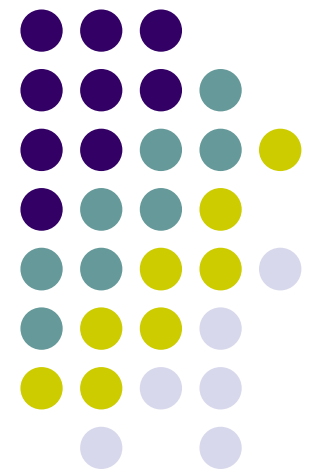
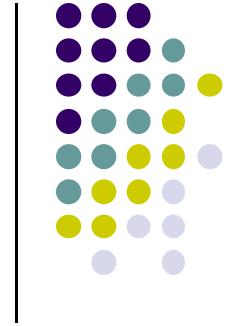


CS 403X Mobile and Ubiquitous Computing

Lecture 3: Android UI, WebView, Android Activity Lifecycle

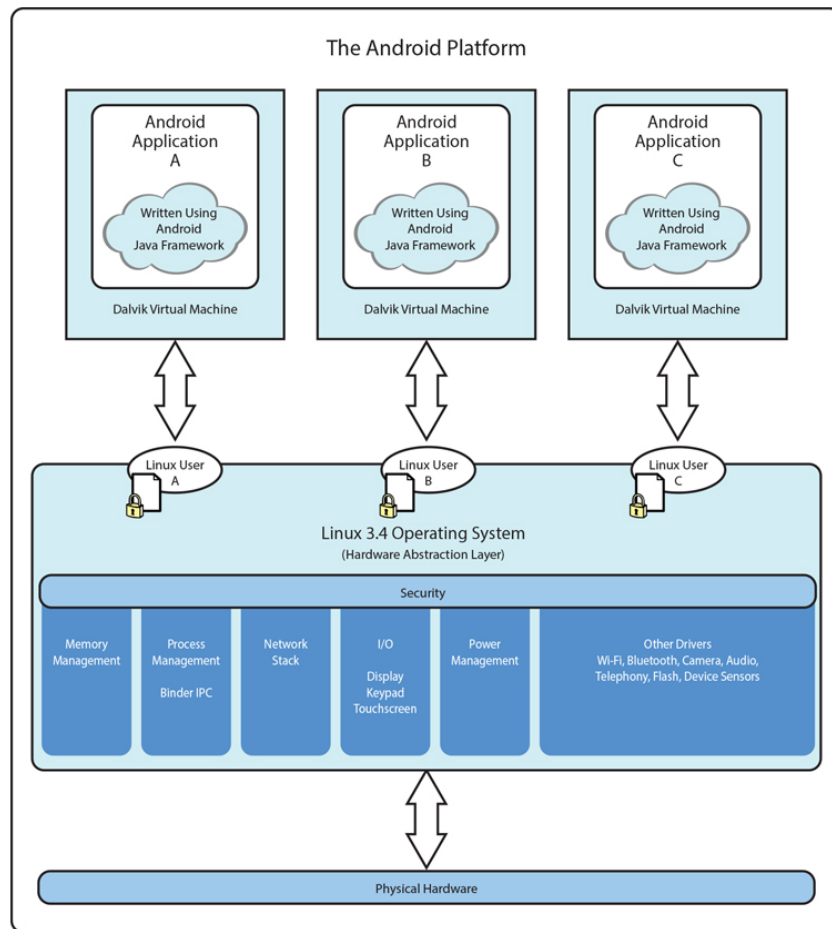
Emmanuel Agu





Android Software Framework

Android Software Framework



- Multi-user Linux system
- Each Android app is a different Linux user
- Each Android app runs in its own VM, minimizes complete system crashes
- Android system assigns each app a unique Linux user ID
 - ID is unknown to the application
- App's files are private to it
- **Android** starts app's process when its components need to be executed, shuts down the process when no longer needed

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

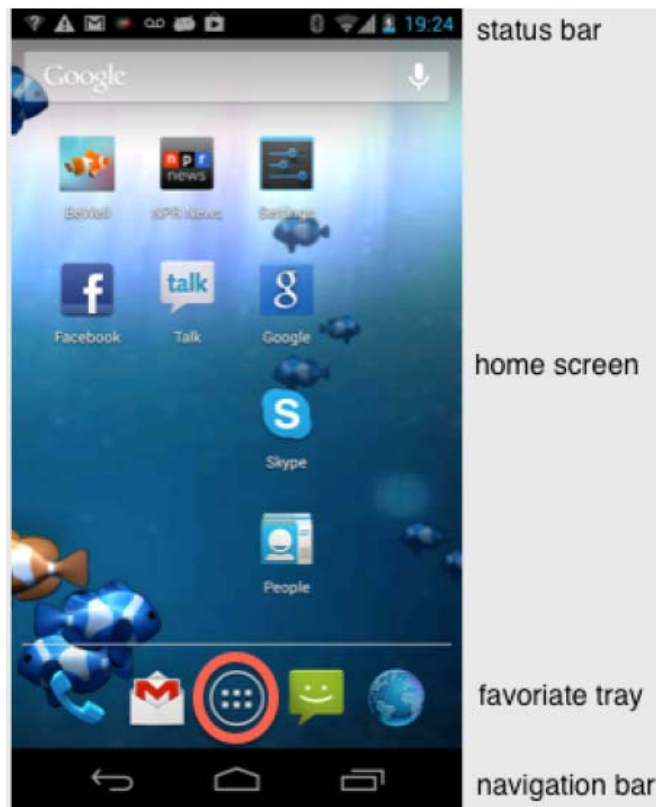


Android UI Tour

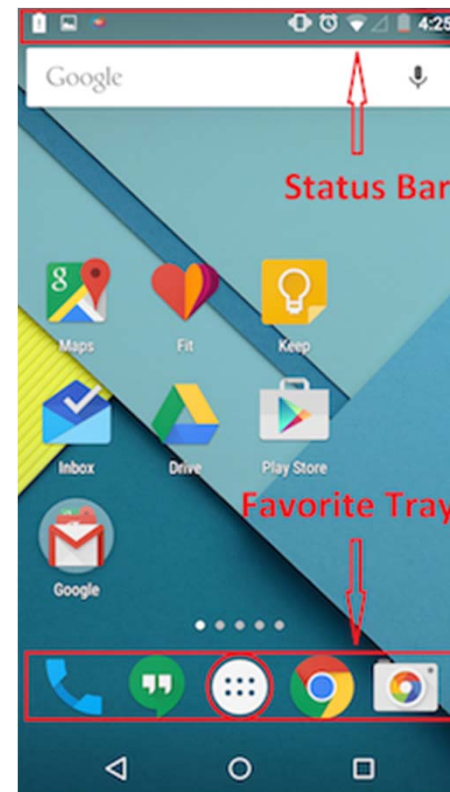


Home Screen

- First screen after unlocking phone or hitting **home** button
- Includes **favorites** tray (e.g phone, mail, messaging, web, etc)



Android 4.0

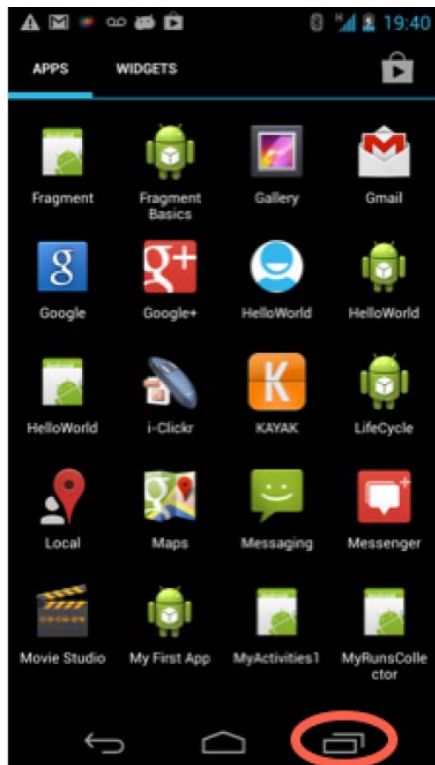


Android 5.0

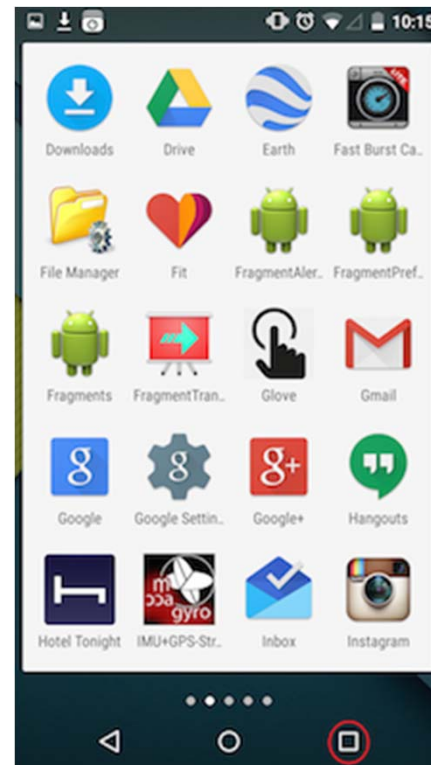


All Apps Screen

- Accessed by touching **all apps button** in favorites tray
- Users can swipe through multiple app and widget screens
- Can be customized by dragging and dropping items



Android 4.0

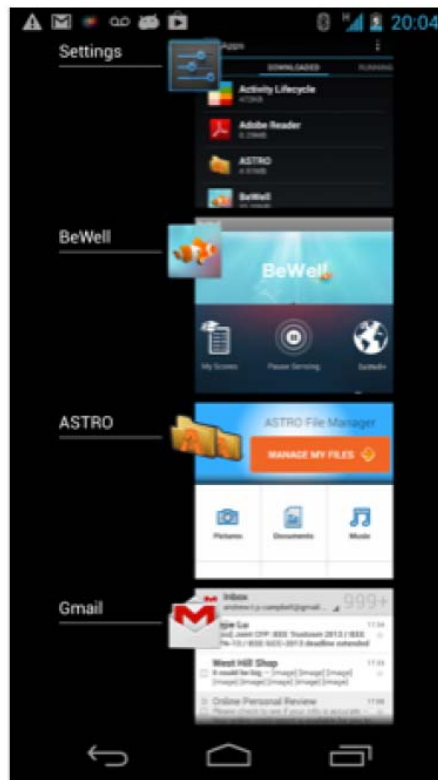


Android 5.0

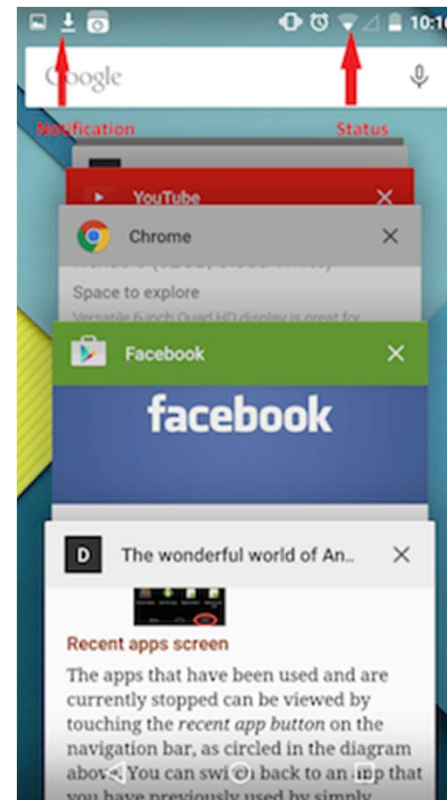


Recent Apps Screen

- Accessed by touching the **recent apps button**
- Shows recently used and currently stopped apps
- Can switch to a recently used app by touching it in list



Android 4.0

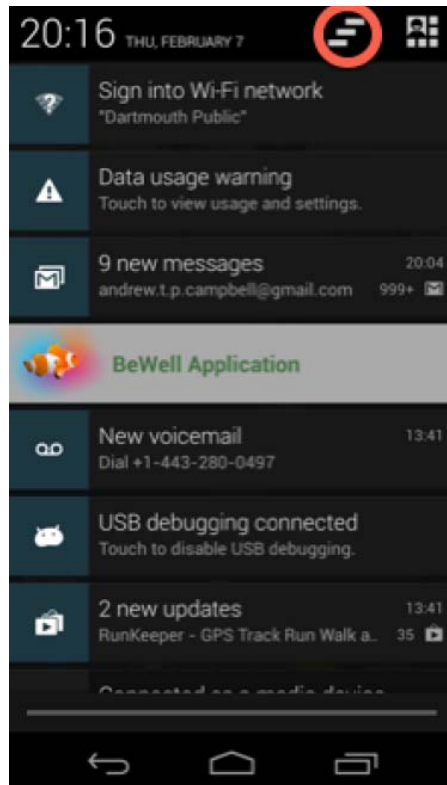


Android 5.0

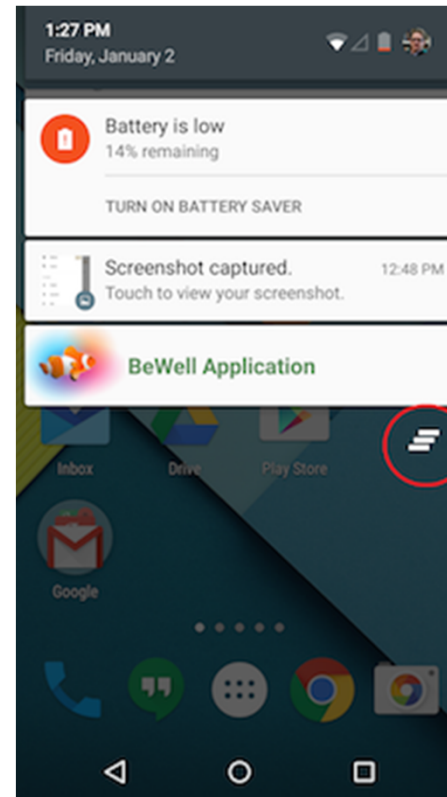
Status Bar and Notification Screen



- Displays notifications (on left) and status (on right)
- **Status:** time, battery, cell signal strength, bluetooth enabled, etc
- **Notification:** wifi, mail, bewell, voicemail, usb active, music, etc



