Case Study: Mobile Devices

Smartphones, tablets and other devices

Design considerations

Android as a platform

Design

What makes mobile design different?

Recall: Touch Interfaces

In this course, we have mostly discussed the development of computer interfaces, with the assumption of standard input devices (e.g., mouse, keyboards). These also apply to web interaction on desktop/laptop computers.

Mobile devices often rely on direct input using touch interaction or a stylus (plus gestures, plus voice).

Device Characteristics

- Small form factor
  - Limited screen size
  - Multiple orientations with dynamic layout
  - Single application focus
- Hybrid interaction
  - Virtual keyboard
  - Direct manipulation
  - Surface gestures
  - Voice
- Limited resources
  - Limited memory, processing power
  - Battery life is critical!

It's all about tradeoffs

"One way to look at design — at any kind of design — is that it's essentially about constraints (things you have to do and things you can't do) and tradeoffs (the less-than-ideal choices you make to live within the constraints)."

- Steve Krug ("Don't Make Me Think Revisited")

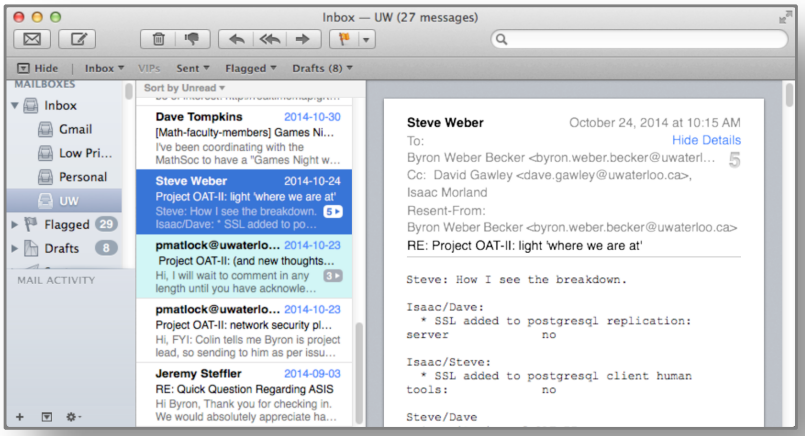
Compromise

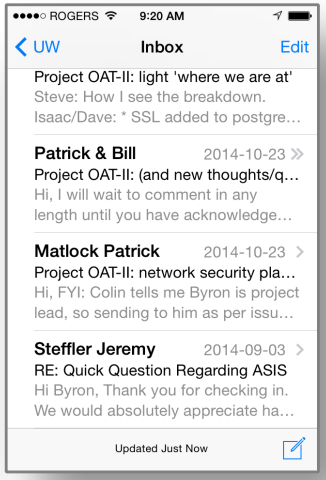
Touch interfaces introduce new challenges to the design and implementation of user interfaces.

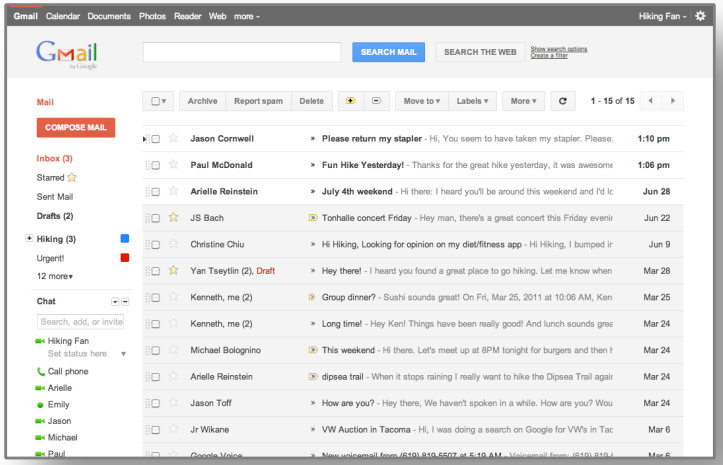
To build effective user interfaces for mobile devices and tabletop, you need to be aware of the limitations of the sensing display, input methods, then design interfaces and interaction to fit those limitations, e.g.,

