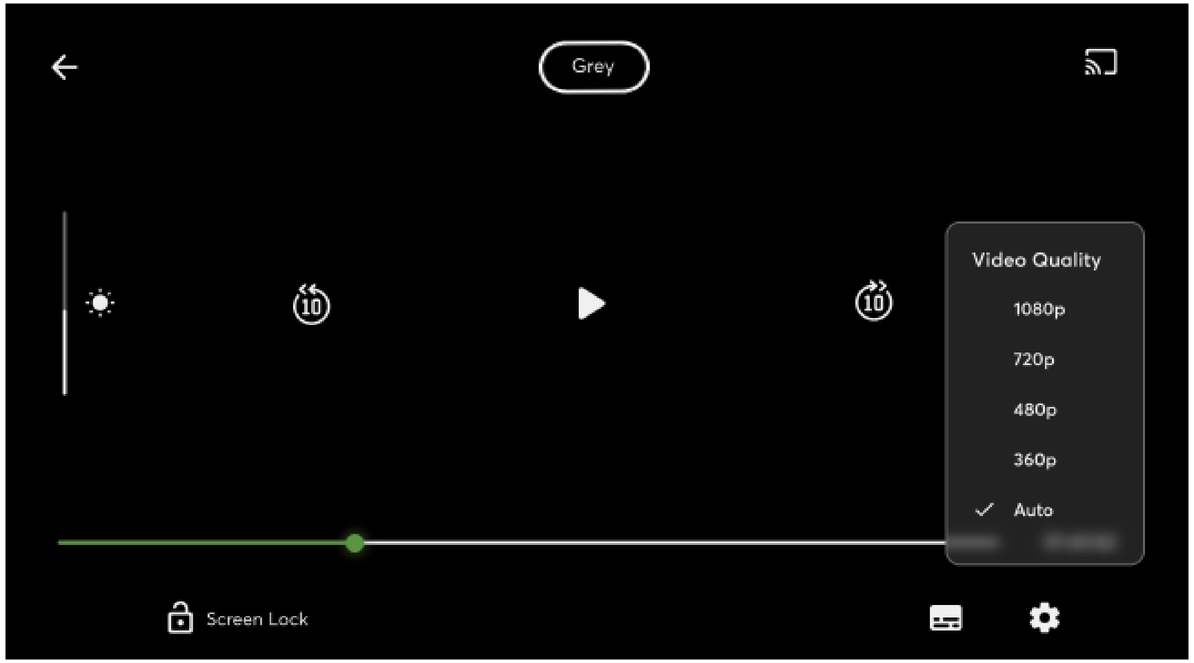


Media Player API

- Provides a basic framework for playing audio and video in an Android app.
- Supports common media formats, including MP4, 3GP, and MP3.
- Provides methods for controlling playback, such as play, pause, stop, and seek.
- Supports playback from different sources, such as local files, remote URLs, and streams.
- Can be used to play media in the foreground or the background.

ExoPlayer

- A more advanced media player that provides additional features and flexibility compared to MediaPlayer.
- Supports a wider range of media formats and codecs, including adaptive streaming formats such as DASH and HLS.
- Provides more control over buffering, network connectivity, and quality levels.
- Supports features such as seamless switching between audio and video tracks, trick play (fast-forward and rewind), and DRM.
- Offers better performance and battery efficiency compared to MediaPlayer



MediaStore

- A content provider that provides access to the media files on the device, including images, videos, and audio.
- Provides methods for querying and retrieving media metadata, such as title, artist, and duration.
- Allows apps to create, update, and delete media files in the device's media library.
- Can be used to display and play media files from the device's media library in an app.