Android 4.0

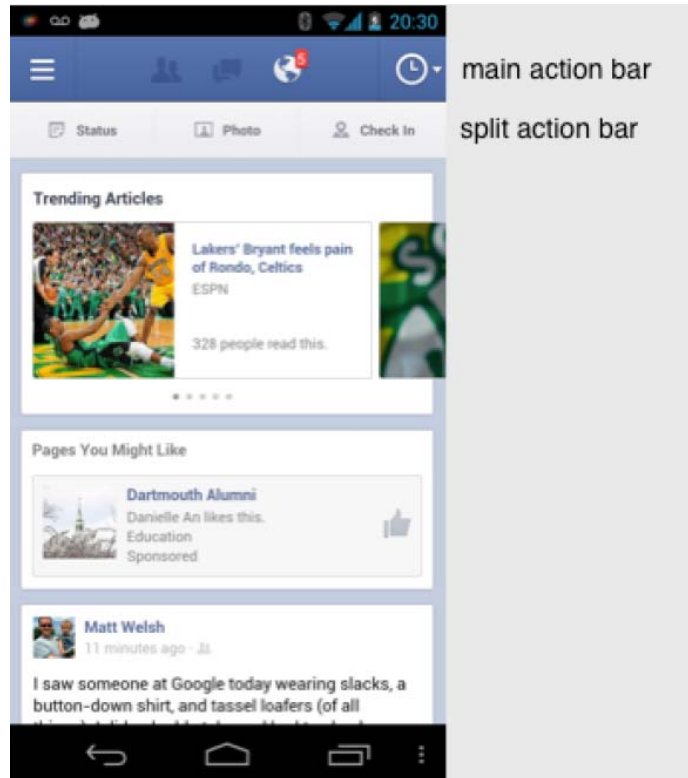


Android 5.0



Facebook UI

- Uses many standard Android UI components
- Shows main action bar and split action bar
- **Action bar:** configurable, handles user action and app navigation



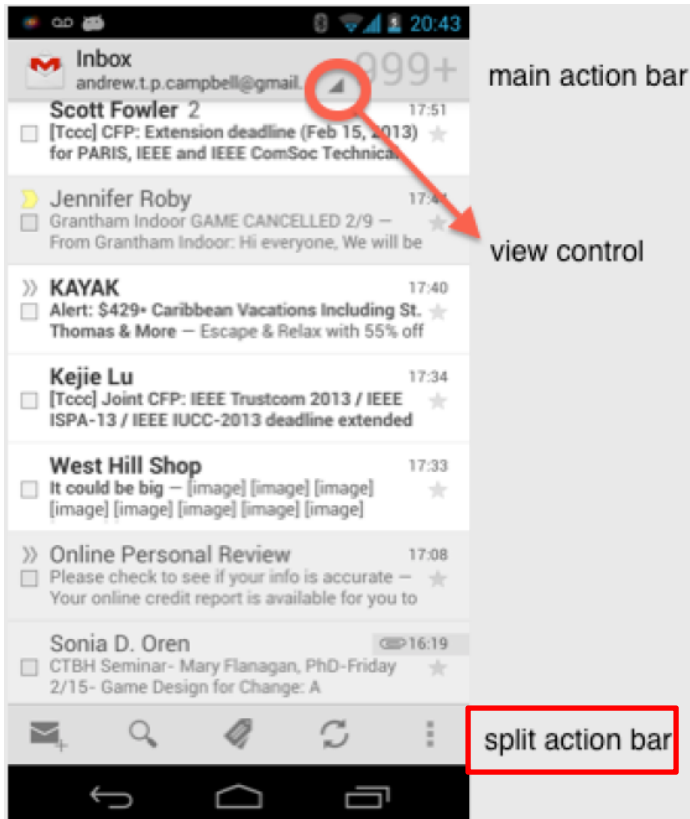
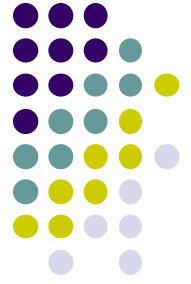
Android 4.0



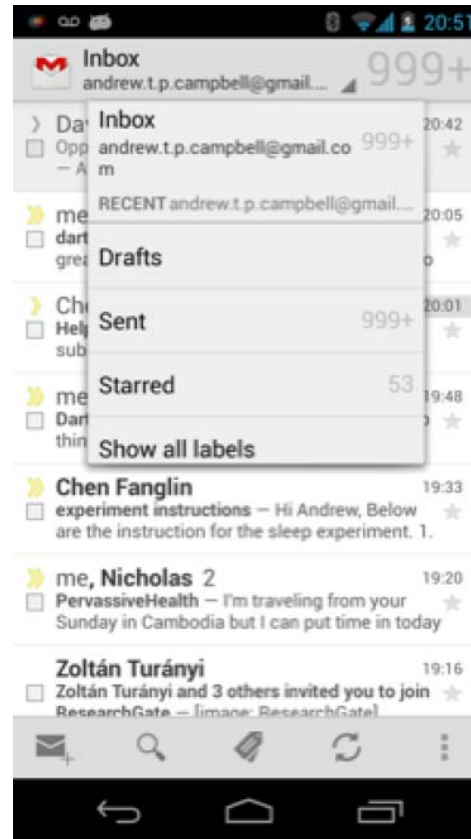
Android 5.0

Gmail

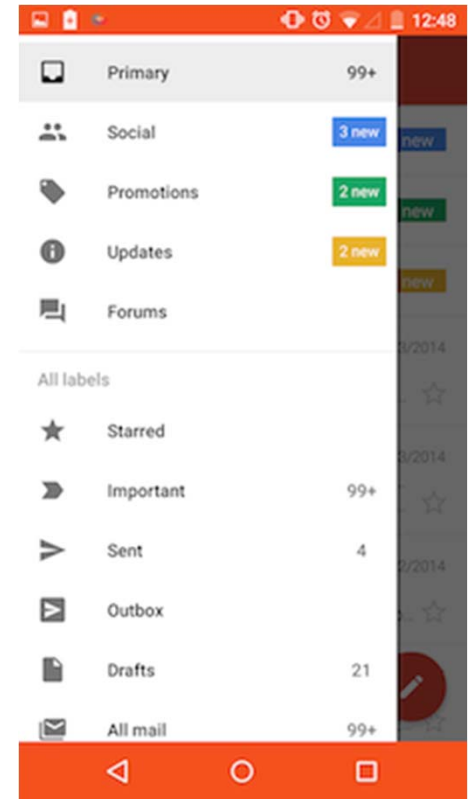
- Split action bar at bottom of screen
- **View control:** allows users switch between different views (inbox, recent, drafts, sent)



Android 4.0



Android 4.0



Android 5.0

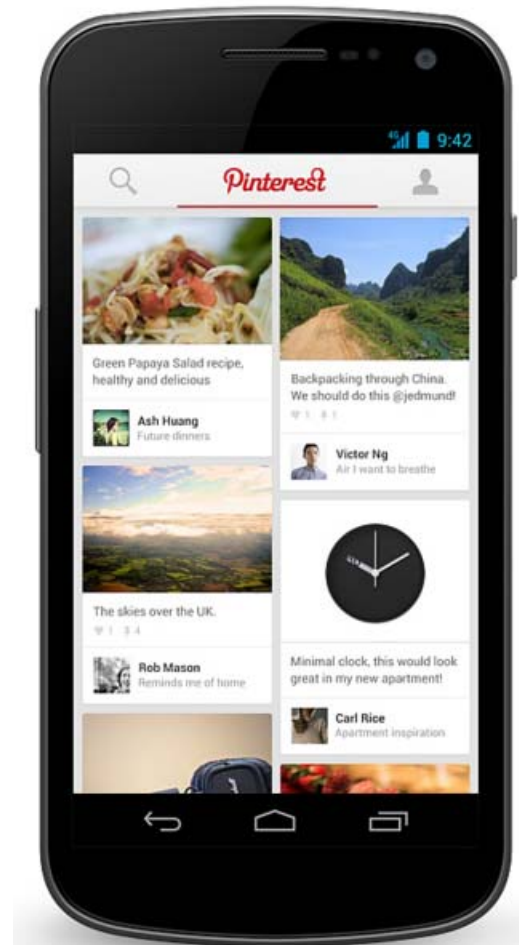


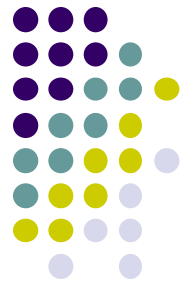
Resources

Recall: Example: Files in an Android Project



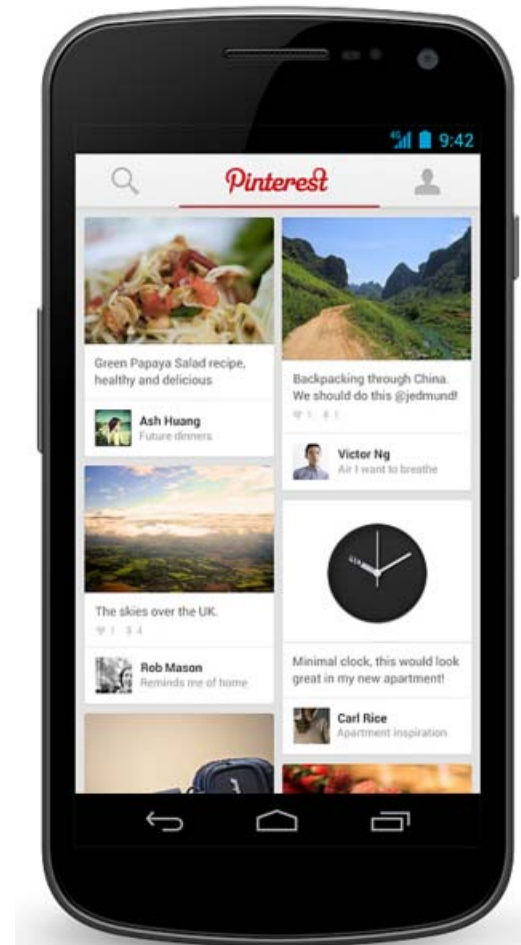
- **res/layout:** The width, height, layout of screen cells are specified in XML file here
- **res/drawable-xyz/:** The images stored in jpg or other format here
- **java/:** App's behavior when user clicks on screen (e.g. button) specified in java file here
- **AndroidManifest.XML:** Contains app name (Pinterest), list of app screens, etc





Resource Files in an Android Project

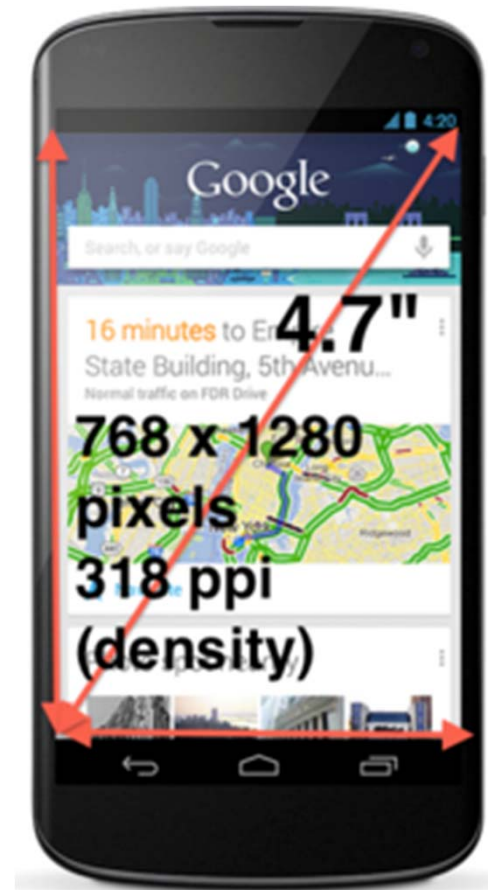
- Resources (stored in **/res** folder) are static bits of information outside java code (e.g. layout, images, etc). E.g.
 - **res/drawable-xyz/**
 - **res/layout:**
- Can have multiple resource definitions, used under different conditions. E.g internationalization (text in different languages)
- In Android Studio, the **res/** folder is **app/src/main/**

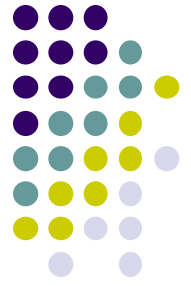


Phone Dimensions Used in Android UI



- Physical dimensions measured diagonally
 - E.g. Nexus 4 is 4.7 inches diagonally
- Resolution in pixels
 - E.g. Nexus 4 resolution 768 x 1280 pixels
- Pixels per inch (PPI) =
 - $\text{Sqrt}[(768 \times 768) + (1280 \times 1280)] = 318$
- Dots per inch (DPI) is number of pixels in a physical area
 - Low density (ldpi) = 120 dpi
 - Medium density (mdpi) = 160 dpi
 - High density (hdpi) = 240 dpi
 - Extra High Density (xhdpi) = 320 dpi





Adding Pictures

- Android supports images in PNG, JPEG and GIF formats
- GIF officially discouraged, PNG preferred format
- Default directory for images (drawables) is **res/drawable-xyz**
- Images in **res/drawable-xyz** can be referenced by XML and java files
 - **res/drawable-ldpi**: low dpi images (~ 120 dpi of dots per inch)
 - **res/drawable-mdpi**: medium dpi images (~ 160 dpi)
 - **res/drawable-hdpi**: high dpi images (~ 240 dpi)
 - **res/drawable-xhdpi**: extra high dpi images (~ 320 dpi)
 - **res/drawable-xxhdpi**: extra extra high dpi images (~ 480 dpi)
 - **res/drawable-xxxhdpi**: high dpi images (~ 640 dpi)
- Images in these directories are **same size, different resolutions**

Adding Pictures



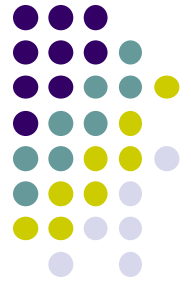
- Just the generic picture name is used
 - No format e.g. .png,
 - No specification of what resolution to use
 - E.g. to reference an image **ic_launcher.png**

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

- Android chooses which directory (e.g. -mdpi) based on actual device
- Android studio tools for generating icons
 - **Icon wizard or Android asset studio:** generates icons in various densities from starter image
 - Cannot edit images (e.g. dimensions) with these tools

