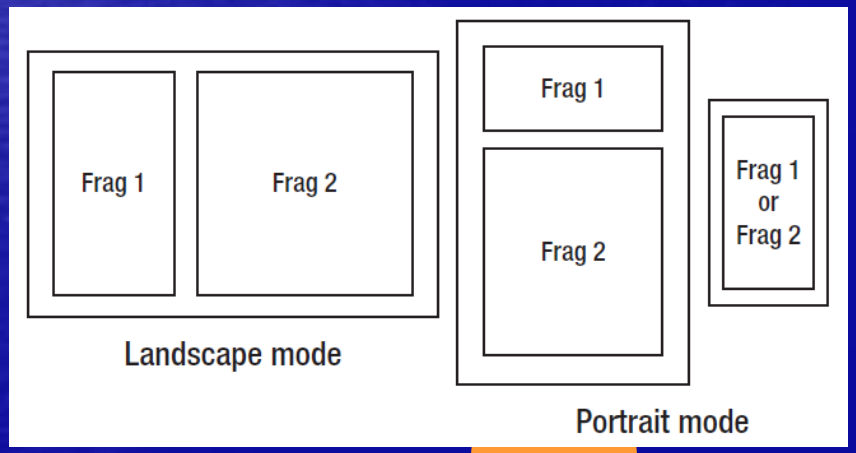
Mobile Computing

Fragments

Fragments (1)

- Activities are used to define a full screen interface and its functionality
  - That's right for small screen devices (smartphones)
  - In bigger devices we can have more interface elements and functionality
    - e.g. a list of e-mails and the content of the selected item
    - That's difficult to adapt using only activities



  - Recent versions of Android defined the fragment interface

Fragments (2)

- Fragments can be viewed as sub-activities
  - They have their own View hierarchy (layout)
  - They have their own lifecycle, but related to the parent activity lifecycle
  - They can respond to the back button, like activities
  - But fragments are contained inside an activity
    - They use the same thread (the UI thread)
    - They use the same context of the activity (they are inside)
    - Fragments can coexist with other elements of the activity
  - The activity layout can contain one or more fragments
  - The fragment lifecycle is related with its activity lifecycle
    - Lifecycle callbacks are called intermixed with the activity callbacks and executed by the same thread

Fragment layouts

- Fragments have their own internal layout
- Those layouts are included in activity layouts
  - Permanently through the <fragment> tag
  - Dynamically attaching and removing to some ViewGroup in the activity layout
    - The parent ViewGroup is usually a FrameLayout

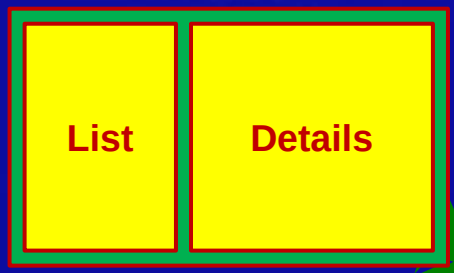
Recently we can also use in a layout for a fragment, a FragmentContainerView

