Module 5: Advanced Features & Device Integration

Mobile Application Development using Android

Subject	Mobile Application Development using Android
Module	Module 5 - Advanced Features and Device Integration
Topics Covered	Multimedia, Camera, Sensors, Location, Background Tasks, Permissions

1. Overview of Advanced Features

1.1 Module Overview

Module 5 covers advanced Android features that enable rich, interactive, and feature-complete applications. Key topics include:

- Multimedia (Audio, Video Playback)
- Camera Integration
- Device Sensors
- Location Services (GPS and Google Maps)
- Background Tasks (AsyncTask, WorkManager)
- Permissions and Security
- Advanced UI Components

1.2 Device Integration Capabilities

Android provides extensive APIs for accessing device hardware and features through a layered integration stack:

API Layer	Description
Camera API	Capture photos and videos using device camera hardware
Sensors API	Access accelerometer, gyroscope, light, and other sensors
Location API	Get GPS/network-based location data
Multimedia API	Play and record audio and video content
Background API	Schedule and run tasks in the background
Storage API	Read/write files to device storage
Hardware Abstraction Layer	Provides a unified interface to device hardware

2. Multimedia: Audio and Video Playback

2.1 Audio Playback using MediaPlayer

MediaPlayer is the primary class for audio and video playback. Key steps include initializing MediaPlayer, preparing an audio source, setting up listeners, and managing lifecycle properly.

Key MediaPlayer Methods

Method	Purpose
MediaPlayer.create(context, R.raw.file)	Create and prepare player from raw resource
mediaPlayer.start()	Begin or resume playback
mediaPlayer.pause()	Pause playback
mediaPlayer.stop()	Stop playback (requires prepare() to restart)
mediaPlayer.release()	Release resources — always call in onDestroy()
mediaPlayer.seekTo(position)	Jump to position in milliseconds
mediaPlayer.getCurrentPosition()	Get current playback position
mediaPlayer.getDuration()	Get total duration of media
mediaPlayer.isPlaying()	Check if playback is active

Core Java Implementation

```
mediaPlayer = MediaPlayer.create(this, R.raw.sample_audio);

// Set listeners
mediaPlayer.setOnPreparedListener(mp -> {
    seekBar.setMax(mediaPlayer.getDuration());
});

mediaPlayer.setOnCompletionListener(mp -> {
    tvStatus.setText("Playback completed");
});

// Play / Pause / Stop controls
private void playAudio() {
    if (mediaPlayer != null && !mediaPlayer.isPlaying()) {
        mediaPlayer.start();
    }
}

// IMPORTANT: always release in onDestroy()
@Override
protected void onDestroy() {
    super.onDestroy();
    if (mediaPlayer != null) {
        mediaPlayer.release();
```

```
            mediaPlayer = null;
        }
    }
```

2.2 Video Playback using VideoView

VideoView is a UI widget that simplifies video playback. It can be paired with MediaController for built-in transport controls.

VideoView Feature	Details
setVideoURI(uri)	Load video from a URI (resource or file)
setVideoPath(path)	Load video from a file path
setMediaController(mc)	Attach playback controls (play, pause, seek)
start() / pause()	Control playback
stopPlayback()	Stop the video
isPlaying()	Check playback state
setOnCompletionListener()	Callback when video ends
setOnErrorListener()	Callback on playback errors

VideoView Sample

```
String uriPath = "android.resource://" + getPackageName() + "/" + R.raw.sample_video;
Uri uri = Uri.parse(uriPath);
videoView.setVideoURI(uri);

MediaController mc = new MediaController(this);
mc.setAnchorView(videoView);
videoView.setMediaController(mc);

// Start playback
videoView.start();
```

2.3 Audio Focus Management

Android requires apps to request audio focus before playing audio, and to respond appropriately when focus is lost — for example, when a phone call arrives.

Focus Change Event	Recommended Action
AUDIOFOCUS_GAIN	Restore full volume and resume playback
AUDIOFOCUS_LOSS	Pause playback permanently
AUDIOFOCUS_LOSS_TRANSIENT	Pause playback temporarily
AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK	Lower volume to ~30%

NOTE: Always call abandonAudioFocus() when your app is done playing audio to release focus for other apps.

3. Camera Integration

3.1 Simple Method: Using Intent

The simplest way to capture a photo is to delegate to an existing camera app via an Intent. This avoids managing the camera lifecycle but gives less control.

```
// Launch the system camera app
Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);

// Create a file and get a FileProvider URI for it
photoUri = FileProvider.getUriForFile(this,
    getPackageName() + ".fileprovider", photoFile);
takePictureIntent.putExtra(MediaStore.EXTRA_OUTPUT, photoUri);

startActivityForResult(takePictureIntent, REQUEST_IMAGE_CAPTURE);
```

NOTE: FileProvider is required for Android 7.0+ (API 24+). Register it in AndroidManifest.xml and provide a file_paths.xml resource.

