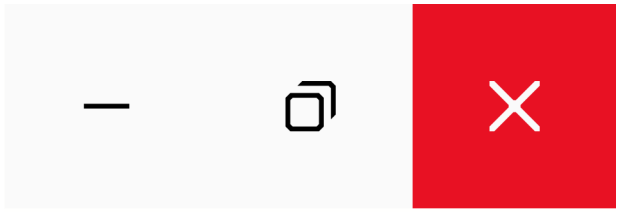


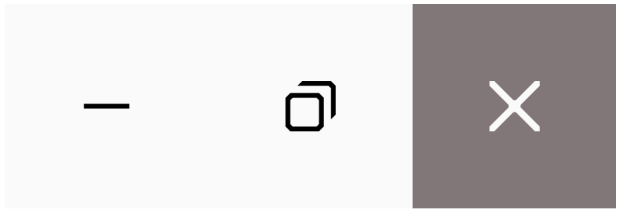
Android 2

App Lifecycle
Fragments

CS 349

July 12

App Lifecycle



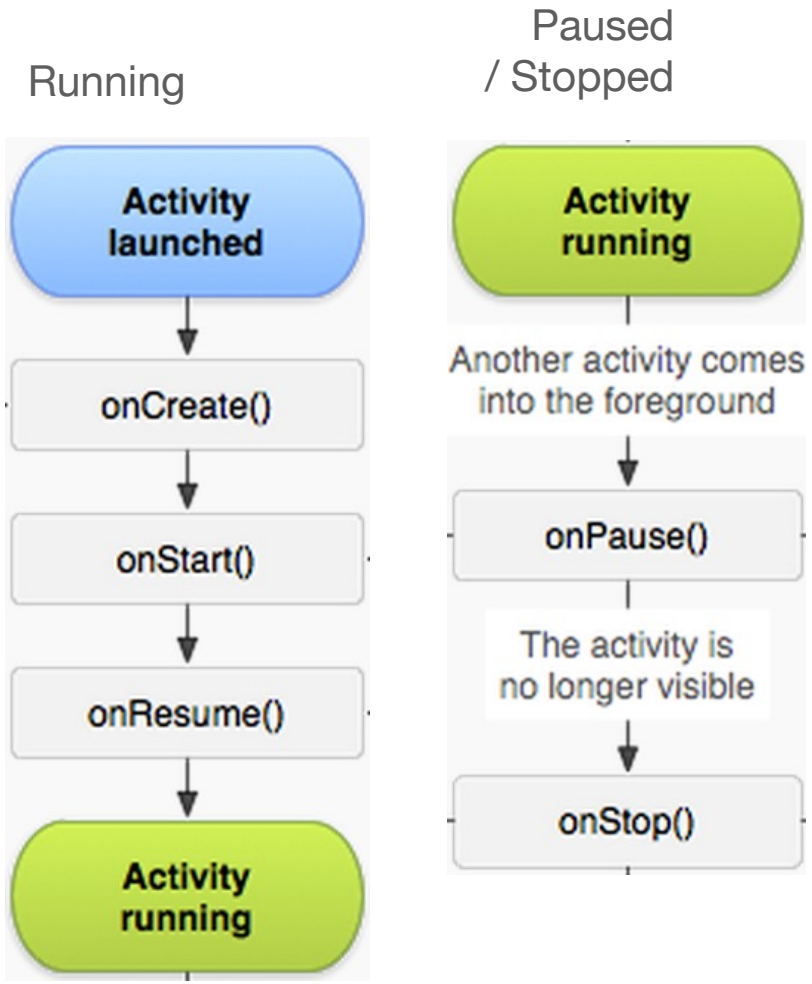
CS 349

Activity Lifecycle

Activities transition through a sequence of lifecycle states.

When a new state is entered, a corresponding callback method is called:

- e.g., `onCreate()` is called when Activity is first launched
- e.g., `onPause()` is called when another Activity comes into the foreground



Activity Destruction and Application State

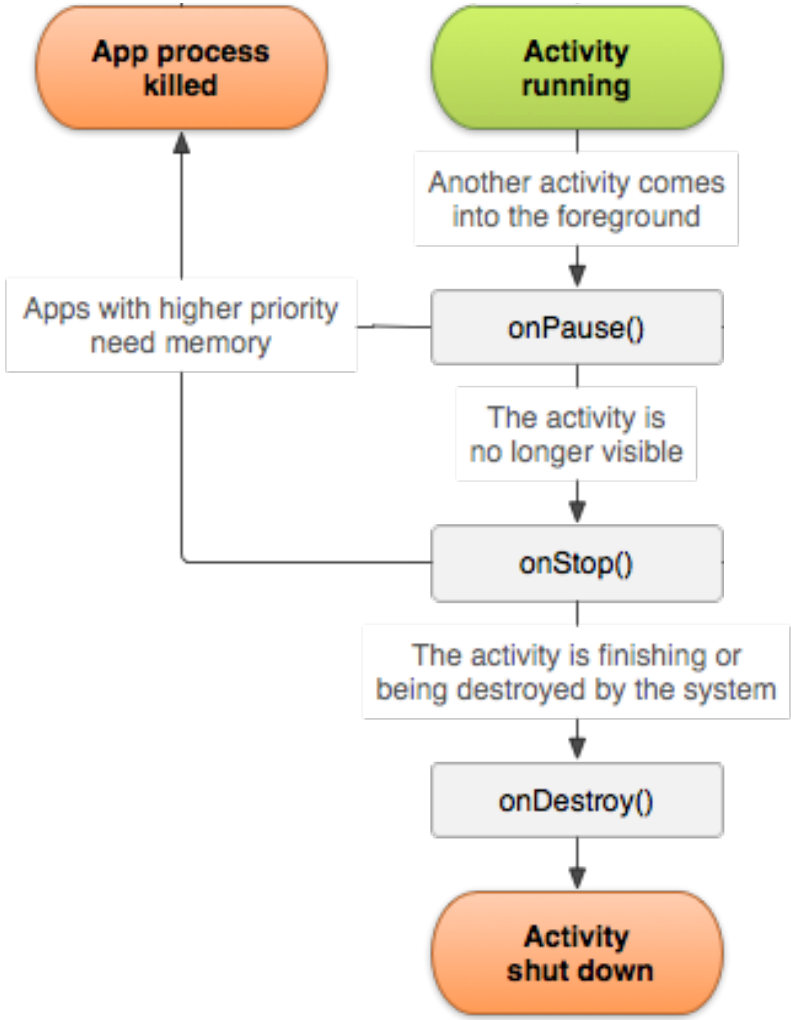
Activity can stop at any time:

- due to user-initiated actions, e.g.,
  - pressing back button
  - “configuration change”, like rotate device
- due to system-initiated actions, e.g.,
  - reclaim resources

Returning to a destroyed Activity causes it to be created again.

If an app’s Activities are destroyed, the system may kill Application state.

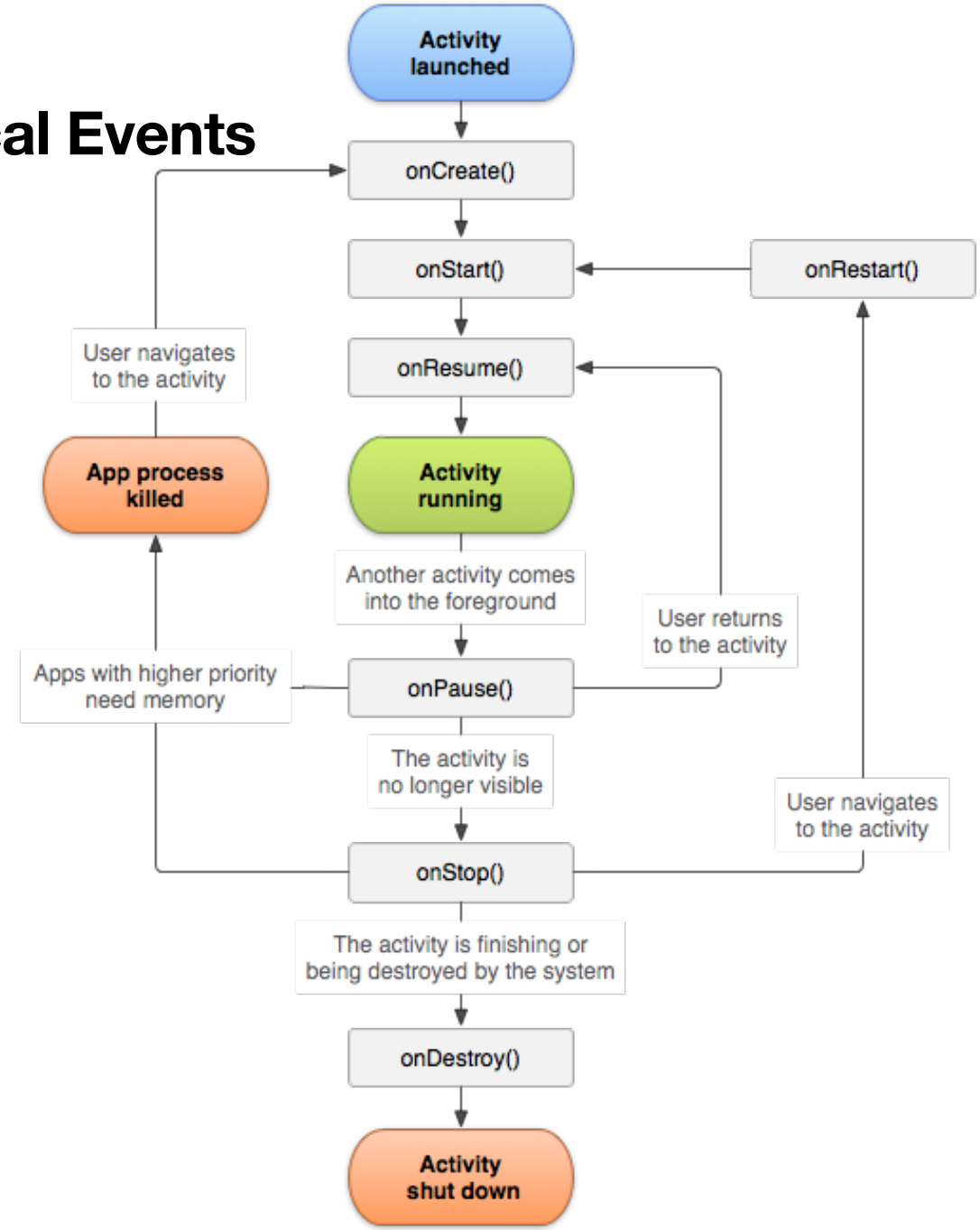
Persisting application state can be challenging.