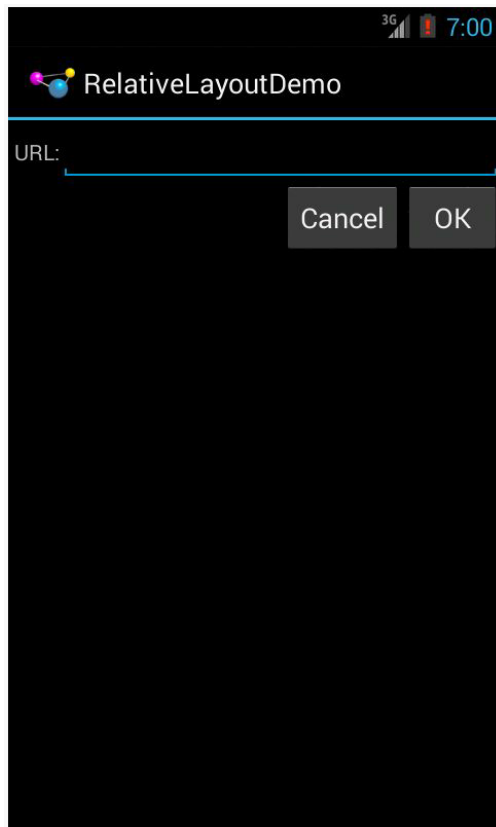
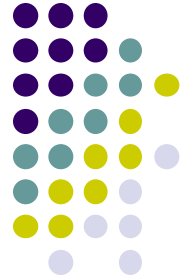
Styles

- Styles specify rules for look of Android screen
- Similar to Cascaded Style Sheets (CSS) in HTML
- E.g CSS enables setting look of certain types of tags.
 - E.g. font and size of all <h1> and <h2> elements
- Android widgets have properties
 - E.g. Foreground color = red
- **Styles in Android:** collection of values for properties
- Styles can be specified one by one or themes (e.g. Theme, Theme.holo and Theme.material) can be used

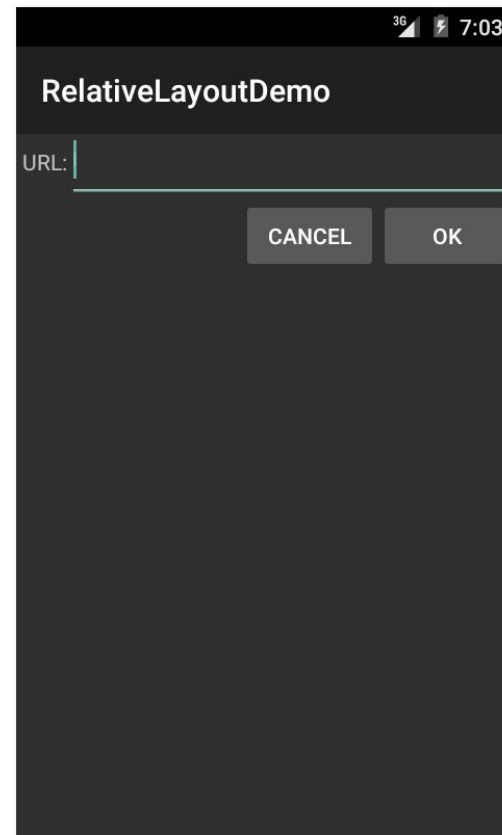


Default Themes

- Android chooses a default theme if you specify none
- Also many stock themes to choose from

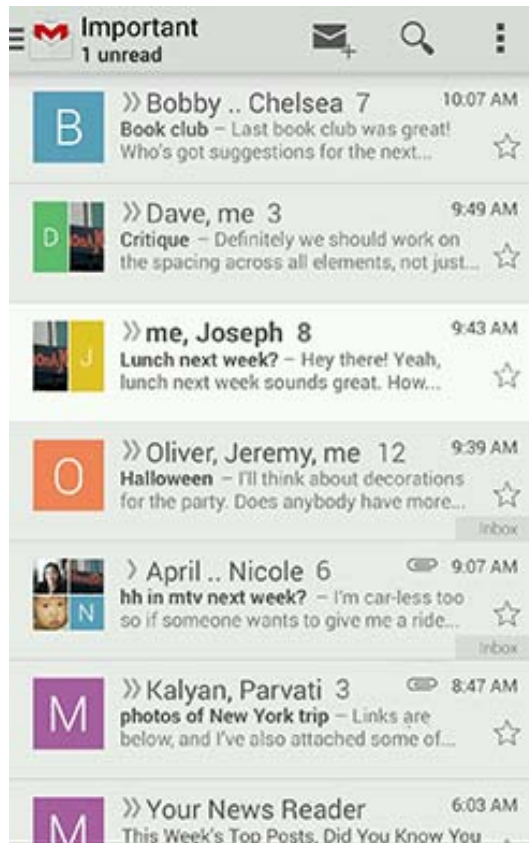


Theme.Holo: default theme in Android 3.0



Theme.Material: default theme in Android 5.0

Examples of Themes in Use



GMAIL in Holo Light



Settings in Holo Dark



Android UI in Java



Android UI: 2 Options

- **Option 1:** Design Android UI in XML file
 - Call setContentView() in java file

- **Option 2:** Design Android UI in Java file

Recall: GeoQuiz: activity_quiz.xml (Android UI in XML)



```
<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" />

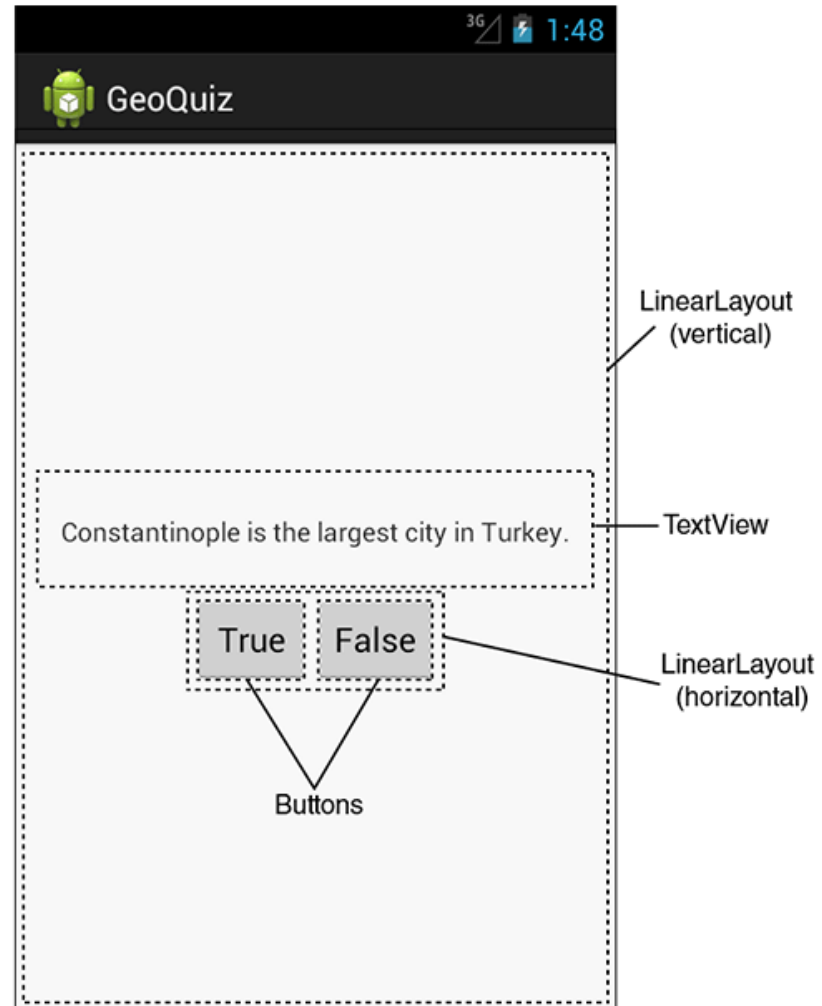
  <LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal" >

    <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>
```



Recall: QuizActivity.java: (Android UI in XML)

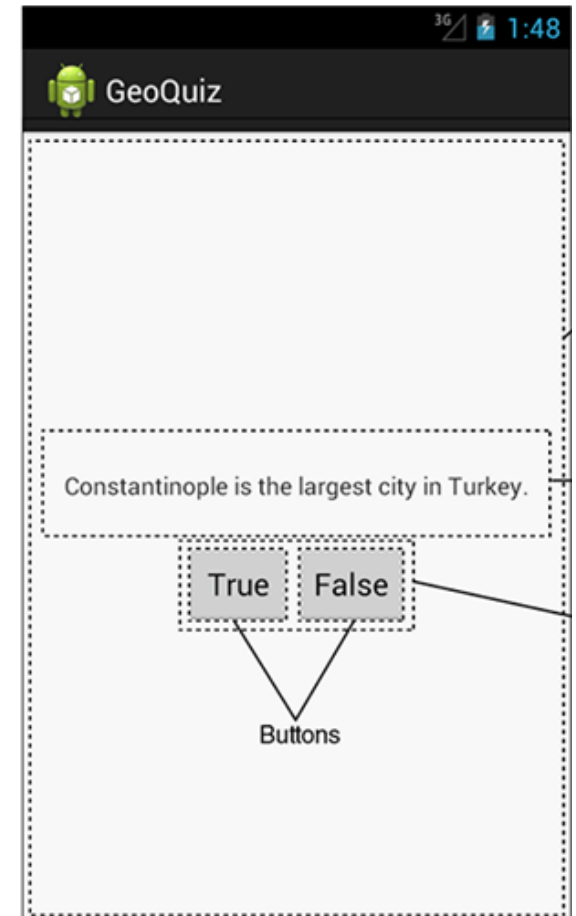


- Initial QuizActivity.java code

```
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



Example: Creating Android UI in Java



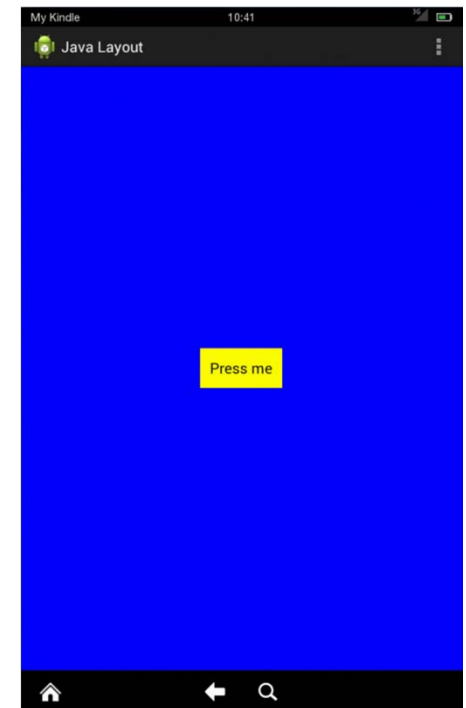
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Button myButton = new Button(this);
    myButton.setText("Press me");
    myButton.setBackgroundColor(Color.YELLOW);

    RelativeLayout myLayout = new RelativeLayout(this);
    myLayout.setBackgroundColor(Color.BLUE);

    RelativeLayout.LayoutParams buttonParams =
        new RelativeLayout.LayoutParams(
            RelativeLayout.LayoutParams.WRAP_CONTENT,
            RelativeLayout.LayoutParams.WRAP_CONTENT);

    buttonParams.addRule(RelativeLayout.CENTER_HORIZONTAL);
    buttonParams.addRule(RelativeLayout.CENTER_VERTICAL);

    myLayout.addView(myButton, buttonParams);
    setContentView(myLayout);
}
```





Pros and Cons of UI Design in Java vs XML

- XML advantages:
 - Graphical Layout tool can be used. It generates XML
 - After app is complete, UI easily changed by modifying XML without recompiling code

- Java advantages:
 - Good for designing dynamic Uis. XML limited to static screens



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



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 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
- 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 {
```

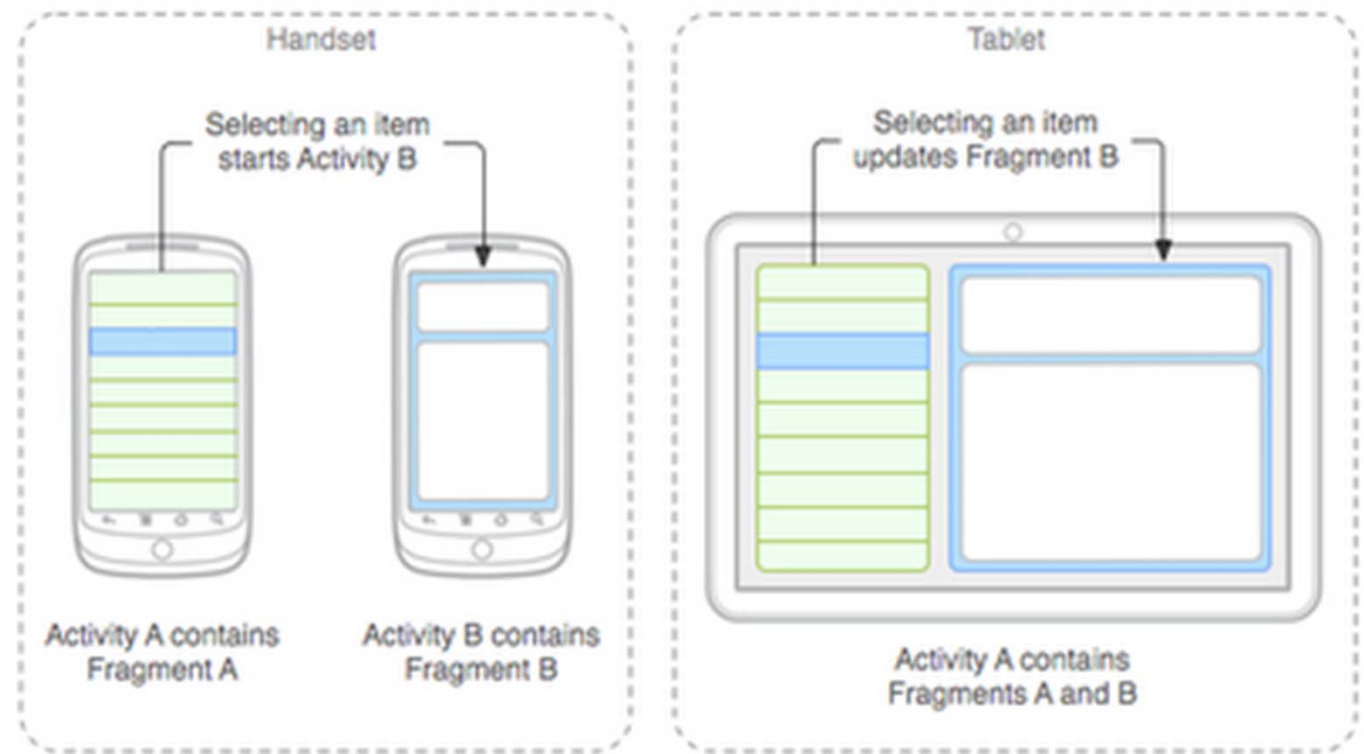
Activity



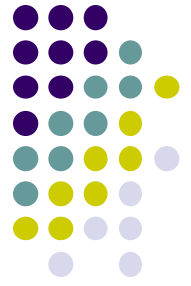


Fragments

- Fragments enables single 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



- Similar to Linux/Unix CRON job (background, long-running)
- 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
- 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