3.2 CameraX (Modern Approach)

CameraX is the recommended approach for camera integration. It is lifecycle-aware, consistent across devices, and significantly easier to use than Camera2.

Gradle Dependencies

```
def camerax_version = "1.3.0"
implementation "androidx.camera:camera-core:$camerax_version"
implementation "androidx.camera:camera-camera2:$camerax_version"
implementation "androidx.camera:camera-lifecycle:$camerax_version"
implementation "androidx.camera:camera-view:$camerax_version"
```

CameraX Key Classes

Class / Concept	Purpose
ProcessCameraProvider	Manages camera lifecycle binding
Preview	Displays camera viewfinder on screen
ImageCapture	Captures still images to file or memory buffer
CameraSelector	Selects front or back camera
PreviewView	UI widget to display the camera preview

ExecutorService	Thread pool for camera operations

Capture Photo (CameraX)

```
private void capturePhoto() {
    File photoFile = new File(getExternalFilesDir(Environment.DIRECTORY_PICTURES),
        "photo_" + System.currentTimeMillis() + ".jpg");

    ImageCapture.OutputFileOptions options =
        new ImageCapture.OutputFileOptions.Builder(photoFile).build();

    imageCapture.takePicture(options,
        ContextCompat.getMainExecutor(this),
        new ImageCapture.OnImageSavedCallback() {
            @Override
            public void onImageSaved(OutputFileResults result) {
                Toast.makeText(ctx, "Photo saved!", Toast.LENGTH_SHORT).show();
            }
            @Override
            public void onError(ImageCaptureException e) { /* handle error */ }
        });
}
```

4. Accessing Device Sensors

4.1 Sensor Overview

Android provides a rich set of hardware sensors. Apps access them through the SensorManager service.

Sensor Type	Description	Use Cases
Accelerometer	Measures acceleration forces	Motion detection, shake gesture
Gyroscope	Measures rotation rate	Orientation, gaming
Magnetometer	Measures magnetic field	Compass
Proximity	Detects nearby objects	Screen off during call
Light	Measures ambient light	Auto-brightness
Gravity	Measures gravitational force	Orientation
Linear Acceleration	Acceleration without gravity	Motion tracking

4.2 Basic Sensor Usage Pattern

Using sensors follows a consistent pattern: obtain SensorManager, get the desired sensor, register/unregister a listener, and read values in onSensorChanged().

```
// 1. Get SensorManager
```

```
SensorManager sm = (SensorManager) getSystemService(Context.SENSOR_SERVICE);

// 2. Get sensor
Sensor accel = sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

// 3. Register in onResume()
sm.registerListener(this, accel, SensorManager.SENSOR_DELAY_NORMAL);

// 4. Unregister in onPause() to save battery
sm.unregisterListener(this);

// 5. Read values
@Override
public void onSensorChanged(SensorEvent event) {
    float x = event.values[0];
    float y = event.values[1];
    float z = event.values[2];
}
```

Sensor Delay Constant	Approximate Rate
SENSOR_DELAY_FASTEST	As fast as possible (~0ms)
SENSOR_DELAY_GAME	~20ms — suitable for games
SENSOR_DELAY_UI	~60ms — for UI updates
SENSOR_DELAY_NORMAL	~200ms — general purpose

4.3 Shake Detection

Shake detection works by computing the magnitude of the acceleration vector and comparing it against a threshold. A minimum time between shakes prevents false triggers.

```
// Compute acceleration magnitude (normalized to gravity)
float gX = x / SensorManager.GRAVITY_EARTH;
float gY = y / SensorManager.GRAVITY_EARTH;
float gZ = z / SensorManager.GRAVITY_EARTH;

float magnitude = (float) Math.sqrt(gX*gX + gY*gY + gZ*gZ);

if (magnitude > SHAKE_THRESHOLD) {
    long now = System.currentTimeMillis();
    if (now - lastShakeTime > MIN_TIME_BETWEEN_SHAKES) {
        lastShakeTime = now;
        listener.onShake();
    }
}
```

4.4 Step Counter (Android 4.4+)

The step counter sensor (TYPE_STEP_COUNTER) provides a running total of steps since the last device reboot. It is hardware-accelerated and battery-efficient.

```
stepCounter = sm.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);
```

```
@Override
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == Sensor.TYPE_STEP_COUNTER) {
        int steps = (int) event.values[0];
        tvSteps.setText("Steps: " + steps);
    }
}
```

5. Location and Maps Integration

5.1 Setup

Gradle Dependencies

```
implementation 'com.google.android.gms:play-services-location:21.0.1'
implementation 'com.google.android.gms:play-services-maps:18.1.0'
```

Required Permissions (AndroidManifest.xml)

