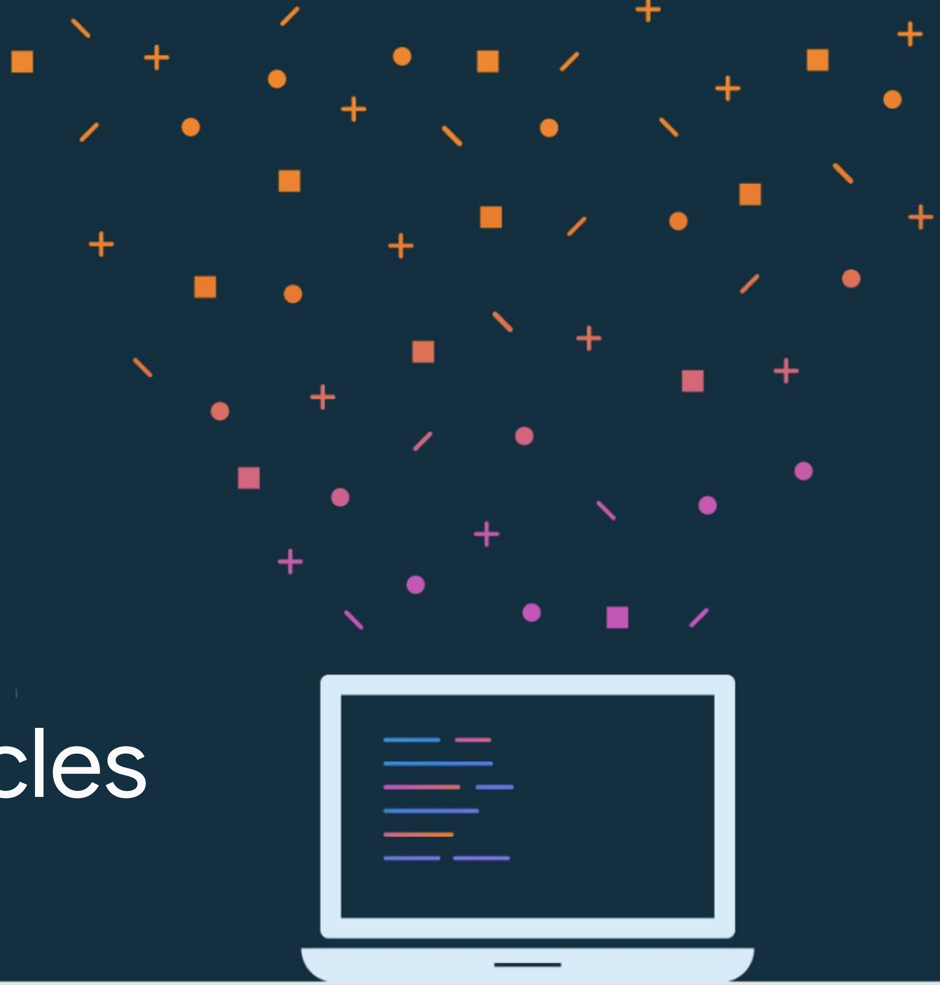
Lesson 7: Activity and fragment lifecycles



About this lesson

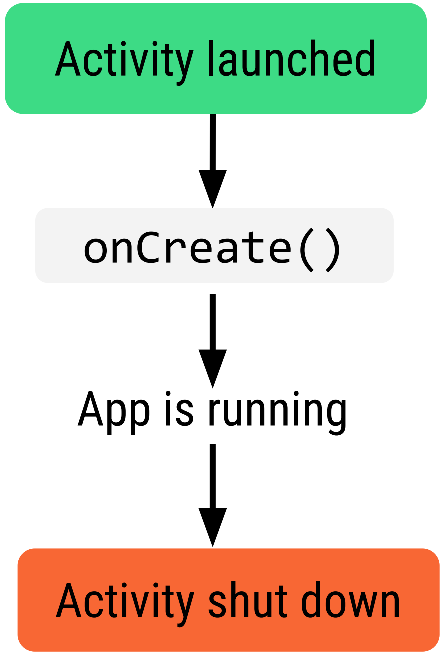
Lesson 7: Activity and fragment lifecycles

