S7 Inf A3 — Java for Graphical and Mobile Programming · Centrale Lyon

Stéphane Derrode

CM6

Android: Architecture & First App

- Android ecosystem — OS, SDK, emulator
- Project structure — Manifest, R class, resources
- Activity lifecycle — onCreate, onPause, onResume…
- First layout in XML — TextView, Button
- Handling a click — android:onClick & Toast

Module Overview

Part 1 — Java Fundamentals	Part 2 — GUI with Swing	Part 3 — Android
CM1 From Python to Java	**CM4** Swing: Components & Layouts	**CM6** Android Architecture
CM2 Collections, Exceptions & I/O	**CM5** Graphics 2D & Events	**CM7** Layouts, Navigation & Intents
CM3 Inheritance, Interfaces		**CM8** Persistence & Threads

Projects: ★ Spider Game (Part 2) ★ Calculator / Currency Converter (Part 3)

Exam 50% + BEs 50%

Stéphane Derrode — Centrale Lyon · S7 Inf A3

CM6 — Agenda

01 Android ecosystem & constraints

02 App architecture — components

03 Project structure in Android Studio

04 Activity & lifecycle

05 First XML layout — widgets

06 Handling clicks — Toast

Android — Ecosystem & Constraints

Key facts

Open source	Apache licence — free to use and modify
Linux kernel	All Android devices run a Linux kernel underneath
Java/Kotlin SDK	Apps written in Java or Kotlin; compiled to DEX bytecode
ART runtime	Android Runtime executes DEX bytecode (replaces Dalvik since API 21)
API levels	Each Android version = one API level. API 33 = Android 13 (2022)
Market share	~72% of smartphones worldwide (2024)

Constraints vs desktop

Touch interface — no mouse, no hover

Battery & memory limited — avoid expensive loops

Connectivity not guaranteed — handle offline

Screen diversity — many sizes and densities

App can be interrupted anytime (call, notification)

Event-driven — same model as Swing

Android is event-driven: the app displays an interface and waits for the user to act. No sequential main() — instead, the system calls your Activity methods when it needs to.

App Architecture — Components & Intents

The 4 component types

Activity
One screen with a UI — the main building block

★ used in this course

Service
Background task without UI (music player, download…)

not covered

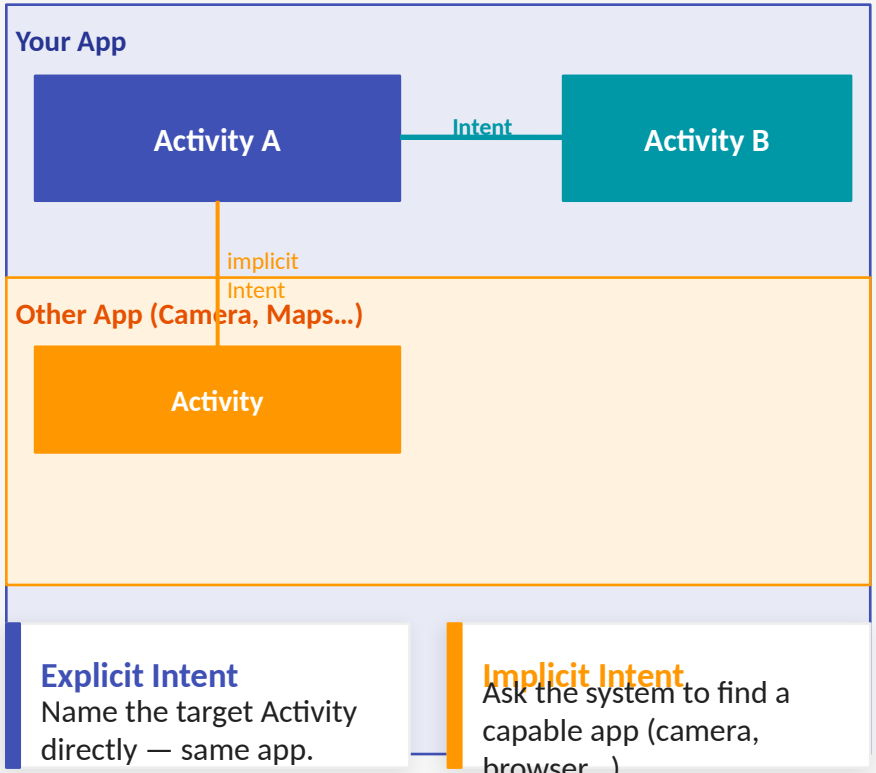
BroadcastReceiver
Listens for system events (battery low, SMS…)

not covered

ContentProvider
Shared data store for inter-app access

not covered

Components communicate via Intents



Explicit Intent
Name the target Activity directly — same app.

