Practical class # 13 – Fragments

1) Download the starting code from the course website.

android-basics-kotlin-words-app-activities.zip

2) Create two fragments

1. With app selected in the Project Navigator, add the following fragments (File > New > Fragment > Fragment (Blank)) and both a class and layout file should be generated for each.

- For the first fragment, set the Fragment Name to LetterListFragment. The Fragment Layout Name should populate as fragment_letter_list.
- For the second fragment, set the Fragment Name to WordListFragment. The Fragment Layout Name should populate as fragment_word_list.xml.

2. Delete the boilerplate code from the two kt files. At the end the two files will be:

```
package com.example.wordsapp

import androidx.fragment.app.Fragment

class LetterListFragment : Fragment() {

}
```

```
package com.example.wordsapp

import androidx.fragment.app.Fragment

class WordListFragment : Fragment() {

}
```

3. Copy the contents of activity_main.xml into fragment_letter_list.xml and the contents of activity_detail.xml into fragment_word_list.xml. Update tools:context in fragment_letter_list.xml to .LetterListFragment and tools:context in fragment_word_list.xml to .WordListFragment

```
<?xml version="1.0" encoding="utf-8"?>
<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=".LetterListFragment">
```

```
  <androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:clipToPadding="false"
    android:padding="16dp" />

</FrameLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>
<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=".WordListFragment">

  <androidx.recyclerview.widget.RecyclerView
    android:id="@+id/recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:clipToPadding="false"
    android:padding="16dp"
    tools:listitem="@layout/item_view" />

</FrameLayout>
```

3) Implement LetterListFragment

1. In LetterListFragment.kt, start by getting a reference to the FragmentLetterListBinding, and name the reference _binding.

```
private var _binding: FragmentLetterListBinding? = null
```

2. Create a new property, called binding (without the underscore) and set it equal to _binding!!

```
private val binding get() = _binding!!
```

3. To display the options menu, override onCreate().
   Inside onCreate() call setHasOptionsMenu() passing in true.

```
override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setHasOptionsMenu(true)
}
```

4. Remember that with fragments, the layout is inflated in onCreateView(). Implement onCreateView() by inflating the view, setting the value of _binding, and returning the root view.

```
override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
        ): View? {
                _binding = FragmentLetterListBinding.inflate(inflater, container, false)
                val view = binding.root
                return view
        }
```

5. Below the binding property, create a property for the recycler view.

```
private lateinit var recyclerView: RecyclerView
```

6. Then set the value of the recyclerView property in onViewCreated(), and call chooseLayout() like you did in MainActivity. You'll move the chooseLayout() method into LetterListFragment soon, so don't worry that there's an error.

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        recyclerView = binding.recyclerView
        chooseLayout()
}
```

7. Finally, in onDestroyView(), reset the _binding property to null, as the view no longer exists.

```
override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
}
```

8. The only other thing to note is there are some subtle differences with the onCreateOptionsMenu() method when working with fragments. While the Activity class has a global property called menuInflater, Fragment does not have this property. The menu inflater is instead passed into onCreateOptionsMenu(). Also note that the onCreateOptionsMenu() method used with fragments doesn't require a return statement. Implement the method as shown:

```
override fun onCreateOptionsMenu(menu: Menu, inflater: MenuInflater) {
        inflater.inflate(R.menu.layout_menu, menu)

        val layoutButton = menu.findItem(R.id.action_switch_layout)
        setIcon(layoutButton)
}
```

9. Move the remaining code for chooseLayout(), setIcon(), and onOptionsItemSelected() from MainActivity as-is. The only other difference to note is that, unlike an activity, a fragment is not a Context. You can't pass in this (referring to the fragment object) as the layout manager's context. However, fragments provide a context property you can use instead. The rest of the code is identical to MainActivity.

```
private fun chooseLayout() {
  when (isLinearLayoutManager) {
    true -> {
      recyclerView.layoutManager = LinearLayoutManager(context)
      recyclerView.adapter = LetterAdapter()
    }
    false -> {
      recyclerView.layoutManager = GridLayoutManager(context, 4)
      recyclerView.adapter = LetterAdapter()
    }
  }
}

private fun setIcon(menuItem: MenuItem?) {
  if (menuItem == null)
    return

  menuItem.icon =
    if (isLinearLayoutManager)
      ContextCompat.getDrawable(this.requireContext(), R.drawable.ic_grid_layout)
    else ContextCompat.getDrawable(this.requireContext(), R.drawable.ic_linear_layout)
}

override fun onOptionsItemSelected(item: MenuItem): Boolean {
  return when (item.itemId) {
    R.id.action_switch_layout -> {
      isLinearLayoutManager = !isLinearLayoutManager
      chooseLayout()
      setIcon(item)

      return true
    }

    else -> super.onOptionsItemSelected(item)
  }
}
```

10. Finally, copy over the isLinearLayoutManager property from MainActivity. Put this right below the declaration of the recyclerView property.

