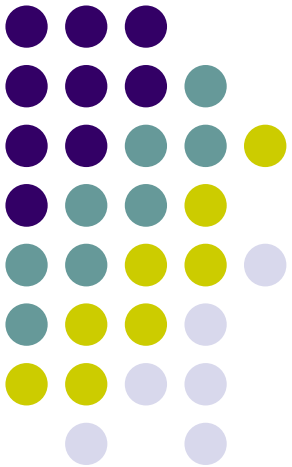
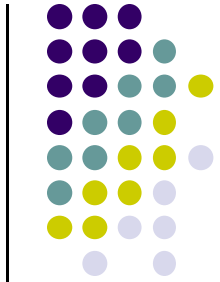
CS 403X Mobile and Ubiquitous Computing

Lecture 5: WebView, Android UI Example

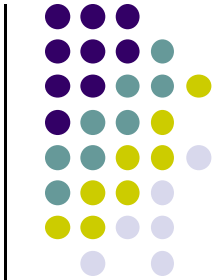
Emmanuel Agu





WebView Widget

WebView Widget



- A View that displays web pages
  - Can be used for creating your own web browser
  - OR just display some online content inside your app
- Two rendering options:
  - WebKit rendering engine (http://www.webkit.org/)
  - Chromium (http://www.chromium.org/)
- Webkit used in many web browsers including Safari

- Web pages in WebView same look same as in Safari

WebView

- Android 4.4, API level 19 added **Chromium** as alternative to WebKit

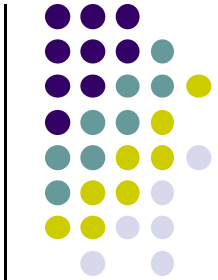
- "Chromium WebView provides broad support for HTML5, CSS3, and JavaScript.
- Supports most HTML5 features available in Chrome.
- Also has faster JavaScript Engine (V8)

https://developer.android.com/about/versions/kitkat.html

WebView Widget Functionality

- Display Web page containing HTML, CSS, Javascript
- Navigate previous URLs (forward and backwards)
- zoom in and out
- perform searches
- Additional functionality:
  - Embed images in page
  - Search page for string
  - Deal with cookies

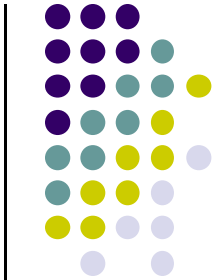
WebView Example



- Simple app to view and navigate web pages
- XML code (e.g in res/layout/main.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<WebView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/webview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
/>
```

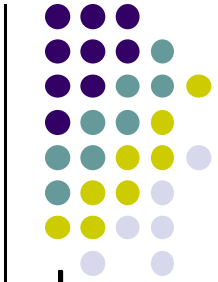
WebView Activity



- In onCreate, use loadURL to load website
- If website contains Javascript, enable Javascript
- loadUrl( ) can also load files on Android local filesystem (file://)

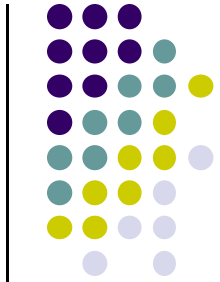
```
public class HelloWebView extends Activity {

    private WebView mWebView;

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

        mWebView = (WebView) findViewById(R.id.webview);
        mWebView.getSettings().setJavaScriptEnabled(true);
        mWebView.loadUrl("http://m.utexas.edu");
    }
}
```

WebView: Request Internet Access



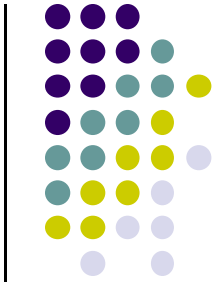
- Request **permission to use Internet** in AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="scottm.examples"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="10" />

    <uses-permission android:name="android.permission.INTERNET" />
```



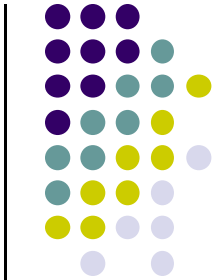
Android UI Design Example

GeoQuiz App
Reference: Android Nerd Ranch, pgs 1-32



- App presents questions to test user's knowledge of geography
- User answers by pressing **True** or **False** buttons
- How to get this book?