Implicit Intent
Ask the system to find a capable app (camera, browser…).

Project Structure in Android Studio

```
NotesPad/
├── app/
│   ├── src/main/
│   │   ├── java/com/s7infa3/notespad/
│   │   │   └── MainActivity.java      ← Activity code
│   │   ├── res/
│   │   │   ├── layout/
│   │   │   │   └── activity_main.xml  ← UI definition
│   │   │   ├── values/
│   │   │   │   ├── strings.xml        ← text
constants
│   │   │   │   └── colors.xml
│   │   │   └── drawable/              ← images &
icons
│   │   └── AndroidManifest.xml        ← app
declaration
│   └── build.gradle                   ← dependencies
└── build.gradle                       ← project
config
```

The 3 files you touch every day

AndroidManifest.xml

Declares every Activity, the app name, icon, min SDK, and permissions. Generated by Android Studio — you edit it to add new Activities.

```
<activity
  android:name=".MainActivity"
  android:exported="true">
  <intent-filter>
    <action android:name=
      "android.intent.action.MAIN"/>
  </intent-filter>
</activity>
```

strings.xml — never hardcode text

android:text="@string/add_note" — always use resource references for multi-language support.

The R Class & Resource References

R — auto-generated bridge between XML and Java

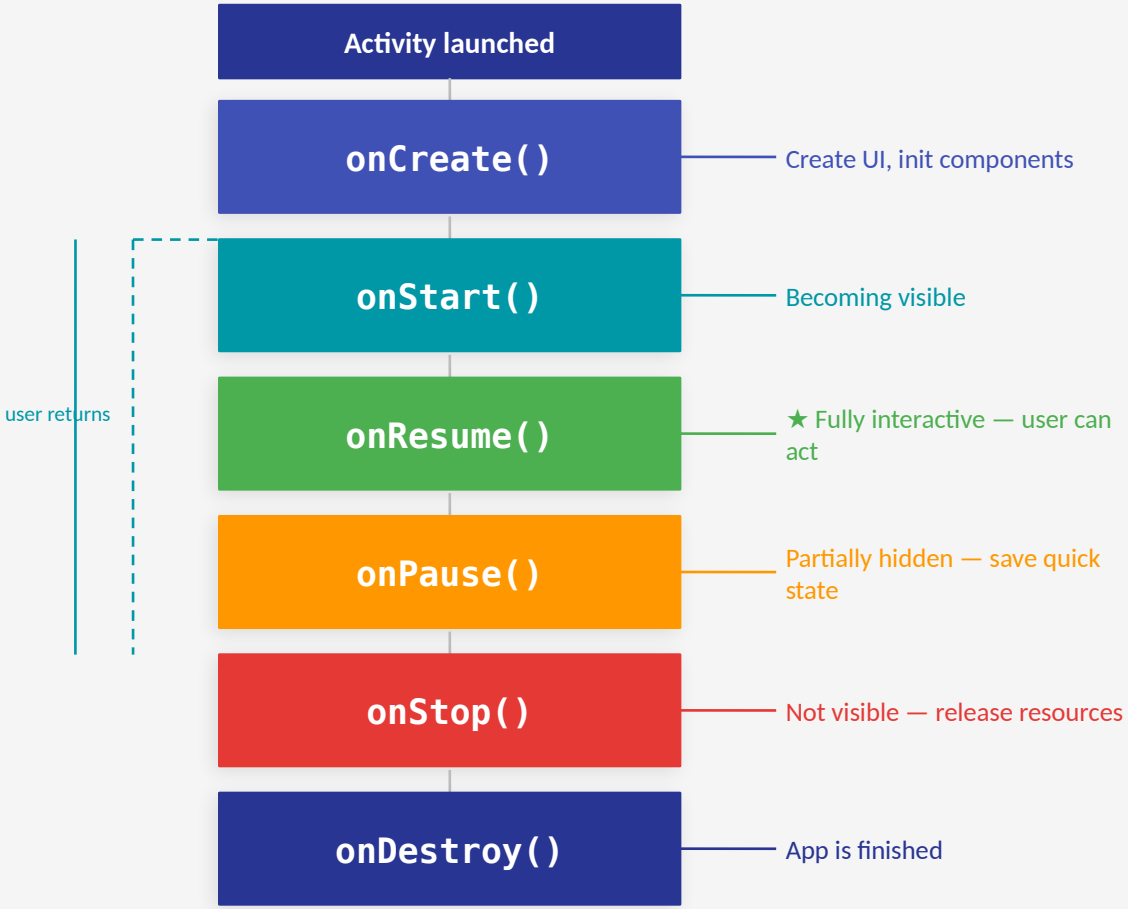
```
// In Java — access resources via R.type.name
Button btn = findViewById(R.id.addButton);
String title = getString(R.string.app_name);
int color = getColor(R.color.colorPrimary);
setContentView(R.layout.activity_main);

// Common R sub-classes:
// R.id        — widget IDs defined in XML
// R.layout    — XML layout files
// R.string    — strings from strings.xml
// R.drawable  — images from res/drawable/
// R.color     — colors from colors.xml
```

```
<!-- In XML — reference other resources with @
-->

<!-- Reference a string resource -->
<TextView
    android:text="@string/app_name" />

<!-- Reference a color -->
<Button
    android:backgroundTint="@color/indigo" />

<!-- Give a widget an ID (used by R.id) -->
<Button
    android:id="@+id/addButton"
    android:text="@string/add_note" />

<!-- @+ creates a new ID; @ references existing
```

