Android: Session Three

Part 1: Fragments

A Fragment is a piece of application user interface or behavior that is placed in an Activity.

Fragments are like mini-activities actually, so in this kind of project architecture, our application will be made of only one activity acting like a main frame with screens, or pages being fragments.

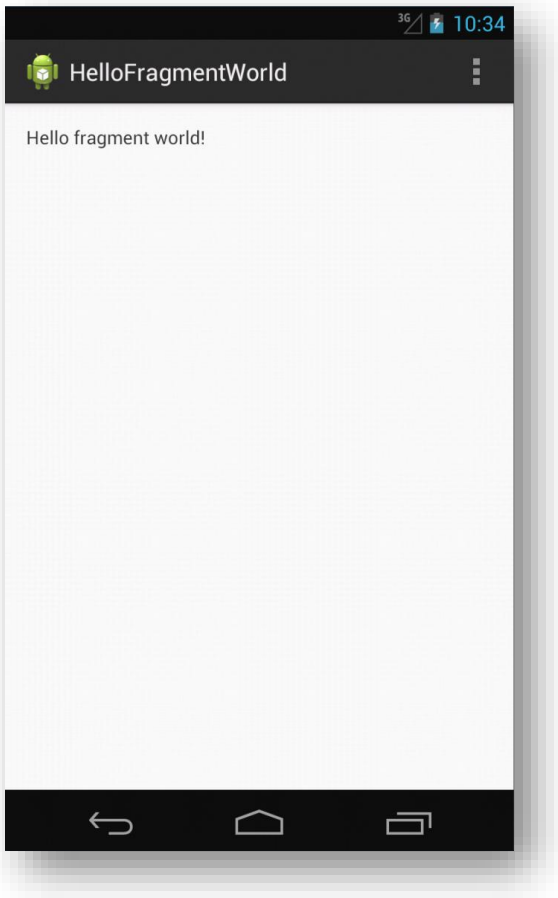
What we are going to do first is to use a fragment to display the main content of an activity, instead of having the content directly in the activity.

Step 1: Create a basic project

We start by creating a new simple Android project called HelloFragmentWorld.

To do:

- Create the project so that your app looks like this:



Step 2: Change your activity layout

When working with fragments, activities are used more like structures, or fragment containers.

To do:

Change your main activity layout. Transform it into a simple fragment holder:

- You can use a ConstraintLayout but most of the time we will use a simple FrameLayout
- Make it take the whole space on the screen
- We are going to retrieve this layout as a groupview later on so do not forget to give it an id. Make this id meaningful, like “main_container”, or something like that.

Step 3: Create the first fragment

Technically, a fragment is a class that extends the `Fragment` class. It has to be attached to an activity.

Note: Like the AlertDialog, the Fragment class is available in more than one package, *android.app* and *android.support.v4.app*. Both do (almost exactly) the same thing but what is important is to be consistent once you picked one since app fragments and support fragments (and their related classes) cannot be mixed.
In this lesson, and in this document, we use support fragments.

The most important part here is the onCreateView() callback method. It is called when the fragment is about to be drawn on the screen and it must return a view : the fragment view!

Here is an example of a Fragment class:

```
class HelloFragment : Fragment() {

    // onCreateView : called every time the fragment needs to draw its UI
    // Must return the inflated fragment layout view
    // params :
    // inflater : a ready to use inflater. Use it to inflate fragment layout
    // container : the viewgroup that will hold the fragment
    // savedInstanceState : fragment saved state information if it has
    //                      already been saved, else null
    override fun onCreateView(
            inflater: LayoutInflater?,
            container: ViewGroup?,
            savedInstanceState: Bundle?): View? {

        // Inflate the layout for this fragment
        val rootView = inflater!!.inflate(
                                        R.layout.fragment_hello,
                                        container,
                                        false)

        // Return layout view
        return rootView
    }
```

To do:

- Create a new layout resource for your fragment. The goal is to have a same layout you had in the activity, i.e. a screen-whole ConstraintLayout and some Hello world textview.
- Create the HelloFragmentWorld fragment. Use the *New -> Kotlin Class* entry on your app java package contextual menu. Make it inflate the layout you just created.

Step 4: Put the fragment in the activity

We now have a container activity, and a standalone fragment. Time to mix them together!

Managing fragments is done by using the Fragment Manager, which enables us to act on fragments.
All actions on fragments (add, remove, replace…) are packed in FragmentTransaction objects, and to execute the actions in a Fragment Transaction, it must be committed.

To retrieve an instance of the Fragment Manager, just use the ***supportFragmentManager*** available in your activity.

Fragment Transaction is an API for performing a set of Fragment operations, like add a fragment or remove one.
Once you have a Fragment Manager instance, use its ***beginTransaction()*** method to get a brand new Fragment Transaction object.

