ENCS5150 · EXP 7 COMPANION

Activities and Fragments

Lifecycles, how Fragments share data, why they exist, and what modern Android uses instead.

Five questions

01 What happens to an Activity, from launch to destroy?

02 How is a Fragment’s lifecycle different?

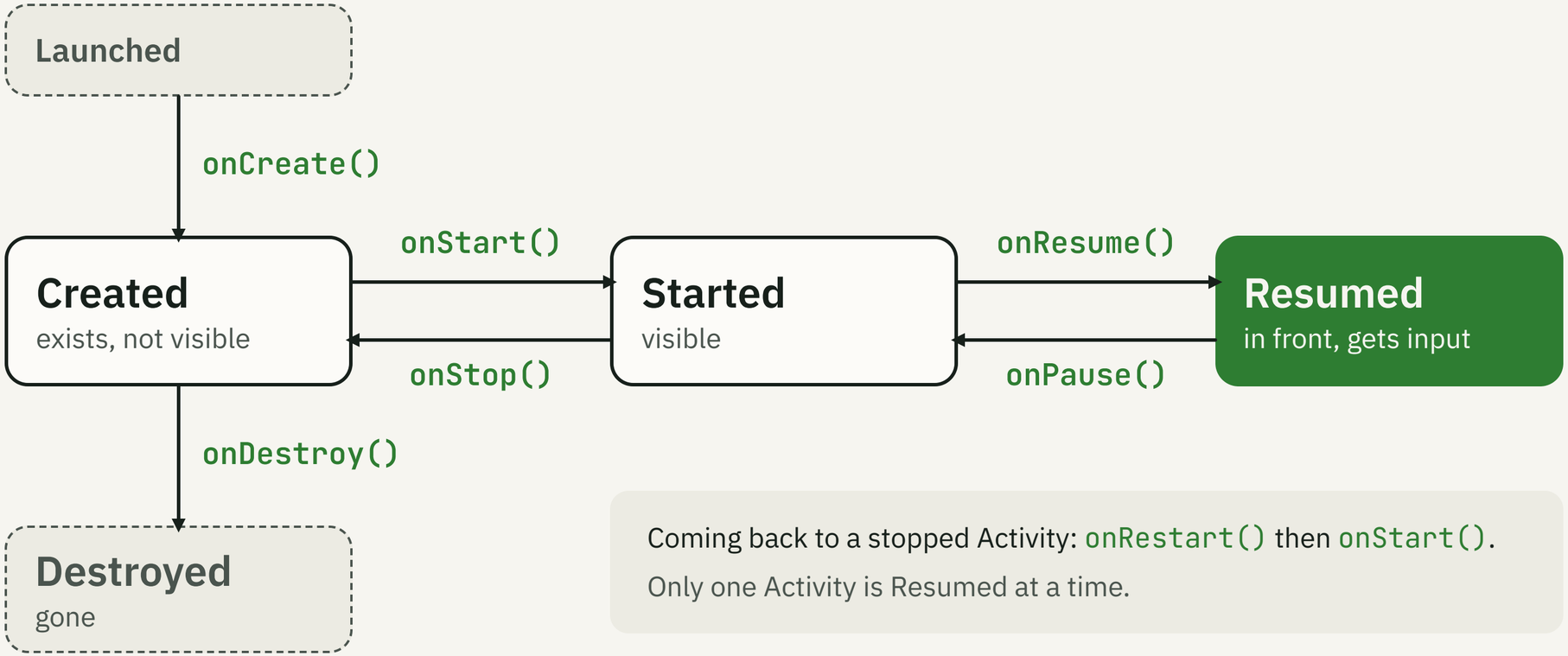
03 How should Fragments talk to each other?

04 Why use Fragments at all, and when?

05 Is there something better?

01 · ACTIVITY LIFECYCLE

Three states, six callbacks



Coming back to a stopped Activity: `onRestart()` then `onStart()`.

Only one Activity is Resumed at a time.

