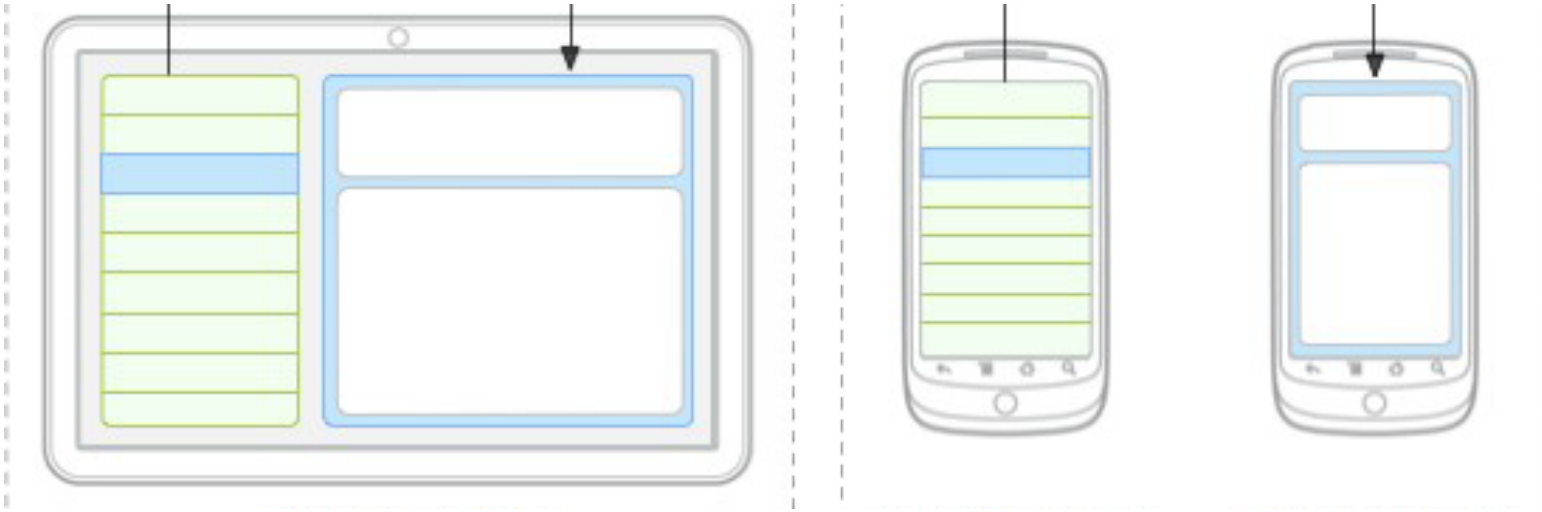
Fragments

Can Liu

Situational Layouts

- Different device type (tablet vs phone vs watch)
- Different screen size
- Different orientation (portrait vs. landscape)
- Different country or locale (language, etc.)