Activity lifecycle

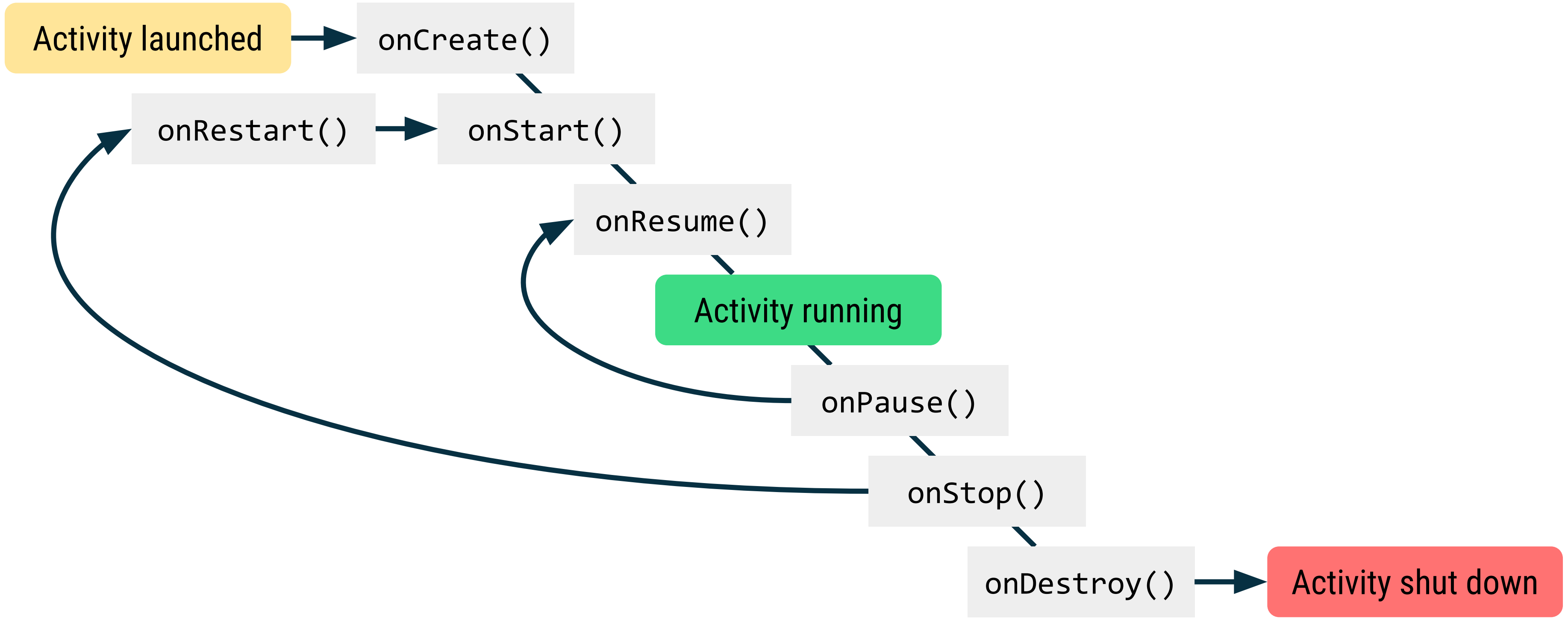
Why it matters

- Preserve user data and state if:
  - User temporarily leaves app and then returns
  - User is interrupted (for example, a phone call)
  - User rotates device
- Avoid memory leaks and app crashes.

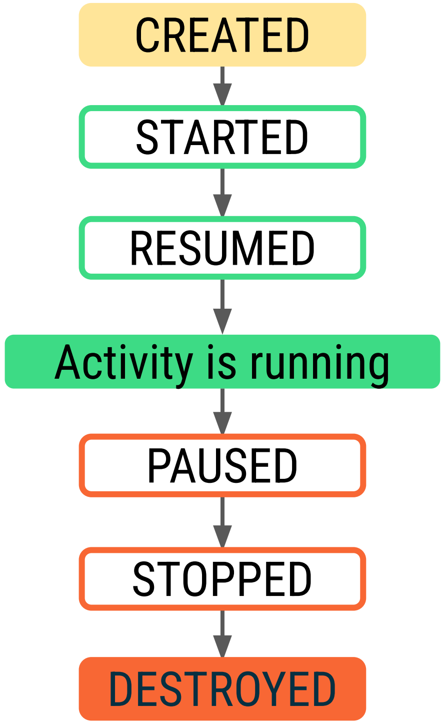
Simplified activity lifecycle



Activity lifecycle



Activity states



onCreate()

