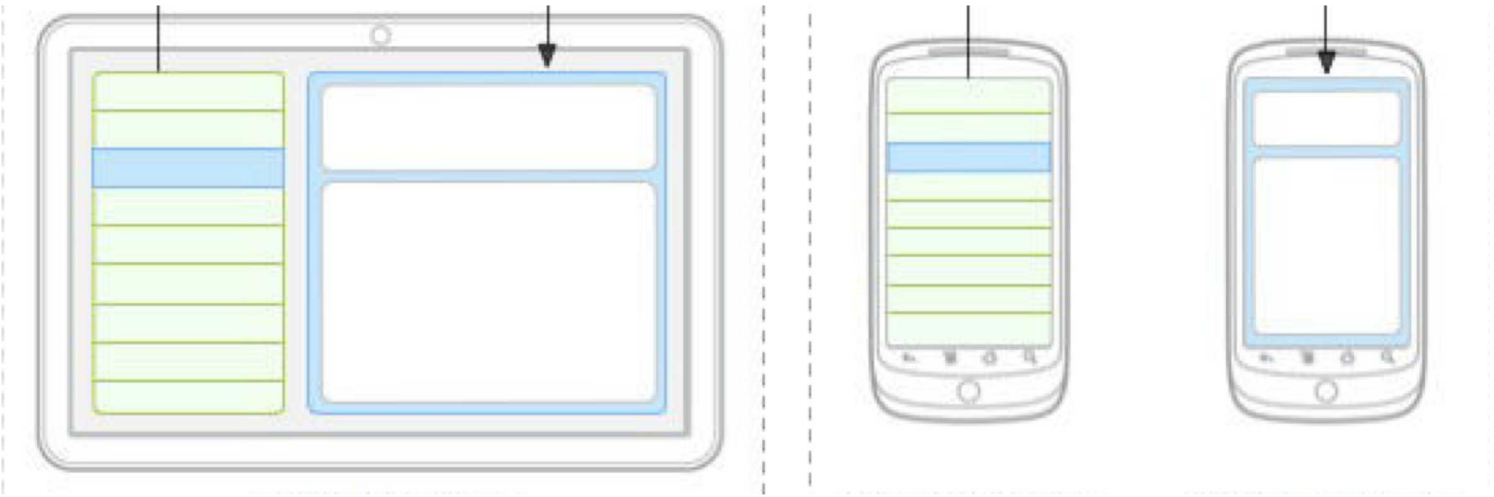
# 14. Fragments

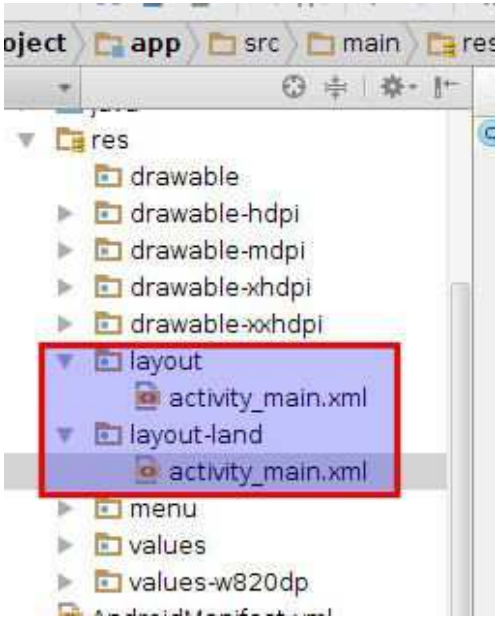
# Situational layouts

- Your app can use different layout in different situations:
  - different device type (tablet vs phone vs watch)
  - different screen size
  - different orientation (portrait vs. landscape)
  - different country or locale (language, etc.)