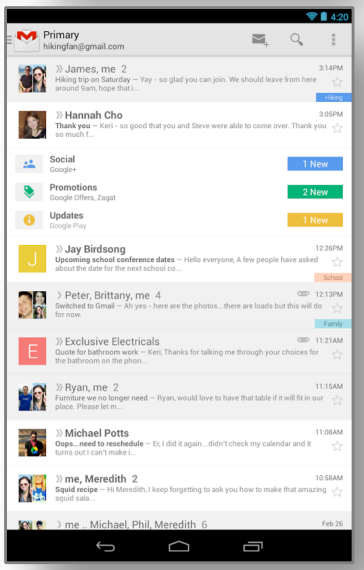
- varying screen sizes (too small to too big)
- fat finger problem (occlusion and imprecision)
- high-variable input (i.e., gesture) to output mapping
- ambiguity in input interpretation and feedback

Desktop vs. Mobile









Constraints: Display Size

Display is Smaller

- large variety of sizes (phone, phablet, tablet)
- orientation changes from portrait to landscape
- large variety of spatial resolutions

Constraints: Interface

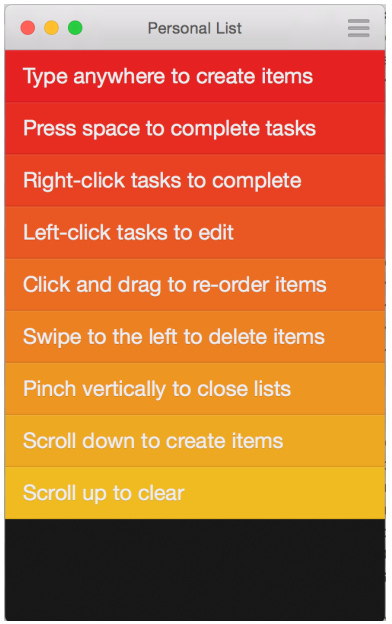
Touch-based interface

- Controls need to be large (precision)
- Interface needs to be simple, usable with one-hand
- Interaction is a sequence of screens

Minimal help

- Needs to be intuitive and easy to learn
- No hints available via hover

"Clear"



Steve Krug ("Don't Make Me Think Revisited")

Constraints: Limited Processing

Limited processing capabilities

- Intensive tasks need to be done offline/preprocessed

Single application model

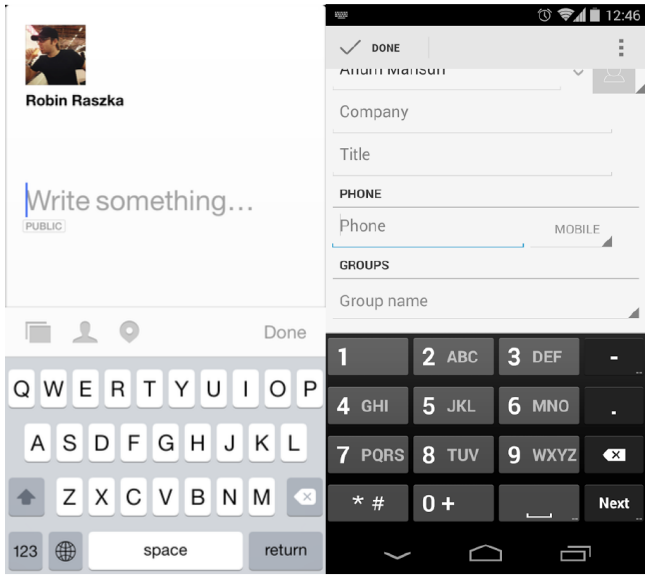
- One app in the foreground, others suspended
- Few active background processes
- Primarily full-screen apps, consisting of a sequence of screens
  - Limits interaction but also limits processing requirements