Never edit R.java
R.java is regenerated every time you build. Any manual change is overwritten. If you see 'cannot find symbol R', it means the build failed — check your XML for errors.

findViewById<T>() in Java
Button btn = findViewById(R.id.addButton);
Cast is automatic since API 26. Always call after setContentView() in onCreate().

Activity Lifecycle



What to put where

onCreate()

setContentView() • findViewById() • init logic

onResume()

restart animations • re-register listeners

onPause()

save unsaved data • pause animations

onStop()

release heavy resources

```
@Override
protected void onCreate(
        Bundle savedInstanceState) {
```

First XML Layout — activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android=
      "http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <!-- Title -->
    <TextView
        android:id="@+id/titleView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/app_name"
        android:textSize="24sp"
        android:textStyle="bold"
        android:layout_marginBottom="16dp"/>

    <!-- Input field -->
    <EditText
        android:id="@+id/noteInput"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/hint_note"
        android:inputType="text"/>

    <!-- Add button -->
```

Android dimension units

`dp`	**Density-independent pixels** — All sizes and margins — scales with screen density
`sp`	**Scale-independent pixels** — Text sizes — also scales with user font preference
`px`	**Raw pixels** — AVOID — not density-independent
`wrap_content`	**Shrinks to content** — Width/height: use when you want the view just big enough
`match_parent`	**Fills parent** — Width/height: expand to fill available space

Common Widgets — Android vs Swing

Android widget	XML tag	Key attributes	Swing equiv.
Text display	`<TextView>`	text textSize textStyle textColor	`JLabel`
Text input	`<EditText>`	hint inputType maxLines	`JTextField`
Button	`<Button>`	text onClick enabled drawableLeft	`JButton`
Image	`<ImageView>`	src scaleType contentDescription	`JLabel(icon)`
Checkbox	`<CheckBox>`	checked text onCheckedChanged	`JCheckBox`
Dropdown	`<Spinner>`	entries prompt onItemSelected	`JComboBox`
Scrollable list	`<RecyclerView>`	layoutManager adapter	`JList+scroll`

inputType — always set it on EditText

android:inputType="number" → numeric keypad "textPassword" → hidden "text" → standard keyboard "phone" → phone pad. Omitting it shows a generic keyboard — always specify.

Handling Clicks — android:onClick vs Listener

Option 1 — android:onClick in XML (simple, quick)

```
<!-- In activity_main.xml -->
<Button
    android:id="@+id/addButton"
    android:text="@string/btn_add"
    android:onClick="onAddClicked" />
```

