Mobile Application Development

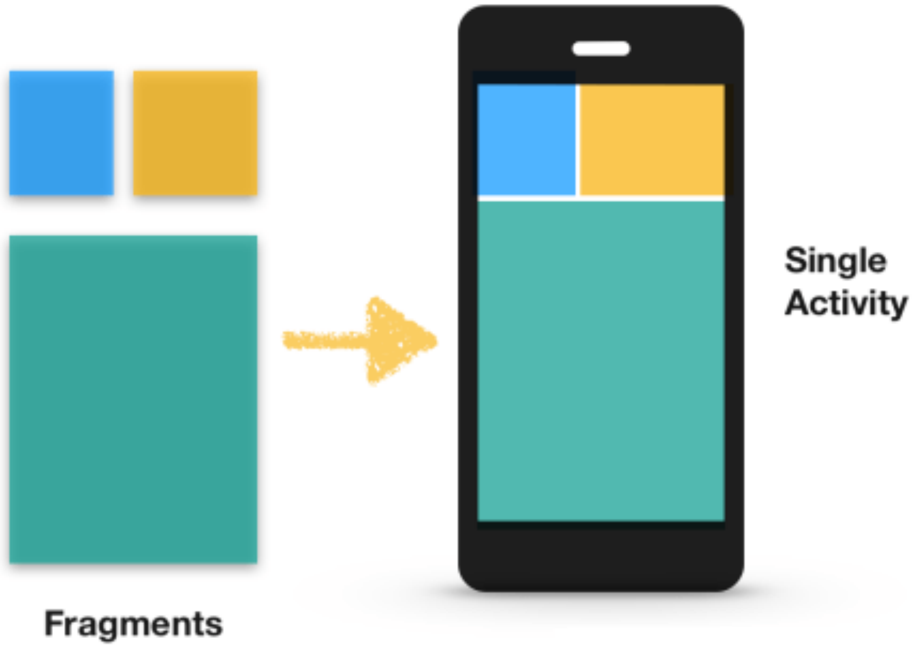
Produced by

David Drohan (ddrohan@wit.ie)

Department of Computing & Mathematics
Waterford Institute of Technology
http://www.wit.ie

Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Introduction to Fragments



Agenda

- ❑ Recap on Activities
- ❑ Introduction to Fragments
- ❑ The Fragment Lifecycle
- ❑ Managing Fragments
- ❑ Conclusion

Activities Recap

- ❑ An activity is a single, focused thing that the user can do.
- ❑ Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with **`setContentView(View)`**.
- ❑ While activities are often presented to the user as full-screen windows, they can also be used in other ways (as floating windows, in Multi-Window mode or embedded into other windows.
- ❑ There are two methods almost all subclasses of Activity will implement:

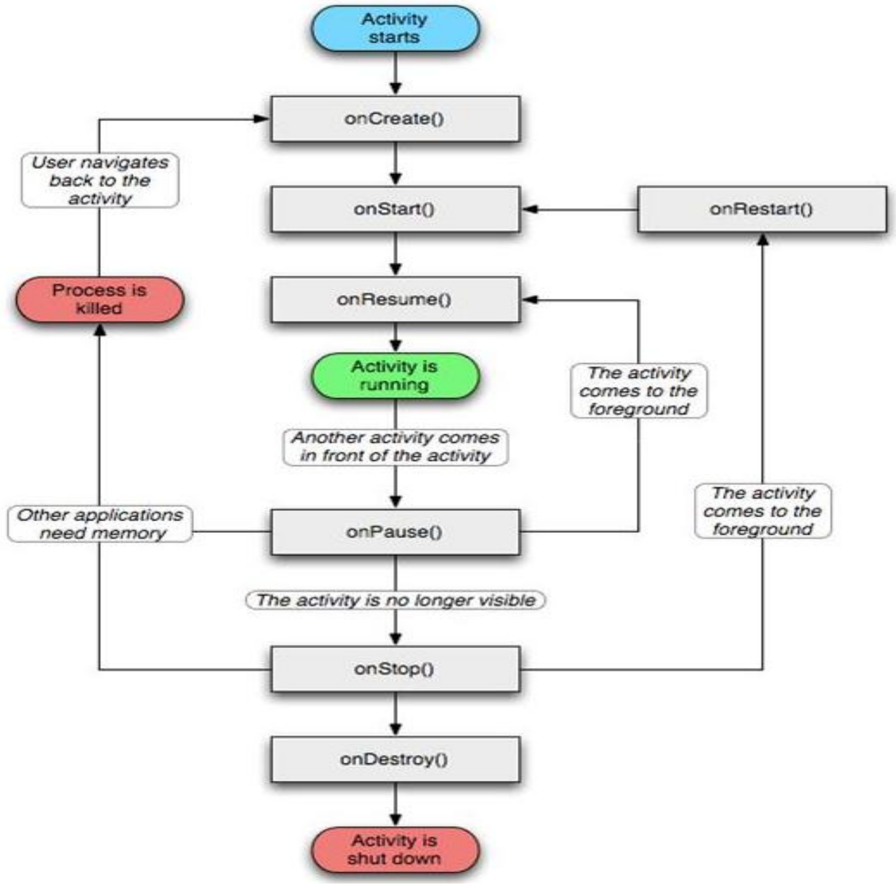
Activities Recap

- ❑ **`onCreate(Bundle)`** is where you initialize your activity, calling **`setContentView(int)`** with a layout resource defining your UI

- ❑ **`onPause()`** is where you deal with the user pausing active interaction with the activity. Any changes made by the user should at this point be committed (usually to the **`ContentProvider`** holding the data). In this state the activity is still visible on screen.

The Activity Life Cycle

- At the very minimum ,you need (and is supplied) `onCreate()`
- `onStop()` and `onDestroy()` are optional and may never be called
- If you need persistence, the save needs to happen in `onPause()`



Activity starts
onCreate()
onStart()
onRestart()
onResume()
User navigates back to the activity
Process is killed
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
Other applications need memory
onPause()
The activity comes to the foreground
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Introduction to Fragments

Reusable UI Components

Introduction to Fragments

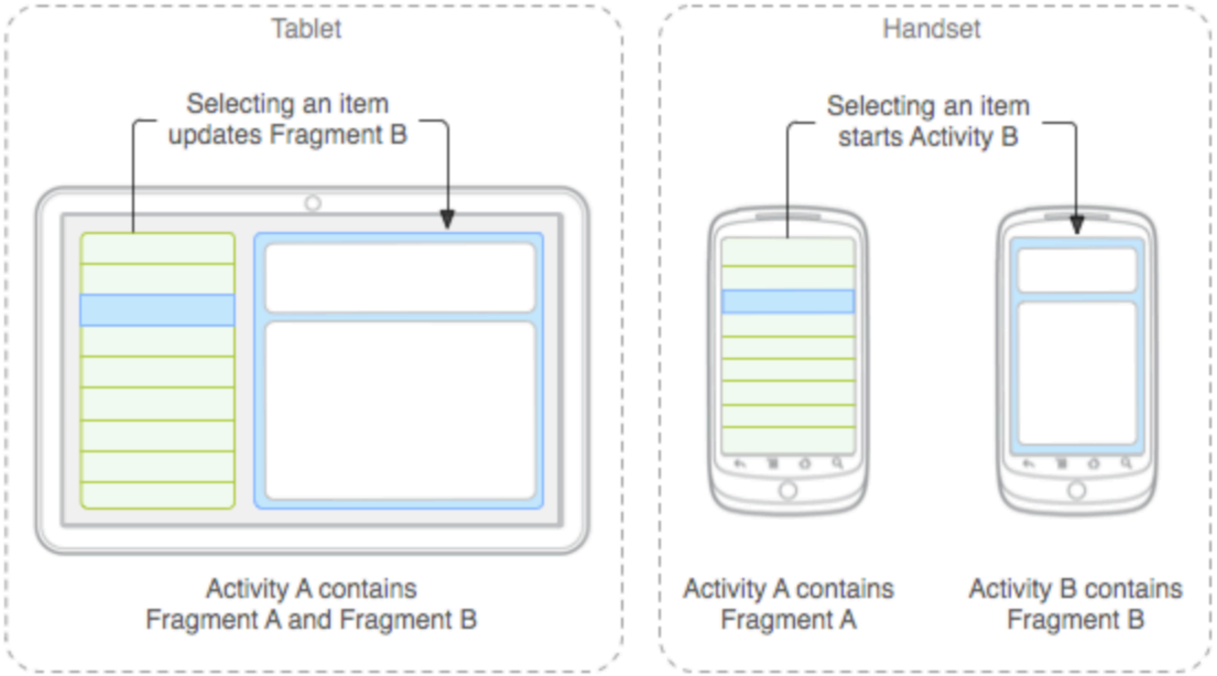
- **Fragments** represents a behaviour or a portion of a user interface in an Activity (specifically a **`FragmentActivity`**)
- Introduced in Android 3.0 (API level 11), primarily supports more dynamic and flexible UI designs on larger screens
- You can combine **multiple fragments in a single activity** to build a multi-pane UI and **reuse a fragment in multiple activities**
- Each Fragment has its **own lifecycle**, receives its own input events, and you can add or remove it while the activity is running