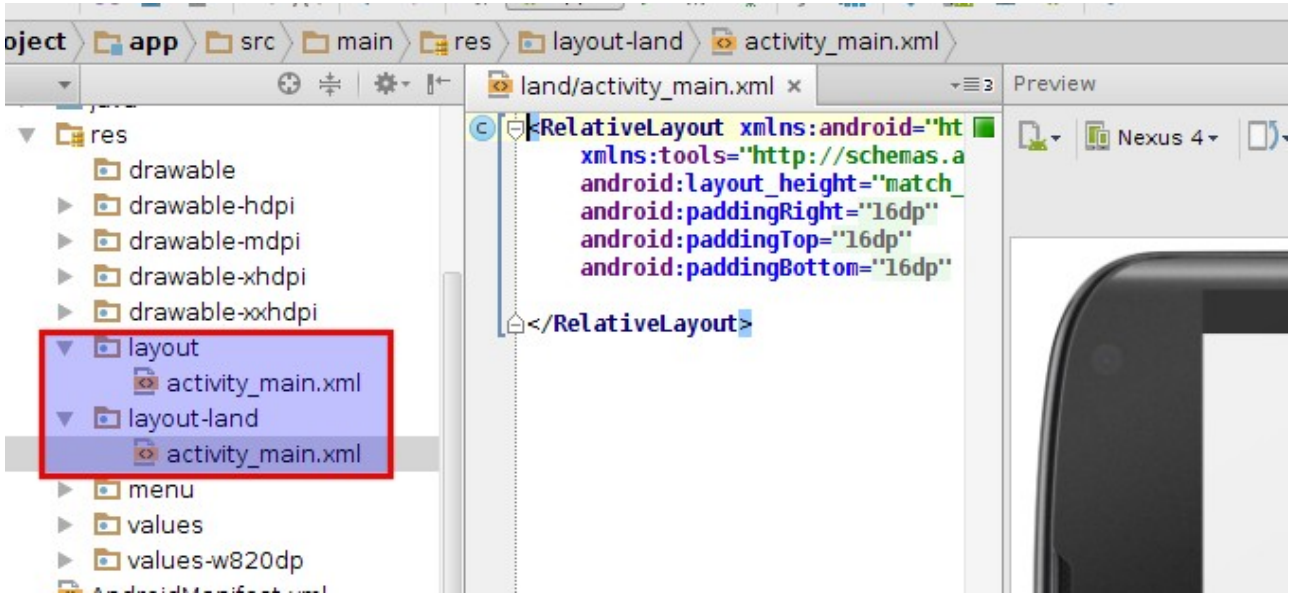


Situation-specific folders

- Your app will look for resource folder names with suffixes:
  - screen density (e.g. drawable-hdpi)
    - xhdpi: 2.0 (twice as many pixels/dots per inch) hdpi: 1.5
    - mdpi: 1.0 (baseline)
    - ldpi: 0.75
  - screen size (e.g. layout-large)
    - small, normal, large, xlarge
  - orientation (e.g. layout-land)
    - portrait (), land (landscape)

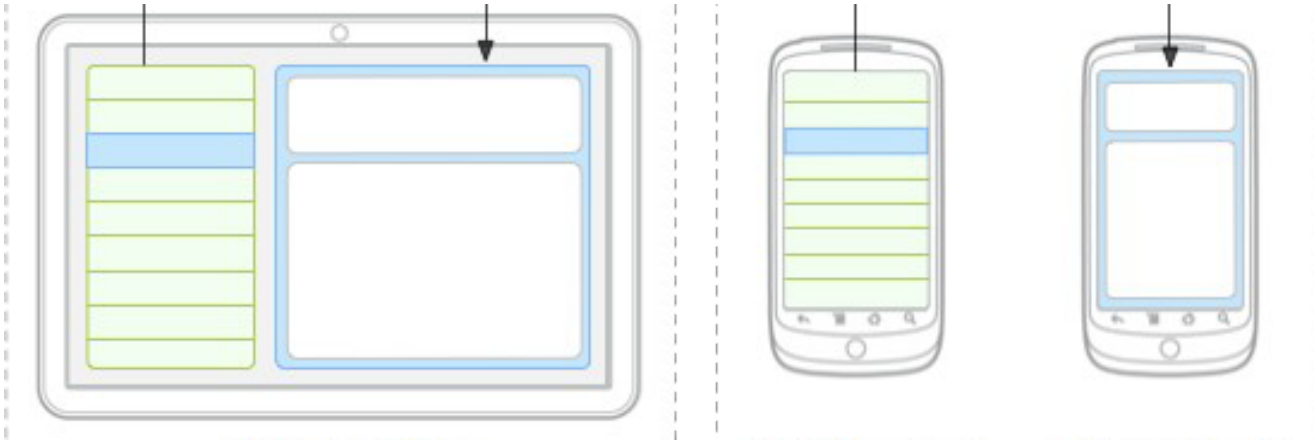
Portrait vs landscape layout

- To create a different layout in landscape mode:
  - create a folder in your project called res/layout-land
  - place another copy of your activity's layout XML file there
  - modify it as needed to represent the differences