Example Activity layout

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:baselineAligned="false"
  android:orientation="horizontal" >
  <fragment                                          static
    android:id="@+id/titles"
    android:name="org.feup.apm.shakespearefragments.TitlesFragment"
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:layout_weight="1" />
  <FrameLayout                                       dynamic
    android:id="@+id/details"
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:layout_weight="2" />
</LinearLayout>
```

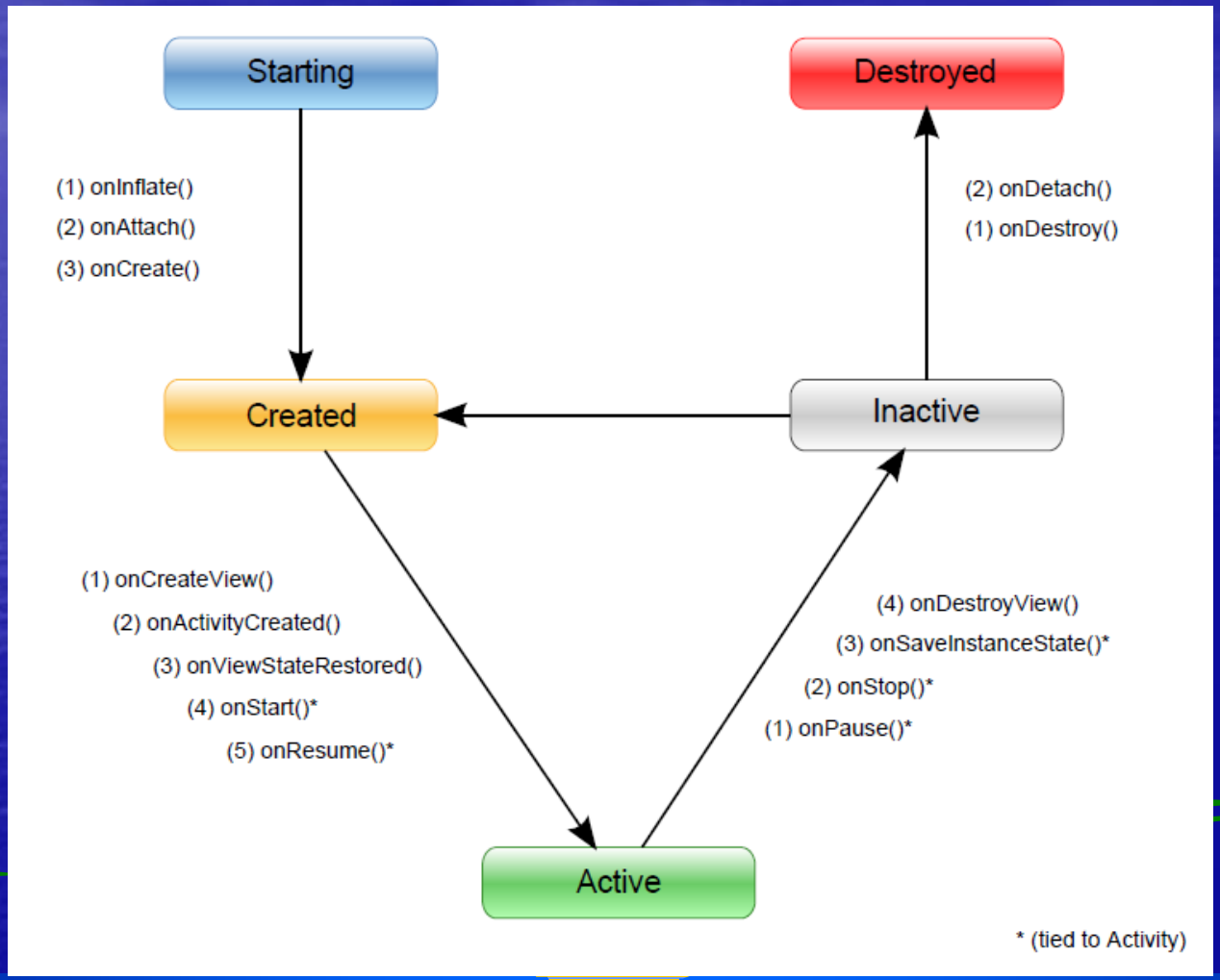
class implementing the fragment

placeholder for attaching fragments



Fragment lifecycle

- The fragment lifecycle follows the parent activity



Lifecycle callbacks (1)

Entry sequence callbacks:

onInflate(activity: Activity, attrs: AttributeSet, savedInstanceState: Bundle) – called in the beginning whenever the activity sets its content layout and has a <fragment> in it. The AttributeSet contains the attributes defined in the activity layout. They should be parsed and saved. The fragment is not yet attached to the activity.

onAttach(activity: Activity) – The activity containing the fragment is now attached. You can save it, or get it, while the fragment is attached, using getActivity(). You can get the initialization arguments (set by setArguments() until this point) anytime, after with getArguments().

onCreate(savedInstanceState: Bundle) – called at the beginning of the owner activity onCreate() callback. Usually, the activity View hierarchy is not yet inflated. You can create here another thread to do lengthy data loading operations.

onCreateView(inflater: LayoutInflater, container: ViewGroup, savedInstanceState: Bundle) – Here you should return the inflated View of this fragment. If the container is null, you should return null (the fragment will not be displayed). The container is the ViewGroup in the activity layout that will display the fragment. Do not attach the fragment to this container. Some specialized fragments (e.g., ListFragment) do not need this callback.

onActivityCreated(savedInstanceState: Bundle) – here the onCreate() method of the activity is now complete. The complete interface is now built, including other present fragments.

Lifecycle callbacks (2)

onStart() and onResume() – tied with the activity corresponding callbacks.

The exit sequence of callbacks include the following, and they are called when the activity is also exiting or the fragment is being replaced (back button, other actions ...)

onPause() – the first to be called (the fragment can be put on the fragment back stack). You should stop playing sounds, related to this fragment, here.

onStop() – tied with the onStop() callback of the activity. A stopped fragment can go straight to the onStart() callback.

onDestroyView() – when the fragment is being killed or saved, this will be called. Here its View hierarchy is already detached from the activity layout.

onDestroy() – is called when the fragment is no longer in use (but still existing in the activity).

onDetach() – here the fragment does not belong anymore to the activity and the interface resources are already freed.

onSaveIntanceState(outstate: Bundle) – called somewhere before onDestroy(). It should save internal state in the provided Bundle. That Bundle is passed to the entry callbacks.

Fragment creation

- Fragments can be constructed by the system
  - Whenever the activity inflates its layout and has a <fragment> element in it (static)
- They can be built in your code
  - If you want to replace or attach a new fragment to some container (dynamic)
    - In this case you should write a static factory method in your fragment class (if you need to pass arguments)
    - Fragment classes should not have constructors, but a factory

Example