01 · ACTIVITY LIFECYCLE

What the user does, what Android calls

What happens	Callbacks
The app is launched	onCreate → onStart → onResume
Another Activity covers it	onPause → onStop
The user comes back to it	onRestart → onStart → onResume
Home button, or the screen turns off	onPause → onStop
Back button, or finish()	onPause → onStop → onDestroy
A see-through Activity on top (permission prompt)	onPause only
The screen rotates	onPause → onStop → onDestroy, then a new object: onCreate → onStart → onResume

Rule of thumb: start things in **onStart** and stop them in **onStop**. Save anything the user would hate to lose by **onStop**: after it, Android may kill the process without calling **onDestroy**.

01 · ACTIVITY LIFECYCLE

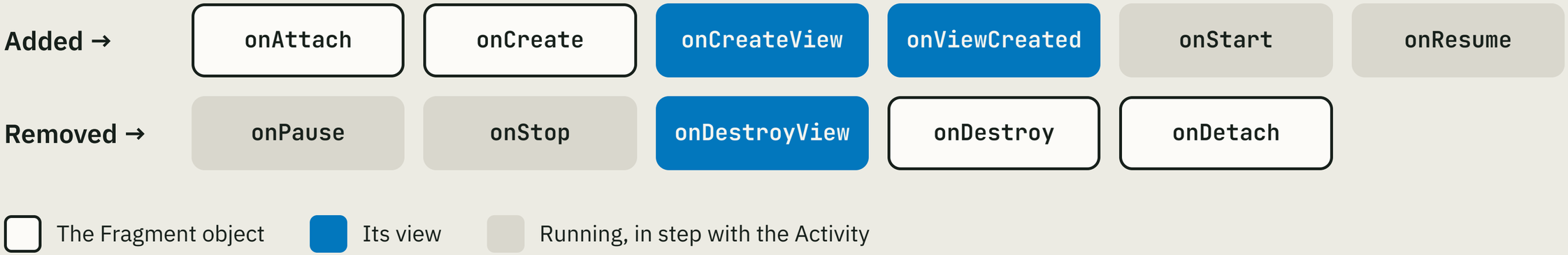
Rotating the screen destroys the Activity and builds a new one

Where the data lives	Rotation	Process killed
A field in the Activity or Fragment	Lost	Lost
Text in an EditText that has an id	Kept	Kept
The onSaveInstanceState Bundle	Kept	Kept
A ViewModel	Kept	Lost
SharedPreferences or SQLite (Exp 6, Exp 4)	Kept	Kept

In Exp 7 the click count lives in a ViewModel, so rotating keeps it. To survive the process being killed too, keep it in the ViewModel’s SavedStateHandle.

02 · FRAGMENT LIFECYCLE

Two lifecycles: the Fragment’s and its view’s



The view can die first

The view exists only from **onCreateView** to **onDestroyView**. Detach, or a Fragment on the back stack, destroys the view and keeps the object.

Never ahead of the Activity

A Fragment can’t be more alive than its host: when the Activity stops, its Fragments stop; when it is destroyed, so are they.

02 · FRAGMENT LIFECYCLE

What each transaction does, as logged in Exp 7

Operation	The Fragment object	Callbacks, in order
add	Created	onAttach, onCreate, onCreateView, onViewCreated, onStart, onResume
detach	Kept	onPause, onStop, onDestroyView
attach	The same object again	onCreateView, onViewCreated, onStart, onResume
remove	Destroyed	onPause, onStop, onDestroyView, onDestroy, onDetach
replace	Old destroyed, new created	old: onPause, onStop · new: onAttach … onStart · old: onDestroyView, onDestroy, onDetach · new: onResume
rotation	Recreated by the FragmentManager	every Fragment: the remove callbacks, then the add callbacks

Detach and remove look the same on screen. Only **remove** reaches onDestroy, so only remove loses the Fragment object.

