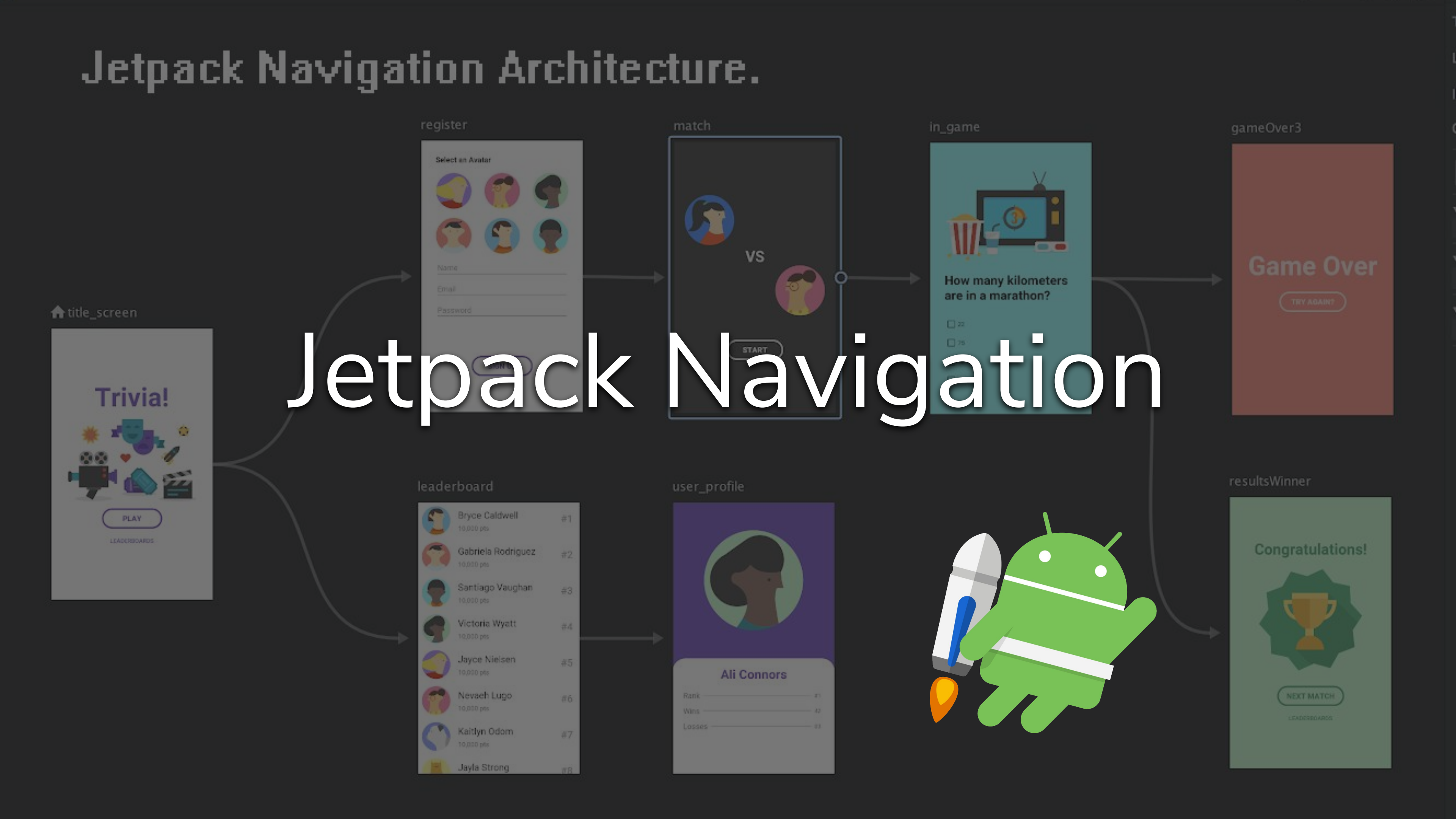
Jetpack Navigation



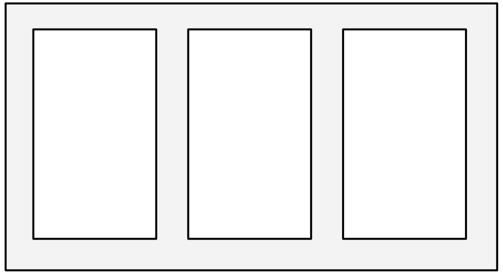
Overview

- Single Activity App Architecture
- Navigation
- Back Stack
- Safe Args

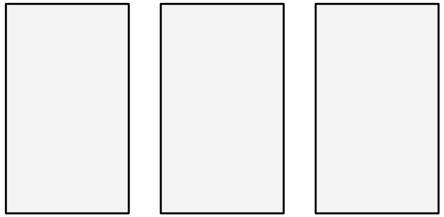
Single Activity Approach

- Use the best practice benefits of fragments
- Host all your fragments inside one Activity
- Treat your fragments as your base screens
- Keeps data localized and safe from Services and ContentProviders
- Easier to define things in the Manifest file
- & more!

