

UNIT 4

Android Activity

(4 Hours) — Mobile Programming (CACS351)

4.1 The Activity Life Cycle

An **Activity** represents a single screen with a user interface. Android manages activities on a **back stack**; as the user navigates, an activity moves between states, and the system calls **callback methods** so the developer can react. Understanding the life cycle prevents crashes, memory leaks and lost data.

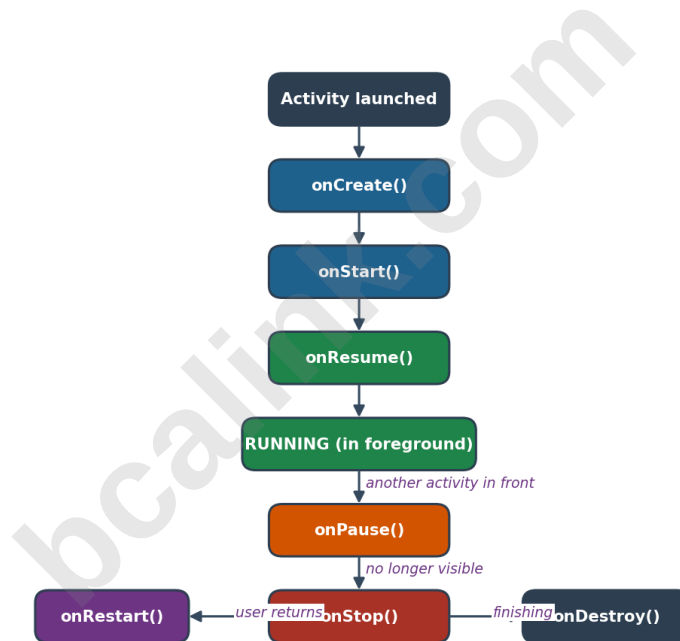


Figure 4.1: Simplified activity life cycle

Callback	Called when	Typical use
<code>onCreate()</code>	Activity first created	<code>setContentView()</code> , initialise views, restore <code>savedInstanceState</code>
<code>onStart()</code>	Activity becomes visible	Prepare UI, register visible-only resources
<code>onResume()</code>	Activity in foreground, user can interact	Start animations, camera, sensors
<code>onPause()</code>	Another activity comes in front (partly visible)	Pause animations/video; save light-weight state; must be quick
<code>onStop()</code>	Activity no longer visible	Release heavy resources, save data

Callback	Called when	Typical use
onRestart()	Stopped activity is coming back	Re-initialise things released in onStop()
onDestroy()	Activity finishing or destroyed by the system	Final clean-up

Three Nested Loops

- **Entire lifetime:** onCreate() → onDestroy().
- **Visible lifetime:** onStart() → onStop().
- **Foreground lifetime:** onResume() → onPause().

```
public class MainActivity extends AppCompatActivity {
    private static final String TAG = "MainActivity";

    @Override protected void onCreate(Bundle b) {
        super.onCreate(b);
        setContentView(R.layout.activity_main);
        Log.d(TAG, "onCreate");
    }
    @Override protected void onStart() { super.onStart(); Log.d(TAG, "onStart"); }
    @Override protected void onResume() { super.onResume(); Log.d(TAG, "onResume"); }
    @Override protected void onPause() { super.onPause(); Log.d(TAG, "onPause"); }
    @Override protected void onStop() { super.onStop(); Log.d(TAG, "onStop"); }
    @Override protected void onDestroy() { super.onDestroy(); Log.d(TAG, "onDestroy"); }
}
```

Listing 4.1: Logging life-cycle callbacks (always call the super method)

Example 1 — Rotation

The user rotates the device while typing in an EditText. Which callbacks run, and how do we keep the data?

Solution

By default, rotation is a configuration change: the activity is destroyed and recreated: onPause → onStop → onDestroy → onCreate → onStart → onResume.

Save data in *onSaveInstanceState(Bundle)* and restore it in *onCreate()*; or use a ViewModel, which survives rotation.

```
private static final String KEY_INDEX = "index";

@Override protected void onSaveInstanceState(Bundle out) {
    super.onSaveInstanceState(out);
    out.putInt(KEY_INDEX, mCurrentIndex);
}
// in onCreate():
if (savedInstanceState != null)
    mCurrentIndex = savedInstanceState.getInt(KEY_INDEX, 0);
```

Listing 4.2: Saving and restoring state

4.2 Creating Multiple Activities

Real apps have several screens. To add one: **File → New → Activity → Empty Views Activity**. Android Studio creates a Java class, an XML layout and the manifest entry. Example: *MainActivity* (enter name) and *SecondActivity* (show greeting).

4.3 Declaring Activities in the Manifest

Every activity must be declared in **AndroidManifest.xml** using an `<activity>` element; otherwise starting it throws *ActivityNotFoundException*. The launcher activity has an intent filter with **MAIN** and **LAUNCHER**.