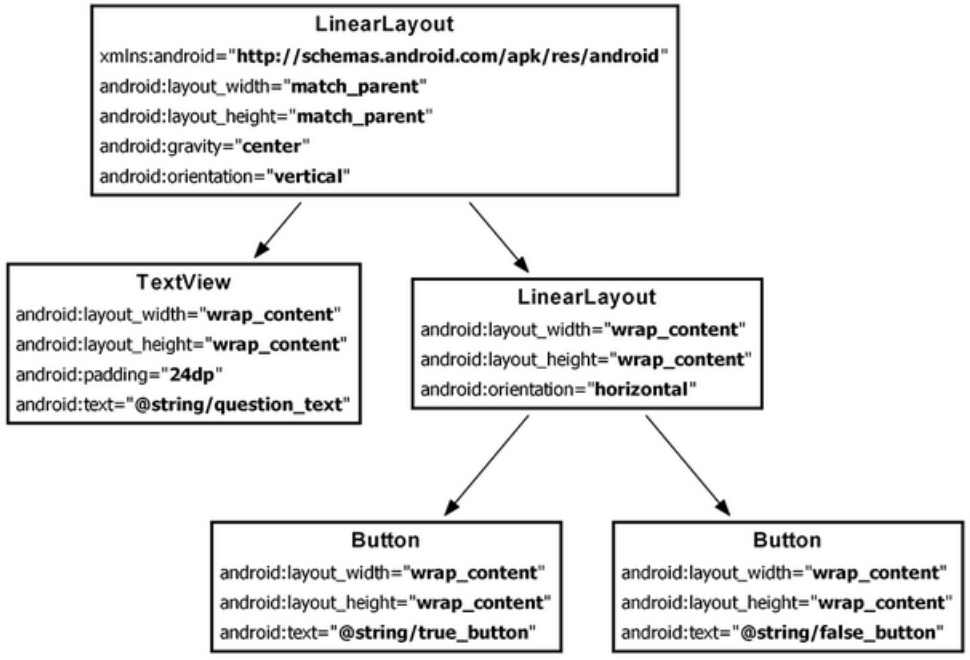
GeoQuiz App

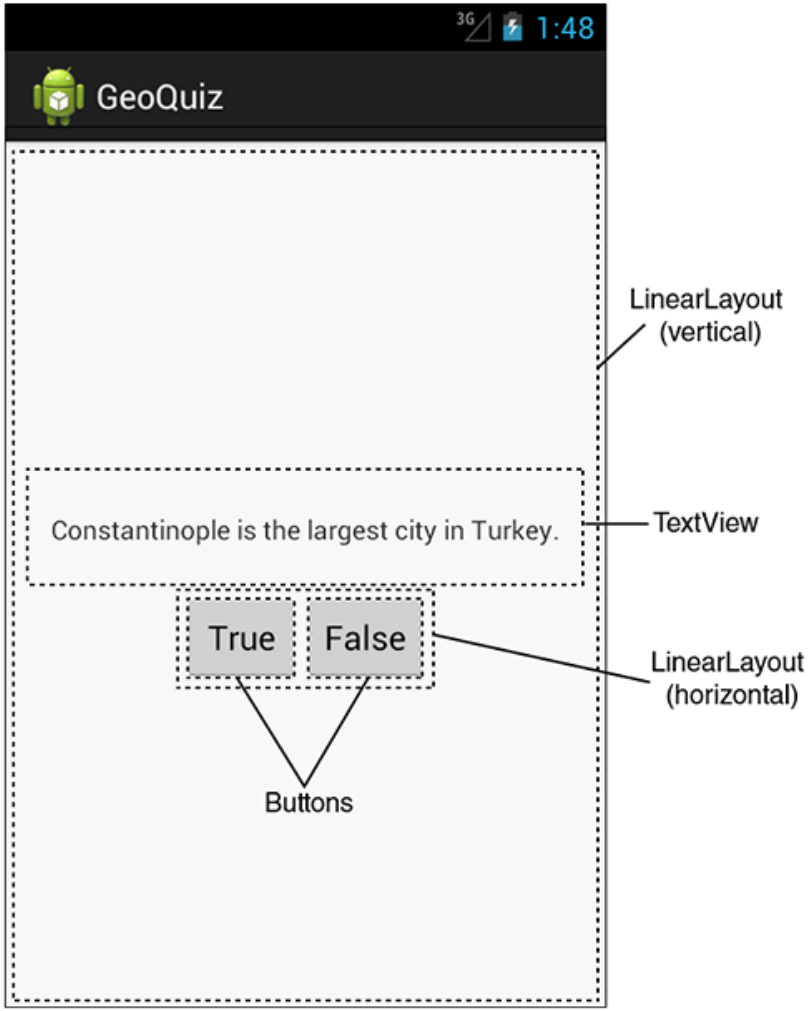


- 2 main files:
  - **activity_quiz.xml:** to format app screen
  - **QuizActivity.java:** To present question, accept True/False response
- **AndroidManifest.xml** lists all app components, auto-generated

GeoQuiz: Plan Out App Widgets

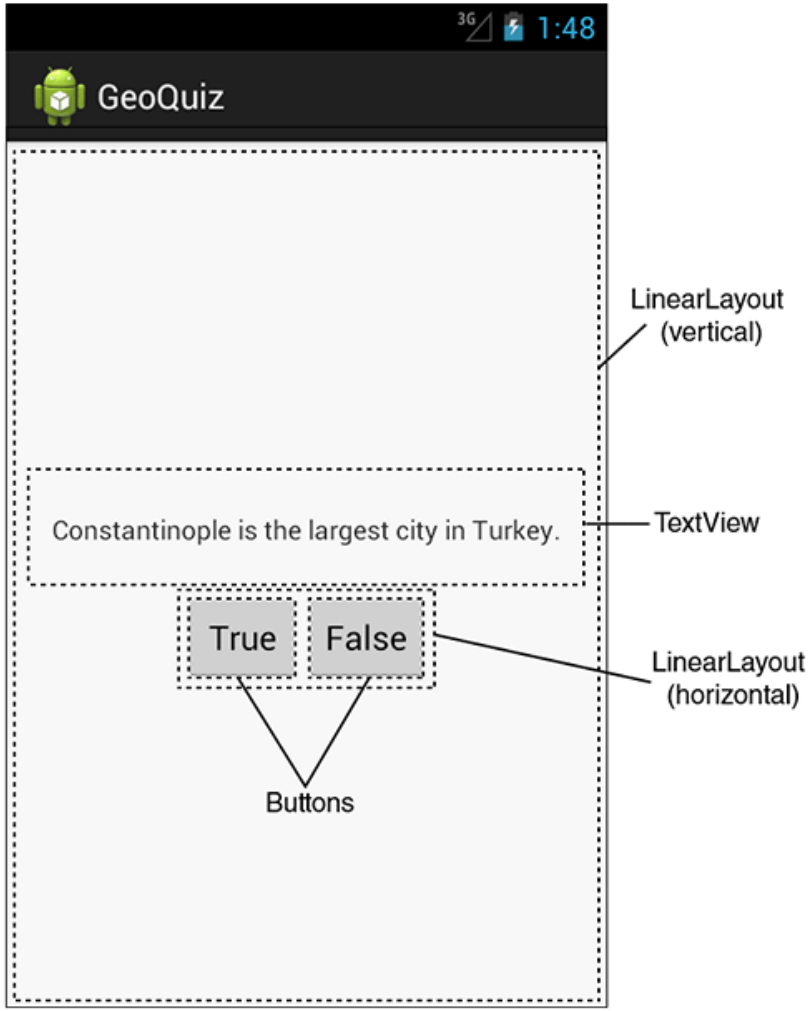
- 5 Widgets arranged hierarchically





GeoQuiz: activity_quiz.xml File listing

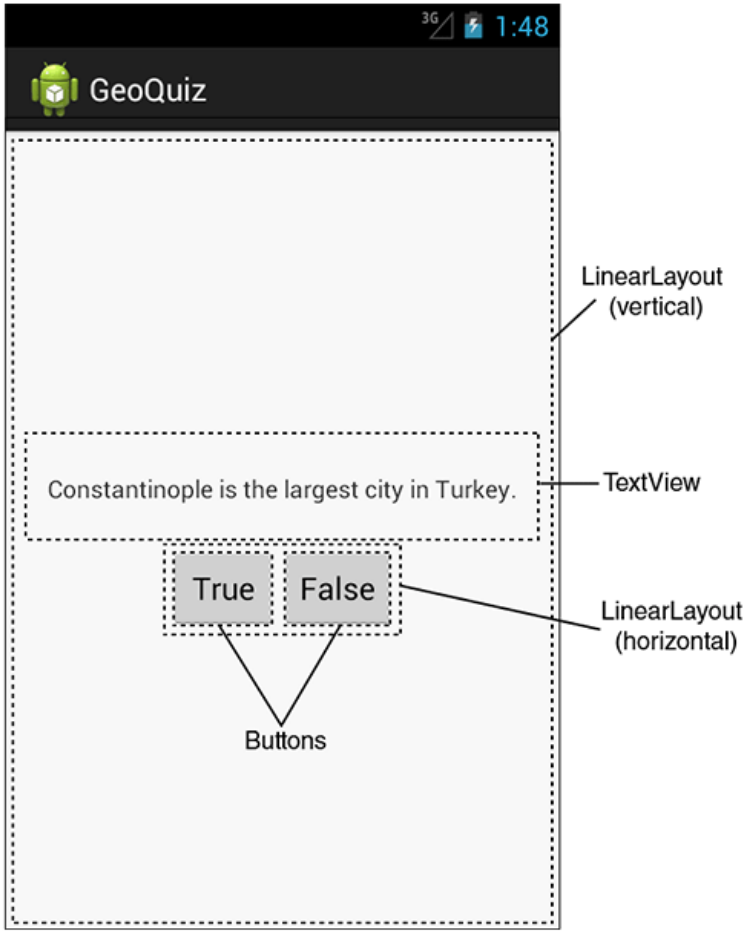
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:gravity="center"
  android:orientation="vertical" >