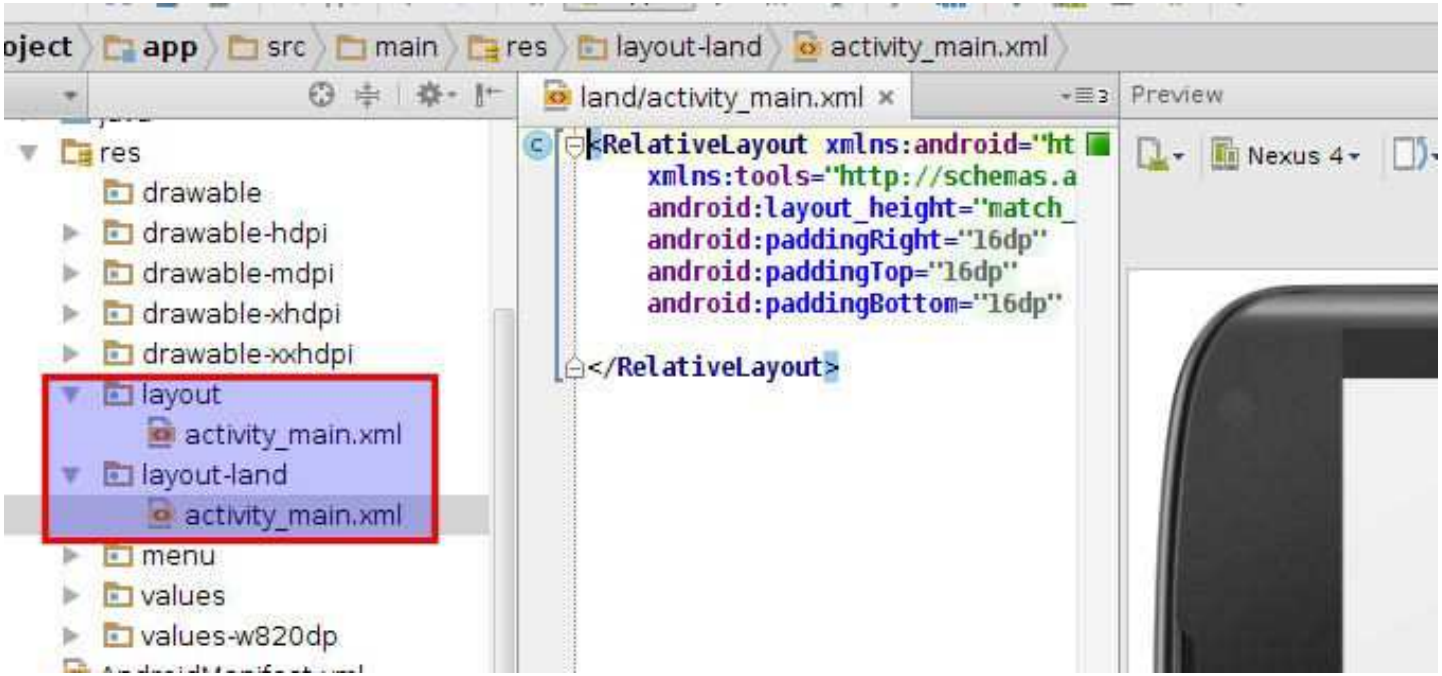
# Situation-specific folders

- Your app will look for resource folder names with suffixes:
  - screen density (e.g. **drawable-hdpi**) (link)
    - 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) (link)
    - small, normal, large, xlarge
  - orientation (e.g. layout-land)
    - portrait (), land (landscape)



# Portrait vs landscape layout

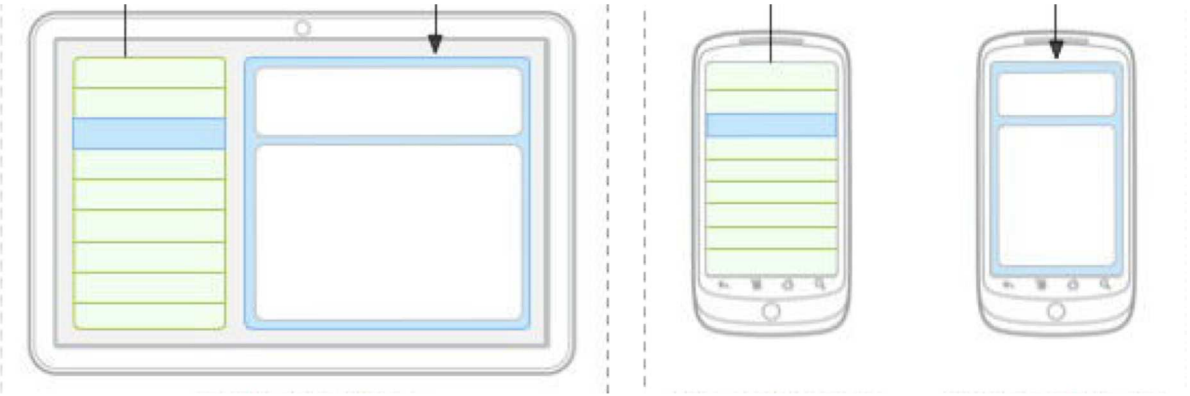
- To create a different layout in landscape mode:
  - create a folder in your project called **res/layout-land**
  - place another copy of your activity's **layout XML file** there
  - modify it as needed to represent the differences



