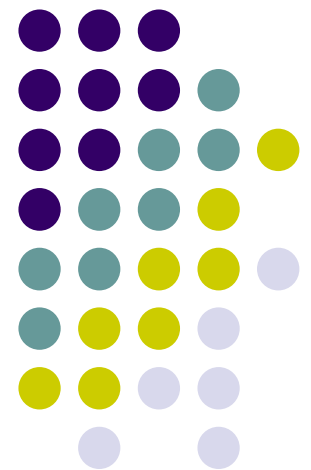
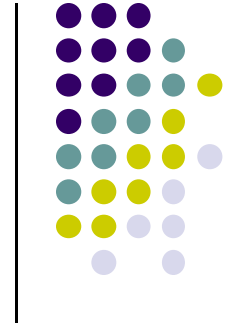


CS 528 Mobile and Ubiquitous Computing

Lecture 3: Android UI, WebView, Android Activity Lifecycle

Emmanuel Agu

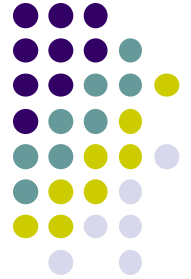




Android UI Design Example

GeoQuiz App

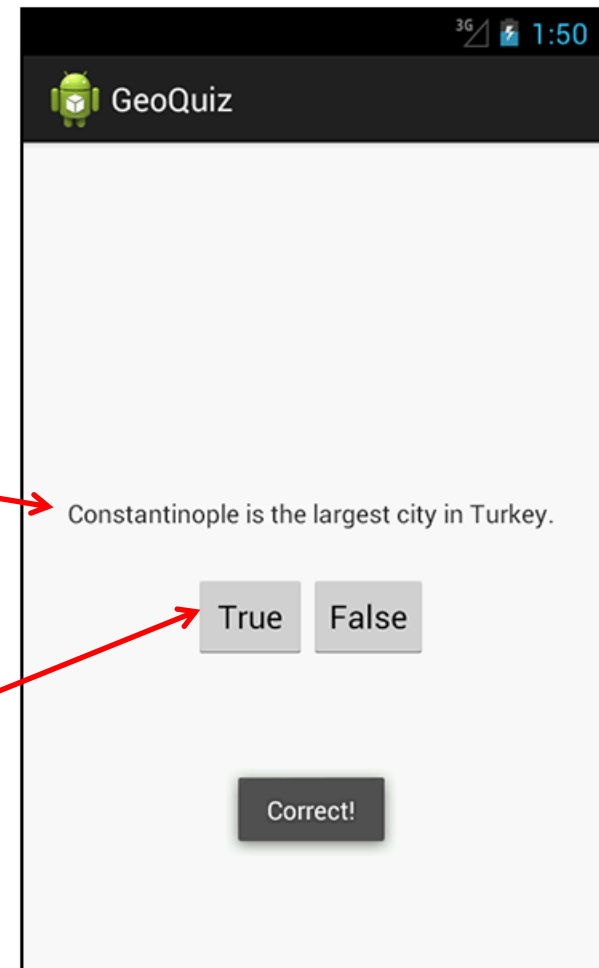
Reference: Android Nerd Ranch, pgs 1-30

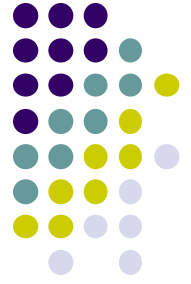


- App presents questions to test user's knowledge of geography
- User answers by pressing **True** or **False** buttons
- How to get this book?

Question

User responds
by clicking True
or False

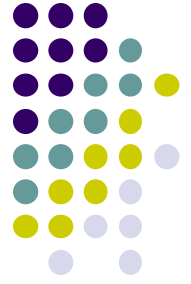




GeoQuiz App

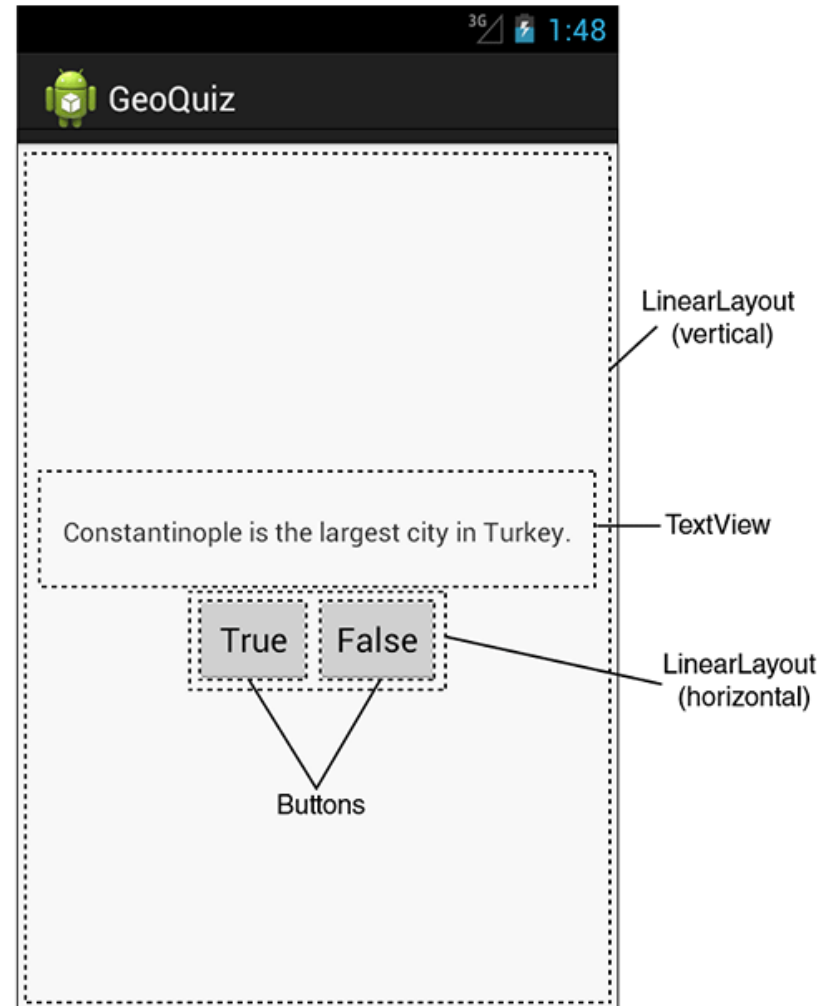
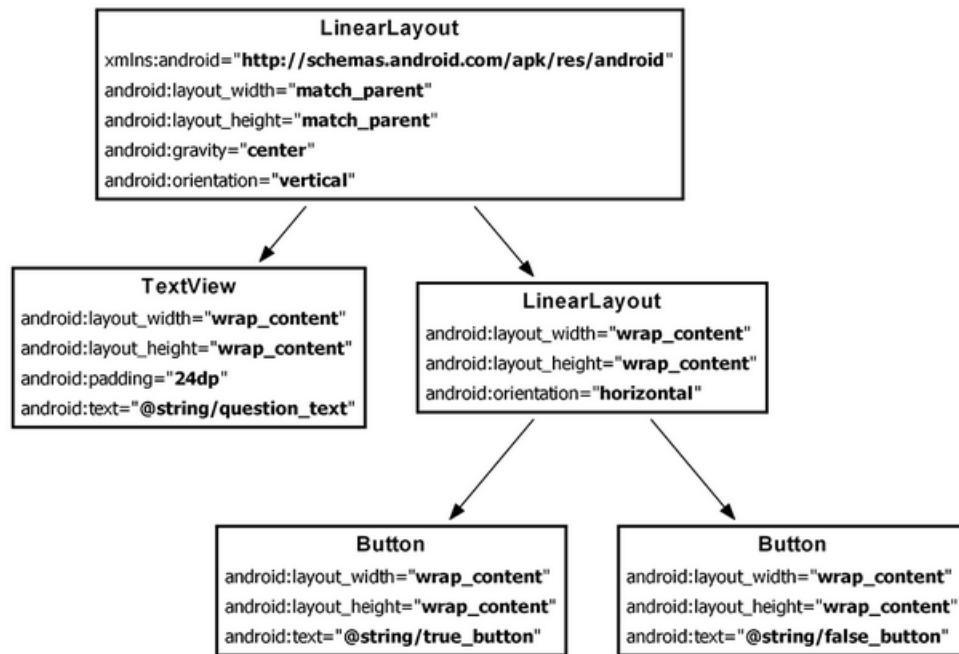
- 2 main files:
 - **activity_quiz.xml**: to format app screen
 - **QuizActivity.java**: To present question, accept True/False response
- **AndroidManifest.xml** also auto-generated



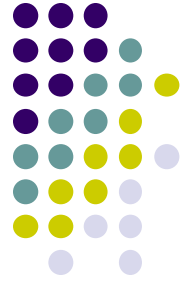


GeoQuiz: Plan Out App Widgets

- 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" />

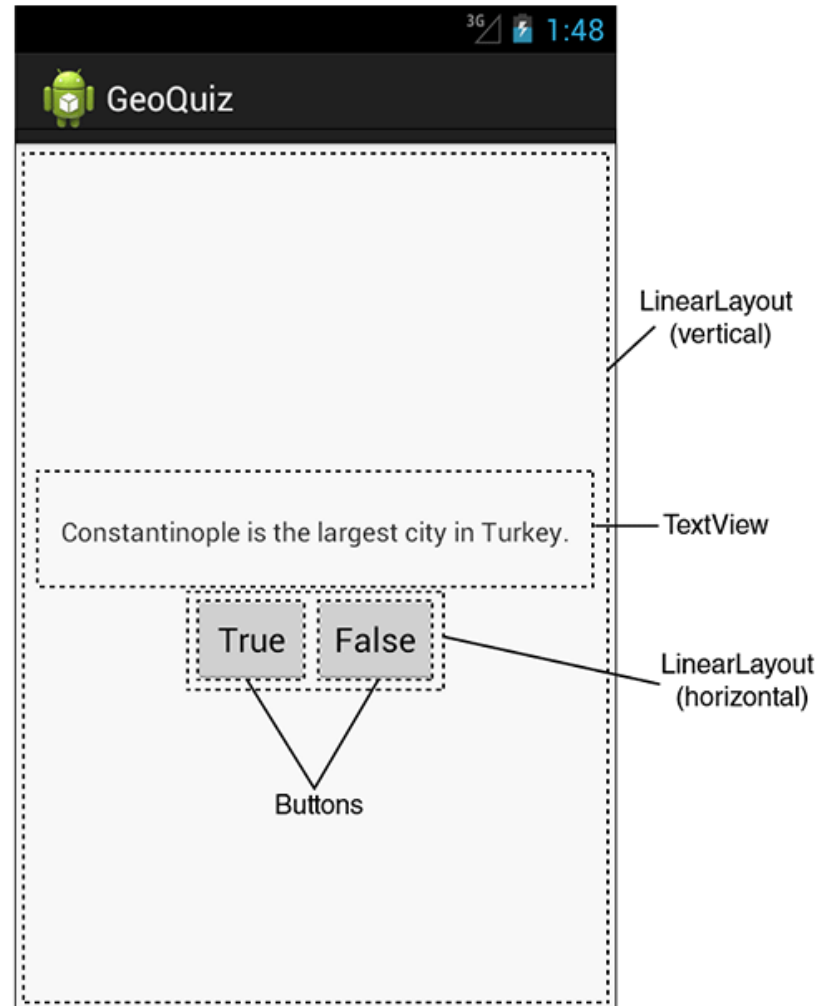
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

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

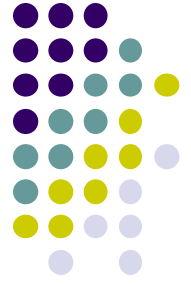
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/false_button" />

    </LinearLayout>

</LinearLayout>
```



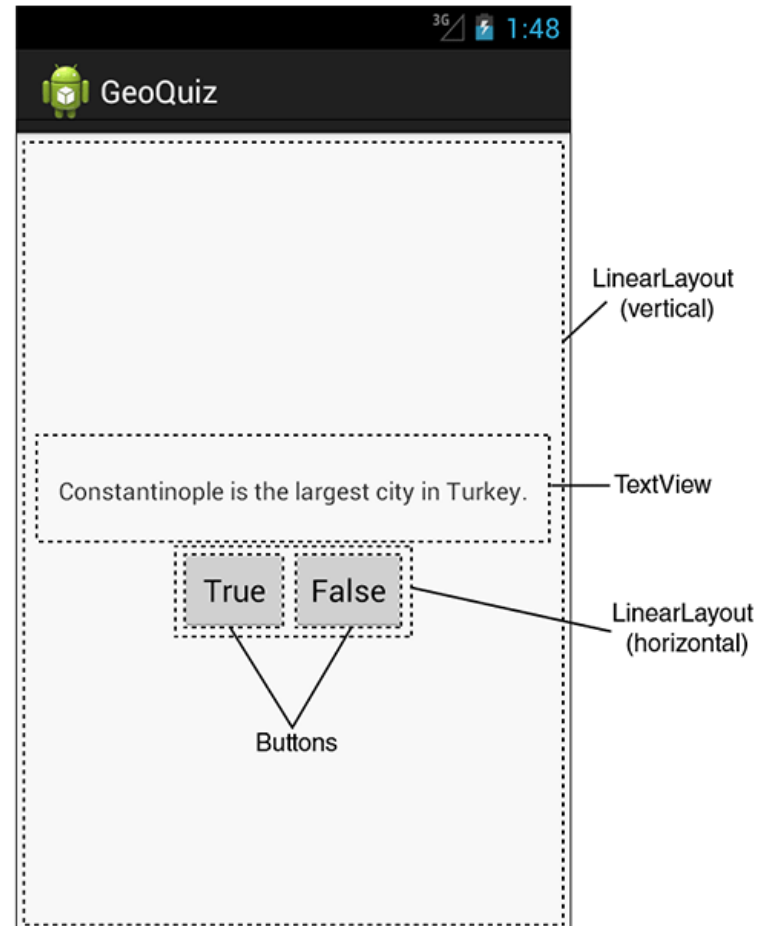
GeoQuiz: strings.xml File listing

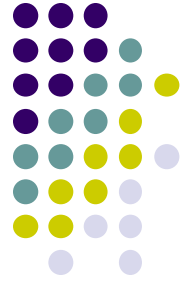


```
<?xml version="1.0" encoding="utf-8"?>
<resources>

  <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;

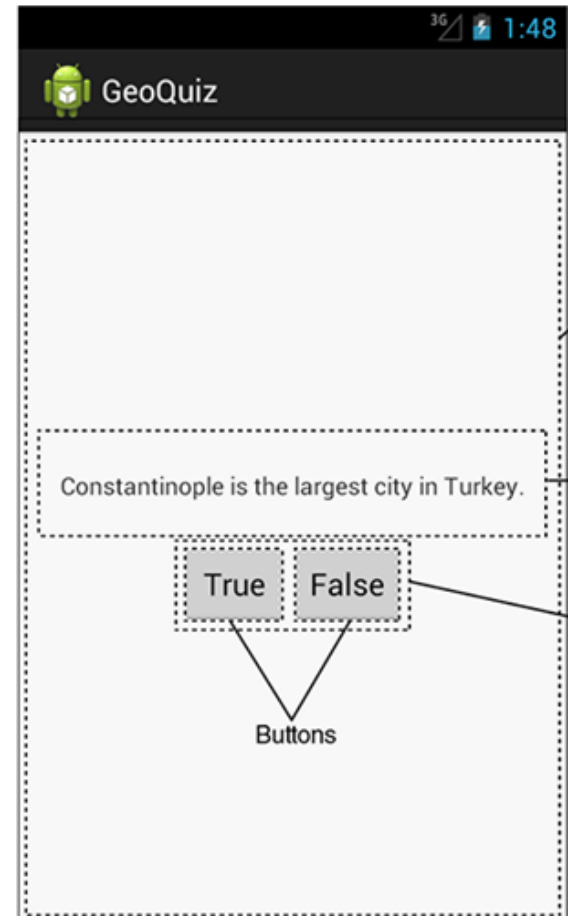
public class QuizActivity extends Activity {

    @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

- 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">
```

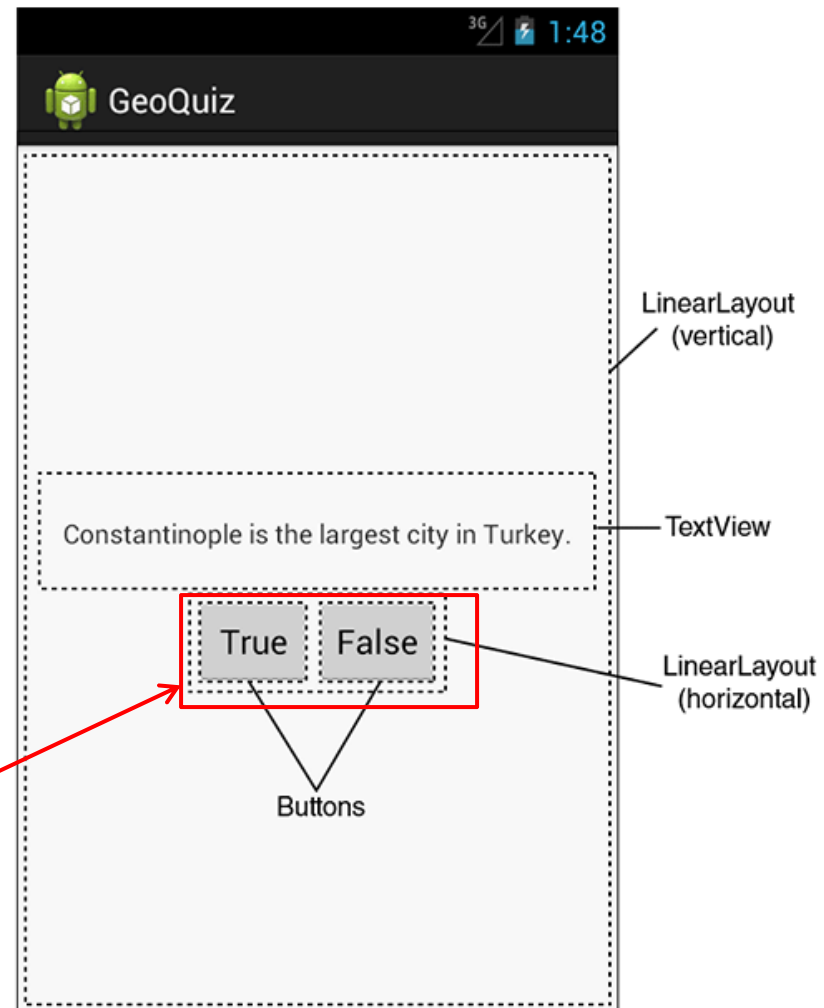
```
<Button
  android:id="@+id/true_button"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/true_button" />
```

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

```
</LinearLayout>
```

```
</LinearLayout>
```

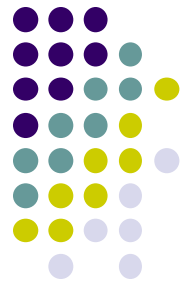
Write code in Java file to specify app's response when True/False buttons are clicked





2 Ways to Respond to Button Clicks

1. In XML: set `android:onClick` attribute
2. In java create a `ClickListener` object, override `onClick` method
 - typically done with anonymous inner class



Approach 1: Button that responds to Clicks

Reference: Head First Android

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

The Button definition from main.xml

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

XML
main.xml

The `onClick` attribute added to the Button. Pointing to the `onLoveButtonClicked` method.

2. In Java file (e.g. `AndroidLove.java`) declare and implement method/handler to take desired action

The new `onLoveButtonClicked` method that's referenced from the `android:onClick` Button attribute.

