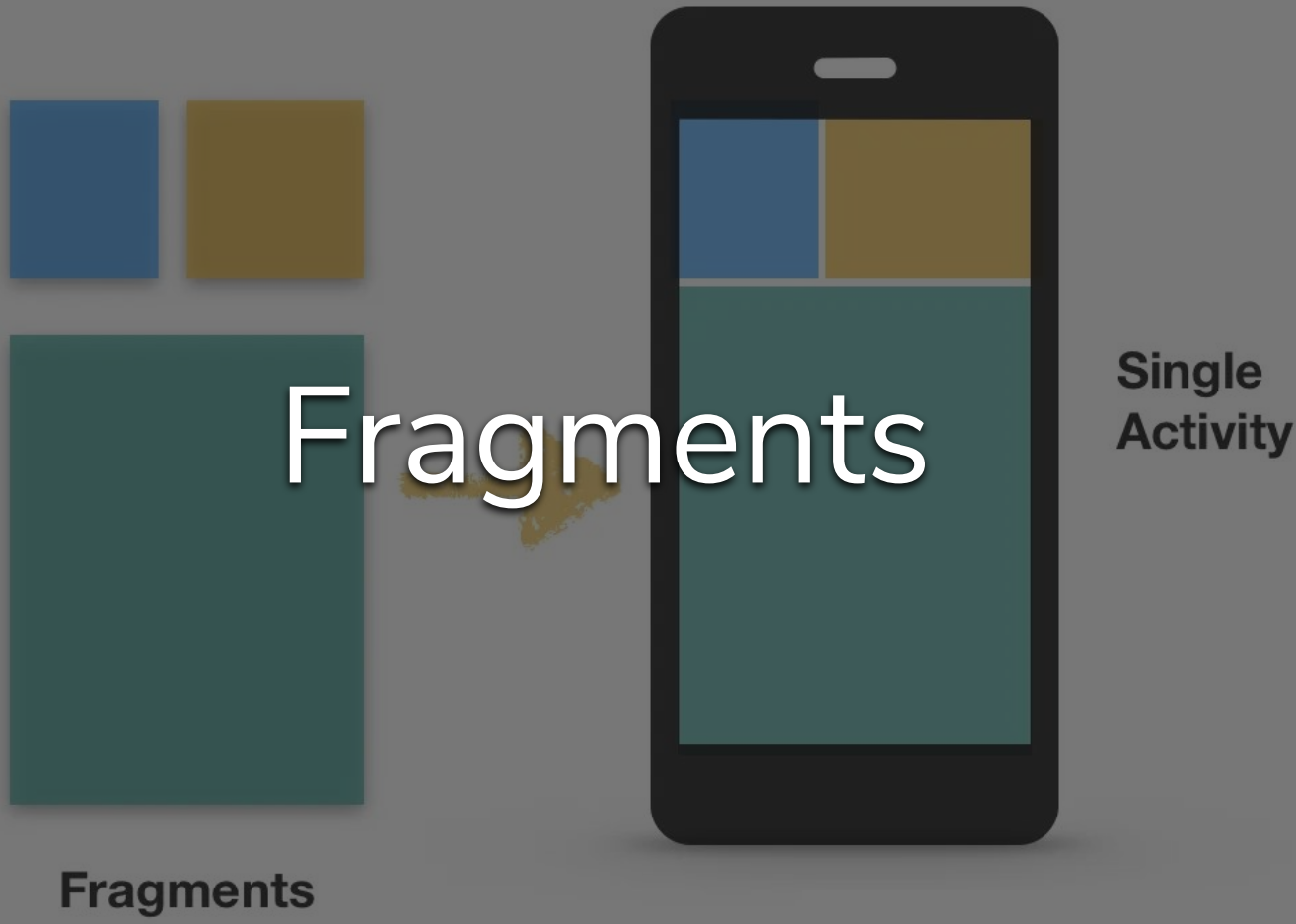
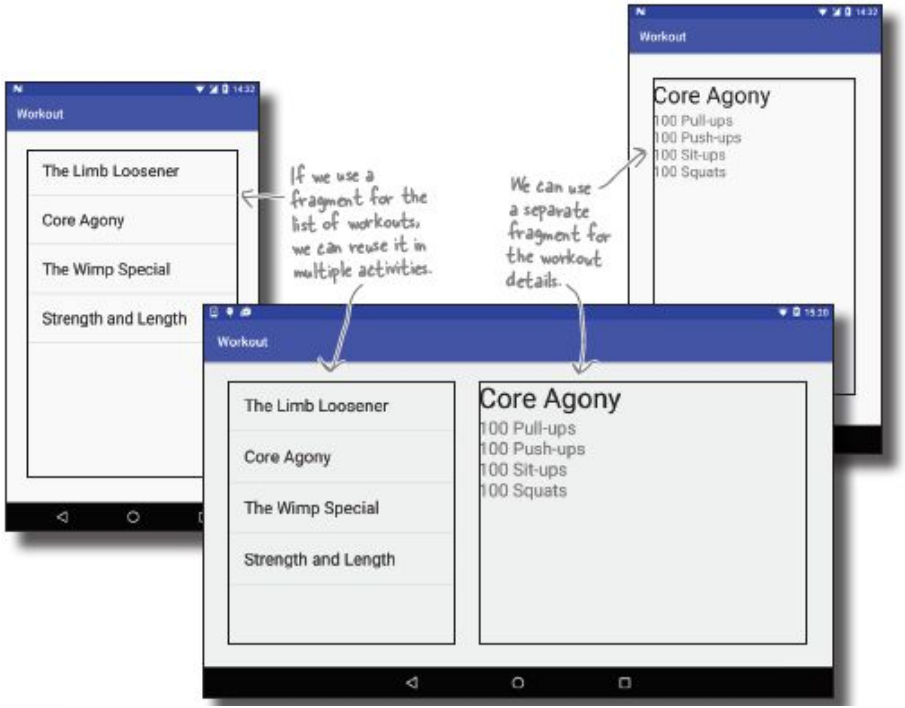

