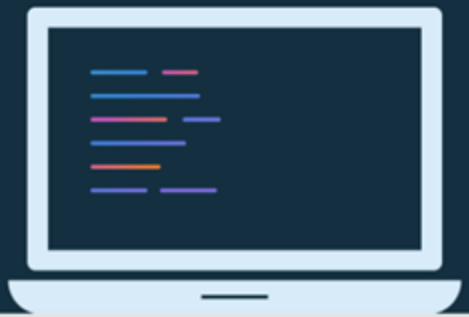
Lesson 6: App navigation



About this lesson

Lesson 6: App navigation

- Multiple activities and intents
- App bar, navigation drawer, and menus
- Fragments
- Navigation in an app
- More custom navigation behavior
- Navigation UI
- Summary

Multiple activities and intents

Multiple screens in an app

Sometimes app functionality may be separated into multiple screens.

Examples:

- View details of a single item (for example, product in a shopping app)
- Create a new item (for example, new email)
- Show settings for an app
- Access services in other apps (for example, photo gallery or browse documents)

Intent

Requests an action from another app component, such as another Activity

- An `Intent` usually has two primary pieces of information:
  - Action to be performed (for example, `ACTION_VIEW`, `ACTION_EDIT`, `ACTION_MAIN`)
  - Data to operate on (for example, a person's record in the contacts database)
- Commonly used to specify a request to transition to another Activity

Explicit intent

- Fulfills a request **using a specific component**
- Navigates internally to an Activity in your app
- Navigates to a specific third-party app or another app you've written

Explicit intent examples

Navigate between activities in your app:

```
fun viewNoteDetail() {
    val intent = Intent(this, NoteDetailActivity::class.java)
    intent.putExtra(NOTE_ID, note.id)
    startActivity(intent)
}
```

Navigate to a specific external app:

```
fun openExternalApp() {
    val intent = Intent("com.example.workapp.FILE_OPEN")
    if (intent.resolveActivity(packageManager) != null) {
        startActivity(intent)
    }
}
```

Implicit intent

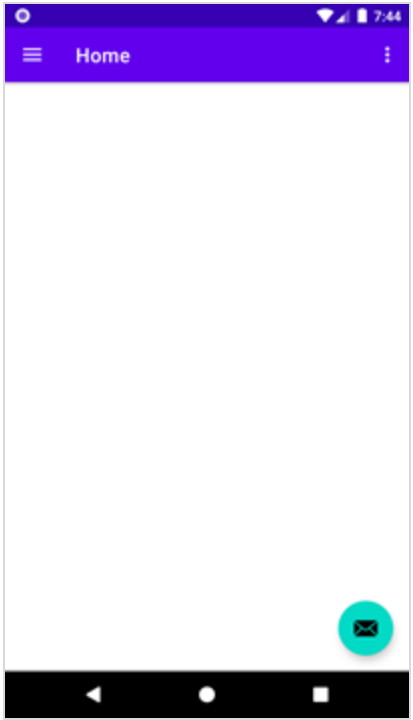
- Provides generic action the app can perform
- Resolved using mapping of the data type and action to known components
- Allows any app that matches the criteria to handle the request

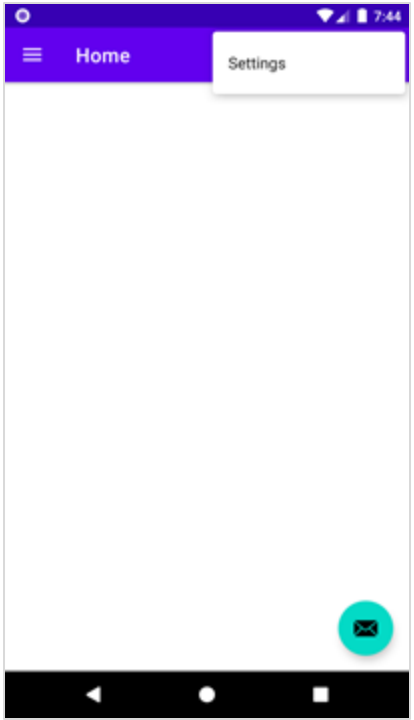
Implicit intent example

```
fun sendEmail() {
    val intent = Intent(Intent.ACTION_SEND)
    intent.type = "text/plain"
    intent.putExtra(Intent.EXTRA_EMAIL, emailAddresses)
    intent.putExtra(Intent.EXTRA_TEXT, "How are you?")

    if (intent.resolveActivity(packageManager) != null) {
        startActivity(intent)
    }
}
```