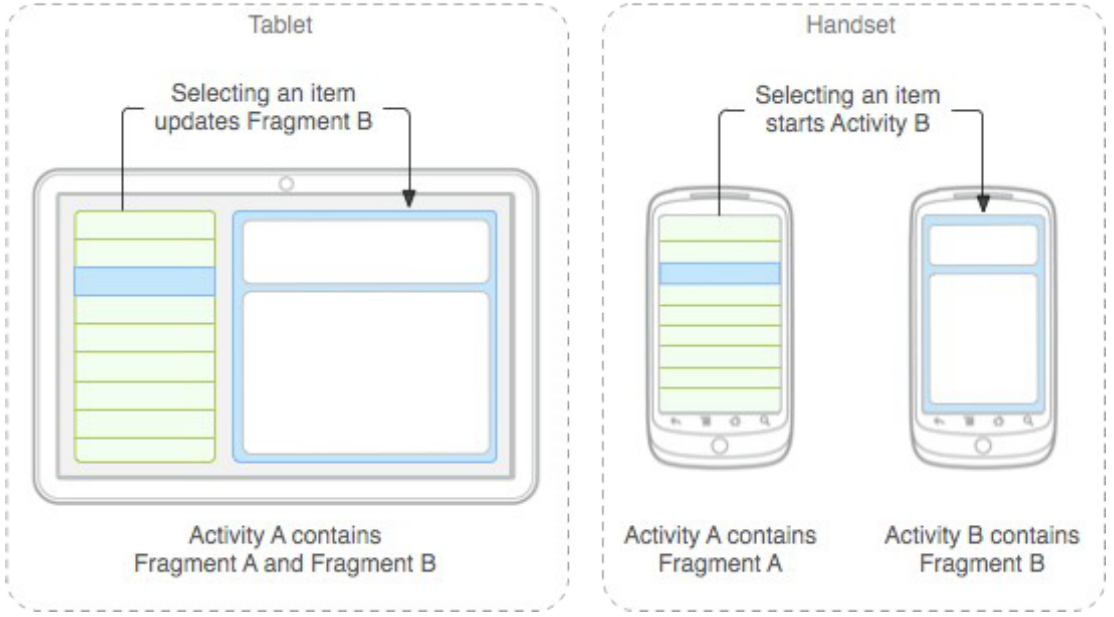
Problem: redundant layouts

- With situational layout you begin to encounter redundancy.
  - The layout in one case (e.g. portrait or medium) is very similar to the layout in another case (e.g. landscape or large).
  - You don‘t want to represent the same XML or Java code multiple times in multiple places.
- You sometimes want your code to behave situationally.
  - In portrait mode, clicking a button should launch a new activity.
  - In landscape mode, clicking a button should launch a new view.