Fragments
Single
Activity
Fragments

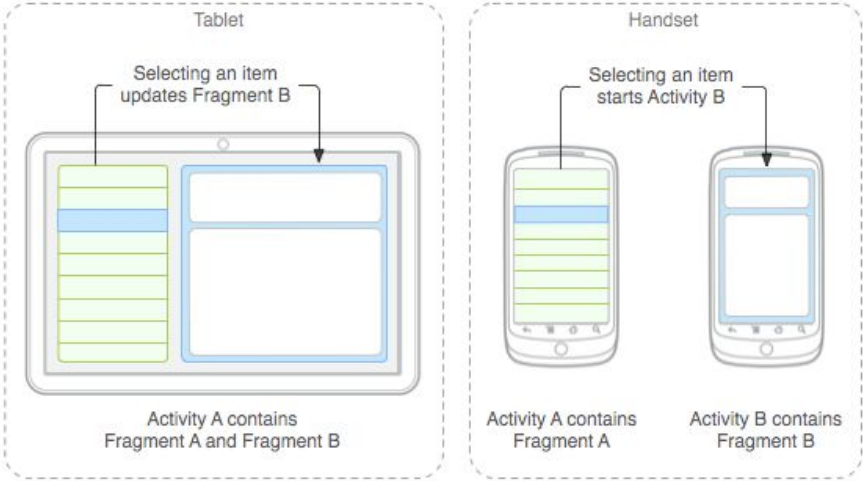
What are fragments?

- Combination of XML and Kotlin Class
  - Different from an Activity which is a full screen of UI
- Have own layouts, events, & lifecycle
  - Lifecycle is tied to host activity lifecycle
- Encapsulates views and logic
  - Can be added or removed at runtime
- Must be hosted inside Activities
  - Ie: a behavior/portion of UI in an activity ("microactivity")



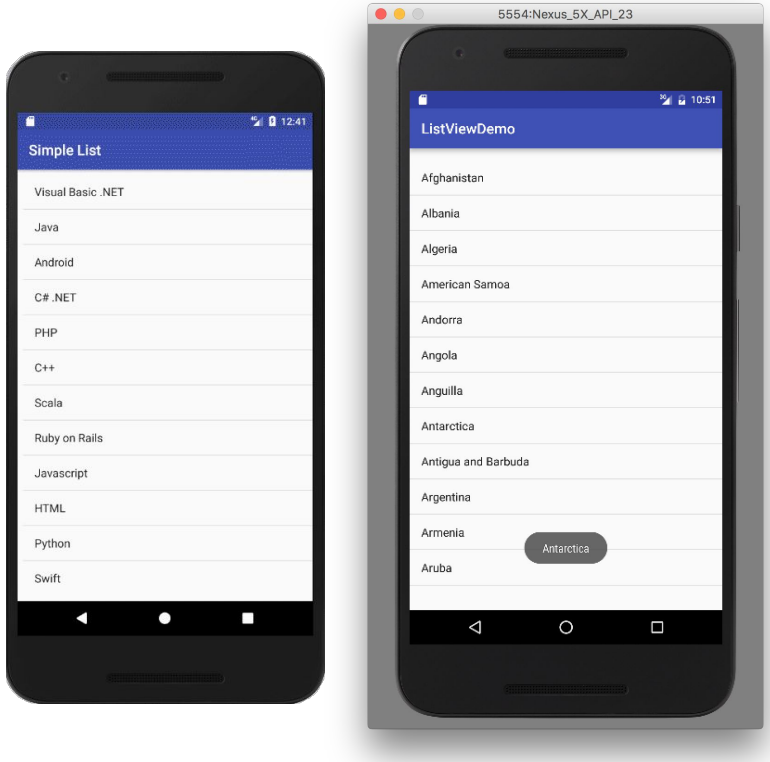
Why Fragments?