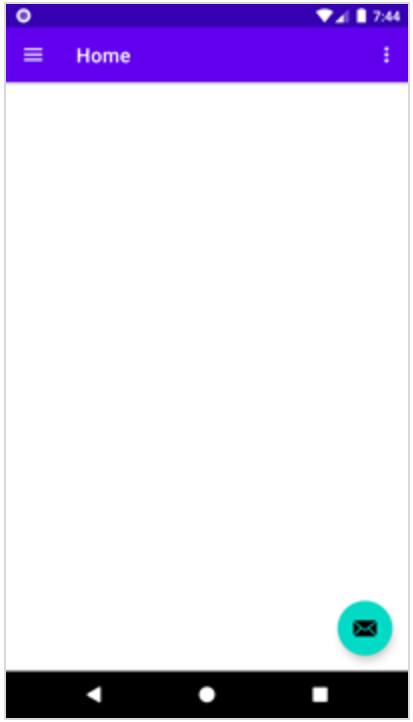
App bar, navigation drawer, and menus

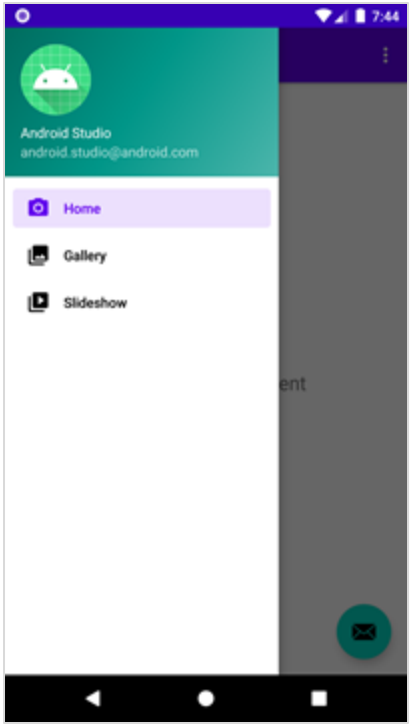
App bar





Navigation drawer





Menu

Define menu items in XML menu resource (located in `res/menu` folder)

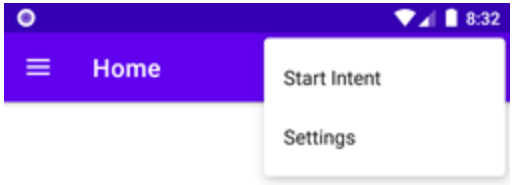
```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="@string/action_settings"
        app:showAsAction="never" />
</menu>
```

More menu options

```
<menu>
    <group android:checkableBehavior="single">
        <item
            android:id="@+id/nav_home"
            android:icon="@drawable/ic_menu_camera"
            android:title="@string/menu_home" />
        <item
            android:id="@+id/nav_gallery"
            android:icon="@drawable/ic_menu_gallery"
            android:title="@string/menu_gallery" />
        <item
            android:id="@+id/nav_slideshow"
            android:icon="@drawable/ic_menu_slideshow"
            android:title="@string/menu_slideshow" />
    </group>

</menu>
```

Options menu example

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item android:id="@+id/action_intent"
        android:title="@string/action_intent" />