Activity Lifecycle and Typical Events

Core callback functions:

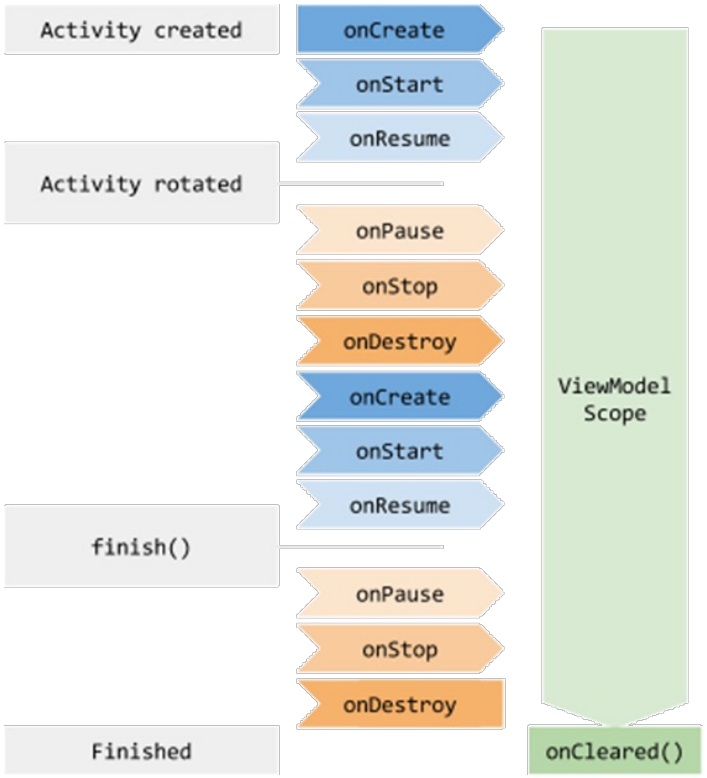
- `onCreate()`: app created or launching
- `onStart()`: becomes visible to user
- `onResume()`: prior to user interaction
- `onPause()`: loses focus or background
- `onStop()`: no longer visible to user
- `onDestroy()`: being recycled and freed



Options for Preserving State

ViewModel

- retains data while user is actively using app
- saved in memory



Saved Instance State

- to recreate activity destroyed by system
- save data to “disk” (using lightweight “bundle”)

Persistent Storage

- retains data as long as app is installed
- save data to “disk” (using heavyweight storage, database, etc.)

Options for Preserving State – MVVM

MyViewModel.kt:

```
class MyViewModel : ViewModel() {

    data class Contact(val name: String, val cell: String)
    private var myData = MutableLiveData<Contact>()

    init {
        // get data from model
        myData.value = Contact("Adrian Reetz", "+1 (800) 555-2368") // faking model
    }

    fun getContact() : LiveData<Contact> {
        return myData
    }

    fun setContact(cell: String) {
        if (myData.value != null) {
            // submit data to model
            Log.i("MVVM", myData.value!!.toString())
            myData.value = Contact("${myData.value!!.name}z" , cell) // faking model
        }
    }
}
```

Options for Preserving State – MVVM

activity_main.xml:

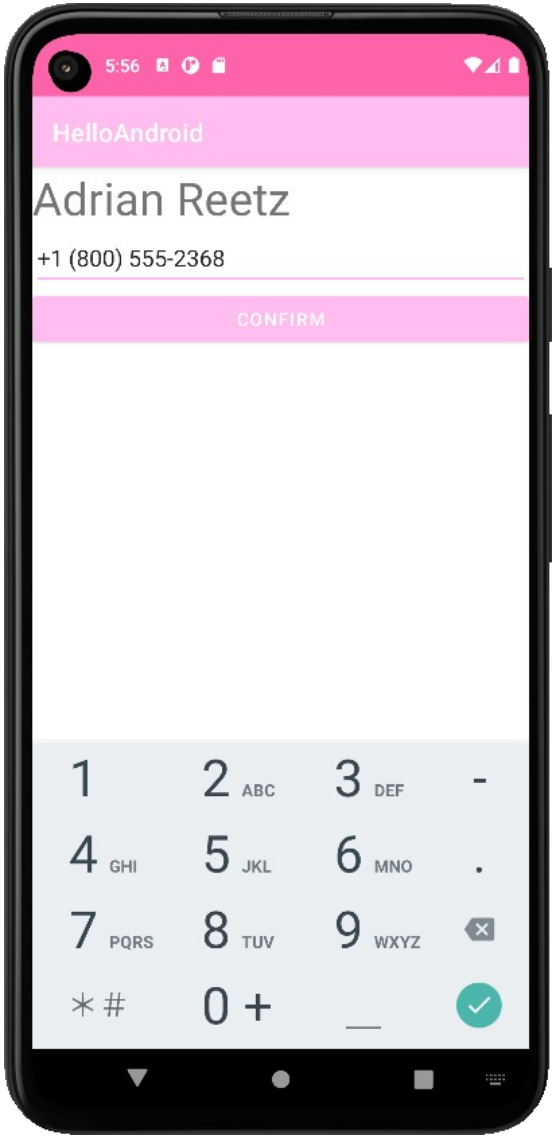
```
<LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

    <TextView android:id="@+id/contact_name"
              android:layout_width="match_parent"
              android:layout_height="@dimen/viewHeight"
              android:autoSizeTextType="uniform" />

    <EditText android:id="@+id/contact_cell"
              android:layout_width="match_parent"
              android:layout_height="@dimen/viewHeight“
              android:inputType="phone" />

    <Button android:id="@+id/action"
            android:layout_width="match_parent"
            android:layout_height="@dimen/viewHeight"
            android:text="Confirm"/>
</LinearLayout>
```