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

<!-- Google Maps API Key -->
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="YOUR_API_KEY_HERE" />
```

5.2 FusedLocationProviderClient

FusedLocationProviderClient is the recommended way to get location. It automatically selects the best provider (GPS, Wi-Fi, cellular) for accuracy and battery efficiency.

Method	Description
getLastLocation()	Quickly returns the last known cached location
requestLocationUpdates(request, callback, looper)	Receive continuous location updates
removeLocationUpdates(callback)	Stop receiving updates — call in onStop()
LocationRequest.create()	Build a location request with priority and interval
PRIORITY_HIGH_ACCURACY	Uses GPS — most accurate, highest battery use
PRIORITY_BALANCED_POWER_ACCURACY	Uses Wi-Fi/cell — good balance
PRIORITY_LOW_POWER	City-level accuracy — lowest battery use

Location Update Code

```
LocationRequest request = LocationRequest.create()
    .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY)
    .setInterval(10000)       // 10 seconds
    .setFastestInterval(5000); // 5 seconds minimum

LocationCallback callback = new LocationCallback() {
    @Override
    public void onLocationResult(LocationResult result) {
        for (Location loc : result.getLocations()) {
            double lat = loc.getLatitude();
            double lng = loc.getLongitude();
            // update UI ...
        }
    }
};

fusedClient.requestLocationUpdates(request, callback, Looper.getMainLooper());
```

5.3 Google Maps Integration

Google Maps is displayed through SupportMapFragment. Use OnMapReadyCallback to get a GoogleMap instance once the map is loaded.

Map Feature	Code / Method
Set map type	googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL)
Enable My Location layer	googleMap.setMyLocationEnabled(true)
Add marker	googleMap.addMarker(new MarkerOptions().position(latLng).title("..."))
Animate camera	googleMap.animateCamera(CameraUpdateFactory.newLatLngZoom(pos, 15))
Clear markers	googleMap.clear()
Long press to pin	googleMap.setOnMapLongClickListener(latLng -> { ... })
Reverse geocode	new Geocoder(this).getFromLocation(lat, lng, 1)
Forward geocode	new Geocoder(this).getFromLocationName("query", 1)

NOTE: Always check for ACCESS_FINE_LOCATION permission at runtime before calling setMyLocationEnabled(true) or requesting location updates.

6. Background Tasks

6.1 AsyncTask (Legacy — Deprecated in API 30+)

AsyncTask allowed background work with easy UI thread callbacks. Though deprecated, it appears in legacy codebases and exam questions.

AsyncTask Method	Thread & Purpose
onPreExecute()	UI thread — prepare UI before work starts
doInBackground()	Background thread — perform long operation
publishProgress()	Signal progress from background thread
onProgressUpdate()	UI thread — update progress bar or text
onPostExecute()	UI thread — handle result, update UI
onCancelled()	UI thread — handle task cancellation

NOTE: Use Kotlin Coroutines, Executors, or WorkManager for new code. AsyncTask was removed from AOSP in Android 11 (API 30).

6.2 Handler and Runnable

Handler/Runnable is ideal for periodic UI updates (e.g., updating a seek bar every second). It runs on the main thread by default.

```
Handler handler = new Handler(Looper.getMainLooper());
Runnable updateTask = new Runnable() {
    @Override
    public void run() {
        // Update UI ...
        handler.postDelayed(this, 1000); // repeat every 1s
    }
};

// Start in onResume()
handler.post(updateTask);

// Stop in onPause()
handler.removeCallbacks(updateTask);
```

6.3 WorkManager (Recommended)

WorkManager is the recommended solution for deferrable, guaranteed background work. It survives app restarts and device reboots, and supports constraints, chaining, and observability.

Gradle Dependency

```
implementation "androidx.work:work-runtime:2.9.0"
```

WorkManager Concept	Details
OneTimeWorkRequest	Execute a task once
PeriodicWorkRequest	Execute on a repeating schedule (min 15 min interval)
Worker.doWork()	Override this to perform background logic; return Result
Result.success()	Work completed successfully
Result.failure()	Work failed permanently
Result.retry()	Work should be retried
Constraints	Network type, battery, storage, charging requirements
enqueueUniquePeriodicWork()	Ensure only one instance of periodic work runs
getWorkInfoByIdLiveData()	Observe work state in real time with LiveData

```
// Define constraints
Constraints constraints = new Constraints.Builder()
    .setRequiredNetworkType(NetworkType.CONNECTED)
    .setRequiresBatteryNotLow(true)
    .build();

// Create and enqueue one-time request
OneTimeWorkRequest request = new OneTimeWorkRequest.Builder(UploadWorker.class)
    .setConstraints(constraints)
    .build();

WorkManager.getInstance(this).enqueue(request);

// Worker class
public class UploadWorker extends Worker {
    @NonNull @Override
    public Result doWork() {
        // perform upload ...
        return Result.success();
    }
}
```

7. Permissions and Security

7.1 Runtime Permissions

Since Android 6.0 (API 23), dangerous permissions must be requested at runtime. Apps should check, request, and handle denial gracefully.