- Activity is created and other initialization work occurs
- You must implement this callback
- Inflate activity UI and perform other app startup logic

onStart()

- Activity becomes visible to the user
- Called after activity:
    - `onCreate()`

      or
    - `onRestart()` if activity was previously stopped

onResume()

- Activity gains input focus:
    - User can interact with the activity
- Activity stays in resumed state until system triggers activity to be paused

onPause()

- Activity has lost focus (not in foreground)
- Activity is still visible, but user is not actively interacting with it
- Counterpart to `onResume()`

onStop()

- Activity is no longer visible to the user
- Release resources that aren’t needed anymore
- Save any persistent state that the user is in the process of editing so they don’t lose their work

onDestroy()

- Activity is about to be destroyed, which can be caused by:
  - Activity has finished or been dismissed by the user
  - Configuration change
- Perform any final cleanup of resources.
- Don’t rely on this method to save user data (do that earlier)

Summary of activity states

State	Callbacks	Description
Created	`onCreate()`	Activity is being initialized.
Started	`onStart()`	Activity is visible to the user.
Resumed	`onResume()`	Activity has input focus.
Paused	`onPause()`	Activity does not have input focus.
Stopped	`onStop()`	Activity is no longer visible.
Destroyed	`onDestroy()`	Activity is destroyed.

Save state

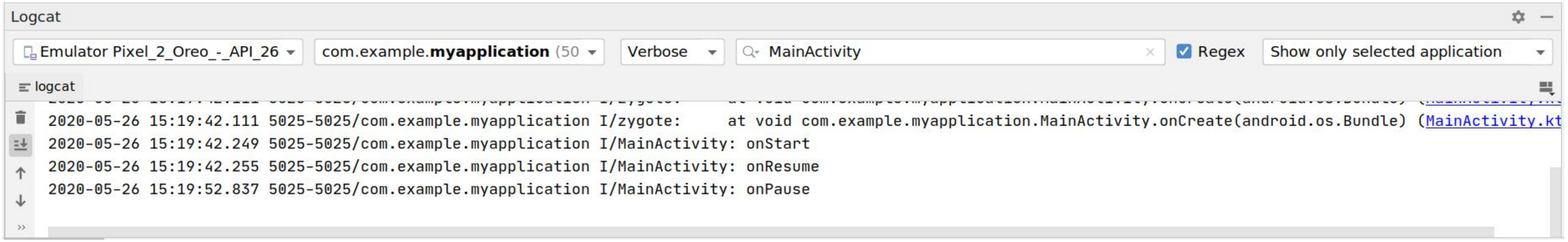
User expects UI state to stay the same after a config change or if the app is terminated when in the background.

- Activity is destroyed and restarted, or app is terminated and activity is started.
- Store user data needed to reconstruct app and activity Lifecycle changes:
  - Use `Bundle` provided by `onSaveInstanceState()`.
  - `onCreate()` receives the `Bundle` as an argument when activity is created again.

Logging

Logging in Android

- Monitor the flow of events or state of your app.
- Use the built-in `Log` class or third-party library.
- Example `Log` method call: `Log.d(TAG, "Message")`



Logcat
Emulator Pixel_2_Oreo_-_API_26
com.example.myapplication (50
Verbose
MainActivity
Regex
Show only selected application
logcat

