Mobile Programming

Android Native Development

Lecturer: Ali Najimi
Faculty of Computer Engineering
Sharif University of Technology
Fall 2027

Kotlin | Jetpack Compose | Services | Platform Channels



Table of Contents

1. Why Learn Android Native?
2. Kotlin for Android — Essentials
3. Android App Components
4. Activities & Fragments
5. Jetpack Compose Basics
6. Services & Background Work
7. Content Providers & Broadcast Receivers
8. Permissions Model
9. Platform Channels (Flutter w/ Native)
10. Mini Lab: Native Bridge
11. Summary & Key Takeaways

Why Learn Android Native?

Even as a Flutter developer, you need Android native knowledge.

When you need native Android:

- **Platform Channels:** Access hardware APIs not available in Flutter plugins.
- **Background Services:** Foreground services, WorkManager integration.
- **Custom Permissions:** Define and enforce app-level permissions.
- **APK/AAB internals:** Understand build output for optimization.
- **Debugging:** Read native stack traces, debug ANRs and crashes.
- **Security:** Understand Android's security model for hardening.

> **Key Insight:** Flutter runs *on top of* Android. Understanding the host platform makes you a better Flutter engineer.

Kotlin for Android — Essentials

Kotlin is the official language for Android development.

```kotlin
// Data class (like Dart's model classes)
data class User(val id: Int, val name: String, val email: String)
// Null safety (similar to Dart)
val name: String = "Ali"       // Non-null
val nickname: String? = null   // Nullable
val length = nickname?.length ?: 0  // Safe call + elvis operator
// Extension functions (like Dart extensions)
fun String.isEmail(): Boolean =
    this.contains("@") && this.contains(".")
// Coroutines (like Dart Futures/Streams)
suspend fun fetchUsers(): List<User> {
    delay(1000) // Non-blocking delay
    return listOf(User(1, "Ali", "ali@example.com"))
}
// Sealed classes (like Dart sealed classes)
sealed class Result<out T> {
    data class Success<T>(val data: T) : Result<T>()
    data class Error(val message: String) : Result<Nothing>()
}
```

Android App Components

Android apps are built from **four core components**:

1. Activity

- A single screen with a UI.
- Manages user interaction.
- Has its own lifecycle.

2. Service

- Runs in the background (no UI).
- Music playback, file download, sync.

3. Broadcast Receiver

- Listens for system-wide events.
- Battery low, boot complete, SMS received.

4. Content Provider

- Shares data between apps.
- Contacts, MediaStore, Calendar.

> All components must be declared in `AndroidManifest.xml`.

Activities & Fragments

Activity = A full screen. **Fragment** = A reusable piece of UI within an Activity.

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Handle intent extras
        val userId = intent.getIntExtra("user_id", -1)

        // Load a fragment
        supportFragmentManager.beginTransaction()
            .replace(R.id.fragment_container, DetailFragment.newInstance(userId))
            .addToBackStack(null)
            .commit()
    }
}
```

Activities & Fragments

```
class DetailFragment : Fragment() {
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        return inflater.inflate(R.layout.fragment_detail, container, false)
    }
}
```

Jetpack Compose Basics

Android's modern declarative UI toolkit — similar to Flutter!

```
@Composable
fun UserCard(user: User, onClick: (Int) -> Unit) {
    Card(
        modifier = Modifier.fillMaxWidth().padding(8.dp).clickable { onClick(user.id) },
        elevation = CardDefaults.cardElevation(defaultElevation = 4.dp)
    ) {
        Row(modifier = Modifier.padding(16.dp)) {
            Icon(Icons.Default.Person, contentDescription = "User")
            Spacer(modifier = Modifier.width(12.dp))
            Column {
                Text(text = user.name, style = MaterialTheme.typography.titleMedium)
                Text(text = user.email, style = MaterialTheme.typography.bodySmall)
            }
        }
    }
}
@Preview(showBackground = true)
@Composable
fun PreviewUserCard() {
    UserCard(User(1, "Ali Najimi", "ali@example.com"), onClick = {})
}
```

Services & Background Work

Android restricts background execution to save battery. Use the right tool.

Foreground Service

- User-visible (notification required).
- Music, navigation, file upload.
- Survives Doze mode.

WorkManager

- Guaranteed execution.
- Constraints (WiFi, charging).
- Periodic or one-time.