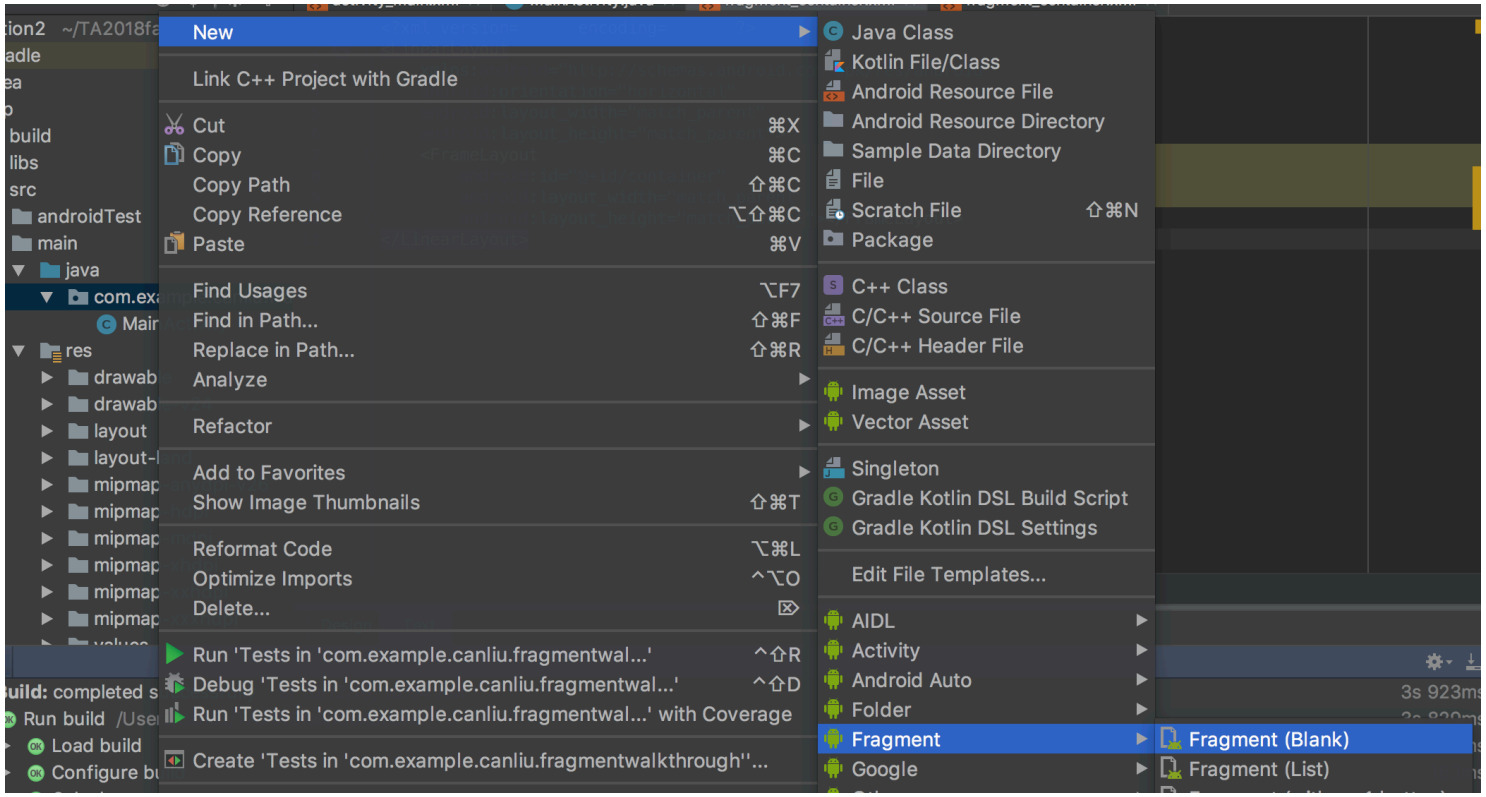
Fragments

- **fragment:** A reusable segment of Android UI that can appear in an activity.
  - can help handle different devices and screen sizes
  - can reuse a common fragment across multiple activities
  - First added in Android 3.0 (usable in older versions if necessary)



Creating a fragment

- In Android Studio, right-click app, click:
- New → Fragment → Fragment (blank)
  - – un-check boxes about "Include_methods"
  - – now create layout XML and Java event code as in an Activity