```
// Check
if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA)
        != PackageManager.PERMISSION_GRANTED) {

    // Show rationale if needed
    if (ActivityCompat.shouldShowRequestPermissionRationale(this,
```

```
            Manifest.permission.CAMERA)) {
        // Show explanation dialog
    }

    // Request
    ActivityCompat.requestPermissions(this,
        new String[]{Manifest.permission.CAMERA}, REQUEST_CODE);
}
```

Permission	Category
ACCESS_FINE_LOCATION	Dangerous — runtime request required
ACCESS_COARSE_LOCATION	Dangerous — runtime request required
CAMERA	Dangerous — runtime request required
READ_EXTERNAL_STORAGE	Dangerous — runtime request required (pre-API 33)
WRITE_EXTERNAL_STORAGE	Dangerous — runtime request required (pre-API 29)
ACTIVITY_RECOGNITION	Dangerous — required for step counter
INTERNET	Normal — granted automatically
VIBRATE	Normal — granted automatically

7.2 Security Best Practices

Practice	Implementation
Encrypted Storage	Use EncryptedSharedPreferences (AndroidX Security Crypto)
Network Security	Use network_security_config.xml; enable certificate pinning
Disable Backup	Set android:allowBackup="false" if sensitive data is stored
Block Cleartext Traffic	Set android:usesCleartextTraffic="false"
File Provider	Never expose raw file paths — use FileProvider for sharing
API Keys	Store in local.properties, not in source code
ProGuard/R8	Obfuscate release builds to protect business logic

8. Practice Exercises

Exercise 1	Music Player App — play/pause/stop with SeekBar, audio focus, and track info
Exercise 2	Camera App — CameraX preview, capture, flash/zoom controls, save to gallery

Exercise 3	Shake Detection — accelerometer shake triggers vibration and Toast with adjustable sensitivity
Exercise 4	Location Tracker — FusedLocationProvider, map display, custom marker, Geocoder address
Exercise 5	Background Uploader — WorkManager file upload, notification progress, network constraints, retry

9. Key Concepts Summary

Feature	Key Class	Purpose
Audio Playback	MediaPlayer	Play audio files with full control
Video Playback	VideoView	Display video with built-in controller
Camera	CameraX / ImageCapture	Capture photos and videos
Sensors	SensorManager	Access all device sensors
Location	FusedLocationProviderClient	Get accurate device location
Maps	GoogleMap	Display and interact with maps
One-time Background	WorkManager / Worker	Schedule guaranteed background work
Periodic Background	PeriodicWorkRequest	Repeat background tasks on a schedule
UI Thread Updates	Handler / Runnable	Post delayed or repeated Runnables
Audio Focus	AudioManager	Coordinate audio playback with other apps

10. Interview Questions

1. How do you play audio files in Android? What class do you use?
2. What is the difference between MediaPlayer and VideoView?
3. What are the two methods for camera integration? When would you choose CameraX over Intent?
4. What are the different sensor types available in Android?
5. How do you get the device location? Which API is recommended?
6. What are the different location providers? (GPS, Network, Fused)

7. What is the difference between FusedLocationProvider and the raw GPS provider?
8. How do you implement background tasks in Android?
9. What is the difference between Service and WorkManager?
10. How do you handle runtime permissions for location and camera?

Answer Highlights

Question	Key Points
Audio playback	Use MediaPlayer; create from raw resource or file; always release() in onDestroy()
MediaPlayer vs VideoView	MediaPlayer: low-level, for audio + video; VideoView: UI widget, simpler for video with built-in controls
Camera approaches	Intent: simple, no lifecycle management; CameraX: modern, lifecycle-aware, consistent across devices
Sensor types	Accelerometer, Gyroscope, Magnetometer, Proximity, Light, Gravity, Linear Acceleration, Step Counter
Location API	FusedLocationProviderClient from Google Play Services; automatically picks best provider
Location providers	GPS (outdoor, precise), Network (indoor, fast), Fused (combines both intelligently)
FusedLocation vs GPS	Fused uses multiple sources, more battery-efficient, faster first fix, consistent API
Background tasks	WorkManager (deferrable, guaranteed); Coroutines/Executor (immediate); Handler (periodic UI)
Service vs WorkManager	Service: immediate, runs until stopped; WorkManager: deferrable, survives reboots, constraint-aware
Runtime permissions	checkSelfPermission → shouldShowRationale → requestPermissions → handle result in callback

End of Module 5 Study Material — For practical implementation, refer to Android Studio sample projects and official Android documentation.