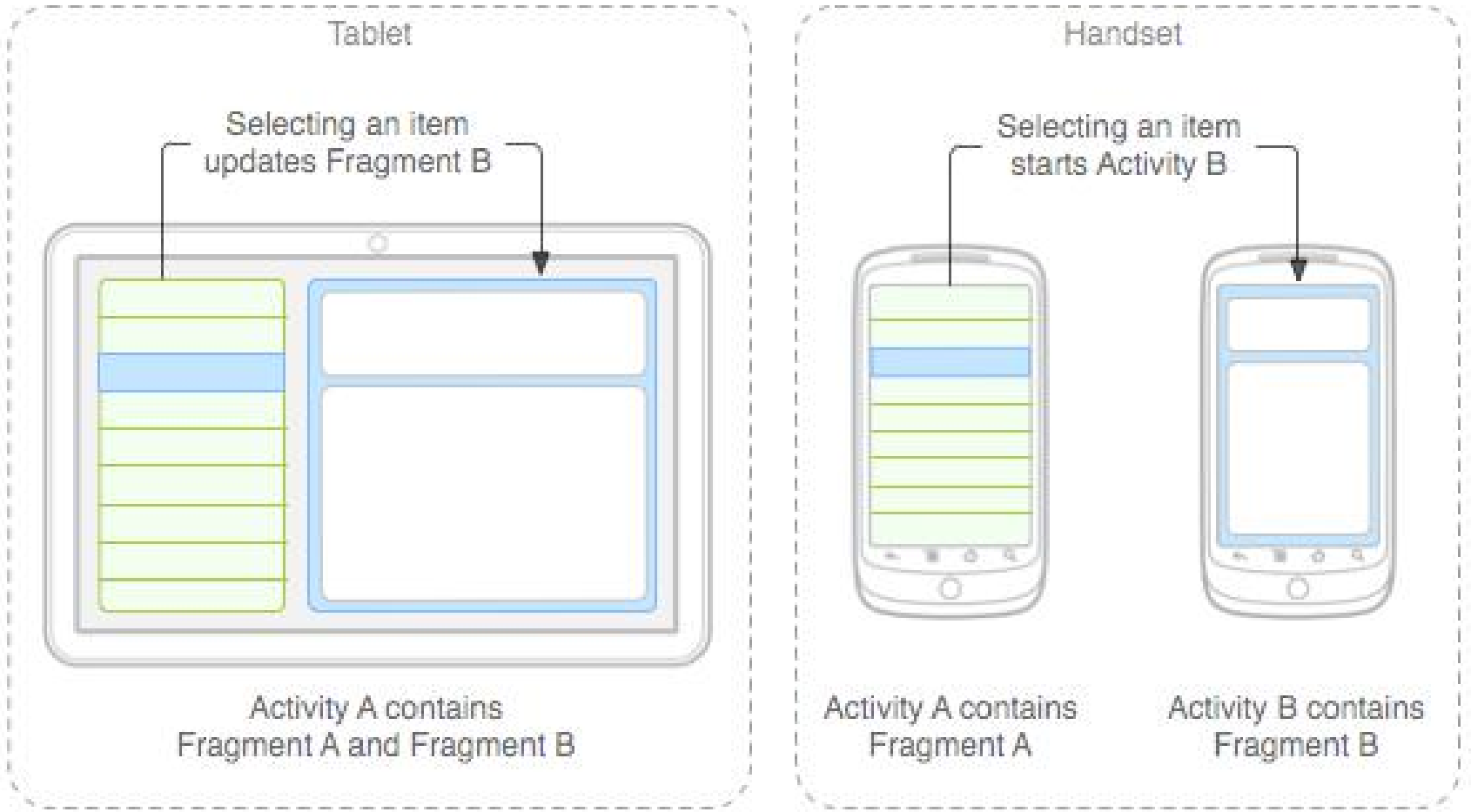
Lecture 7: Fragments, and Adaptive Layouts

Fragments

Fragment

- A Fragment represents a behavior or a portion of user interface in an Activity.
- You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities.
- You can think of a fragment as a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running (sort of like a "sub activity" that you can reuse in different activities).


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

Fragment

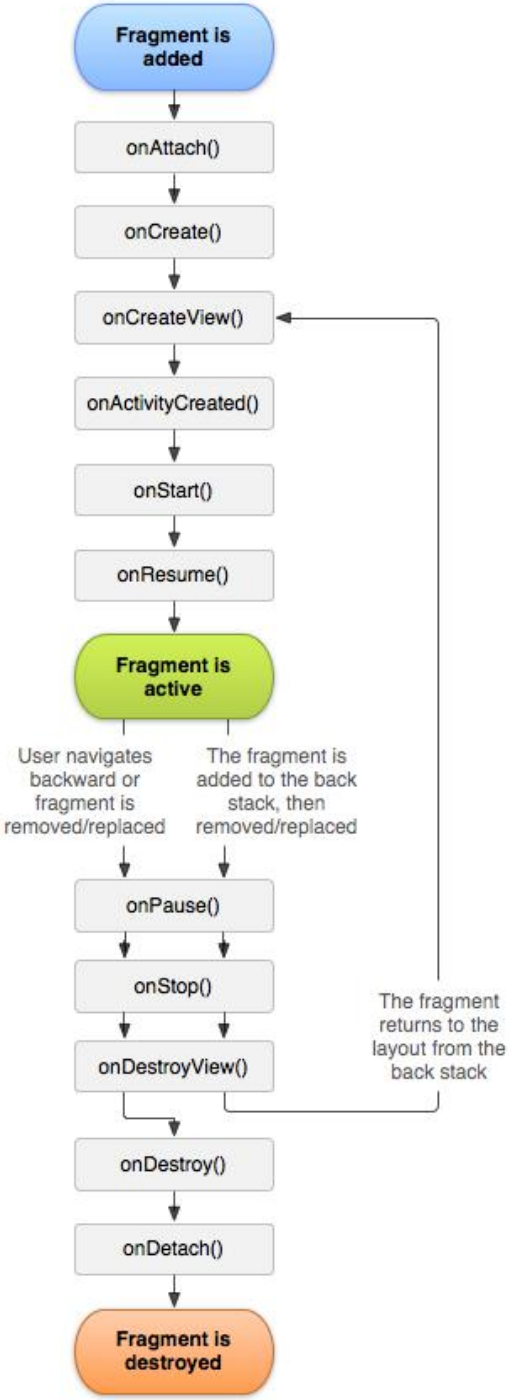
- When you have a larger screen device than a phone –like a tablet it can look too simple to use phone interface here.
- Fragments
  - Mini-activities, each with its own set of views
  - One or more fragments can be embedded in an Activity
  - You can do this dynamically as a function of the device type (tablet or not) or orientation

How to Use Fragments

- First of all decide how many fragments you want to use in an activity. For example let's we want to use two fragments to handle landscape and portrait modes of the device.
- Next based on number of fragments, create classes which will extend the *Fragment* class. The Fragment class has above mentioned callback functions. You can override any of the functions based on your requirements.
- Corresponding to each fragment, you will need to create layout files in XML file. These files will have layout for the defined fragments.
- Finally modify activity file to define the actual logic of replacing fragments based on your requirement.