```
fun MyFragment newInstance(index: Int) {
    val f = MyFragment()
    val args = Bundle()
    args.putInt("index", index)
    f.arguments = args
    return f
}
```

. The standard way to build a Fragment and pass it the initialization parameters. They should be 'bundled' and put in the class assigning to the arguments property.
. They will be available inside the fragment code in that property.
. This arguments bundle is automatically preserved in rotations

FragmentManager

- Activities and Fragments can manipulate the active fragments
  - That is the task of the FragmentManager object
    - It is obtained on the supportFragmentManager property from an Activity or Fragment
      - It can find Fragments
      - It can manipulate the fragment back stack
      - It can save and restore references to fragments and fragment internal state
    - Adding, replacing, removing, hiding and showing existing fragments must be done inside a transaction
      - beginTransaction() begins a new transaction and returns a FragmentTransaction object
      - FragmentTransaction do the stated operations on fragments
      - At the end you should call commit() on the FragmentTransaction

Example

Suppose we have an activity showing some information, in a fragment, according to a list selection (position).
The fragment, when constructed, is supplied with this position info, and its layout will contain the corresponding data.

The handler for a new selection in the original list could be the following, replacing the visualizing fragment by a new one:

```
// See if there is a fragment displayed already
var details = supportFragmentManager.findFragmentById(R.id.details) as DetailsFragment?
if (details == null || details.mPos !== position) {          // no fragment or not the current data
      // Make a new fragment to show this selection.
      details = DetailsFragment.newInstance(position)

      // Execute a transaction, replacing any existing fragment with the new one.
      var ft = supportFragmentManager.beginTransaction()
      ft.setCustomAnimations(R.animator.slide_in_left, R.animator.slide_out_right)
      ft.replace(R.id.details, details)
      // The transaction is added to the back stack to allow undoing it (with the back button)
      ft.addToBackStack("Details")
      ft.commit()
}
```

Fragment implementation example

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/scroller"
  android:layout_width="match_parent"
  android:layout_height="match_parent" >
  <TextView
    android:id="@+id/text1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
</ScrollView>
```