# Problem: redundant layouts

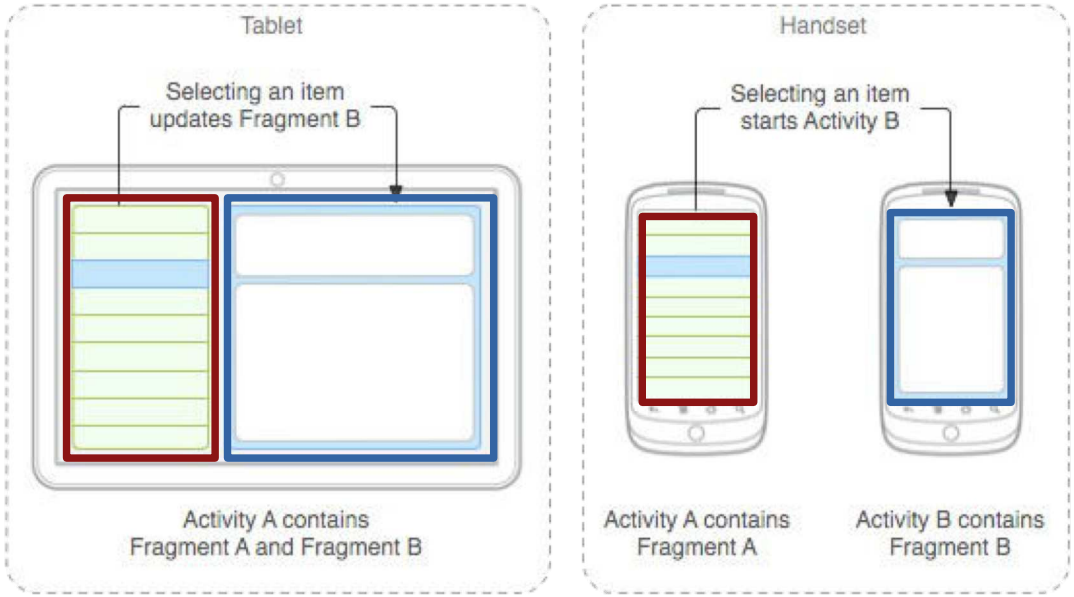
- With situational layout you begin to encounter **redundancy**.
  - The layout in one case (e.g. portrait or medium) is very similar to the layout in another case (e.g. landscape or large).
  - You don't want to represent the same XML or Java code multiple times in multiple places.

- 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**.