Options for Preserving State – MVVM



MainActivity.kt:

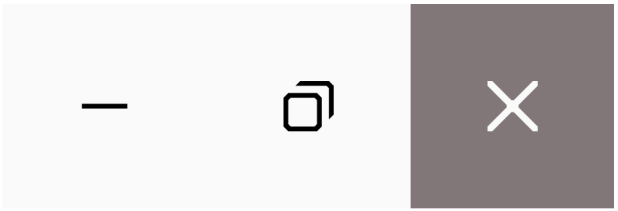
```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    val myVM =
        ViewModelProvider(this)[MyViewModel::class.java]

    myVM.getContact().observe(this) {
        findViewById<TextView>(R.id.contact_name).text =
            it.name
        findViewById<TextView>(R.id.contact_cell).text =
            it.cell
    }

    findViewById<Button>(R.id.action).setOnClickListener {
        myVM.setContact(findViewById<TextView>(R.id.contact_cell).text.toString())
    }
}
```

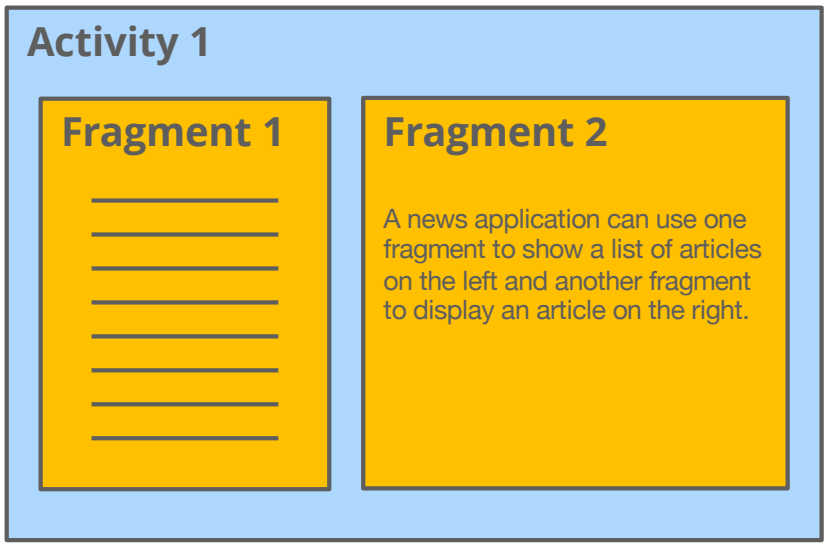
Fragments



Fragments

Fragments are a re-usable portions of a UI. They have their own lifecycle, and their own layout.

An activity can contain (and manage) multiple fragments. Fragments can be used like multiple Activities by swapping out root layout with different Fragments to navigate



Tooling Configuration for Fragments

You might have to manually add these dependencies to `build.gradle`:

```
dependencies {
    ...
    implementation 'androidx.fragment:fragment-ktx:1.5.4'
}
```

Fragment Definition

A Fragment is a layout (`.xml` + `.kt`-file):

fragment_new.xml

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
             xmlns:tools="http://schemas.android.com/tools"
             android:layout_width="match_parent"
             android:layout_height="match_parent"
             tools:context=".BlankFragment">

    <TextView android:text="@string/hello_blank_fragment"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:background="@color/uw1" />

</FrameLayout>
```

NewFragment.kt

```
class NewFragment : Fragment() {

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

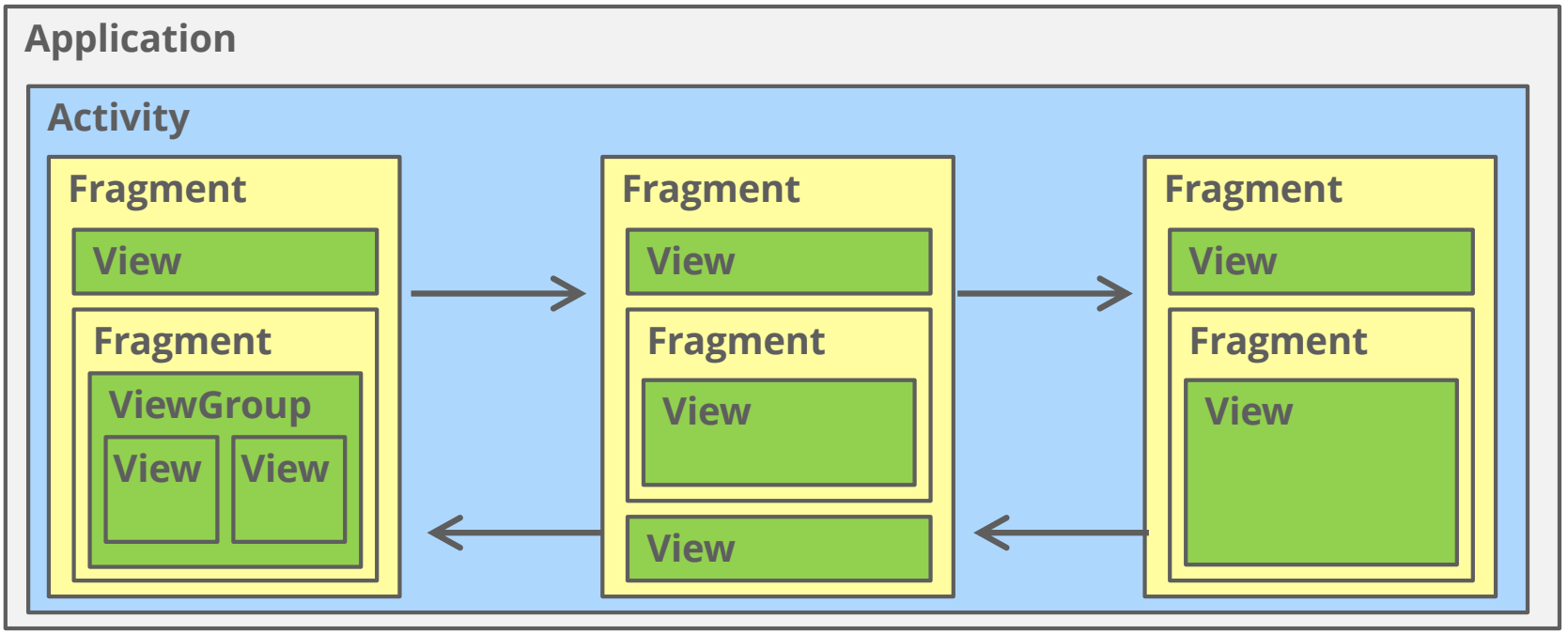
Inserting a Fragment

```
<androidx.fragment.app.FragmentContainerView
    android:id="@+id/fragmentContainerViewBlank"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:name="ui.lectures.helloandroid.BlankFragment"
    ... />

<androidx.fragment.app.FragmentContainerView
    android:id="@+id/fragmentContainerViewOther"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:name="ui.lectures.helloandroid.OtherFragment"
    ... />
```