```
public class HelloIntentService extends IntentService {
```


Android Platform Services



- Android Services can either be:
 - **Local:** Android Platform
 - **Remote:** Google

- 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

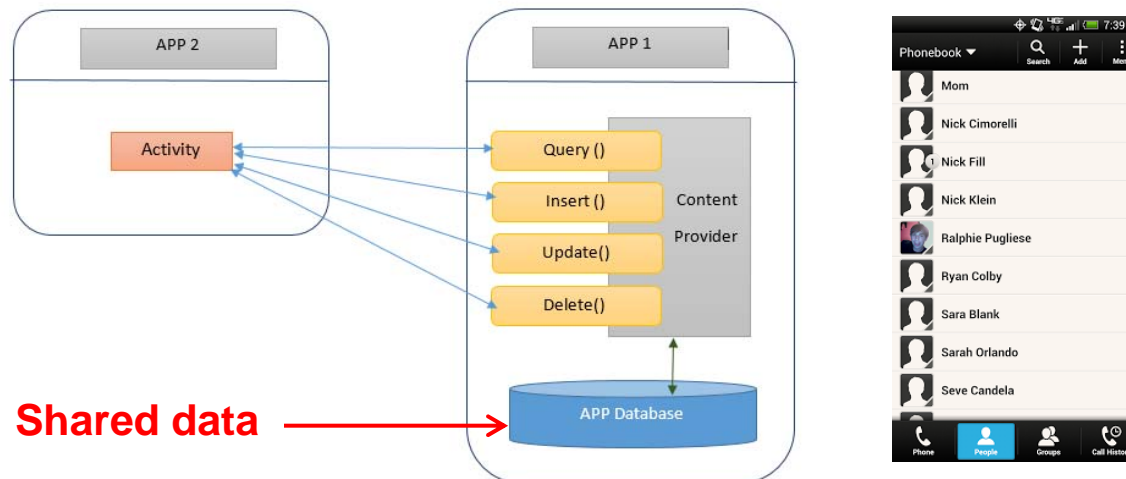


- Maps
- Location-based services
- Game Services
- Authorization APIs
- Google Plus
- Play Services
- In-app Billing
- Google Cloud Messaging
- Google Analytics
- Google AdMob ads



Content Providers

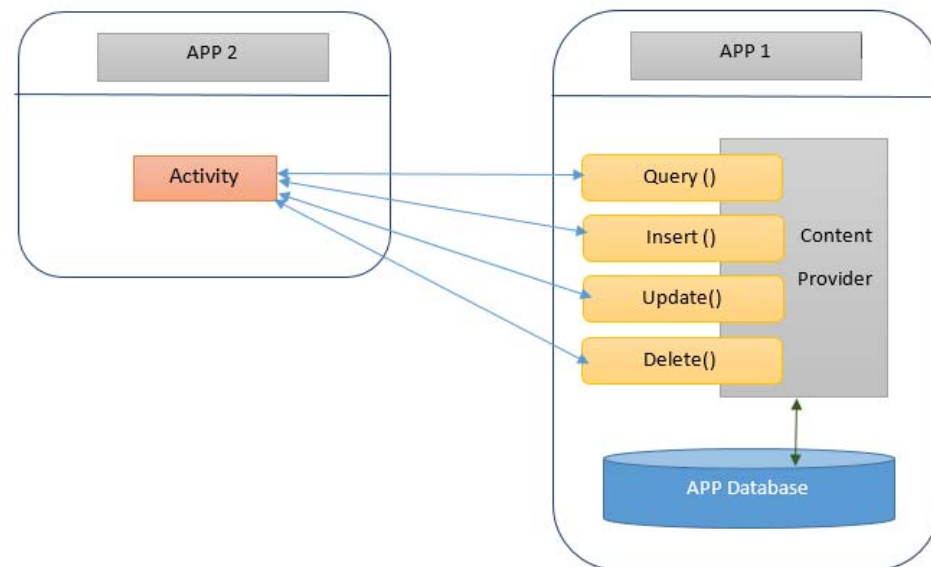
- Android apps can share data
- Content Provider:
 - Abstracts shareable data, makes it accessible through methods
 - Applications can access that shared data by calling methods for the relevant **content provider**
- **Example:** We can write an app that:
 - Retrieve's contacts list from contacts content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





Content Providers

- Apps can also **ADD** to data through content provider.
 - E.g. Add contact
- Our app can also share its data



- Content Providers are sub-class of **ContentProvider** class. E.g

```
public class ExampleProvider extends ContentProvider {
```

Broadcast Receivers



- The system, or applications, periodically *broadcasts* events. E.g.
 - 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
- Broadcast Receivers are sub-class of **BroadcastReceiver** class. E.g