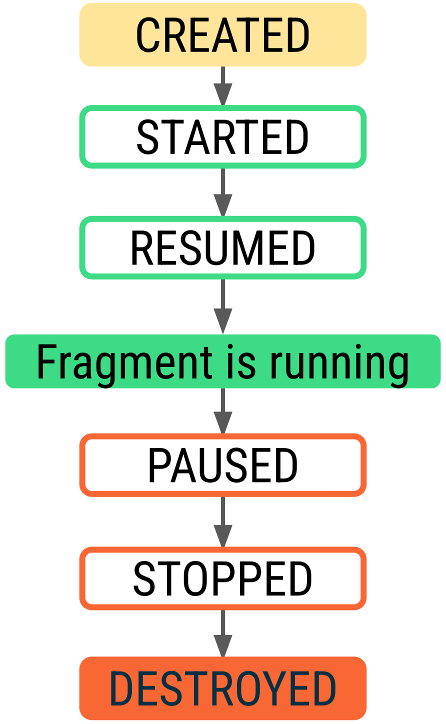
```
2020-05-26 15:19:42.111 5025-5025/com.example.myapplication I/zygote:     at void com.example.myapplication.MainActivity.onCreate(android.os.Bundle) (MainActivity.kt
2020-05-26 15:19:42.249 5025-5025/com.example.myapplication I/MainActivity: onStart
2020-05-26 15:19:42.255 5025-5025/com.example.myapplication I/MainActivity: onResume
2020-05-26 15:19:52.837 5025-5025/com.example.myapplication I/MainActivity: onPause
```

Write logs

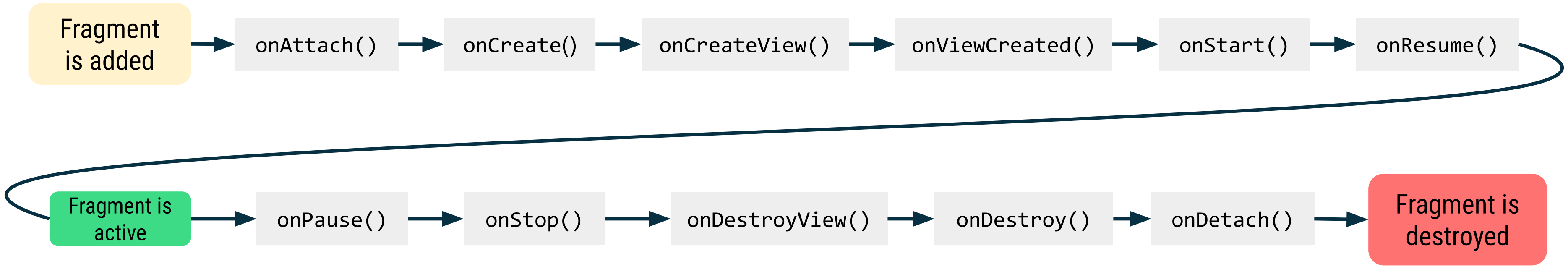
Priority level	**Log method**
Verbose	`Log.v(String, String)`
Debug	`Log.d(String, String)`
Info	`Log.i(String, String)`
Warning	`Log.w(String, String)`
Error	`Log.e(String, String)`

Fragment lifecycle

Fragment states



Fragment lifecycle diagram



onAttach()

- Called when a fragment is attached to a context
- Immediately precedes `onCreate()`

onCreateView()

- Called to create the view hierarchy associated with the fragment
- Inflate the fragment layout here and return the root view

onViewCreated()

- Called when view hierarchy has already been created
- Perform any remaining initialization here (for example, restore state from `Bundle`)

onDestroyView() and onDetach()

- `onDestroyView()` is called when view hierarchy of fragment is removed.
- `onDetach()` is called when fragment is no longer attached to the host.

Summary of fragment states

State	Callbacks	Description
Initialized	`onAttach()`	Fragment is attached to host.
Created	`onCreate()`, `onCreateView()`, `onViewCreated()`	Fragment is created and layout is being initialized.
Started	`onStart()`	Fragment is started and visible.
Resumed	`onResume()`	Fragment has input focus.
Paused	`onPause()`	Fragment no longer has input focus.
Stopped	`onStop()`	Fragment is not visible.
Destroyed	`onDestroyView()`, `onDestroy()`, `onDetach()`	Fragment is removed from host.