Single Activity



Multiple Activity



Navigation

Navigation Component

Collection of libraries/tooling for creating navigational paths in an app

- Assumes one Activity per graph with many Fragment destinations

Navigation Graph

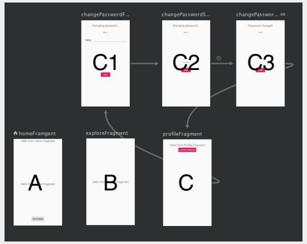
- A centralized XML resource containing navigation information such as destinations and paths

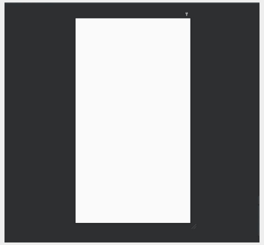
NavHost

- An empty container that displays destinations from the Navigation Graph

NavController

- Manages app navigation within a NavHost by swapping destination contents in the NavHost





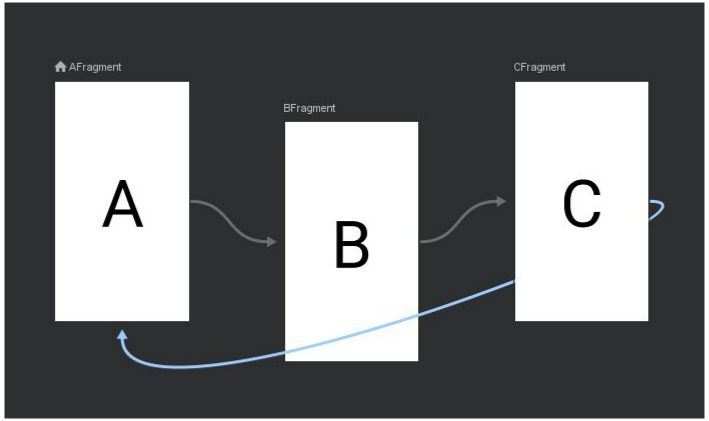
Navigation Graph Breakdown

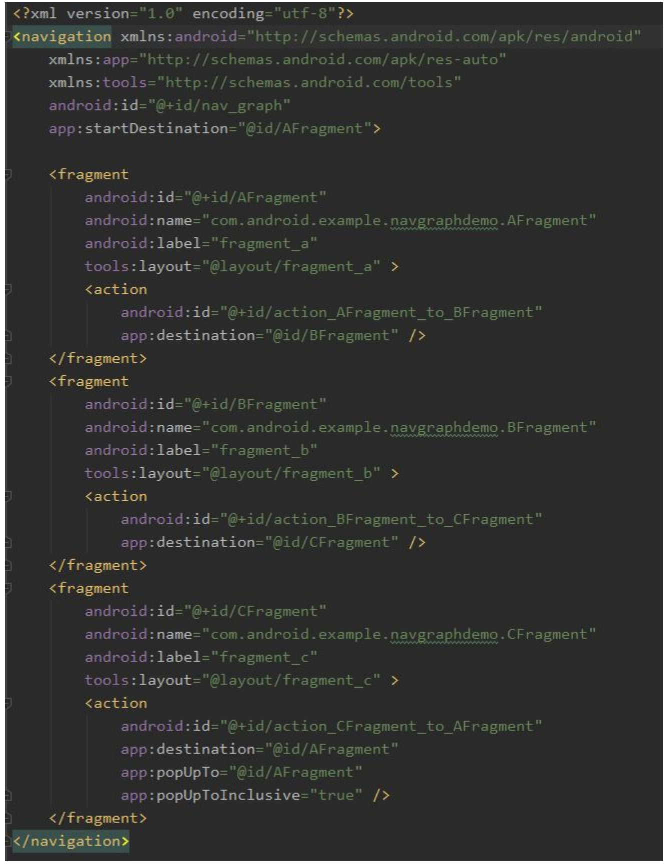
Destination

- Different content areas in your app

Action

- Logical connections between your destinations that represent paths that users can take





```xml
<?xml version="1.0" encoding="utf-8"?>
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/nav_graph"
    app:startDestination="@id/AFragment">

    <fragment
        android:id="@+id/AFragment"
        android:name="com.android.example.navgraphdemo.AFragment"
        android:label="fragment_a"
        tools:layout="@layout/fragment_a" >
        <action
            android:id="@+id/action_AFragment_to_BFragment"
            app:destination="@id/BFragment" />
    </fragment>
    <fragment
        android:id="@+id/BFragment"
        android:name="com.android.example.navgraphdemo.BFragment"
        android:label="fragment_b"
        tools:layout="@layout/fragment_b" >
        <action
            android:id="@+id/action_BFragment_to_CFragment"
            app:destination="@id/CFragment" />
    </fragment>
    <fragment