02 · FRAGMENT LIFECYCLE

Three rules that prevent most Fragment bugs

RULE 1

Touch views only while they exist

Set up views in onViewCreated. Observe LiveData with the view’s lifecycle, not the Fragment’s:

```
getViewLifecycleOwner()
```

RULE 2

Keep a no-argument constructor

After a rotation Android recreates the Fragment by calling it. Pass data in a Bundle, never as constructor parameters:

```
setArguments(bundle)
```

RULE 3

Find Fragments, don’t keep them

A Fragment saved in an Activity field is a stale copy after a rotation. Ask the manager for the real one:

```
findFragmentById(id)
```

03 · FRAGMENT COMMUNICATION

Three ways for Fragments to share data

BEST FOR SHARED DATA

Shared ViewModel

Every Fragment gets the same ViewModel from the Activity. One changes the data, the others observe it.

Use for data several Fragments show. It survives rotation. Exp 7 uses this.

FOR ONE RESULT

Fragment Result API

One Fragment sends a Bundle under a key. Another listens for that key through the FragmentManager.

Use for handing back a single result: a dialog’s choice, a picked date, “item deleted”.

OLDER PATTERN

Interface through the Activity

The Fragment calls a method on its Activity, which finds the other Fragment and calls it.

The cost: the Activity glues the two together, and crashes if the other Fragment isn’t there. This is the manual’s way.

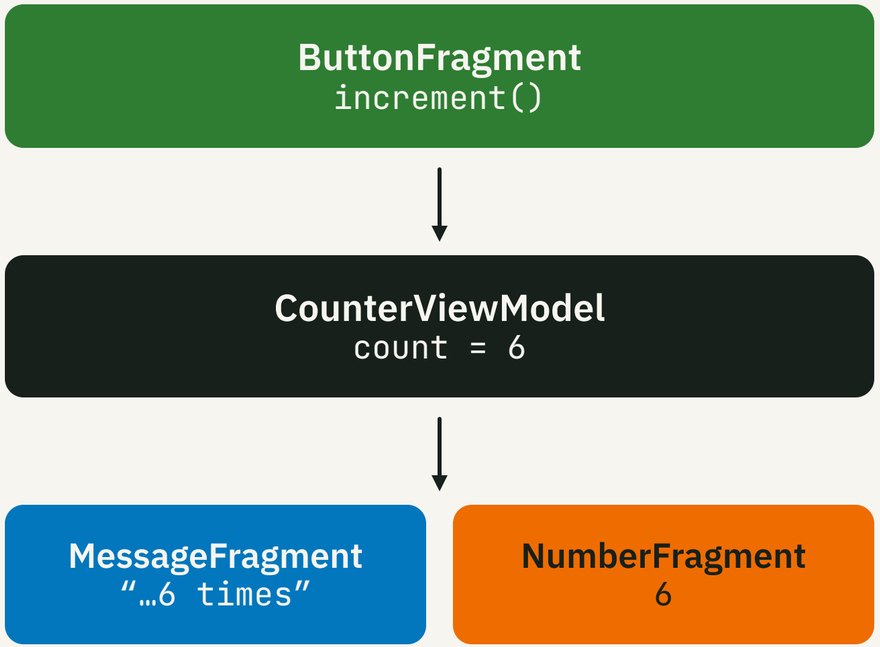
03 · FRAGMENT COMMUNICATION

Shared ViewModel: one owner for the data

```
// In any Fragment of the Activity
CounterViewModel counter =
    new ViewModelProvider(requireActivity())
        .get(CounterViewModel.class);

// ButtonFragment changes it
button.setOnClickListener(v → counter.increment());

// MessageFragment observes it
counter.getCount().observe(getViewLifecycleOwner(),
    count → text.setText("Clicked " + count + " times"));
```



requireActivity() makes every Fragment get the same object. With **this**, each Fragment would get its own ViewModel and see nothing.