```
// In MainActivity.java
// Must be public void, takes View param
public void onAddClicked(View view) {
    EditText input = findViewById(
        R.id.noteInput);
    String note = input.getText()
                       .toString();
    System.out.println("Adding: " + note);
}
```

android:onClick — limitation

The method must be in the Activity class. Doesn't work with fragments or if the button is in a different context. Use setOnClickListener() for anything non-trivial.

Option 2 — setOnClickListener in Java (recommended)

```
// In MainActivity.java  onCreate()
Button addBtn = findViewById(R.id.addButton);
EditText input = findViewById(R.id.noteInput);

// Lambda — same as Swing ActionListener
addBtn.setOnClickListener(v -> {
    String note = input.getText()
                       .toString().trim();
    if (!note.isEmpty()) {
        addNote(note);
        input.setText("");  // clear field
    }
});

// Long click
addBtn.setOnLongClickListener(v -> {
    showHelp();
    return true; // consumed
});
```

Same lambda pattern as Swing

btn.setOnClickListener(v -> doSomething()) is exactly like btn.addActionListener(e -> doSomething()). Same concept, slightly different syntax.

Toast, Logcat & Debugging

```
// ── Toast – brief popup message ──────────────────
// Context, text, duration
Toast.makeText(this,
    "Note added!",
    Toast.LENGTH_SHORT).show();

// From a string resource
Toast.makeText(this,
    R.string.note_added,
    Toast.LENGTH_LONG).show();

// ── Logcat – Android's console ───────────────────
import android.util.Log;

private static final String TAG = "NotesPad";

// In any method:
Log.d(TAG, "Adding note: " + note); // Debug
Log.i(TAG, "App started");          // Info
Log.w(TAG, "Note is empty!");       // Warning
Log.e(TAG, "Error: " + e.getMessage()); // Error

// System.out.println also appears in Logcat
System.out.println("debug value: " + x);
```

Toast — use cases

Brief feedback: 'Note saved', 'Invalid input', 'Deleted'. Toast disappears automatically — SHORT = 2s, LONG = 3.5s. Always call .show() — forgetting it is the #1 Toast bug.

Logcat in Android Studio

View → Tool Windows → Logcat. Filter by tag (e.g. 'NotesPad') to see only your messages. Log.d() is filtered out in release builds — use it freely during development.

Log levels

d = Debug (verbose, dev only) i = Info w = Warning e = Error (always visible)
Use TAG = class name to identify the source.

Rotation crash → check Logcat

If the app crashes on screen rotation, Logcat shows the stack trace immediately. Always read the red lines from top to bottom.

NotesPad v1 — Putting It Together

```
public class MainActivity extends AppCompatActivity {

    private List<String> notes = new ArrayList<>();

    @Override
    protected void onCreate(Bundle saved) {
        super.onCreate(saved);
        setContentView(R.layout.activity_main);

        EditText input  = findViewById(R.id.noteInput);
        Button   addBtn = findViewById(R.id.addButton);
        TextView status = findViewById(R.id.statusView);

        addBtn.setOnClickListener(v -> {
            String note = input.getText()
                               .toString().trim();
            if (note.isEmpty()) {
                Toast.makeText(this,
                    R.string.error_empty,
                    Toast.LENGTH_SHORT).show();
                return;
            }
            notes.add(note);
            status.setText(
                notes.size() + " note(s)");
            input.setText("");  // clear field
            Log.d("NotesPad",
                "Added: " + note);
```

NotesPad v1 — what we have

- ✓ LinearLayout with TextView + EditText + Button
- ✓ setContentView() in onCreate()
- ✓ setOnClickListener lambda
- ✓ getText().toString().trim()
- ✓ Toast for user feedback
- ✓ Log.d() for debugging
- ○ Display the list of notes — CM7 (RecyclerView)
- ○ Navigate to edit screen — CM7 (Intent)
- ○ Save notes across sessions — CM8 (SharedPreferences)

CM6 — Key Takeaways

Activity	One screen = one Activity; extends AppCompatActivity; no main()
Lifecycle	onCreate() → onStart() → onResume() ↔ onPause() → onStop() → onDestroy()
setContentView	Load the XML layout in onCreate() — before any findViewById()
R class	Auto-generated; R.id.name in Java = @+id/name in XML — never edit it
Listener	setOnClickListener(v -> ...) — same lambda pattern as Swing
Toast & Log	Toast for user messages; Log.d(TAG, ...) for debugging in Logcat

Up next → TD6: Android Studio setup + NotesPad v1