Responsiveness

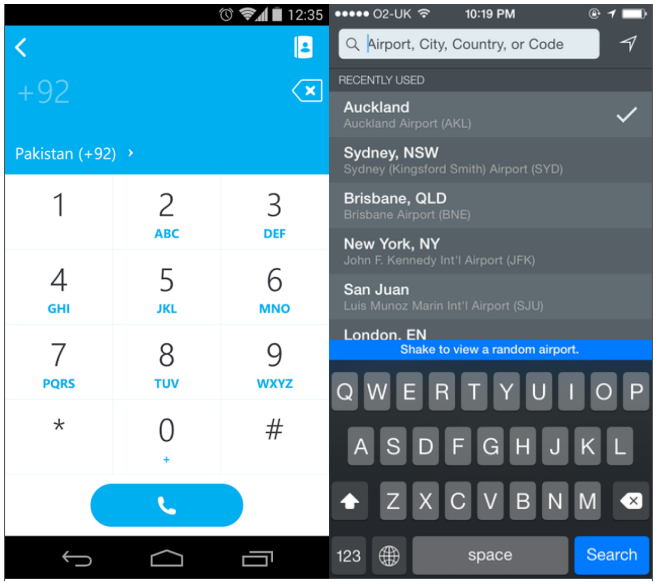
- Variable bandwidth
- Must work in different environments/conditions

Tip: Help Users to Enter Information

Provide the Right Data Entry Tool



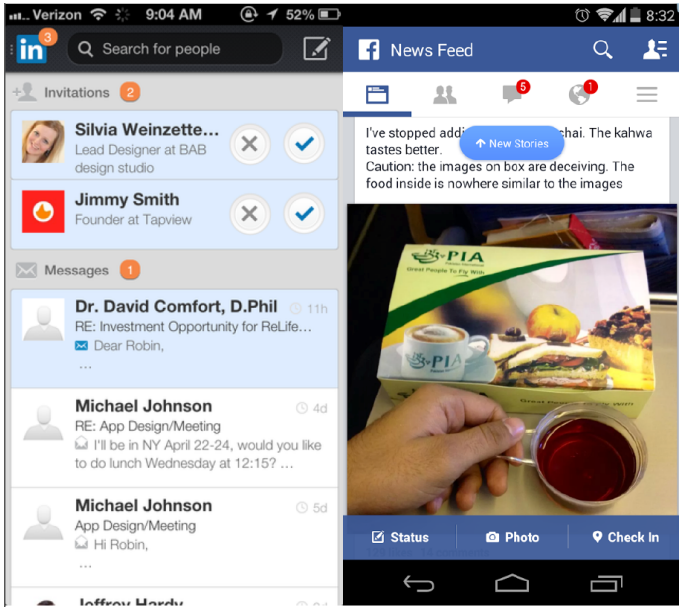
Anticipate and Predict Input

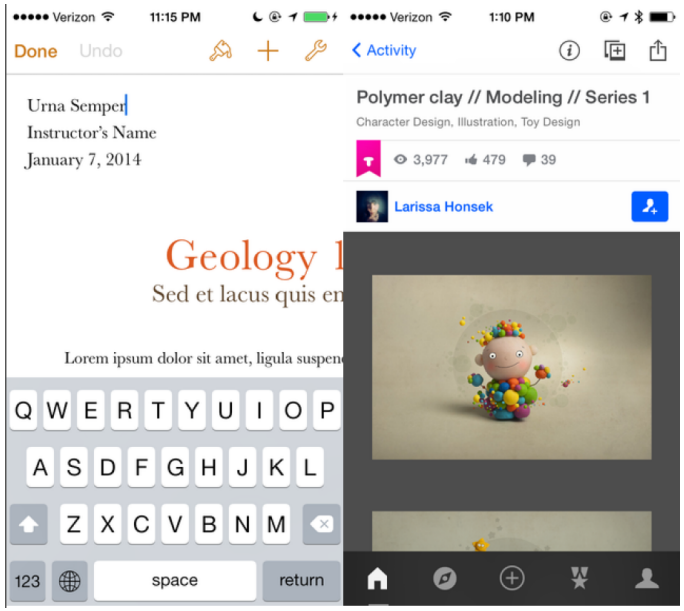


"Mobile UI Design Pattern" (Bank and Zuberi)

