Fragments

Android Development

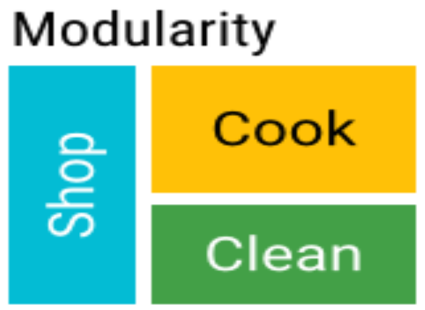
What is a Fragment?

- **fragment** | *noun* | /’frag-mənt/
- An isolated or incomplete part of something

It can be viewed as:

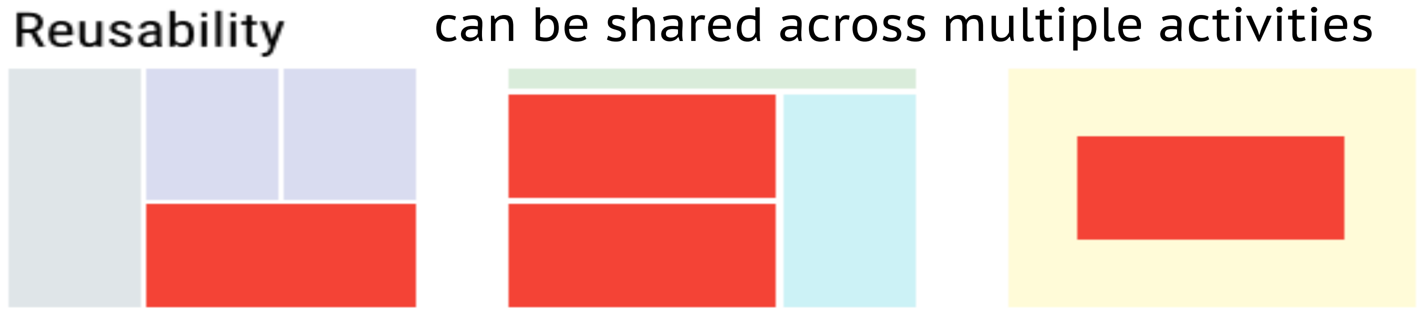
- A module of code that holds part of the behavior and/or UI of an `activity`.
- Has a set of events that signal various stages of its lifecycle.
- Has its own associated `View` object, which defines its UI.
- Functional "sub-activity" with its own lifecycle similar to that of a full `Activity`.

They can provide

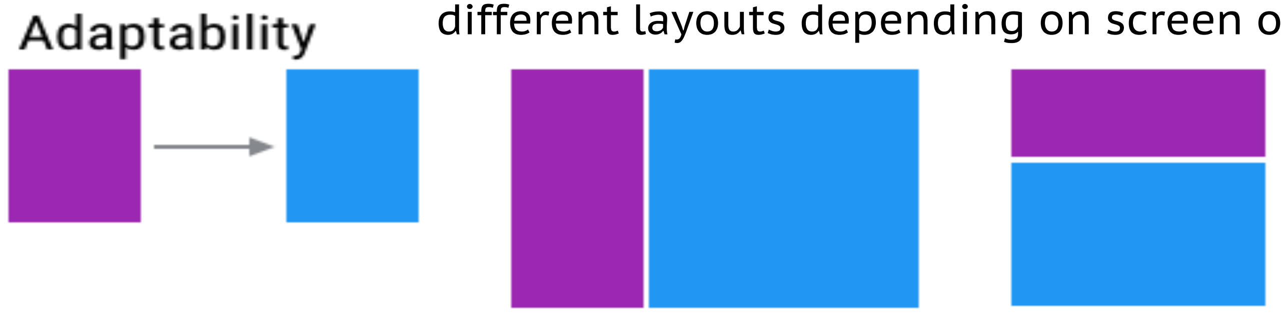


Dividing complex activity code across fragments for better organization and maintenance.

