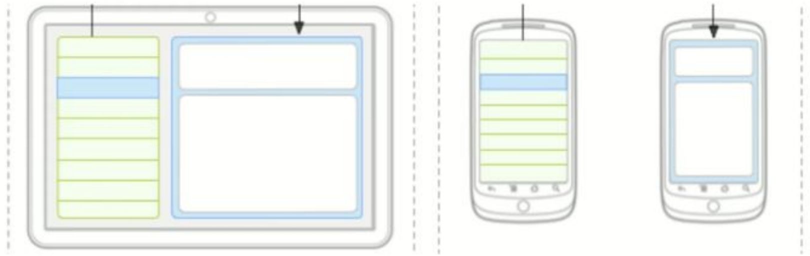
Fragments

Lecture 9

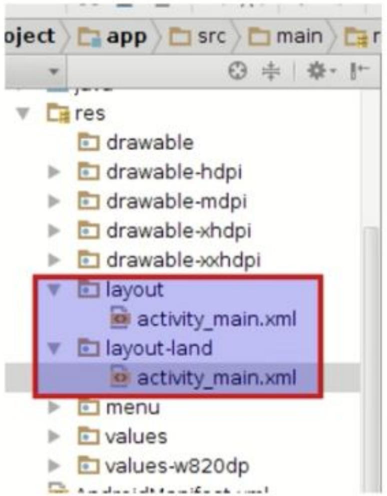
Situational Layouts

- 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.)



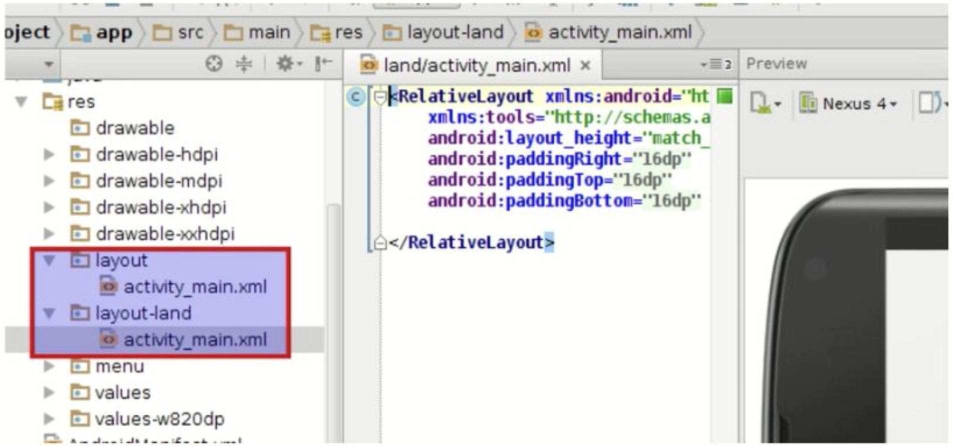
Situation-specific folders

- 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
  - when phone is rotated, activity reloads itself with **layout-land** version

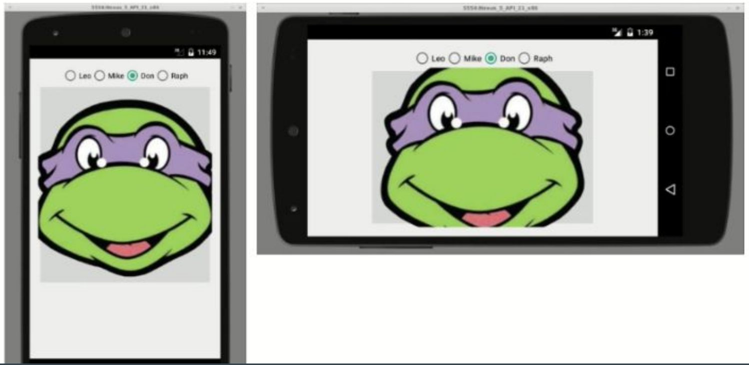


```
<RelativeLayout xmlns:android="ht
    xmlns:tools="http://schemas.a
    android:layout_height="match_
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"

</RelativeLayout>
```

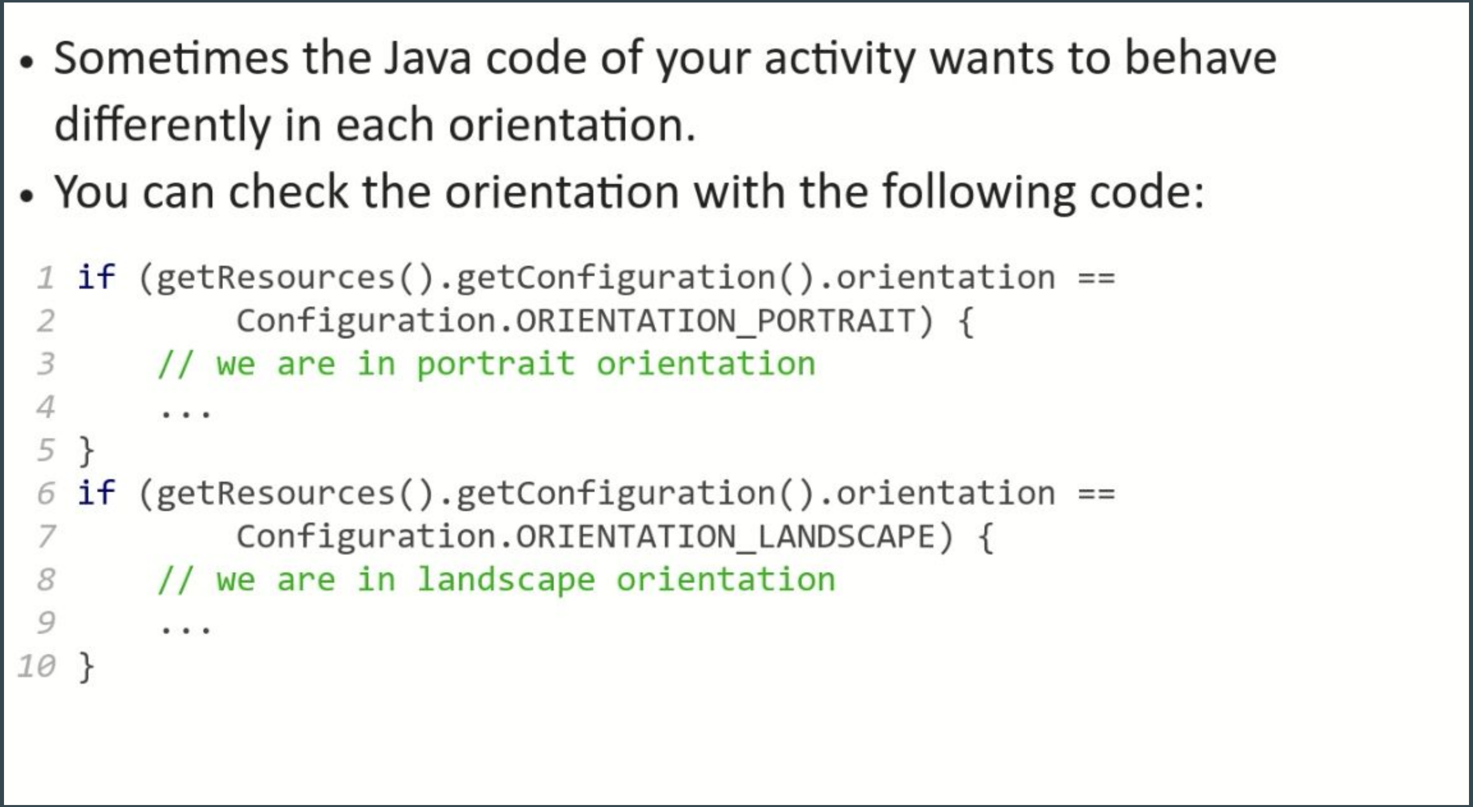
Stop Rotation Layout Reload

- A quick way to retain your activity's GUI state on rotation is to set the `configChanges` attribute of the activity in **AndroidManifest.xml**.
  - Won't reload layout from **layout-land** folder

```
<!-- AndroidManifest.xml -->
<activity android:name=".MainActivity"
    android:configChanges="orientation|screenSize"
    ...>
```