Save fragment state across config changes

Preserve UI state in fragments by storing state in `Bundle`:

- `onSaveInstanceState(outState: Bundle)`

Retrieve that data by receiving the Bundle in these fragment callbacks:

- `onCreate()`
- `onCreateView()`
- `onViewCreated()`

Lifecycle-aware components

Lifecycle-aware components

Adjust their behavior based on activity or fragment lifecycle

- Use the `androidx.lifecycle` library
- `Lifecycle` tracks the lifecycle state of an activity or fragment
    - Holds current lifecycle state
    - Dispatches lifecycle events (when there are state changes)

LifecycleOwner

- Interface that says this class has a lifecycle
- Implementers must implement `getLifecycle()` method

  Examples: `Fragment` and `AppCompatActivity` are implementations of `LifecycleOwner`

LifecycleObserver

Implement `LifecycleObserver` interface:

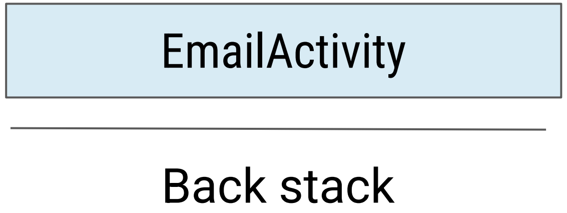
```
class MyObserver : LifecycleObserver {
    @OnLifecycleEvent(Lifecycle.Event.ON_RESUME)
    fun connectListener() {
        ...
    }
```

Add the observer to the lifecycle:

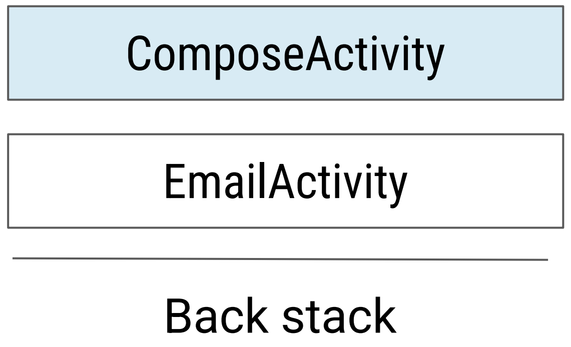
```
myLifecycleOwner.getLifecycle().addObserver(MyObserver())
```