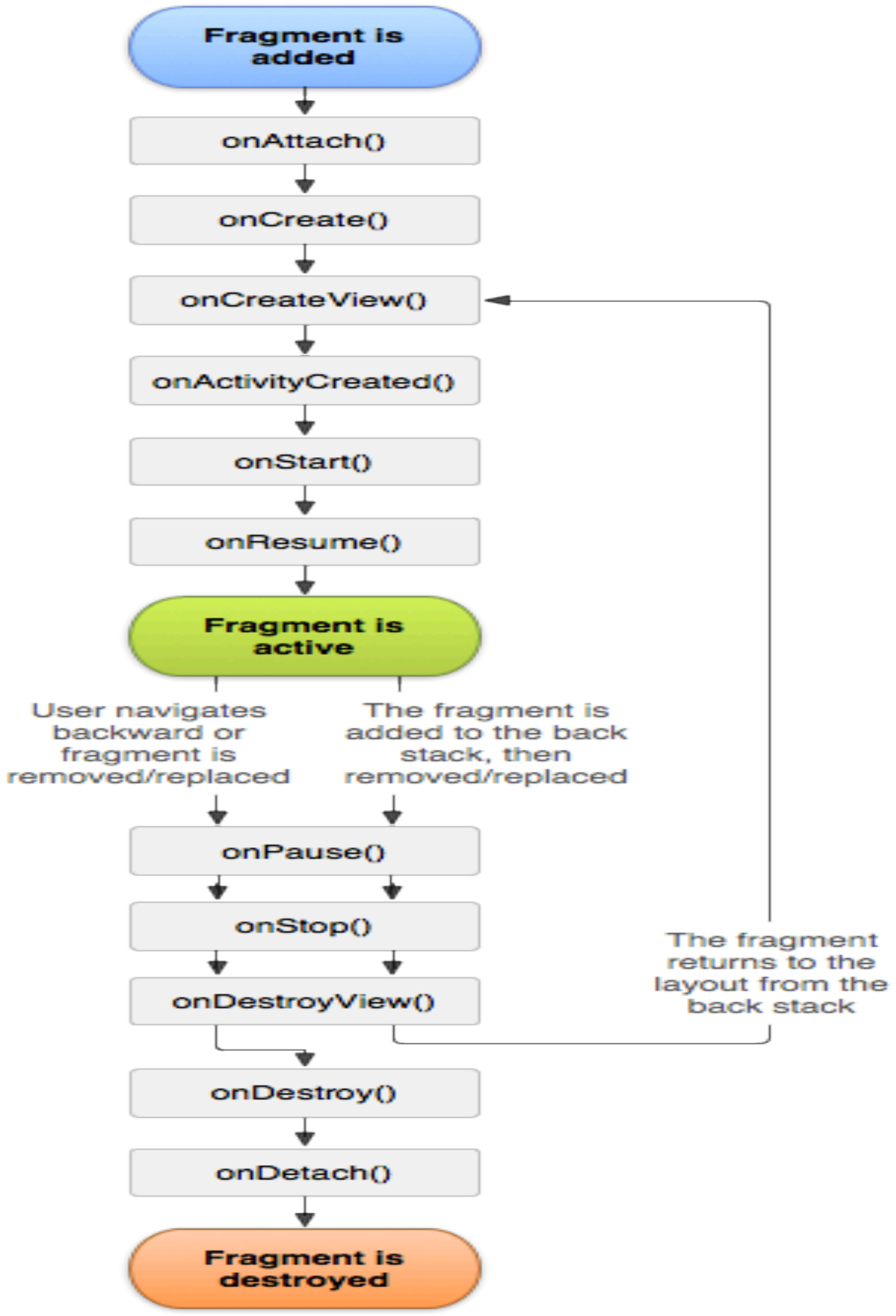
Placing behavior or UI parts into fragments that can be shared across multiple activities



Representing sections of a UI as different fragments Represents different layouts depending on screen orientation and size.