Fragment Lifecycle

- Fragment in an Activity---Activity Lifecyle influences
  - Activity paused all its fragments paused
  - Activity destroyed all its fragments are destroyed
  - Activity running manipulate each fragment independently.
- Fragment transaction add, remove, etc.
  - adds it to a back stack that's managed by the activity—each back stack entry in the activity is a record of the fragment transaction that occurred.
  - The back stack allows the user to reverse a fragment transaction (navigate backwards), by pressing the *Back* button.



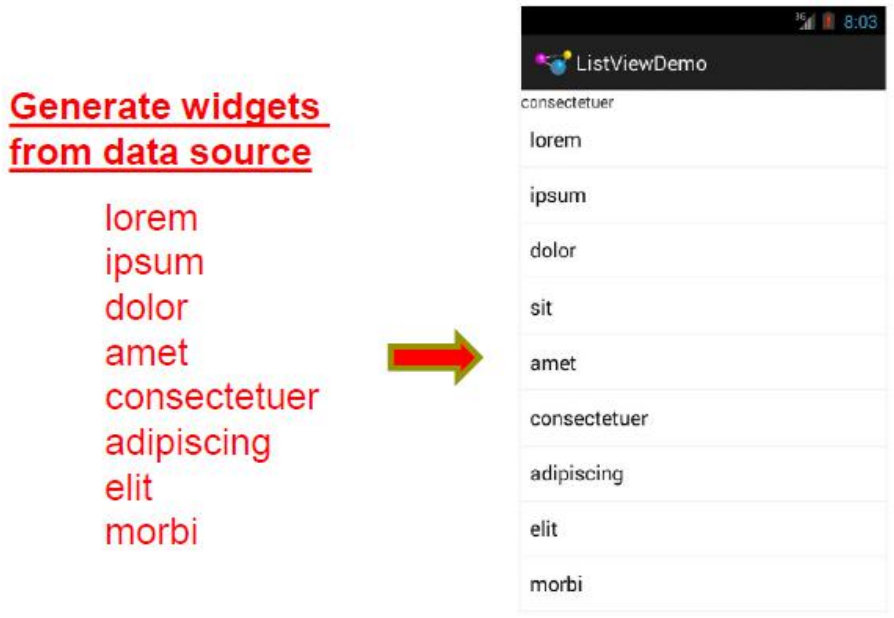
Reading Materials

- Read Chapter 9 Fragments

Data-Driven Layouts

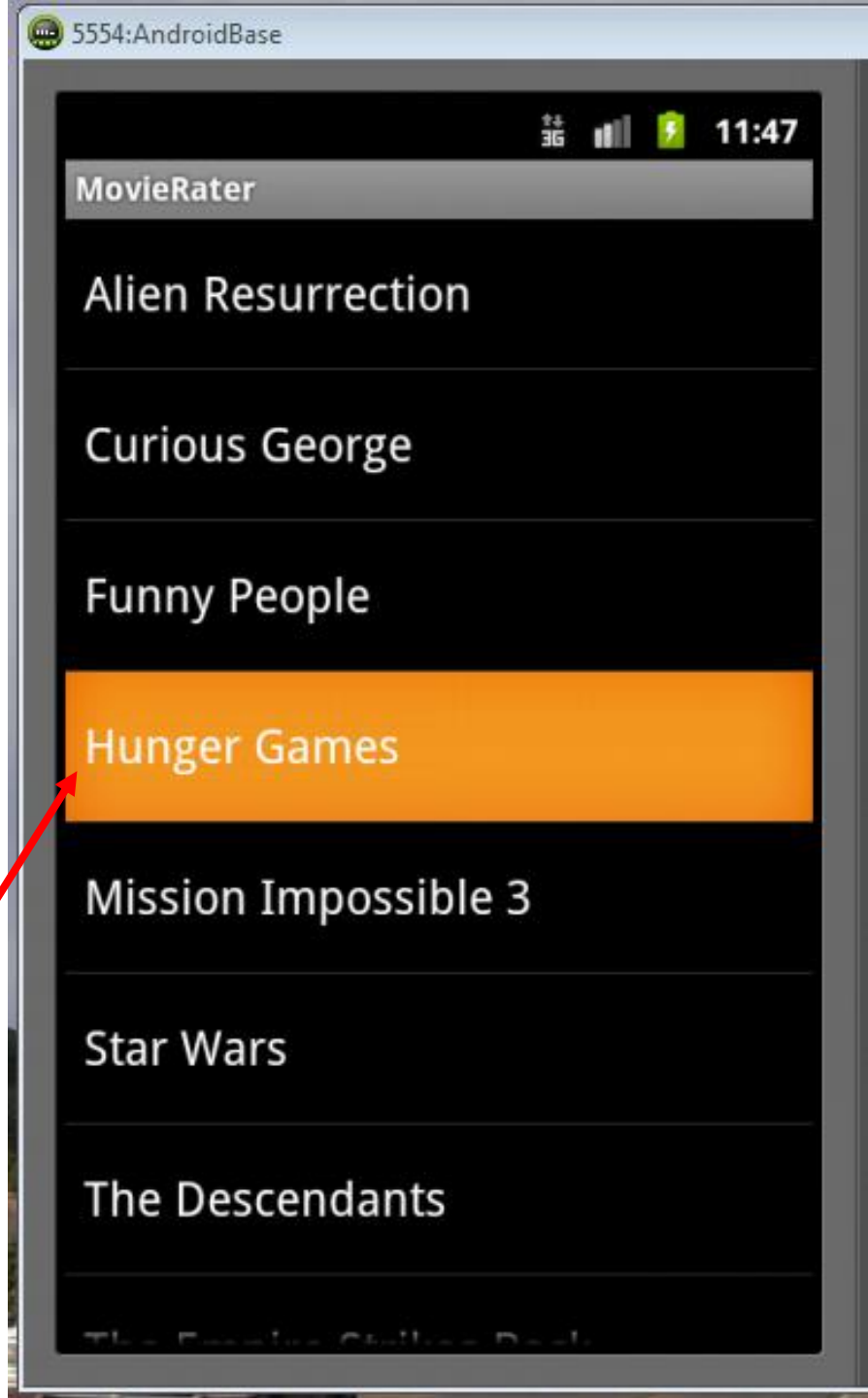
Data-Driven Layouts

- LinearLayout, RelativeLayout, TableLayout, GridLayout useful for positioning UI elements
  - Static preset
- Other layouts dynamically composed from data (e.g. database)
  - ListView, GridView, GalleryView
  - Tabs with TabHost, TabControl



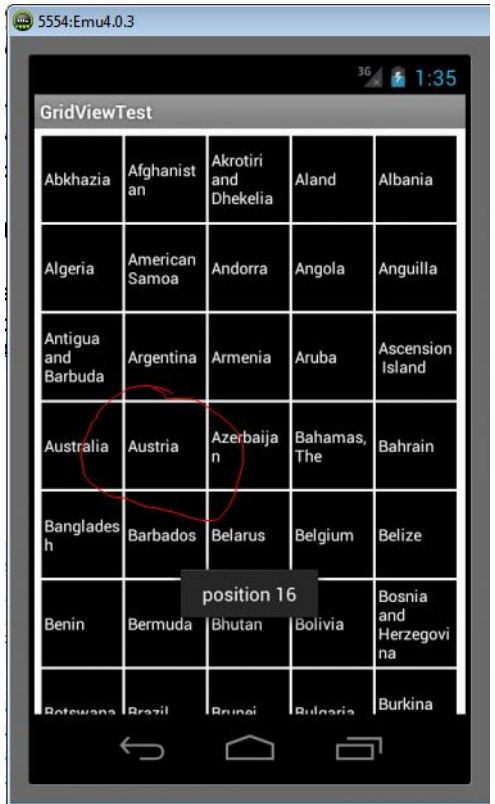
Data Driven Layouts

- May want to populate views from a data source (XML file or database)
- Layouts that display repetitive child widgets from data source
  - ListView
  - GridView
  - GalleryView
- ListView
  - Rows of entries, pick item, vertical scroll