- **Handle different device form factors**
  - Different devices have different amounts of screen real-estate
  - What is the best way to utilize extra space on a tablet?
- Reusable Views and Logic
- Passing data between screens
- Organized User Interface
- Jetpack Navigation (future topic)
- Screen Orientation



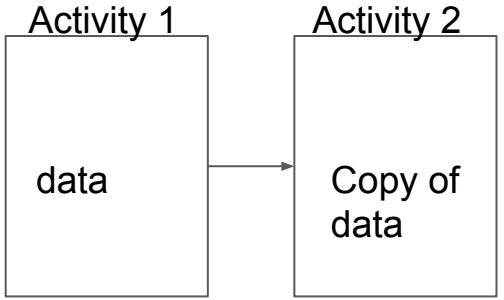
Why Fragments?

- Handle different device form factors
- **Reusable Views and Logic**
- Passing data between screens
- Organized User Interface
- Jetpack Navigation (future topic)
- Screen Orientation



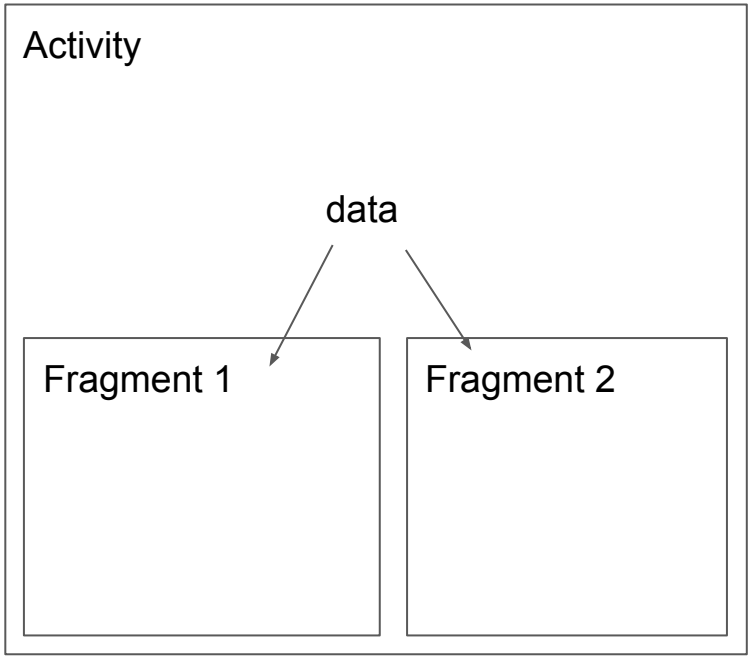
Why Fragments?

- Handle different device form factors
- Reusable Views and Logic
- **Passing data between screens**
  - Activities need to use intents and make sure objects are made parcelable to pass data
  - Have access to data within encapsulating Activity by reference
- Organized User Interface
- Jetpack Navigation (future topic)
- Screen Orientation



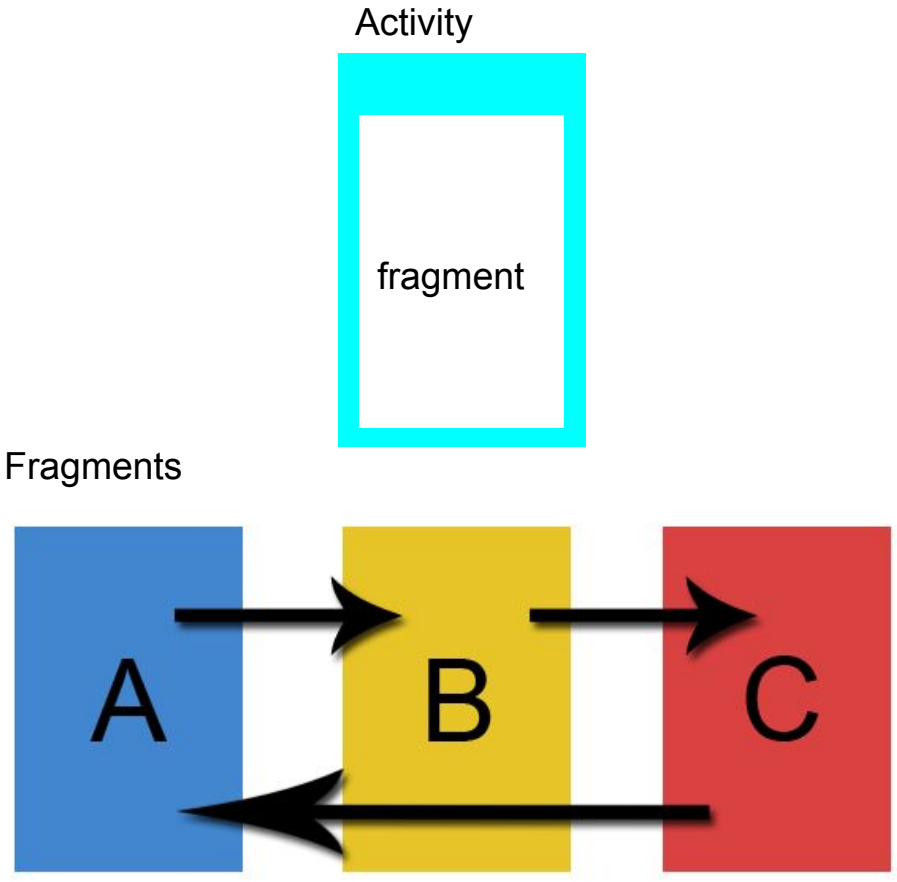
Why Fragments?