Multiple Fragment Navigation

A common way to architect Android apps is to navigate between different Fragments hosted in a single Activity. This is called “Single-Activity Architecture”:

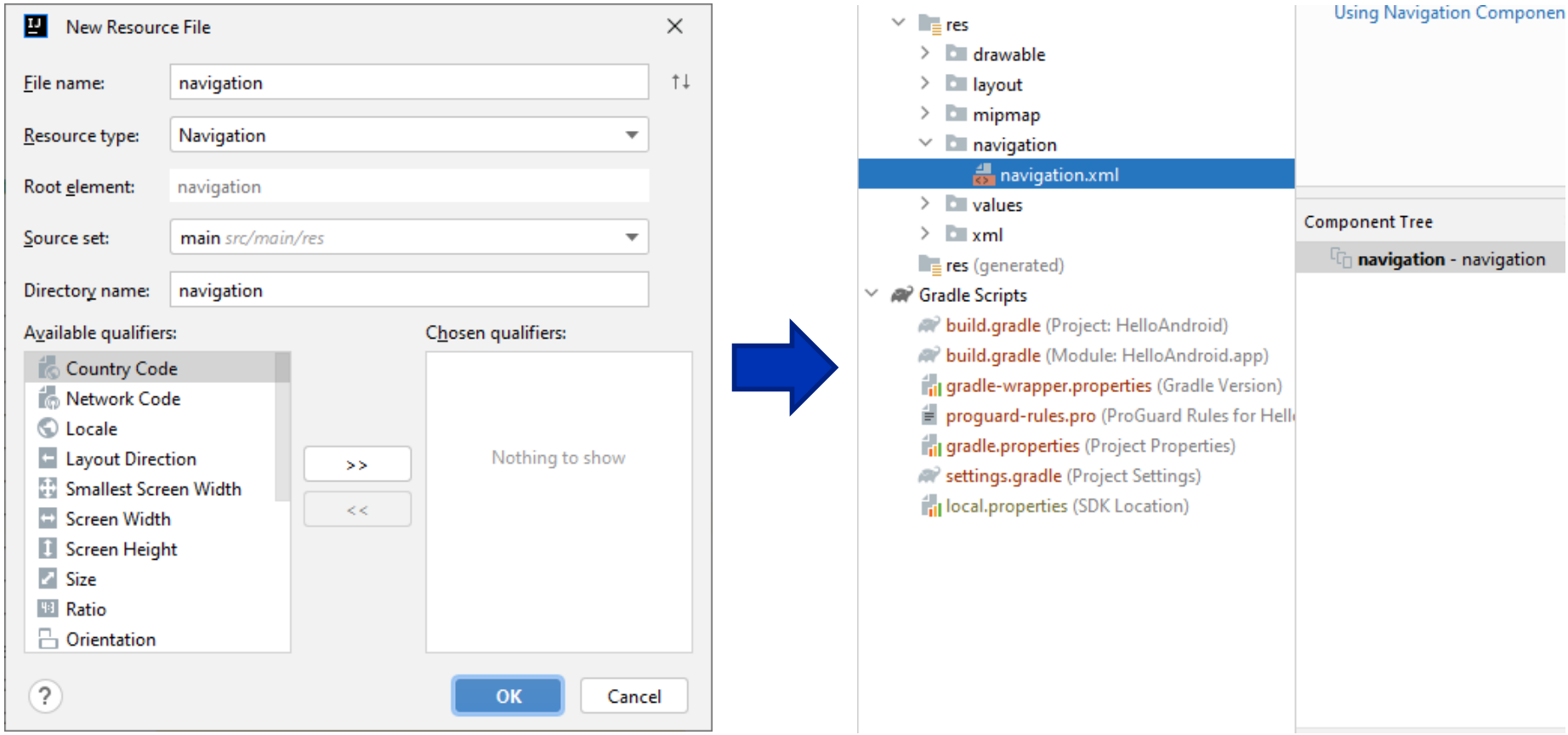


Tooling Configuration for Fragment Navigation

You might have to manually add these dependencies to `build.gradle`:

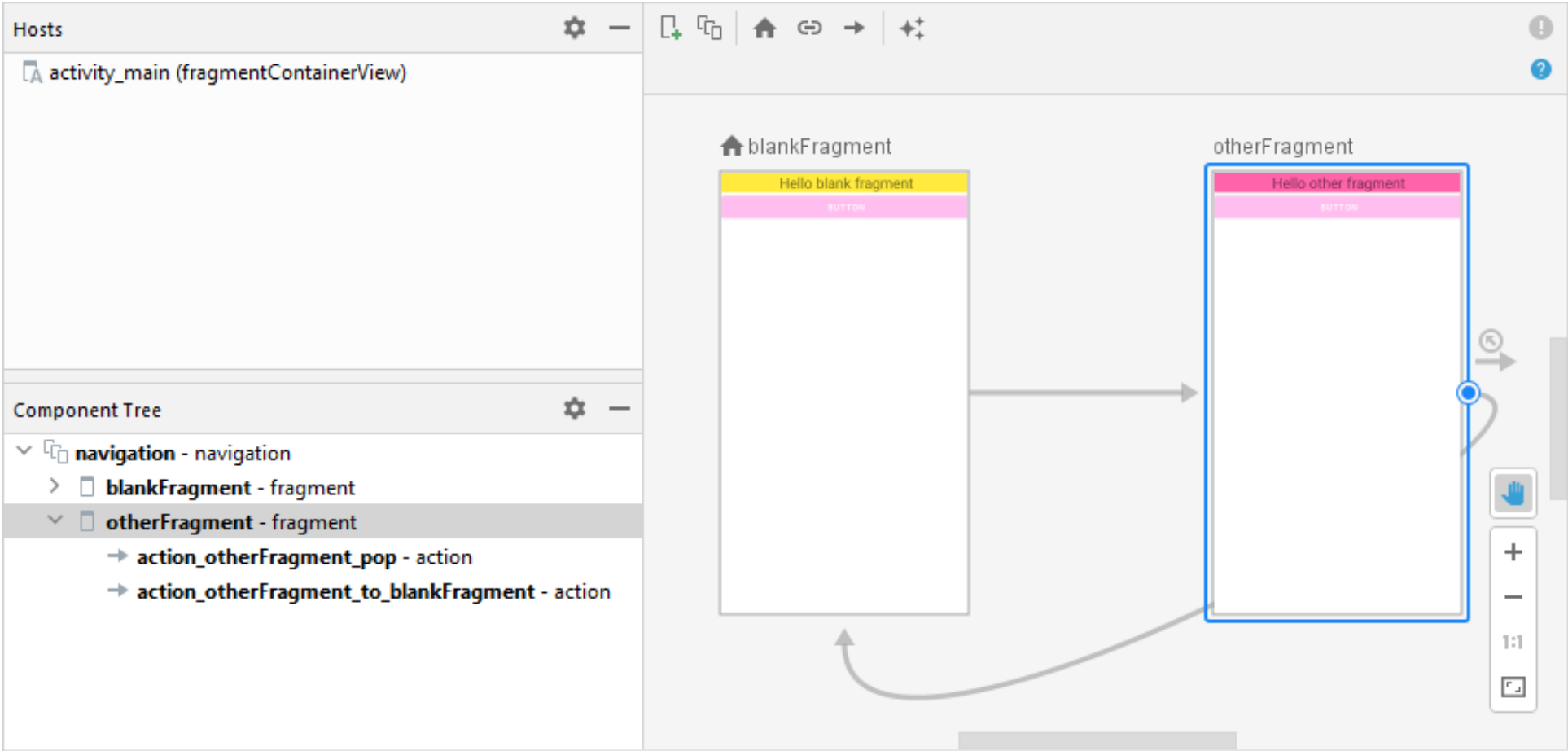
```
dependencies {
    ...
    implementation 'androidx.navigation:navigation-ui-ktx:2.5.3'
    implementation 'androidx.navigation:navigation-fragment-ktx:2.5.3'
}
```

Add `navigation` Resource File



Define Navigation

- right-click to add navigation actions
- drag navigation paths between fragments



Define Navigation

navigation.xml

```
<?xml version="1.0" encoding="utf-8"?>
<navigation android:id="@+id/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"
            app:startDestination="@id/blankFragment">

    <fragment android:id="@+id/blankFragment"
              android:name="ui.lectures.helloandroid.BlankFragment"
              android:label="fragment_blank"
              tools:layout="@layout/fragment_blank">
        <action android:id="@+id/action_blankFragment_to_otherFragment"
                app:destination="@id/otherFragment" />
    </fragment>
    <fragment android:id="@+id/otherFragment"
              android:name="ui.lectures.helloandroid.OtherFragment"
              android:label="fragment_other"
              tools:layout="@layout/fragment_other">
        <action android:id="@+id/action_otherFragment_to_blankFragment"
                app:destination="@id/blankFragment"/>
        <action android:id="@+id/action_otherFragment_pop"
                app:popUpTo="@id/otherFragment"
                app:popUpToInclusive="true"/>
    </fragment>
</navigation>
```

Define Navigation

The property `app:destination` navigates to a new fragment:

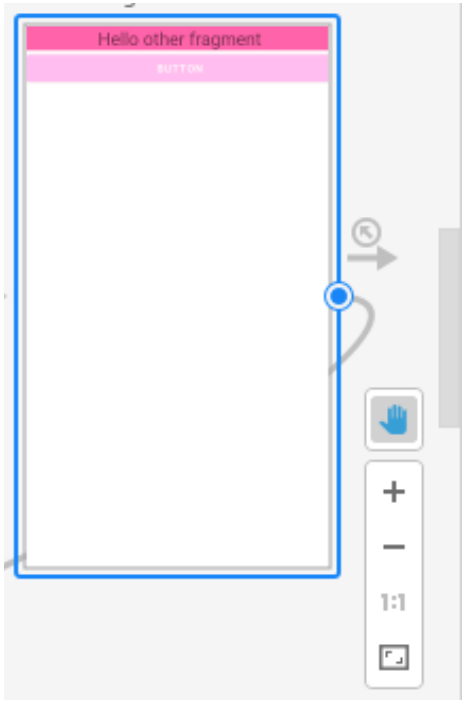
```
<action android:id="@+id/action_otherFragment_to_blankFragment"
        app:destination="@id/blankFragment" />
```

- calling Fragment is saved in back stack
- back button remembers path and will return to calling Fragment

The property `app:popUpTo` navigates to the previous fragment:

```
<action android:id="@+id/action_otherFragment_pop"
        app:popUpTo="@id/otherFragment"
        app:popUpToInclusive="true" />
```

- calling Fragment is "popped off" the back stack
- back button will not return to calling Fragment



Use Navigation

Instead of including a single fragment via `FragmentContainerView`, include a `NavHostFragment`, which will add the necessary fragment container.

activity_main.xml

```
<FrameLayout android:layout_width="match_parent"
android:layout_height="match_parent"
             xmlns:android="http://schemas.android.com/apk/res/android"
             xmlns:app="http://schemas.android.com/apk/res-auto">

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

Use Navigation

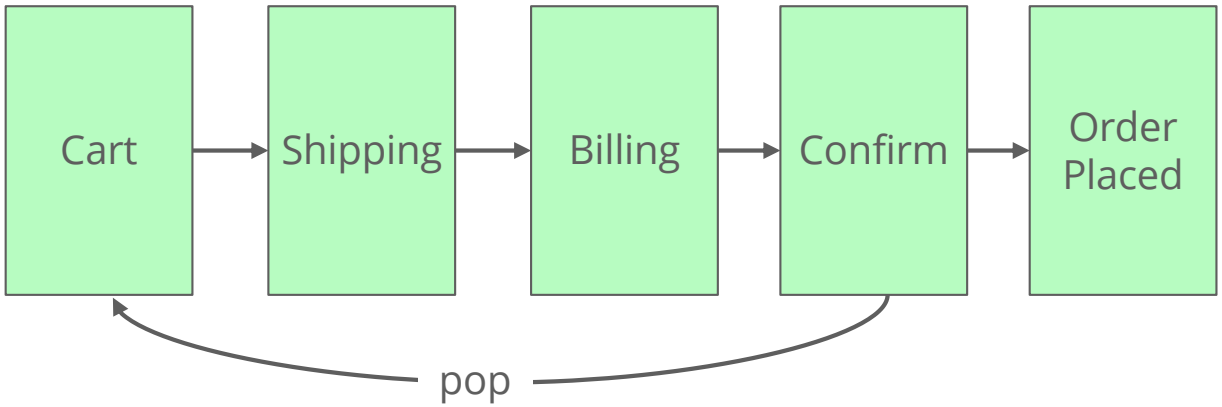
Instead of including a single fragment via `FragmentContainerView`, include a `NavHostFragment`, which will add the necessary fragment container.

BlankFragment.kt

```
override fun onCreateView(..): View? {
    val root = inflater.inflate(R.layout.fragment_main, container, false)
    root.findViewById<Button>(R.id.goToOtherButton).apply {
        setOnClickListener {
            findNavController().navigate(R.id.otherFragment)
        }
    }
    return root
}
```

Navigation and the "Back Stack"

By default, Android's back / up buttons returns to previous Fragment stored on a back stack, which tracks Fragment navigation:



Some app structures need to control back stack behaviour, e.g., cancelling in the middle of an ordering process

Solution: set "Pop Behavior" on actions, e.g., navigate to cart but "pop off" past ordering fragments.

Exchanging Information Between Fragments – MVVM

The ViewModel can be used to transfer information between fragments. The ViewModel belongs to an activity but can be shared amongst fragments.

ViewModel.ks

```
class MyViewModel : ViewModel() {
    // ...
}
```

In an activity (e.g., MainActivity), the ViewModel can be retrieved using:

```
val myVM = ViewModelProvider(this)[MyViewModel::class.java]
```

In a fragment, the ViewModel can be retrieved using:

```
val myVM = ViewModelProvider(requireActivity())[MyViewModel::class.java]
```

Exchanging Information Between Fragments – MVVM

BlankFragment.kt

```
override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                         savedInstanceState: Bundle?): View? {
    val view = inflater.inflate(R.layout.fragment_blank, container, false)

    view.findViewById<Button>(R.id.goToOther).setOnClickListener {
        findNavController().navigate(R.id.action_blankFragment_to_otherFragment)
    }

    val myVM = ViewModelProvider(requireActivity())[MyViewModel::class.java]
    myVM.getContact().observe(viewLifecycleOwner) {
        view.findViewById<TextView>(R.id.viewer).text =
                "${it.name}: ${it.cell}"
    }

    return view
}
```

Exchanging Information Between Fragments – MVVM

OtherFragment.kt

```
override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                         savedInstanceState: Bundle?): View? {
    val view = inflater.inflate(R.layout.fragment_other, container, false)

    val myVM = ViewModelProvider(requireActivity())[MyViewModel::class.java]

    view.findViewById<Button>(R.id.submit).setOnClickListener {
        myVM.setContact(view.findViewById<TextView>(R.id.editor).text.toString())
    }

    myVM.getContact().observe(viewLifecycleOwner) {
        view.findViewById<TextView>(R.id.editor).text = it.cell
    }

    return view
}
```

Exchanging Information Between Fragments – Bundles

Bundles are mappings between String keys and other data:

BlankFragment.kt

```
override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                         savedInstanceState: Bundle?): View? {
    val fragment = inflater.inflate(R.layout.fragment_blank, container, false)

    fragment.findViewById<Button>(R.id.goToOther).setOnClickListener {
        findNavController().navigate(R.id.fragment_other,
                                     Bundle().apply {
                                         putString("key1", "value")
                                         putInt("key2", 0)
                                     })
    return fragment
}
```

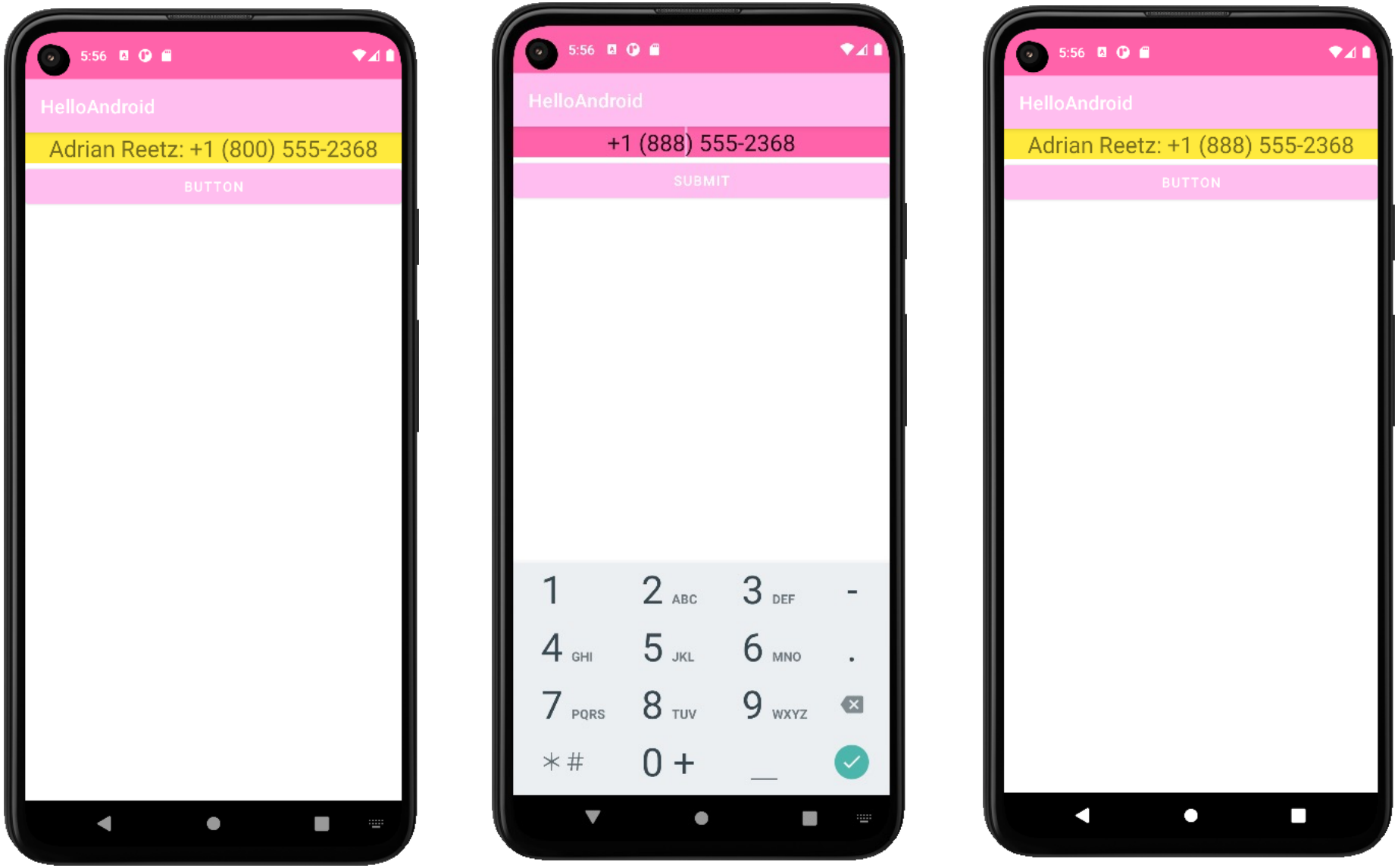
Exchanging Information Between Fragments – Bundles

OtherFragment.kt

```
override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                         savedInstanceState: Bundle?): View? {
    val view = inflater.inflate(R.layout.fragment_other, container, false)
    val key1 = requireArguments().getString("key1") } // String?
    val key2 = requireArguments().getInt("key2") }    // Int

    return view
}
```

Exchanging Information Between Fragments – MVVM



End of Section

- How to implement MVVM in Android.
  - Use `ViewModel`
  - Use `LiveData` / `MutableLiveData`
  - Prevent data mutation from outside the model
- How to navigate between fragments

Any further questions?