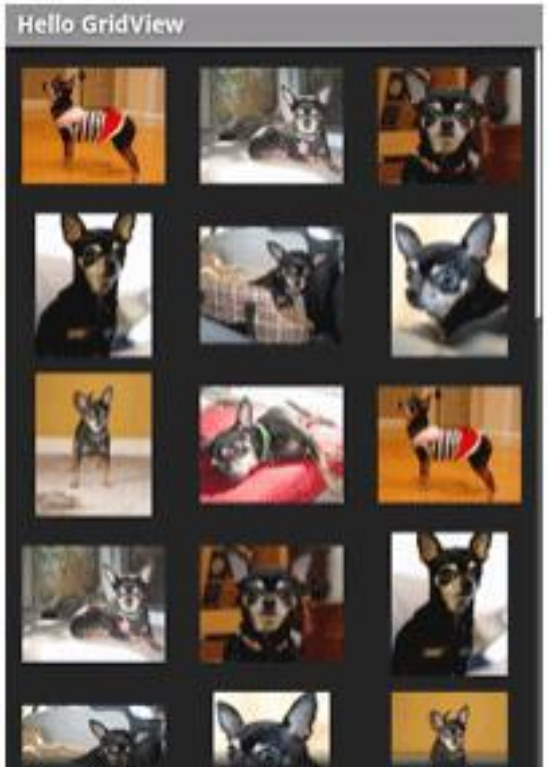


Data Driven Containers

- GridView
  - List of items arranged in rows and columns

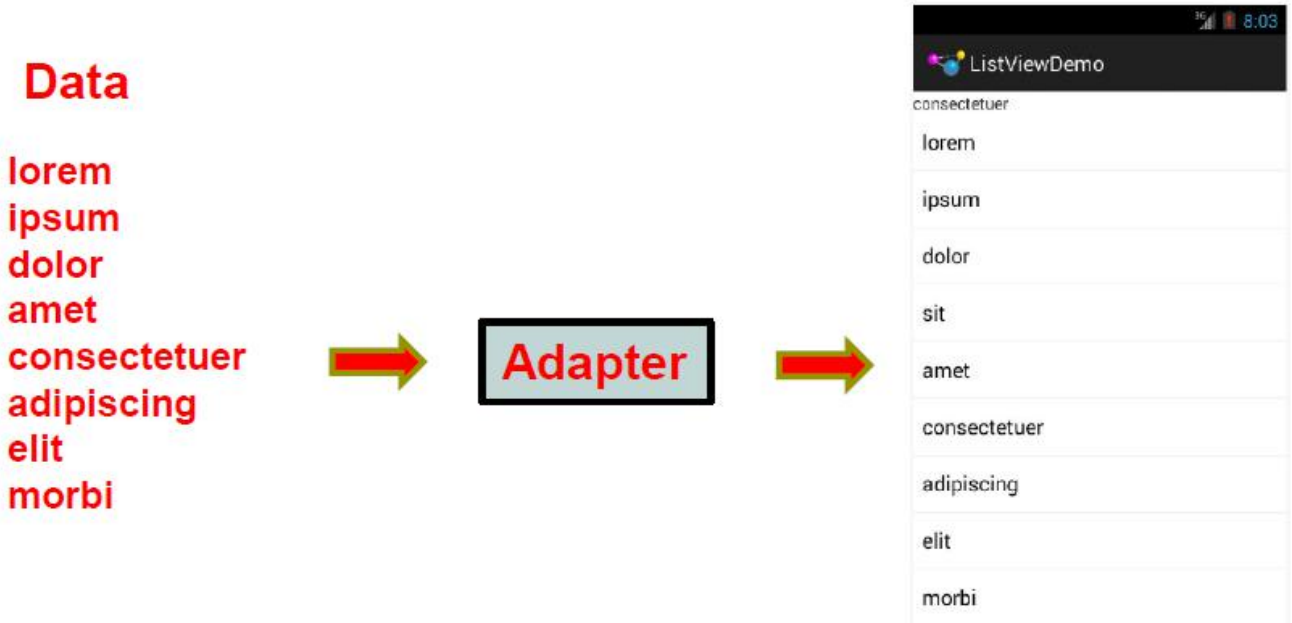


- GalleryView
  - List with horizontal scrolling, typically images



AdapterView

- ListView, GridView, and GalleryView are sub classes of AdapterView (variants)
- **Adapter:** generates widgets from a data source, populates layout
  - E.g. Data is adapted into cells of GridView



AdapterView

- Most common Adapter types:
  - **CursorAdapter:**read from database
  - **ArrayAdapter:**read from resource (e.g. XML file)

Adapters

- When using Adapter, a layout (XML format) is defined for each child element (View)
- The adapter
  - Reads in data (list of items)
  - Creates Views (widgets) using layout for each element in data source
  - Fills the containing layout (List, Grid, Gallery) with the created Views
- Child widgets can be as simple as a TextViewor more complex layouts / controls
  - simple views can be declared in a layout XML file (e.g. android.R.layout)

Example: Creating ListView using AdapterArray

- Goal: create this

```
private static final String[] items={"lorem", "ipsum", "dolor",
        "sit", "amet",
        "consectetuer", "adipiscing", "elit", "morbi", "vel",
        "ligula", "vitae", "arcu", "aliquet", "mollis",
        "etiam", "vel", "erat", "placerat", "ante",
        "porttitor", "sodales", "pellentesque", "augue", "purus"};
```