Tip: Help Users Find Correct Actions

Highlight New Content

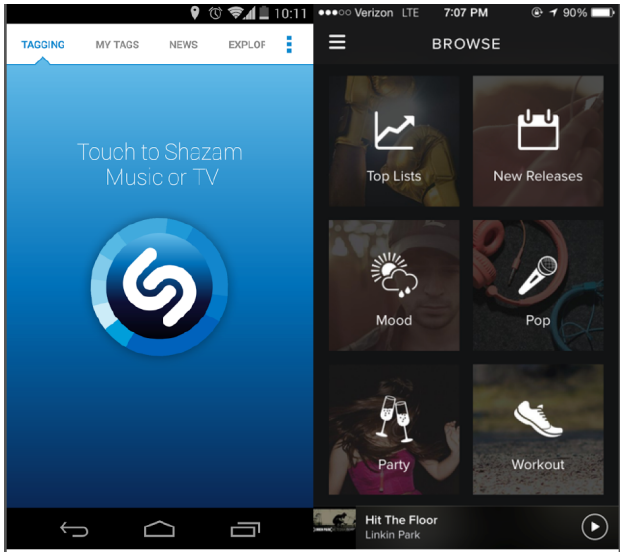




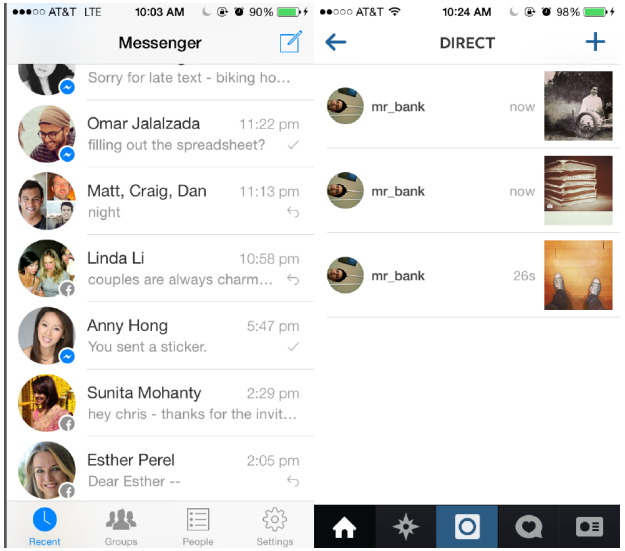
"Mobile UI Design Pattern" (Bank and Zuberi)