- Handle different device form factors
- Reusable Views and Logic
- **Passing data between screens**
  - Activities need to use intents and make sure objects are made parcelable to pass data
  - Have access to data within encapsulating Activity by reference
- Organized User Interface
- Jetpack Navigation (future topic)
- Screen Orientation



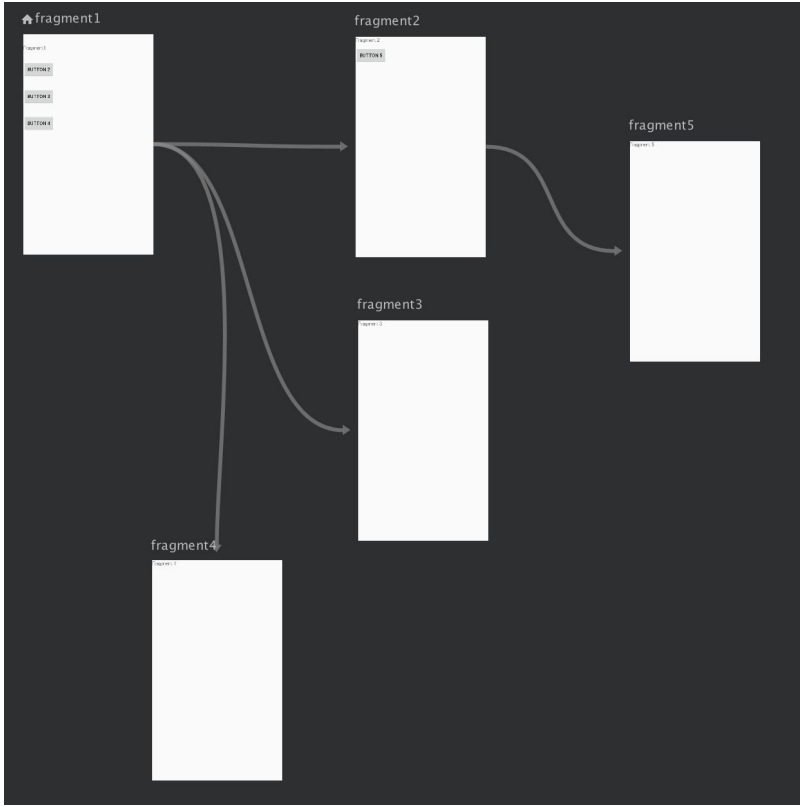
Why Fragments?

- Handle different device form factors
- Reusable Views and Logic
- Passing data between screens
- **Organized User Interface**
  - Makes it easier to understand the flow between different logical sections without replacing everything on the screen
  - Update UI for device rotation
- Jetpack Navigation (future topic)
- Screen Orientation