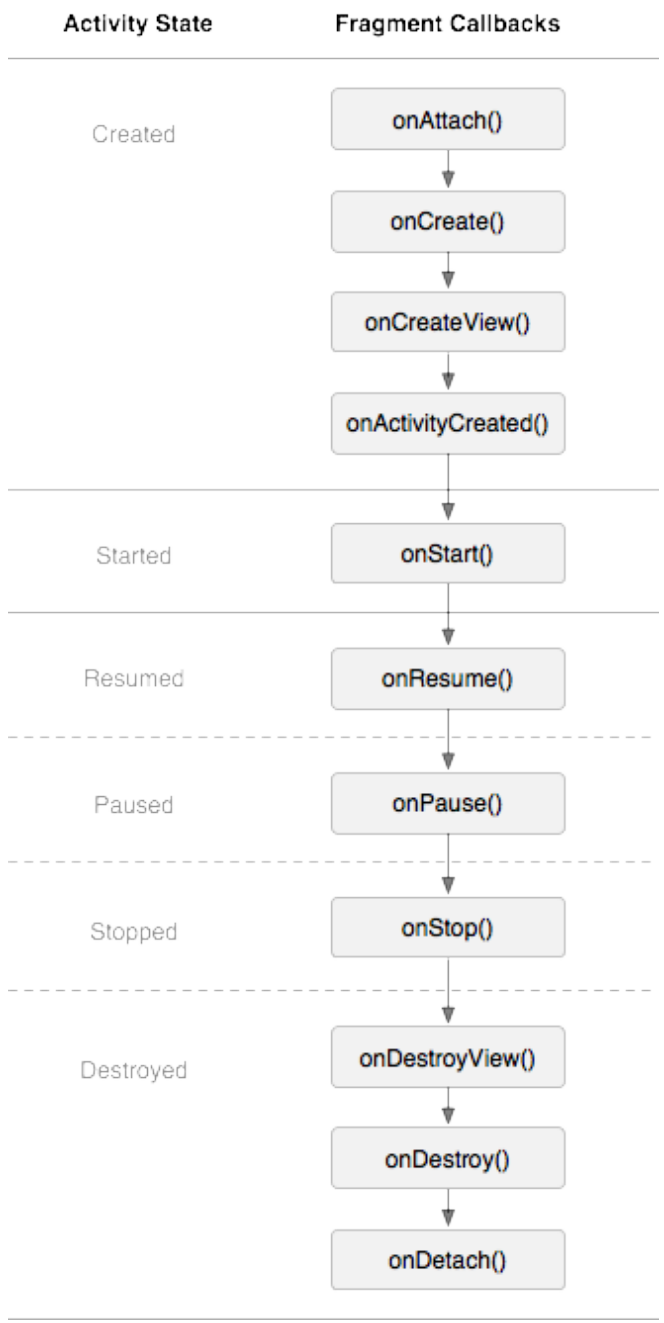
Using fragments in activity XML

- Activity layout XML can include fragments.