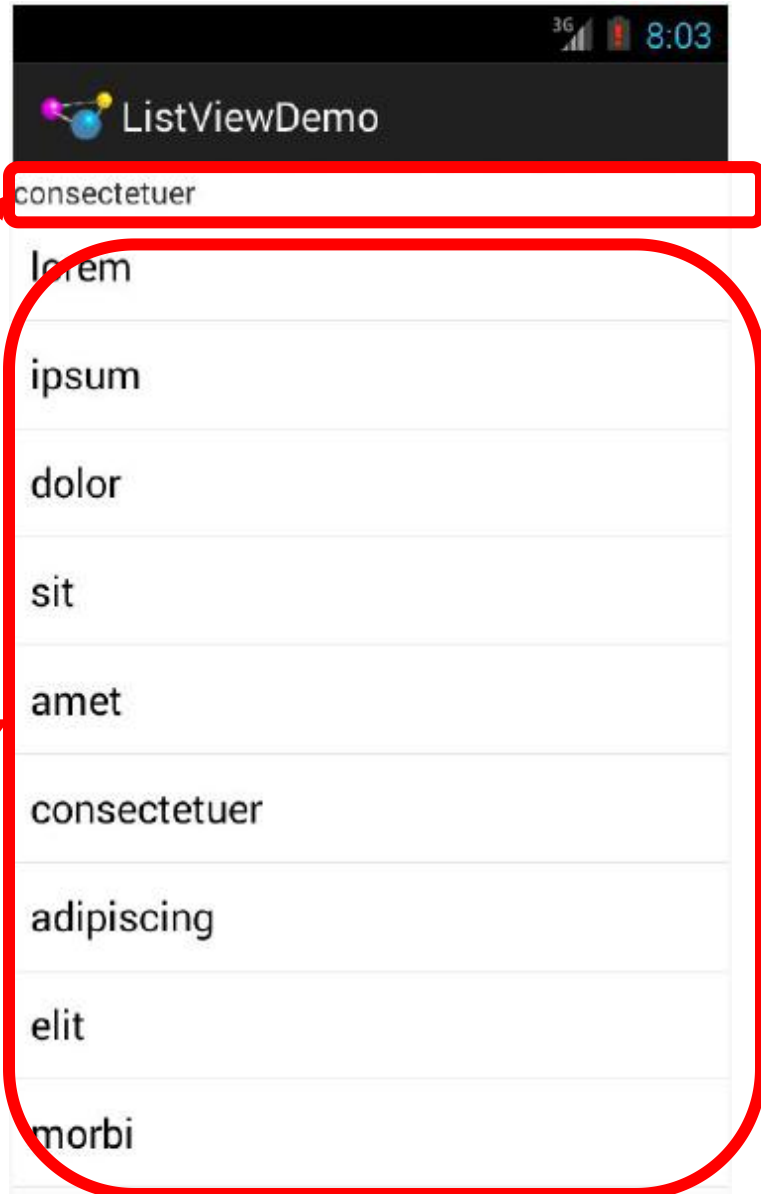
Enumerated list

ListView of items

Example: Creating ListView using AdapterArray

First create Layout file

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.co
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent" >
  <TextView
    android:id="@+id/selection"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>
  <ListView
    android:id="@android:id/list"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    />
</LinearLayout>
```



Using ArrayAdapter

Command used to wrap adapter around array of menu items or **java.util.List** instance

- E.g. **android.R.layout.simple_list_item_1** turns strings into textView widgets

```
String[] items={"this", "is", "a", "really", "silly", "list"};
new ArrayAdapter<String>(this,
                         android.R.layout.simple_list_item_1,
                         items);
```

Context to use.
(e.g app's activity)

Resource ID of
View for formatting

Array of items
to display

```
public class ListViewDemo extends ListActivity {
  private TextView selection;
  private static final String[] items={"lorem", "ipsum", "dolor",
          "sit", "amet",
          "consectetuer", "adipiscing", "elit", "morbi", "vel",
          "ligula", "vitae", "arcu", "aliquet", "mollis",
          "etiam", "vel", "erat", "placerat", "ante",
          "porttitor", "sodales", "pellentesque", "augue", "purus"};

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.main);
    setListAdapter(new ArrayAdapter<String>(this,
                        android.R.layout.simple_list_item_1,
                        items));
    selection=(TextView)findViewById(R.id.selection);
  }

  @Override
  public void onListItemClick(ListView parent, View v, int position,
                              long id) {
    selection.setText(items[position]);
  }
}
```

Set list adapter (Bridge Data source and views)

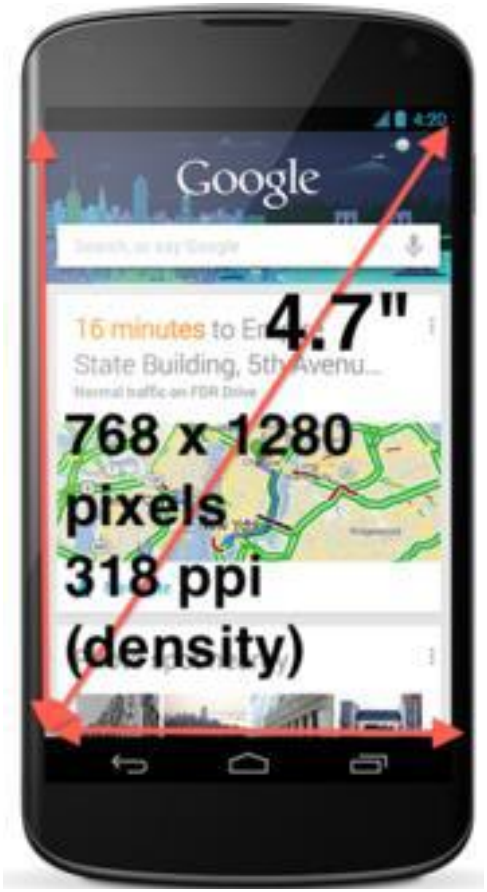
Get handle to TextView of Selected item

Change Text at top to that of selected view when user clicks on selection

Adaptive Image Resolution

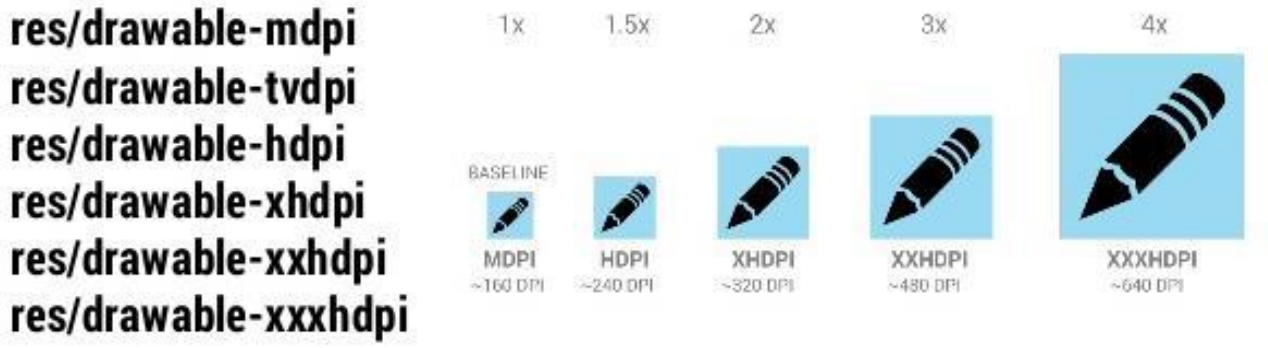
Phone Dimensions Used in Android UI

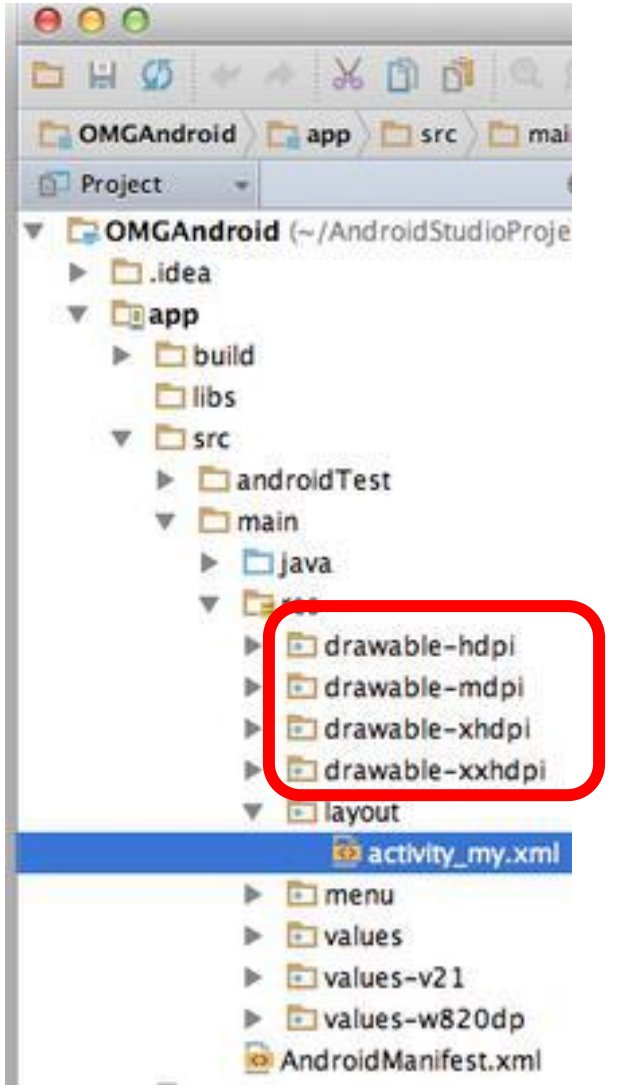
- Physical dimensions (inches) diagonally
  - E.g. Nexus 4 is 4.7 inches diagonally
- Resolution in pixels
  - E.g. Nexus 4 resolution 768 x 1280 pixels
  - Pixels diagonally: Sqrt[(768 x 768) + (1280 x 1280)]
- Pixels per inch (PPI) =
  - Sqrt[(768 x 768) + (1280 x 1280)] / 4.7= 318



Adding Pictures

- Android supports images in PNG, JPEG and GIF formats
- Put different resolutions of **same image** into different directories
  - **res/drawable-ldpi:** low dpi images (~ 120 dpi of dots per inch)
  - **res/drawable-mdpi:** medium dpi images (~ 160 dpi)
  - **res/drawable-hdpi:** high dpi images (~ 240 dpi)
  - **res/drawable-xhdpi:** extra high dpi images (~ 320 dpi)
  - **res/drawable-xxhdpi:** extra extra high dpi images (~ 480 dpi)
  - **res/drawable-xxxhdpi:** high dpi images (~ 640 dpi)





Adding Pictures

- Use generic picture name in code (no .png, .jpg, etc)
  - E.g. to reference an image **ic_launcher.png**
- At run-time, Android chooses which resolution/directory (e.g. –mdpi) based on phone resolution