        android:id="@+id/CFragment"
        android:name="com.android.example.navgraphdemo.CFragment"
        android:label="fragment_c"
        tools:layout="@layout/fragment_c" >
        <action
            android:id="@+id/action_CFragment_to_AFragment"
            app:destination="@id/AFragment"
            app:popUpTo="@id/AFragment"
            app:popUpToInclusive="true" />
    </fragment>
</navigation>
```

Benefits of the Navigation Component

- Handles backstack
- Centralizes and visualizes navigation
- Simplifies common navigation patterns
- Implementing and handling deep linking
- Providing standardized resources for animations and transitions

Navigation Component Supports

- Working with Activities and Fragments and can be extended to custom views
- Passing arguments between screens with safe args
    - A Gradle plugin that provides type safety when navigating and passing data between destinations
- ViewModel
    - Scopes a ViewModel to a navigation graph to share UI-related data between the graph's destinations

Adding Dependencies - Navigation Component

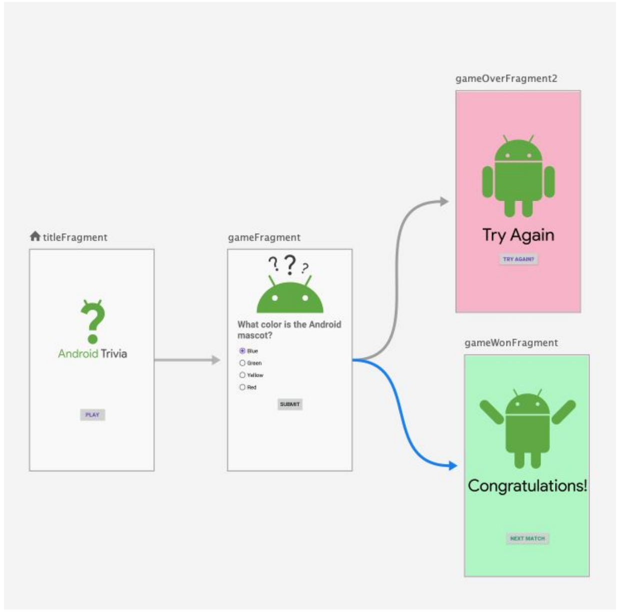
In `build.gradle`, under `dependencies`:

```
dependencies {
  def nav_version = "2.3.3"

  // Kotlin
  implementation "androidx.navigation:navigation-fragment-ktx:$nav_version"
  implementation "androidx.navigation:navigation-ui-ktx:$nav_version"

  // Feature module Support
  implementation "androidx.navigation:navigation-dynamic-features-fragment:$nav_version"

  // Testing Navigation
  androidTestImplementation "androidx.navigation:navigation-testing:$nav_version"

  // Jetpack Compose Integration
  implementation "androidx.navigation:navigation-compose:1.0.0-alpha06"

}
```