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

    <activity android:name=".SecondActivity"
        android:parentActivityName=".MainActivity" />
</application>
```

Listing 4.3: Manifest declarations

4.4 Connecting Activities with Intents

An **Intent** is a messaging object used to request an action from another component. There are two types:

Type	Description	Example
Explicit intent	Names the exact target class	<code>new Intent(this, SecondActivity.class)</code>
Implicit intent	Describes an action; the system finds a matching app	<code>ACTION_VIEW</code> with a URL, <code>ACTION_DIAL</code> , <code>ACTION_SEND</code>

```
// Explicit: start SecondActivity
Intent i = new Intent(MainActivity.this, SecondActivity.class);
startActivity(i);

// Implicit: open a web page
Intent web = new Intent(Intent.ACTION_VIEW, Uri.parse("https://bcalink.com"));
startActivity(web);

// Implicit: dial a number
startActivity(new Intent(Intent.ACTION_DIAL, Uri.parse("tel:9800000000")));
```

Listing 4.4: Explicit and implicit intents

4.5 Passing Data Between Activities

Data is attached to an intent as **extras** (key–value pairs). The receiving activity reads them with *getIntent()*. Use constants for keys. Custom objects must implement *Parcelable* or *Serializable*.

```
// Sender (MainActivity)
Intent i = new Intent(this, SecondActivity.class);
i.putExtra("name", etName.getText().toString());
i.putExtra("age", 21);
startActivity(i);

// Receiver (SecondActivity.onCreate)
String name = getIntent().getStringExtra("name");
int age = getIntent().getIntExtra("age", 0); // 0 = default value
tvGreeting.setText("Hello " + name + " (" + age + ")");
```

Listing 4.5: *putExtra()* and *getStringExtra()*

4.6 Getting a Result Back from a Child Activity

Sometimes the first activity needs an answer from the second (e.g., pick a contact, cheat-check in GeoQuiz). The classic approach is *startActivityForResult()* with *onActivityResult()*. It is **deprecated** in current Android; the recommended replacement is the **Activity Result API** (*registerForActivityResult()*).

```
// Parent: register launcher (Activity Result API)
ActivityResultLauncher<Intent> launcher = registerForActivityResult(
    new ActivityResultContracts.StartActivityForResult(),
    result -> {
        if (result.getResultCode() == RESULT_OK && result.getData() != null) {
            boolean shown = result.getData().getBooleanExtra("answer_shown", false);
            tvStatus.setText(shown ? "You cheated" : "Fair play");
        }
    });
launcher.launch(new Intent(this, CheatActivity.class));

// Child: send the result back
Intent data = new Intent();
data.putExtra("answer_shown", true);
setResult(RESULT_OK, data);
finish();
```

Listing 4.6: Getting a result back

Note: The older pair `startActivityForResult(intent, REQUEST_CODE)` / `onActivityResult(requestCode, resultCode, data)` still appears in exams and legacy code; know both.

4.7 Getting and Setting Data To/From the Layout File

- **Get** text: `editText.getText().toString()`; checked state: `checkBox.isChecked()`; spinner: `spinner.getSelectedItem()`.
- **Set** text: `textView.setText("...")`; image: `imageView.setImageResource(R.drawable.pic)`; visibility: `view.setVisibility(View.GONE)`.
- Convert strings to numbers with `Integer.parseInt()` inside try/catch for `NumberFormatException`.
- **View Binding** (enable `viewBinding true` in Gradle) generates a binding class, avoiding `findViewById()` and null errors.

```
// View Binding
private ActivityMainBinding binding;
@Override protected void onCreate(Bundle b) {
    super.onCreate(b);
    binding = ActivityMainBinding.inflate(getLayoutInflater());
    setContentView(binding.getRoot());
    binding.btnAdd.setOnClickListener(v -> {
        int a = Integer.parseInt(binding.etA.getText().toString());
        int c = Integer.parseInt(binding.etB.getText().toString());
        binding.tvResult.setText(String.valueOf(a + c));
    });
}
```

Listing 4.7: Reading and writing layout data with View Binding

Key Takeaways

- ✓ Life cycle: `onCreate` → `onStart` → `onResume` → (running) → `onPause` → `onStop` → `onDestroy`; `onRestart` when returning.
- ✓ Save UI state with `onSaveInstanceState()` or a `ViewModel` to survive rotation.
- ✓ Each activity must be declared in `AndroidManifest.xml`; the launcher has the `MAIN/LAUNCHER` intent filter.
- ✓ Explicit intents name a class; implicit intents request an action. Data travels in intent extras.
- ✓ Use the Activity Result API (or `startActivityForResult` in older code) to receive results from a child activity.

— End of Unit 4 —