Tasks and back stack

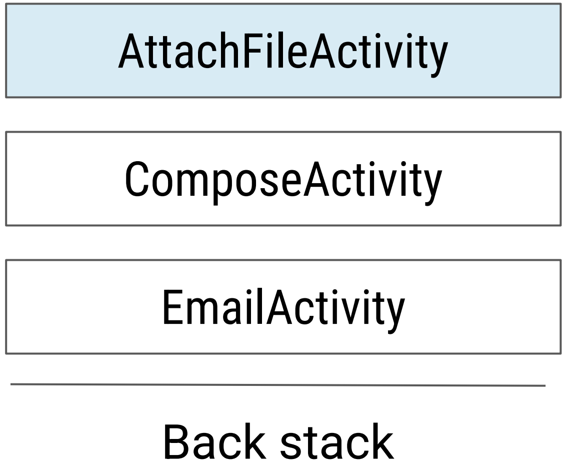
Back stack of activities



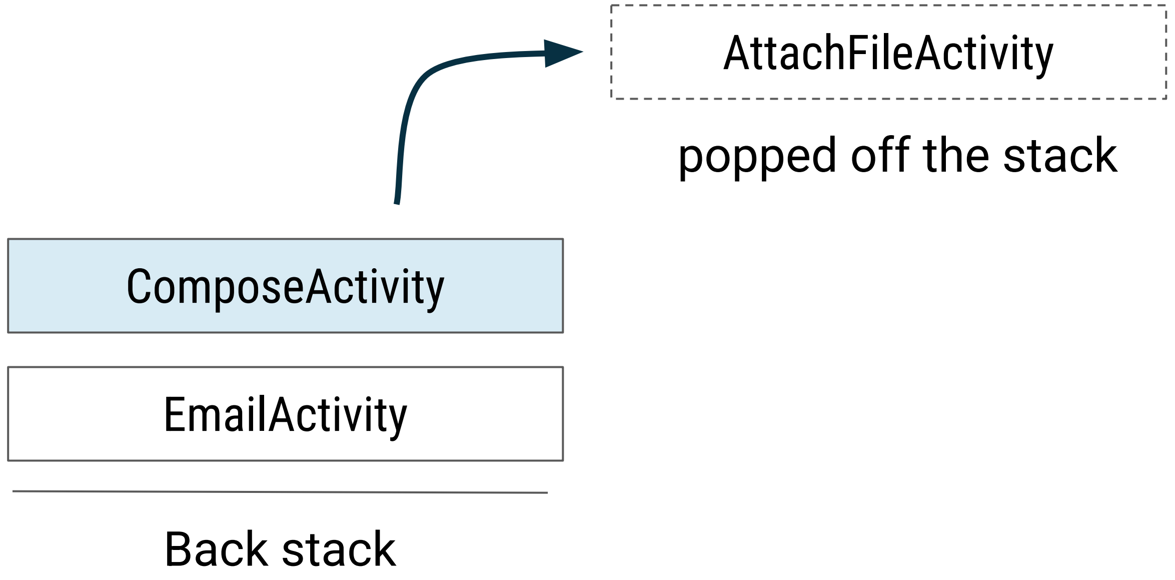
Add to the back stack



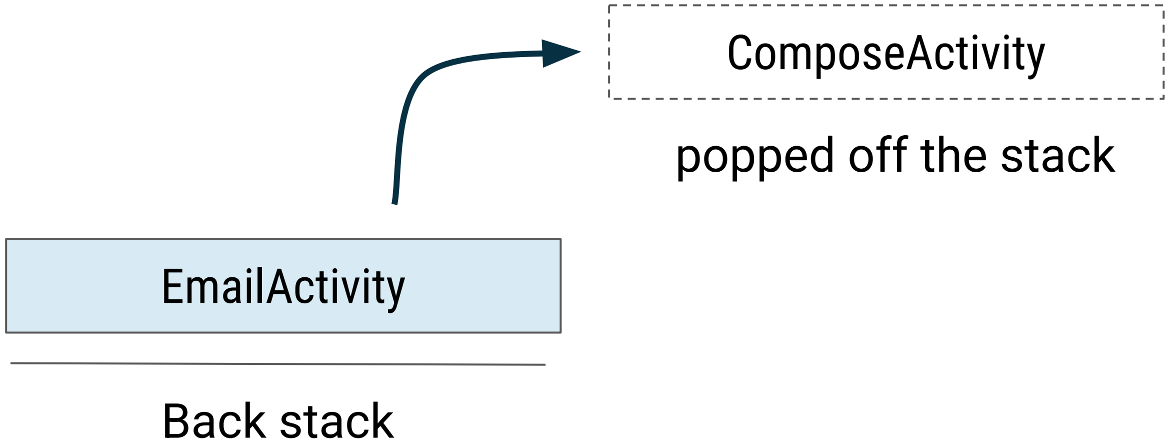
Add to the back stack again



Tap Back button

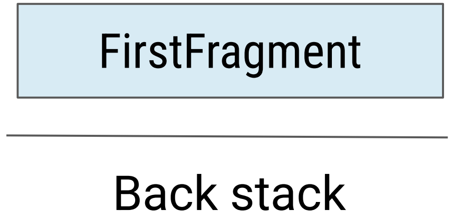


Tap Back button again

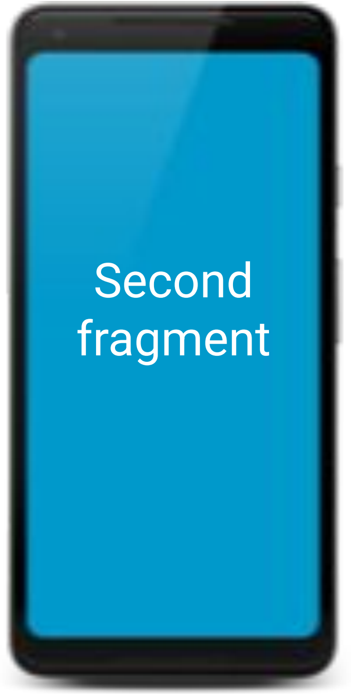


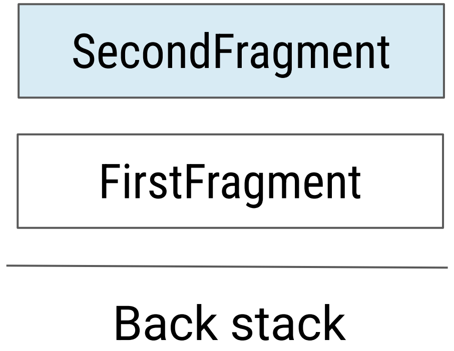
First destination in the back stack

First
fragment

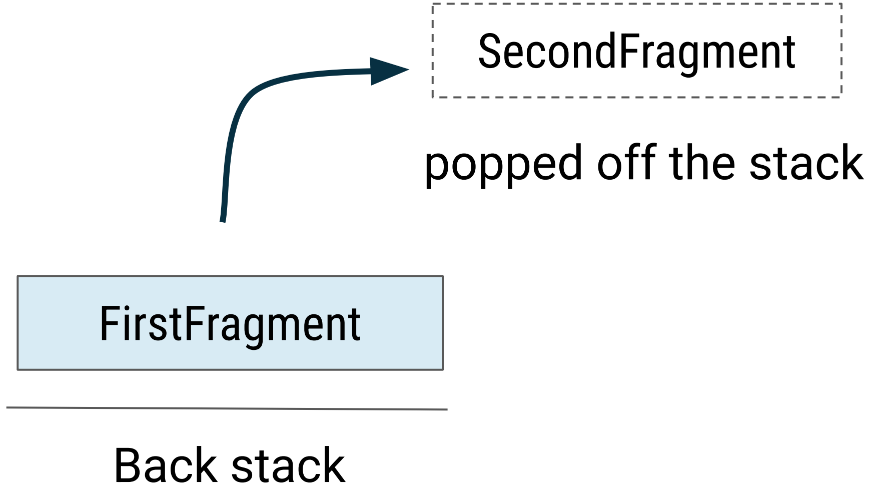