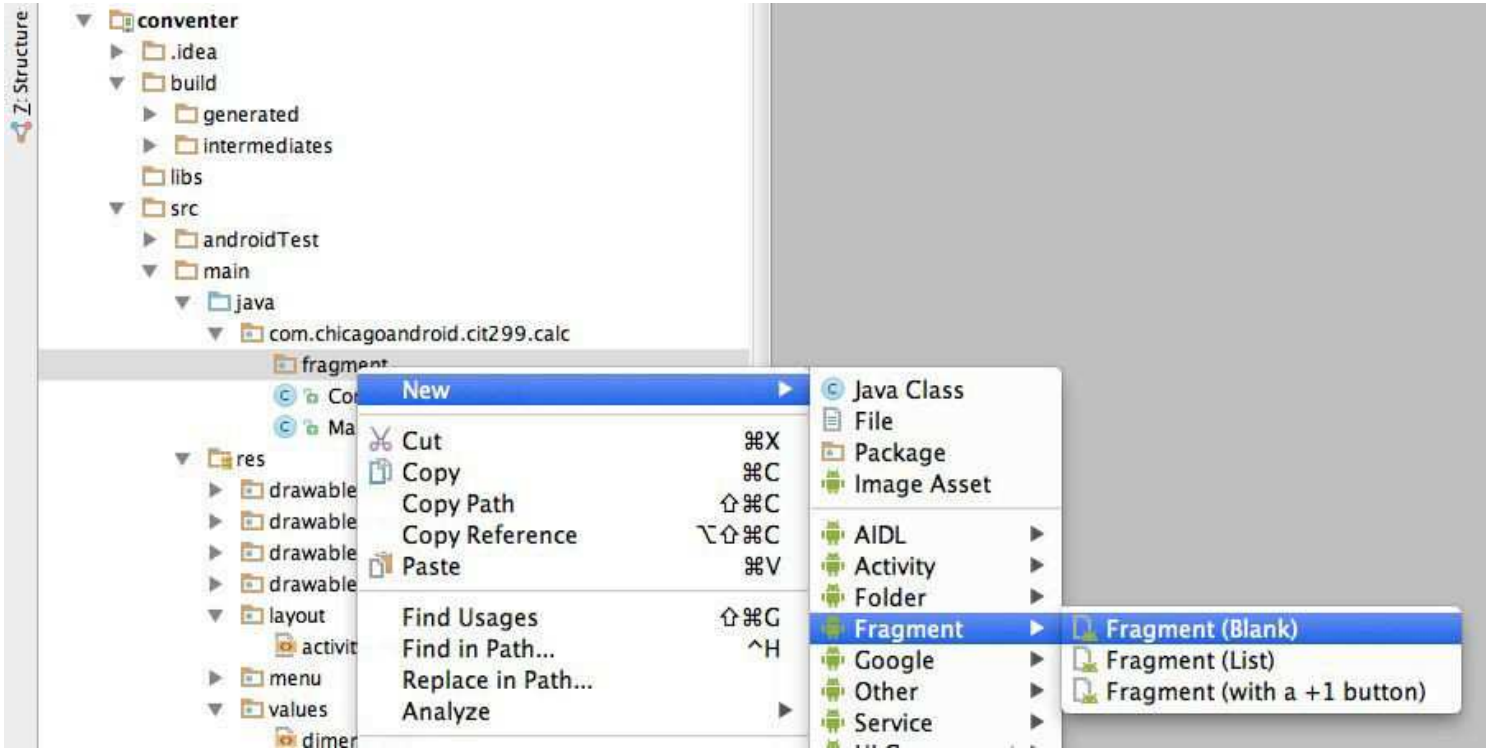
# Fragments

- **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`



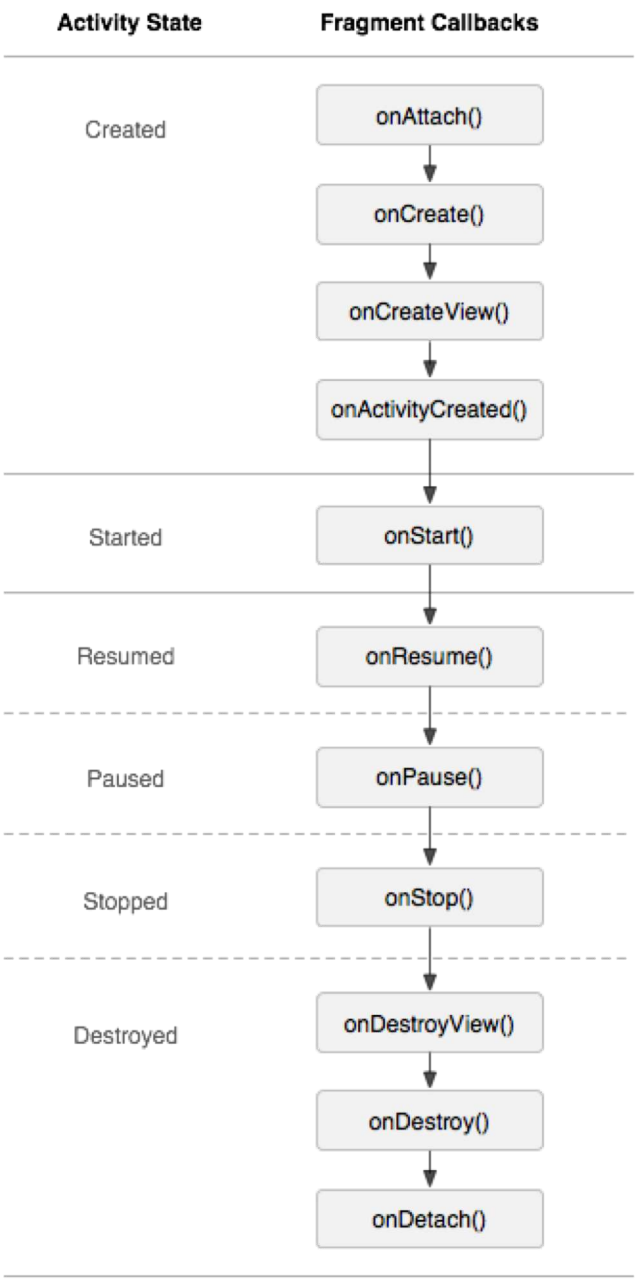
# Using fragments in activity XML

- Activity layout XML can include fragments.

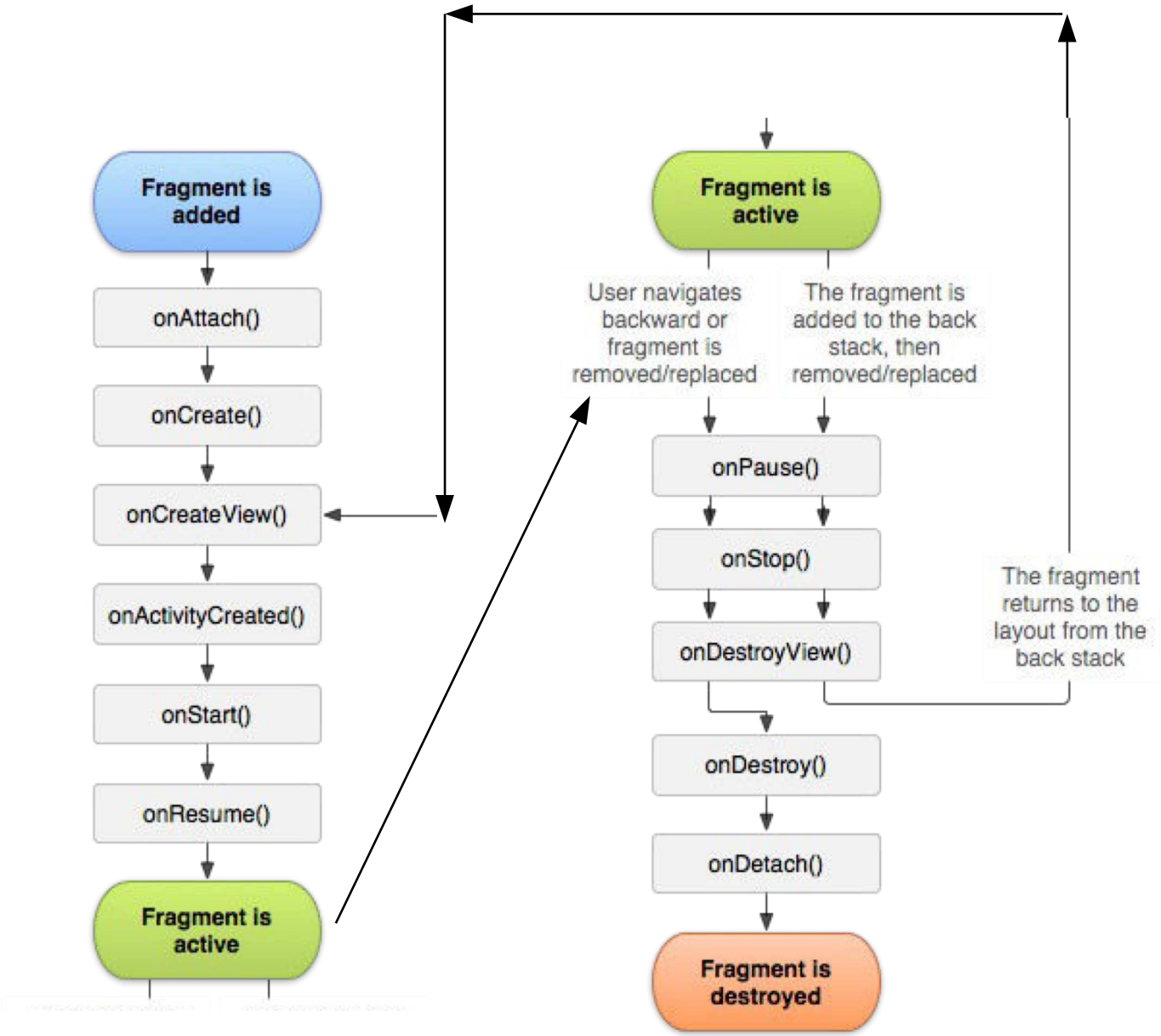
```
<!-- activity_name.xml -->
<LinearLayout ...>
    <fragment ...
        android:id="@+id/id1"
        android:name="ClassName1"
        tools:layout="@layout/name1" />
    <fragment ...
        android:id="@+id/id2"
        android:name="ClassName2"
        tools:layout="@layout/name2" />
</LinearLayout>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

# 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 template