```
private var isLinearLayoutManager = true
```

11. Now that all the functionality has been moved to LetterListFragment, all the MainActivity class needs to do is inflate the layout so that the fragment is displayed in the view. Go

ahead and delete everything except onCreate() from MainActivity. After the changes, MainActivity should contain only the following.

```
override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
}
```

4) Create WordListFragment

1. First, copy the companion object to WordListFragment.

```
companion object {
        val LETTER = "letter"
        val SEARCH_PREFIX = "https://www.google.com/search?q="
}
```

2. Then in LetterAdapter, in the onClickListener() where you perform the intent, you need to update the call to putExtra(),
replacing DetailActivity.LETTER with WordListFragment.LETTER.

```
        intent.putExtra(WordListFragment.LETTER, holder.button.text.toString())
```

3. Similarly, in WordAdapter you need to update the onClickListener() where you navigate to the search results for the word,
replacing DetailActivity.SEARCH_PREFIX with WordListFragment.SEARCH_PREFIX.

```
val queryUrl: Uri = Uri.parse("${WordListFragment.SEARCH_PREFIX}${item}")
```

4. Back in WordListFragment, you add a binding variable of type FragmentWordListBinding?.

```
private var _binding: FragmentWordListBinding? = null
```

5. You then create a get-only variable so that you can reference views without having to use ?.

```
private val binding get() = _binding!!
```

6. Then you inflate the layout, assigning the _binding variable and returning the root view. Remember that for fragments you do this in onCreateView(), not onCreate().

```
override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
): View? {
```

```
        _binding = FragmentWordListBinding.inflate(inflater, container, false)
        return binding.root
    }
```

7. Next, you implement onViewCreated(). This is almost identical to configuring the recyclerView in onCreate() in the DetailActivity. However, because fragments don't have direct access to the intent, you need to reference it with activity.intent. You have to do this in onViewCreated() however, as there's no guarantee the activity exists earlier in the lifecycle.

```
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
            val recyclerView = binding.recyclerView
            recyclerView.layoutManager = LinearLayoutManager(requireContext())
    recyclerView.adapter = WordAdapter(activity?.intent?.extras?.getString(LETTER).toString(),
requireContext())

            recyclerView.addItemDecoration(
            DividerItemDecoration(context, DividerItemDecoration.VERTICAL)
            )
    }
```

8. Finally, you can reset the _binding variable in onDestroyView().