    <item
        android:id="@+id/action_settings"
        android:orderInCategory="100"
        android:title="@string/action_settings"
        app:showAsAction="never" />
</menu>
```



Inflate options menu

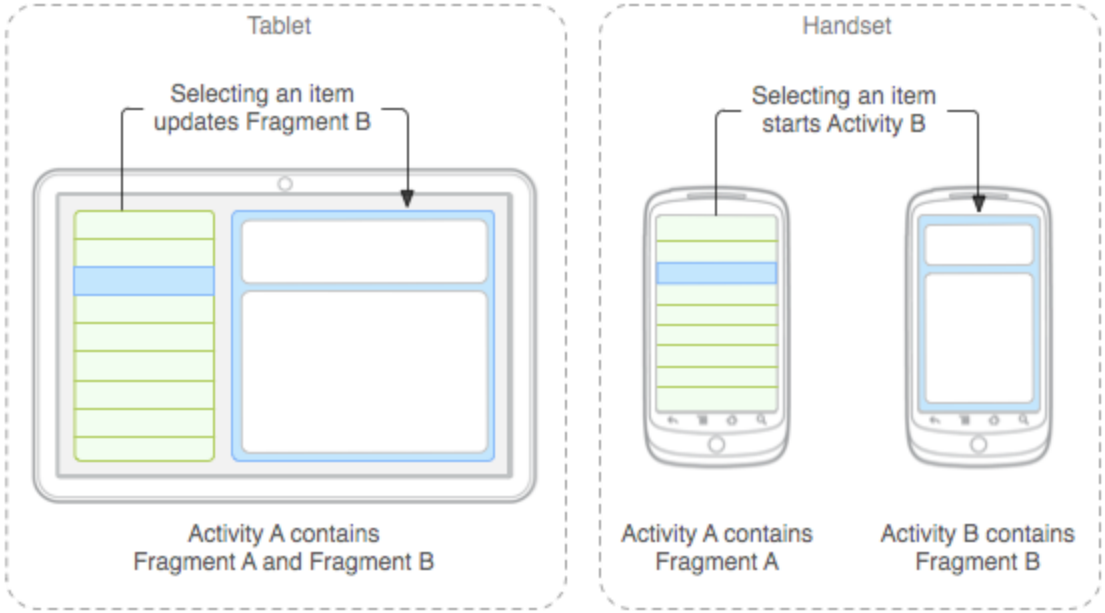
```
override fun onCreateOptionsMenu(menu: Menu): Boolean {
    menuInflater.inflate(R.menu.main, menu)
    return true
}
```

Handle menu options selected

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {

    when (item.itemId) {
        R.id.action_intent -> {
            val intent = Intent(Intent.ACTION_WEB_SEARCH)
            intent.putExtra(SearchManager.QUERY, "pizza")
            if (intent.resolveActivity(packageManager) != null) {
                startActivity(intent)
            }
        }

        else -> Toast.makeText(this, item.title, Toast.LENGTH_LONG).show()
    ...
```

Fragments

Fragments for tablet layouts



Fragment

- Represents a behavior or portion of the UI in an activity ("microactivity")
- Must be hosted in an activity
- Lifecycle tied to host activity's lifecycle
- Can be added or removed at runtime

Note about fragments

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.

Navigation within an app

Navigation component

- Collection of libraries and tooling, including an integrated editor, for creating navigation paths through an app
- Assumes one `Activity` per graph with many `Fragment` destinations
- Consists of three major parts:
  - Navigation graph
  - Navigation Host (`NavHost`)
  - Navigation Controller (`NavController`)

Add dependencies

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

```
implementation "androidx.navigation:navigation-fragment-ktx:$nav_version"

implementation "androidx.navigation:navigation-ui-ktx:$nav_version"
```

Navigation host (NavHost)