```
public class Name extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater,
            ViewGroup vg, Bundle bundle) {
        // load the GUI layout from the XML
        return inflater.inflate(R.layout.id, vg, false);
    }

    public void onActivityCreated(Bundle savedState) {
        super.onActivityCreated(savedState);
        // ... any other GUI initialization needed
    }

    // any other code (e.g. event-handling)
}
```

# 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 view the fragment is inside of.

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

  - Sometimes also use `getActivity` to refer to the activity itself
  - Event handlers cannot be attached 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`.

# Fragment onClick listener

- Activity:

```
<Button android:id="@+id/b1"
        android:onClick="onClickB1" ... />
```

- Fragment:

```
<Button android:id="@+id/b1" ... />

// in fragment's Java file
Button b = (Button) getView().findViewById(r.id.b1);
b.setOnClickListener(new View.OnClickListener() {
    @Override public void onClick(View view) {
        // whatever code would have been in onClickB1
    }
});
```

# Activity that accepts parameters

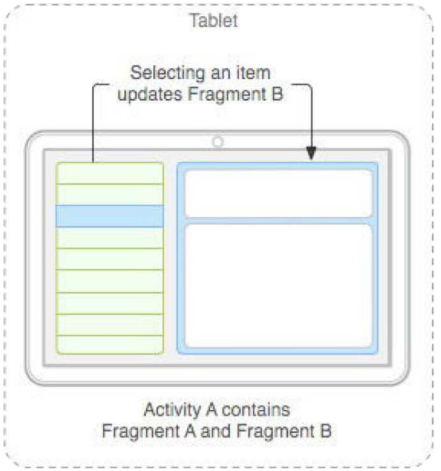
```
public class Name extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.name);

        // extract parameters passed to activity from intent
        Intent intent = getIntent();
        int name1 = intent.getIntExtra("id1", default);
        String name2 = intent.getStringExtra("id2", "default");

        // use parameters to set up the initial state
        ...
    }
    ...
}
```

# Fragment that accepts parameters

```
public class Name extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater,
            ViewGroup container, Bundle savedInstanceState) {
        return inflater.inflate(R.layout.name, container, false);
    }

    @Override
    public void onActivityCreated(Bundle savedState) {
        super.onActivityCreated(savedState);

        // extract parameters passed to activity from intent
        Intent intent = getActivity().getIntent();
        int name1 = intent.getIntExtra("id1", default);
        String name2 = intent.getStringExtra("id2", "default");

        // use parameters to set up the initial state
        ...
    }
```

# Communication between fragments

- One activity might contain multiple fragments.
- The fragments may want to talk to each other.
  - Use activity's `getFragmentManager` method.
  - its `findFragmentById` method can access any fragment that has an id.




```
FragmentClass fragment = (FragmentClass)
getFragmentManager().findFragmentById(R.id.id);
fragment.methodName(parameters);
```

# Fragment subclasses

- `DialogFragment` - a fragment meant to be shown as a dialog box that pops up on top of the current activity.
- `ListFragment` - a fragment that shows a list of items as its main content.
- `PreferenceFragment` - a fragment whose main content is meant to allow the user to change settings for the app.