```
// Foreground Service
class MusicService : Service() {
    override fun onStartCommand(
        intent: Intent?, flags: Int, startId: Int
    ): Int {
        val notification = NotificationCompat
            .Builder(this, "music")
            .setContentTitle("Playing music")
            .setSmallIcon(R.drawable.ic_music)
            .build()

        startForeground(1, notification)
        // Play music...
        return START_STICKY
    }

    override fun onBind(intent: Intent?): IBinder? = null
}
```

WorkManager — Scheduled Background Tasks

Define a worker

```
class SyncWorker(appContext: Context, params: WorkerParameters) :
    CoroutineWorker(appContext, params) {

    override suspend fun doWork(): Result {
        return try {
            // Do background work
            val data = fetchRemoteData()
            saveToLocal(data)
            Result.success()
        } catch (e: Exception) {
            Result.retry() // or Result.failure()
        }
    }
}
```

WorkManager — Scheduled Background Tasks

Schedule the work

```
val syncRequest = PeriodicWorkRequestBuilder<SyncWorker>(
    15, TimeUnit.MINUTES  // Minimum interval
).setConstraints(
    Constraints.Builder()
        .setRequiredNetworkType(NetworkType.CONNECTED)
        .setRequiresBatteryNotLow(true)
        .build()
).build()

WorkManager.getInstance(context).enqueueUniquePeriodicWork(
    "sync", ExistingPeriodicWorkPolicy.KEEP, syncRequest
)
```

Permissions Model

Android permissions protect user data and hardware.

Permission Categories:

- **Normal:** Auto-granted at install (INTERNET, VIBRATE).
- **Dangerous:** User must grant at runtime (CAMERA, LOCATION, STORAGE).
- **Special:** Device admin, overlay, system alert window.

Permissions Model

```
// Request runtime permission
if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA)
    != PackageManager.PERMISSION_GRANTED
) {
    ActivityCompat.requestPermissions(
        this,
        arrayOf(Manifest.permission.CAMERA),
        REQUEST_CODE_CAMERA
    )
}
// Handle result
override fun onRequestPermissionsResult(
    requestCode: Int, permissions: Array<String>, grantResults: IntArray
) {
    if (requestCode == REQUEST_CODE_CAMERA &&
        grantResults.isNotEmpty() &&
        grantResults[0] == PackageManager.PERMISSION_GRANTED) {
        // Permission granted — open camera
    }
}
```

Platform Channels (Flutter w/ Native)

Platform Channels let Flutter communicate with native Android/iOS code.

How it works:

1. Flutter sends a **message** via a `MethodChannel`.
2. Android receives it in `MainActivity`.
3. Android executes native code and **returns a result**.

```dart
// Flutter side
const channel = MethodChannel('com.example/battery');

Future<int> getBatteryLevel() async {
  try {
    final level = await channel.invokeMethod<int>('getBatteryLevel');
    return level!;
  } on PlatformException catch (e) {
    print('Failed: ${e.message}');
    return -1;
  }
}
```

Platform Channels — Android Side

```kotlin
// MainActivity.kt
class MainActivity : FlutterActivity() {
    override fun configureFlutterEngine(flutterEngine: FlutterEngine) {
        super.configureFlutterEngine(flutterEngine)
        MethodChannel(flutterEngine.dartExecutor.binaryMessenger,
            "com.example/battery"
        ).setMethodCallHandler { call, result ->
            when (call.method) {
                "getBatteryLevel" -> {
                    val level = getBatteryLevel()
                    if (level != -1) {
                        result.success(level)
                    } else {
                        result.error("UNAVAILABLE", "Battery level not available", null)
                    }
                }
                else -> result.notImplemented()
            }
        }
    }
    private fun getBatteryLevel(): Int {
        val intent = IntentFilter(Intent.ACTION_BATTERY_CHANGED)
        val battery = registerReceiver(null, intent)
        val level = battery?.getIntExtra(BatteryManager.EXTRA_LEVEL, -1) ?: -1
        val scale = battery?.getIntExtra(BatteryManager.EXTRA_SCALE, -1) ?: -1
        return (level * 100) / scale
    }
}
```

Mini Lab

Native Bridge — Battery + Toast

Objective: Create a Flutter app that calls native Android APIs via Platform Channels.

Requirements:

1. Create a `MethodChannel` named `com.example.native`.
2. Implement `getBatteryLevel` — returns battery percentage.
3. Implement `showToast` — shows a native Android Toast.
4. Implement `getDeviceInfo` — returns device model + Android version.
5. Display results in a Flutter UI.
6. Implement Charging Information (Amperage, Wattage, Number of charges, etc.)

Summary: Key Takeaways

Android Native

- Kotlin is the official language.
- Four components: Activity, Service, Receiver, Provider.
- Jetpack Compose ≈ Flutter's declarative UI.
- WorkManager for reliable background work.

Permissions

- Normal vs Dangerous permissions.
- Runtime permission requests.
- Declare in Manifest.

Platform Channels

- Flutter w/ Native communication bridge.
- `MethodChannel` for method calls.
- `EventChannel` for streaming data.
- Essential for accessing native-only APIs.

Why It Matters

- Flutter runs on top of Android.
- Understanding native helps debugging.
- Some features require native code.

Thank You! 🙌

Questions?

"An Android developer walks into a bar.
The bartender says, 'What'll you have?'
The developer replies, 'I'll have a Kotlin, on the Main thread.'
The bartender: 'Sorry, we only serve Coroutines here.'"