Fragment Result API: hand back one answer

```
// Receiver, in onCreate
getParentFragmentManager().setFragmentResultListener(
    "pickColor", this, (key, bundle) -> {
        String color = bundle.getString("color");
        // use it
    });

// Sender, when the user picks
Bundle result = new Bundle();
result.putString("color", "blue");
getParentFragmentManager()
    .setFragmentResult("pickColor", result);
```

Matched by key

Sender and receiver only share the string `"pickColor"`.

Delivered when ready

The listener runs once the receiver is started, even if it was in the background.

Only the latest

One result per key is kept. It is not a stream of updates: for that, use a ViewModel.

What not to do

```
otherFragment.showMessage(text);
```

Calling another Fragment. The two can never be used apart, and the call crashes when the other one isn’t on screen.

```
getActivity().findViewById(R.id.text);
```

Reaching into the Activity’s views. It searches every Fragment’s layout, and returns null when the view isn’t there.

```
((MainActivity) getActivity()).update();
```

Casting to one Activity class. The Fragment now works in that Activity only, and throws a ClassCastException anywhere else.

A Fragment should know nothing about who shows it, or who listens.

04 · WHY FRAGMENTS?

What a Fragment gives you that an Activity can’t

Screens that adapt

A list and its details side by side on a tablet, one at a time on a phone, from the same two Fragments.

Many screens, one Activity

Tabs, bottom navigation and the Navigation component swap Fragments inside one Activity, with their own back stack.

Reuse with logic included

A piece of screen keeps its layout and its code together, so it can sit in several Activities.

Pieces that come and go

Add, replace and remove parts of a screen at runtime, each with its own lifecycle. DialogFragment and ViewPager2 pages are Fragments too.

04 · WHEN TO USE THEM

Which piece of UI should it be?

Use	When it is	For example
An Activity	An entry point into the app: the launcher, a deep link, a share target	The main screen, a “Share to” screen
A Fragment	A part of a screen, or one screen of several in an Activity, with its own logic and lifecycle	Tabs, list and detail, Navigation destinations, dialogs
A custom View or <include>	A reusable widget with no lifecycle of its own	A rating bar, a shared header layout
A composable	Any UI in a Jetpack Compose app	Everything, from a button to a whole screen

Not worth a Fragment: a block of layout you only want to repeat. An <include> is enough, and has no lifecycle to get wrong.

05 · IS THERE SOMETHING BETTER?

Jetpack Compose: UI as functions of state

Google’s recommended toolkit

for new Android UI, written in Kotlin instead of XML layouts.

No transactions

To swap part of a screen, change the state. The UI is redrawn to match it.

The ViewModel stays

The same CounterViewModel from Exp 7 drives this screen unchanged.

```
// The whole Exp 7 counter, in Compose (Kotlin)
@Composable
fun CounterScreen(vm: CounterViewModel = viewModel()) {
    val count by vm.getCount().observeAsState(0)
    Column {
        Button(onClick = { vm.increment() }) {
            Text("Click me")
        }
        Text("The button was clicked $count times")
    }
}
```

05 · IS THERE SOMETHING BETTER?

Fragments or Compose?

	Fragments with Views	Jetpack Compose
Language	Java or Kotlin	Kotlin only
UI written in	XML layouts plus code	Kotlin functions
Swap part of a screen	A FragmentTransaction	An if on the state
Lifecycle to learn	The Fragment’s and its view’s	A composable enters and leaves the screen
Navigation	Navigation component, Fragment destinations	Navigation Compose
Where you meet it	Most existing apps, and this course	New apps
Mixing the two	ComposeView puts Compose in a Fragment	AndroidFragment puts a Fragment in Compose

Learn Fragments: you will maintain them. Start new apps in Compose.

TAKEAWAYS

Four things to remember

01 Callbacks come in pairs: start in onStart, stop in onStop.

02 A Fragment’s view dies before the Fragment: use onViewCreated and getViewLifecycleOwner().

03 Share data through a ViewModel and results through the Fragment Result API, never Fragment to Fragment.

04 Fragments run today’s View-based apps. Compose is where new apps start.