Working With Images

- Using the ImageView widget
    - ImageView is a UI widget in Android that is used to display images in an app's user interface.
    - To display an image in an ImageView, you need to set the image resource or image URL using the setImageResource() or setImageURI() method, respectively.
    - You can also set the image programmatically using a Bitmap object with the setImageBitmap() method.
    - To resize or scale the image to fit the ImageView, you can use the setScaleType() method with one of the available scale types, such as FIT_CENTER or CENTER_CROP.
- Loading images from the network
- Loading images from the device's storage:

Working With Images

- Using the ImageView widget
- Loading images from the network
  - To load images from the network, you can use a networking library such as Retrofit or Volley to fetch the image from a remote URL.
  - Once you have the image data, you can create a Bitmap object from it using the BitmapFactory class, and then set the Bitmap in an ImageView using the setImageBitmap() method.
- Loading images from the device's storage:

Working With Images

- Using the ImageView widget
- Loading images from the network
- Loading images from the device's storage:
  - To load images from the device's storage, you can use the MediaStore API to query the device's media library for the image file.
  - Once you have the image file URI, you can use the ContentResolver and BitmapFactory classes to create a Bitmap object from the file, and then set the Bitmap in an ImageView using the setImageBitmap() method.

Optimizing image loading and handling large images in an Android app

- Caching:
  - Caching involves storing frequently used images in memory or on disk to avoid the overhead of reloading the image every time it's needed.
  - There are several caching libraries available for Android, such as Picasso, Glide, and Fresco, that can handle caching and other optimizations automatically.
- Down sampling:
  - Down sampling involves reducing the resolution of the image to a smaller size to reduce memory usage and improve performance.
  - You can use the BitmapFactory options to downsample images when loading them, for example by setting the inSampleSize option to a value that reduces the image size by a factor of 2, 4, or 8.
- Compressing:
  - Compressing involves reducing the size of the image file itself to reduce the amount of data that needs to be loaded and displayed.
  - You can use image compression tools or libraries, such as WebP or PNGQuant, to reduce the file size of the image without compromising quality.
- Handling large images:
  - Large images can cause performance issues, especially on older or less powerful devices.
  - One technique for handling large images is to use tiling or slicing to break the image into smaller pieces and load only the visible portion of the image at any given time.
  - Another technique is to load a lower resolution version of the image initially and then load the full resolution version when the user zooms in or requests it.

Working with Videos

Handling different video formats and resolutions:

- Android supports a wide range of video formats and resolutions, including MP4, 3GP, and WebM.
- To play different video formats, you can use the MediaPlayer or ExoPlayer APIs, which support a range of codecs and formats.
- To handle different video resolutions, you can use the SurfaceView or TextureView classes to display the video, which can handle different aspect ratios and resolutions.

Working with Videos

Implementing seeking and playback controls:

- To implement seeking and playback controls, you can use the MediaController class, which provides a set of standard playback controls, such as play, pause, seek, and volume control.
- You can attach a MediaController to a MediaPlayer or ExoPlayer instance using the setMediaController() method.
- Alternatively, you can create custom playback controls using UI widgets, such as buttons or seek bars, and handle the playback actions in code.

Working with Videos

Video playback in the background:

- To play video in the background, you can use the MediaPlayer or ExoPlayer APIs in conjunction with a Service or a Foreground Service.
- When using a Service, you can create a new thread or handler to manage the playback and send updates to the UI as needed.
- When using a Foreground Service, you need to create a notification to inform the user that video playback is ongoing, and handle user interaction with the notification to control the playback.

Working with audio

- Android supports a wide range of audio formats, including MP3, AAC, Ogg Vorbis, and WAV.
- To play different audio formats, you can use the MediaPlayer or ExoPlayer APIs, which support a range of codecs and formats.
- To play audio in the background, you can use the MediaPlayer or ExoPlayer APIs in conjunction with a Service or a Foreground Service.
- When using a Service, you can create a new thread or handler to manage the playback and send updates to the UI as needed.
- When using a Foreground Service, you need to create a notification to inform the user that audio playback is ongoing, and handle user interaction with the notification to control the playback.

Thank You!