Designing Fragments

- ❑ You should design each fragment as a modular and reusable activity component.
- ❑ When designing your application to support both tablets and handsets, you can ***reuse your fragments*** in different layout configurations to optimize the user experience based on the available screen space.

- ❑ For example, on a handset, it might be necessary for separate fragments to provide a single-pane UI when more than one cannot fit within the same activity. (Next)

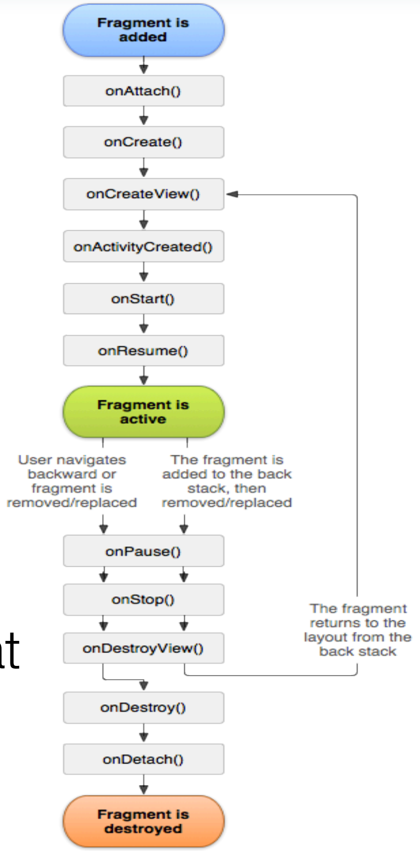
Designing Fragments

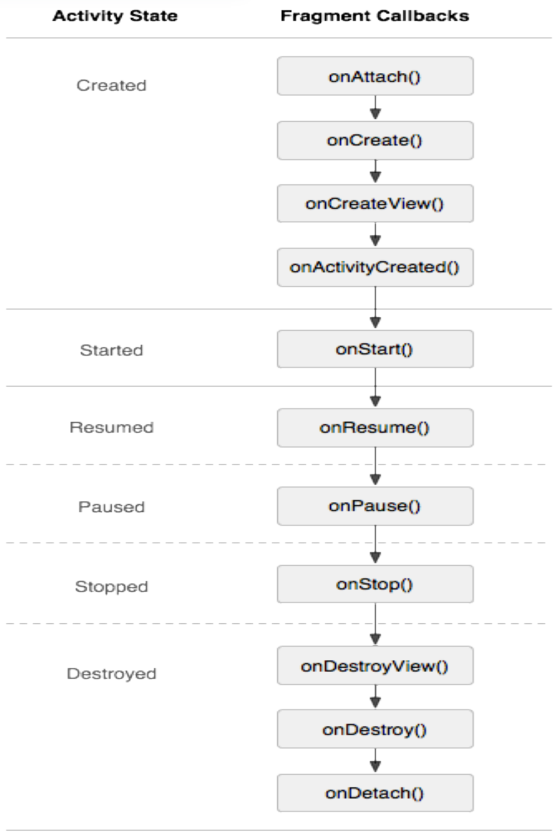

Tablet
Selecting an item updates Fragment B
Activity A contains Fragment A and Fragment B
Handset
Selecting an item starts Activity B
Activity A contains Fragment A
Activity B contains Fragment B

An example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design.