```
    override fun onDestroyView() {
            super.onDestroyView()
            _binding = null
    }
```

9. With all this functionality moved into WordListFragment, you can now delete the code from DetailActivity. All that should be left is the onCreate() method.

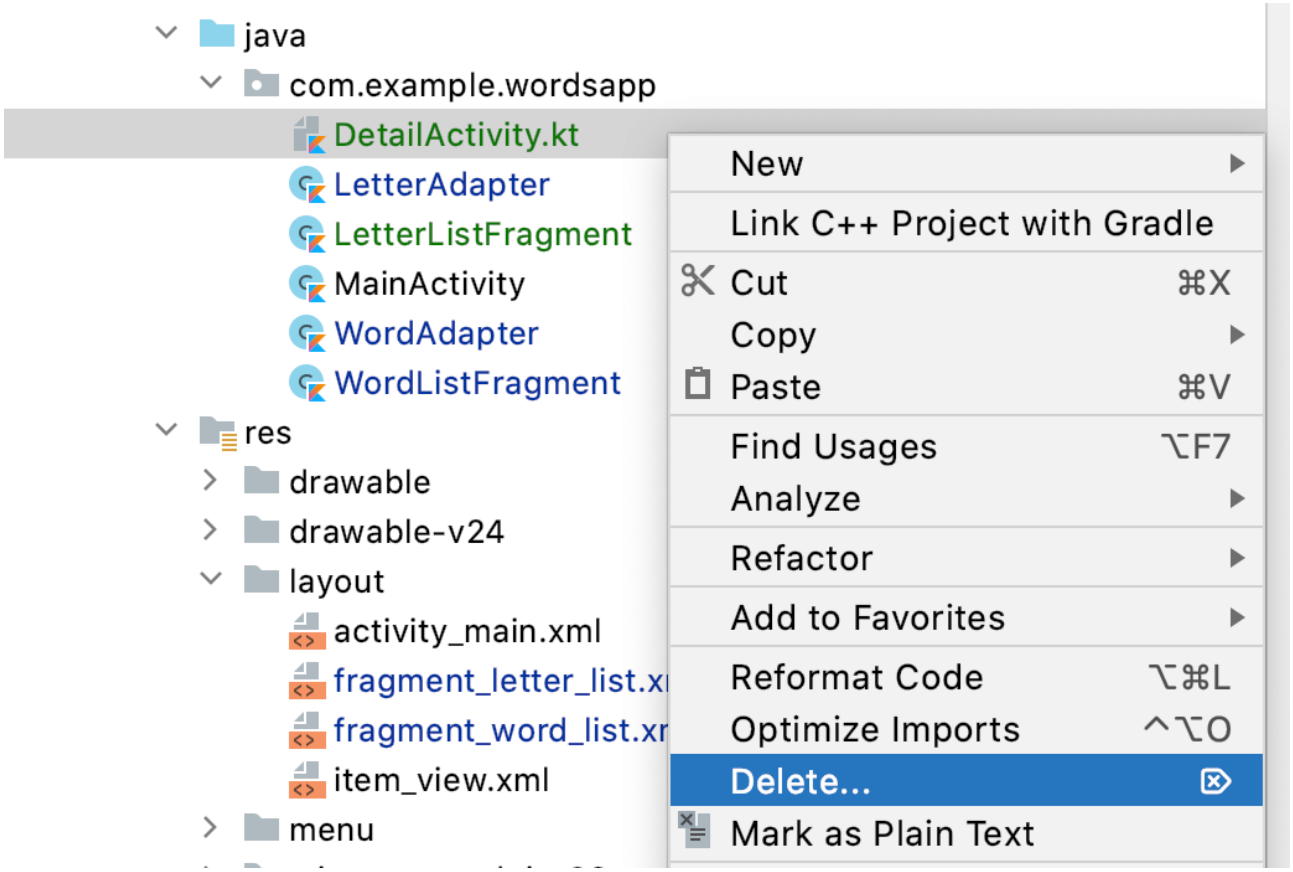
```
    override fun onCreate(savedInstanceState: Bundle?) {
            super.onCreate(savedInstanceState)

            val binding = ActivityDetailBinding.inflate(layoutInflater)
            setContentView(binding.root)
    }
```

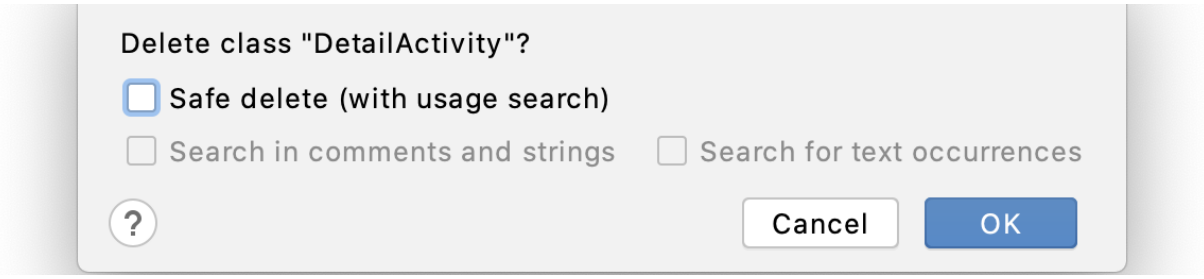
5) Remove DetailActivity

Now that you've successfully migrated the functionality of DetailActivity into WordListFragment, you no longer need DetailActivity. You can go ahead and delete both the DetailActivity.kt and activity_detail.xml as well as make a small change to the manifest.

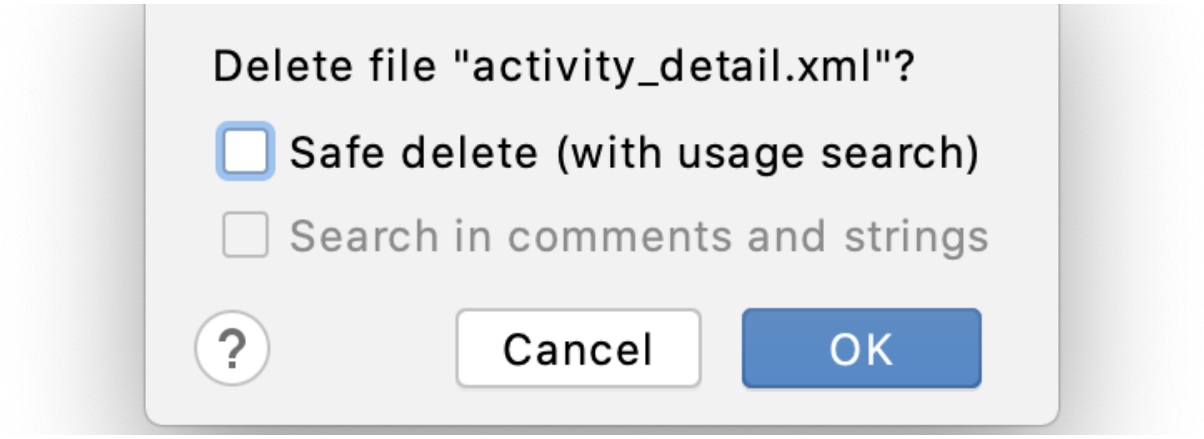
1. First, delete DetailActivity.kt



2. Make sure Safe Delete is Unchecked and click OK.



3. Next, delete activity_detail.xml. Again, make sure Safe Delete is unchecked.



4. Finally, as DetailActivity no longer exists, remove the following from AndroidManifest.xml.