```
public class AndroidLove extends Activity {

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

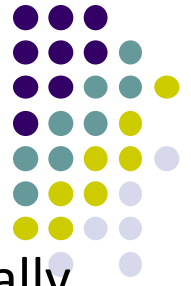
    public void onLoveButtonClicked(View view) {
        //doesn't do anything yet
    }

}
```

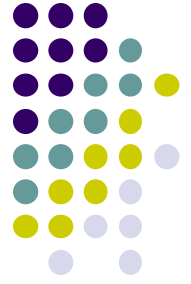
class Foo {
public...
}

AndroidLove.java

Background: User Interface Elements



- When views (buttons, widgets, etc) are declared in XML are actually Java classes within Android
- Using XML declarations, Android actually creates corresponding Java objects (called inflating a view)
- **View**
 - basic building block for Android UI
 - Android class that represents a rectangular area on the screen
 - Responsible for drawing and event handling
- View is the super class for:
 - Textview, Imageview
 - Controls such as buttons, spinners, seek bars, etc.
 - ViewGroups which in turn is the super class for layouts



ViewGroups - Layouts

- **Layouts:**
 - invisible containers that store other Views
 - Subclasses of ViewGroup
- Still a view but doesn't actually draw anything
- A container for other views
- Specifies options on how sub views (and view groups) are arranged

Approach 2: Create a ClickListener object, override onClick

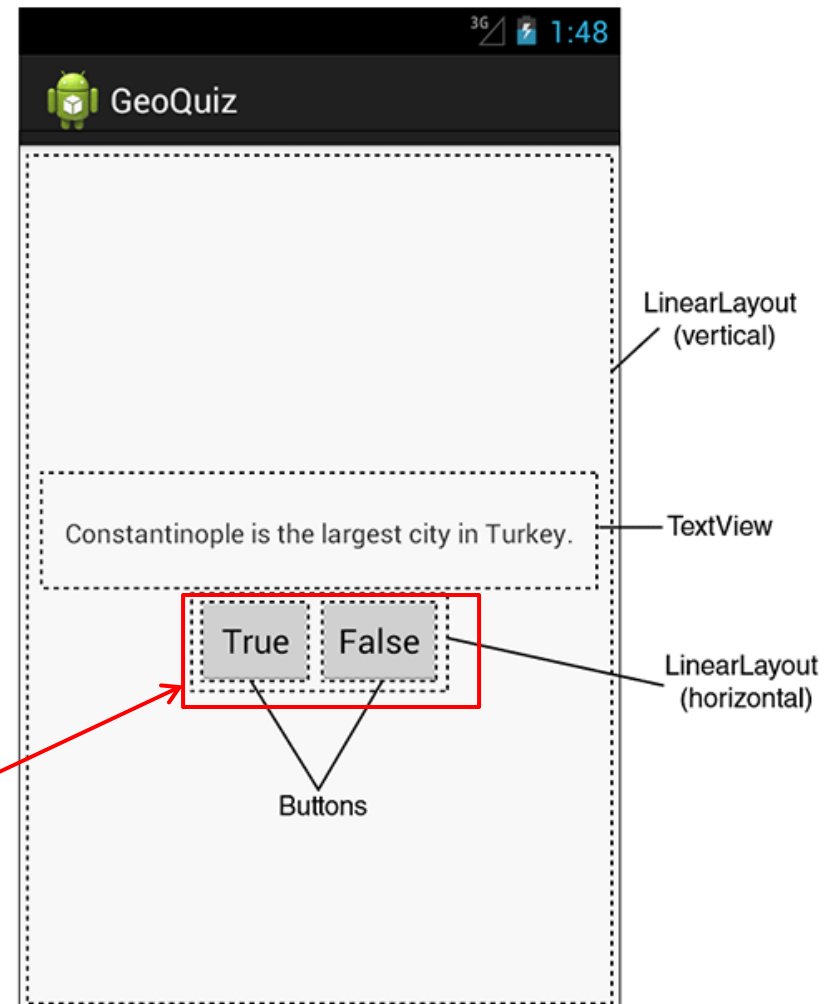


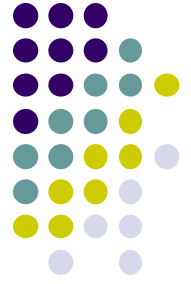
- First, get reference to Button in our Java file. How?

```
<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





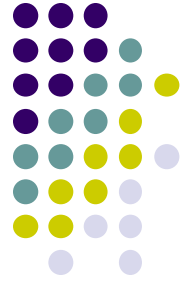
R.Java Constants

- During compilation, XML resources (drawables, layouts, strings, views with IDs, etc) are assigned constants
- Sample R.Java file

```
public final class R {  
    public static final class attr {}  
    public static final class drawable {  
        public static final int icon=0x7f020000;  
    }  
    public static final class id {  
        public static final int Button01=0x7f050000;  
    }  
    public static final class layout {  
        public static final int main=0x7f030000;  
    }  
    public static final class string {  
        public static final int app_name=0x7f040001;  
        public static final int haiku=0x7f040000;  
        public static final int love_button_text=0x7f040002;  
    }  
}
```

Interfaces grouping the constants.

Constants referring to XML resource.



Referring to Resources in Java File

- Can refer to resources in Java file using these constants
- Example

```
public static final class layout {  
    public static final int main=0x7f030000;  
}
```

Constant assigned to
R.layout.main at runtime

- In java file, R.java the constant corresponding to main.xml is argument of setContentView

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

Pass in layout file as
constant assigned to
R.layout.main



Referencing Widgets by ID

- Many widgets and containers appear only in XML. E.g. TextView
 - No need to be referenced in Java code
- To reference a widget in Java code, you need its **android:id**

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

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

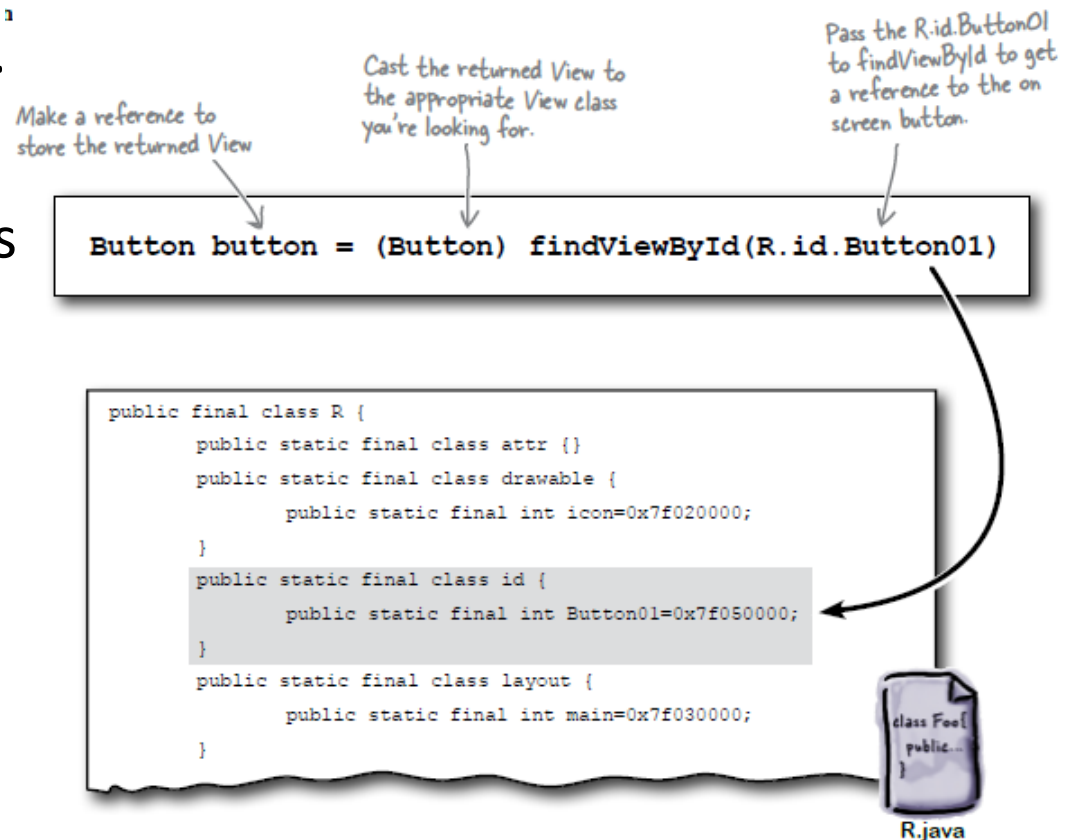
In java file, to reference/manipulate
view/widget use its ID to find it
(call findViewById())

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

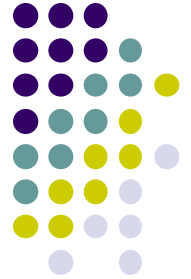
Getting View References



- **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);
```

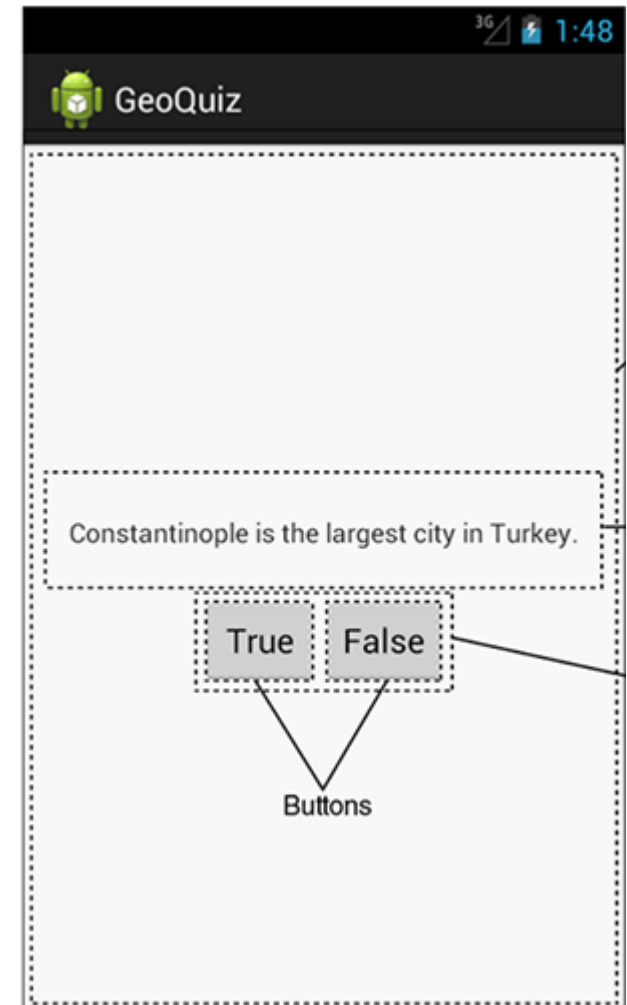
```
}
```

```
...  
}
```

**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

...

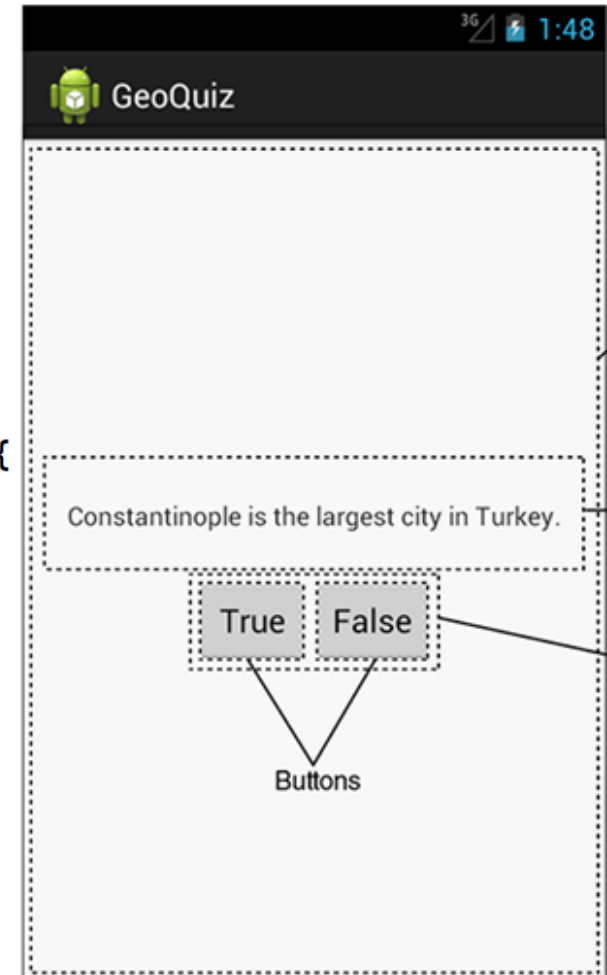
```
mTrueButton.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        // Does nothing yet, but soon!  
    }  
});
```

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

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

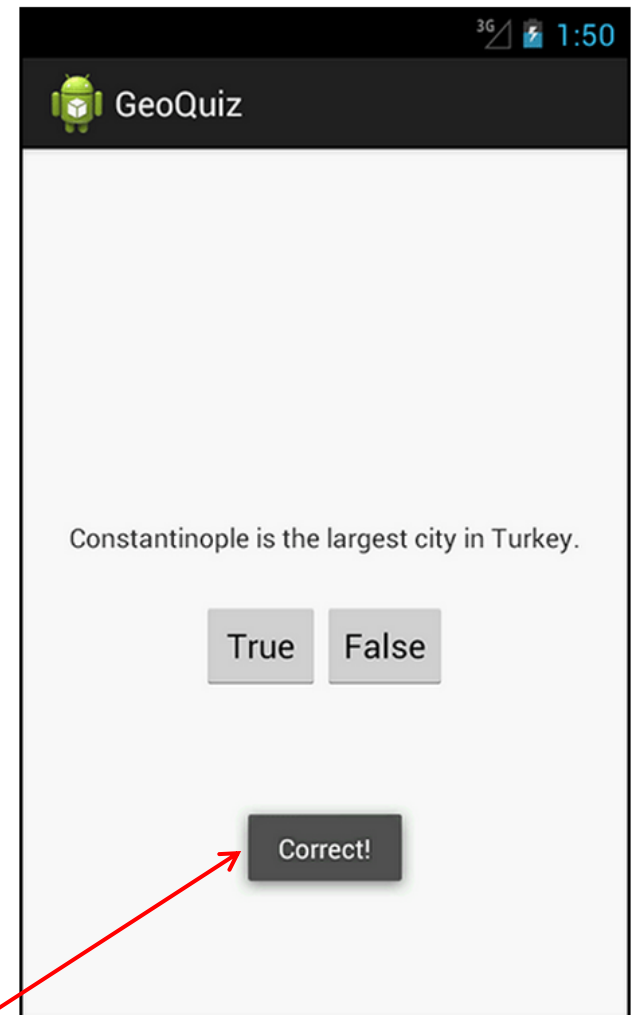




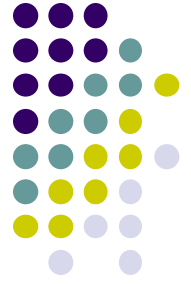
QuizActivity.java: Adding a Toast

- 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>
```



A toast



QuizActivity.java: Adding a Toast

- To create a toast, call the method:

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

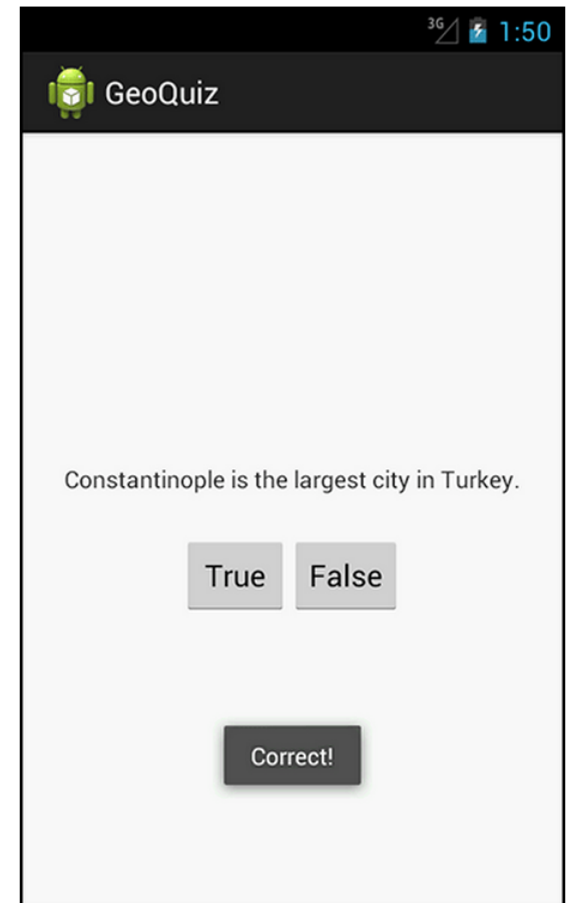
Instance of Activity
(Activity is a subclass
of context)

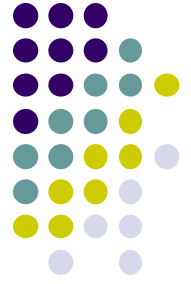
Resource ID of the
string that toast
should display

Constant to specify
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

...