Fragment Transactions also have their share of methods, but we are going to use mostly ***replace()*** and ***commit()***.

```
// Use the Fragment manager to create a Fragment transaction
val fragmentTransaction = supportFragmentManager.beginTransaction()

// Create a new fragment instance
val helloFragment = HelloFragment()

// Do fragment actions in the transaction
// Remove an old fragment
fragmentTransaction.remove(oldFragment)
// Add a new one
fragmentTransaction.add(R.id.main_container, helloFragment)
// Or do both operations in one step
fragmentTransaction.replace(R.id.main_container, helloFragment)

// Commit transaction
fragmentTransaction.commit()
```

With all this information, you should be able to make main activity display the HelloFragmentWorld

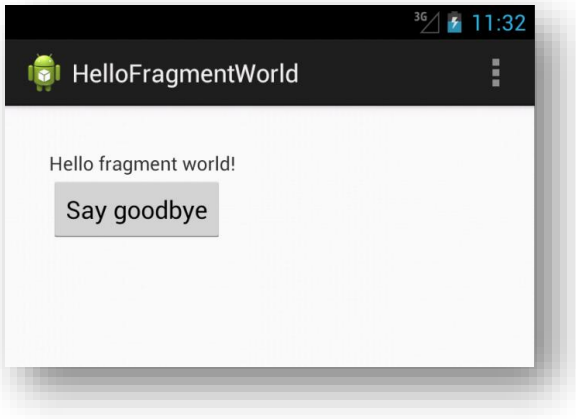
fragment when your application starts

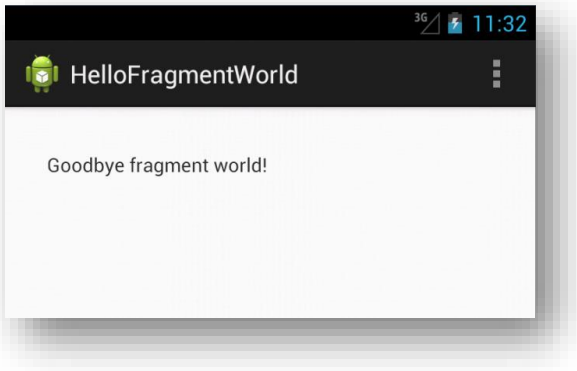
To do:

- Complete project to display HelloWorld fragment

Step 5: Create a second fragment

Now out application will contain two screens, and we navigate from the first page to the second page by clicking a button.





In activities, most UI binding is done in the the ***onCreate()*** callback method, after the ***setContentView()*** is done. In fragments, things are a little bit different, as fragment view is not available before it is returned by ***onCreateView()*** callback.

This is why binding is done after kotlinx can do its thing : in the ***onViewCreated()*** callback

```
override fun onViewCreated(rootView: View?, savedInstanceState: Bundle?) {
    fragment_hello_btn_toast.setOnClickListener {
        Toast.makeText(activity, "Clicked ;-)", Toast.LENGTH_SHORT).show()
    }
}
```

To do:

- Create a second fragment
- Create its layout file
- Implement navigation from the first fragment to the second one

Step 6: Transmit information between fragments

To transmit information from one fragment to the other, we must store the information in a Bundle object and set it as arguments when creating the new fragment.

```
// Create a Bundle to hold the information
val dataBundle = Bundle()

// Put some information inside the bundle
dataBundle.putString("message", "I am the message")

// Create the fragment we want to display
val secondFragment = SecondFragment()

// Send the bundle to the fragment
secondFragment.arguments = dataBundle

// fragmentManager stuff to display second fragment...
```

Step 7: The activity is the conductor

Since it is designed as a re-usable separate module, a fragment should not replace itself in the main layout. This is the responsibility of the activity. Fragments should only handle what happens inside their own view, and when necessary, delegate to the activity.

In order to do it right, instead of acting on the layout, fragments declare interfaces that describe their interactions to the outside world. Then, every activity including a fragment should implement its interaction interface.

Now, we are going to replace the fragment switch we did in step 5 using the activity as an interaction manager.

To do:

- In HelloWorld fragment, declare a public interface describing what actions can be done in the fragment.

```
interface HelloFragmentInteractionListener {
    fun onMyButtonWasClicked()
}
```

- Implement this interface in your main activity.

```
class MainActivity : AppCompatActivity(), HelloFragmentInteractionListener {

    override fun onMyButtonWasClicked() {
        // Do whatever has to be done when button was clicked
    }

    …
}
```

- In HelloWorld fragment, when clicking on the Goto Goodbye button, trigger the appropriate action in the activity implementing the fragment interface.

```
fragment_hello_btn_toast.setOnClickListener {
    (activity as MainActivity).onMyButtonWasClicked()
}
```

Part 2: The ViewPager

Fragments, like bowties, are cool. But to show you a practical example, let us see the ViewPager component.

The ViewPager is a common Android Widget that allows to slide horizontally between multiple screens in an application.

There are three steps to make a ViewPager work:

Each screen of the application must be a Fragment

1. Each screen must be made of a separate fragment class
2. Add a ViewPager widget to activity layout

```
<android.support.v4.view.ViewPager
    android:id="@+id/main_viewpager"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

3. Populate the widget with the desired fragment instances.
   To do that, we attach to the ViewPager widget a component that can associate a position in the screen list, with the desired fragment. This component is called a pager adapter and extends from the FragmentPagerAdapter class.

```
class CustomViewpagerAdapter(fm: FragmentManager) : FragmentPagerAdapter(fm) {
    private val nbPages = 2

    override fun getItem(position: Int): Fragment {
        return when (position) {
            1 -> SecondFragment()
            else -> HelloFragment()
        }
    }

    override fun getCount(): Int {
        return nbPages
    }
}
```

4. Use the ***adapter*** property to associate the adapter to the viewpager.

Add tabs navigation

ViewPager works as is, but you can add a navigation bar, like tabs, to increase app useability.

This is a two steps process:

1. Add a PagerTabStrip (or PagerTitleStrip) inside the ViewPager layout

```
<android.support.v4.view.PagerTabStrip
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="top"
    android:background="@android:color/holo_green_light"
    android:textColor="@android:color/holo_red_light"/>
```

2. Override the ***getPageTitle(position)*** method of your ViewPager adapter. This method should give the title of the screen at the specified position.