Checking Orientation in Java

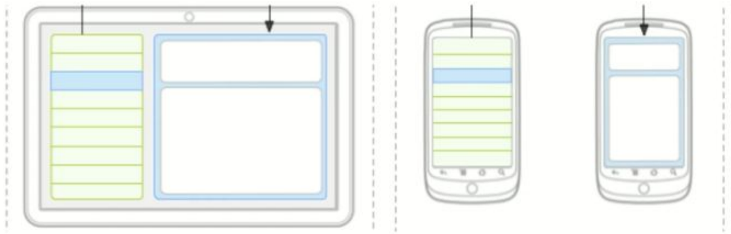


- Sometimes the Java code of your activity wants to behave differently in each orientation.
- You can check the orientation with the following code:

```java
if (getResources().getConfiguration().orientation ==
            Configuration.ORIENTATION_PORTRAIT) {
    // we are in portrait orientation
    ...
}
if (getResources().getConfiguration().orientation ==
            Configuration.ORIENTATION_LANDSCAPE) {
    // we are in landscape orientation
    ...
}
```

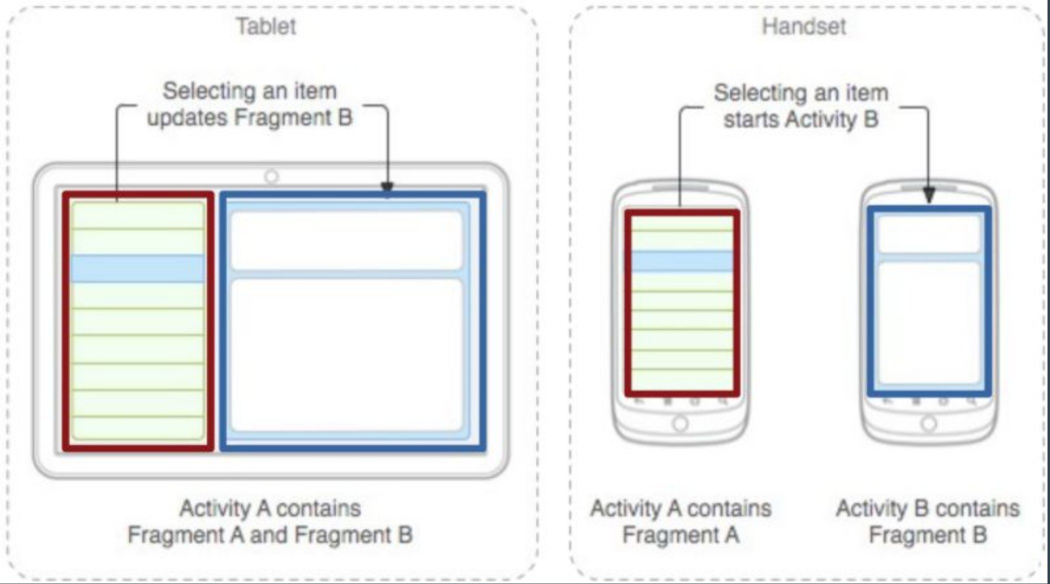
Problem: redundant layouts

- 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, e.g.
  - In **landscape** mode, clicking a button should modify an existing **view**.
  - In **portrait** mode, clicking a button should launch a new **activity**.