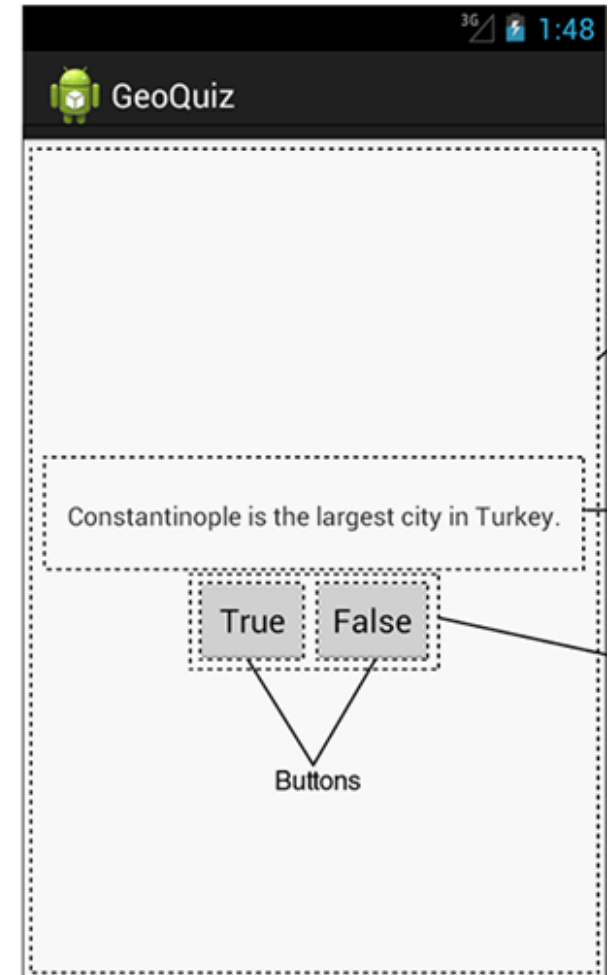
```
mTrueButton.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        Toast.makeText(QuizActivity.this,  
            R.string.incorrect_toast,  
            Toast.LENGTH_SHORT).show();  
    }  
});
```

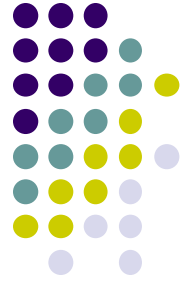
```
mFalseButton.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        Toast.makeText(QuizActivity.this,  
            R.string.correct_toast,  
            Toast.LENGTH_SHORT).show();  
    }  
});
```

1. Set Listener Object
For mTrueButton

3. Override onClick method
Make a toast

2. Create listener
object as anonymous
inner object





```
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;

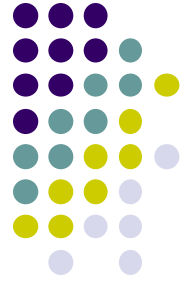
public class QuizActivity extends Activity {

    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();
            }
        });
    }
}
```

QuizActivity.java: Complete Listing



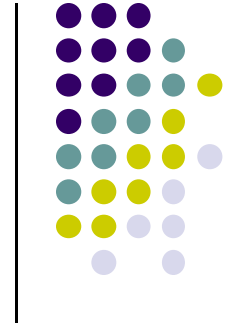
```
mFalseButton = (Button)findViewById(R.id.false_button);  
mFalseButton.setOnClickListener(new View.OnClickListener() {
```

```
    @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 UI in Java

USB debugging

Debug mode when USB is connected

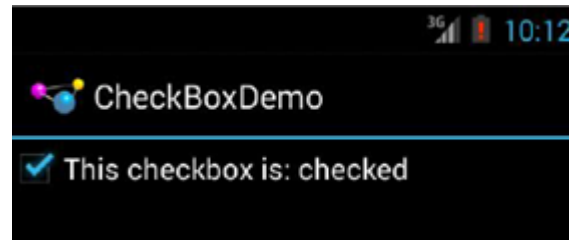
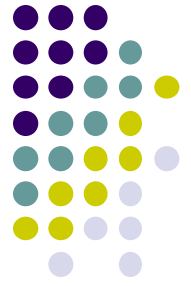


Recall: Checkbox

- Has 2 states: **checked** and **unchecked**
- Clicking on checkbox toggles between these 2 states
- Used to indicate a choice (e.g. Add rush delivery)
- Since Checkbox widget inherits from TextView, its properties (e.g. **android:textColor**) can be used to format checkbox
- XML code to create Checkbox:

```
<?xml version="1.0" encoding="utf-8"?>
<CheckBox xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/check"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/unchecked" />
```

Making Checkbox Responsive



- 2 ways to make Checkbox responsive:
 1. Set **android:onClick** attribute or
 2. Create a ClickListener object, override onClick method, and register it with the checkbox
- In Java code, the following commands may be used
 - **isChecked()**: to determine if checkbox has been checked
 - **setChecked()**: to force checkbox into checked or unchecked state
 - **toggle()**: to toggle current checkbox setting

Checkbox Example Java Code



```
package com.commonware.android.checkbox;

import android.app.Activity;
import android.os.Bundle;
import android.widget.CheckBox;
import android.widget.CompoundButton;

public class CheckBoxDemo extends Activity implements
    CompoundButton.OnCheckedChangeListener {
    CheckBox cb;

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

        cb=(CheckBox)findViewById(R.id.check);
        cb.setOnCheckedChangeListener(this);
    }

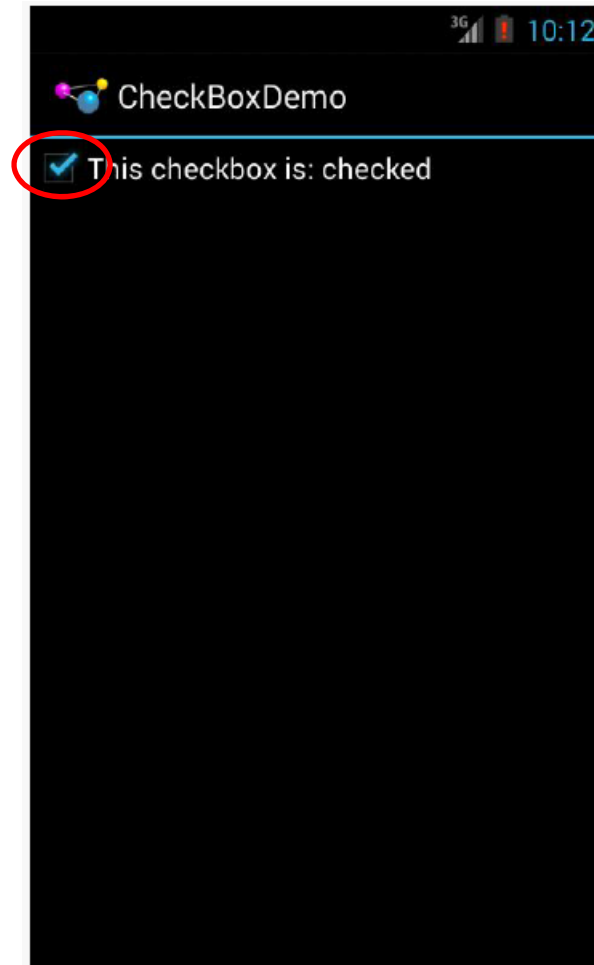
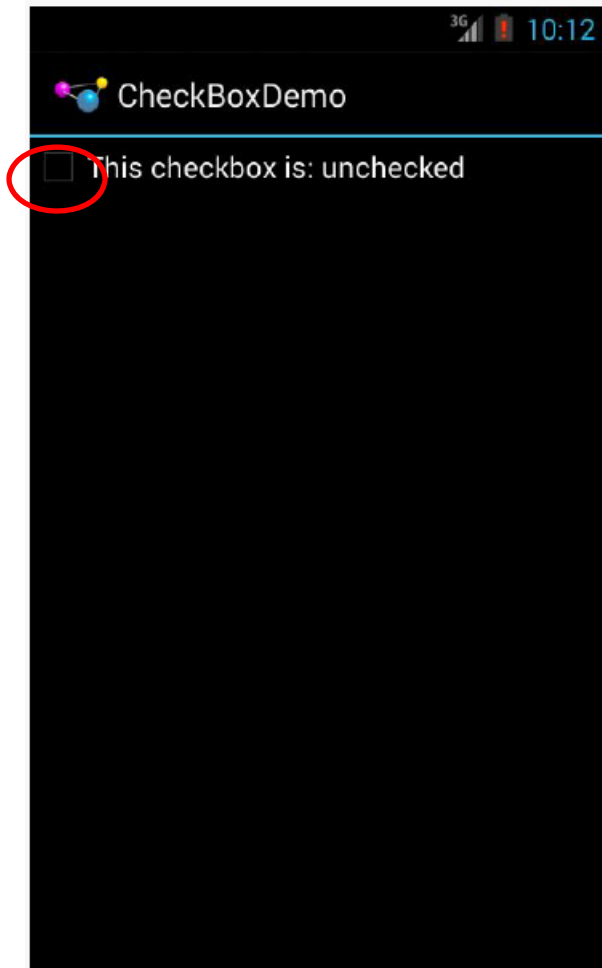
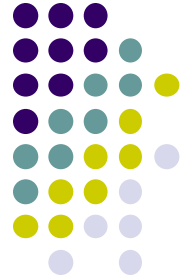
    public void onCheckedChanged(CompoundButton buttonView,
        boolean isChecked) {
        if (isChecked) {
            cb.setText(R.string.checked);
        }
        else {
            cb.setText(R.string.unchecked);
        }
    }
}
```

Checkbox inherits from
CompoundButton

Register listener
OnCheckedChangeListener
to be notified when checkbox
state changes

Callback, called
When checkbox
state changes

Checkbox Example Result





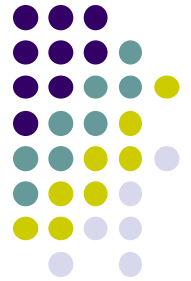
Important Android Packages

- Android programs usually import packages at top. E.g.

```
package com.commonware.android.checkbox;  
  
import android.app.Activity;  
import android.os.Bundle;  
import android.widget.CheckBox;  
import android.widget.CompoundButton;
```

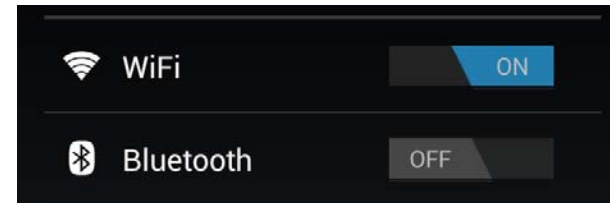
- Important packages
 - **android*** Android application
 - **dalvik*** Dalvik virtual machine support classes
 - **java.*** Core classes and generic utilities (networking, security, math, etc)
 - **org.apache.http:** HTTP protocol support

Ref: Introduction to Android Programming, Annuzzi, Darcey & Conder



Toggle Button

- ToggleButton and Switches
 - Like CheckBox has 2 states
 - However, visually shows states on and off text

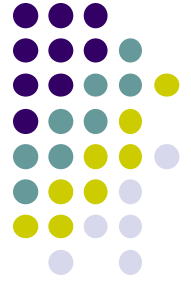


- XML code to create ToggleButton

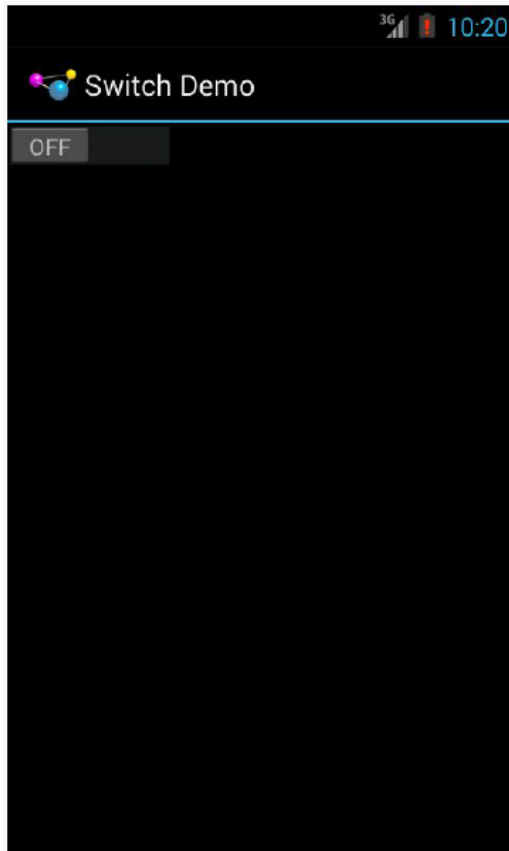
```
<?xml version="1.0" encoding="utf-8"?>  
<ToggleButton xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/toggle"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content" />
```

- Can also set up an `onCheckedChangeListener` to be notified when user changes ToggleButton state

Switch Widget



- Android Switch widget shows state via small ON/OFF slider
- Added in API level 14





Switch Widget

- XML code for creating Switch

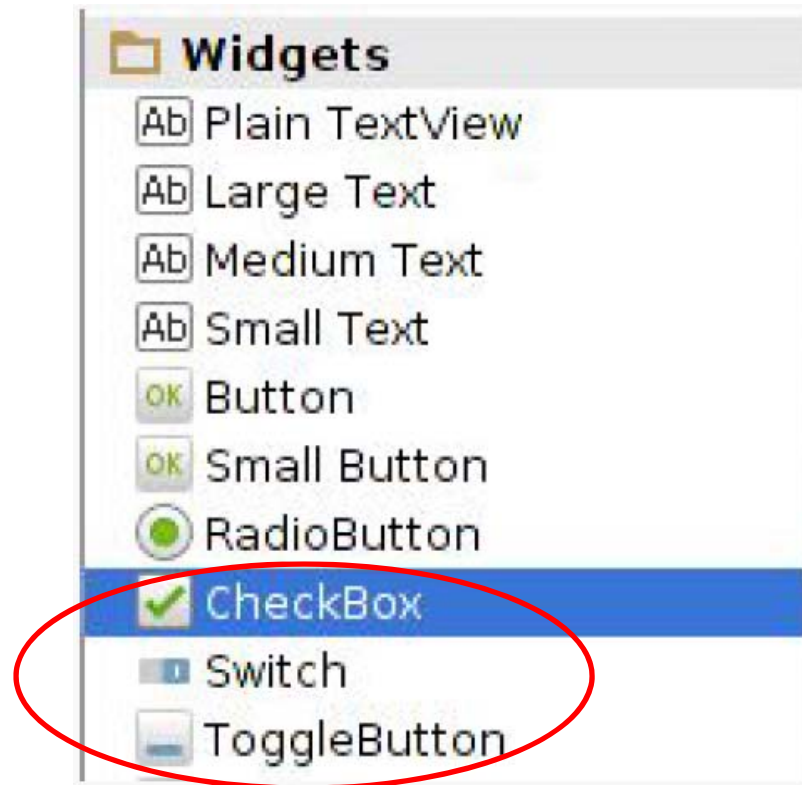
```
<?xml version="1.0" encoding="utf-8"?>  
<Switch xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/toggle"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content" />
```

- **Checkbox, ToggleButton and Switch** inherit from **CompoundButton**
- Common API for
 - `toggle()`
 - `isChecked()`
 - `setChecked()`

Creating Checkbox, ToggleButton or Switch in Android Studio

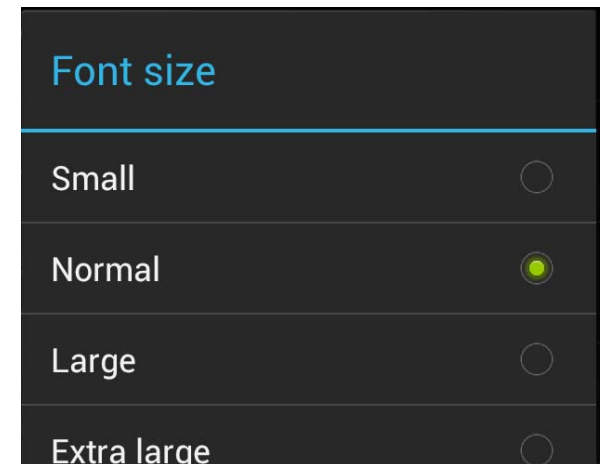
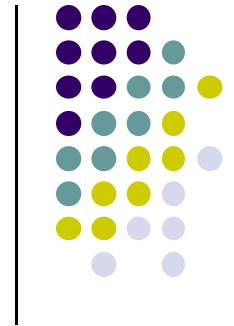


- Checkbox, Togglebutton and Switch widgets are in Android Studio palette
- Can drag and drop them unto app screen then configure properties



RadioButton and RadioGroup

- Select only 1 option from a set
- set onClick method for each button
 - generally same method
- Inherits from **CompoundButton** which inherits from **TextView**
 - Format using TextView properties (font, style, color, etc)
- Can use methods **isChecked()**, **toggle()**
- Collected in RadioGroup
 - sub class of LinearLayout
 - Can have vertical or horizontal orientation



RadioButton and RadioGroup

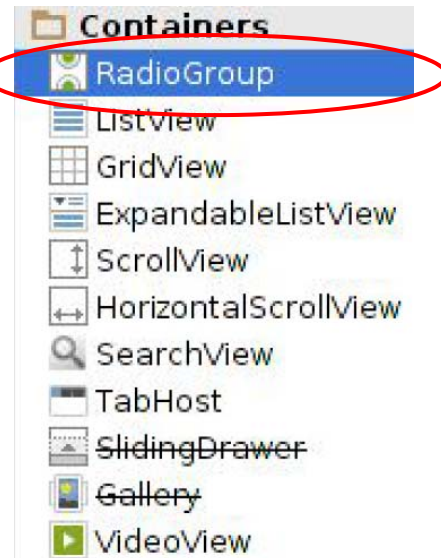
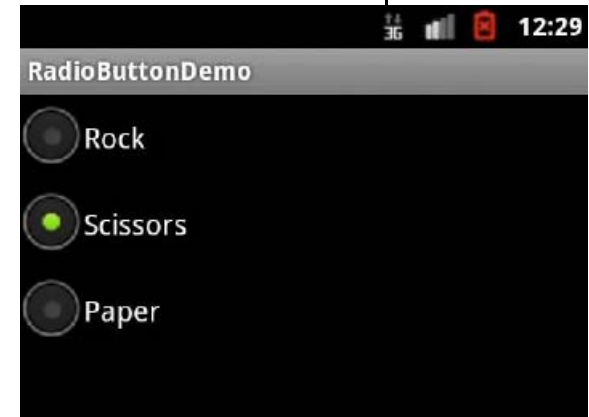
- XML code to create Radio Button

```
<?xml version="1.0" encoding="utf-8"?>
<RadioGroup
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  >
  <RadioButton android:id="@+id/radio1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/rock" />

  <RadioButton android:id="@+id/radio2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/scissors" />