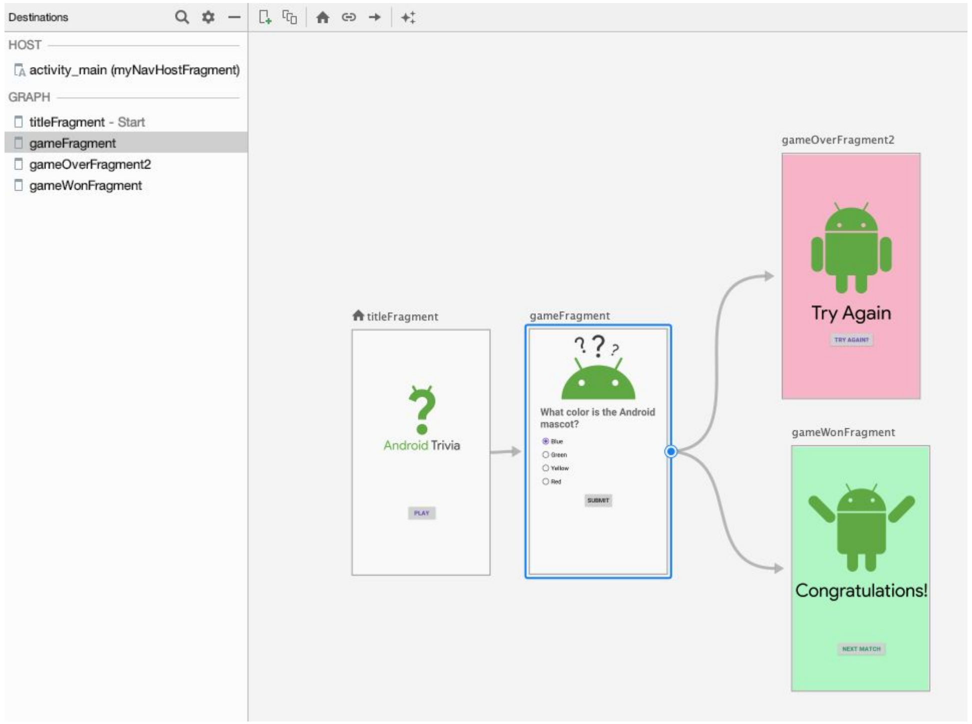
build.gradle

Navigation Graph

- Developer added resource located under `res/navigation`
- XML file containing all of your navigation destinations and actions
- Lists all the (Fragment/Activity) destinations that can be navigated to
- Lists the associated actions to traverse between them
- Optionally lists animations for entering or exiting



Android Studio Navigation Editor



NavHost

A placeholder empty container for our destinations

```xml
<LinearLayout ... >

    ...

    <fragment
        android:id="@+id/nav_host"
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:defaultNavHost="true"
        app:navGraph="@navigation/nav_graph_name"/>

    ...

</LinearLayout>
```

activity_main.xml

NavController

Manages the navigation host's UI navigation in a navigation host

- When specifying a destination path, the action is only named, not executed
- The NavController handles following the path & executing the action

```
class MainActivity : AppCompatActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
       ...
       val navController = findNavController(R.id.myNavHostFragment)
   }

   fun navigateToDetail() {
       navController.navigate(R.id.action_welcomeFragment_to_detailFragment)
   }
}
```

Navigating between Destinations

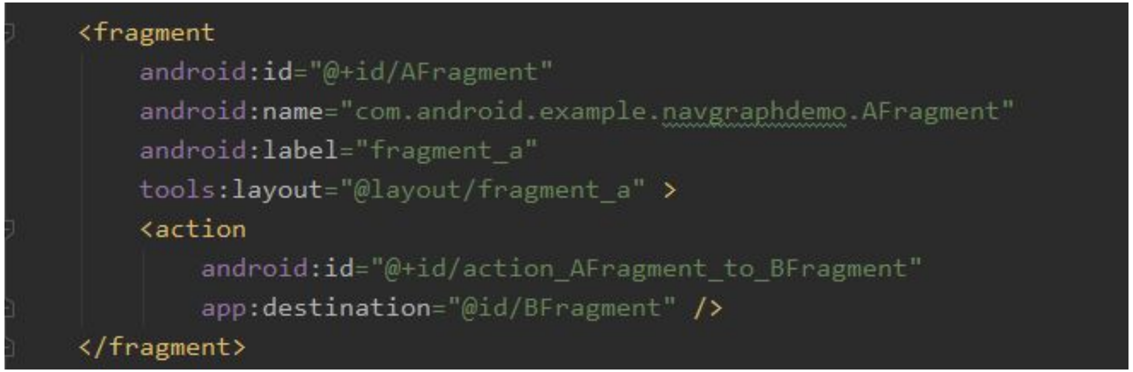
Review Creating a Fragment

- Extend `Fragment` class
- Override `onCreateView()`
- Inflate a layout for the Fragment that you have defined in XML

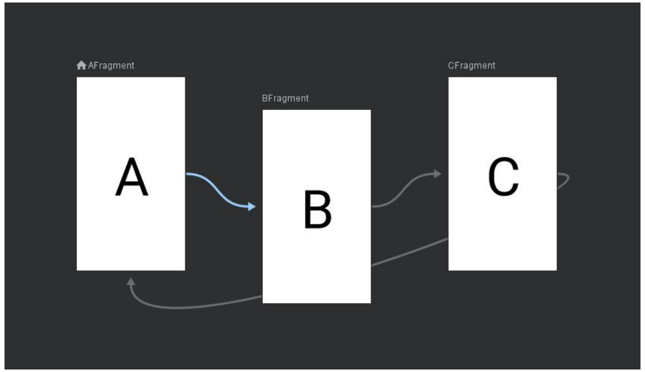
```
class DetailFragment : Fragment() {

   override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
           savedInstanceState: Bundle?): View? {
       return inflater.inflate(R.layout.detail_fragment, container, false)
   }
}
                                                              fragment_detail.xml
```