+
Fragment Lifecycle



+ Lifecycle events when creating a fragment:

Method	Description
`onAttach`	• Called after the `Fragment` is associated with the `Activity`. • This is the first method to be run when the `Fragment` is ready to be used.
`onCreate`	• Called by the `Activity` to create the `Fragment`. • Earliest time at which the `Fragment` may begin gathering the data that it needs. • The `Fragment` is running in the UI thread, so avoid any lengthy processing.
`onCreateView`	• Creates the `view` for the Fragment. • Called once the Activity's `OnCreate()` method is complete. • At this point, it is safe to interact with the view hierarchy of the `Activity`. • Returns the `view` that will be used by the `Fragment`.

+ Contd…

Method	Description
onActivityCreated	• When the fragment’s activity has finished its own onCreate event
onStart	• Called after the containing Activity has been resumed. • Fragment is visible to the user.
onResume	• Last method called before the user can interact with the Fragment. • E.g. enabling features of a device that the user may interact with, such as the camera that the location services.

Lifecycle events when you removing a fragment

Method	Description
`onPause`	• The user is no longer able to interact with the `Fragment`. • When active, it's the first indication that the user is leaving the `Fragment`. • `Fragment` should save any changes.
`onStop`	• `Fragment` is no longer visible.
`onDestroyView`	• This method cleans up the resources associated with the `view`. • Called when the `view` associated with the `Fragment` has been destroyed.

+
Contd..

Method	Description
`onDestroy`	• Called when the `Fragment` is no longer in use. • Still associated with the `Activity`, but the `Fragment` is no longer functional. • Releases any resources that are in use by the `Fragment`.
`onDetach`	• Called just before the `Fragment` is no longer associated with the `Activity`. • The view hierarchy of the `Fragment` no longer exists • All resources that are used by the `Fragment` should be released at this point.

Using `SetRetainInstance`

- Used when a `Fragment` is specifying that it should not be completely destroyed if the `Activity` is being re-created.
- If `true` is passed to this method, then when the `Activity` is restarted, the same instance of the `Fragment` will be used.
- If this happens, then all callback methods will be invoked except the `OnCreate` and `OnDestroy` lifecycle callbacks.

Fragment State Management

- Fragments may save and restore their state by using an instance of a Bundle.
- The Bundle allows a Fragment to save data as key/value pairs - useful for simple data that doesn't require much memory.
- A Fragment can save its state with a call to:

```
public override void OnSaveInstanceState(Bundle
outState)
{
    base.OnSaveInstanceState(outState);
    outState.PutInt("current_choice",
    _currentCheckPosition);
}
```

+
Making the Bundle available

- When a new instance of a `Fragment` is created, the state saved in the `Bundle` will become available to the new instance via the `OnCreate`, `OnCreateView`, and `OnActivityCreated` methods.

- E.g. retrieving the `current_choice` from the `Bundle`:

```
public override void OnActivityCreated(Bundle
savedInstanceState)
{
    base.OnActivityCreated(savedInstanceState);
    if (savedInstanceState != null)
    {
        _currentCheckPosition =
savedInstanceState.GetInt("current_choice", 0);
    }
}
```

+

`Bundle` Limitations

- If the `Fragment` is not added to the back stack, then its state will not be restored when the user presses the `Back` button.
- When the `Bundle` is used to save data, that data is serialized.
- Can lead to processing delays.

+
Creating a Fragment

- Fragments were **not** introduced to Android until version 3.0 of the Android SDK.
- Introduced as part of the `Honeycomb` release for creating device-specific layouts for a single app.
- The `v4 Support Library` provides a `fragment` implementation for devices running Android below 3.0 via the `android.support.v4.app.Fragment.`
- An application that uses `Fragments` must make use of the `android-support-v4 Android Support Library` in order to be compatible with older Android versions

+
UI Fragments

- This is a `fragment` managing a user interface.
- Has a `view` of its own that is inflated from a layout file.
- The `activity’s View` contains a spot where the `fragment’s View` will be inserted.
- Fragments are stored in the form of `XML layout files.`
- Added to an activity either by:
  - Placing appropriate `<fragment>` elements in the `activity’s` layout file
  - Directly through code within the `activity’s` class implementation.
- To display its `View` object, a `Fragment` has to pass it on to an `Activity`