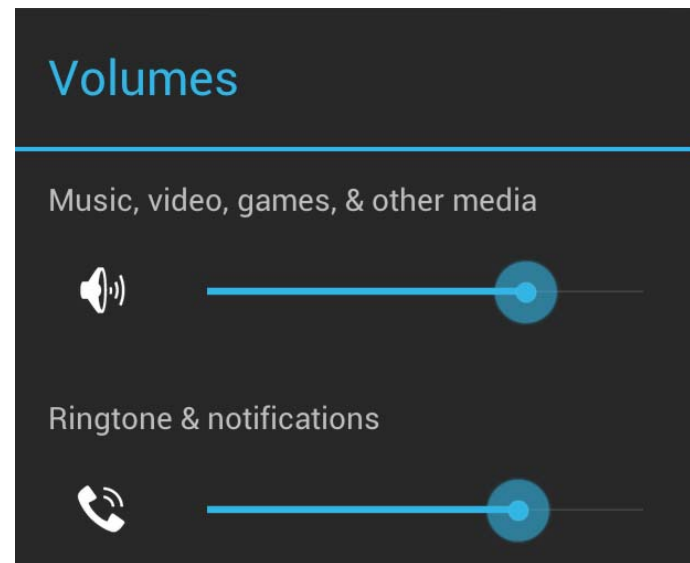
  <RadioButton android:id="@+id/radio3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/paper" />
</RadioGroup>
```

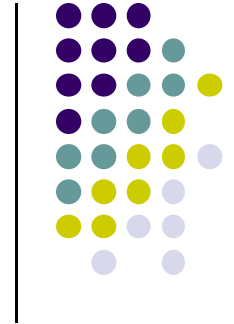
- Available in Android Studio palette



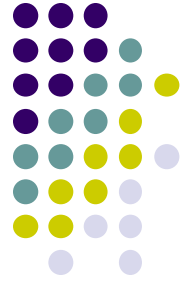
SeekBar

- a slider
- Subclass of progress bar
- implement a [SeekBar.OnSeekBarChangeListener](#) to respond to changes in setting





WebView Widget



WebView Widget

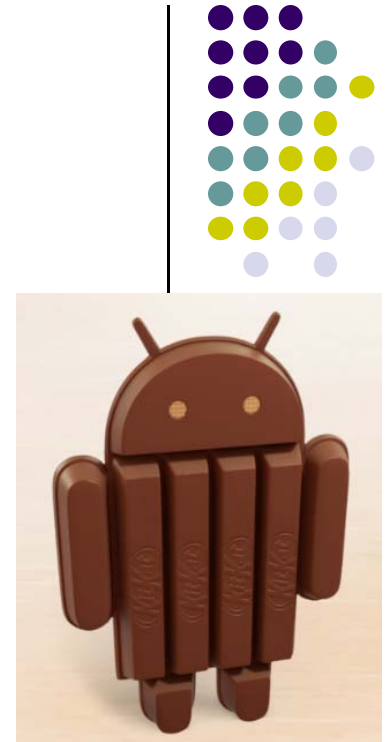
- A View that display web pages
 - Can be used for creating your own web browser
 - OR just display some online content inside your app
- Uses WebKit rendering engine (lots of memory)
 - <http://www.webkit.org/>
- Webkit used in many web browsers including Safari



- Web pages in WebView same look same as in Safari

WebView

- Android 4.4, API level 19 added **Chromium** as alternative to WebKit
- [Chromium](http://www.chromium.org/Home): <http://www.chromium.org/Home>
- "Chromium WebView provides broad support for HTML5, CSS3, and JavaScript.
- Supports most HTML5 features available in Chrome.
- Also has updated JavaScript Engine (V8) with vastly improved JavaScript performance. "



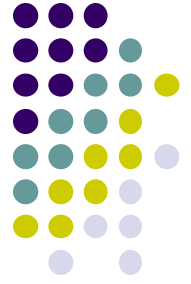


WebView Widget Functionality

- Display Web page containing HTML, CSS, Javascript
- Navigation history of URLs to support forward and backwards
- zoom in and out
- perform searches
- Additional functionality:
 - capture images of page
 - Search page for string
 - Deal with cookies on a per application basis



Scenarios for Using Webview NOT Browser



- Provide info requiring periodic updates.
 - Example: end user agreement or user guide (no need to update app)
- OR display **documents hosted online**
- OR access **application data available on Internet**
- OR **display ads**

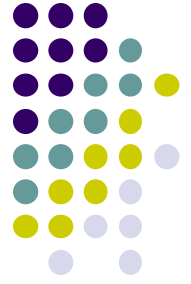
<http://developer.android.com/guide/webapps/webview.html>



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

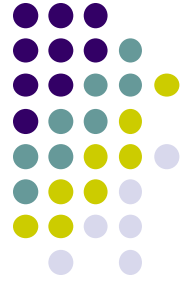
- In onCreate, use loadURL to load website
- If website contains Javascript, enable Javascript

```
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");  
    }  
}
```

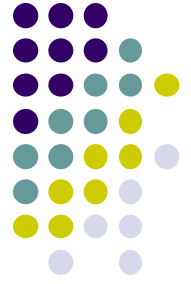
loadUrl()

```
@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");
}
```



- loadUrl() Works with
 - **http://** and **https://** URLs
 - **file//** URLs pointing to local filesystem
 - **file:///** android_asset/ URLs pointing to app's assets (later)
 - **content://** URLs pointing to content provider that is streaming published content



WebView Example

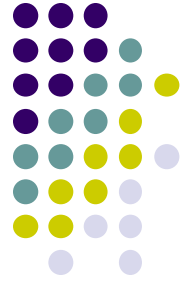
- Add permission to AndroidManifest.xml for app to use Internet
- Also change style so no title bar

```
<?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" />

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:name=".HelloWebView"
            android:label="@string/app_name"
            android:theme="@android:style/Theme.NoTitleBar" >
```



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!");  
    }  
}
```

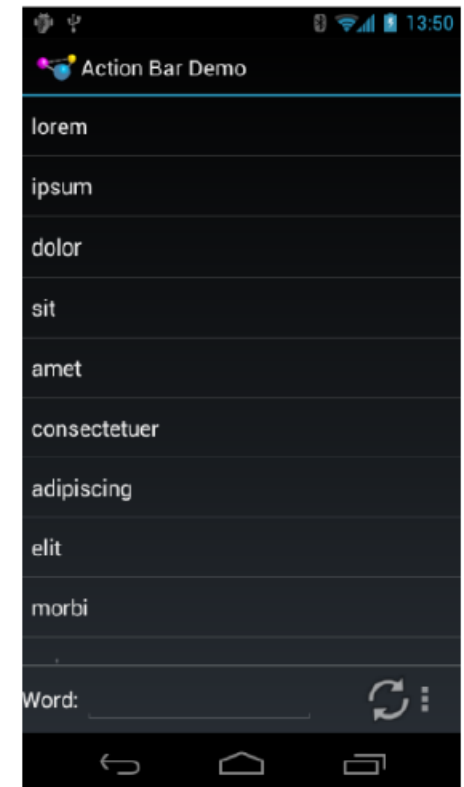
- Android app: No need to write a main
- Just define app components by creating sub-classes of base classes already defined in Android
- 4 main types of Android app components:
 - Activities (already seen this)
 - Services
 - Content providers
 - Broadcast receivers



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

Activity





Recall: Activities

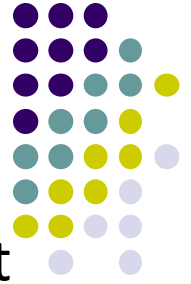
- Each activity controls 1 or more screens
- Activities independent of each other
- Can be coupled by control or data
- App Activities are sub-class of **Activity** class
- Example:

```
public class HelloWebView extends Activity {
```

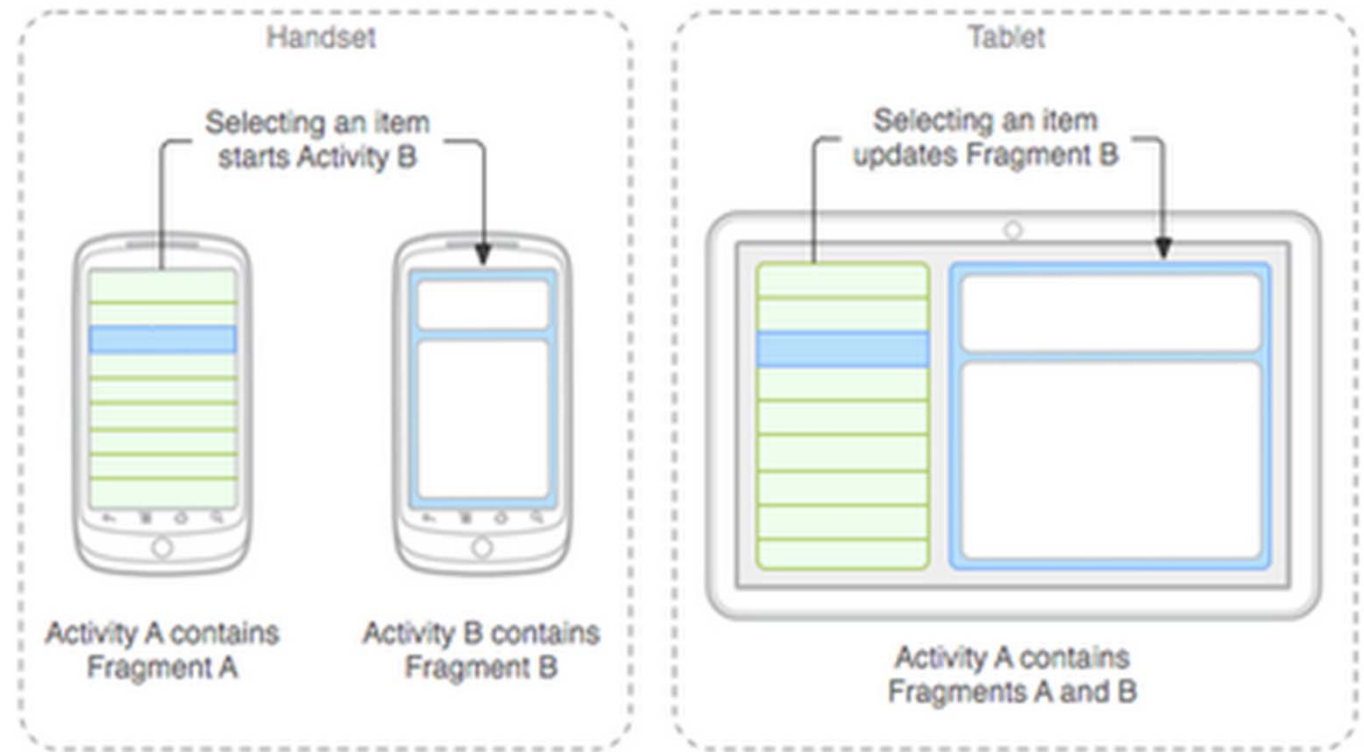
Activity



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

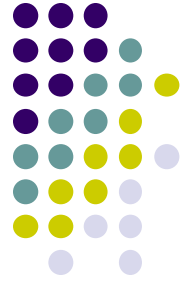


Services

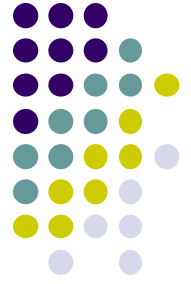


- Activities are short-lived, can be shut down anytime (e.g when user presses back button)
- Services keep running in background
- Minimal need to interact with (independent of) any activity
- Typically an activity will control a service -- start it, pause it, get data from it
- Similar to Linux/Unix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

Android Platform Services

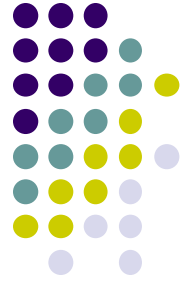


- Android Services can either be on:
 - Android Platform (local)
 - Google (remote)
- Android platform services:
 - **LocationManager**: location-based services.
 - **ViewManager** and **WindowManager**: Manage display and User Interface
 - **AccessibilityManager**: accessibility, support for physically 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.



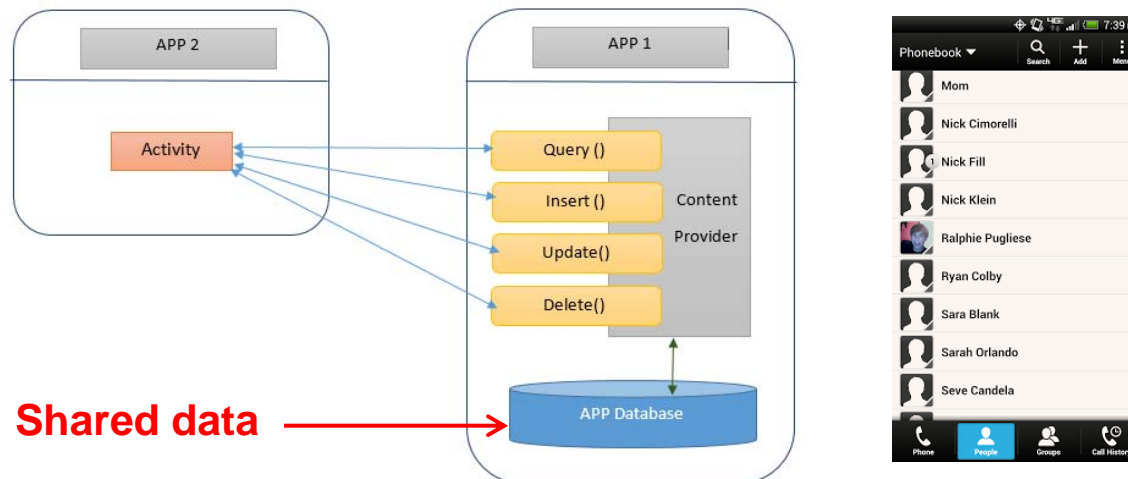
Google Services

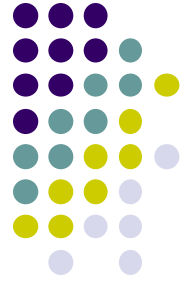
- Maps
- Location-based services
- Game Services
- Authorization APIs
- Google Plus
- Play Services
- In-app Billing
- Google Cloud Messaging
- Google Analytics
- Google AdMob ads



Content Providers

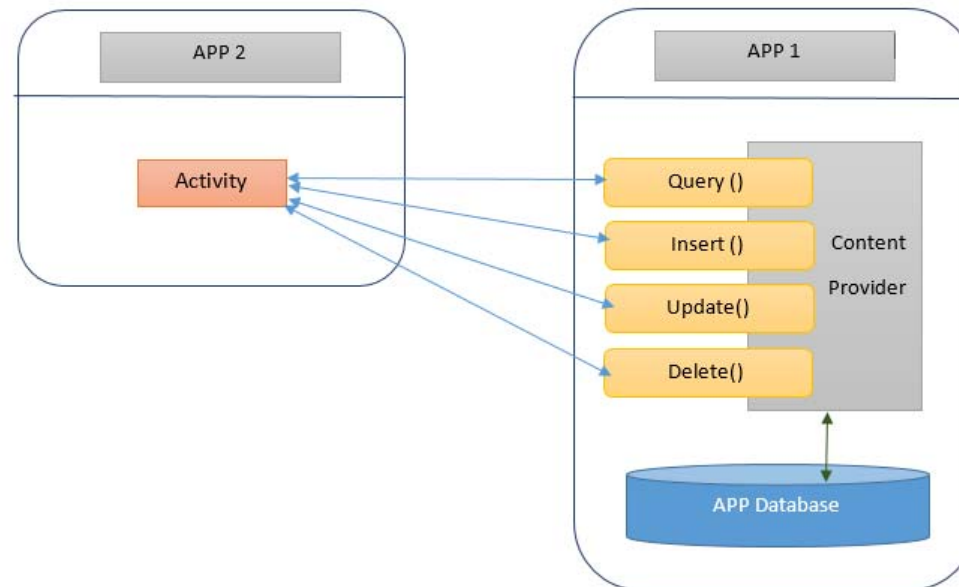
- Android apps can share data
- Content Provider:
 - Abstracts shareable data, makes it accessible through methods
 - Applications can access that shared data by calling methods for the relevant **content provider**
- Example: We can write an app that:
 - Retrieve's contacts list from contacts content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



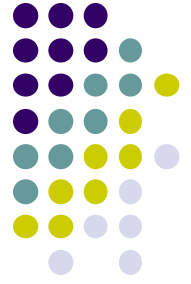


Broadcast Receivers

- The system, or applications, periodically *broadcasts* events
- Example broadcasts:
 - Battery getting low
 - Screen turns off
 - 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



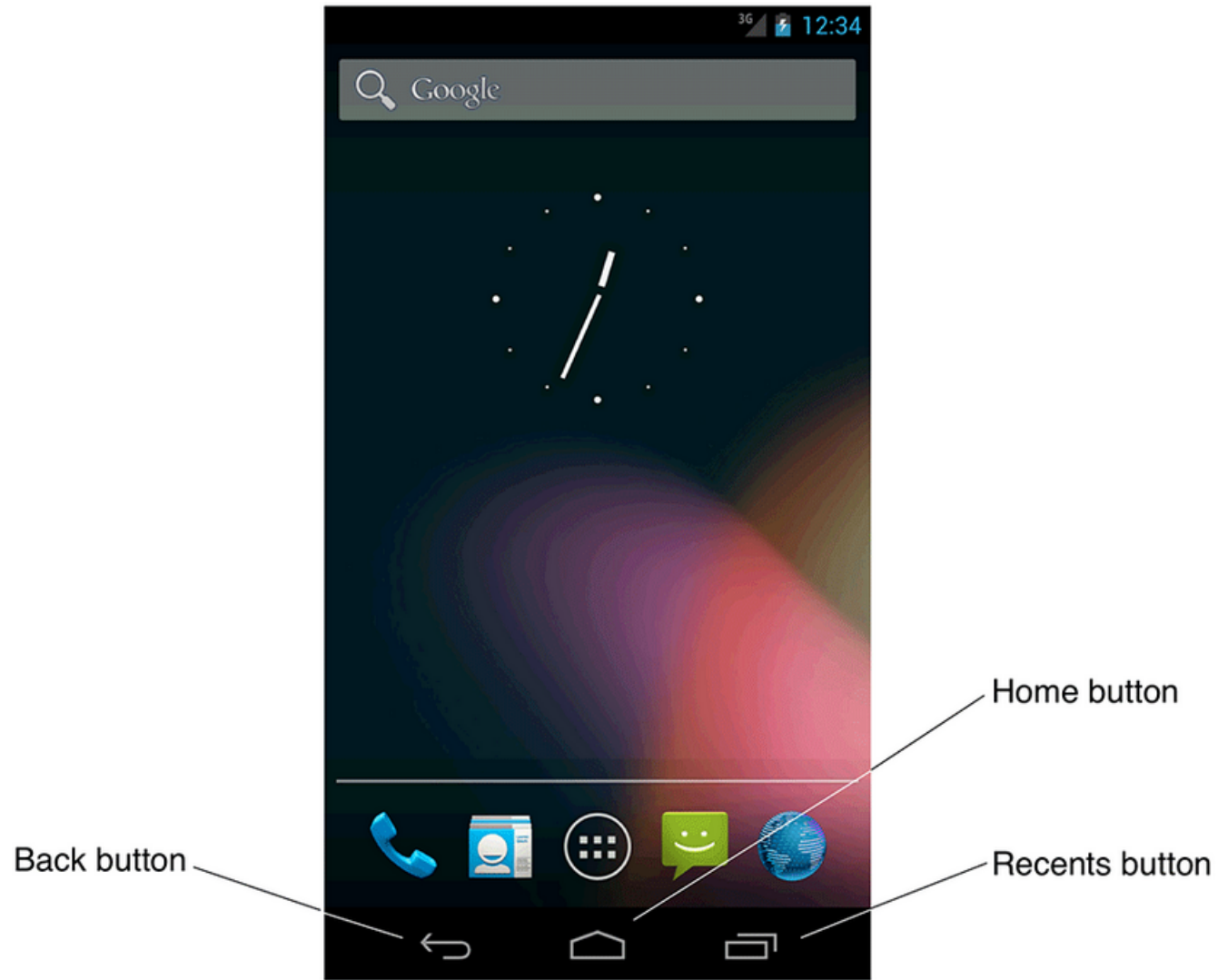
Android's Process Model



Android's Process Model

- When user launches an app, Android forks a copy of a process called **zygote** that receives
 - A copy of of the Virtual Machine (Dalvik)
 - A copy of Android framework classes (e.g. Activity and Button)
 - A copy of user's app classes loaded from their APK file
 - Any objects created by app or framework classes

Recall: Home, Back and Recents Button



Android Activity Stack (Back vs Home Button)



- Android maintains activity stack
- While an app is running,
 - Pressing **Back** button **destroys the app's activity** and returns app to whatever user was doing previously (e.g. HOME screen)
 - If **Home** button is pressed, activity is kept around for some time, **NOT destroyed** immediately

Most recently created is at Top

Activity 1

Activity 2

Activity 3

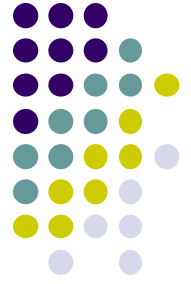


Activity N

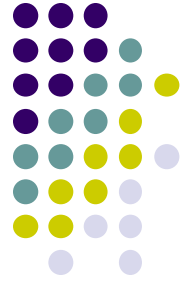
User currently interacting with me

Pressing Back or destroying A1 will bring me to the top

If Activities above me use too many resources, I'll be destroyed!