```
<application
    android:allowBackup="false"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">
```

Rotating Devices

Rotating Device: Using Different Layouts

- Rotating device (e.g. portrait to landscape) kills current activity and creates new activity in landscape mode
- Rotation changes **device configuration**
- **Device configuration**: screen orientation/density/size, keyboard type, dock mode, language, etc.
- Apps can specify different resources (e.g. XML layout files, images) to use for different device configurations

- E.g. use different app layouts for portrait vs landscape screen orientation

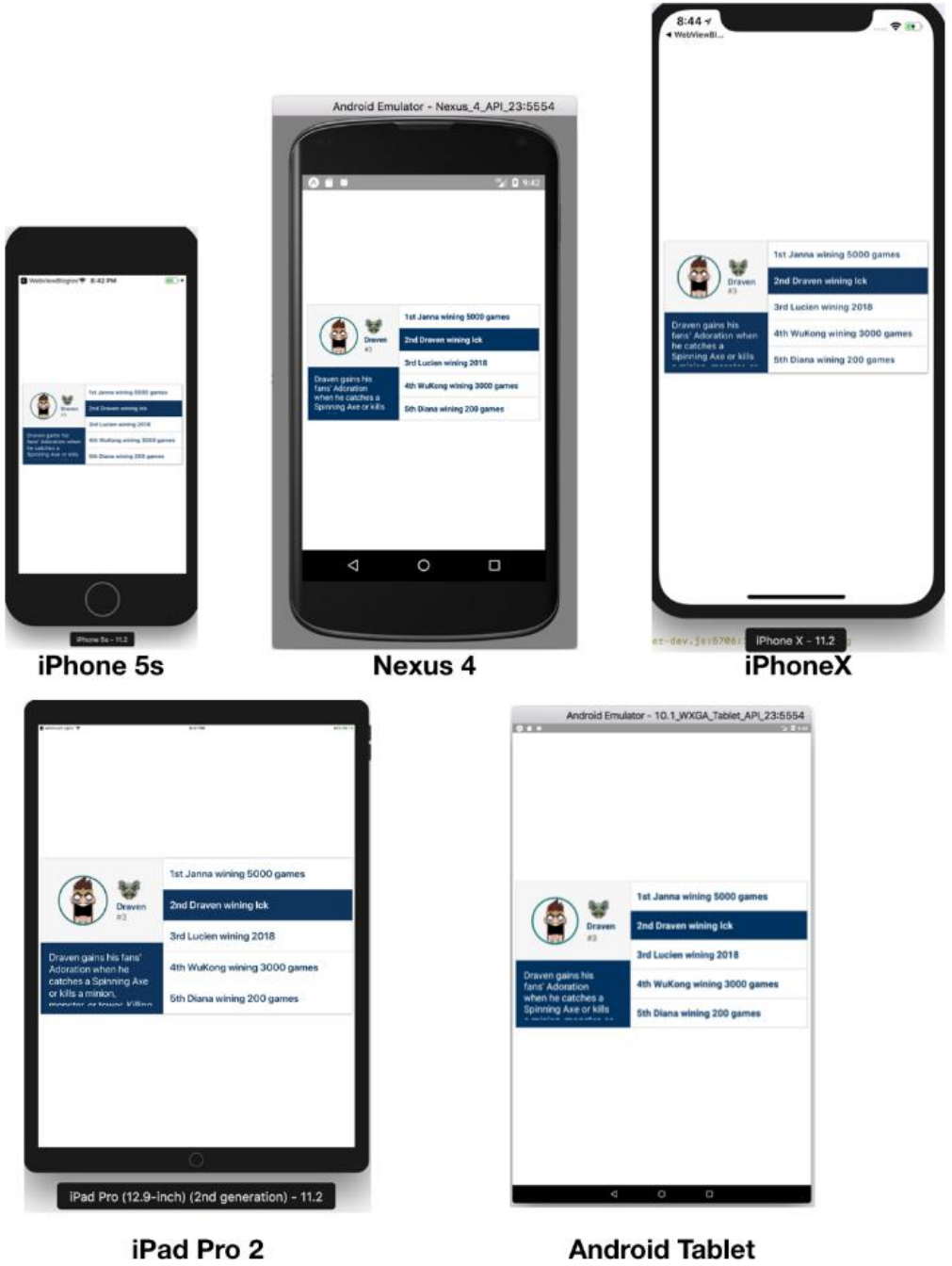
Rotating Device: Using Different Layouts

- **Portrait:** use XML layout file in **res/layout**
- **Landscape:** use XML layout file in **res/layout-land/**
- Copy XML layout file (activity_quiz.xml) from **res/layout** to **res/layout-land/** and customize it
- If configuration changes, current activity destroyed, **onCreate -> setContentView (R.layout.activity_quiz)** called again