Specifying a Destination

- Fragment destinations are connected by the `action` tags in the navigation graph.
- Actions can be defined in XML directly or in the Navigation Editor by dragging from source to destination.
- Autogenerated action IDs take the form of `action_<sourceFragment>_to_<destinationFragment>` .



```
<fragment
    android:id="@+id/AFragment"
    android:name="com.android.example.navgraphdemo.AFragment"
    android:label="fragment_a"
    tools:layout="@layout/fragment_a" >
    <action
        android:id="@+id/action_AFragment_to_BFragment"
        app:destination="@id/BFragment" />
</fragment>
```



Destination Example

```
...

<fragment
    android:id="@+id/welcomeFragment"
    android:name="com.example.android.navigation.WelcomeFragment"
    android:label="fragment_welcome"
    tools:layout="@layout/fragment_welcome" >
    <action
        android:id="@+id/action_welcomeFragment_to_detailFragment"
        app:destination="@id/detailFragment" />
</fragment>


...
```

nav_graph.xml

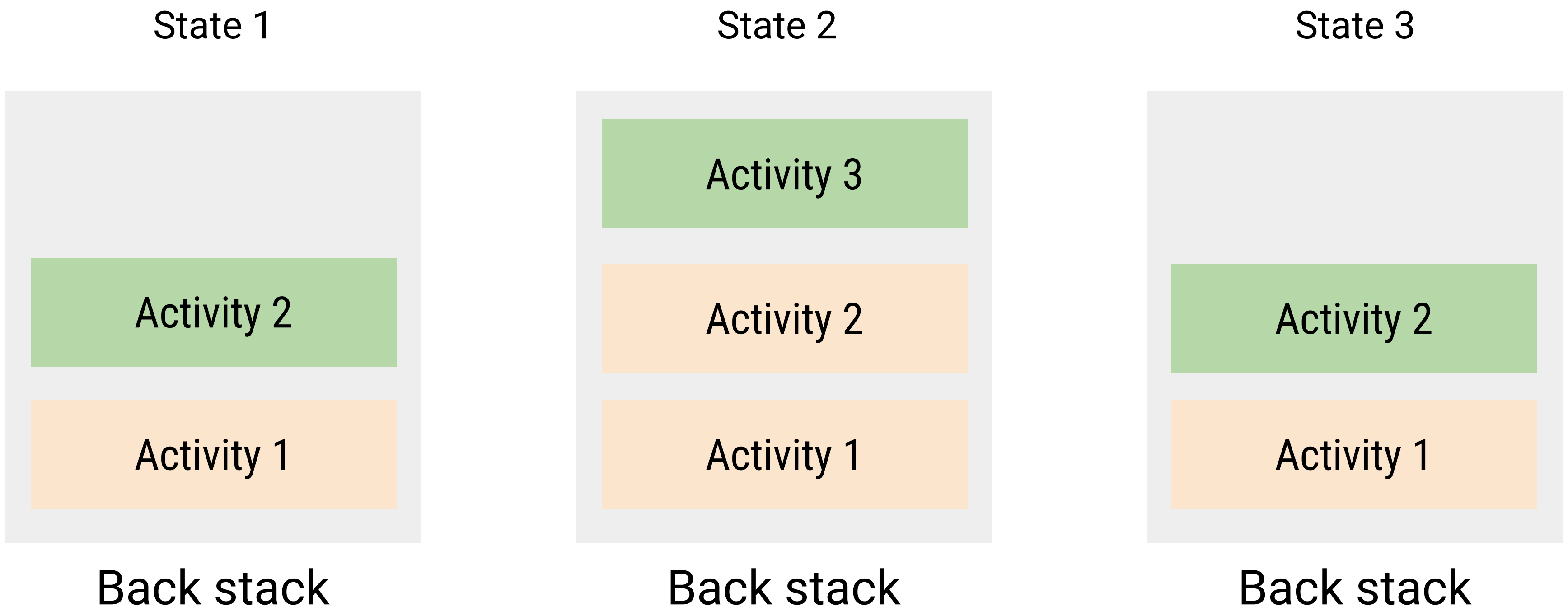
NavController Example

```kotlin
class MainActivity : AppCompatActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
       ...
       val navController = findNavController(R.id.myNavHostFragment)
   }

   fun navigateToDetail() {
       navController.navigate(R.id.action_welcomeFragment_to_detailFragment)
   }
}
```