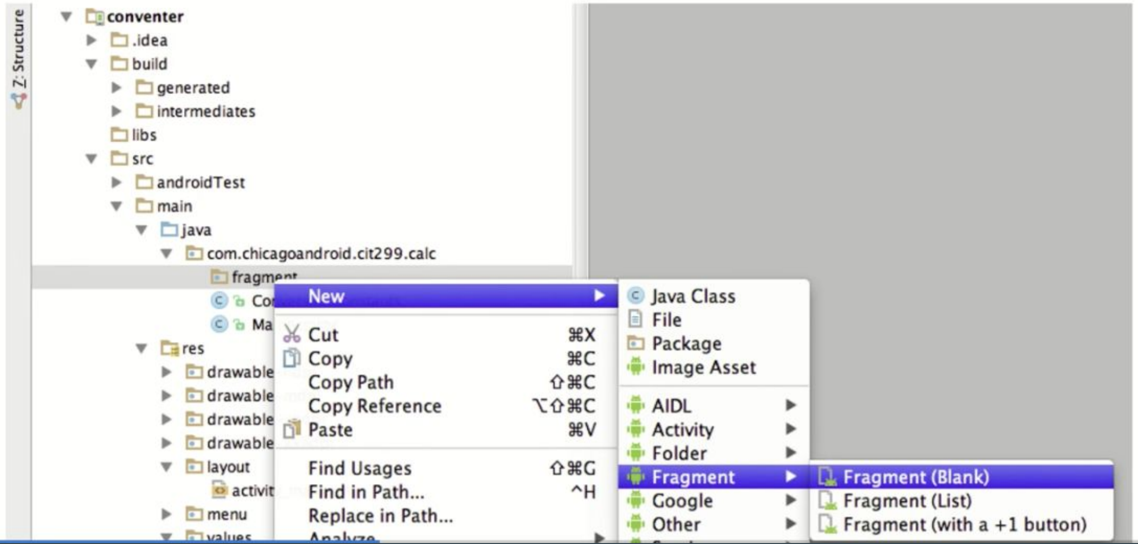
Fragments

- **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)*

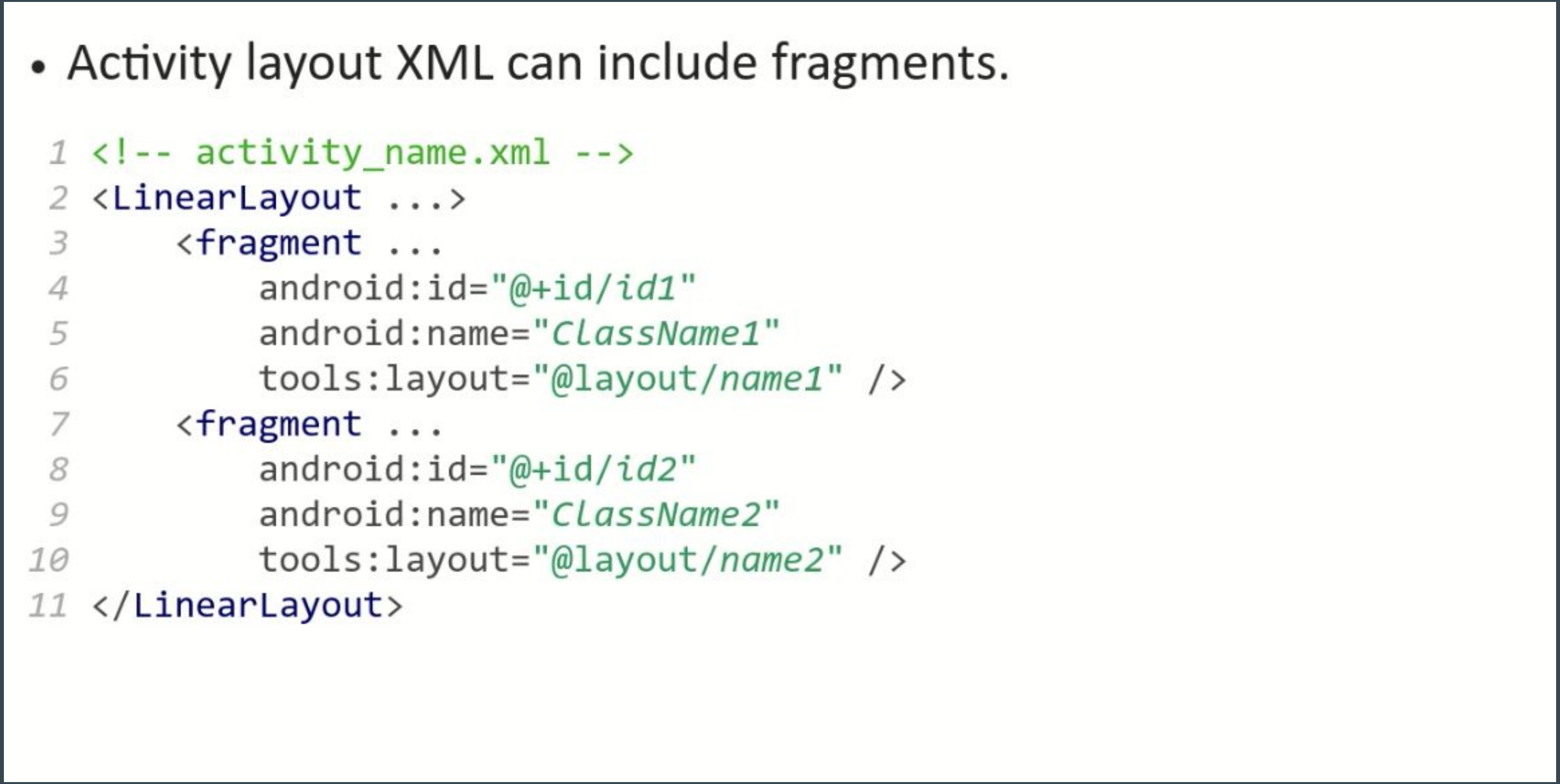

Tablet
Selecting an item updates Fragment B
Activity A contains Fragment A and Fragment B
Handset
Selecting an item starts Activity B
Activity A contains Fragment A
Activity B contains Fragment B

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

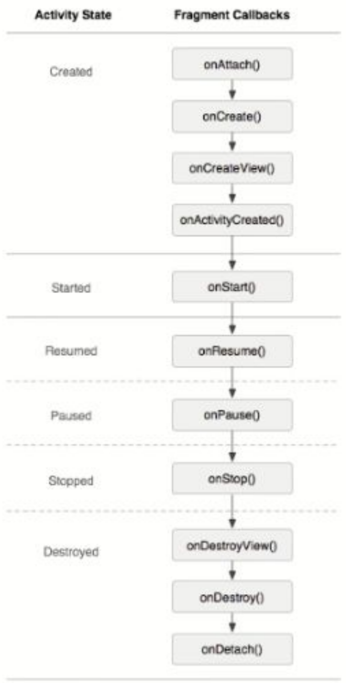


- Activity layout XML can include fragments.