  <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>
```



GeoQuiz: strings.xml File listing

- **Define app strings**
  - **Question**
  - **True**
  - **False**

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

  <string name="app_name">GeoQuiz</string>
  <string name="hello_world">Hello, world!</string>
  <string name="question_text">Constantinople is the largest city in
Turkey.</string>
  <string name="true_button">True</string>
  <string name="false_button">False</string>
  <string name="menu_settings">Settings</string>

</resources>
```



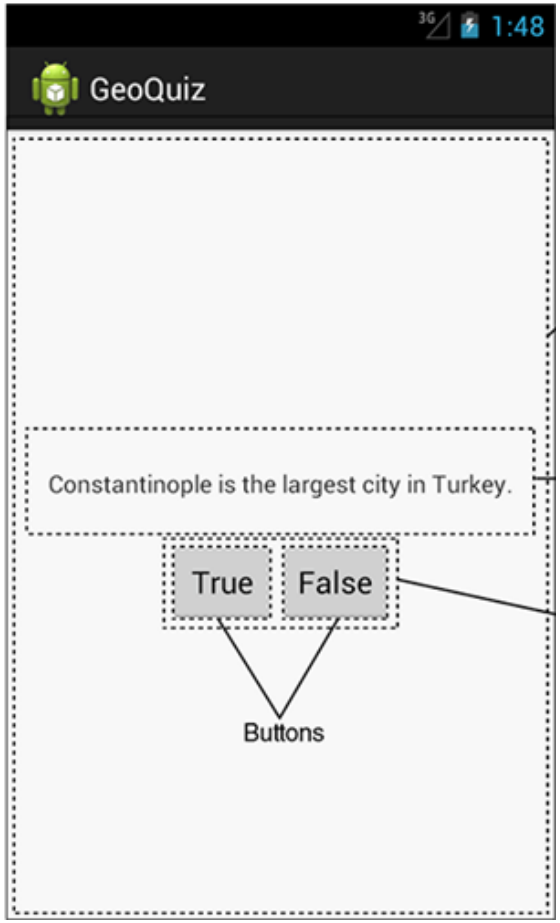
QuizActivity.java

- Initial QuizActivity.java code

```
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);
    }
```

onCreate Method is called once Activity is created

specify layout XML file (**activity_quiz.xml**)

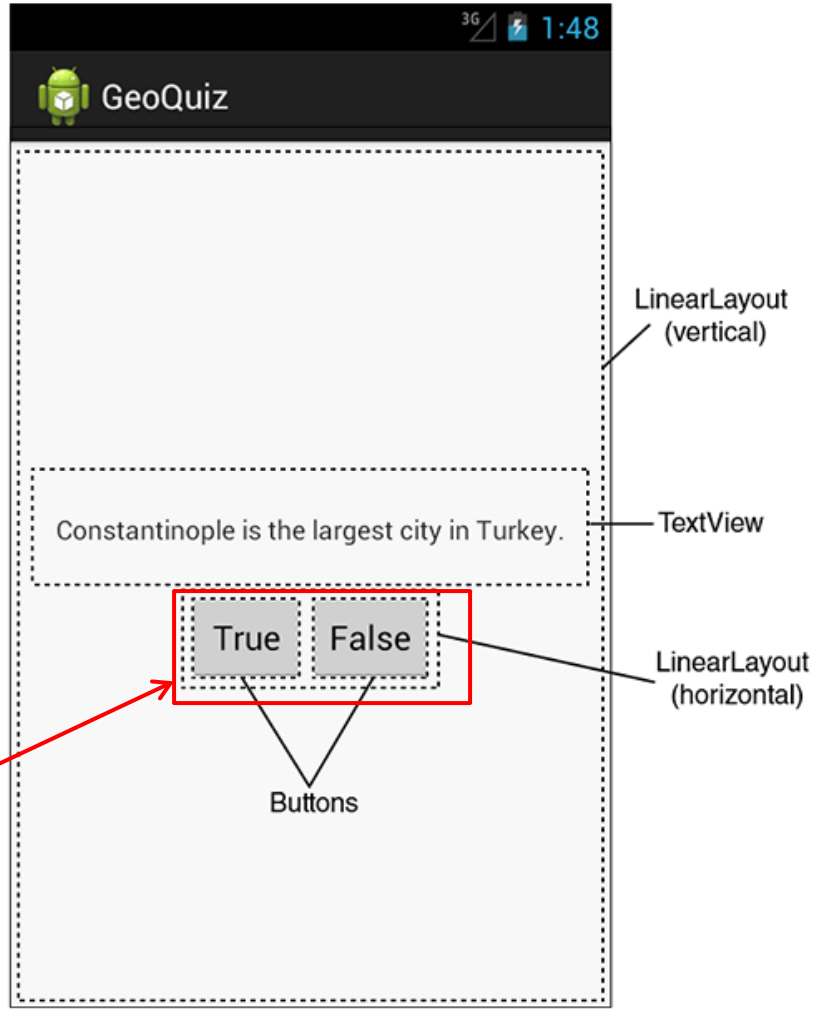


- Would like java code to respond to True/False buttons being clicked

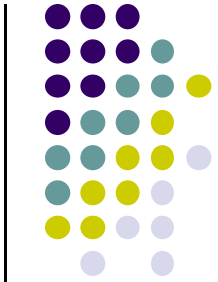
Responding to True/False Buttons in Java

```
<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>
```

Write code in Java file to specify app's response when True/False buttons are clicked



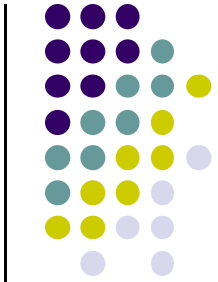
2 Alternative Ways to Respond to Button Clicks



1. In XML: set android:onClick attribute (already seen this)

2. In java create a ClickListener object, override onClick method
   - typically done with anonymous inner class

Recall: Approach 1: Responding to Button Clicks



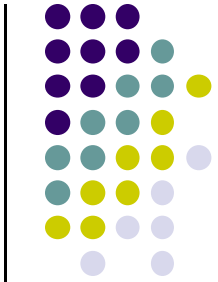
1. In XML file (e.g. Activity_my.xml), set android:onClick attribute to specify method to be invoked