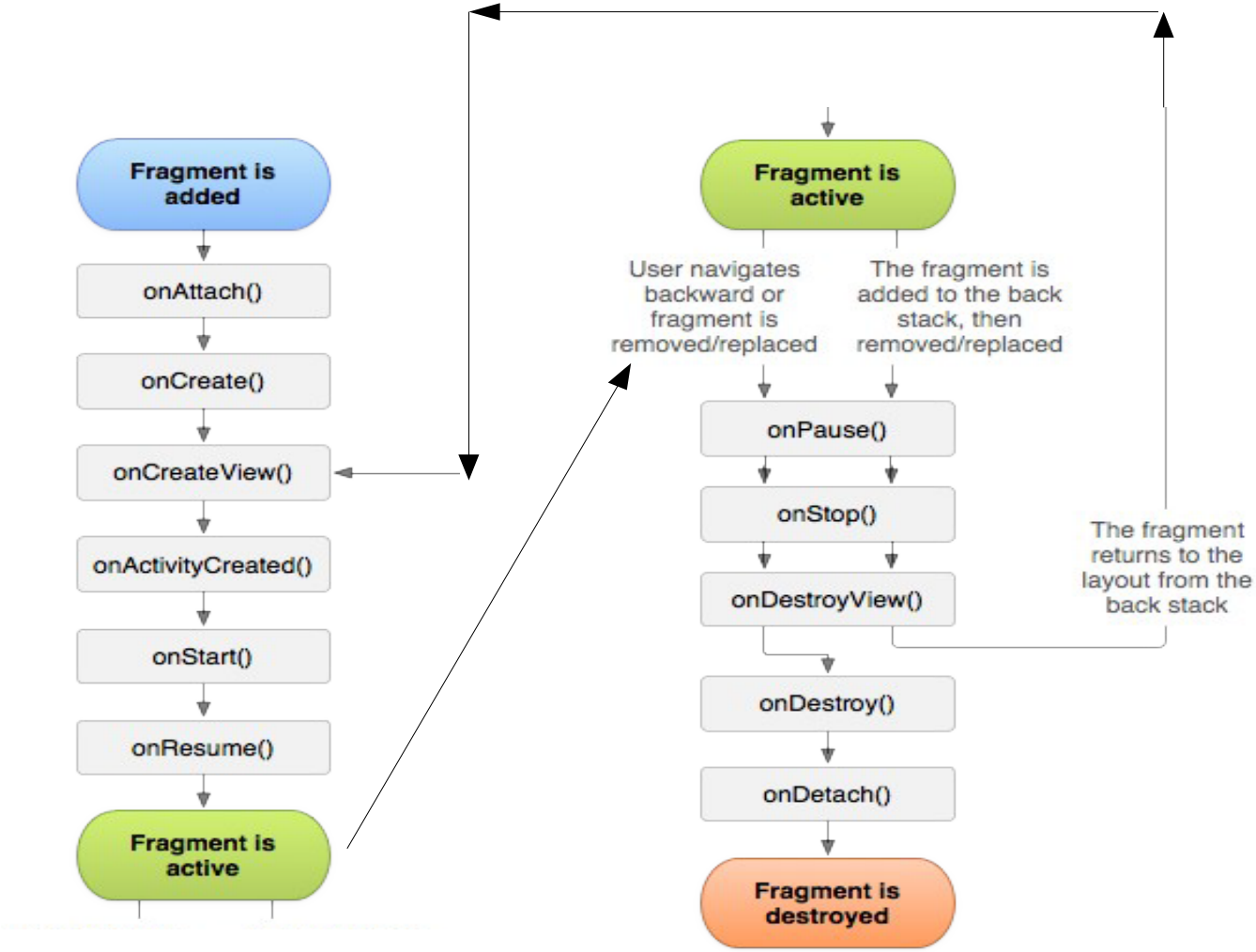
```
<!-- activity_name.xml -->
<LinearLayout ...>
        <fragment ...
                android:id="@+id/id1"
                android:name="ClassName1"
                tools:layout="@layout/name1" />
        <fragment ...
                android:id="@+id/id2"
                android:name="ClassName2"
                tools:layout="@layout/name2" />
</LinearLayout>
```

Fragment life cycle

- Fragments have a similar life cycle and events as activities.
- Important methods:
  - – **onAttach** to glue fragment to its surrounding activity
  - – **onCreate** when fragment is loading
  - – **onCreateView** method that must return fragment's root UI view
  - – **onActivityCreated** method that indicates the enclosing activity is ready
  - – **onPause** when fragment is being left/exited
  - – **onDetach** just as fragment is being deleted



Another fragment lifecycle view



Fragment vs. activity

- Fragment code is similar to activity code, with a few changes:
  - Many activity methods aren‘t present in the fragment, but you can call getActivity to access the activity the fragment is inside of.

    ~~Button b = (Button) findViewById(R.id.but);~~

    Button b = (Button) getActivity().findViewById(R.id.but);
  - Sometimes also use getView to refer to the activity's layout
  - Event handlers cannot be a ached in the XML any more. :-(
    - Must be attached in Java code instead.
  - Passing information to a fragment (via Intents/Bundles) is trickier.
    - The fragment must ask its enclosing activity for the information.
  - Fragment initialization code must be mindful of order of execution.
    - Does it depend on the surrounding activity being loaded? Etc.
    - Typically move onCreate code to onActivityCreated