Android Activity LifeCycle

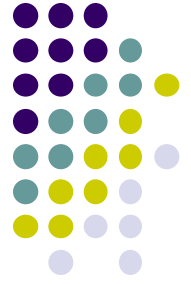


Starting Activities

- Android applications don't start with a call to `main(String[])`
- Instead callbacks invoked corresponding to app state.
- Examples:
 - When activity is created, its `onCreate()` method invoked
 - When activity is paused, its `onPause()` method invoked
- callback methods also invoked to destroy Activity /app

Activity Callbacks

- onCreate()
- onStart()
- onResume()
- onPause()
- onStop()
- onRestart()
- onDestroy()



Understanding the Lifecycle



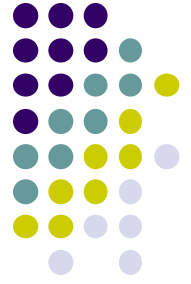
- Many things could happen while app is running
 - Incoming call or text message, user switches to another app, etc
- Well designed app should NOT:
 - Crash if interrupted or user switches to other app
 - Consume valuable system resources when user is not actively using it.
 - Lose the user's state/progress (e.g state of chess game app) if they leave your app and return to it at a later time.
 - Crash or lose the user's progress when the screen rotates between landscape and portrait orientation.
 - E.g. Youtube video should continue at correct point after rotation
- To ensure the above, appropriate callback methods must be invoked appropriately

<http://developer.android.com/training/basics/activity-lifecycle/starting.html>



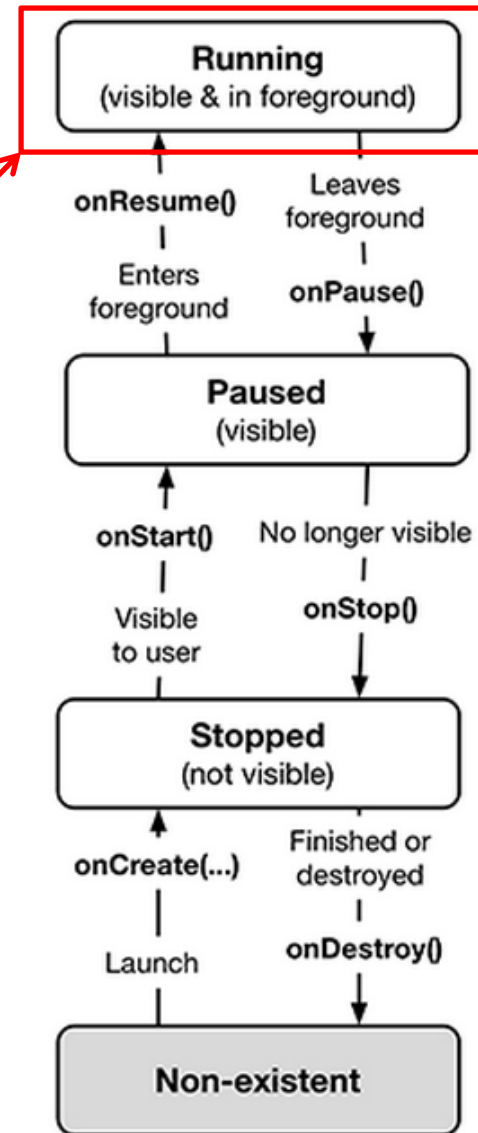
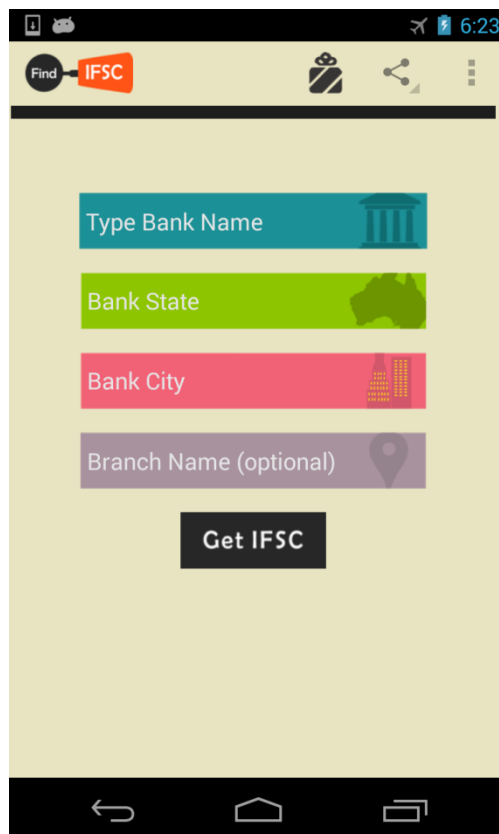
onCreate()

- The following operations are typically performed in onCreate() method:
 - Inflate widgets and put them on the screen (e.g. using layout files with setContentView())
 - Getting references to inflated widgets (using findViewById())
 - Setting widget listeners to handle user interaction
- **Note:** Android OS calls apps' onCreate() method, NOT the app



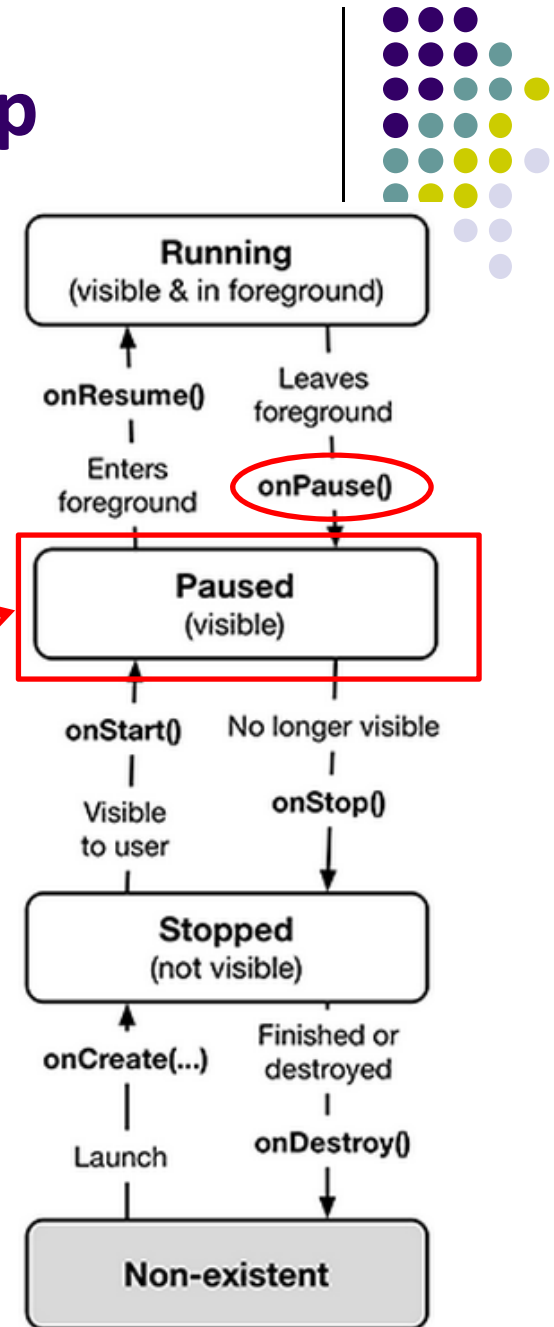
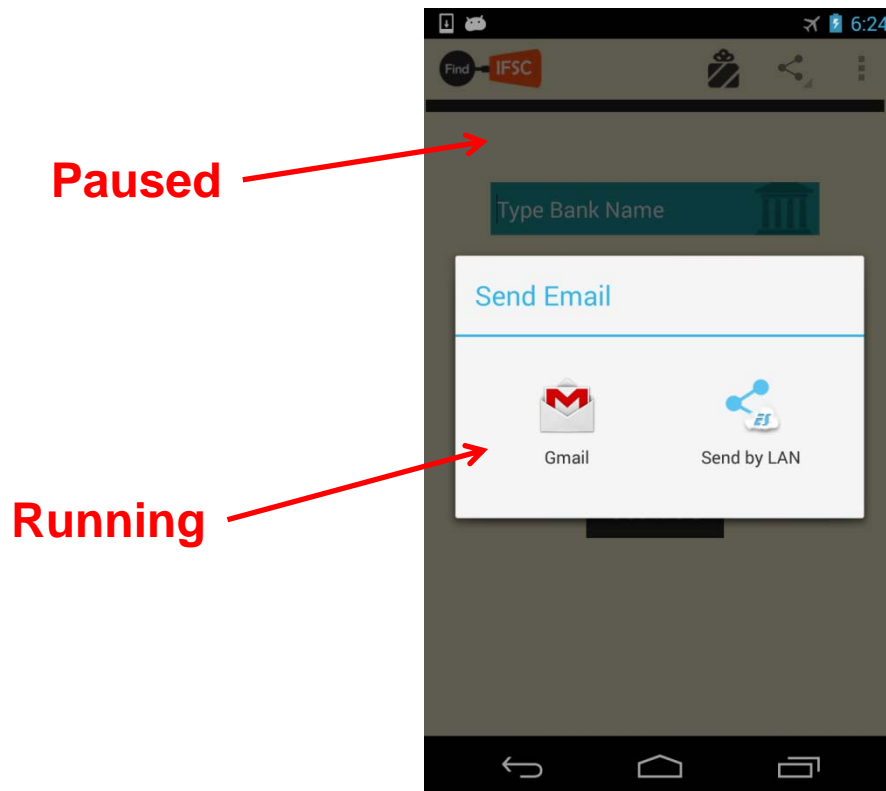
Activity State Diagram: Running App

- A running app is one that the user is currently using or interacting with
 - App is visible and in foreground



Activity State Diagram: Paused App

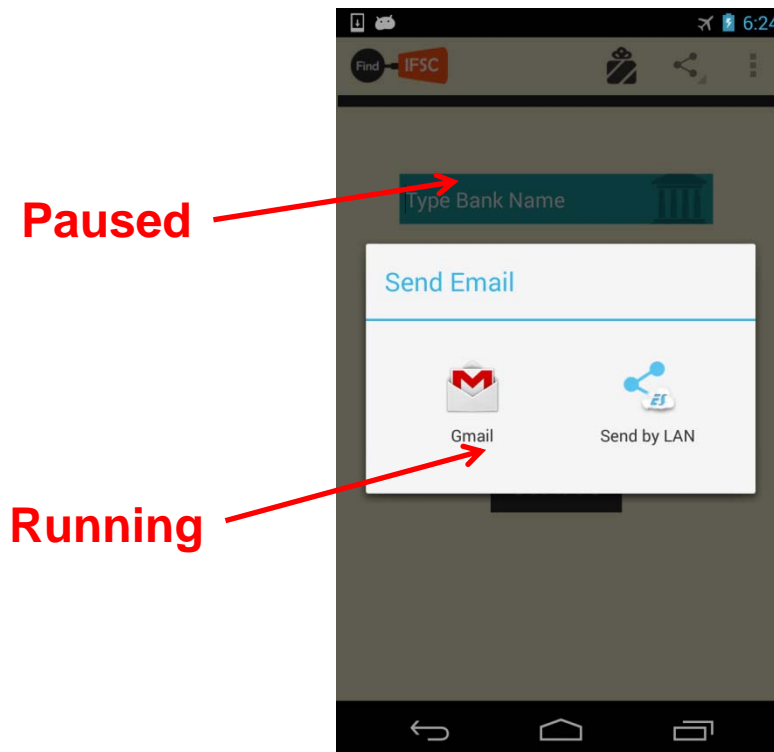
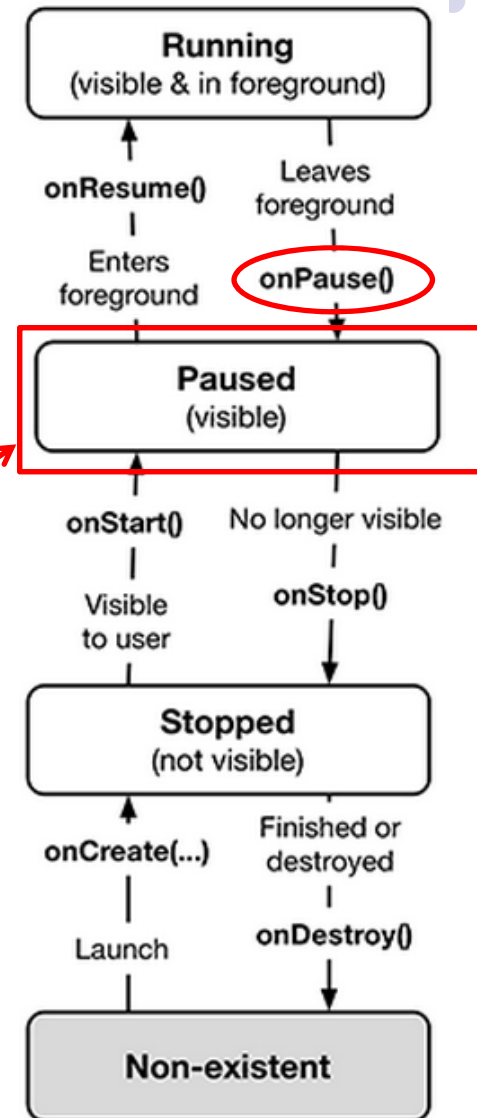
- An app is **paused** if it is **visible** but no longer in foreground
- E.g. blocked by a pop-up dialog box
- App's **onPause()** method is called to transition from running to paused state



Activity State Diagram: onPause() Method

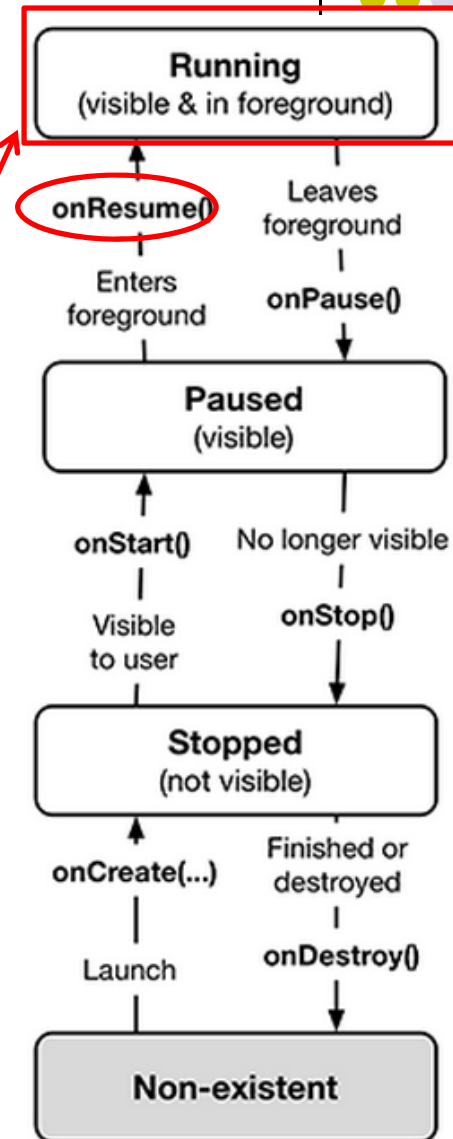
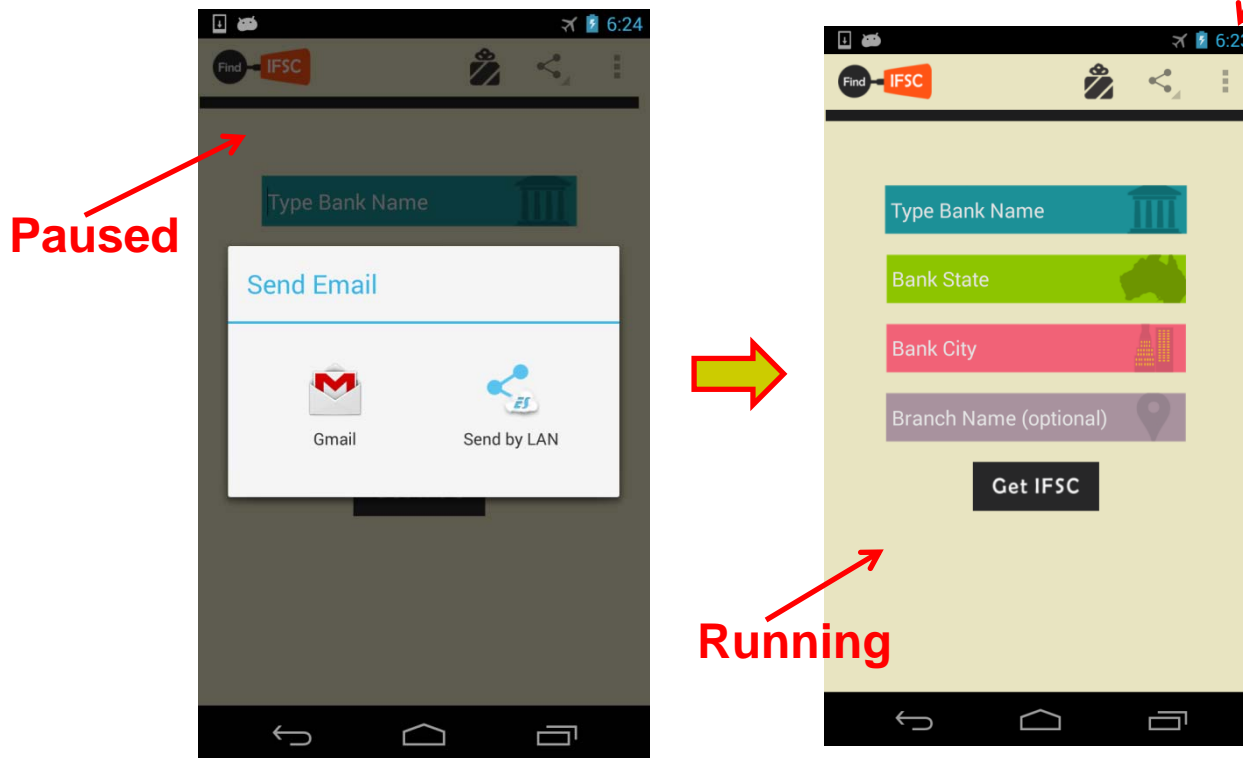


- Typical actions taken in onPause() method
 - Stop animations and CPU intensive tasks
 - Stop listening for GPS, broadcast information
 - Release handles to sensors (e.g GPS, camera)
 - Stop audio and video if appropriate



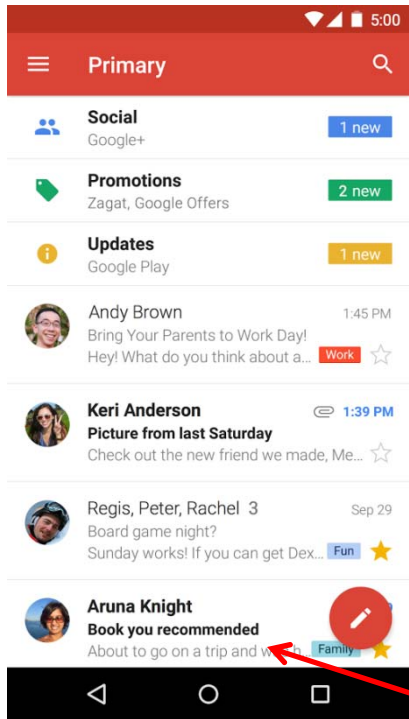
Activity State Diagram: Resuming Paused App