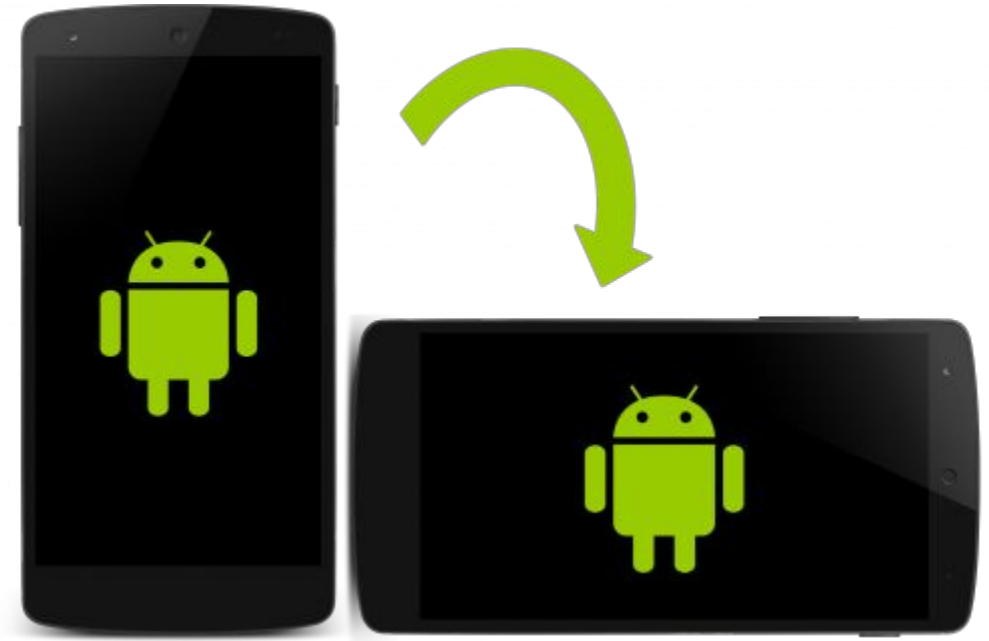
Why Fragments?

- Handle different device form factors
- Reusable Views and Logic
- Passing data between screens
- Organized User Interface
- **Jetpack Navigation (future topic)**
  - Simplify and standardize switching what the user sees on the screen and handles data passing
  - Built with fragment in mind
- Screen Orientation



Why Fragments?

- Handle different device form factors
- Reusable Views and Logic
- Passing data between screens
- Organized User Interface
- Jetpack Navigation (future topic)
- **Screen Orientation**



Fragments vs Activities

- A Fragment is like a mini-Activity within an Activity
  - Manages its own lifecycle
  - Receives its own input events

When to use Fragments

- Can be added or removed while parent Activity is running
- Multiple fragments can be combined in a single Activity
- Can be reused in more than one Activity

Organizing Activities and Fragments

Activities are **navigation controllers** primarily responsible for:

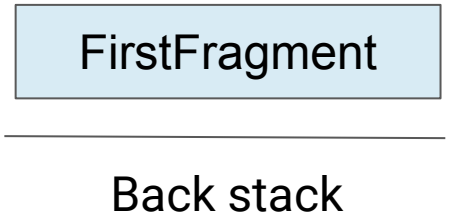
- Navigation to other activities through intents.
- Presenting navigational components such as the navigation drawer or the viewpager.
- Hiding and showing relevant fragments using the fragment manager.
- Receiving data from intents and passing data between fragments.

Fragments are **content controllers** and contain most views, layouts, and event logic including:

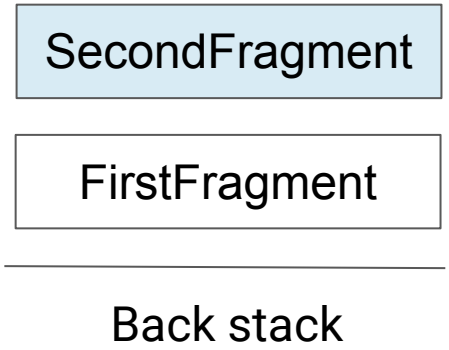
- Layouts and views displaying relevant app content.
- Event handling logic associated with relevant views.
- View state management logic such as visibility or error handling.
- Triggering of network request through a client object.
- Retrieval and storage of data from persistence through model objects.