The Fragment Life Cycle

- ❑ To create a fragment, you must subclass Fragment (or an existing subclass of it).
- ❑ Has code that looks a lot like an Activity. Contains callback methods similar to an activity, such as *onCreate()*, *onStart()*, *onPause()*, and *onStop()*.
- ❑ Usually, you should implement at least *onCreate()*, *onCreateView()* and *onPause()*





Example Fragment

```
class ExampleFragment : Fragment() {

    override fun onCreateView(
            inflater: LayoutInflater,
            container: ViewGroup?,
            savedInstanceState: Bundle?
    ): View {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.example_fragment, container, false)
    }
}
```

Example Usage (in Activity Layout)

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="com.example.news.ArticleListFragment"
            android:id="@+id/list"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
    <fragment android:name="com.example.news.ArticleReaderFragment"
            android:id="@+id/viewer"
            android:layout_weight="2"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
</LinearLayout>
```

Fragment Managers & Transactions

- ❑ A great feature about using fragments in your activity is the ability to add, remove, replace, and perform other actions with them, in response to user interaction.

- ❑ Each set of changes that you commit to the activity is called a transaction and you can perform one by using APIs in `FragmentTransaction`.

- ❑ You can also save each transaction to a back stack managed by the activity, allowing the user to navigate backward through the fragment changes (similar to navigating backward through activities).

Managing Fragments

- ❑ To manage the fragments in your activity, you need to use `FragmentManager`. To get it, call `getSupportFragmentManager()` from your activity
- ❑ You acquire an instance of `FragmentTransaction` from the `FragmentManager` like this:

methods/references can be ‘chained’

```
val fragmentManager = supportFragmentManager
val fragmentTransaction = fragmentManager.beginTransaction()
```

- ❑ Each transaction is a set of changes that you want to perform at the same time. You can perform transaction methods such as `add()`, `remove()`, and `replace()`
- ❑ Then, to apply the transaction to the activity, you must call `commit()`

Managing Fragments

- Here we **add** a fragment using the **`add()`** method, specifying the fragment to add and the view in which to insert it

```kotlin
val fragment = ExampleFragment()
fragmentTransaction.add(R.id.fragment_container, fragment)
fragmentTransaction.commit()
```

- And render it like so

```xml
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Managing Fragments

- ❑ Before you call `commit()`, however, you might want to call `addToBackStack()`, in order to add the transaction to a back stack of fragment transactions

- ❑ This back stack is managed by the activity and allows the user to return to the previous fragment state, by pressing the Back button

- ❑ For example, here's how you can replace one fragment with another, and preserve the previous state in the back stack

Managing Fragments

```
val newFragment = ExampleFragment()
val transaction = supportFragmentManager.beginTransaction()
transaction.replace(R.id.fragment_container, newFragment)
transaction.addToBackStack(null)
transaction.commit()
```

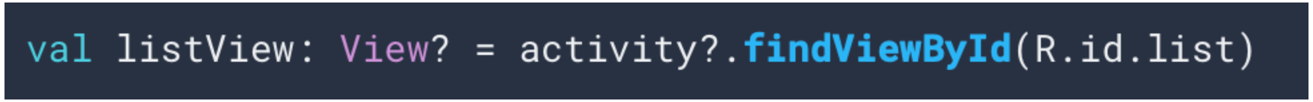
- In this example, **`newFragment`** replaces whatever fragment (if any) is currently in the layout container identified by the **`R.id.fragment_container`** ID. By calling **`addToBackStack()`**, the replace transaction is saved to the back stack so the user can reverse the transaction and bring back the previous fragment by pressing the Back button.

- **`FragmentActivity`** then automatically retrieves fragments from the back stack via **`onBackPressed()`**

Communicating with the Activity

- ❏ Although a **`Fragment`** is implemented as an object that's independent from a **`FragmentActivity`** and can be used inside multiple activities, a given instance of a fragment is directly tied to the activity that hosts it

- ❏ Specifically, the fragment can access the **`FragmentActivity`** instance with **`getActivity()`** and easily perform tasks such as find a view in the activity layout



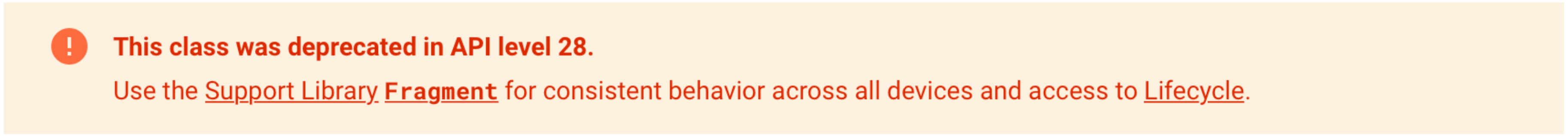
```
val listView: View? = activity?.findViewById(R.id.list)
```

Communicating with the Activity

- Likewise, your activity can call methods in the fragment by acquiring a reference to the **`Fragment`** from **`FragmentManager`**, using **`findFragmentById()`** or **`findFragmentByTag()`**

```
val fragment = supportFragmentManager.findFragmentById(R.id.example_fragment) as ExampleFragment
```

Fragments & Android Pie (API 28)



This class was deprecated in API level 28.