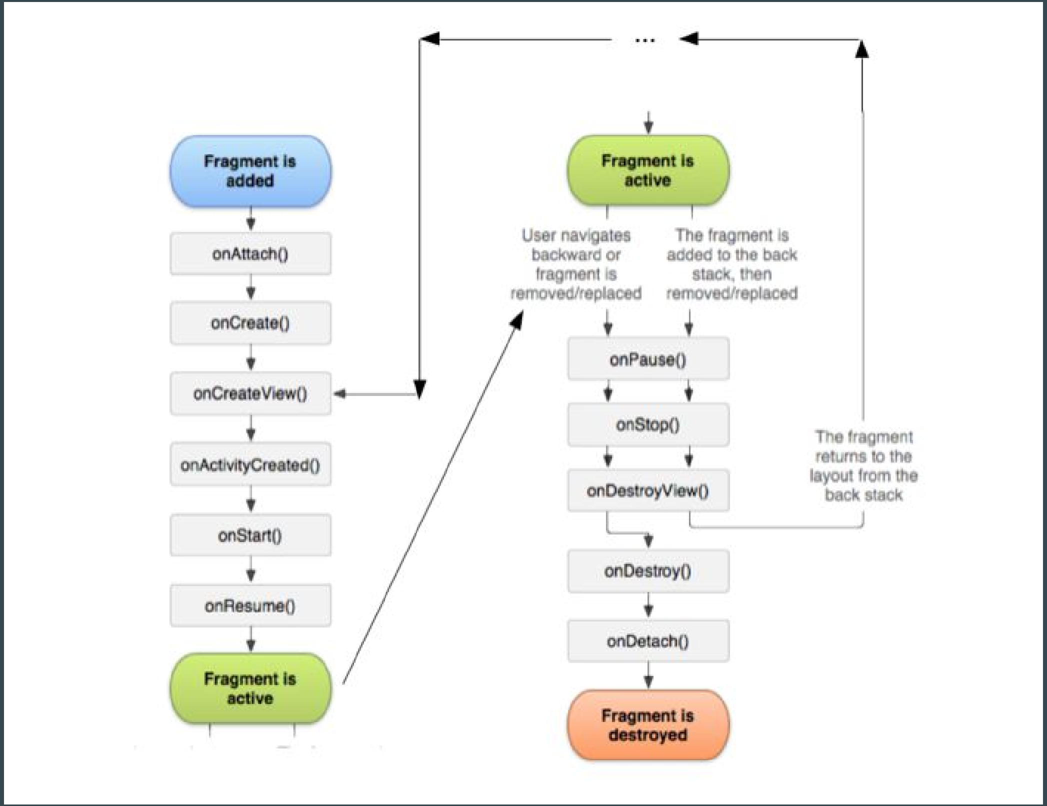
```xml
<!-- activity_name.xml -->
<LinearLayout ...>
    <fragment ...
        android:id="@+id/id1"
        android:name="ClassName1"
        tools:layout="@layout/name1" />
    <fragment ...
        android:id="@+id/id2"
        android:name="ClassName2"
        tools:layout="@layout/name2" />
</LinearLayout>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

Fragment lifecycle

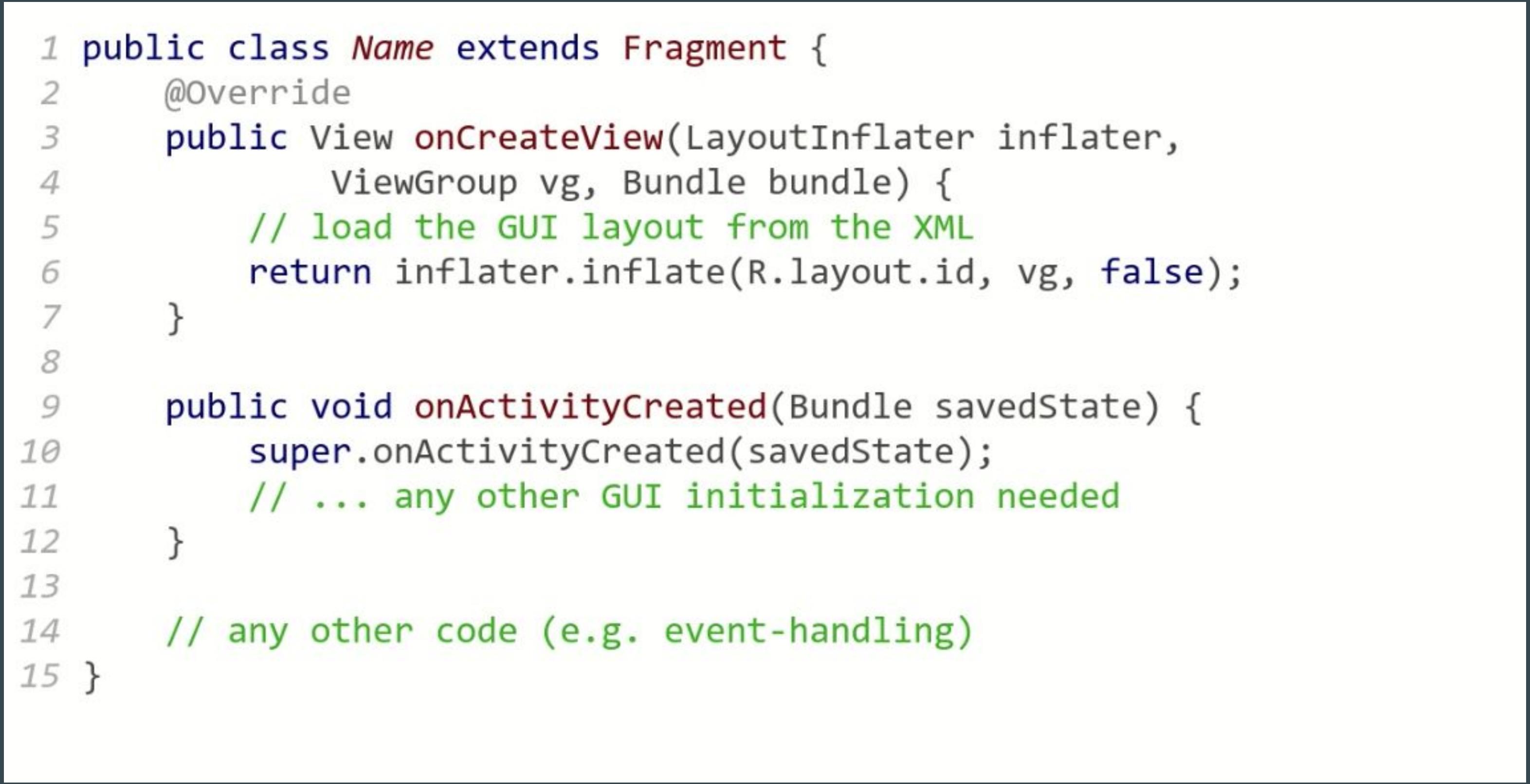
- Fragments have a similar life cycle and events as activities.
  - onAttach to glue fragment to its surrounding activity
  - onCreate when fragment is loading
  - **onCreateView** method that must return fragment's root UI view
  - **onActivityCreated** method that indicates the enclosing activity is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



Fragment lifecycle

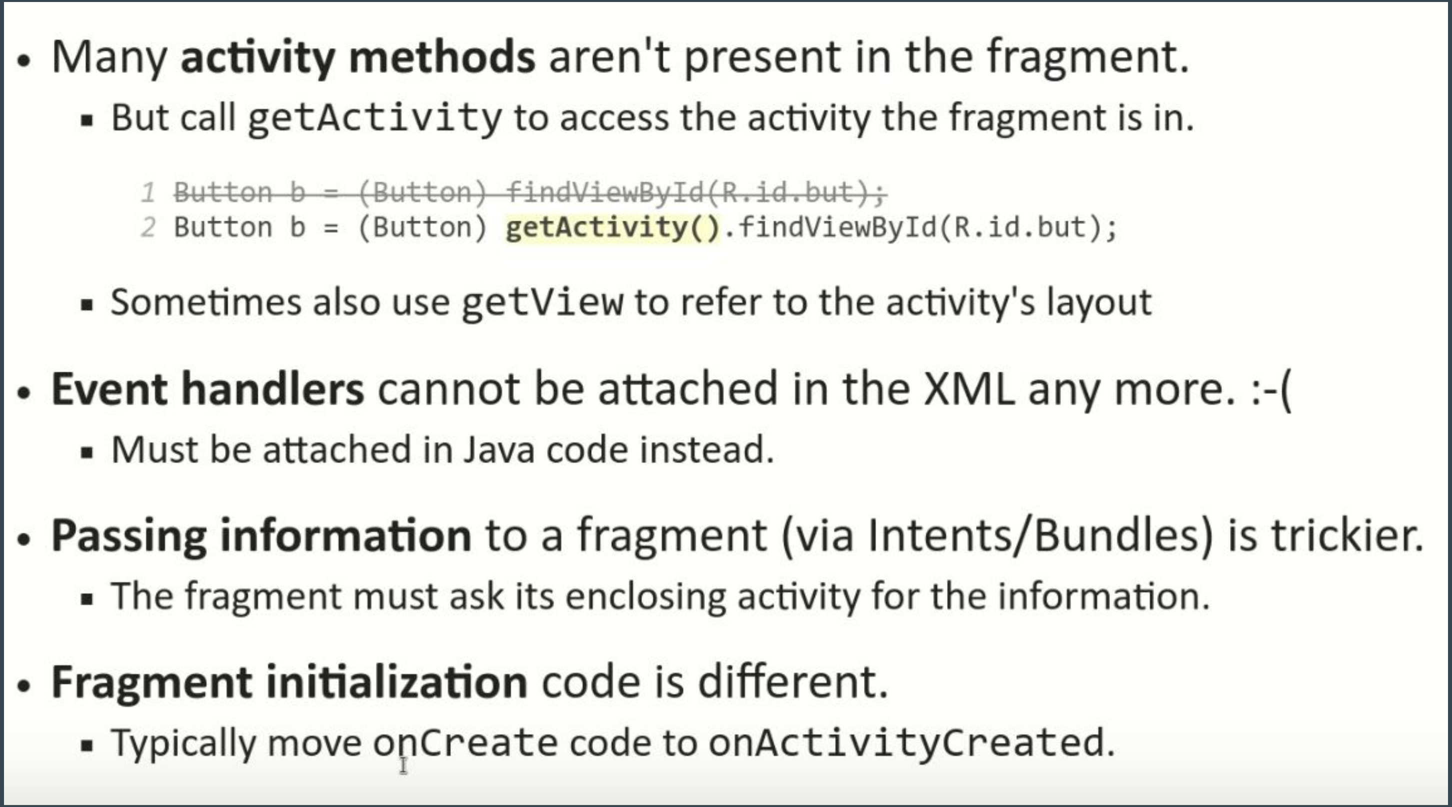


Fragment Template