```
public class MyBroadcastReceiver extends BroadcastReceiver {
```



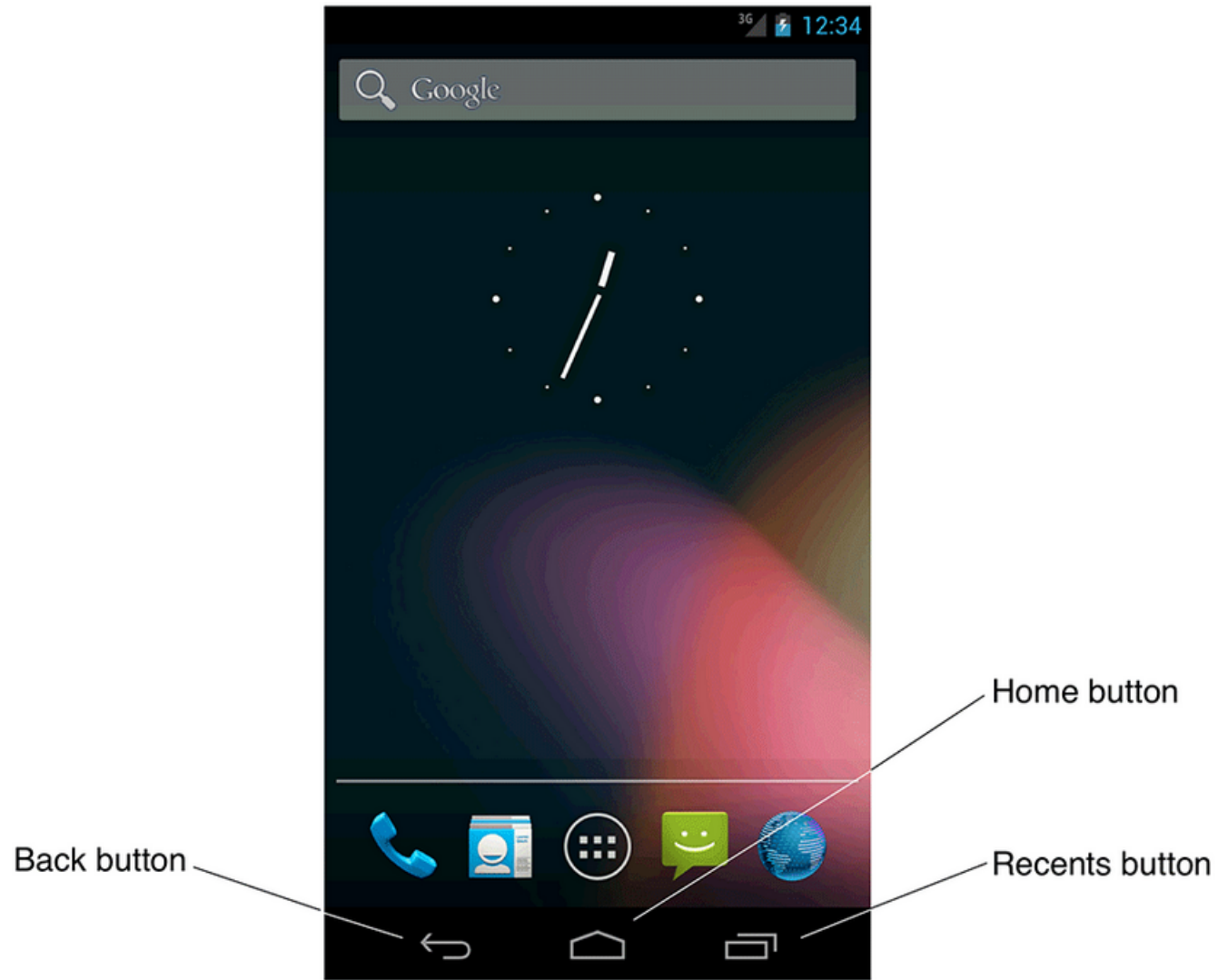
Android's Process Model



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

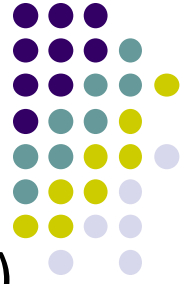
Pressing Back or destroying A1 will bring me to the top

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



Android Activity LifeCycle

Activity Callbacks



- Android applications don't start with a call to `main(String[])`
- Instead callbacks invoked corresponding to app state.
- Activity callbacks:
 - `onCreate()`
 - `onStart()`
 - `onResume()`
 - `onPause()`
 - `onStop()`
 - `onRestart()`
 - `onDestroy()`
- Examples:
 - When activity is created, its `onCreate()` method invoked
 - When activity is paused, its `onPause()` method invoked

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>



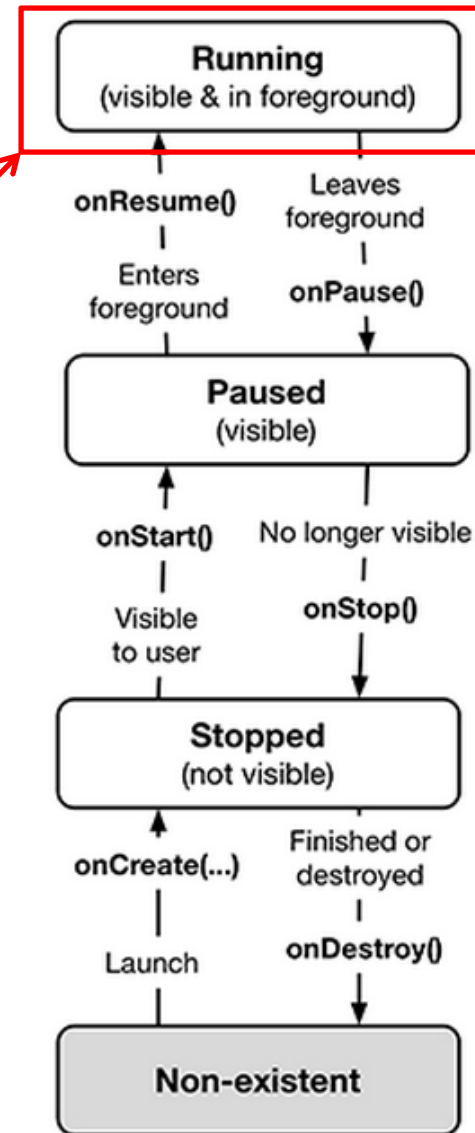
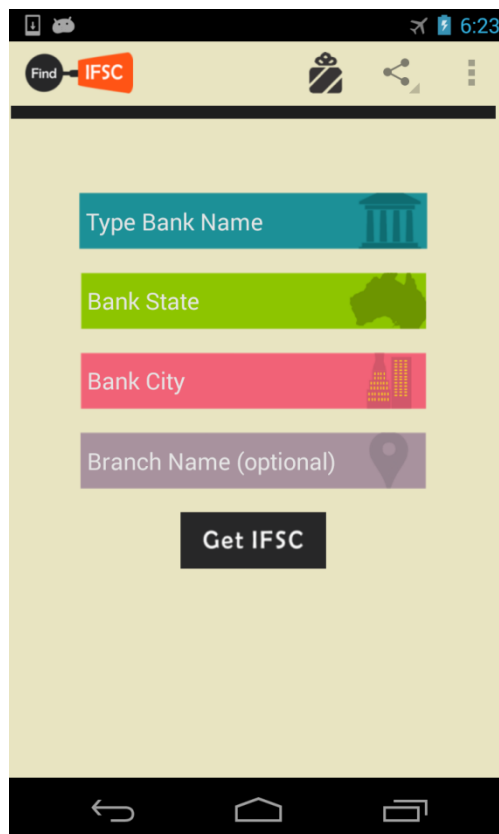
onCreate()

- 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



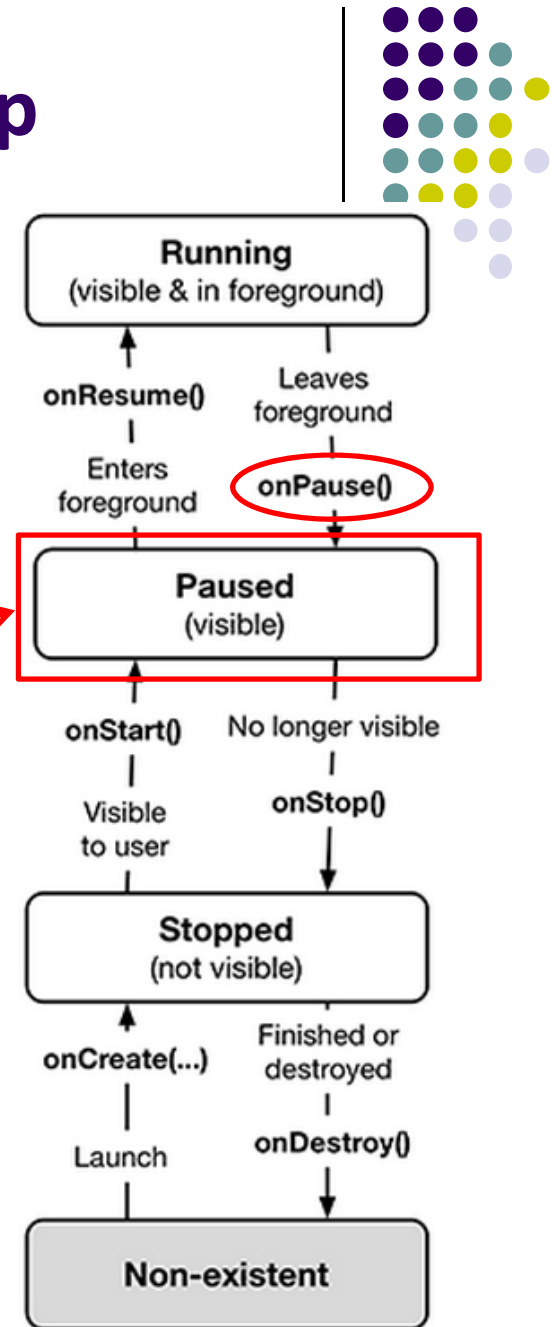
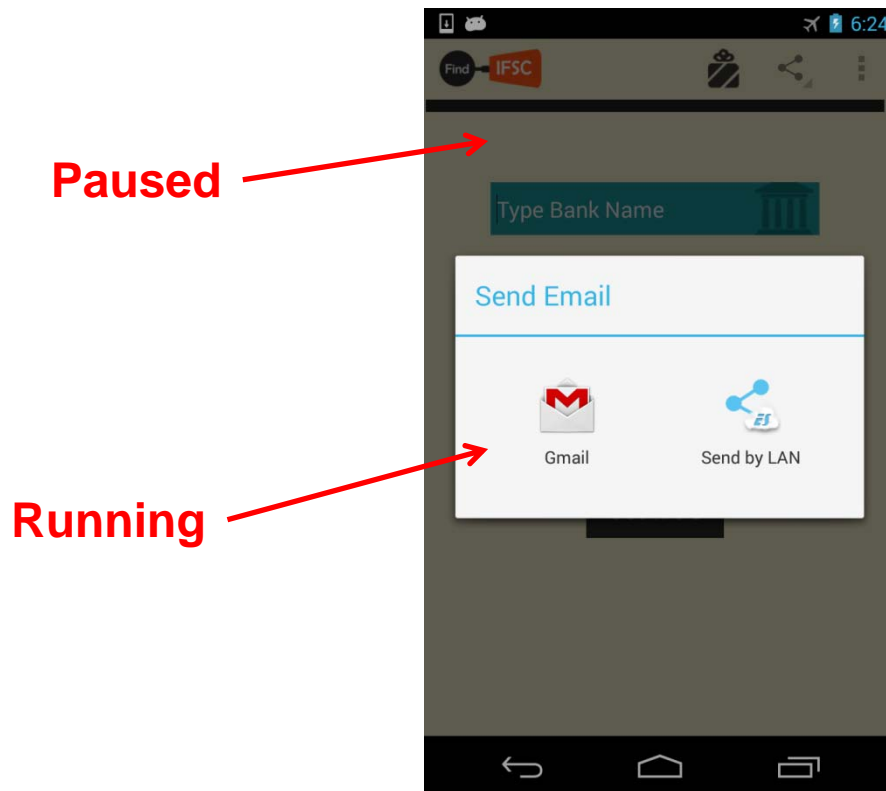
Activity State Diagram: Running App

- 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

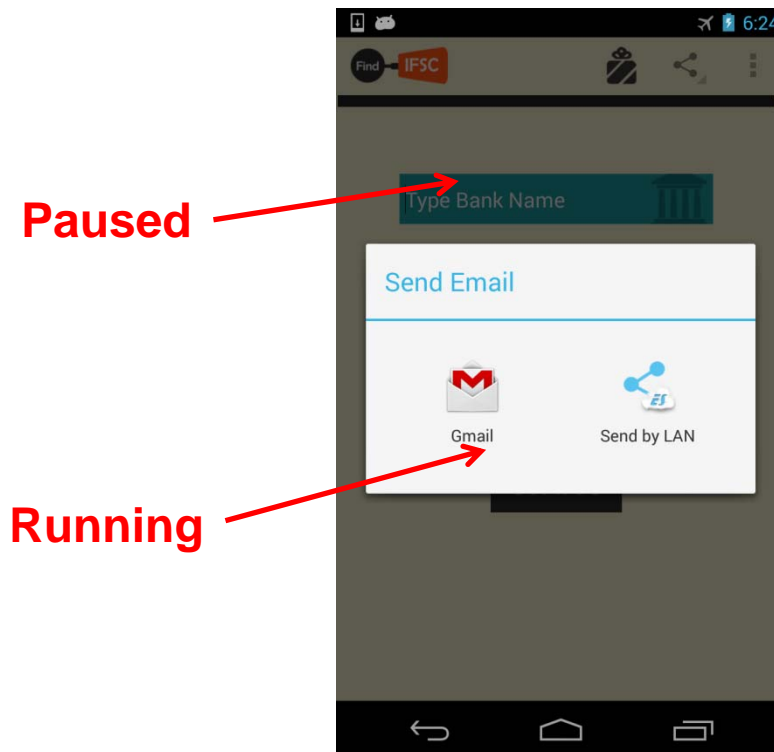
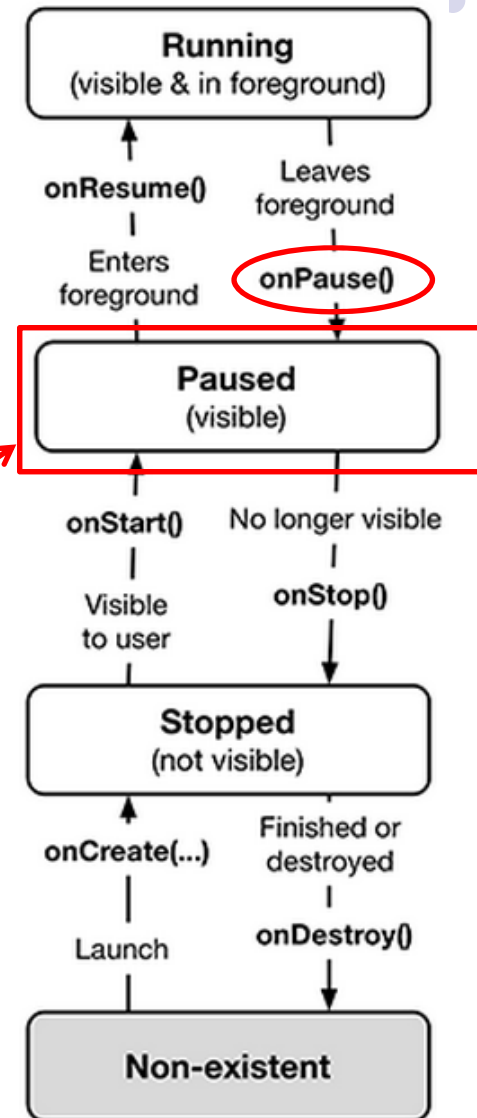
- An app is **paused** if it is **visible** but no longer in foreground
- E.g. blocked by a pop-up dialog box
- App's **onPause()** method is called to transition from running to paused state



Activity State Diagram: onPause() Method

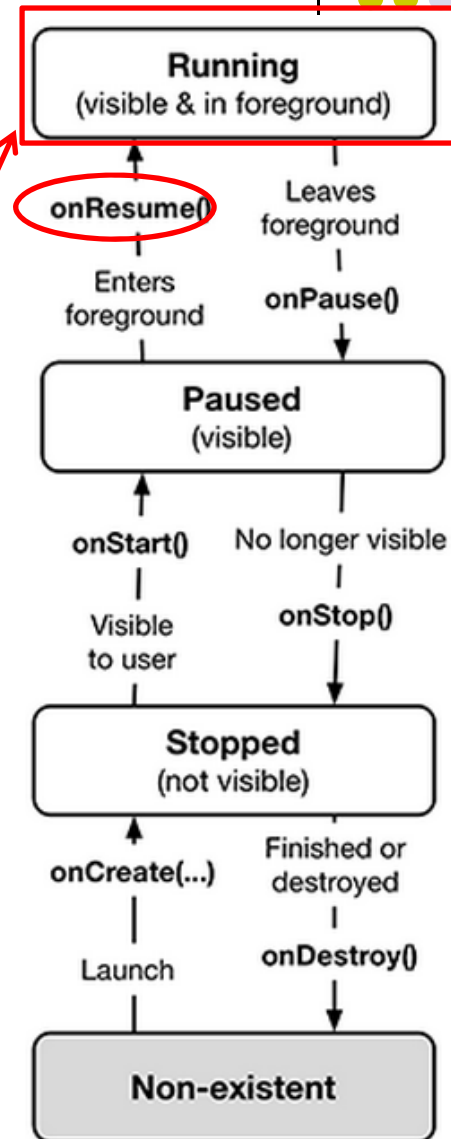
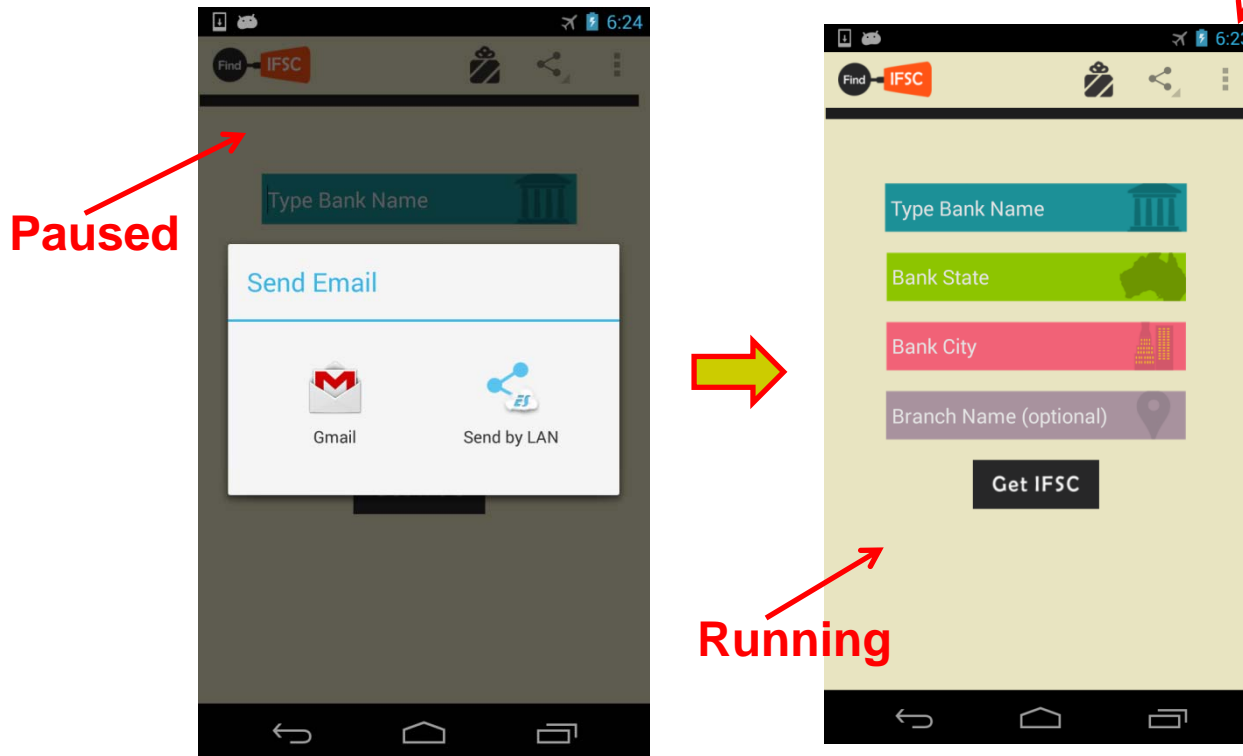


- Typical actions taken in onPause() method
 - Stop animations and CPU intensive tasks
 - Stop listening for GPS, broadcast information