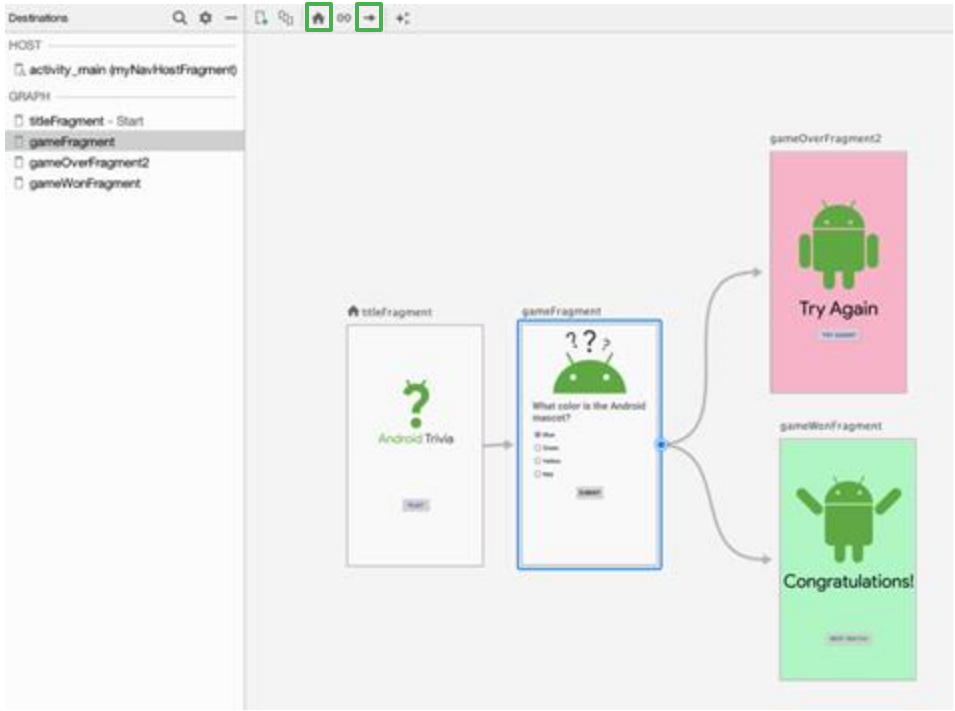
```
<fragment
    android:id="@+id/nav_host"
    android:name="androidx.navigation.fragment.NavHostFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:defaultNavHost="true"
    app:navGraph="@navigation/nav_graph_name"/>
```

Navigation graph

New resource type located in `res/navigation` directory

- 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

Navigation Editor in Android Studio



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)
    }
}
```

Specifying Fragment destinations

- Fragment destinations are denoted by the `action` tag 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>`.

Example fragment destination

```
<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>
```

Navigation Controller (NavController)

`NavController` manages UI navigation in a navigation host.

- Specifying a destination path only names the action, but it doesn't execute it.
- To follow a path, use `NavController`.

Example NavController

```
class MainActivity : AppCompatActivity() {

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

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

More custom navigation behavior

Passing data between destinations

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

Setting up Safe Args

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:

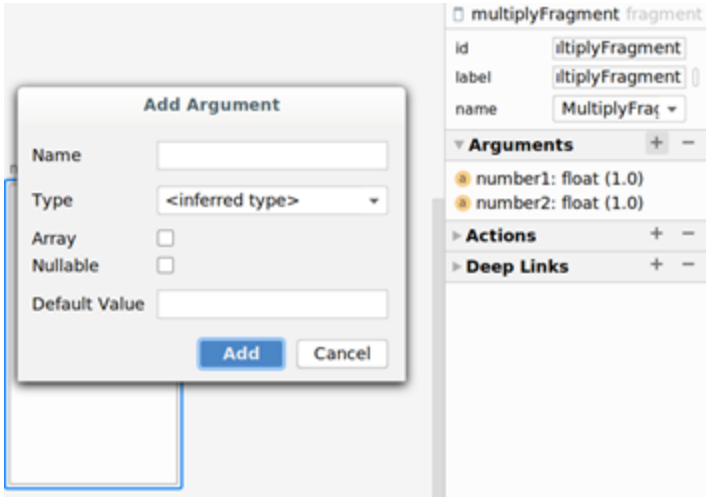
```
apply plugin: "androidx.navigation.safeargs.kotlin"
```

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 `<Source>FragmentDirections`.
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>
```



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

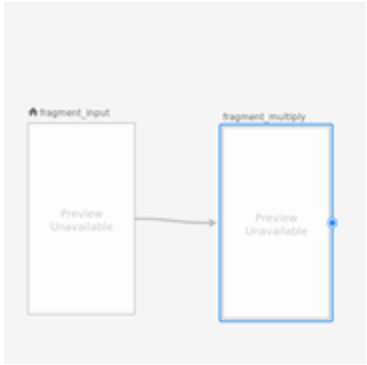
Supported argument types: Custom classes

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

Create action from source to destination

In `nav_graph.xml`:

```
<fragment
    android:id="@+id/fragment_input"
    android:name="com.example.arithmetic.InputFragment">

    <action
        android:id="@+id/action_to_multiplyFragment"
        app:destination="@id/multiplyFragment" />