```java
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

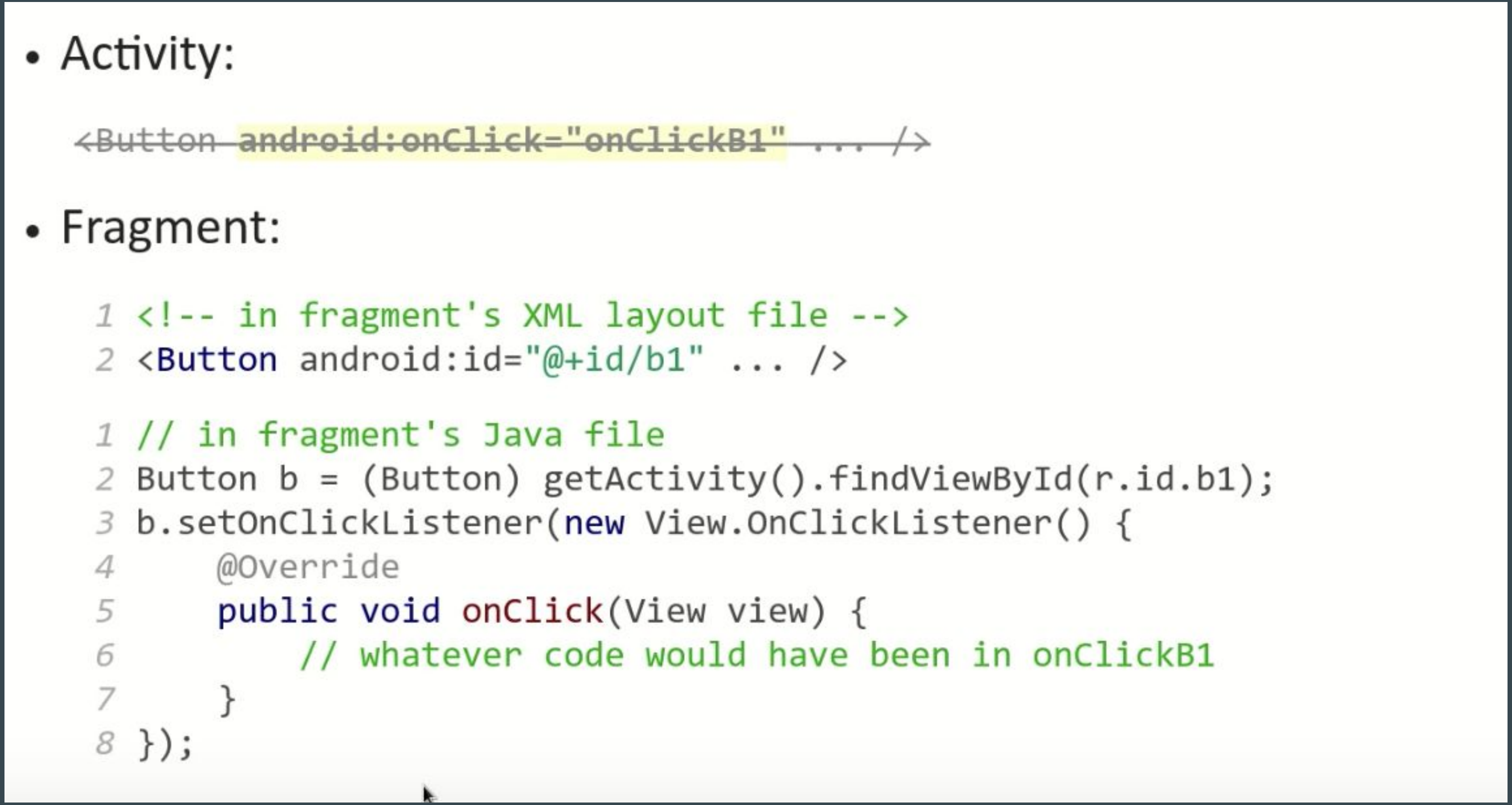


- Many **activity methods** aren't present in the fragment.
  - But call `getActivity` to access the activity the fragment is in.

```
~~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 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 is different.
  - Typically move `onCreate` code to `onActivityCreated`.

Fragment onClick Listener



- Activity:

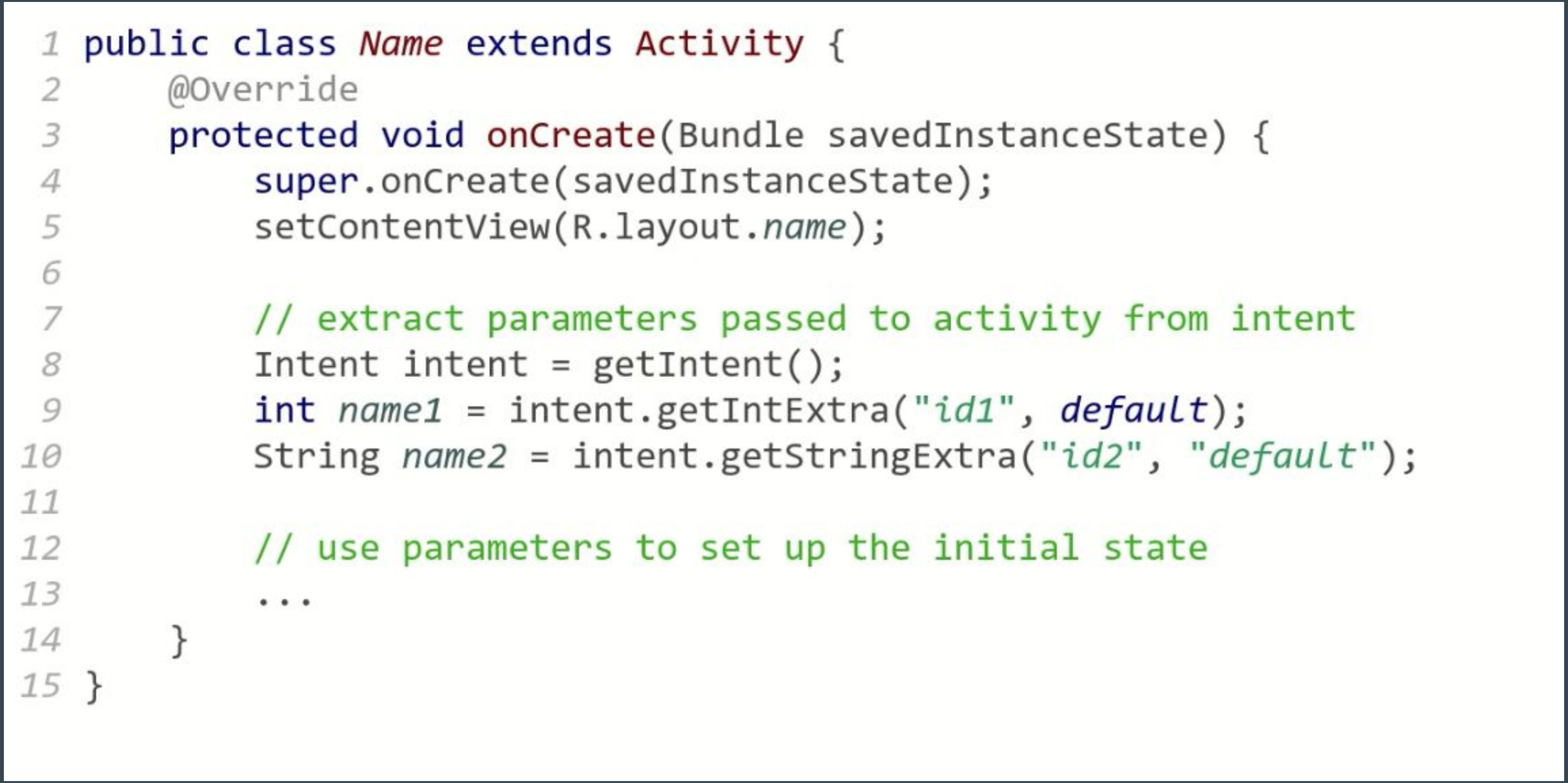
~~<Button android:onClick="onClickB1" ... />~~

- Fragment:

```
<!-- in fragment's XML layout file -->
<Button android:id="@+id/b1" ... />
```

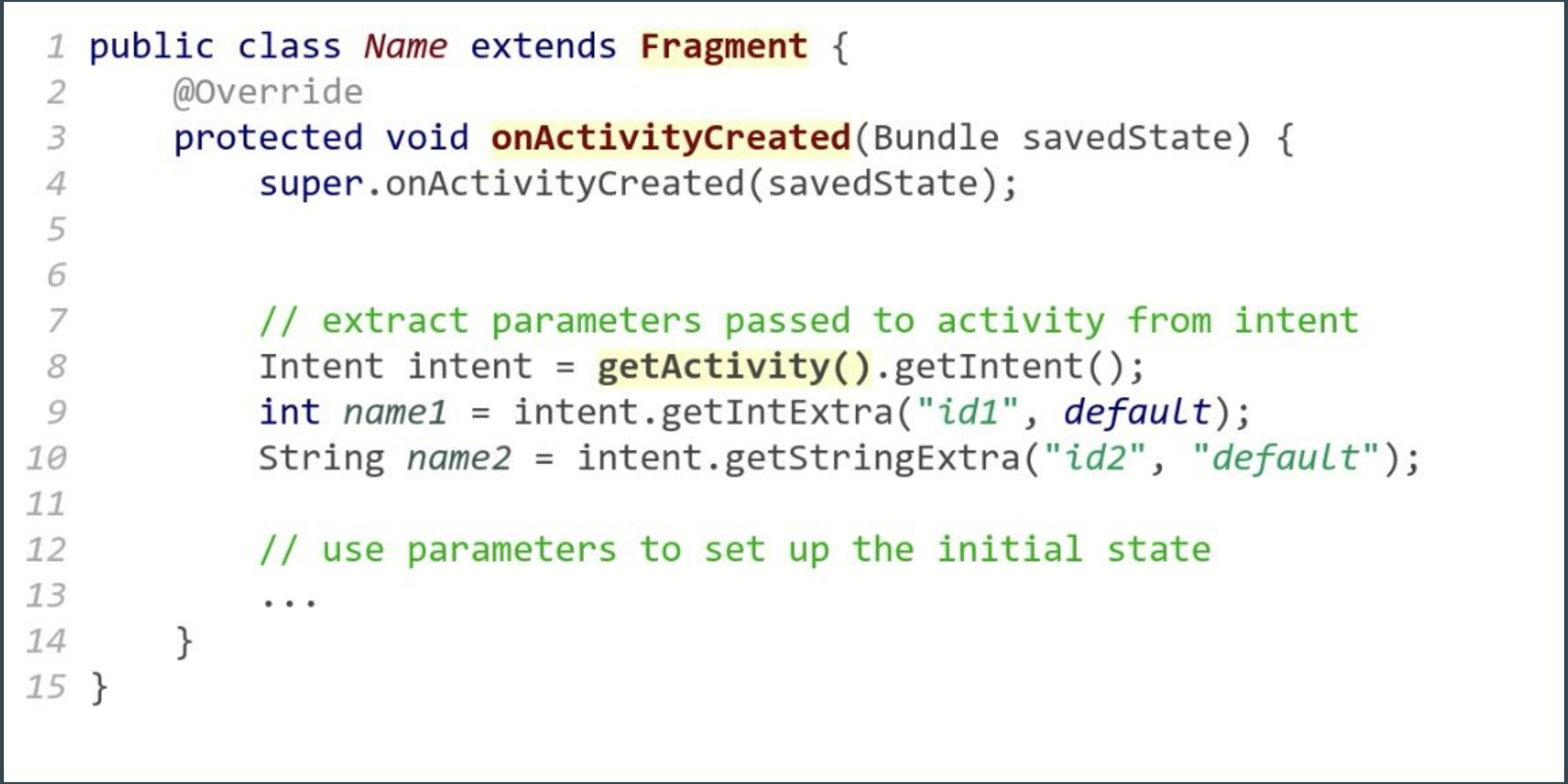
```
// in fragment's Java file
Button b = (Button) getActivity().findViewById(r.id.b1);
b.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // whatever code would have been in onClickB1
    }
});
```

Activity with parameters



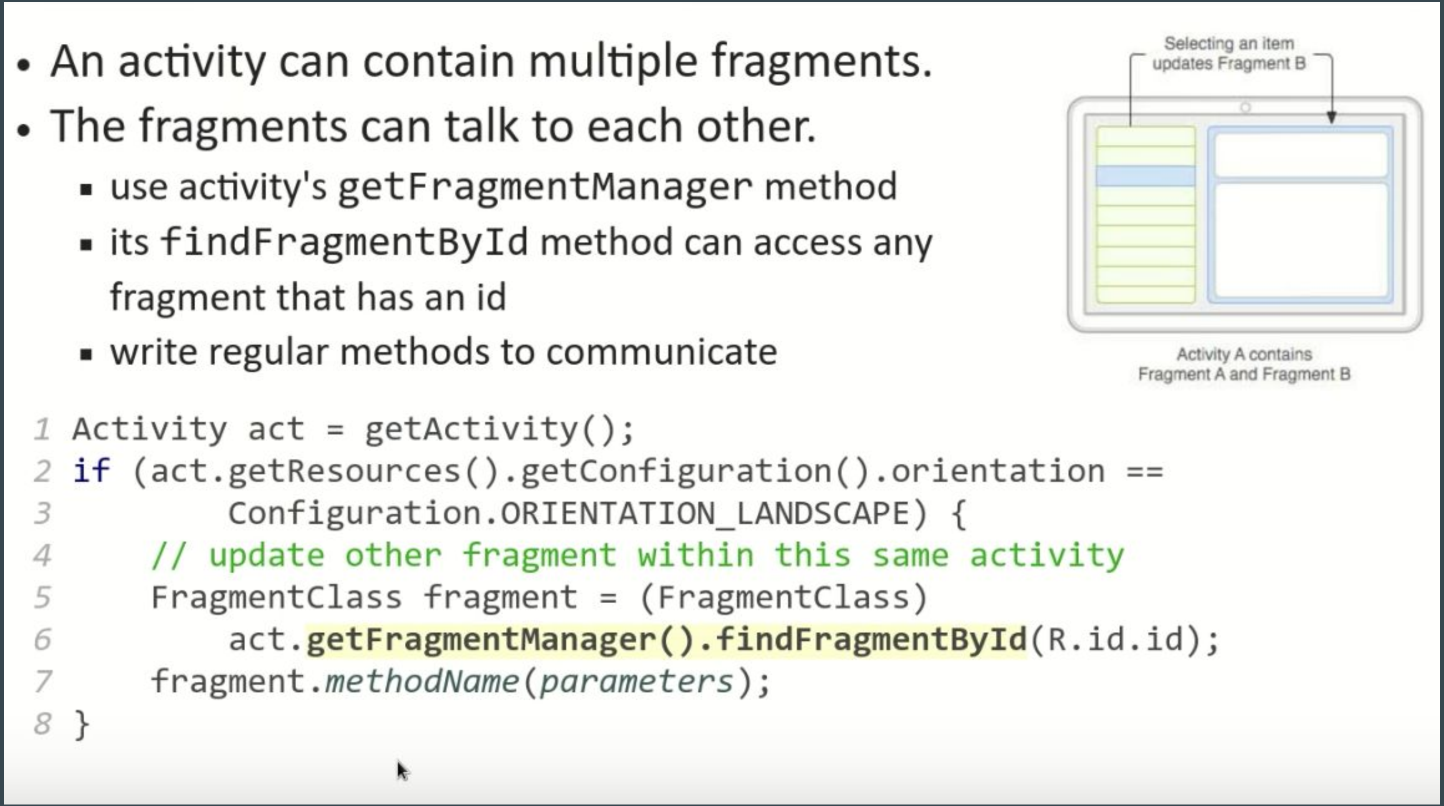
```
public class Name extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.name);