 - Release handles to sensors (e.g GPS, camera)
 - Stop audio and video if appropriate



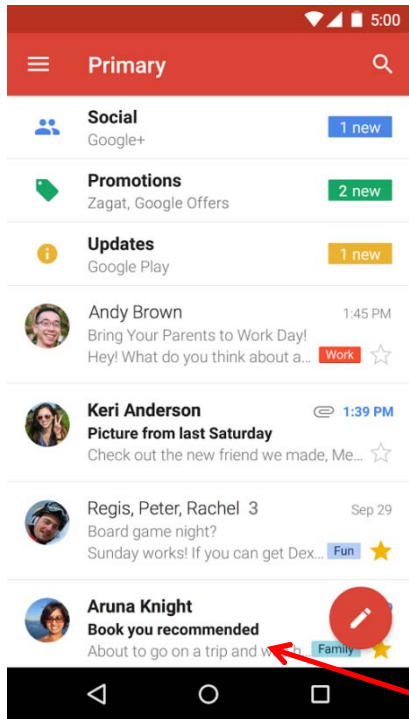
Activity State Diagram: Resuming Paused App

- 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

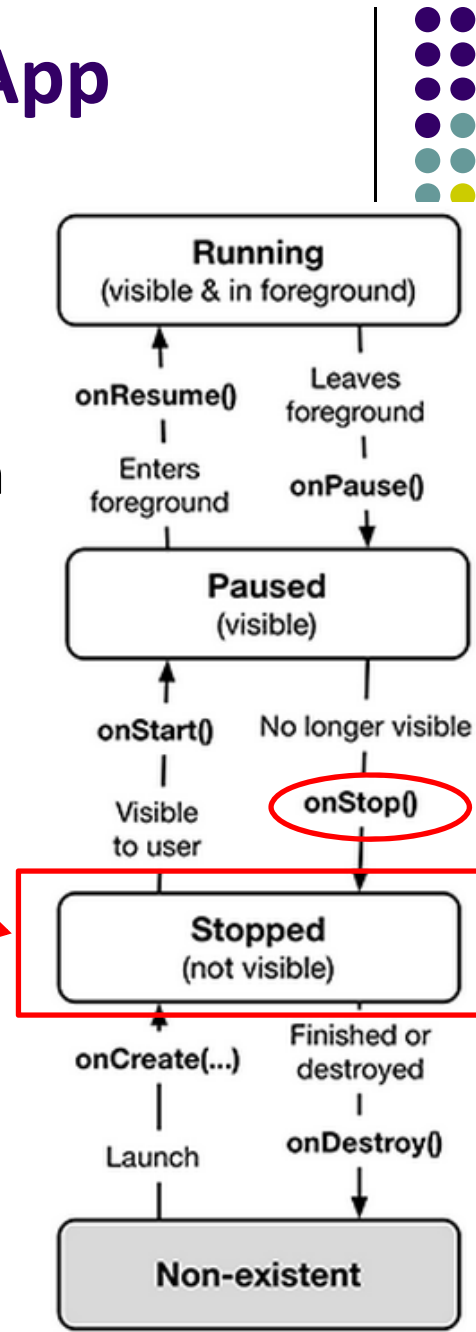
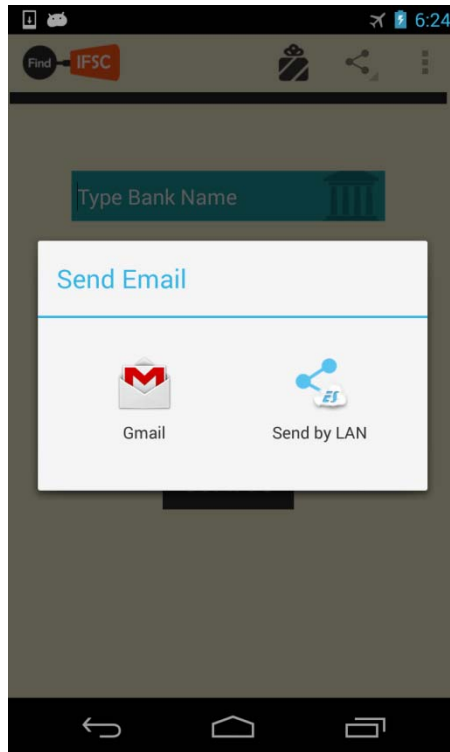


Activity State Diagram: Stopped App

- 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



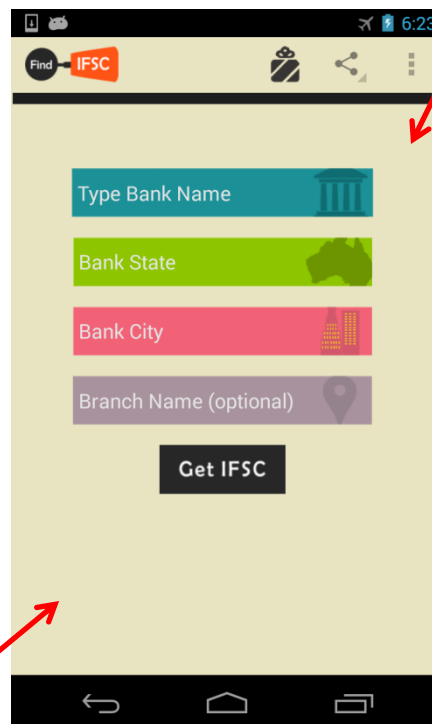
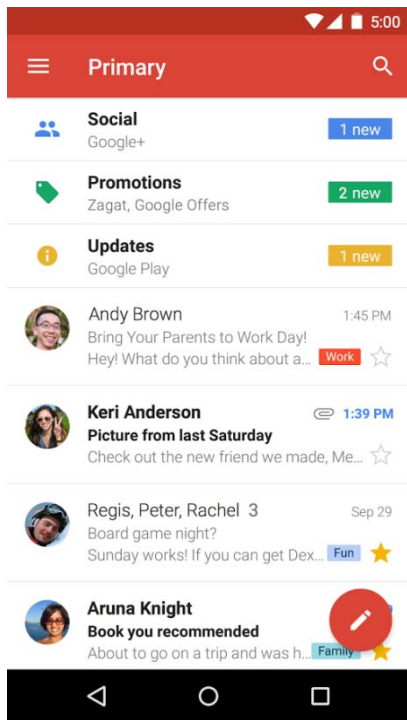


Saving State

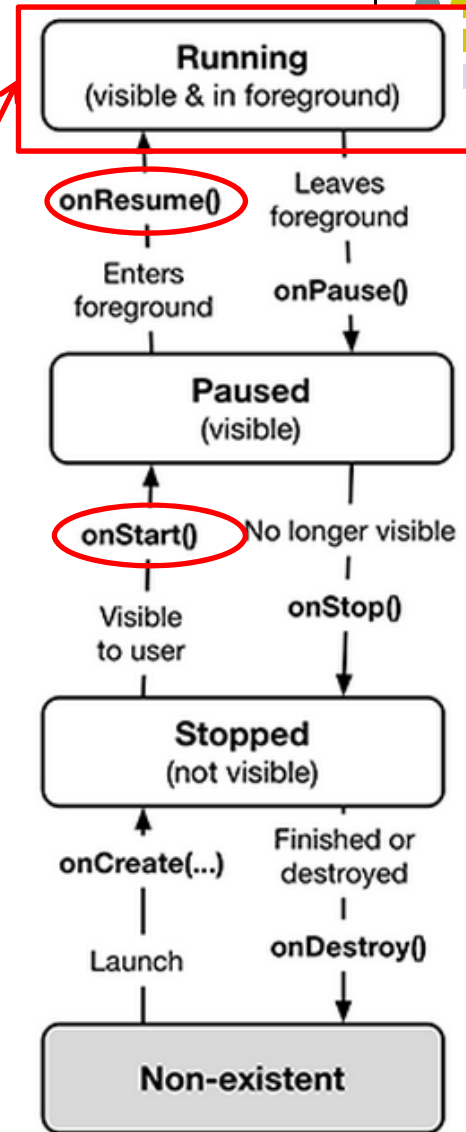
- If activity is stopped, in `onStop()` method, well behaved apps should
 - save progress (e.g. app state) to enable seamless restart later
 - Release all resources and save information (persistence)
- When activity is destroyed the Activity object is destroyed
 - Can save data by writing it to Bundle (a data structure)
 - Bundle given back when restarted
 - can save information via `onSaveInstanceState(Bundle outState)` method.

Activity State Diagram: Stopped App

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

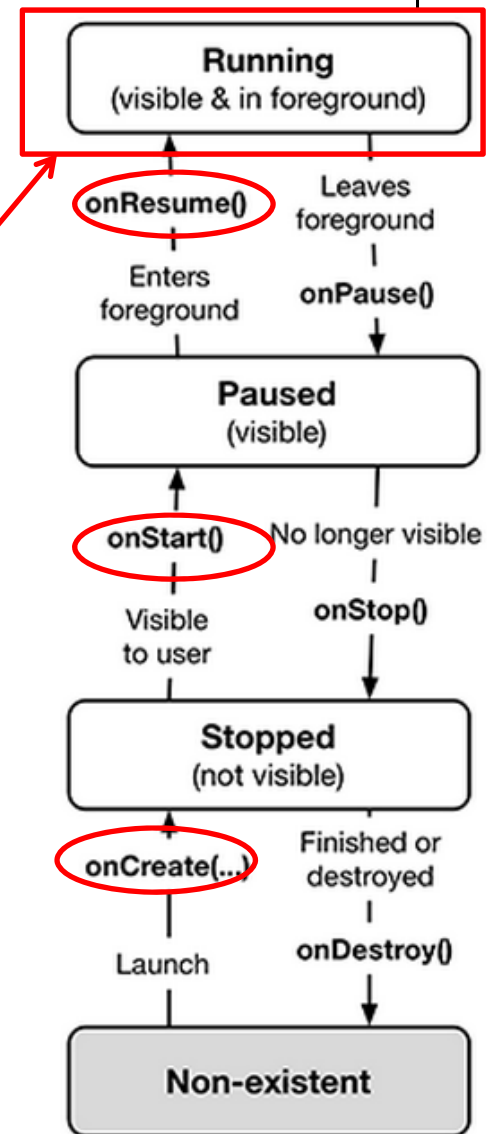
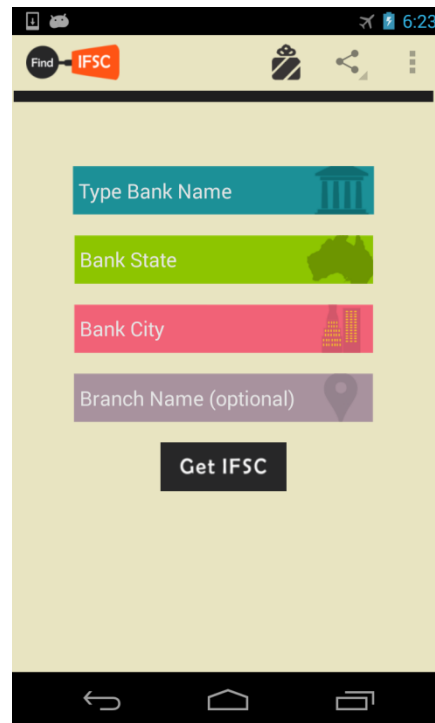


Running



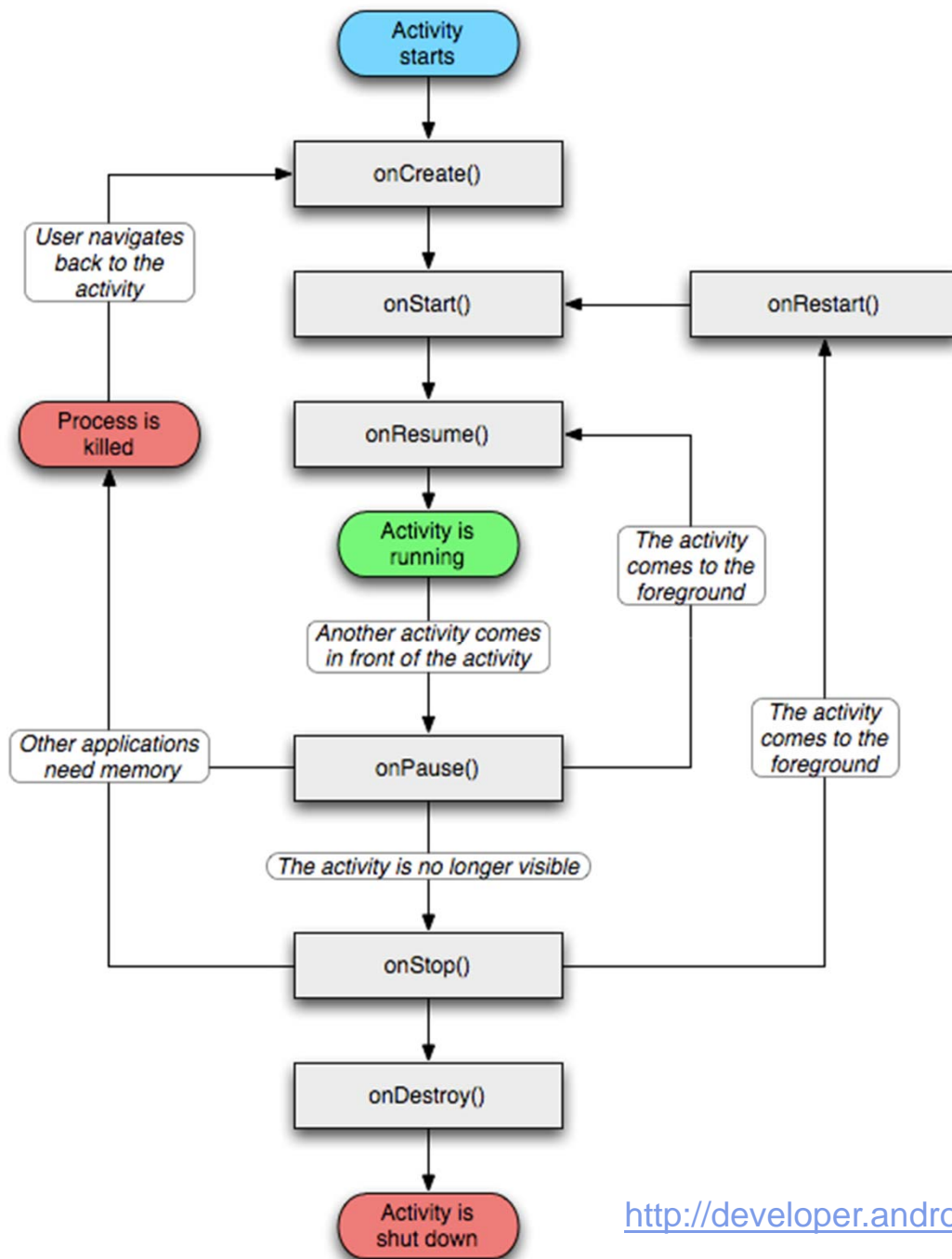
Activity State Diagram: Starting New App

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





Activity Lifecycle Diagram



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



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

(Logging Error Example: Ref. Android Nerd Ranch pg 53-56)

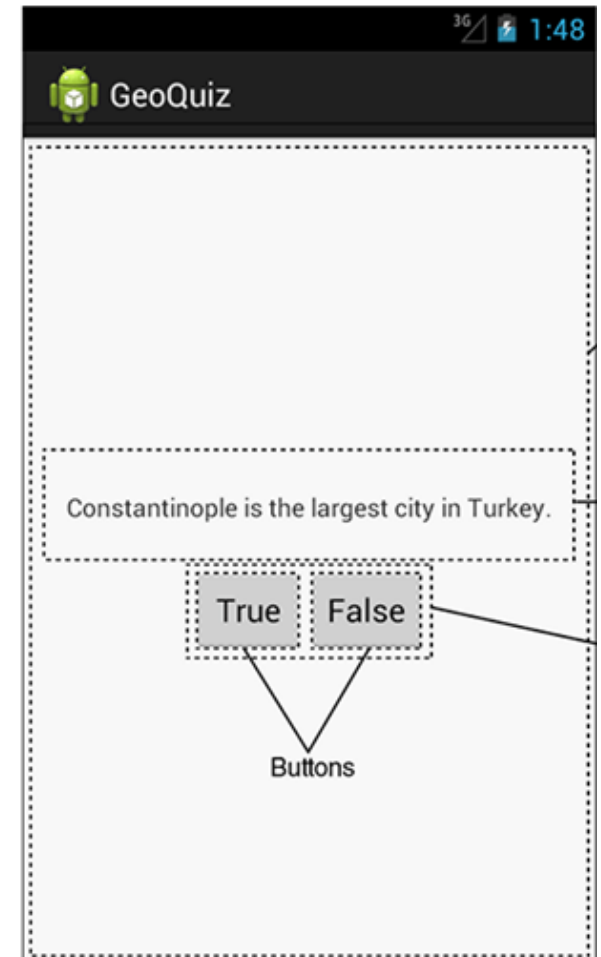
- A good way to understand Android lifecycle methods is to print debug messages when they are called
- E.g. print debug message from onCreate method below