```
<activity
android:name=".DetailActivity"
android:parentActivityName=".MainActivity" />
```

After deleting the detail activity, you're left with two fragments (LetterListFragment and WordListFragment) and a single activity (MainActivity).

6) Enable the navigation component

1. In the project-level build.gradle file, in buildscript > ext, below material_version set the nav_version equal to 2.5.2.

```
buildscript {
        ext {
        appcompat_version = "1.5.1"
        constraintlayout_version = "2.1.4"
        core_ktx_version = "1.9.0"
        kotlin_version = "1.7.10"
        material_version = "1.7.0-alpha2"
        nav_version = "2.5.2"
        }

...
}
```

2. In the app-level build.gradle file, add the following to the dependencies group:

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

7) Enabel Safe Args plugin

1. In the top-level build.gradle file, in buildscript > dependencies, add the following classpath.

```
classpath "androidx.navigation:navigation-safe-args-gradle-plugin:$nav_version"
```

2. In the app-level build.gradle file, within plugins at the top, add androidx.navigation.safeargs.kotlin.

```
plugins {
        id 'com.android.application'
        id 'kotlin-android'
        id 'kotlin-kapt'
        id 'androidx.navigation.safeargs.kotlin'
}
```

3. Once you've edited the Gradle files, you may see a yellow banner at the top asking you to sync the project. Click "Sync Now" and wait a minute or two while Gradle updates your project's dependencies to reflect your changes.

8) Create a navigation graph

Because your layouts are now contained in fragment_letter_list.xml and fragment_word_list.xml, your activity_main.xml file no longer needs to contain the layout for the first screen in your app. Instead, you'll repurpose MainActivity to contain a FragmentContainerView to act as the NavHost for your fragments. From this point forward, all the navigation in the app will take place within the FragmentContainerView.

1. Replace the content of the FrameLayout in activity_main.xml that is androidx.recyclerview.widget.RecyclerView with a FragmentContainerView. Give it an ID of nav_host_fragment and set its height and width to match_parent to fill the entire frame layout.

Replace this:

```
<androidx.recyclerview.widget.RecyclerView
  android:id="@+id/recycler_view"
  ...
  android:padding="16dp" />
```

With this:

```
<androidx.fragment.app.FragmentContainerView
  android:id="@+id/nav_host_fragment"
  android:layout_width="match_parent"
  android:layout_height="match_parent" />
```

2. Below the id attribute, add a name attribute and set it to androidx.navigation.fragment.NavHostFragment. While you can specify a specific fragment for this attribute, setting it to NavHostFragment allows your FragmentContainerView to navigate between fragments.

```
android:name="androidx.navigation.fragment.NavHostFragment"
```

3. Below the layout_height and layout_width attributes, add an attribute called app:defaultNavHost and set it equal to "true". This allows the fragment container to interact with the navigation hierarchy. For example, if the system back button is pressed, then the container will navigate back to the previously shown fragment, just like what happens when a new activity is presented.