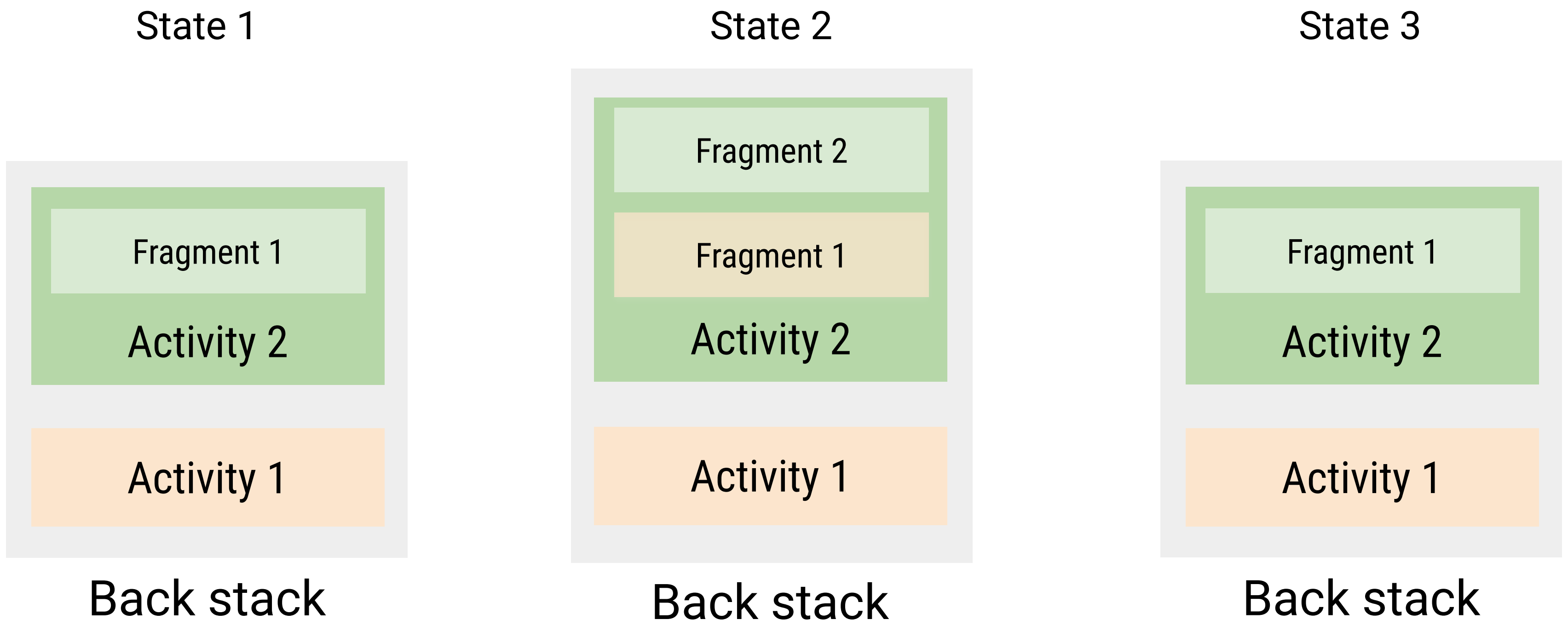
MainActivity.kt

Navigation Back Stack

Activity Back Stack Review



Fragment Back Stack Review



Jetpack Navigation Back Stack

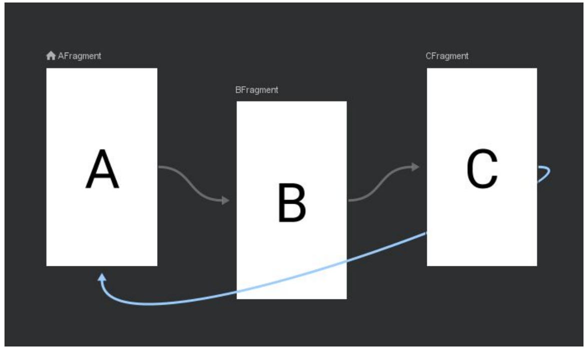
Common back stack interactions can be described in the Navigation Graph

popUpTo

- Pop destinations when navigating from one destination to another

popUpToInclusive

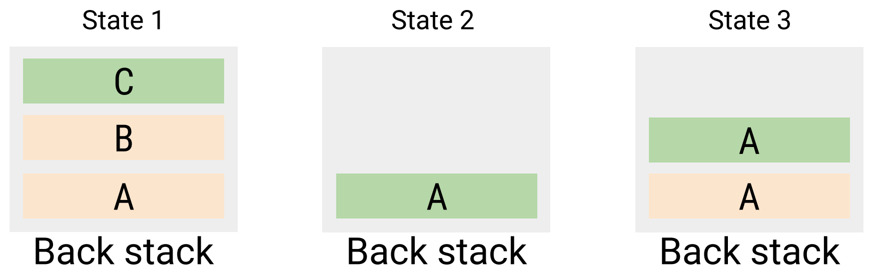
- Indicate whether the destination specified in app:popUpTo should also be removed from the back stack or kept