- A **paused** app resumes **running** if it becomes fully visible and in foreground
- E.g. pop-up dialog box blocking it goes away
- App's **onResume()** method is called to transition from **paused** to **running** state

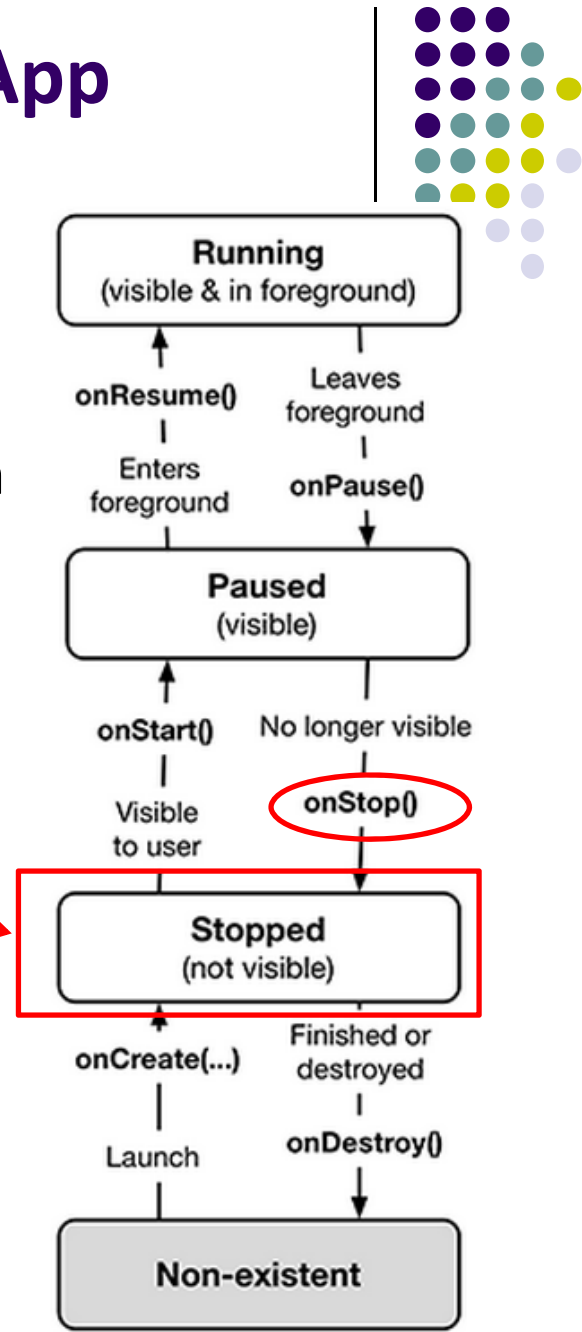
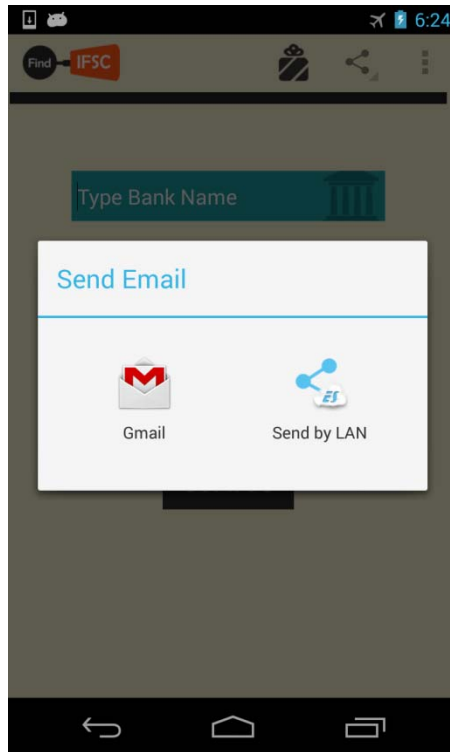


Activity State Diagram: Stopped App

- An app is **stopped** if it **no longer visible and no longer in foreground**
- E.g. user starts using another app
- App's **onStop()** method is called to transition from paused to stopped state



Running



onStop() Method

- An activity is stopped when the user:
 - Receives phone call
 - Opens Recent Apps window and starts a new application
 - Performs action in activity that starts another activity in the application
- Activity instance and variables of stopped app are retained but no code is being executed by the activity
- If activity is stopped, in onStop() method, well behaved apps should
 - save progress to enable seamless restart later
 - Release all resources and save information (persistence)



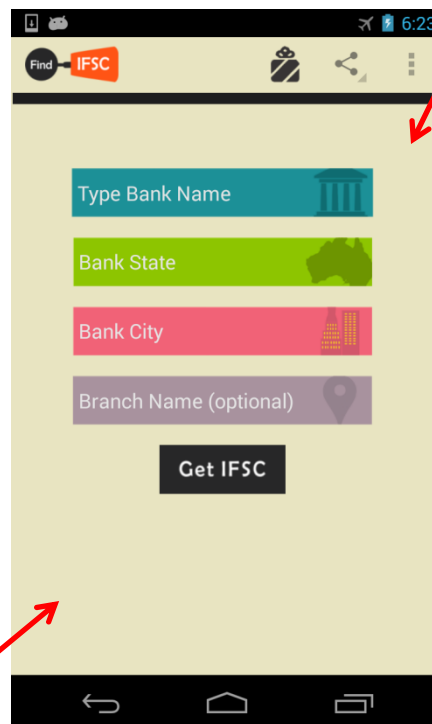
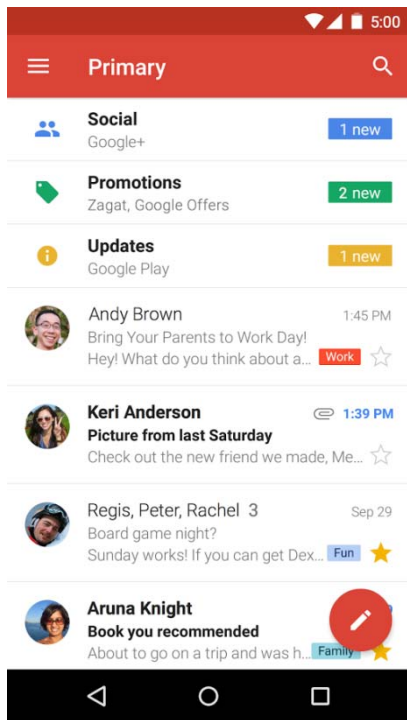


Saving State

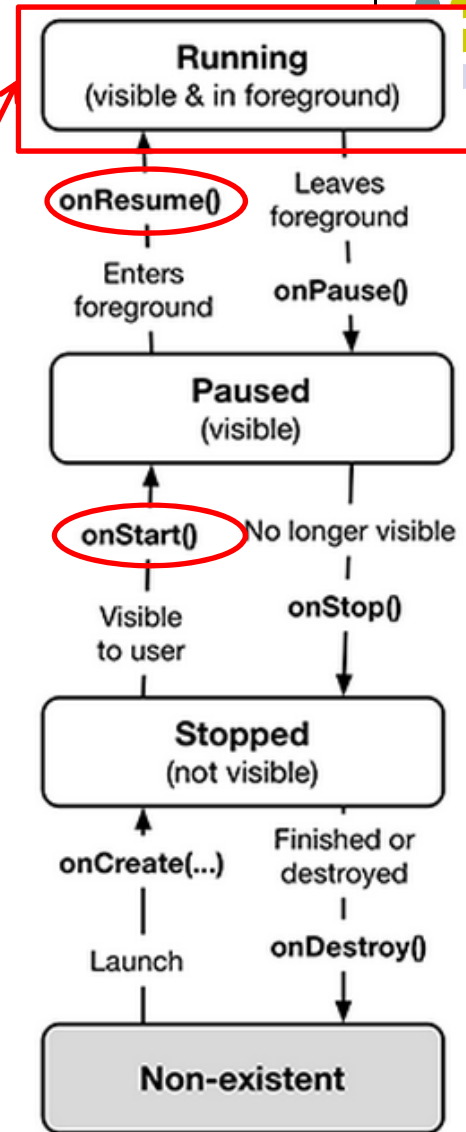
- If activities are paused or stopped, their states (instance vars) are retained
 - Even if activity is not in foreground
- When activity is destroyed the Activity object is destroyed
 - can save information via `onSaveInstanceState(Bundle outState)` method.
 - Write data to Bundle (a data structure)
 - Bundle given back when restarted

Activity State Diagram: Stopped App

- A **stopped** app can go back into **running** state if becomes visible and in foreground
- App's **onStop()**, **onRestart()** and **onResume()** methods called to transition from **stopped** to **running** state

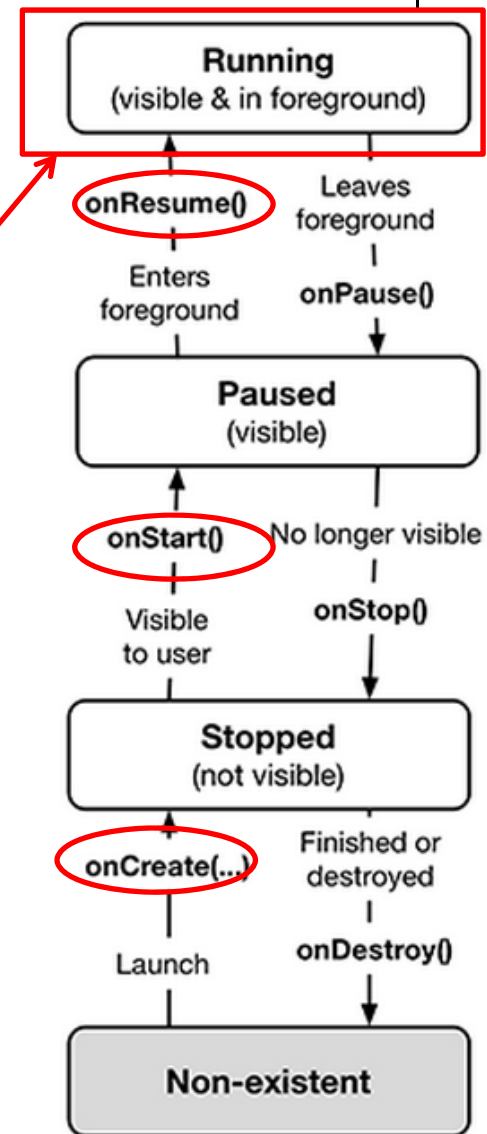
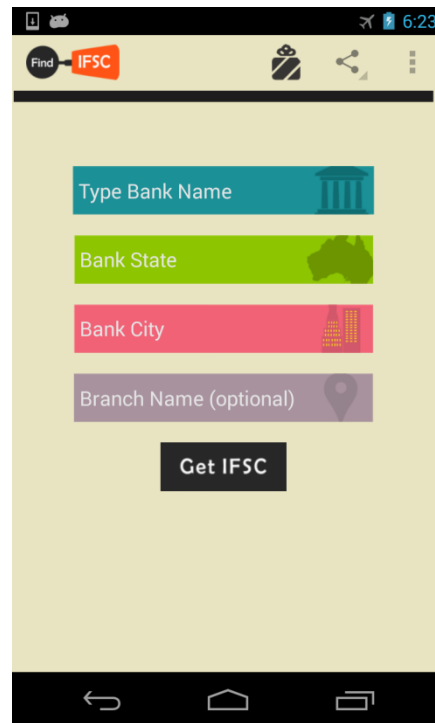


Running



Activity State Diagram: Starting New App

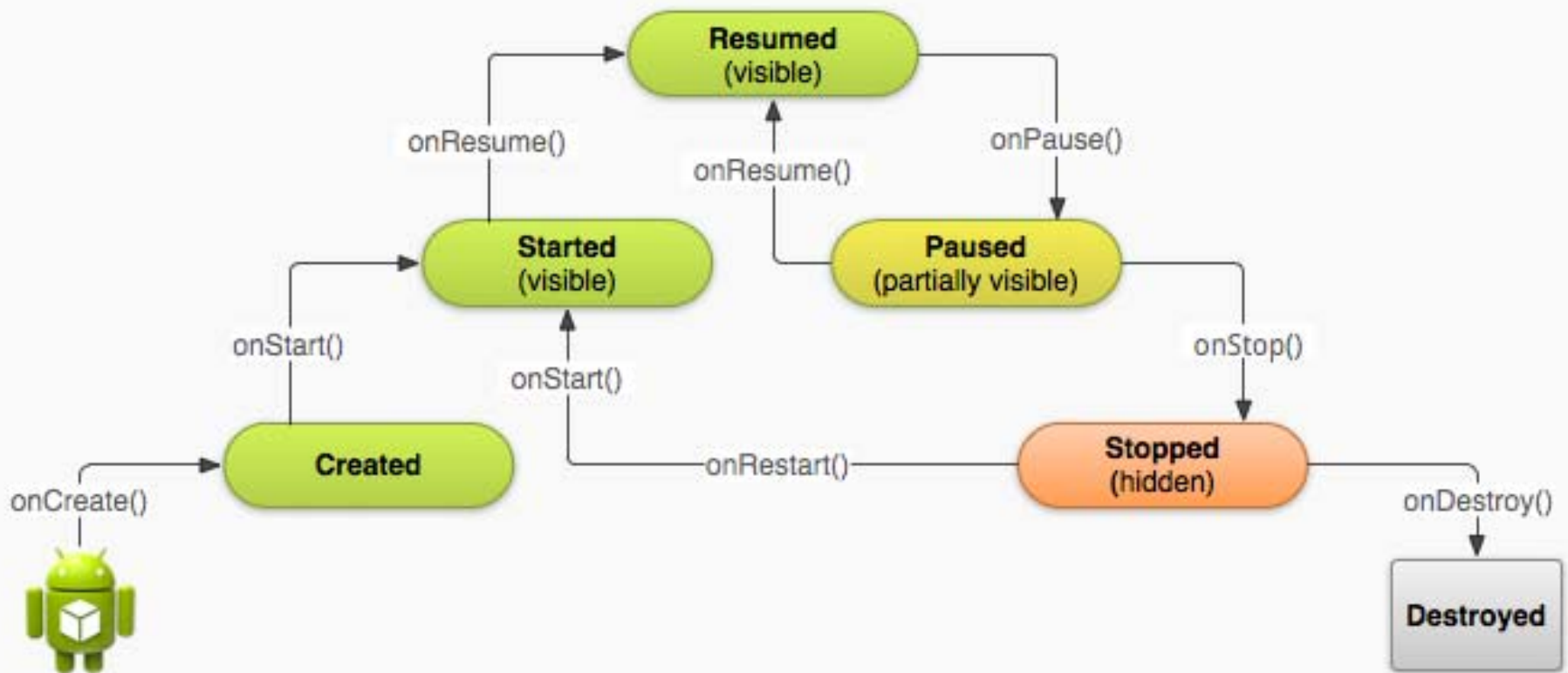
- To start new app, app is launched
- App's **onCreate()**, **onStart()** and **onResume()** methods are called
- Afterwards new app is **running**

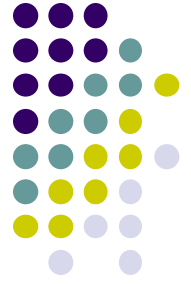


Simplified Lifecycle Diagram

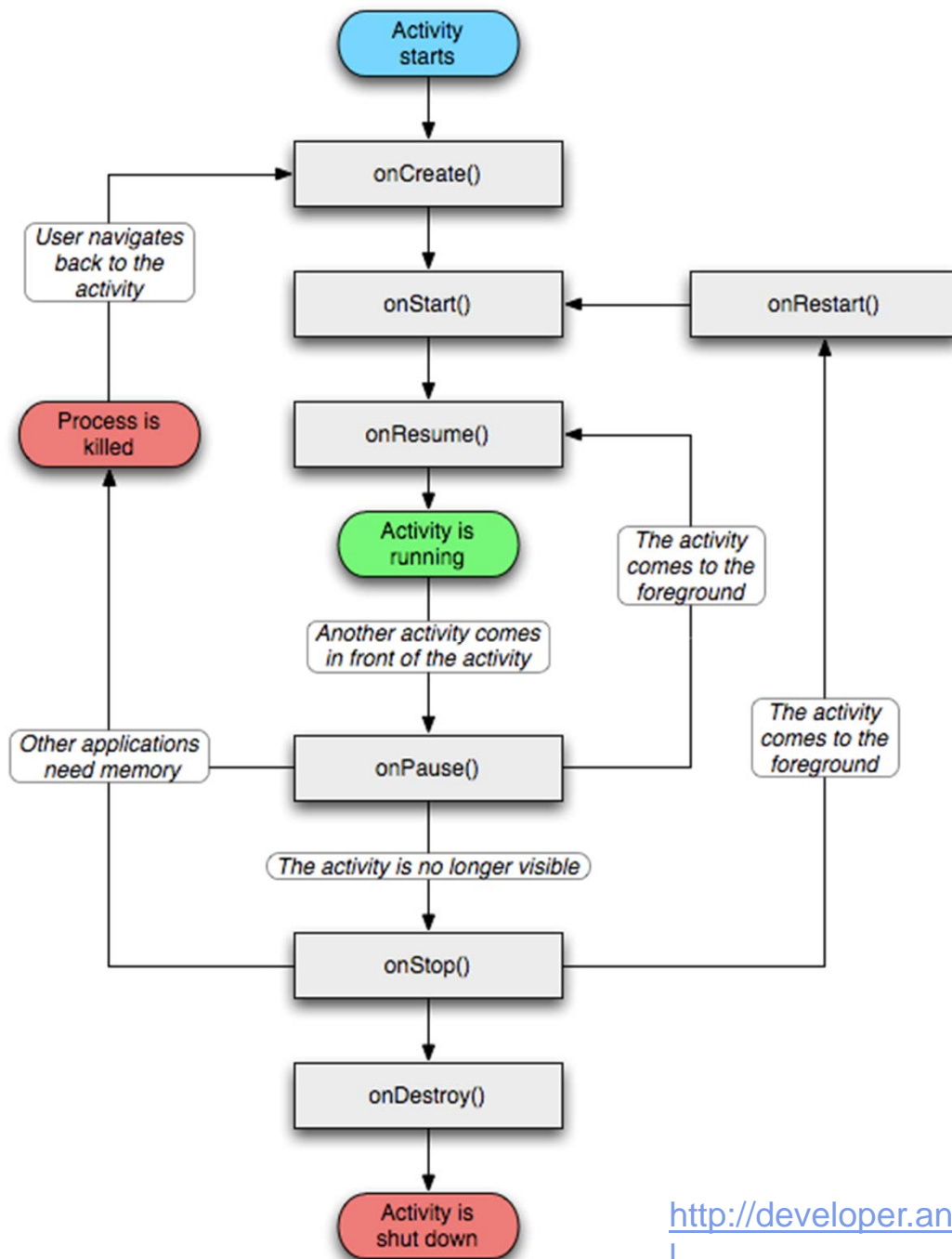


ready to interact with user

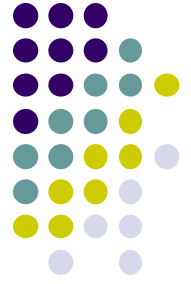




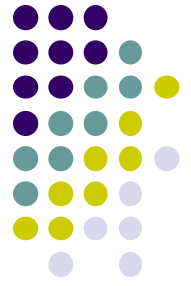
Activity Lifecycle (Another Diagram)



<http://developer.android.com/reference/android/app/Activity.htm>
!