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

2. In Java file (e.g. MainActivity.java) declare method/handler to take desired action

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

Approach 2: Create a ClickListener object, override onClick

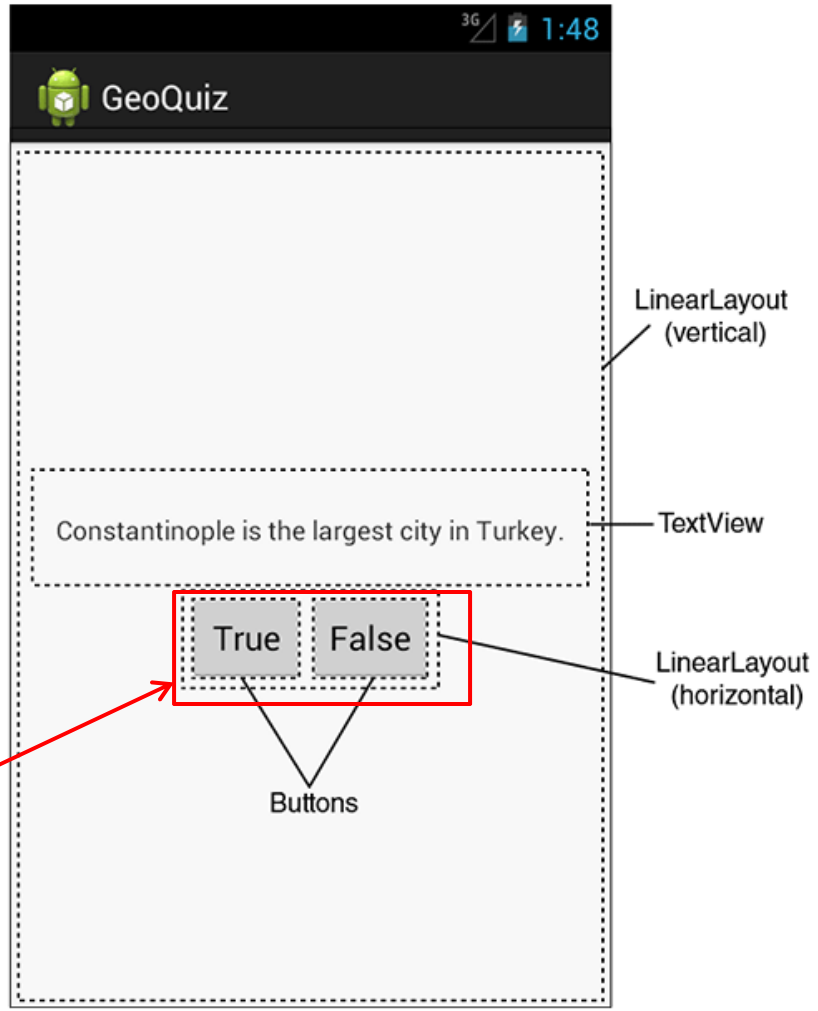


- First, get reference to Button in our Java file. How?

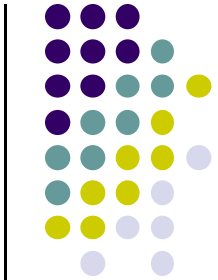
```
<Button
  android:id="@+id/true_button"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/true_button" />
```

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

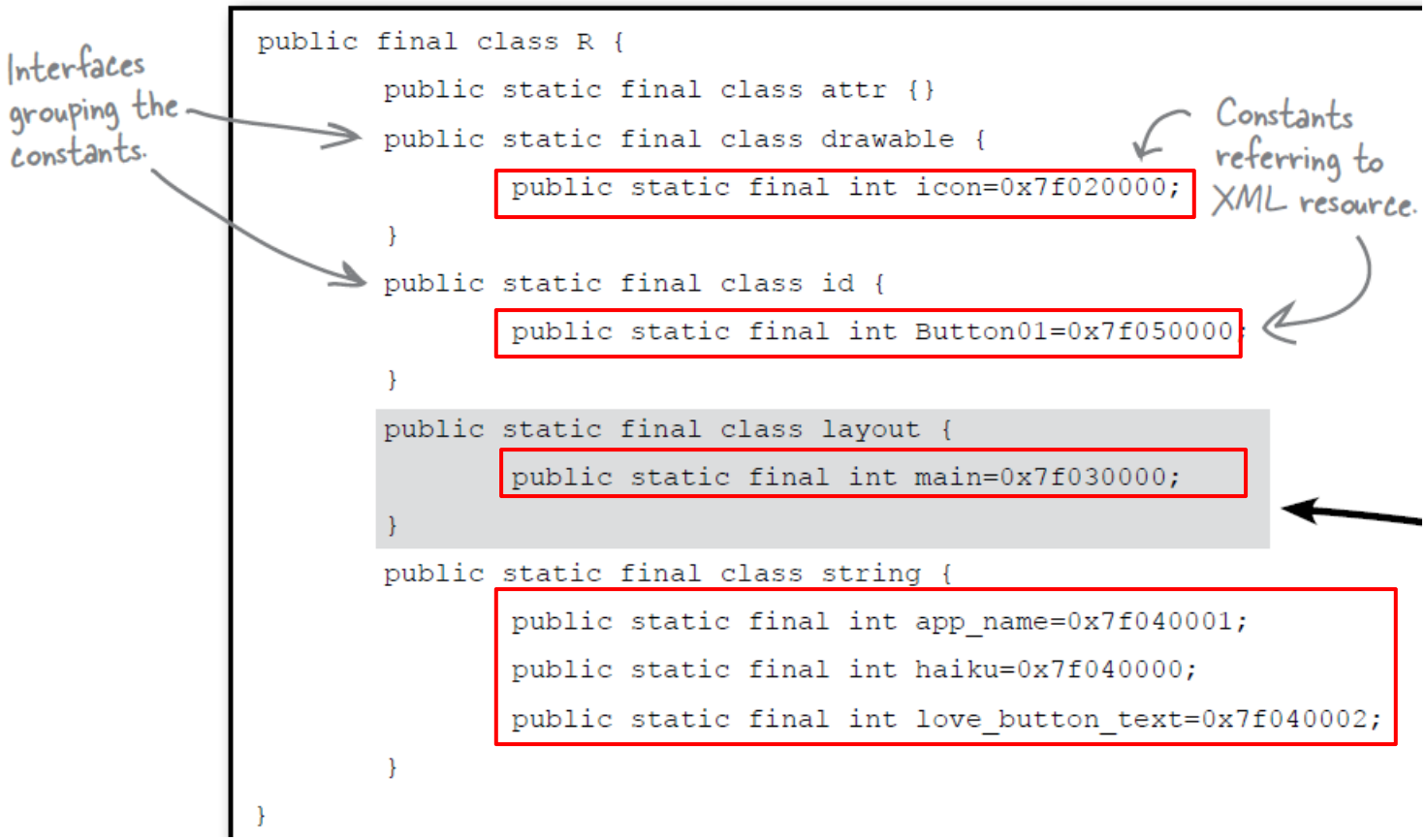
Need reference to Buttons



R.Java Constants



- During compilation, XML resources (drawables, layouts, strings, views with IDs, etc) are assigned constants
- Sample R.Java file
- In Java file, can refer to these resources using their constants