Back Stack
& Lifecycle

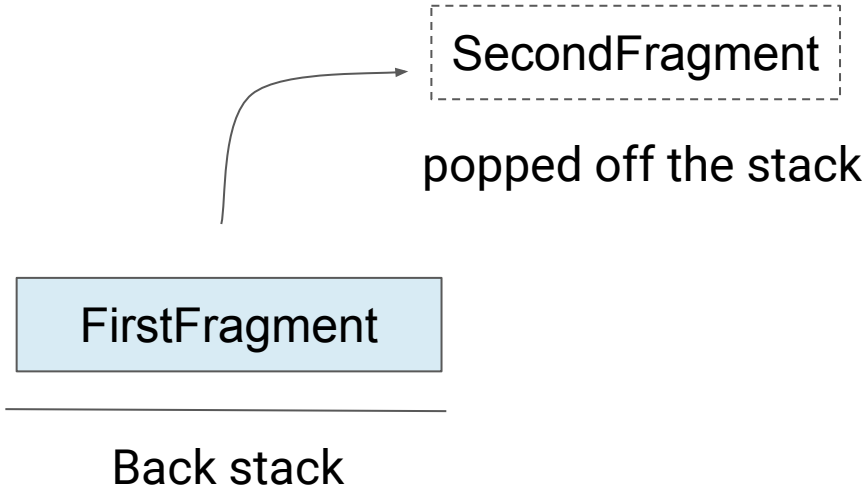
First destination in the back stack



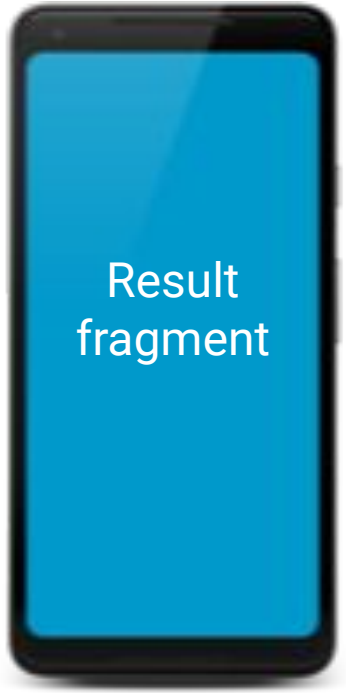
Add a destination to the back stack

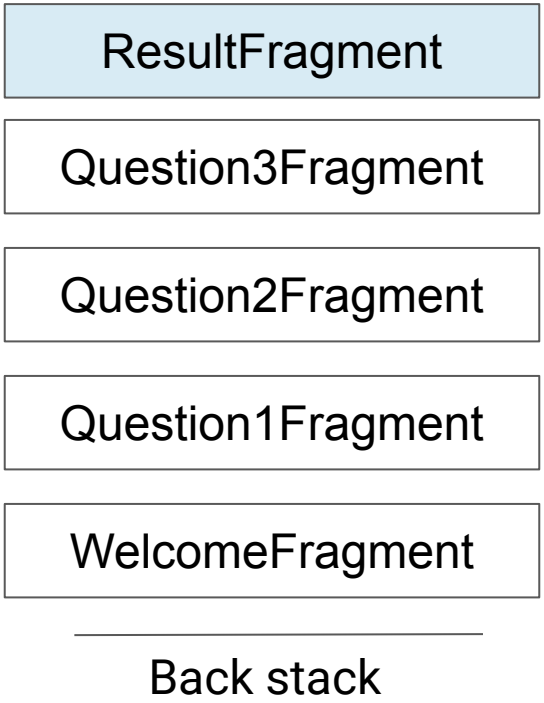


Tap Back button



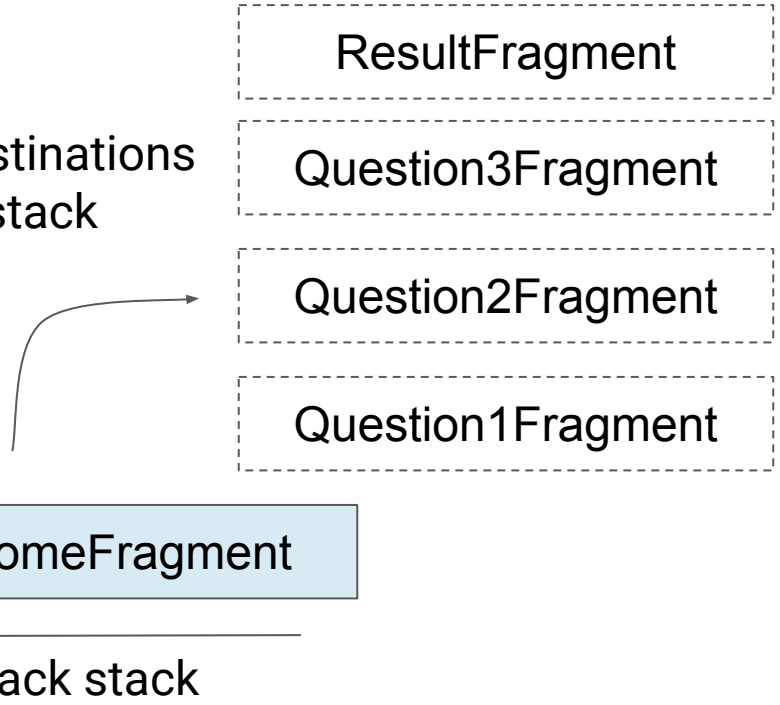
Another back stack example





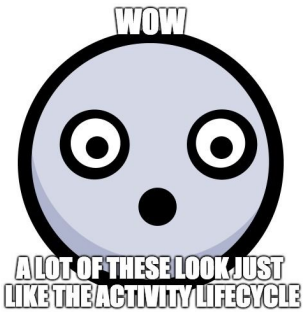
Modify Back button behavior

pop additional destinations
off the back stack



Fragment Lifecycle

- LifeCycle methods are inherited from a superclass
- Always call up to the superclass when implementing fragment lifecycle methods