Logging Errors in Android



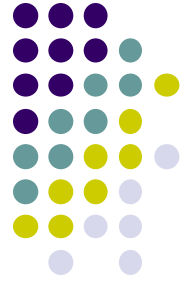
Logging Errors in Android

- Android can log and display various levels of errors
- Error logging is in **Log** class of **android.util** package
- Turn on logging of different message types by calling appropriate method

Method	Purpose
<code>Log.e()</code>	Log errors
<code>Log.w()</code>	Log warnings
<code>Log.i()</code>	Log informational messages
<code>Log.d()</code>	Log debug messages
<code>Log.v()</code>	Log verbose messages

Ref: Introduction to Android Programming, Annuzzi, Darcey & Conder

- Before calling any logging
import android.util.Log;



QuizActivity.java

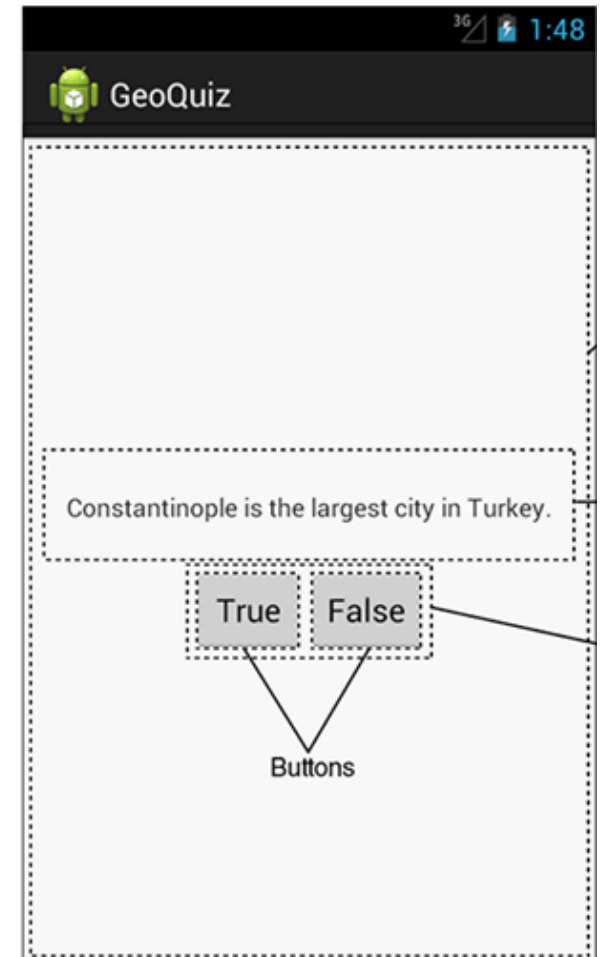
- A good way to understand Android lifecycle methods is to print debug messages when they are called
- E.g. print debug message from onCreate method below

```
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);
    }
}
```





QuizActivity.java

- Debug (d) messages have the form

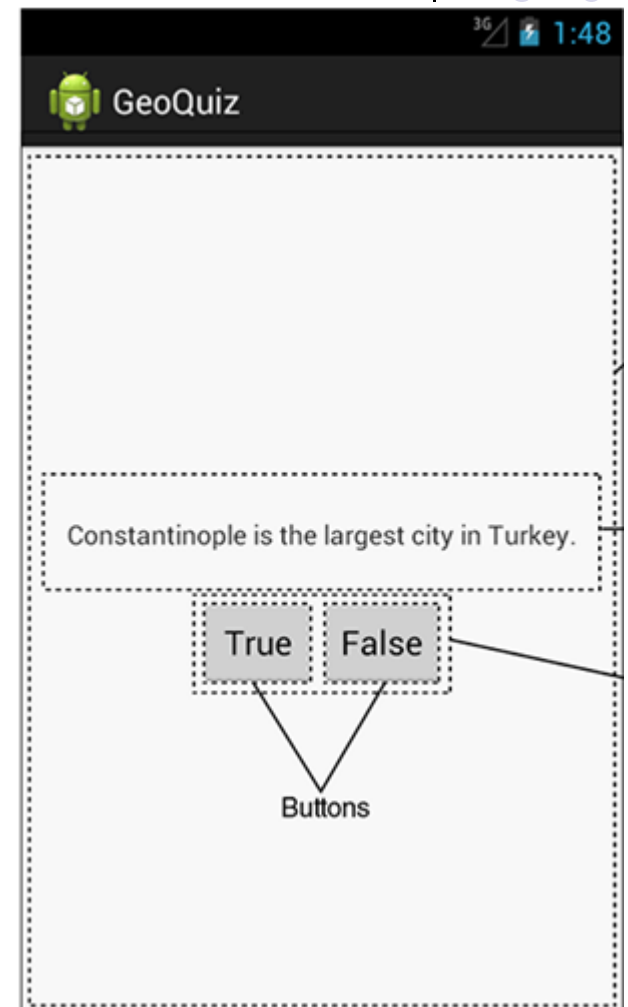
```
public static int d(String tag, String msg)
```

- **TAG** indicates source of message
- Declare string for **TAG**

```
public class QuizActivity extends Activity {  
    private static final String TAG = "QuizActivity";  
    ...  
}
```

- Can then print a message in **onCreate()**

```
Log.d(TAG, "onCreate(Bundle) called");
```

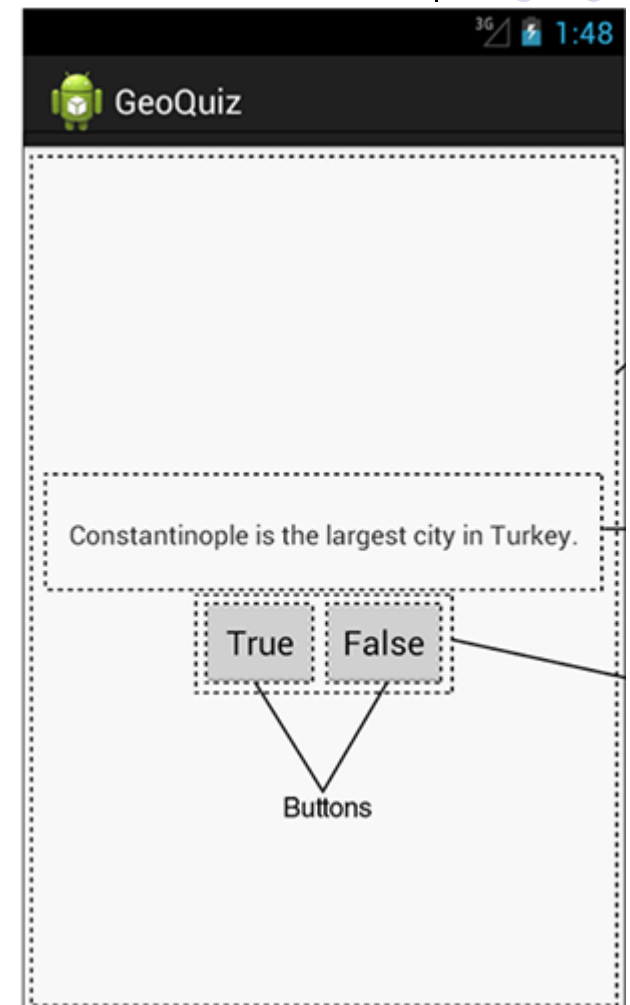




QuizActivity.java

- Putting it all together

```
public class QuizActivity extends Activity {  
  
    ...  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        Log.d(TAG, "onCreate(Bundle) called");  
        setContentView(R.layout.activity_quiz);  
  
        ...  
    }  
}
```



QuizActivity.java

- Can override more lifecycle methods
- Print debug messages from each method
- Superclass calls called in each method
- `@Override` asks compiler to ensure method exists in super class

```
} // End of onCreate(Bundle)

@Override
public void onStart() {
    super.onStart();
    Log.d(TAG, "onStart() called");
}

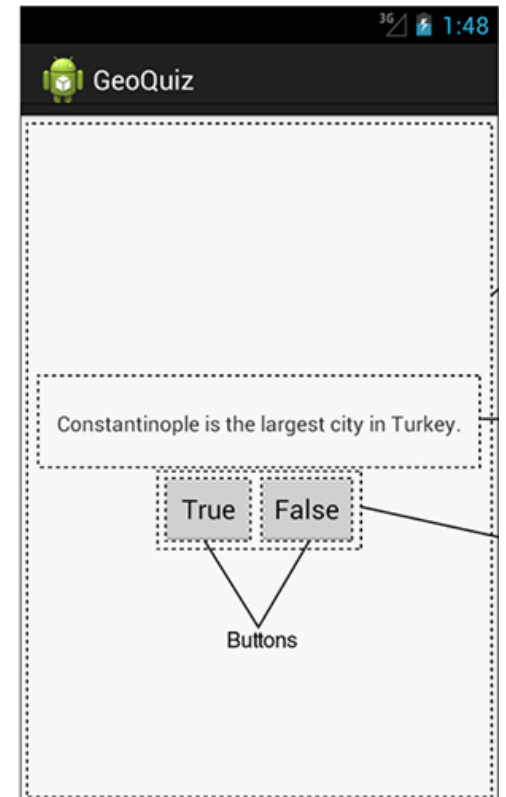
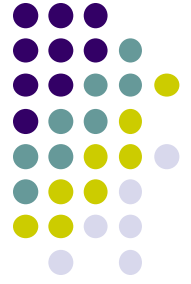
@Override
public void onPause() {
    super.onPause();
    Log.d(TAG, "onPause() called");
}

@Override
public void onResume() {
    super.onResume();
    Log.d(TAG, "onResume() called");
}

@Override
public void onStop() {
    super.onStop();
    Log.d(TAG, "onStop() called");
}

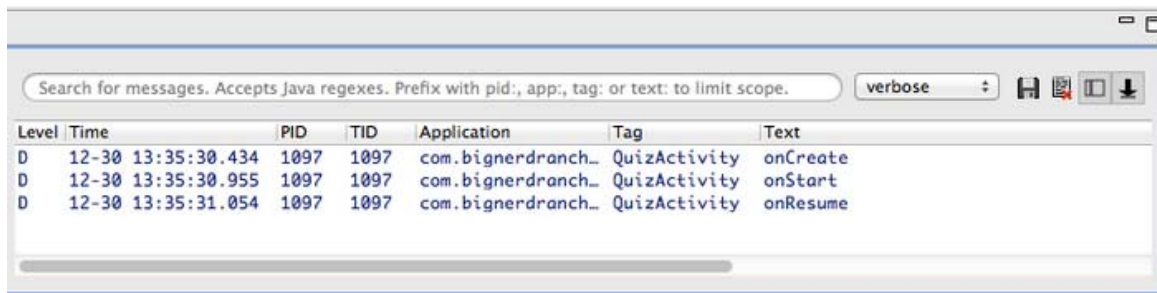
@Override
public void onDestroy() {
    super.onDestroy();
    Log.d(TAG, "onDestroy() called");
}

}
```



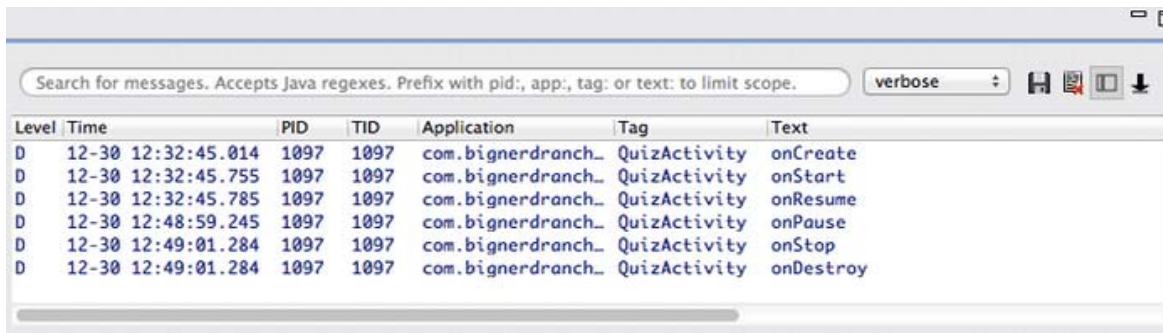
QuizActivity.java Debug Messages

- Launching GeoQuiz app **creates, starts and resumes** an activity

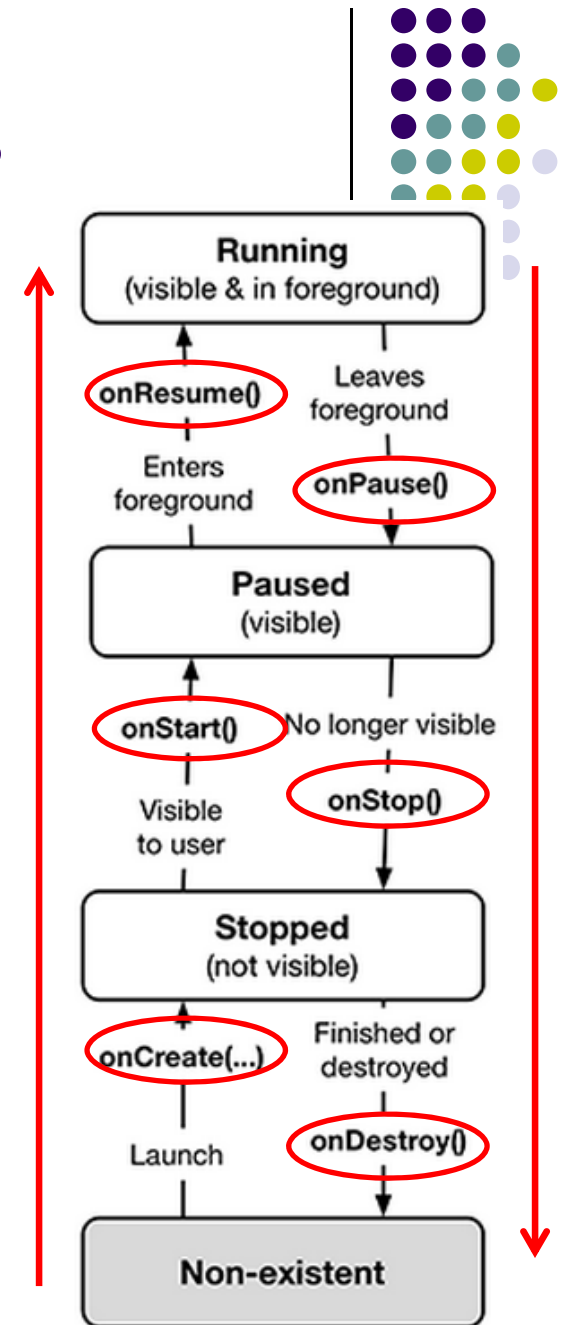


Level	Time	PID	TID	Application	Tag	Text
D	12-30 13:35:30.434	1097	1097	com.bignerdranch...	QuizActivity	onCreate
D	12-30 13:35:30.955	1097	1097	com.bignerdranch...	QuizActivity	onStart
D	12-30 13:35:31.054	1097	1097	com.bignerdranch...	QuizActivity	onResume

- Pressing **Back** button destroys the activity (calls onPause, onStop and onDestroy)



Level	Time	PID	TID	Application	Tag	Text
D	12-30 12:32:45.014	1097	1097	com.bignerdranch...	QuizActivity	onCreate
D	12-30 12:32:45.755	1097	1097	com.bignerdranch...	QuizActivity	onStart
D	12-30 12:32:45.785	1097	1097	com.bignerdranch...	QuizActivity	onResume
D	12-30 12:48:59.245	1097	1097	com.bignerdranch...	QuizActivity	onPause
D	12-30 12:49:01.284	1097	1097	com.bignerdranch...	QuizActivity	onStop
D	12-30 12:49:01.284	1097	1097	com.bignerdranch...	QuizActivity	onDestroy

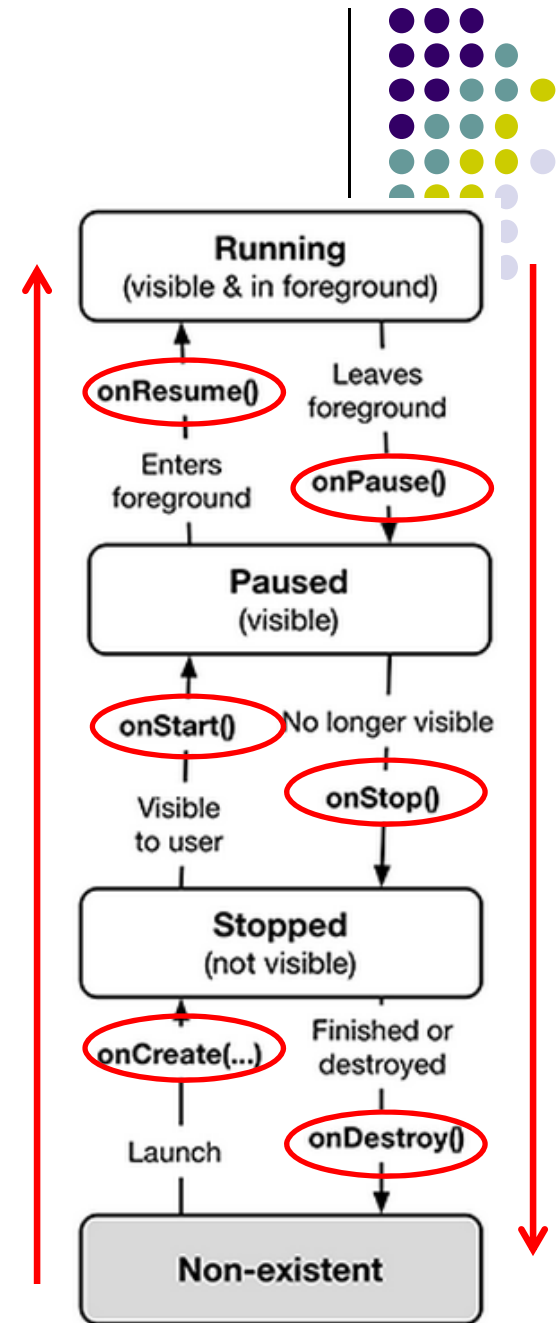


QuizActivity.java Debug Messages

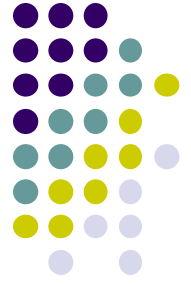
- Pressing **Home** button stops the activity