Use the Support Library **Fragment** for consistent behavior across all devices and access to Lifecycle.

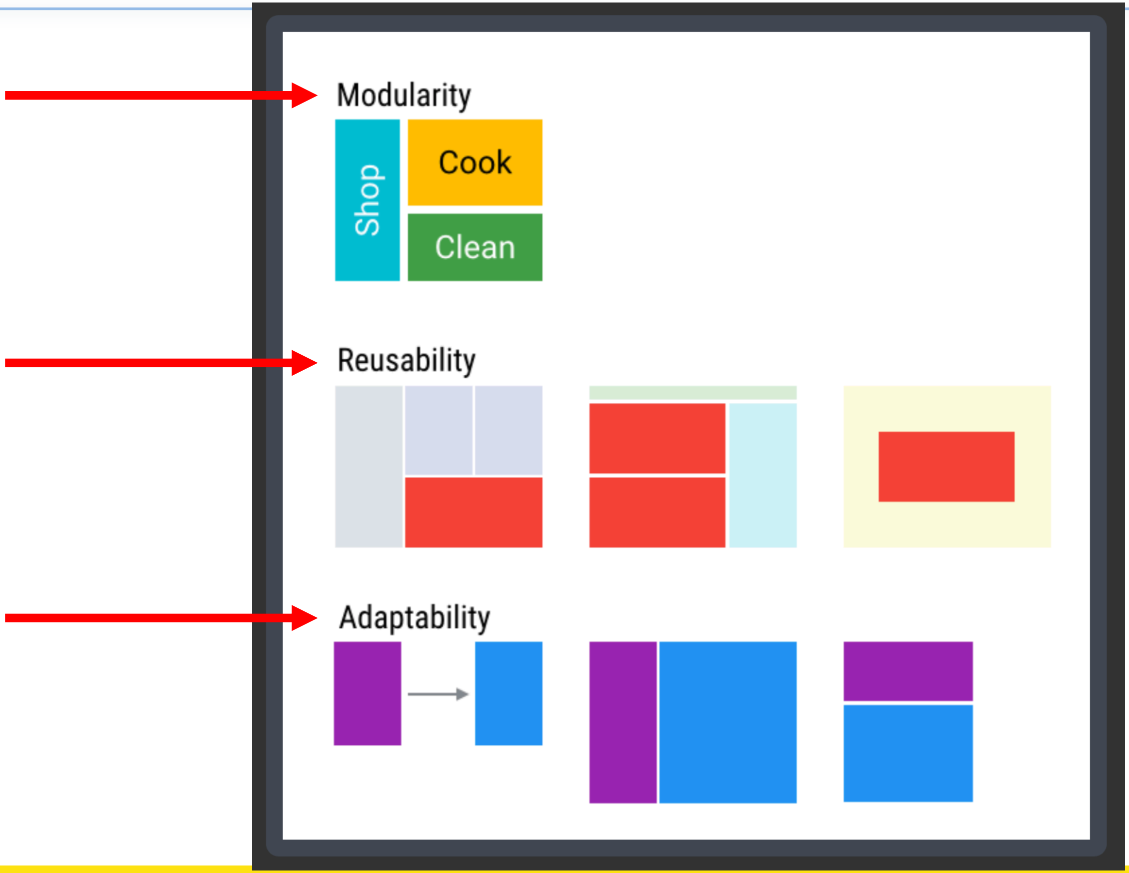
```
android.app.Fragment

getFragmentManager().beginTransaction()
```

Fragments Conclusion

- It’s worth noting you don’t have to use fragments - however, if you use them well, they can provide:
  - Modularity: Dividing complex activity code across fragments for better organization and maintenance.
  - Reusability: Placing behavior or UI parts into fragments that multiple activities can share.
  - Adaptability: Representing sections of a UI as different fragments and utilizing different layouts depending on screen orientation and size.

Fragments Conclusion



References

Sources: https://www.raywenderlich.com/1364094-android-fragments-tutorial-an-introduction-with-kotlin
https://developer.android.com/guide/components/fragments

Thanks.