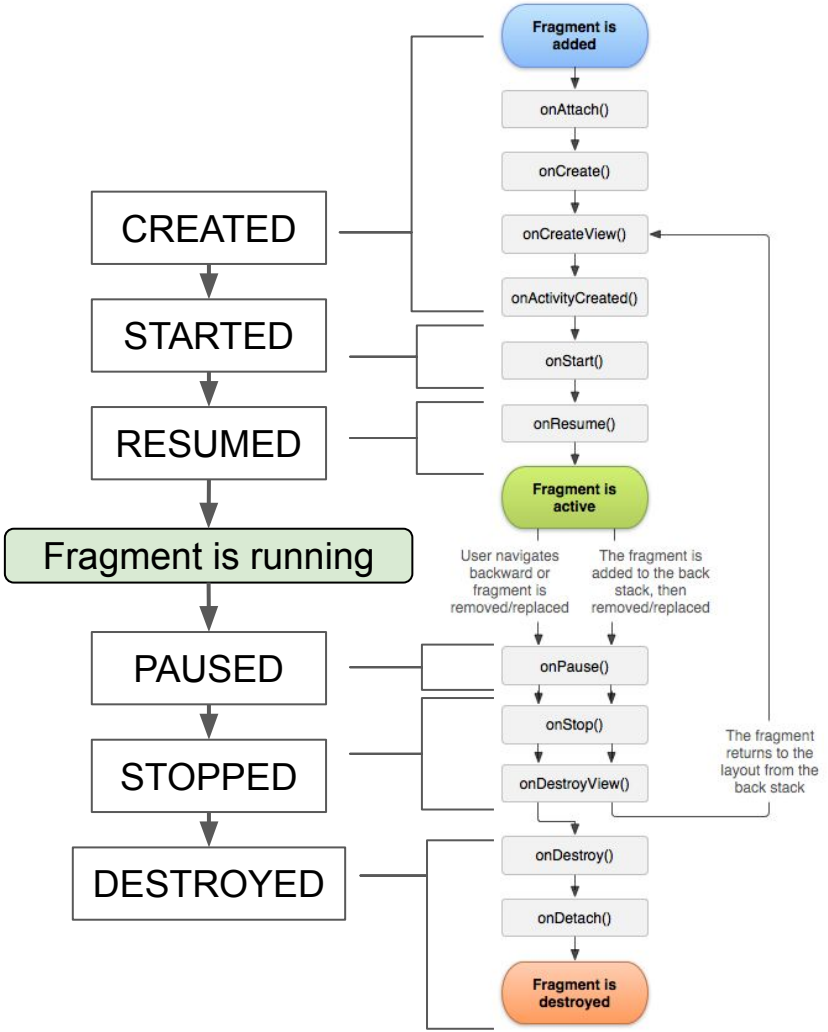
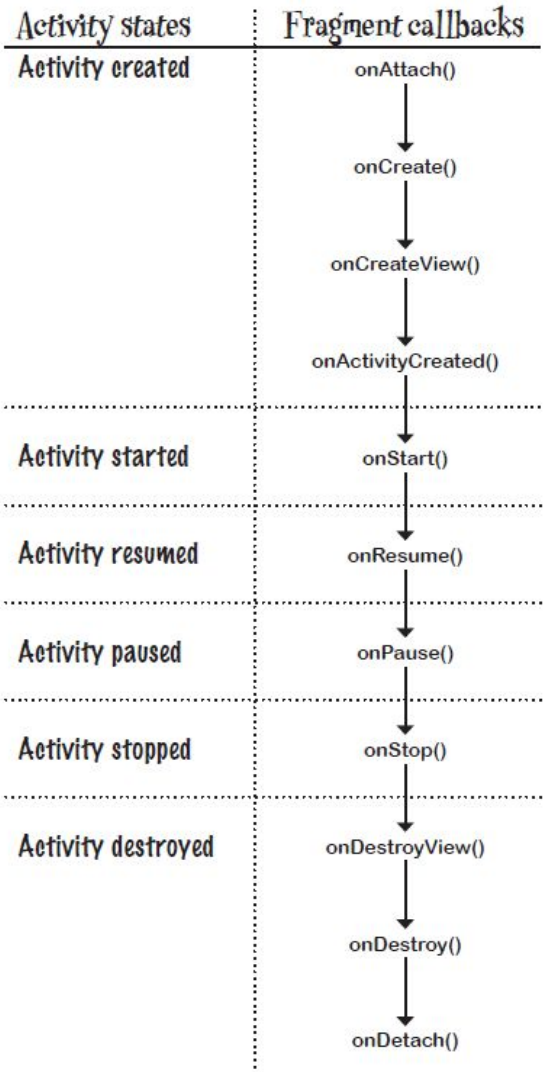