```
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

(Logging Error Example: Ref. Android Nerd Ranch pg 53-56)

- Debug (d) messages have the form

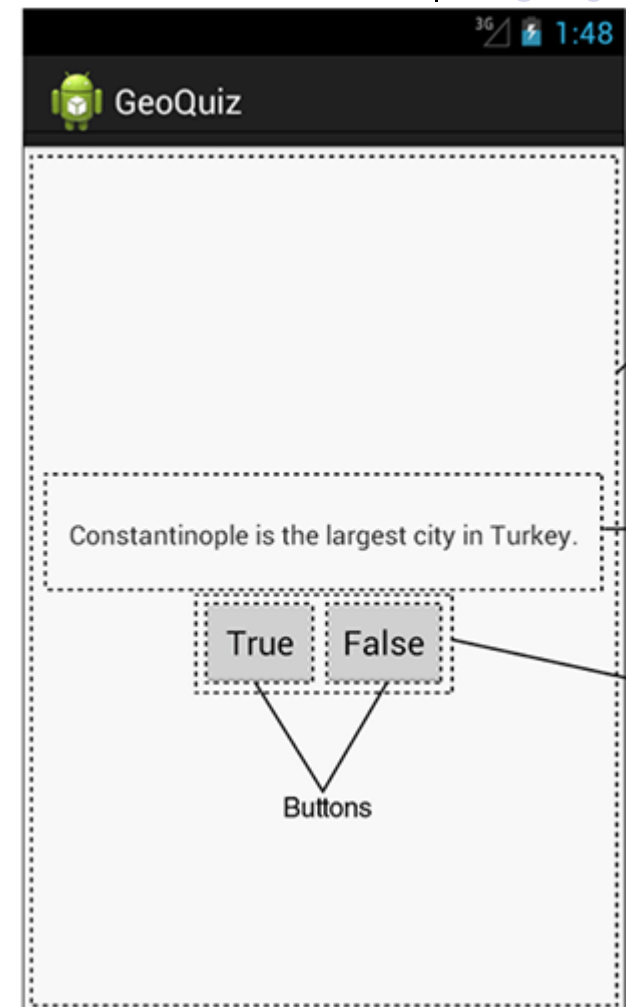
```
public static int d(String tag, String msg)
```

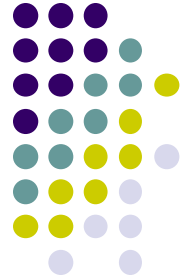
- **TAG:** Activity sending message (source)
- Declare string for **TAG**

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

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

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



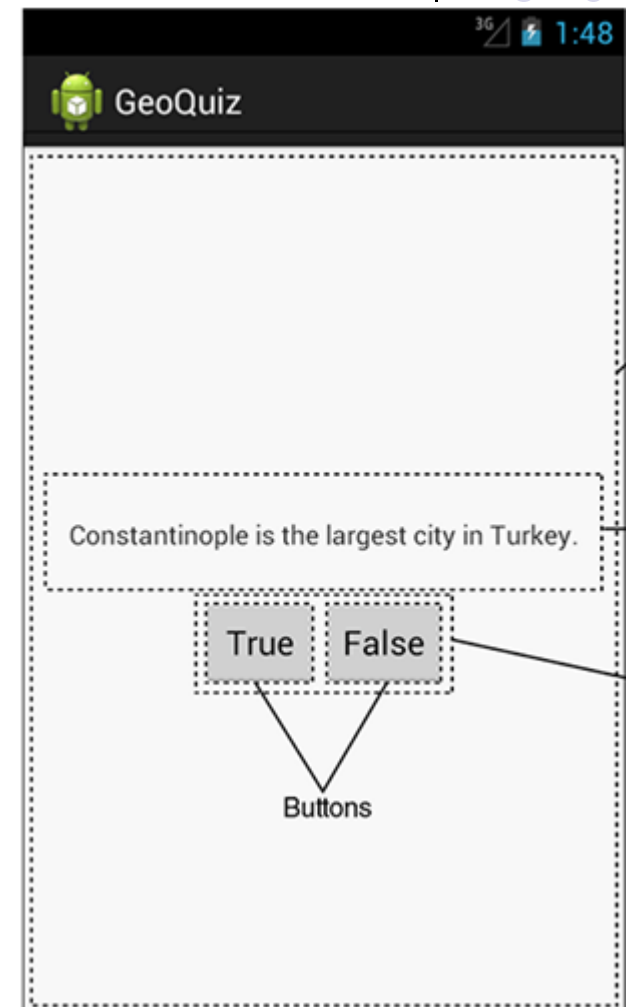


QuizActivity.java

(Logging Error Example: Ref. Android Nerd Ranch pg 53-56)

- 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");
}

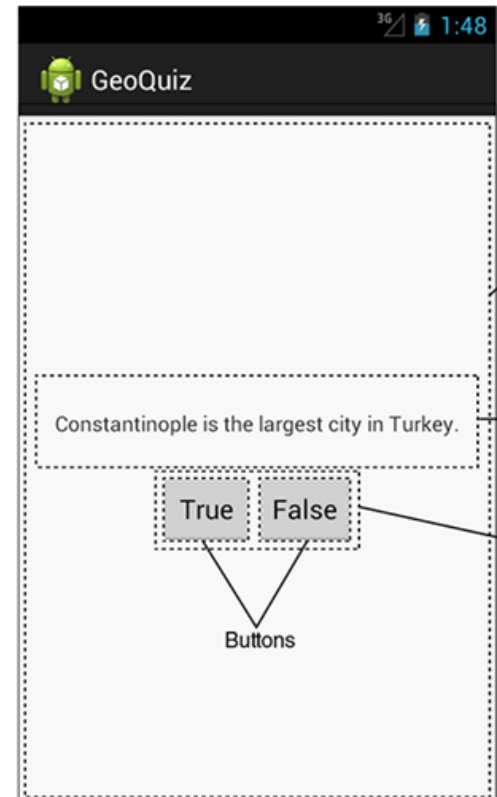
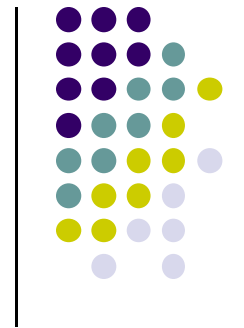
@Override
public void onPause() {
    super.onPause();
    Log.d(TAG, "onPause() called");
}

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

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

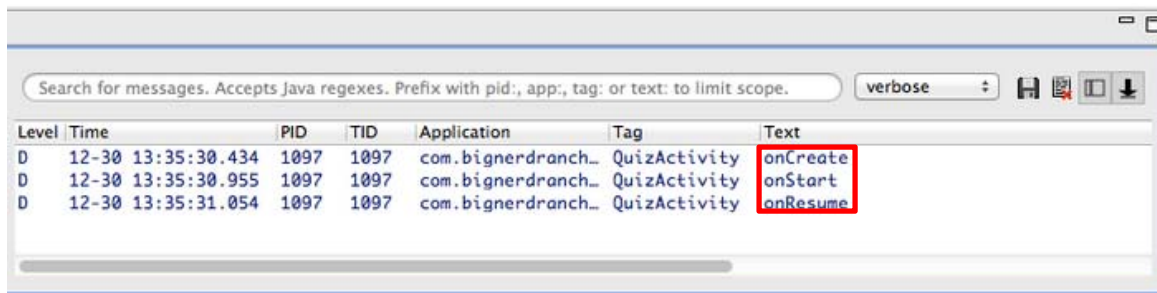
@Override
public void onDestroy() {
    super.onDestroy();
    Log.d(TAG, "onDestroy() called");
}