Level	Time	PID	TID	Application	Tag	Text
D	12-30 12:49:01.284	1097	1097	com.bignerdranch_	QuizActivity	onStop
D	12-30 12:49:01.284	1097	1097	com.bignerdranch_	QuizActivity	onDestroy
D	12-30 12:50:01.087	1097	1097	com.bignerdranch_	QuizActivity	onCreate
D	12-30 12:50:01.715	1097	1097	com.bignerdranch_	QuizActivity	onStart
D	12-30 12:50:01.715	1097	1097	com.bignerdranch_	QuizActivity	onResume
D	12-30 12:50:47.075	1097	1097	com.bignerdranch_	QuizActivity	onPause
D	12-30 12:50:49.945	1097	1097	com.bignerdranch_	QuizActivity	onStop

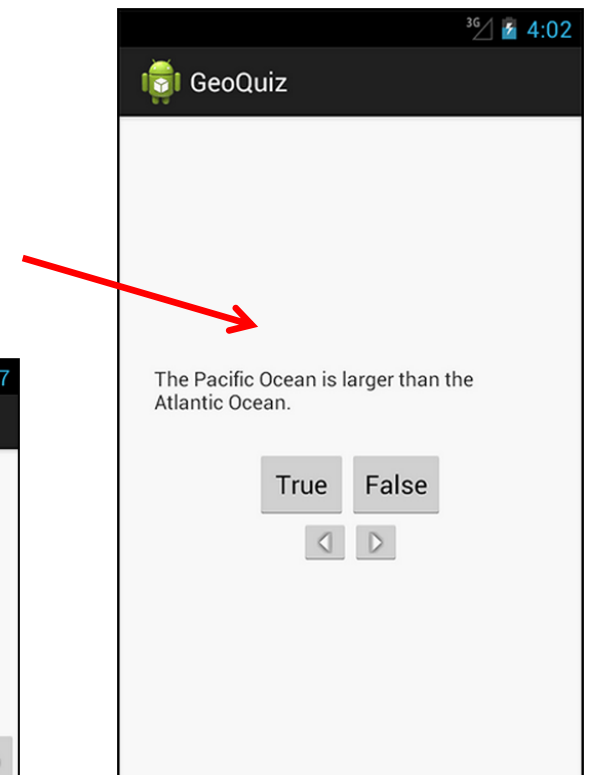
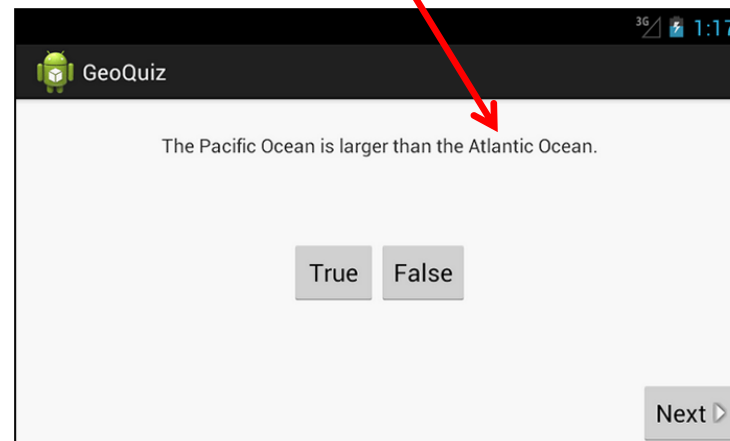
- Rotating device (e.g. portrait to landscape) kills current activity and creates new activity in landscape mode



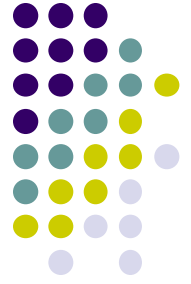
Rotating Device & Device Configuration



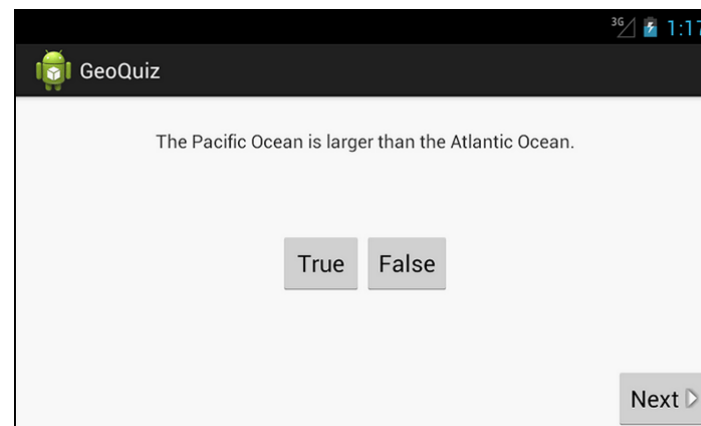
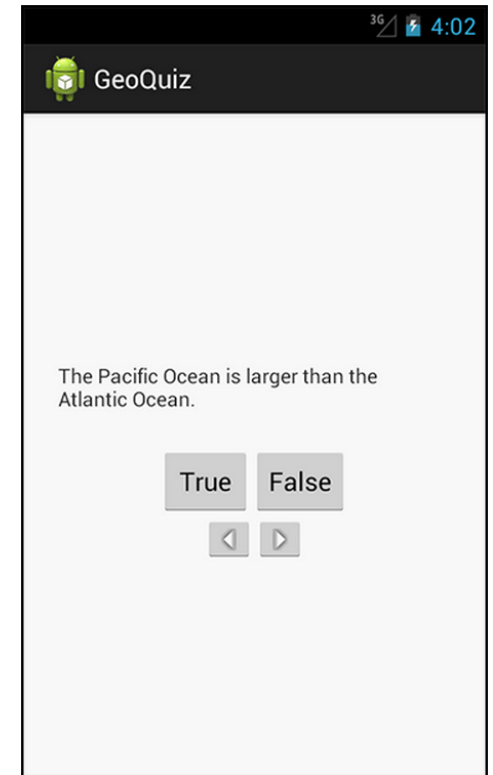
- Rotation changes **device configuration**
- **Device configuration:** screen orientation/density/size, keyboard type, dock mode, language, etc.
- Apps can specify different resources to use for different device configurations
- E.g. use different app layouts for portrait vs landscape screen orientation



Rotating Device & Device Configuration

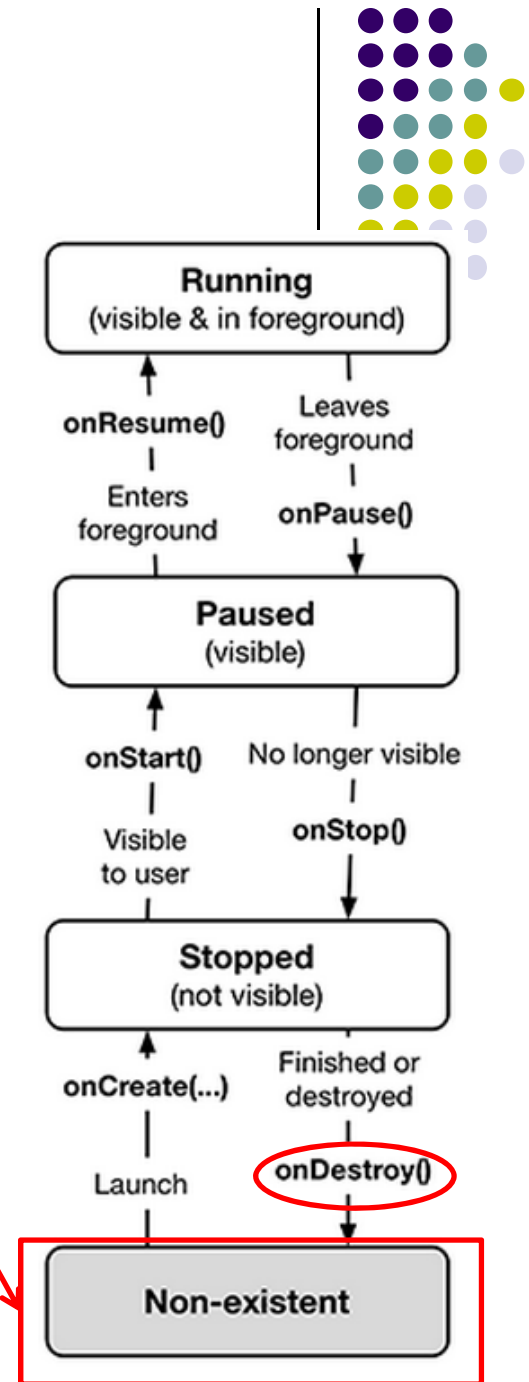


- How to use different app layouts for portrait vs landscape screen orientation?
- When device in landscape, uses resources in **res/layout-land/**
- Copy XML layout file (activity_quiz.xml) from **res/layout** to **res/layout-land/** and tailor it
- When configuration changes, current activity destroyed, **onCreate (setContentView (R.layout.activity_quiz))** called again



Dead or Destroyed Activity

- Dead, activity terminated (or never started)
- onDestroy() called to destroy a stopped app
- Two other states, Created and Started, but they are transitory onCreate -> onStart -> onResume

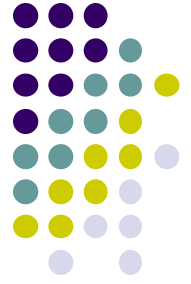




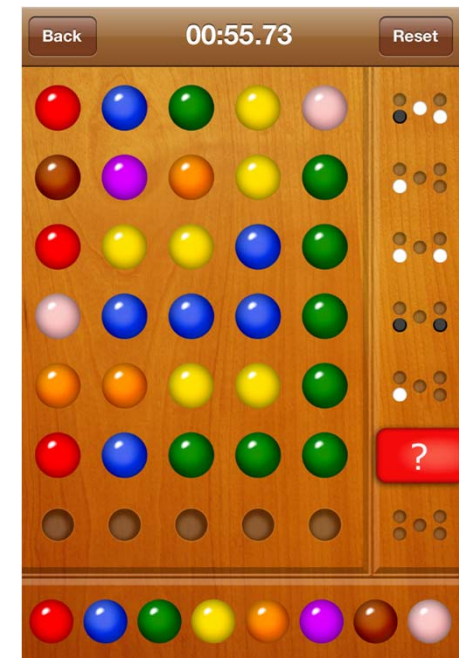
Activity Destruction

- App may be destroyed
 - On its own by calling finish
 - If user presses **back button** to navigate away from app
 - Normal lifecycle methods handle this
onPause() -> onStop() -> onDestroy
- If the system must destroy the activity (to recover resources or on an orientation change) must be able to recreate Activity
- If Activity destroyed with potential to be recreate later, system calls onSaveInstanceState (Bundle outState) method

onSaveInstanceState onRestoreInstanceState()

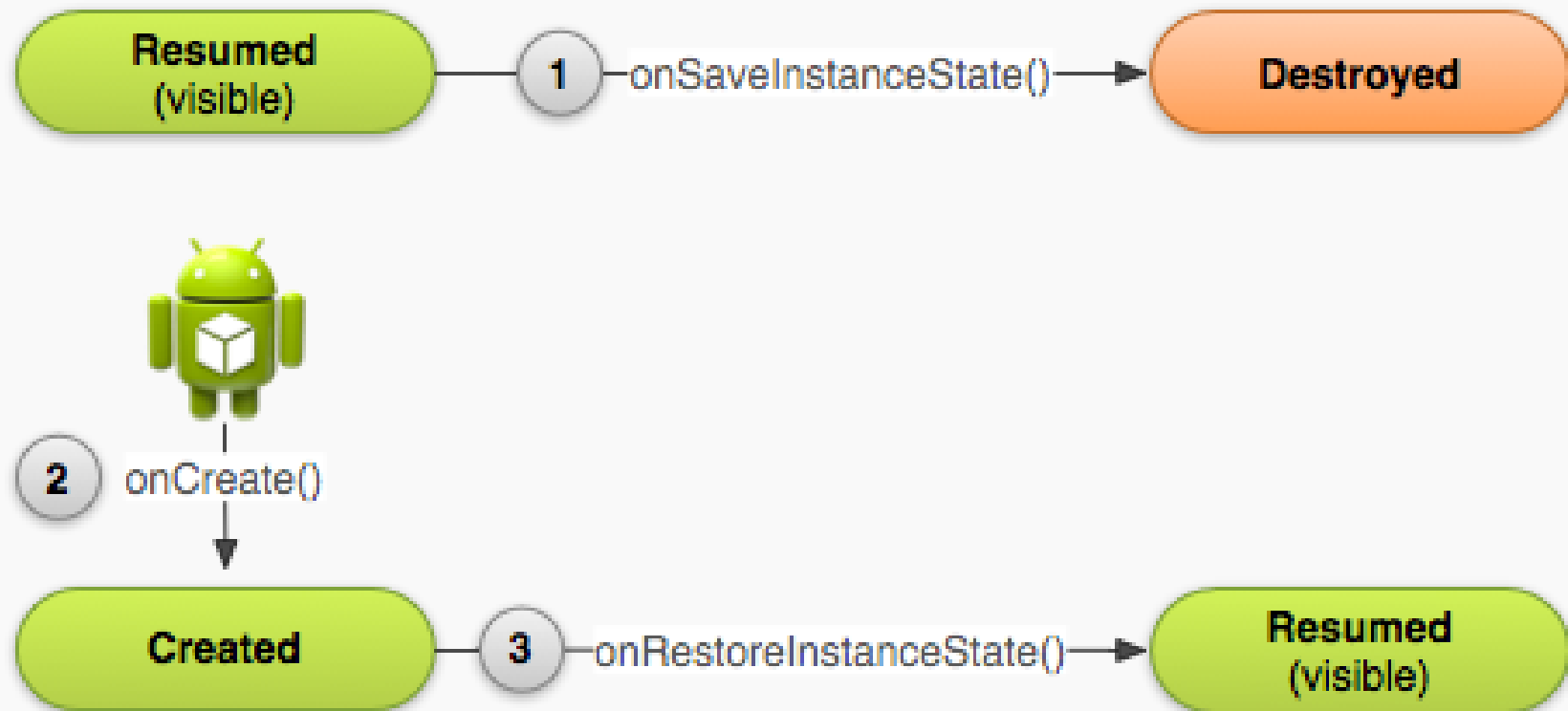


- Systems write info about views to Bundle
- other (app-specific) information must be added by programmer
 - E.g. board state in a board game such as mastermind
- When Activity recreated Bundle sent to onCreate and onRestoreInstanceState()
- use either method to restore state data / instance variables



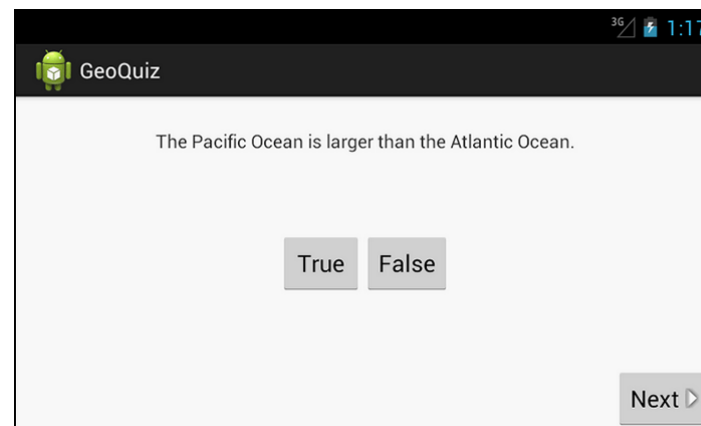
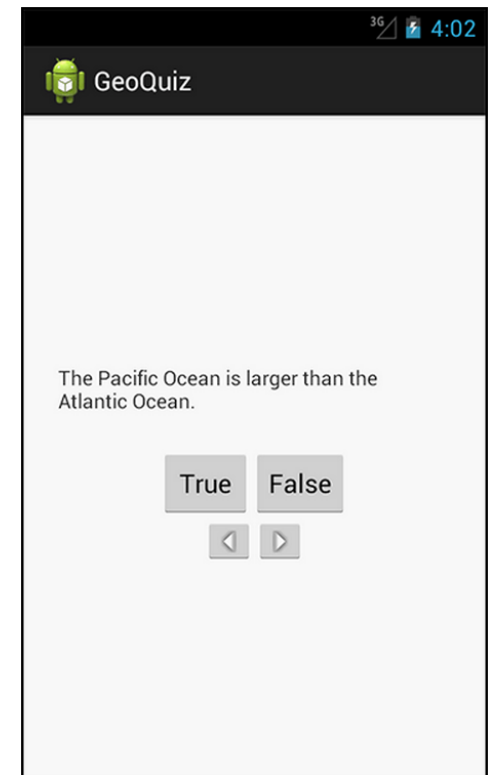


Saving State on Activity Destruction



Saving Data Across Device Rotation

- Since rotation causes activity to be destroyed and new one created, values of variables lost or reset
- To stop lost or reset values, save them using **onSaveInstanceState** before activity is destroyed
- System calls **onSaveInstanceState** before **onPause()**, **onStop()** and **onDestroy()**



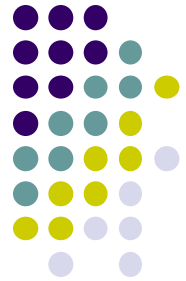
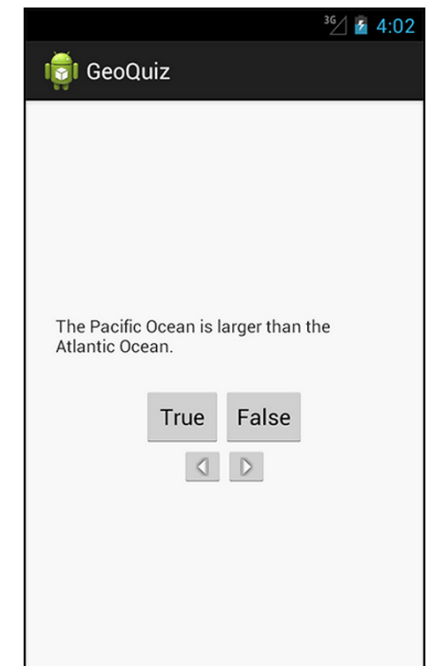
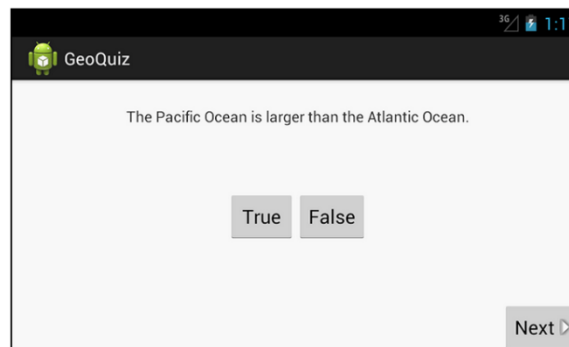
Saving Data Across Device Rotation

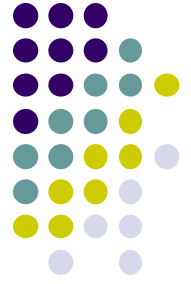
- For example, if we want to save the value of a variable **mCurrentIndex** during rotation
- First, create a constant as a key for storing data in the bundle

```
private static final String KEY_INDEX = "index";
```

- Then override **onSaveInstanceState** method

```
@Override  
public void onSaveInstanceState(Bundle savedInstanceState) {  
    super.onSaveInstanceState(savedInstanceState);  
    Log.i(TAG, "onSaveInstanceState");  
    savedInstanceState.putInt(KEY_INDEX, mCurrentIndex);  
}
```





References

- 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