        // 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 with parameters

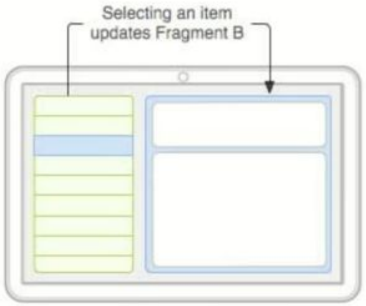


```java
public class Name extends Fragment {
    @Override
    protected void onActivityCreated(Bundle savedState) {
        super.onActivityCreated(savedState);


        // 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
        ...
    }
}
```

Fragment communication



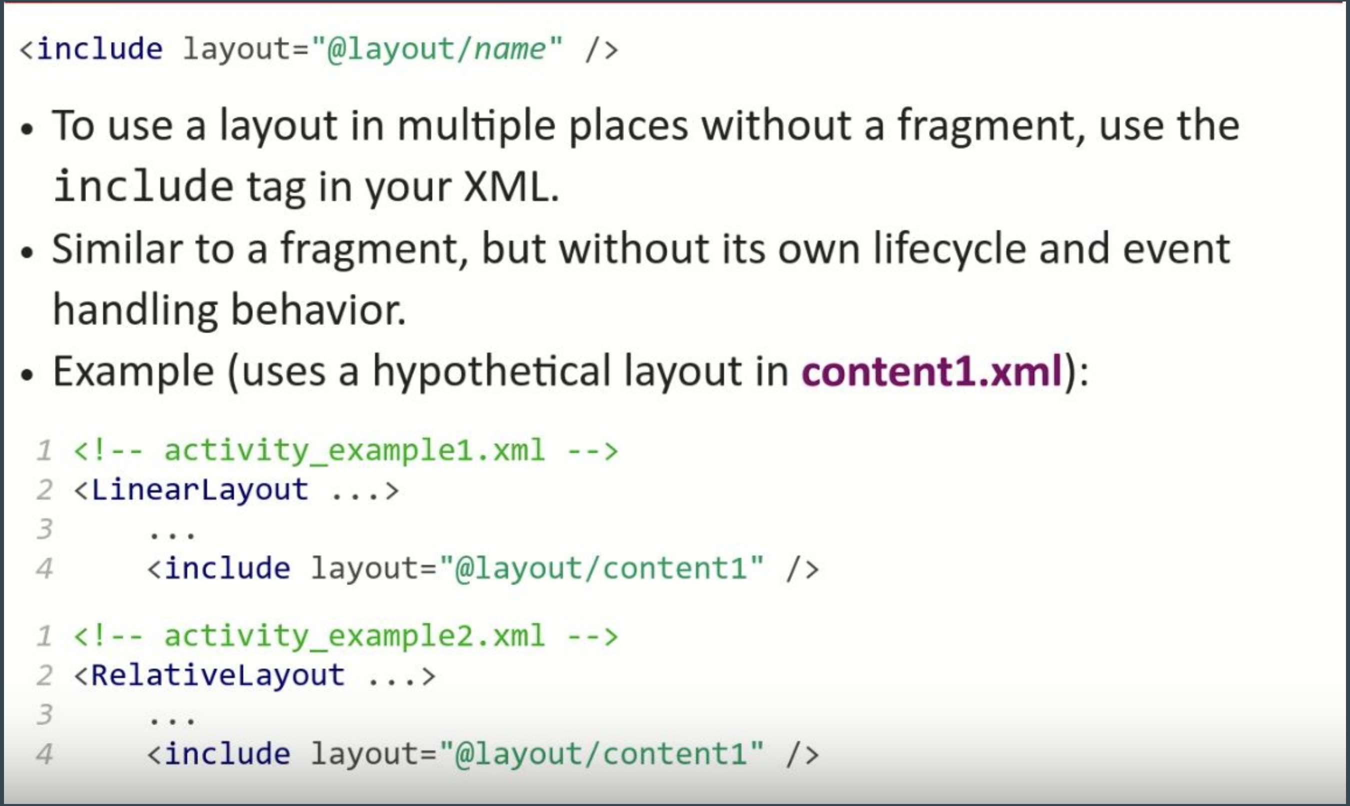
- An activity can contain multiple fragments.
- The fragments can talk to each other.
  - use activity's `getFragmentManager` method
  - its `findFragmentById` method can access any fragment that has an id
  - write regular methods to communicate



Activity A contains Fragment A and Fragment B

```java
Activity act = getActivity();
if (act.getResources().getConfiguration().orientation ==
        Configuration.ORIENTATION_LANDSCAPE) {
    // update other fragment within this same activity
    FragmentClass fragment = (FragmentClass)
        act.getFragmentManager().findFragmentById(R.id.id);
    fragment.methodName(parameters);
}
```