```
app:defaultNavHost="true"
```

4. Add an attribute called app:navGraph and set it equal to "@navigation/nav_graph". This points to an XML file that defines how your app's fragments can navigate to one another. For now, the Android studio will show you an unresolved symbol error. You will address this in the next task.

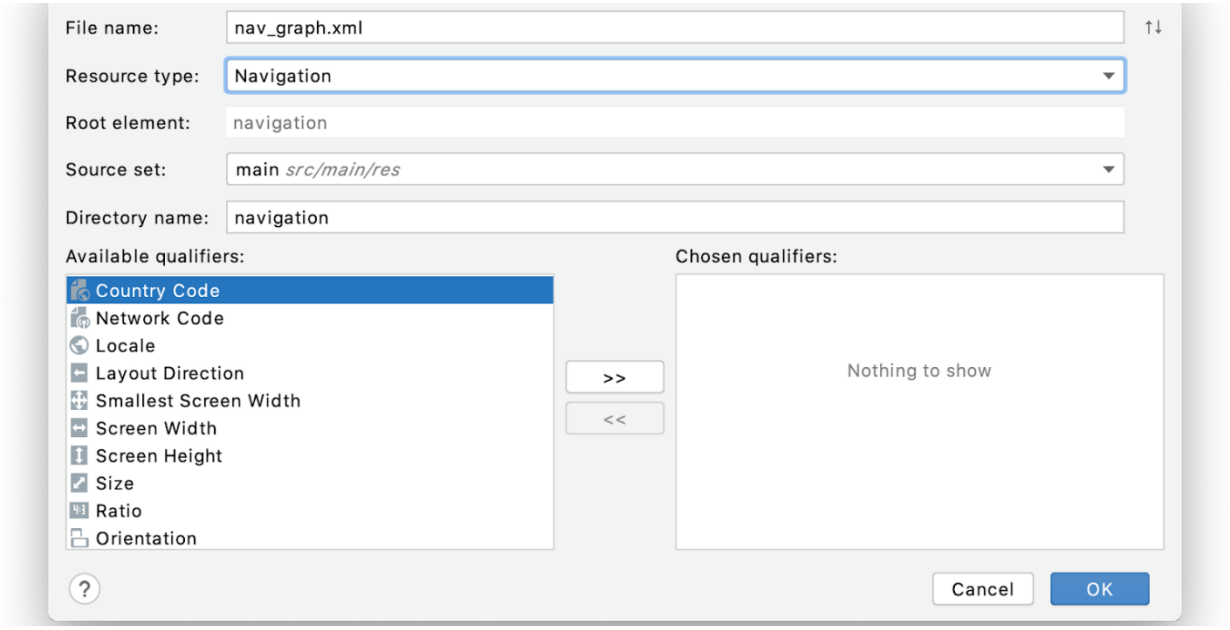
```
app:navGraph="@navigation/nav_graph"
```

5. Finally, because you added two attributes with the app namespace, be sure to add the xmlns:app attribute to the FrameLayout.

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

6. Add a navigation graph file (File > New > Android Resource File) and filling the fields as follows.

- File name: nav_graph.xml. This is the same as the name you set for the app:navGraph attribute.
- Resource type: Navigation. The Directory name should then automatically change to navigation. A new resource folder called "navigation" will be created.



Upon creating the XML file, you're presented with a new visual editor. Because you've already referenced nav_graph in the FragmentContainerView's navGraph property, to add a new destination, click the new button in the top left of the screen and create a destination for each fragment (one for fragment_letter_list and one for fragment_word_list).

To create a navigation action between the letterListFragment to
the wordListFragment destinations, hover your mouse over the letterListFragment destination and drag from the circle that appears on the right onto the wordListFragment destination.

You should now see an arrow has been created to represent the action between the two destinations. Click on the arrow, and you can see in the attributes pane that this action has a name action_letterListFragment_to_wordListFragment that can be referenced in code.

Select the wordListFragment destination and in the attributes pane, under **Arguments**, click the plus button to create a new argument.

The argument should be called letter and the type should be String. This is where the Safe Args plugin you added earlier comes in. Specifying this argument as a string ensures that a String will be expected when your navigation action is performed in code.

On the NavGraph, you need to set the letter list as a start destination.

Set the start destination by selecting letterListFragment and clicking the **Assign start destination** button.

That's all you need to do with the NavGraph editor for now. At this point, go ahead and build the project. In Android Studio select Build > Rebuild Project from the menu bar. This will generate some code based on your navigation graph so that you can use the navigation action you just created.

9)Perform the navigation Action

Open up LetterAdapter.kt to perform the navigation action. This only requires two steps.

1. Delete the contents of the button's setOnClickListener(). Instead, you need to retrieve the navigation action you just created. Add the following to the setOnClickListener().