Referencing Widgets by ID

- To reference a widget in Java code, use **findviewbyID** need its **android:id**
- Use **findviewbyID**

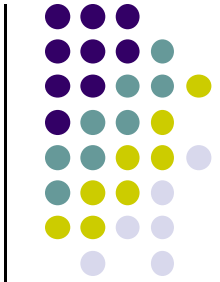
In XML file, give the widget/view an ID i.e. assign android:id

In java file, to reference/manipulate view/widget use its ID to find it (call findviewbyID( ))

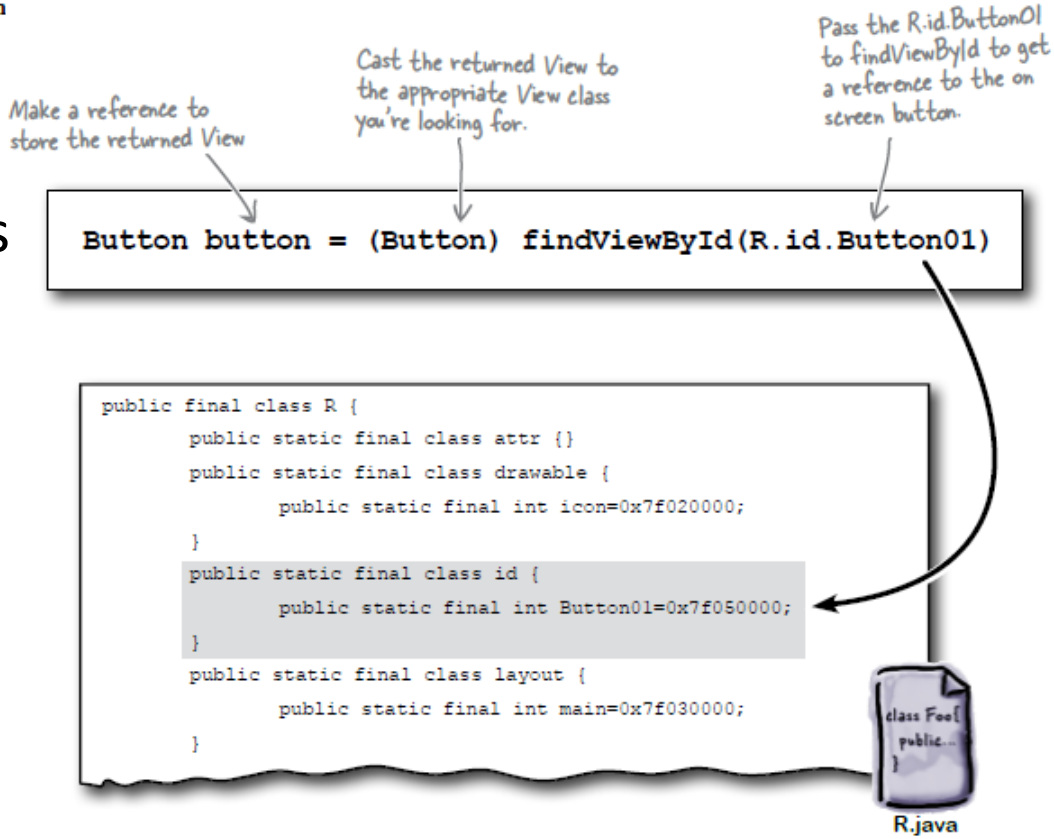
```
<Button android:text="@+id/Button01"
        android:id="@+id/Button01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
findViewById(R.id.Button01)
```

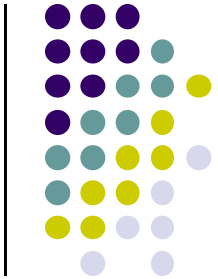
Getting View References



- **findViewById** is implemented in base Activity class so it can be called in our java file (e.g. MainActivity.java)
- Argument of **findViewById** is constant of resource
- A generic view is returned (not subclasses e.g. buttons, TextView), so needs to cast



QuizActivity.java: Getting References to Buttons



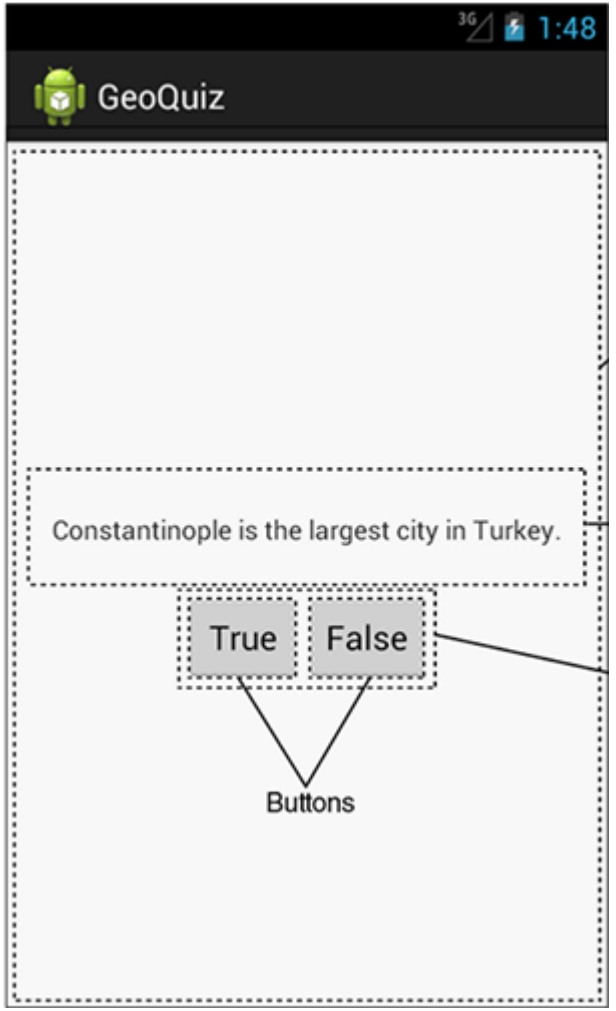
- To get reference to buttons in java code

```
public class QuizActivity extends Activity {

    private Button mTrueButton;
    private Button mFalseButton;

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

        mTrueButton = (Button)findViewById(R.id.true_button);
        mFalseButton = (Button)findViewById(R.id.false_button);
    }

...
}
```

Declaration in XML

```
<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" />
```



QuizActivity.java: Setting Listeners

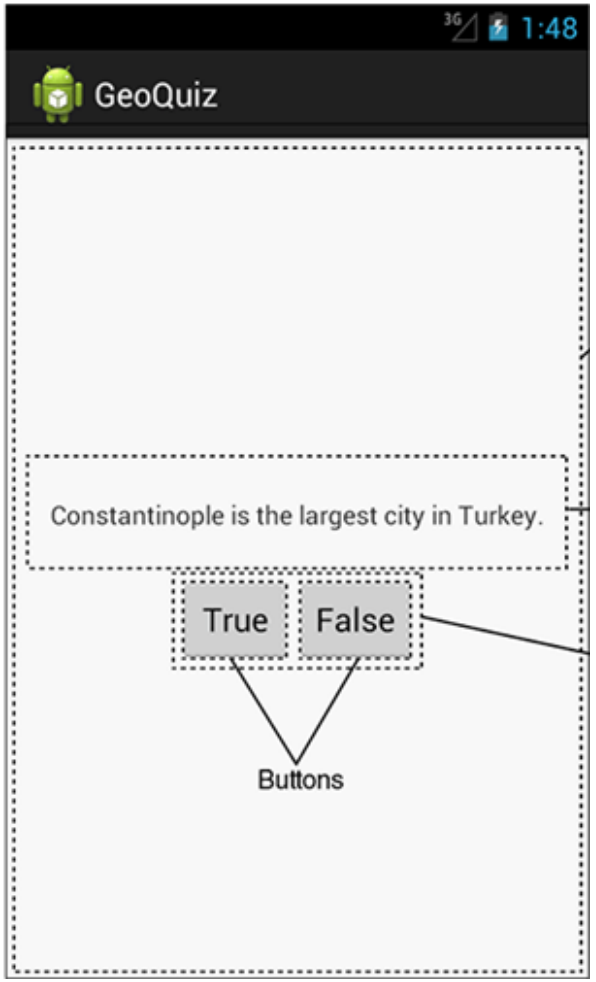
- Set listeners for **True** and **False** button

```
...
    mTrueButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // Does nothing yet, but soon!
        }
    });

    mFalseButton = (Button)findViewById(R.id.false_button);
    mFalseButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // Does nothing yet, but soon!
        }
    });
}
```

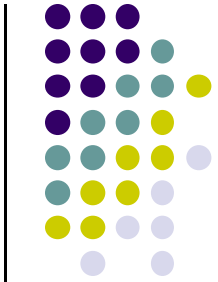
1.Set Listener Object For mTrueButton

3. Overide onClick method (insert your code to do whatever you want as mouse response here)

2. Create listener object as anonymous (unnamed) inner object



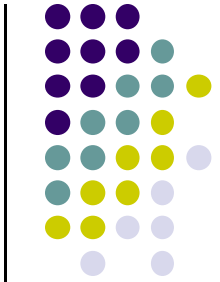
QuizActivity.java: Adding a Toast



- 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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">GeoQuiz</string>
  <string name="question_text">Constantinople is the largest city in
Turkey.</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>
  <string name="menu_settings">Settings</string>
</resources>
```

QuizActivity.java: Adding a Toast



- To create a toast, call the method:

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

Instance of Activity (Activity is a subclass of context)

Resouce ID of the string that toast should display

Constant to specifiy how long toast should be visible

- 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();
}
```

QuizActivity.java: Adding a Toast

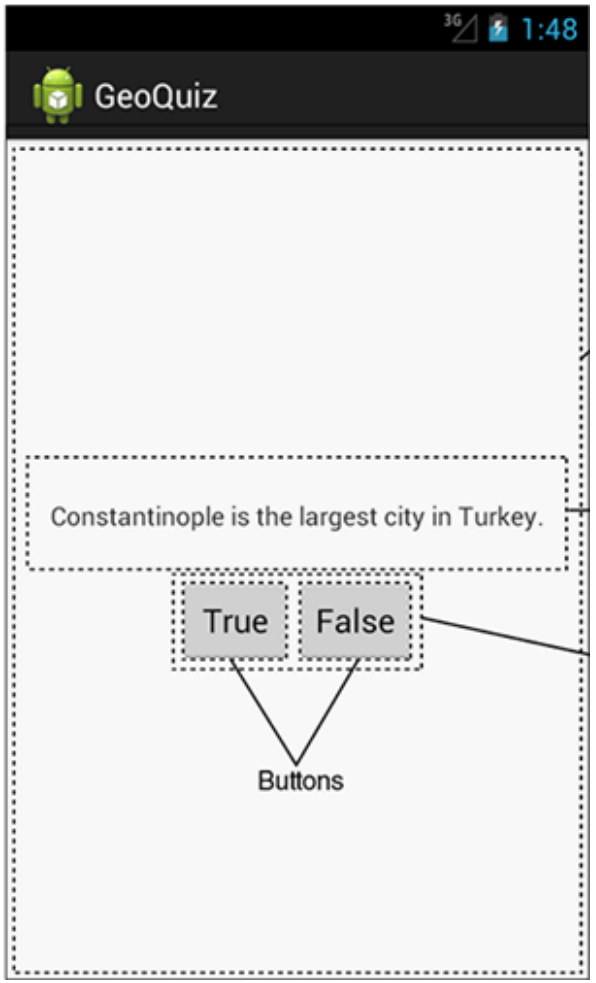
- Code for adding a toast

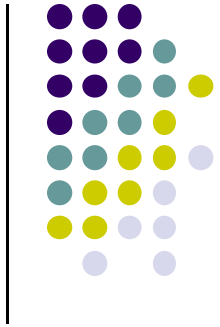
```
...
mTrueButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(QuizActivity.this,
                       R.string.incorrect_toast,
                       Toast.LENGTH_SHORT).show();
    }
});

mFalseButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(QuizActivity.this,
                       R.string.correct_toast,
                       Toast.LENGTH_SHORT).show();
    }
});
```

1.Set Listener Object For mTrueButton

3. Overide onClick method Make a toast

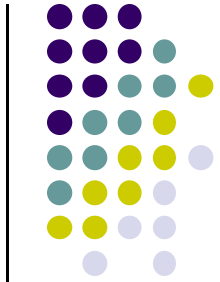
2. Create listener object as anonymous innner object





QuizActivity.java: Complete Listing

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

import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class QuizActivity extends Activity {

    Button mTrueButton;
    Button mFalseButton;

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

        mTrueButton = (Button)findViewById(R.id.true_button);
        mTrueButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(QuizActivity.this,
                        R.string.incorrect_toast, Toast.LENGTH_SHORT)
                        .show();
            }
        });
```

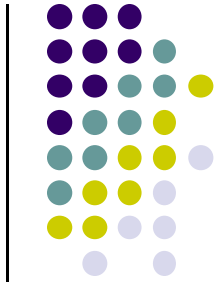


```
    mFalseButton = (Button)findViewById(R.id.false_button);
    mFalseButton.setOnClickListener(new View.OnClickListener() {

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

QuizActivity.java: Complete Listing (Contd)

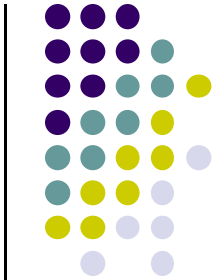
```
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {

     // Inflate the menu;
     // this adds items to the action bar if it is present.

        getMenuInflater().inflate(R.menu.activity_quiz, menu);
        return true;
    }

}
```

Used if app has an Action bar menu



Android App Components

Android App Components



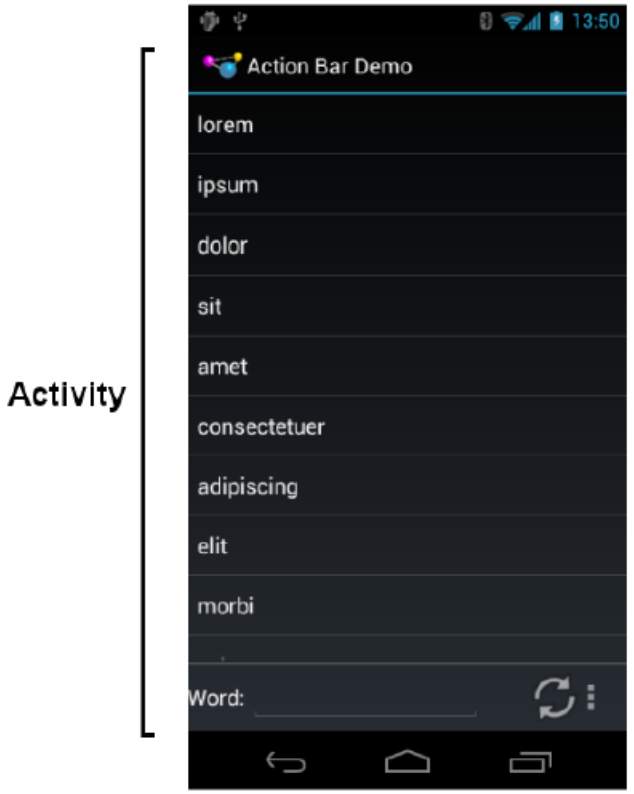
- Typical Java program starts from main( )

```
class SillyApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

- Android app: No need to write a main
- Just define app components by creating sub-classes of base classes already defined in Android
- 4 main types of Android app components:
  - Activities (already seen this)
  - Services
  - Content providers
  - Broadcast receivers

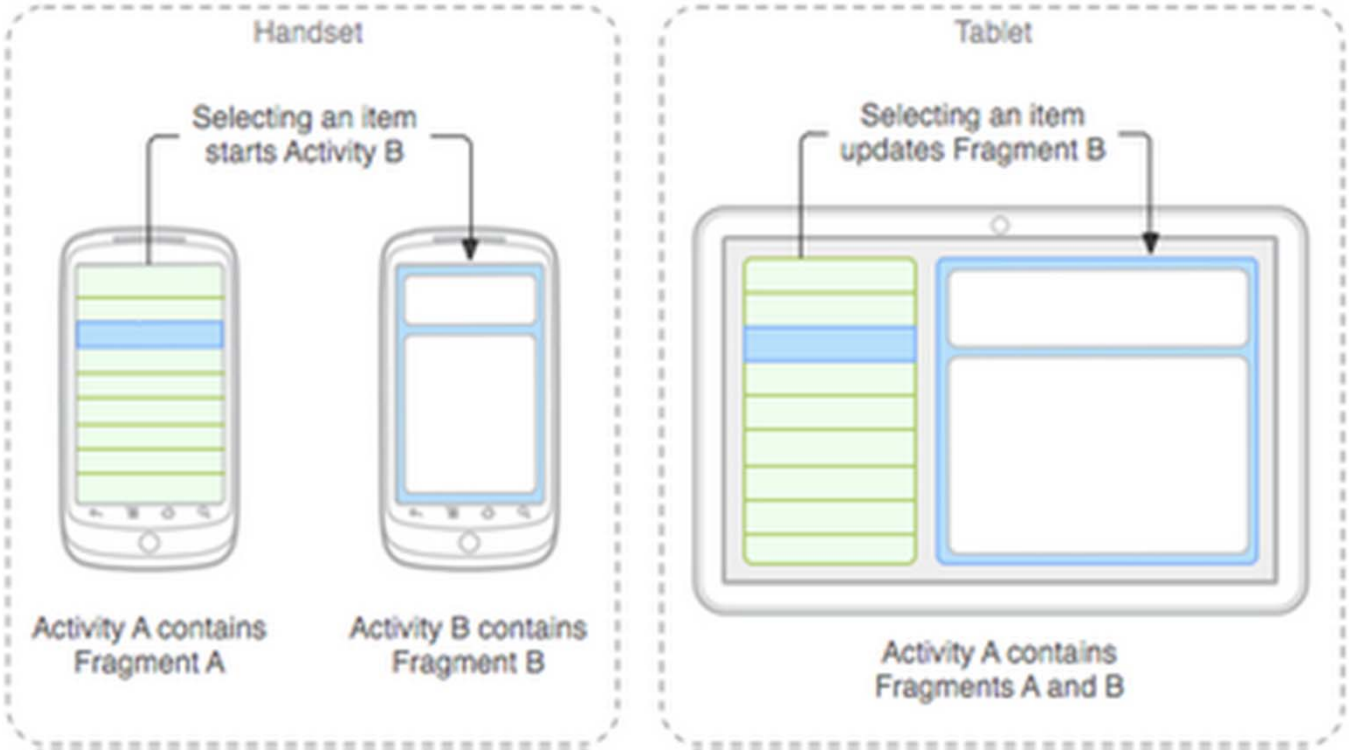
Recall: Activities

- Activity: main building block of Android UI
- Analogous to a window or dialog box in a desktop application
- Apps
  - have at least 1 activity that deals with UI
  - Entry point of app similar to **main( )** in C
  - typically have multiple activities
- Example: A camera app
  - **Activity 1:** to focus, take photo, start activity 2
  - **Activity 2:** to present photo for viewing, save it

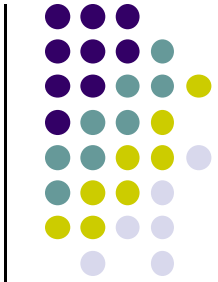


Fragments

- Fragments enables 1 app to look different on phone vs tablet
- An activity can contain multiple fragments that are organized differently for phone vs tablet
- Fragments are UI components that can be attached to different Activities.
- More later

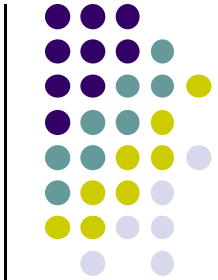


Services



- Activities are short-lived, can be shut down anytime (e.g when user presses back button)
- Services keep running in background
- Minimal need to interact with (independent of) any activity
- Typically an activity will control a service -- start it, pause it, get data from it
- Similar to Linux/Unix CRON job
- Example uses of services:
  - Periodically check device's GPS location
  - Check for updates to RSS feed
- App Services are sub-class of **Services** class

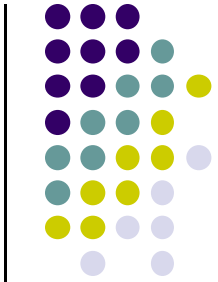
Android Platform Services



- Android Services can either be on:
  - Android Platform (local)
  - Google (remote)

- Android platform services:
  - **LocationManager:** location-based services.
  - **ViewManager** and **WindowManager:** Manage display and User Interface
  - **AccessibilityManager:** accessibility, support for physically impaired users
  - **ClipboardManager:** access to device's clipboard, for cutting and pasting content.
  - **DownloadManager:** manages HTTP downloads in the background
  - **FragmentManager:** manages the fragments of an activity.
  - **AudioManager:** provides access to audio and ringer controls.

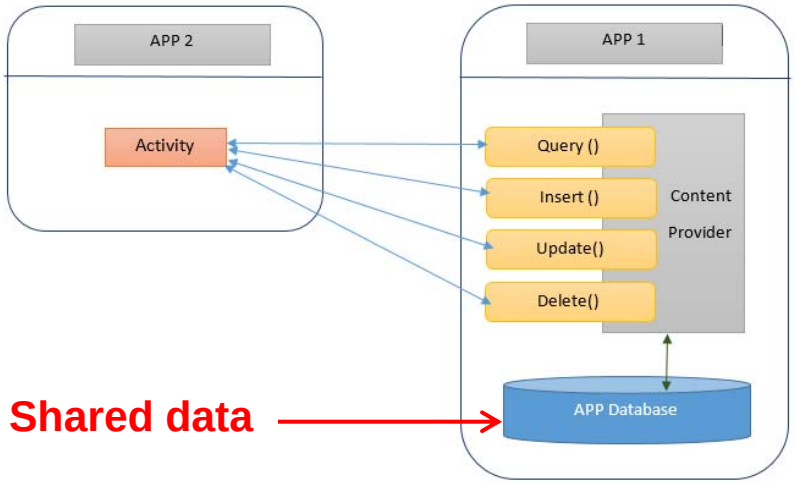
Google Services (In Google Cloud)

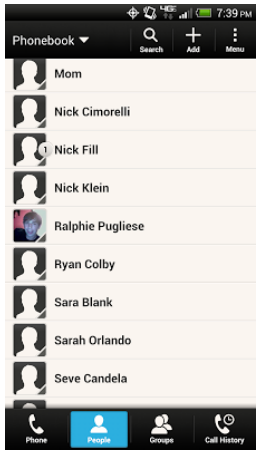


- Maps
- Location-based services
- Game Services
- Authorization APIs
- Google Plus
- Play Services
- In-app Billing
- Google Cloud Messaging
- Google Analytics
- Google AdMob ads

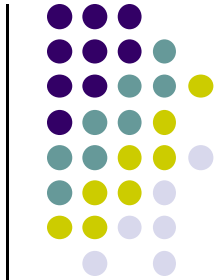
Content Providers

- Android apps can share data (e.g. contacts)
- Content Provider:
  - Abstracts shareable data, makes it accessible through methods
  - Applications can access that shared data by calling methods for the relevant **content provider**
- Example: We can write an app that:
  - Retrieve's contacts list from contacts content provider
  - Adds contacts to social networking (e.g. Facebook)

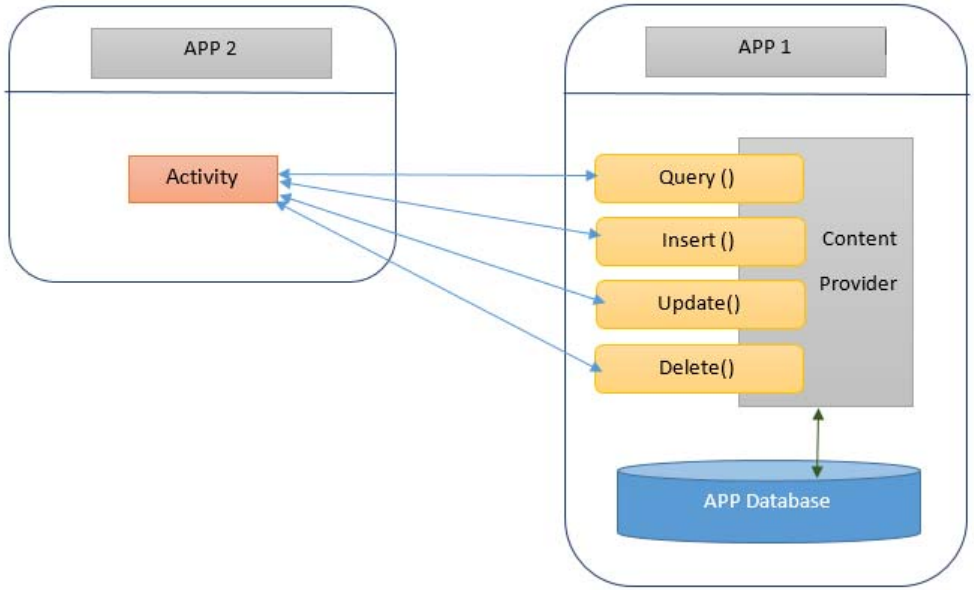




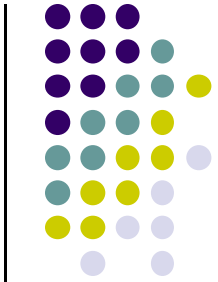
Content Providers



- Apps can also **ADD** to data through content provider. E.g. Add contact
- E.g. Our app can also share its data
- App Content Providers are sub-class of **ContentProvider** class

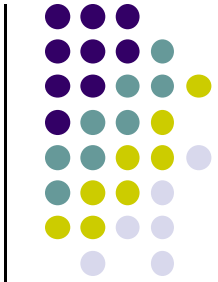


Broadcast Receivers



- The system, or applications, periodically *broadcasts* events
- Example broadcasts:
  - Battery getting low
  - Download completed
  - New email arrived
- A broadcast receiver can listen for broadcasts, respond
- Our app can also initiate broadcasts
- Broadcast receivers
  - Typically don't interact with the UI
  - Commonly create a status bar notification to alert the user when broadcast event occurs
- App Broadcast Receivers are sub-class of **BroadcastReceiver** class

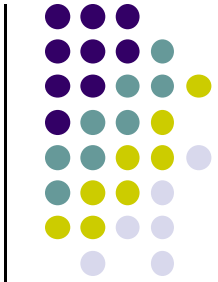
Quiz



- Pedometer App
  - Component A: continously counts user's steps even when user closes app, does other things on phone (e.g. youtube, calls)
  - Component B: Displays user's step count
  - Component C: texts user's friends every day with their step totals

- What should component A be declared as (Activity, service, content provider, broadcast receiver)
- What of component B?
- Component C?

References



- Android Nerd Ranch (2nd Edition), 2015
- Busy Coder's guide to Android version 4.4
- CS 65/165 slides, Dartmouth College, Spring 2014
- CS 371M slides, U of Texas Austin, Spring 2014