Reusing Layout with include



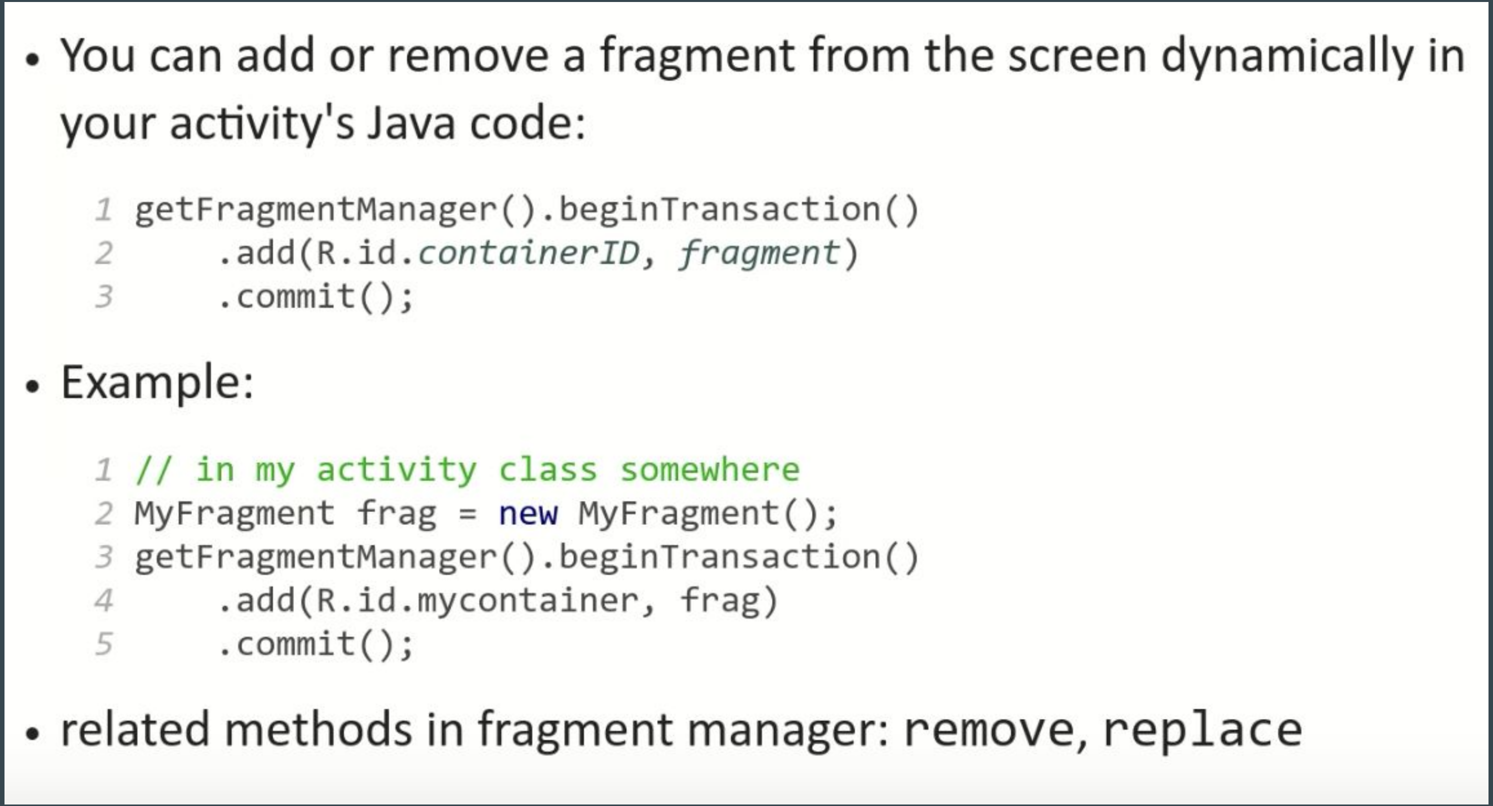
```
<include layout="@layout/name" />
```

- To use a layout in multiple places without a fragment, use the `include` tag in your XML.
- Similar to a fragment, but without its own lifecycle and event handling behavior.
- Example (uses a hypothetical layout in **content1.xml**):

```
<!-- activity_example1.xml -->
<LinearLayout ...>
    ...
    <include layout="@layout/content1" />
```

```
<!-- activity_example2.xml -->
<RelativeLayout ...>
    ...
    <include layout="@layout/content1" />
```

Dynamically add a fragment



- You can add or remove a fragment from the screen dynamically in your activity's Java code:

```java
getFragmentManager().beginTransaction()
    .add(R.id.containerID, fragment)
    .commit();
```

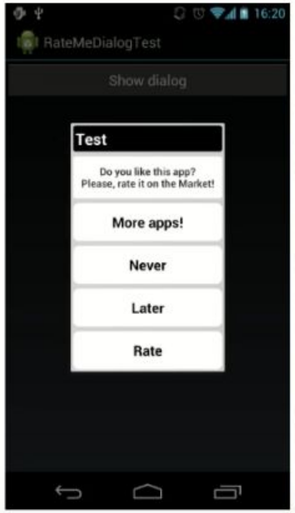
- Example:

```java
// in my activity class somewhere
MyFragment frag = new MyFragment();
getFragmentManager().beginTransaction()
    .add(R.id.mycontainer, frag)
    .commit();
```

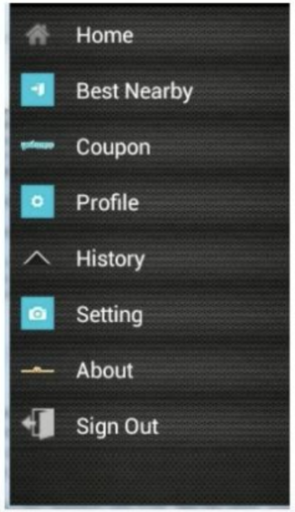
- related methods in fragment manager: `remove`, `replace`

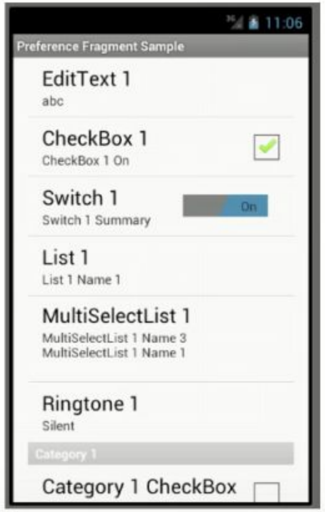
Fragment Subclasses

- `DialogFragment` - Pops up on top of the current activity.
- `ListFragment` - Shows list of items as its main content.
- `PreferenceFragment` - Allows user to change app settings.



Do you like this app?
Please, rate it on the Market!





EditText 1
abc
CheckBox 1 ✓
CheckBox 1 On
Switch 1 On
Switch 1 Summary
List 1
List 1 Name 1
MultiSelectList 1
MultiSelectList 1 Name 3
MultiSelectList 1 Name 1
Ringtone 1
Silent
Category 1 CheckBox ☐