```
val action = LetterListFragmentDirections.actionLetterListFragmentToWordListFragment(letter =
holder.button.text.toString())

holder.view.findNavController().navigate(action)
```

10) Configure MainActivity

The final piece of setup is in MainActivity. There are just a few changes needed in MainActivity to get everything working.

1. Create a navController property. This is marked as lateinit since it will be set in onCreate.

```
private lateinit var navController: NavController
```

2. Then, after the call to setContentView() in onCreate(), get a reference to the nav_host_fragment (this is the ID of your FragmentContainerView) and assign it to your navController property.

```
val navHostFragment = supportFragmentManager.findFragmentById(R.id.nav_host_fragment) as
NavHostFragment
navController = navHostFragment.navController
```

3. Then in onCreate(), call setupActionBarWithNavController(), passing in navController. This ensures action bar (app bar) buttons, like the menu option in LetterListFragment are visible.

```
setupActionBarWithNavController(navController)
```

4. Finally, implement onSupportNavigateUp(). Along with setting defaultNavHost to true in the XML, this method allows you to handle the up button. However, your activity needs to provide the implementation.

```
override fun onSupportNavigateUp(): Boolean {
  return navController.navigateUp() || super.onSupportNavigateUp()
}
```

11. Getting arguments

1. In WordListFragment, create a letterId property. You can mark this as lateinit so that you don't have to make it nullable.

```
private lateinit var letterId: String
```

2. Then override onCreate() (not onCreateView() or onViewCreated()!), add the following:

```
override fun onCreate(savedInstanceState: Bundle?) {
   super.onCreate(savedInstanceState)

   arguments?.let {
      letterId = it.getString(LETTER).toString()
   }
}
```

3. Because it's possible for arguments to be optional, notice you call let() and pass in a lambda. This code will execute assuming arguments is not null, passing in the non null arguments for the it parameter. If arguments is null, however, the lambda will not execute.

```
arguments?.let { it: Bundle
    letterId = it.getString(LETTER).toString()
}
```

3. Finally, you can access the letterId when you set the recycler view's adapter. Replace activity?.intent?.extras?.getString(LETTER).toString() in onViewCreated() with letterId.

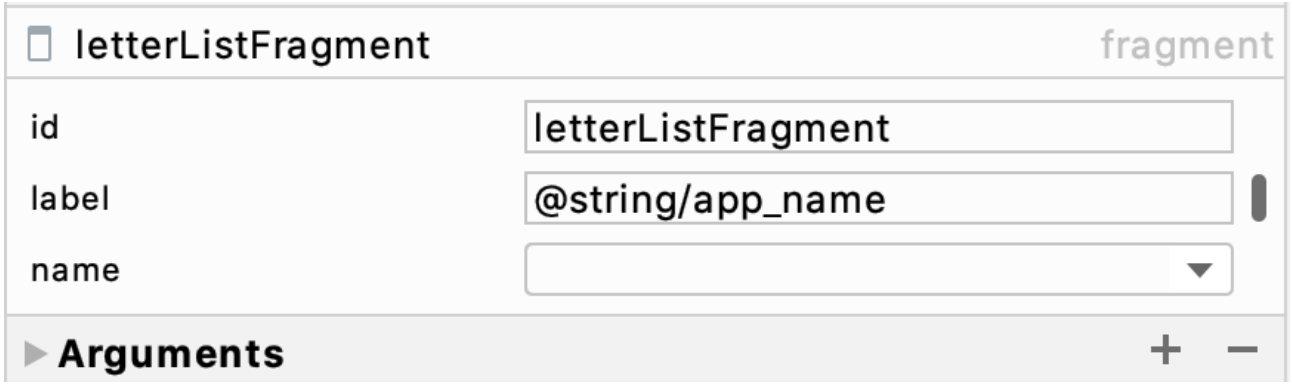
```
recyclerView.adapter = WordAdapter(letterId, requireContext())
```

12. Update Fragments labels

1. In strings.xml, after the app name, add the following constant.

```
<string name="word_list_fragment_label">Words That Start With {letter}</string>
```

2. You can set the label for each fragment on the navigation graph. Go back into nav_graph.xml and select letterListFragment in the component tree, and in the attributes pane, set the label to the app_name string:



3. Select wordListFragment and set the label to word_list_fragment_label: