**Section 11: Activity Lifecycle, Intents & AndroidManifest**

**1️⃣ Activity Lifecycle: Core Principles**

**Why it matters**:

* Manages resources, state transitions, and user experience
* Prevents memory leaks and ensures performance
* The **Activity Lifecycle** is the sequence of **callback methods** that Android calls as an activity transitions through different states (created, visible, running, paused, stopped, destroyed).
* Proper lifecycle handling helps avoid:
  + **Memory leaks**
  + **UI inconsistencies**
  + **Performance issues**

**Lifecycle Methods Deep Dive**:

| **Method** | **Trigger** | **Key Actions** | **Code Snippet (Kotlin/Java)** |
| --- | --- | --- | --- |
| onCreate() | First creation | Initialize UI, bind data, set layout | setContentView(R.layout.activity\_main) |
| onStart() | Activity becomes visible | Start animations/UI tasks | startCameraPreview() (if visible) |
| onResume() | Gains focus (interactive) | Start sensors, location updates, animations | locationClient.requestUpdates() |
| onPause() | Partially obscured (e.g., dialog) | Pause operations, save transient state | mediaPlayer.pause() |
| onStop() | Fully hidden | Release heavy resources (network/DB) | databaseConnection.close() |
| onRestart() | Returning from stopped state | Reinitialize resources | refreshDataFromCache() |
| onDestroy() | Final destruction | Cleanup threads, unregister receivers | handler.removeCallbacks() |

**Lifecycle Flow Visual**:

bash

✅ LAUNCH: onCreate → onStart → onResume

⏸️ HOME: onPause → onStop

↩️ RETURN: onRestart → onStart → onResume

🔄 ROTATE: onPause → onStop → onDestroy → [Recreate]

**2️⃣ Intents: App Navigation Engine**

**Explicit vs. Implicit**:

| **Type** | **Use Case** | **Code Example** |
| --- | --- | --- |
| **Explicit** | Internal screens | Intent(this, ProfileActivity::class.java) |
| **Implicit** | System-wide actions | Intent(Intent.ACTION\_SEND) |

**Key Implementations**:

kotlin

*// 1. Explicit Intent (Kotlin)*

button.setOnClickListener {

Intent(this, SecondActivity::class.java).apply {

startActivity(this)

}

}

*// 2. Implicit Intent (Java)*

public void openMap(Uri geoLocation) {

Intent intent = new Intent(Intent.ACTION\_VIEW);

intent.setData(geoLocation);

if (intent.resolveActivity(getPackageManager()) != null) {

startActivity(intent); *// Safety check!*

}

}

**Critical Best Practices**:

* Always use resolveActivity() for implicit intents
* Add data type filters in Manifest for deep links:

xml

<intent-filter>

<action android:name="android.intent.action.VIEW" />

<data android:scheme="https" android:host="yourapp.com" />

</intent-filter>

**3️⃣ AndroidManifest.xml: App Blueprint**

**Essential Structure**:

xml

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.your.app">

*<!-- PERMISSIONS -->*

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

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

<application

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name">

*<!-- MAIN ACTIVITY -->*

<activity android:name=".MainActivity"

android:exported="true"> *<!-- Android 12+ requirement -->*

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

*<!-- DEEP LINK ACTIVITY -->*

<activity android:name=".DetailActivity"

android:exported="false">

<intent-filter>

<action android:name="android.intent.action.VIEW" />

<category android:name="android.intent.category.DEFAULT" />

<data android:scheme="https" android:host="yourapp.com" />

</intent-filter>

</activity>

</application>

</manifest>

**Manifest Key Rules**:

1. Declare **all activities/services/broadcast receivers**
2. Set android:exported explicitly (security requirement for Android 12+)
3. Use <uses-permission> only for essential permissions

🧠 **Critical Best Practices**

**Lifecycle**:

* Save UI state in onSaveInstanceState(Bundle)
* Use ViewModel + LiveData for data persistence during rotations
* Never perform heavy operations in onResume()/onPause()

**Intents**:

* Validate all external intents with PackageManager checks
* Use Intent.createChooser() for sharing dialogs
* Prefer **explicit intents** for internal navigation

**Manifest**:

* Minimize permissions (users see these at install)
* Set android:requestLegacyExternalStorage="true" for file access (Android 10+)
* Declare android:configChanges only when handling rotations manually

🚀 **Part B: Advanced Essentials**

**1. State Saving/Restoration**:

kotlin

override fun onSaveInstanceState(outState: Bundle) {

super.onSaveInstanceState(outState)

outState.putString("EDIT\_TEXT\_VALUE", editText.text.toString())

}

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

val savedText = savedInstanceState?.getString("EDIT\_TEXT\_VALUE") ?: ""

}

**2. Modern Permission Handling**:

kotlin

*// Using Activity Result API (Kotlin)*

val requestPermission = registerForActivityResult(RequestPermission()) { isGranted ->

if (isGranted) { openCamera() }

else { showRationaleDialog() }

}

requestPermission.launch(Manifest.permission.CAMERA)

**3. Navigation Component**:

* Replace intents with **Safe Args** for type-safe navigation:

gradle

*// build.gradle*

implementation "androidx.navigation:navigation-fragment-ktx:2.5.0"

**4. Deep Linking**:

* Handle incoming links with Navigation Component:

xml

<deepLink app:uri="https://yourapp.com/{id}" />

**5. ViewModel/LiveData**:

kotlin

class UserViewModel : ViewModel() {

private val \_user = MutableLiveData<User>()

val user: LiveData<User> = \_user *// Immutable public view*

}

**🧠 Additional Topics to Explore**

| **Concept** | **Purpose** |
| --- | --- |
| **ViewModel + LiveData** | Lifecycle-aware UI state handling |
| **Task Affinity** | Customize how activities are organized in back stack |
| **LifecycleObserver** | Jetpack utility to decouple lifecycle-aware components |
| **Logcat** | Debug lifecycle and intent flows |
| **Lifecycle Diagrams** | Useful for visualization in interviews or revision |

**✅ Summary Cheat Sheet**

| **Topic** | **API/Component** | **Modern Practice** |
| --- | --- | --- |
| Lifecycle Mgmt | Activity, LifecycleObserver | Use Jetpack ViewModel, LiveData |
| Intent Handling | Intent | Use Intent.ACTION\_\*, resolveActivity() |
| Manifest Declaration | AndroidManifest.xml | Include only necessary permissions/components |
| Permissions | ActivityCompat, ActivityResultContracts | Handle runtime permissions |
| Navigation | startActivity() | Use Navigation Component |

⚠️ **Key Takeaways**

1. Lifecycle methods = **resource management backbone**
2. Intents = **inter-component communication system**
3. Manifest = **security/permission gatekeeper**
4. Modern Android demands:
   * Activity Result API over startActivityForResult()
   * Jetpack Navigation over manual fragment transactions
   * Runtime permission handling (never assume permissions!)

📌 **Pro Tip**: Use Android Studio's **Layout Inspector** + **Logcat** to debug lifecycle issues in real-time!