Consider this example

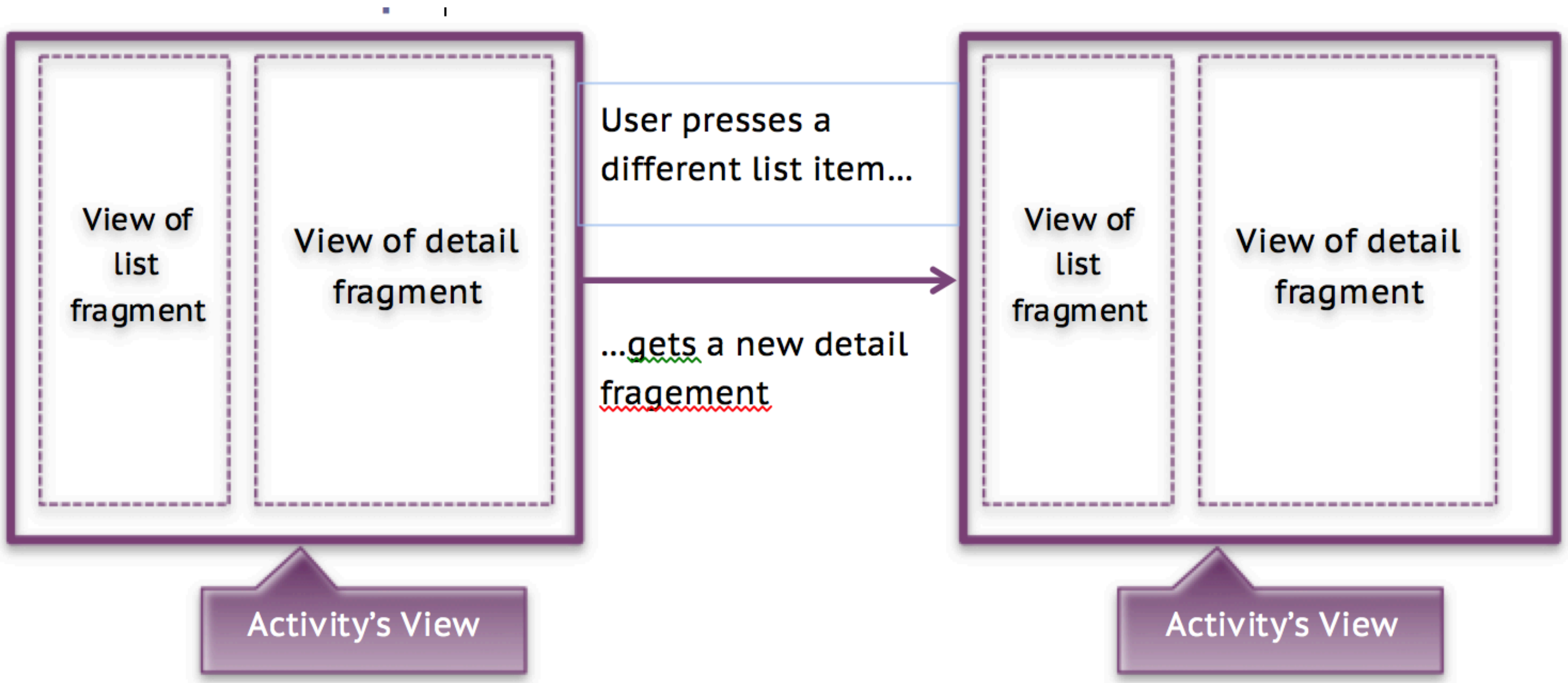


FIGURE 5 FRAGMENTS EXAMPLE

What is happening?

- An app is displaying the `list` and `detail` together.
- The `activity's view` is composed from a `list fragment` and a `detail fragment`.
- The `detail view` shows the details of the `selected list item`.
- Selecting another item should display a `new detail view`.
- No activities need to die for this major view change to happen.

Why use fragments?

- Separates the UI of your app into building blocks
- Useful for more than just list-detail applications.
- Easy to build tab interfaces, tack on animated sidebars, and more.
- <u>Achieving this UI flexibility comes at a cost:</u>
  - more complexity
  - more moving parts
  - more code.