- onCreate called whenever user switches between portrait and landscape

Mobile HCI

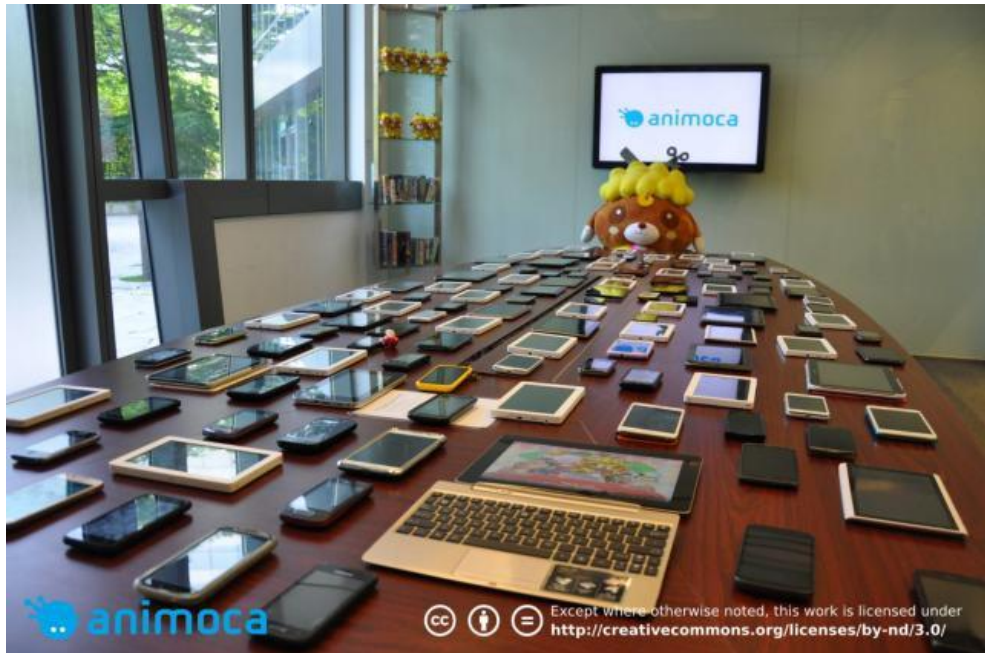
Mobile HCI

- Mobile HCI is important for an enjoyal user experience
- Can’t just reuse screens originally designed for desktops. Why?
  1. Mobile screen is small, need to manag space better
  2. Does your screen look good on wide variety of mobile screen sizes?
  3. Can users reach buttons with one han on different resolutions?
  4. Mobile device will be carried into varie adverse conditions. E.g.
     1. Do colors work well indoor vs outdoor, b vs dim light
     2. Are buttons big enough for frozen hands during winter vs summerr



Mobile HCI: Evaluation

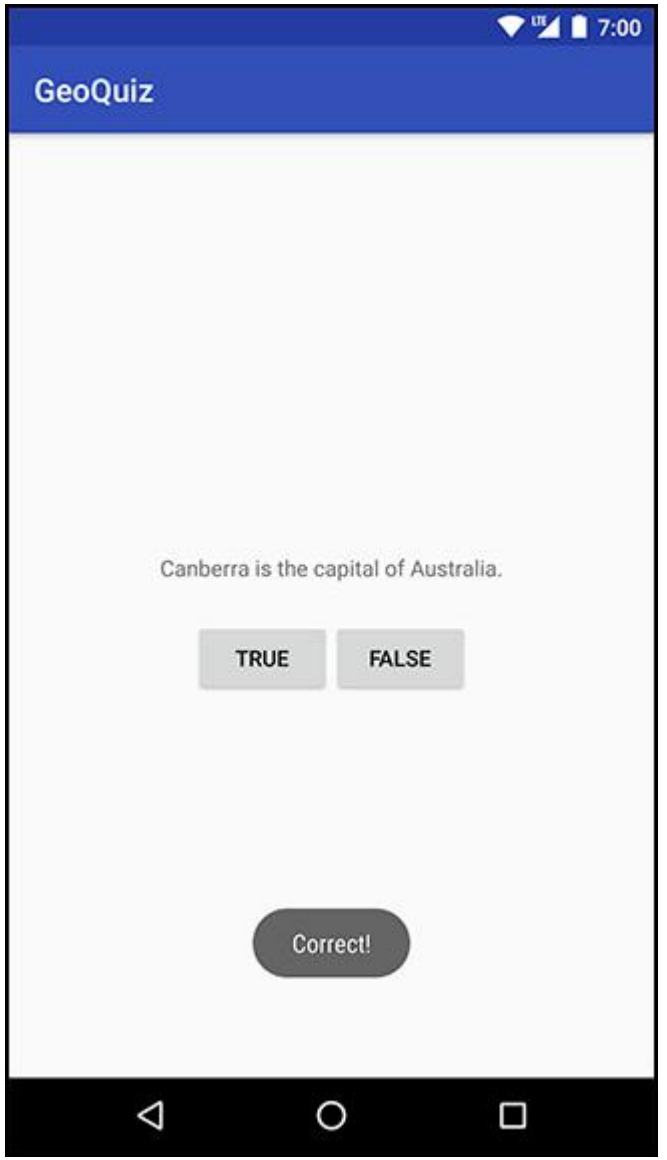
- App evaluation: iterative, user-centered
  - In lab (small) then in the field (large)
  - On wide variety of devices
    - Most poor ratings on Google Play app store are “doesn’t work on my device”
- Example: Android mobile developer tests each game on over 400 different smartphones and tablets
  - Screens
  - Aspect ratios
  - Form factors