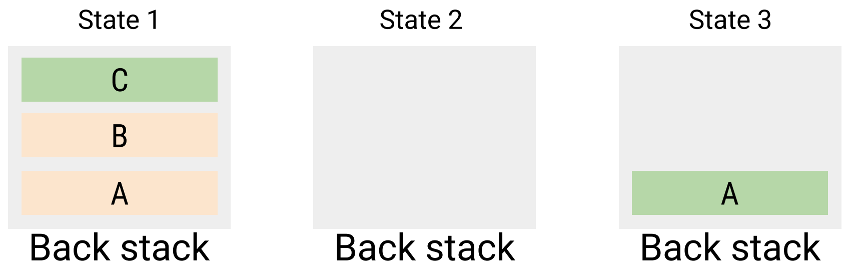
```
<fragment android:id="@+id/CFragment" ... >
    <action
        android:id="@+id/action_CFragment_to_AFragment"
        app:destination="@id/AFragment"
        app:popUpTo="@id/AFragment"
        app:popUpToInclusive="true" />
</fragment>
```

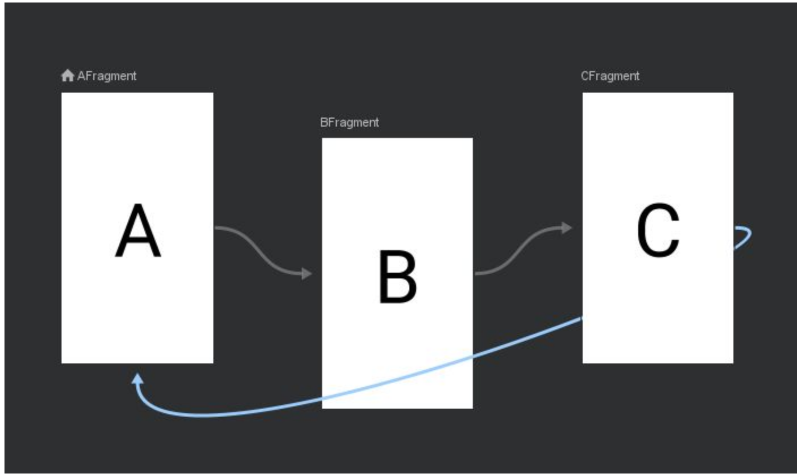
Navigation Back Stack Example

popUpTo A Example



popUpToInclusive Example





Safe Args

Passing Data using Safe Args

Using Safe Args:

- Ensures arguments have a valid type
- Lets you provide default values
- Generates a `<SourceDestination>Directions` class with methods for every action in that destination
- Generates a class to set arguments for every named action
- Generates a `<TargetDestination>Args` class providing access to the destination's arguments

Safe Args Setup

In the project `build.gradle` file:

```
buildscript {
   repositories {
       google()
   }
   dependencies {
       classpath "androidx.navigation:navigation-safe-args-gradle-plugin:$nav_version"
   }
}
```

In the app's or module's `build.gradle` file:

```
apply plugin: "androidx.navigation.safeargs.kotlin"
```

Supported argument types

Type	Type Syntax `app:argType=<type>`	Supports Default Values	Supports Null Values
Integer	`"integer"`	Yes	No
Float	`"float"`	Yes	No
Long	`"long"`	Yes	No
Boolean	`"boolean"`	Yes ("`true`" or "`false`")	No
String	`"string"`	Yes	Yes
Array	above type + "`[]`" (for example, "`string[]`" "`long[]`")	Yes (only "`@null`")	Yes
Enum	Fully qualified name of the enum	Yes	No
Resource reference	"reference"	Yes	No

Supported argument types: Custom classes

Type	Type Syntax `app:argType=<type>`	Supports Default Values	Supports Null Values
Serializable	Fully qualified class name	Yes (only "@null")	Yes
Parcelable	Fully qualified class name	Yes (only "@null")	Yes