Add a destination to the back stack

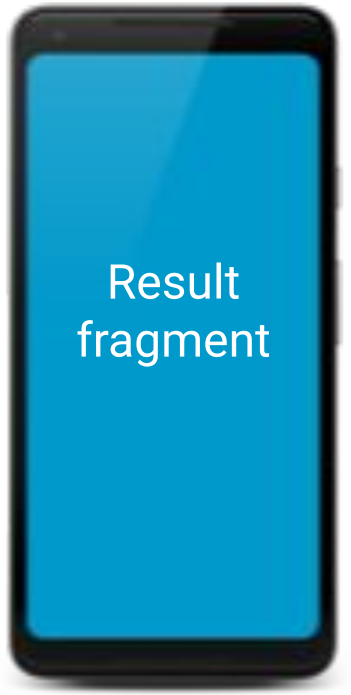


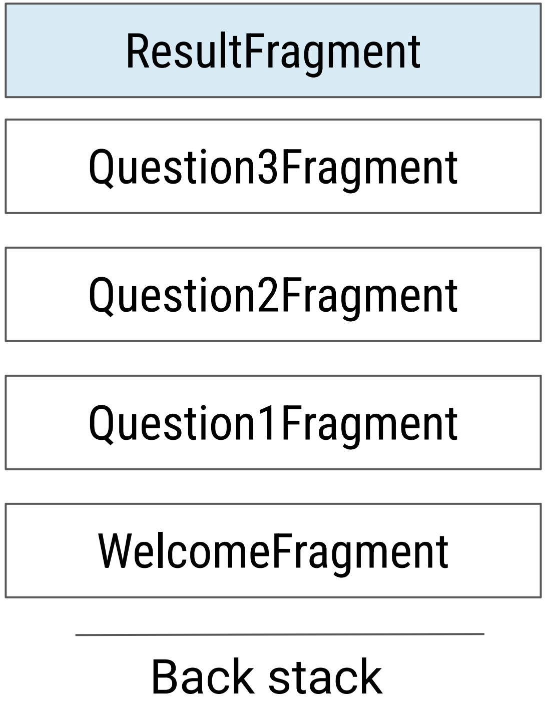


Tap Back button

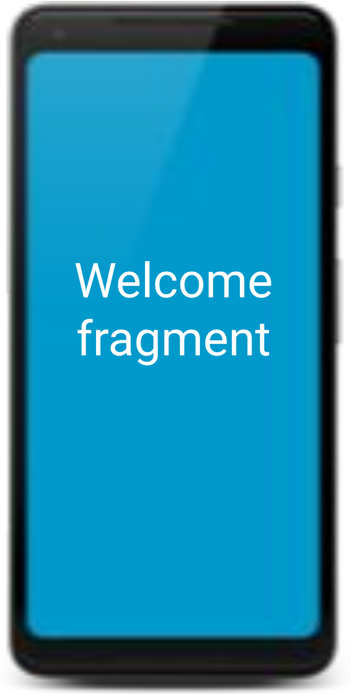


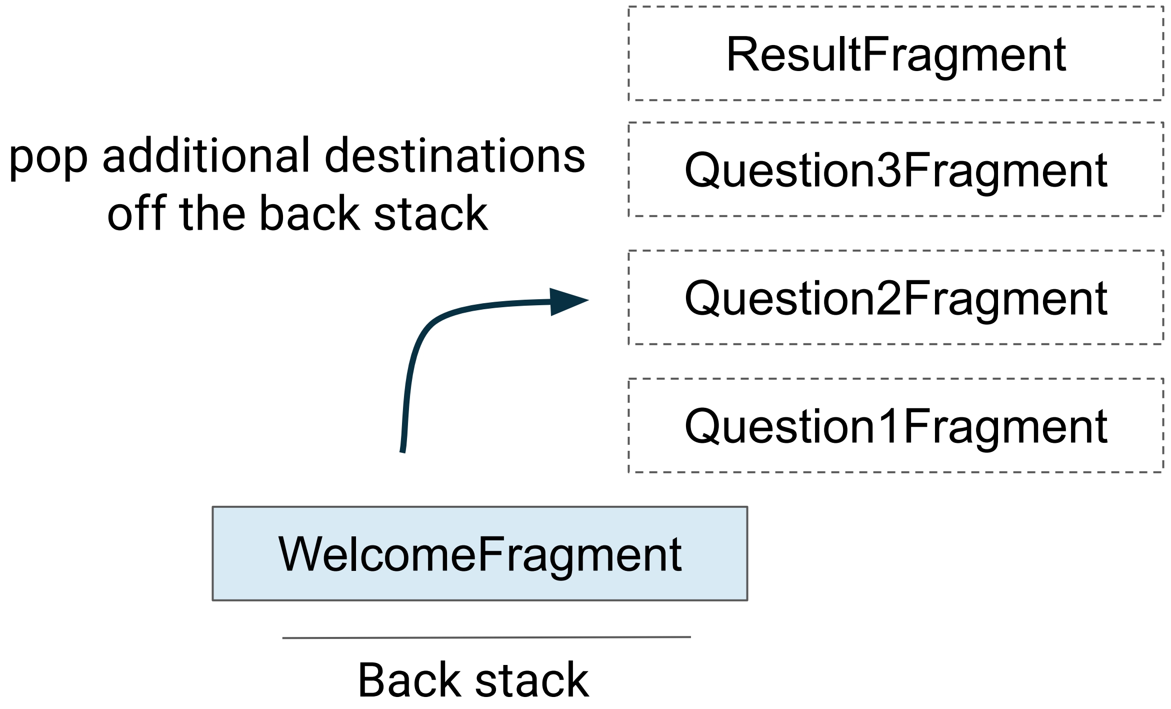
Another back stack example





Modify Back button behavior





Summary

Summary

In Lesson 7, you learned how to:

- Understand how an activity instance transitions through different lifecycle states as the user interacts with or leaves your app
- Reserve UI state across configuration changes using a `Bundle`
- Fragment lifecycle callback methods similar to activity, but with additions
- Use lifecycle-aware components help organize your app code
- Use default or custom back stack behavior
- Use logging to help debug and track the state of the app

Learn more

- Understand the Activity Lifecycle
- Activity class
- Fragments guide and lifecycle
- Fragment class

Pathway

Practice what you've learned by completing the pathway:

[Lesson 7: Activity and Fragment Lifecycles]()