Figure 2. The lifecycle of a fragment (while its



onAttach(Context)
This happens when the fragment is associated with a context, in this case an activity.

onCreate(Bundle)
This is very similar to the activity's `onCreate()` method. It can be used to do the initial setup of the fragment.

onCreateView(LayoutInflater, ViewGroup, Bundle)
Fragments use a layout inflater to create their view at this stage.

onActivityCreated(Bundle)
Called when the `onCreate()` method of the activity has completed.

onStart()
Called when the fragment is about to become visible.

onResume()
Called when the fragment is visible and actively running.

onPause()
Called when the fragment is no longer interacting with the user.

onStop()
Called when the fragment is no longer visible to the user.

onDestroyView()
Gives the fragment the chance to clear away any resources that were associated with its view.

onDestroy()
In this method, the fragment can clear away any other resources it created.

onDetach()
Called when the fragment finally loses contact with the activity.

Notable Fragment Lifecycle Methods

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() & onDetach()

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

What do all the lifecycle methods do?

- onAttach() is called when a fragment is connected to an activity.
- onCreate() is called to do initial creation of the fragment.
- onCreateView() is called by Android once the Fragment should inflate a view.
- onViewCreated() is called after onCreateView() and ensures that the fragment's root view is non-null. Any view setup should happen here. E.g., view lookups, attaching listeners.
- onActivityCreated() is called when host activity has completed its onCreate() method.
- onStart() is called once the fragment is ready to be displayed on screen.
- onResume() - Allocate “expensive” resources such as registering for location, sensor updates, etc.
- onPause() - Release “expensive” resources. Commit any changes
- onDestroyView() is called when fragment's view is being destroyed, but the fragment is still kept around.
- onDestroy() is called when fragment is no longer in use.
- onDetach() is called when fragment is no longer connected to the activity.

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.

Subclasses of Fragments

- List Fragment
  - A type of fragment specializing in working with a list view. It has a default layout that contains the list view.
- DialogFragment
  - DialogFragment is a specialized Fragment used when you want to display an overlay modal window within an activity that floats on top of the rest of the content.
- PreferenceFragmentCompat
  - Displays a hierarchy of Preference objects as a list. This is used to create a settings screen for your application.
- And More!

Managing Fragments

Fragment Coding Guidelines

Use the AndroidX version of the `Fragment` class. (`androidx.fragment.app.Fragment`).

Don't use the platform version of the `Fragment` class (`android.app.Fragment`), which was deprecated.

Defining a Fragment Layout

```
<?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" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="TextView" />

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button" />

  </LinearLayout>
```

fragment_foo.xml

Define Fragment Class

```
import androidx.fragment.app.Fragment;

class FooFragment : Fragment() {
    // The onCreateView method is called when Fragment should create its View object
    // hierarchy, either dynamically or via XML layout inflation.
    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                                                        savedInstanceState: Bundle?): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_blank, container, false)
    }

    // This event is triggered soon after onCreateView().
    // Any view setup should occur here.  E.g., view lookups and attaching view listeners.
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        // Setup any handles to view objects here
        // EditText etFoo = (EditText) view.findViewById(R.id.etFoo);
    }

}
```

FooFragment.kt

Connecting Fragments inside Activities

Statically

- Place the exact fragment you want in the exact spot inside an Activity
- This can simply be done with xml only

Dynamically

- Use a FragmentManager to add, remove, and replace fragments within an activities layout at runtime
- This needs to be done in the activity logic

WARNING You cannot replace a fragment defined statically in the layout file via a FragmentTransaction. You can only replace fragments that you added dynamically

Adding Fragments Statically

```
<?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" >

    <fragment
        android:name="com.example.android.FooFragment"
        android:id="@+id/fooFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

activity_main.xml

Managing Fragments

Fragment Manager

- Unlike Activities, fragments largely need to be managed by the developer
- FragmentManager class (eventually replaced by NavController + NavGraph)

Fragment Transaction

- Perform actions (add, remove, replace) on fragments in response to user interaction inside an activity
- Each change you commit to an activity is called a transaction
- Transactions are committed using the FragmentTransaction API

Tools to Manage Fragments

- findFragmentById()
  - Get existing fragments in Activity
- popBackStack()
  - Pop fragment off Fragment backstack
- Add listener to backstack changes
  - addOnBackStackChangedListener()
- Open fragment transactions for adding and removing fragments

Adding Fragments Dynamically

```
<?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" >
    <FrameLayout
        android:id="@+id/your_placeholder"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    </FrameLayout>
</LinearLayout>
```

activity_main.xml

```
supportFragmentManager.commit {
    replace<FooFragment>(R.id.your_placeholder)
    setReorderingAllowed(true)
    addToBackStack("name") // name can be null
}
```

MainActivity.kt