}
```



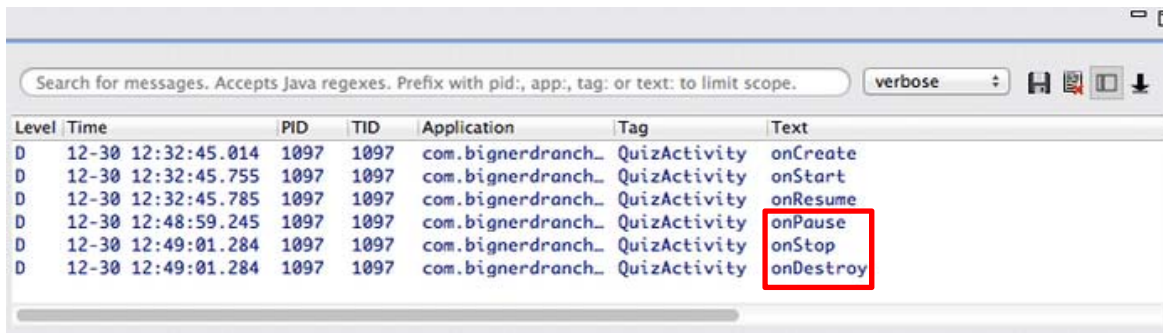
QuizActivity.java Debug Messages

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

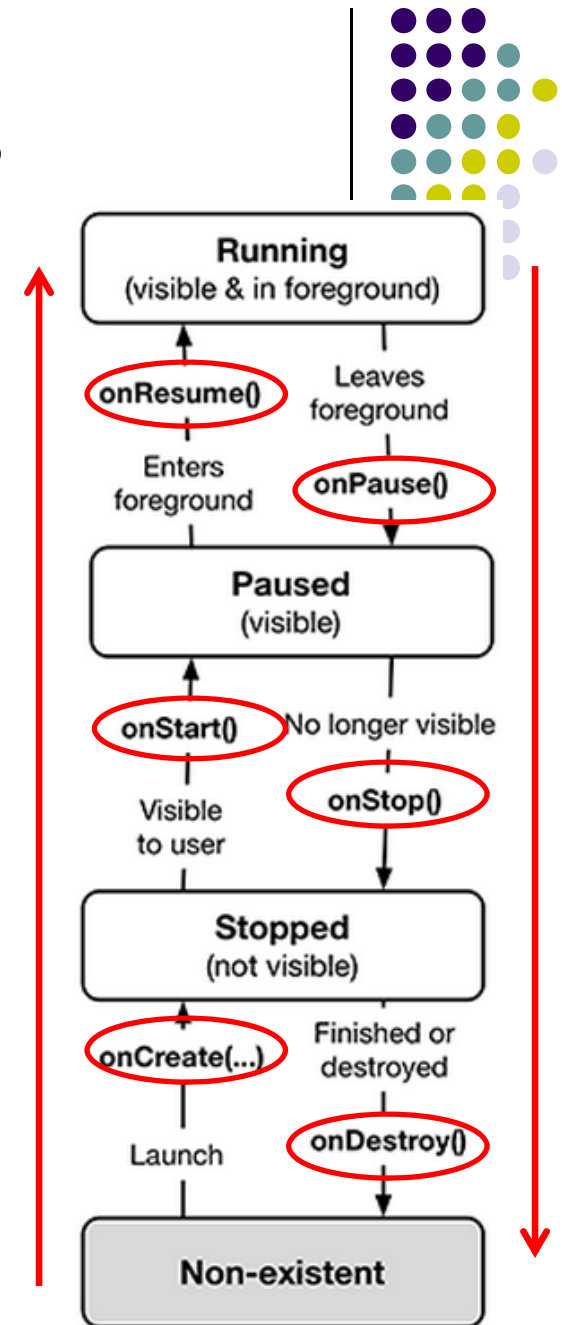


Level	Time	PID	TID	Application	Tag	Text
D	12-30 13:35:30.434	1097	1097	com.bignerdranch...	QuizActivity	onCreate
D	12-30 13:35:30.955	1097	1097	com.bignerdranch...	QuizActivity	onStart
D	12-30 13:35:31.054	1097	1097	com.bignerdranch...	QuizActivity	onResume

- 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

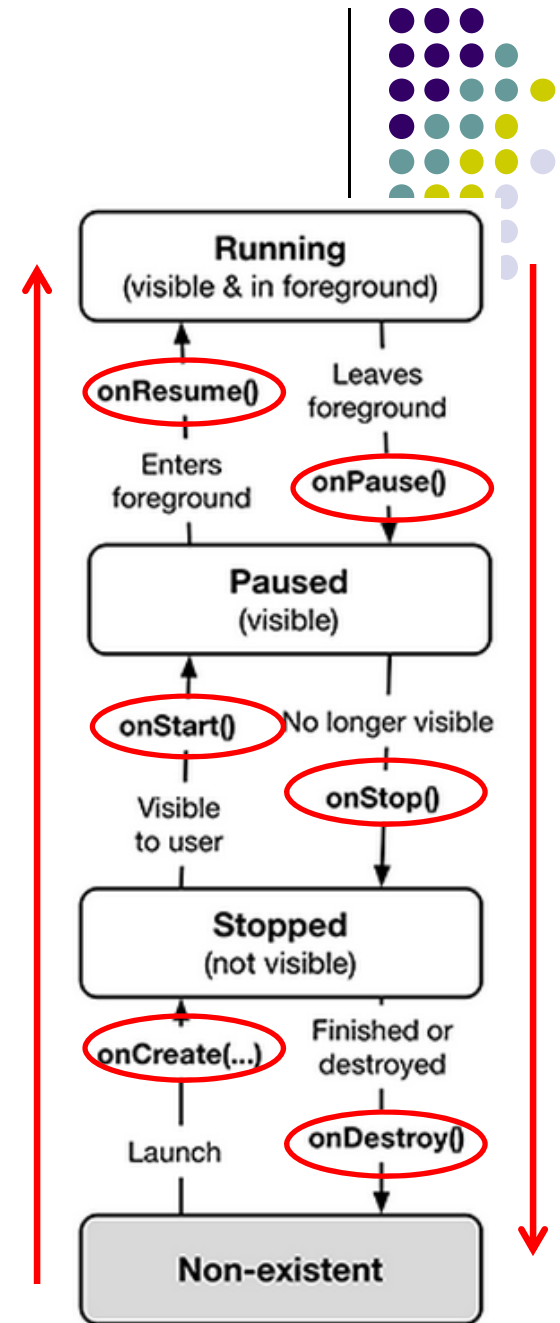


QuizActivity.java Debug Messages

- Pressing **Home** button stops the activity

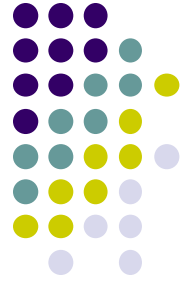
Level	Time	PID	TID	Application	Tag	Text
D	12-30 12:49:01.284	1097	1097	com.bignerdranch_	QuizActivity	onStop
D	12-30 12:49:01.284	1097	1097	com.bignerdranch_	QuizActivity	onDestroy
D	12-30 12:50:01.087	1097	1097	com.bignerdranch_	QuizActivity	onCreate
D	12-30 12:50:01.715	1097	1097	com.bignerdranch_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

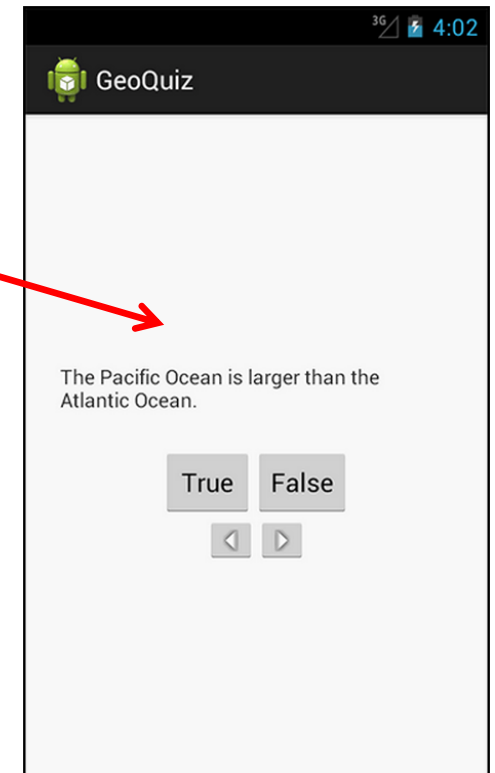
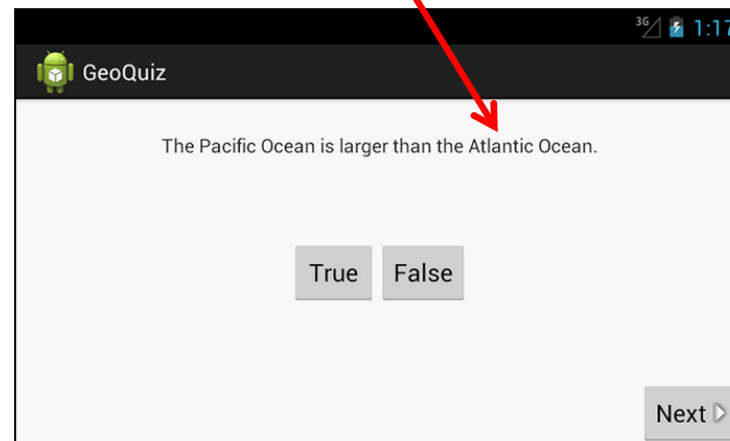


Rotating Device & Device Configuration

(Ref: Android Nerd Ranch pgs 61-64)



- 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 (XML files) for portrait vs landscape screen orientation

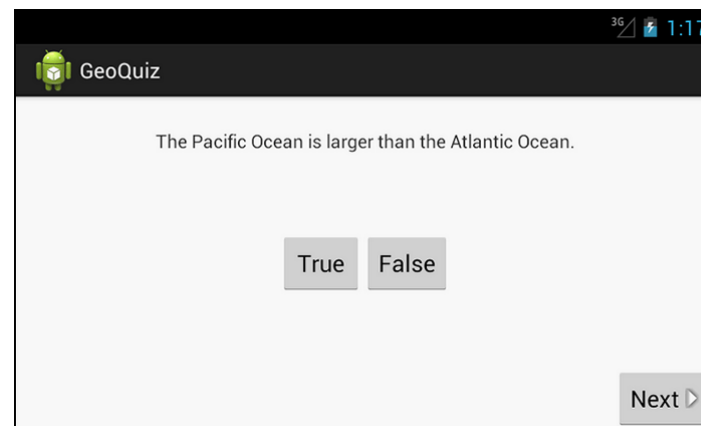
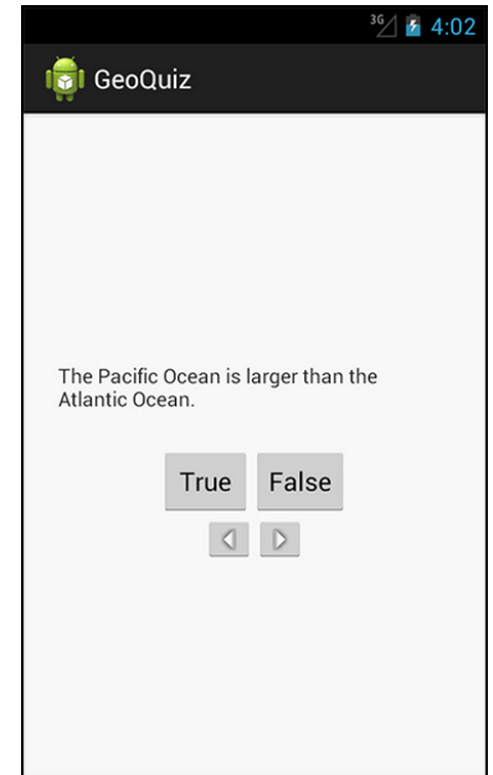


Rotating Device & Device Configuration

(Ref: Android Nerd Ranch pgs 61-64)

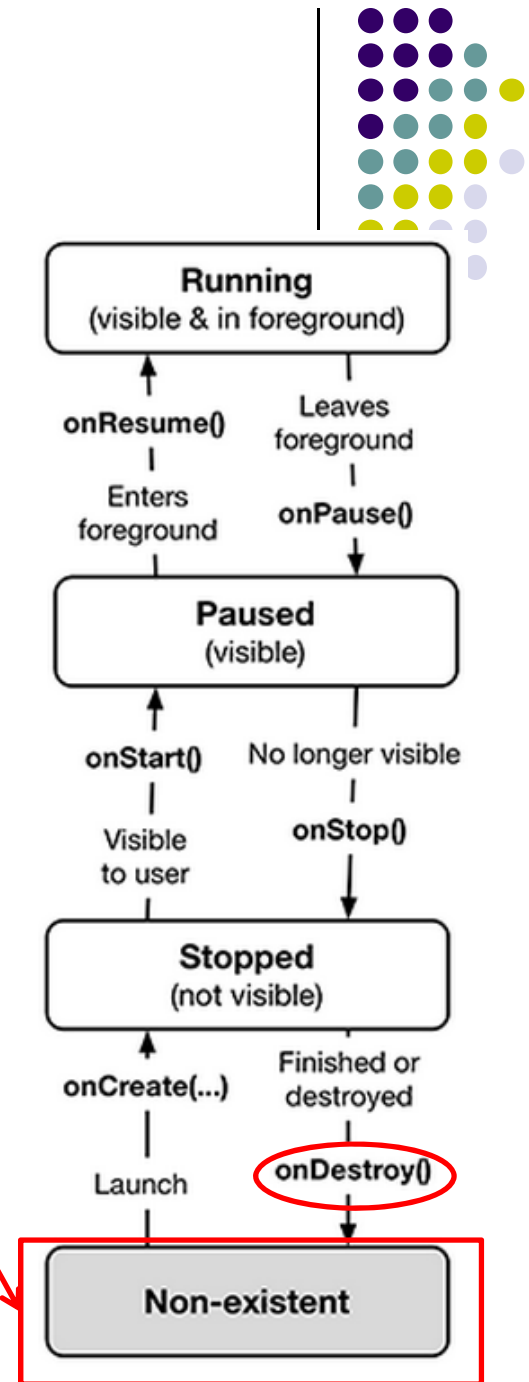


- How to use different app layouts for portrait vs landscape screen orientation?
- When device in landscape, uses resources in **res/layout-land/**
- Copy XML layout file (activity_quiz.xml) from **res/layout** to **res/layout-land/** and tailor it
- When configuration changes, current activity destroyed,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



Destroyed Activity

- Dead, activity terminated (or never started)
- `onDestroy()` called to destroy a stopped app





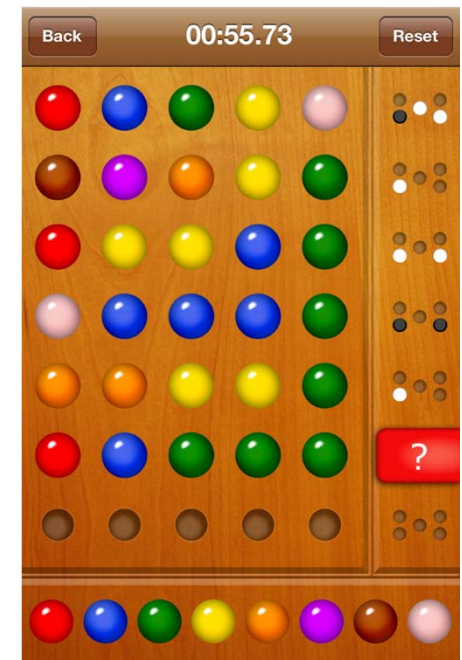
Activity Destruction

- 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 recreated later, system calls onSaveInstanceState (Bundle outState) method

onSaveInstanceState onRestoreInstanceState()



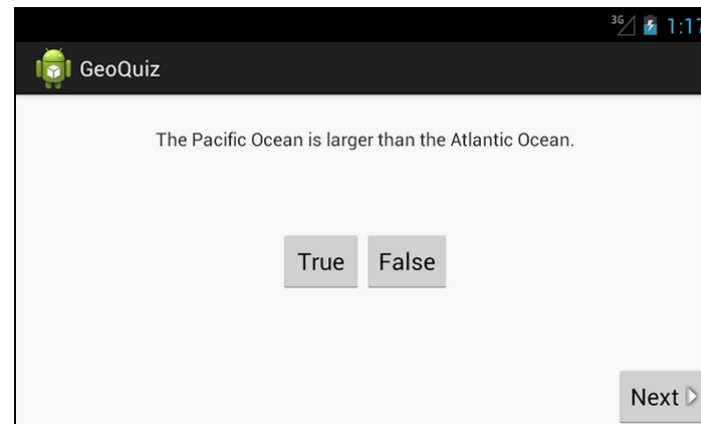
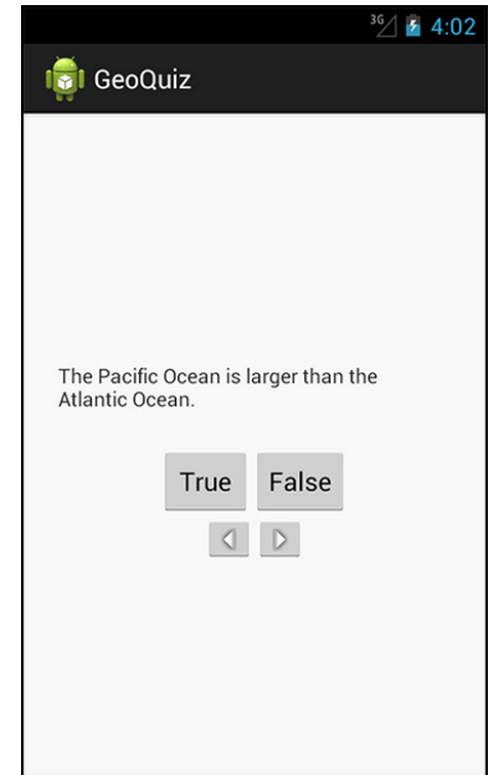
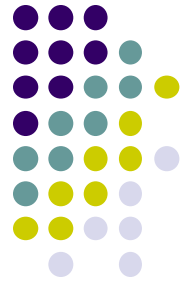
- Systems write info about views to Bundle
- other (app-specific) information must be added by programmer
 - E.g. board state in a board game such as mastermind
- When Activity recreated Bundle sent to onCreate and onRestoreInstanceState()
- use either method to restore state data / instance variables



Saving Data Across Device Rotation

(Ref: Android Nerd Ranch pgs 64-66)

- 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()**



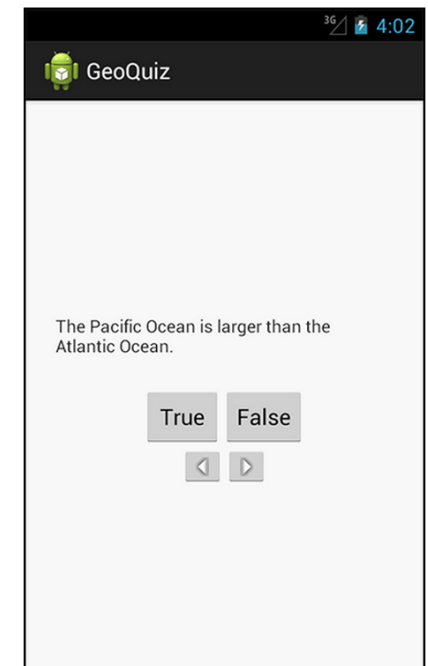
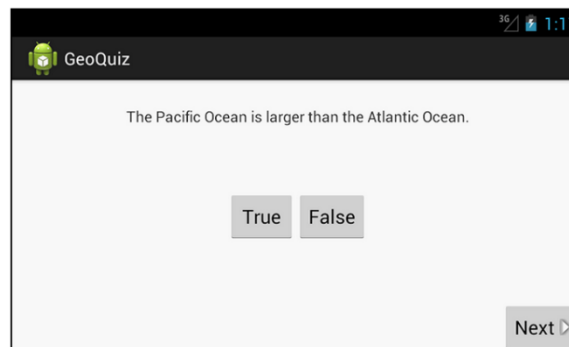
Saving Data Across Device Rotation

- 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