  - Controls
  - OS versions
  - CPU/GPU
  - OpenGL/DirectX



A UI Design Example

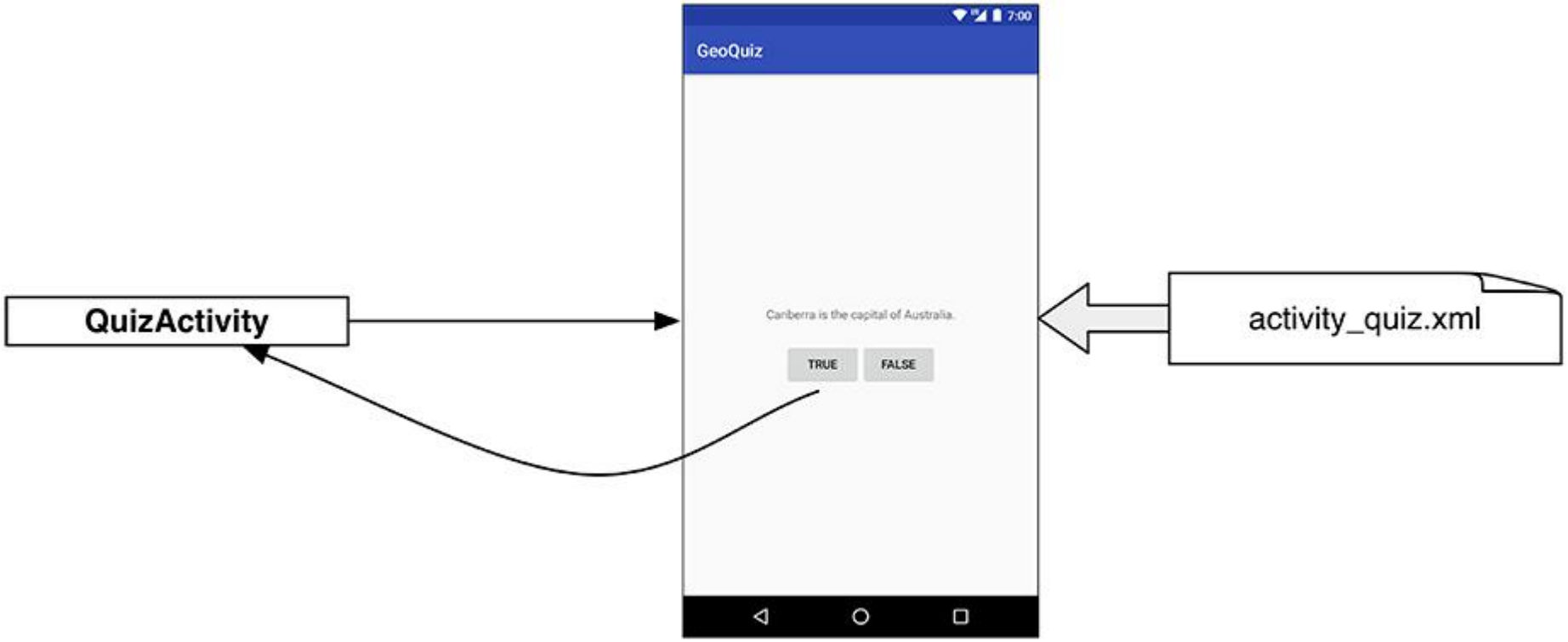
GeoQuiz App

- App presents questions to test user's knowledge of geography
- User answers by pressing True or False buttons

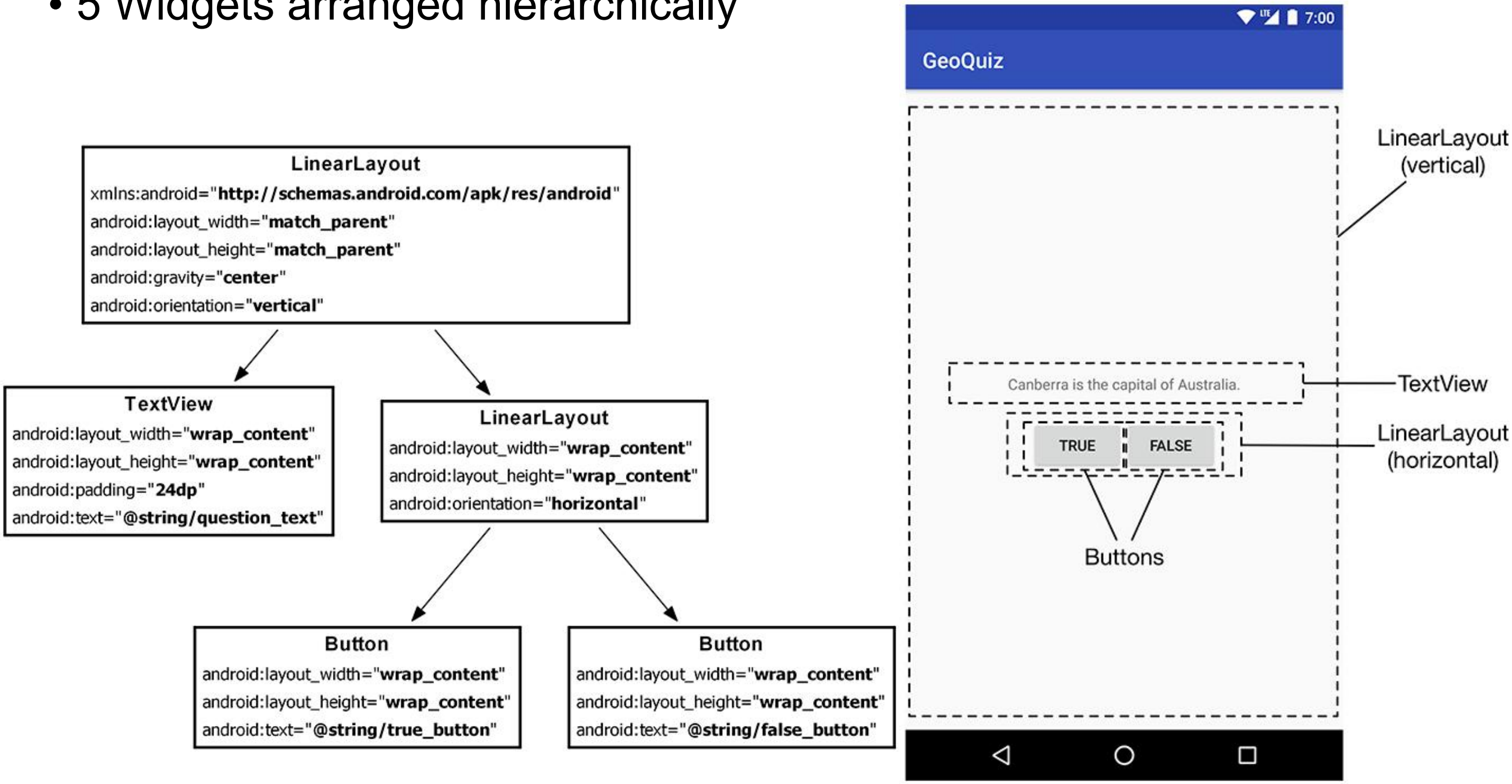


- 2 main files:
  - activity_quiz.xml: to format app screen
  - QuizActivity.java: To present question, accept True/False response

- AndroidManifest.xml lists all app components, auto-generated

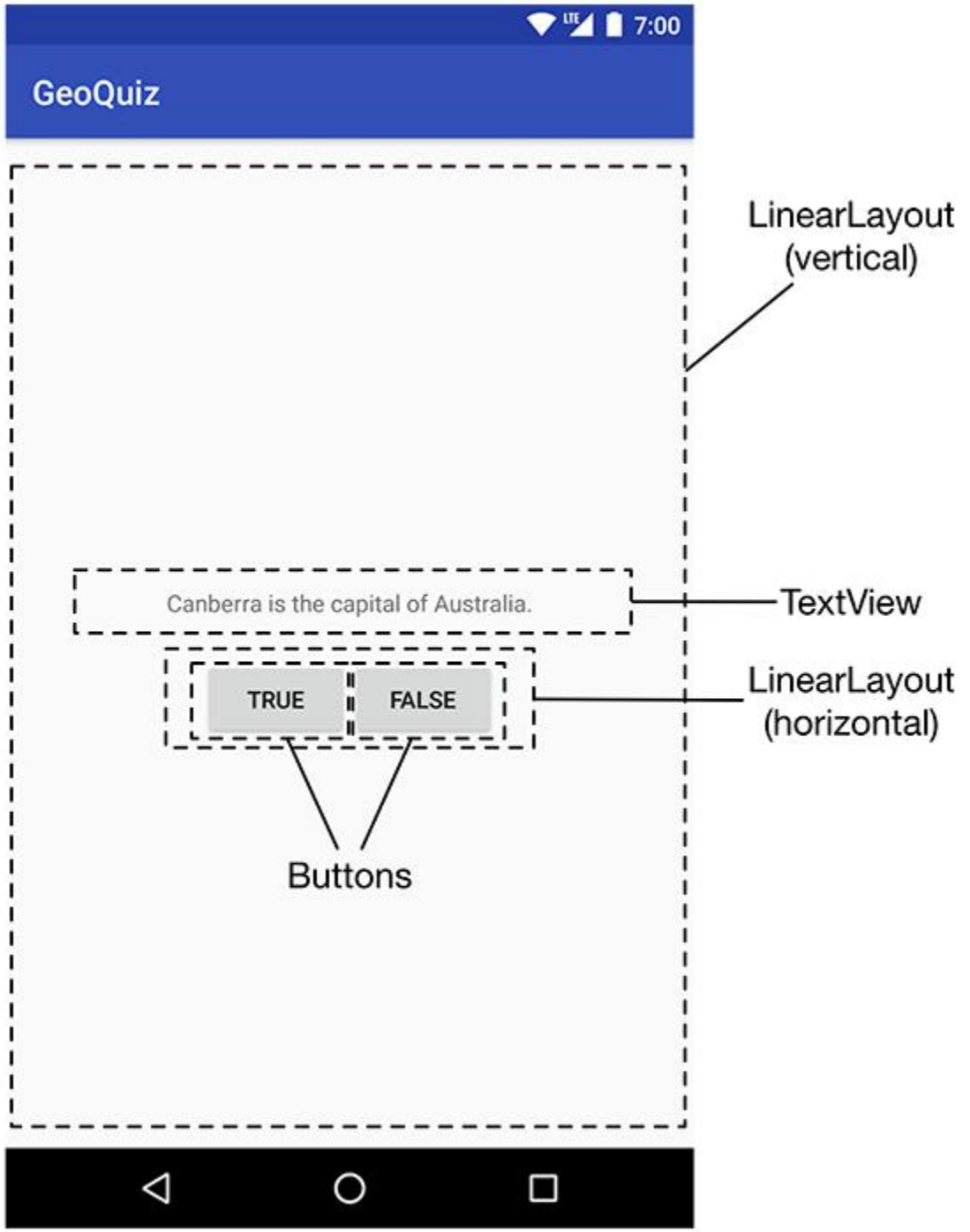


- 5 Widgets arranged hierarchically



- activity_quiz.xml File listing

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/androic
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        android:text="@string/question_text" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

      <Button
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="@string/true_button" />

      <Button
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="@string/false_button" />

    </LinearLayout>

</LinearLayout>
```



- strings.xml File listing

Define all strings app will use

- Question: “Canberra is.. “
- True
- False

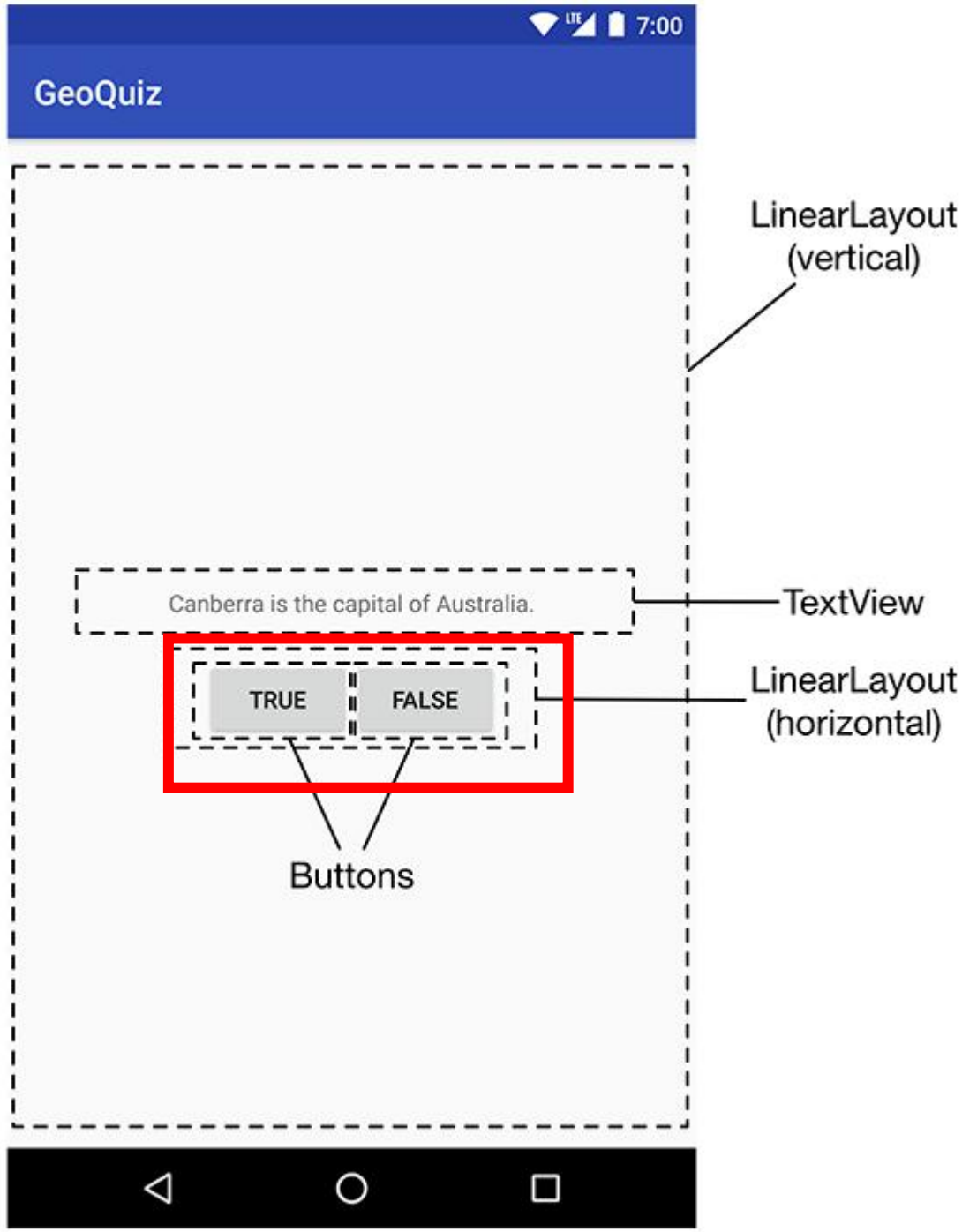
```
<resources>
    <string name="app_name">GeoQuiz</string>
    <string name="question_text">Canberra is the capital of Australia.</string>
    <string name="true_button">True</string>
    <string name="false_button">False</string>
</resources>
```

- QuizActivity.java

```
package com.bignerdranch.android.geoquiz;

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;

public class QuizActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);
    }
```

Would like java code to respond to
True/False buttons being clicked

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
... >

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="24dp"
    android:text="@string/question_text" />

  <LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <Button
      android:id="@+id/true_button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/true_button" />

    <Button
      android:id="@+id/false_button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/false_button" />

  </LinearLayout>

</LinearLayout>
```



- Responding to button clicks

```
<Button
  android:onClick="someMethod"
  ...
/>
```

```
public void someMethod(View theButton) {
  // do something useful here
}
```

Adding a Toast

- A toast is a short pop-up message
- Does not require any input or action
- After user clicks True or False button, our app will pop-up a toast to inform the user if they were right or wrong
- First, we need to add toast strings (Correct, Incorrect) to strings.xml

```
<resources>
    <string name="app_name">GeoQuiz</string>
    <string name="question_text">Canberra is the capital of Australia.</string>
    <string name="true_button">True</string>
    <string name="false_button">False</string>
    <string name="correct_toast">Correct!</string>
    <string name="incorrect_toast">Incorrect!</string>
</resources>
```

- To create a toast, call the method:

```
public static Toast makeText(Context context, int resId, int duration)
```

- After creating toast, call toast.show( ) to display it
- For example to add a toast to our onClick( ) method:

```
public void onClick(View v) {
    Toast.makeText(QuizActivity.this,
                   R.string.incorrect_toast,
                   Toast.LENGTH_SHORT).show();
}
```