</fragment>
```



Navigating with actions

In `InputFragment.kt`:

```
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)
    }
}
```

Retrieving Fragment 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}"
    }
}
```

Navigation UI

Menus revisited

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    val navController = findNavController(R.id.nav_host_fragment)
    return item.onNavDestinationSelected(navController) ||
            super.onOptionsItemSelected(item)
}
```

DrawerLayout for navigation drawer

```
<androidx.drawerlayout.widget.DrawerLayout
    android:id="@+id/drawer_layout" ...>

    <fragment
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:id="@+id/nav_host_fragment" ... />

    <com.google.android.material.navigation.NavigationView
        android:id="@+id/nav_view"
        app:menu="@menu/activity_main_drawer" ... />


</androidx.drawerlayout.widget.DrawerLayout>
```

Finish setting up navigation drawer

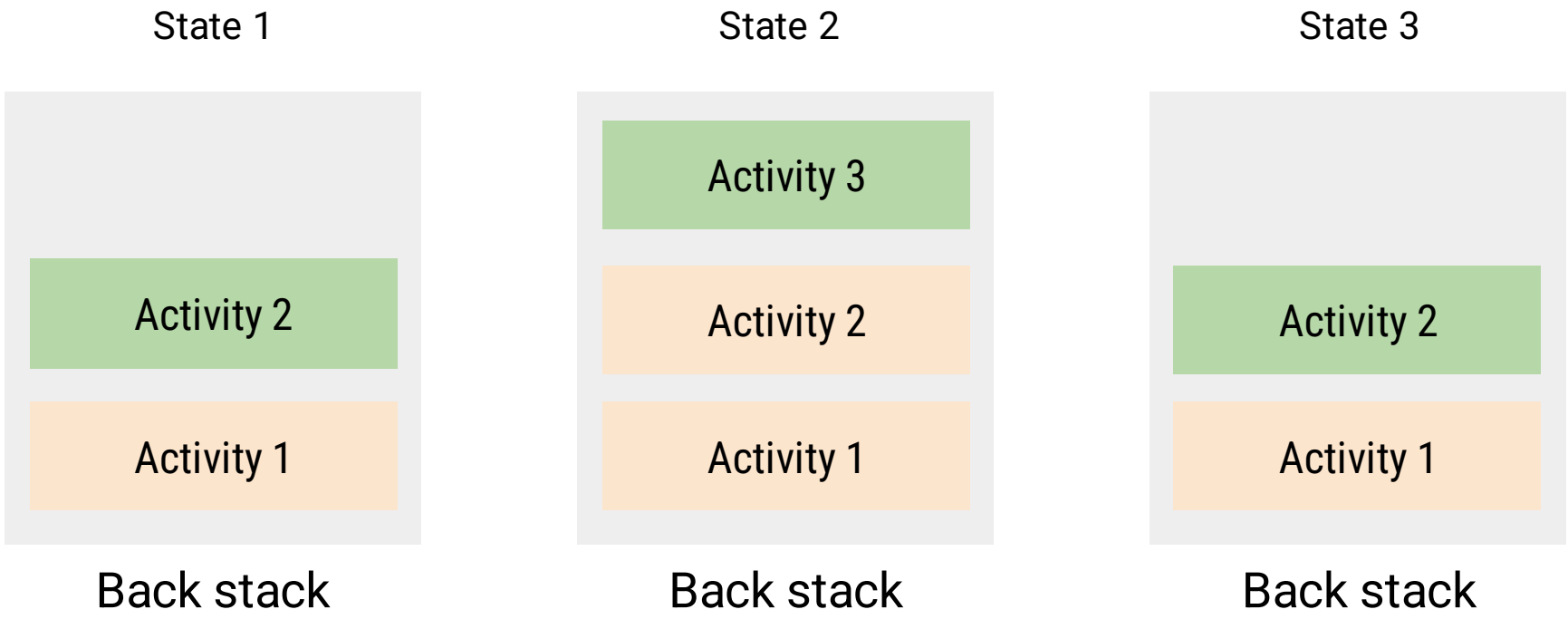
Connect `DrawerLayout` to navigation graph:

```
val appBarConfiguration = AppBarConfig(navController.graph, drawer)
```

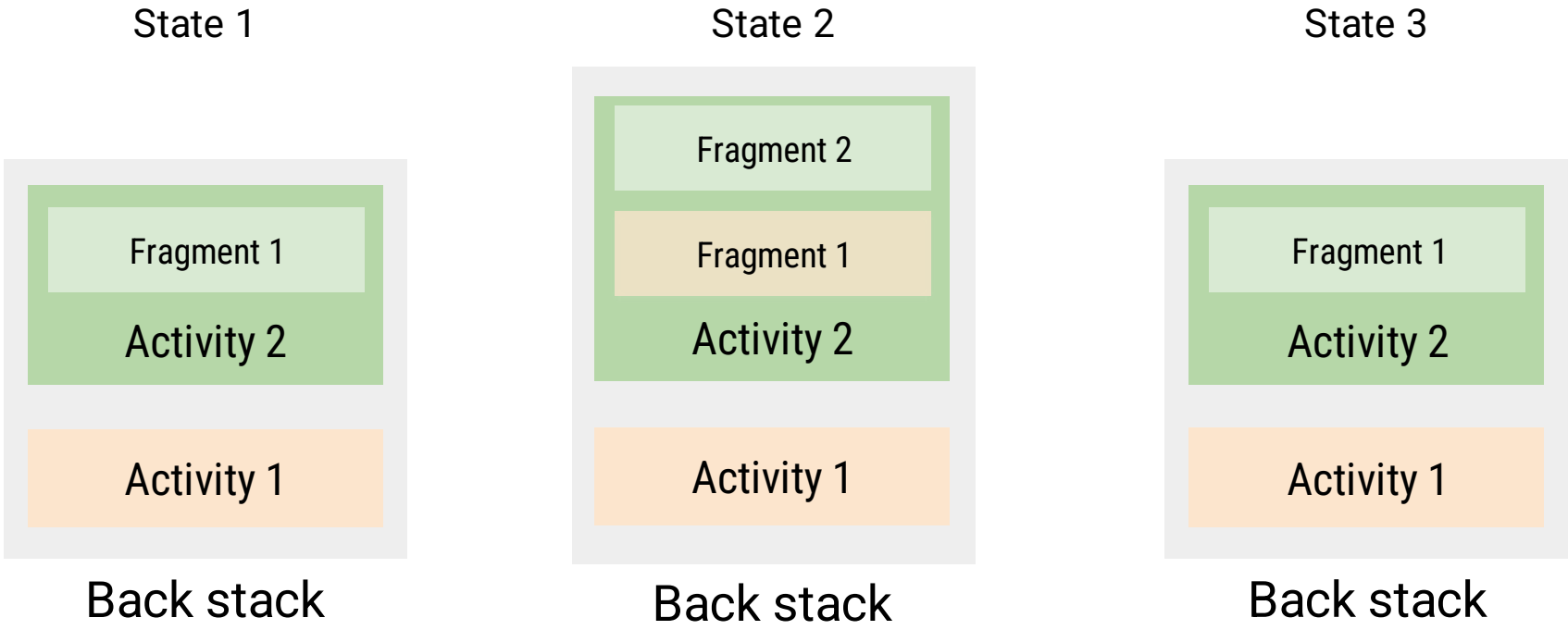
Set up `NavigationView` for use with a `NavController`:

```
val navView = findViewById<NavigationView>(R.id.nav_view)
navView.setupWithNavController(navController)
```

Understanding the back stack



Fragments and the back stack



Summary

Summary

In Lesson 6, you learned how to:

- Use explicit and implicit intents to navigate between Activities
- Structure apps using fragments instead of putting all UI code in the Activity
- Handle navigation with NavGraph, NavHost, and NavController
- Use Safe Args to pass data between fragment destinations
- Use NavigationUI to hook up top app bar, navigation drawer, and bottom navigation
- Android keeps a back stack of all the destinations you've visited, with each new destination being pushed onto the stack.

Learn more

- Principles of navigation
- Navigation component
- Pass data between destinations
- NavigationUI

Pathway

Practice what you've learned by completing the pathway:

Lesson 6: App navigation