The Details fragment internal layout

(for a big TextView (with scroll))

```
class DetailsFragment : Fragment() {
  var mPos = 0;

    companion object {
        fun newInstance(index: Int)  =
            DetailsFragment().apply {
                arguments = Bundle().apply {
                    putInt("index", index)
                }
            }
    }

    override onCreate(myBundle: Bundle) {
        super.onCreate(myBundle);
        mPos = arguments.getInt("index", 0);
    }
}
```

The Details fragment implementation

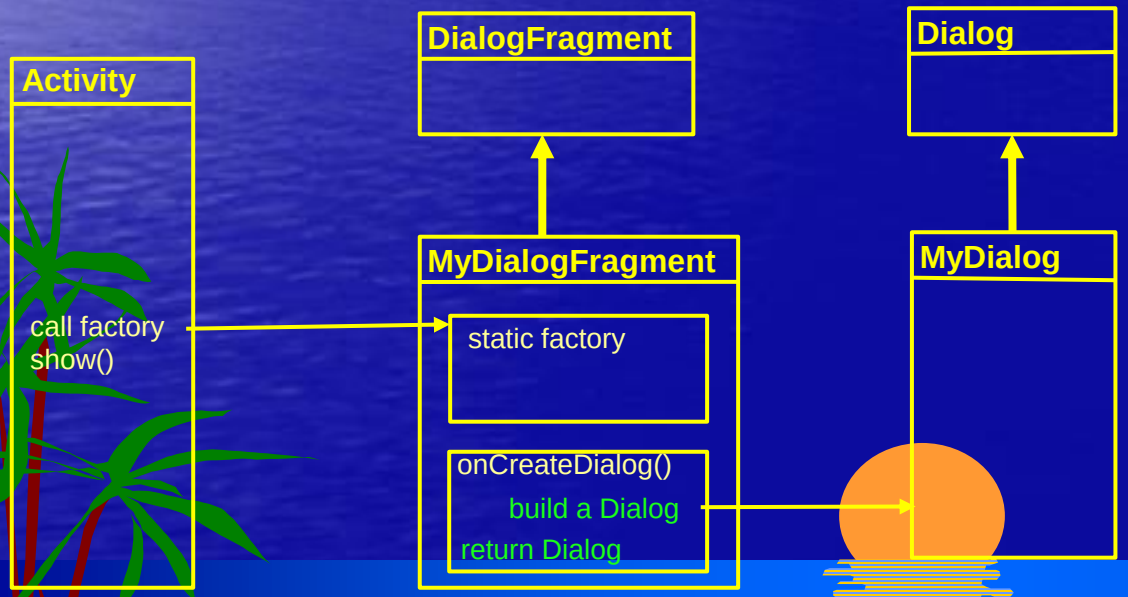
(the method that returns the fragment layout)

```
override onCreateView(inflater: LayoutInflater, container: ViewGroup?,
                      savedInstanceState: Bundle?): View? {
    if(container == null)
        return null;

    // Don't tie this fragment to anything through the inflater.
    val v = inflater.inflate(R.layout.details, container, false)
    v.findViewById<TextView>(R.id.text1).text =
                Shakespeare.DIALOGUE[mPos]    // some text in an array
    return v
}
```

DialogFragment

- The method of building dialogs from the Dialog class doesn't take care automatically of lifecycle events, like the destruction and re-creation
  - Embedding the dialog in a DialogFragment does that
  - It also allows to display the whole dialog in a part of the screen, like a fragment



To show a dialog through the DialogFragment we need the FragmentManager

```
val newFragment = MyDialogFragment.newInstance(…);
newFragment.show(supportFragmentManager,"tag");
```