Navigation with Safe Args

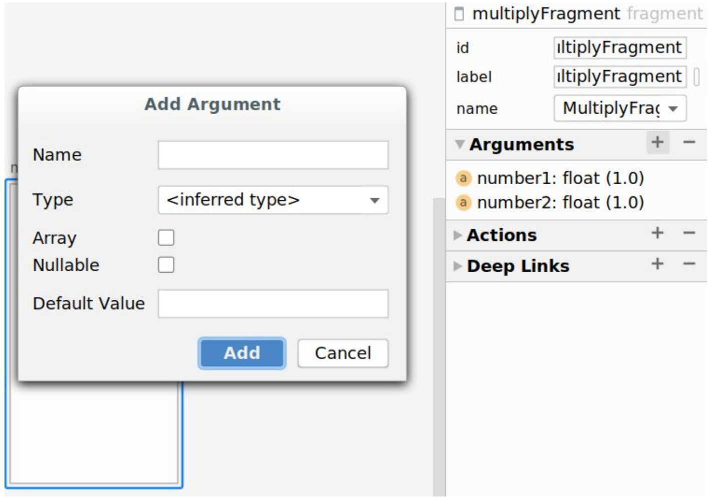
Sending Data to a Fragment

1. Create arguments the destination fragment will expect.
2. Create action to link from source to destination.
3. Set the arguments in the action method on `<SourceDestination>Directions`.
4. Navigate according to that action using the Navigation Controller.
5. Retrieve the arguments in the destination fragment.

Destination arguments

```
...

<fragment
    android:id="@+id/multiplyFragment"
    android:name="com.example.arithmetic.MultiplyFragment"
    android:label="MultiplyFragment" >
    <argument
        android:name="number1"
        app:argType="float"
        android:defaultValue="1.0" />
    <argument
        android:name="number2"
        app:argType="float"
        android:defaultValue="1.0" />
 </fragment>

...
```

nav_graph.xml



Add Argument

Name	
Type	<inferred type>
Array	☐
Nullable	☐
Default Value	

Add Cancel

multiplyFragment fragment

id	ıltiplyFragment
label	ıltiplyFragment
name	MultiplyFraç

Arguments

number1: float (1.0)
number2: float (1.0)

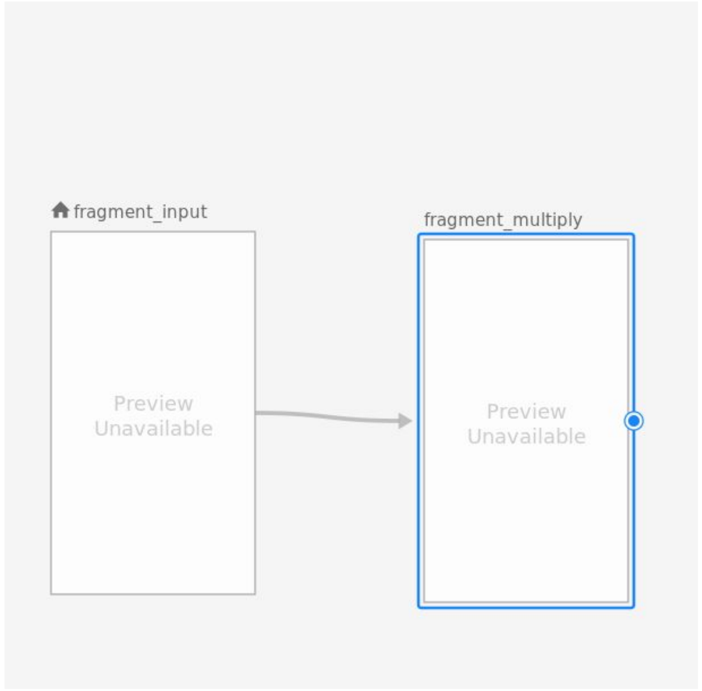
Actions

Deep Links

Create action from source to destination

```
...

<fragment
    android:id="@+id/fragment_input"
    android:name="com.example.arithmetic.InputFragment">
    <action
        android:id="@+id/action_to_multiplyFragment"
        app:destination="@id/multiplyFragment" />
</fragment>

...
```

nav_graph.xml



Navigating with actions

Sending Arguments

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    binding.button.setOnClickListener {
        val n1 = binding.number1.text.toString().toFloatOrNull() ?: 0.0
        val n2 = binding.number2.text.toString().toFloatOrNull() ?: 0.0

        val action = InputFragmentDirections.actionToMultiplyFragment(n1, n2)
        view.findNavController().navigate(action)
    }
}
```

InputFragment.kt

Navigating with actions

Retrieving Arguments

```
class MultiplyFragment : Fragment() {
   val args: MultiplyFragmentArgs by navArgs()
   lateinit var binding: FragmentMultiplyBinding
   override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
       super.onViewCreated(view, savedInstanceState)
       val number1 = args.number1
       val number2 = args.number2
       val result = number1 * number2
       binding.output.text = "${number1} * ${number2} = ${result}"
   }
}
```

MultiplyFragment.kt