Tip: Help Users Find Correct Actions

Make Actions Obvious



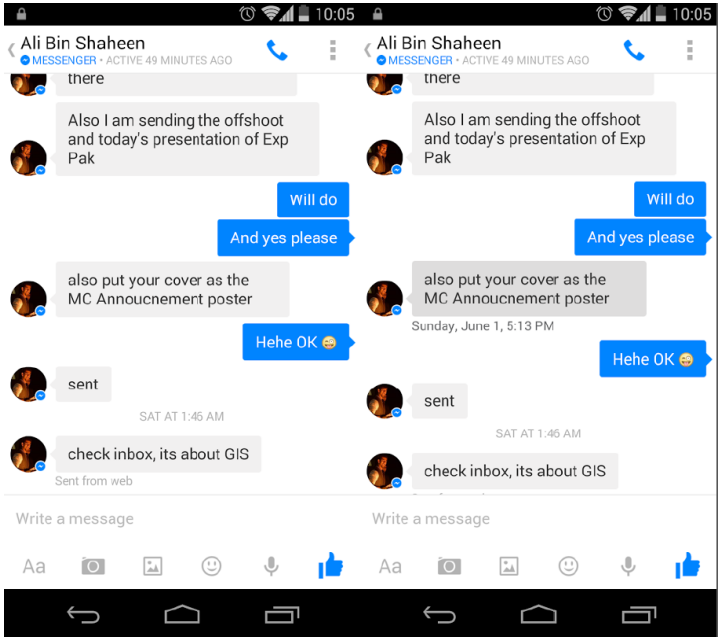
Accordance: Control vs Content



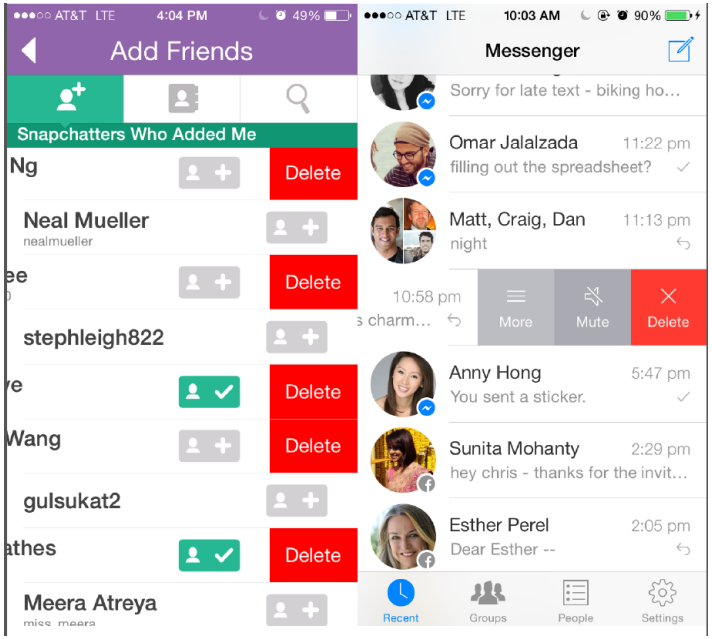
"Mobile UI Design Pattern" (Bank and Zuberi)

Tip: Utilize Real Estate and Avoid Clutter

Hide Metadata

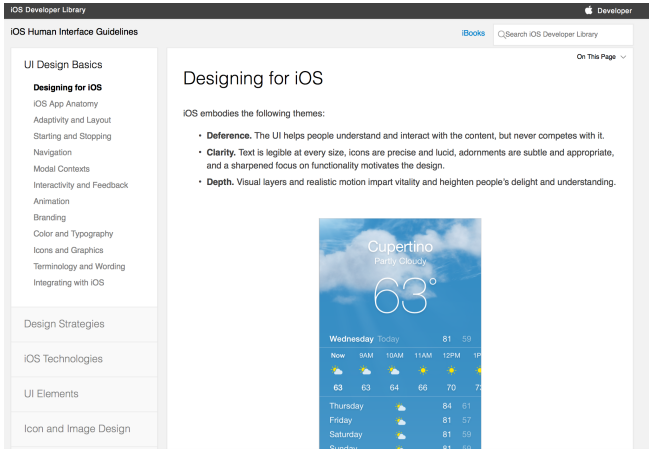


Hide Secondary Menus



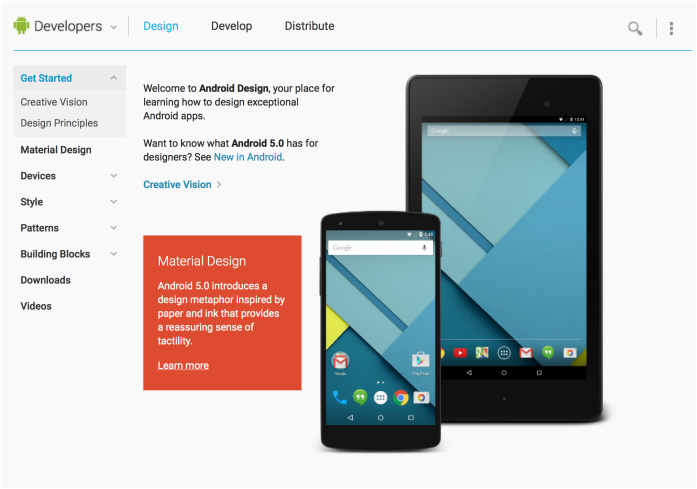
"Mobile UI Design Pattern" (Bank and Zuberi)

Standards: Interface Guidelines



https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/

Platform-specific design guidelines can provide specific usage examples and hints, beyond these basic guidelines.



http://developer.android.com/design

Android Architecture

Application model

Events, Widgets

Painting and Drawing

Constraints

- We also have architectural constraints based on the limitations of a small device.
- Limited processing
  - Single application model, and limited background processing
  - Emphasis on efficient use of memory (e.g. OS reserves the right to “flush” unused apps to free up memory)
- Small battery
  - Limit intensive processing tasks
  - Dim screen, sleep aggressively

Application model

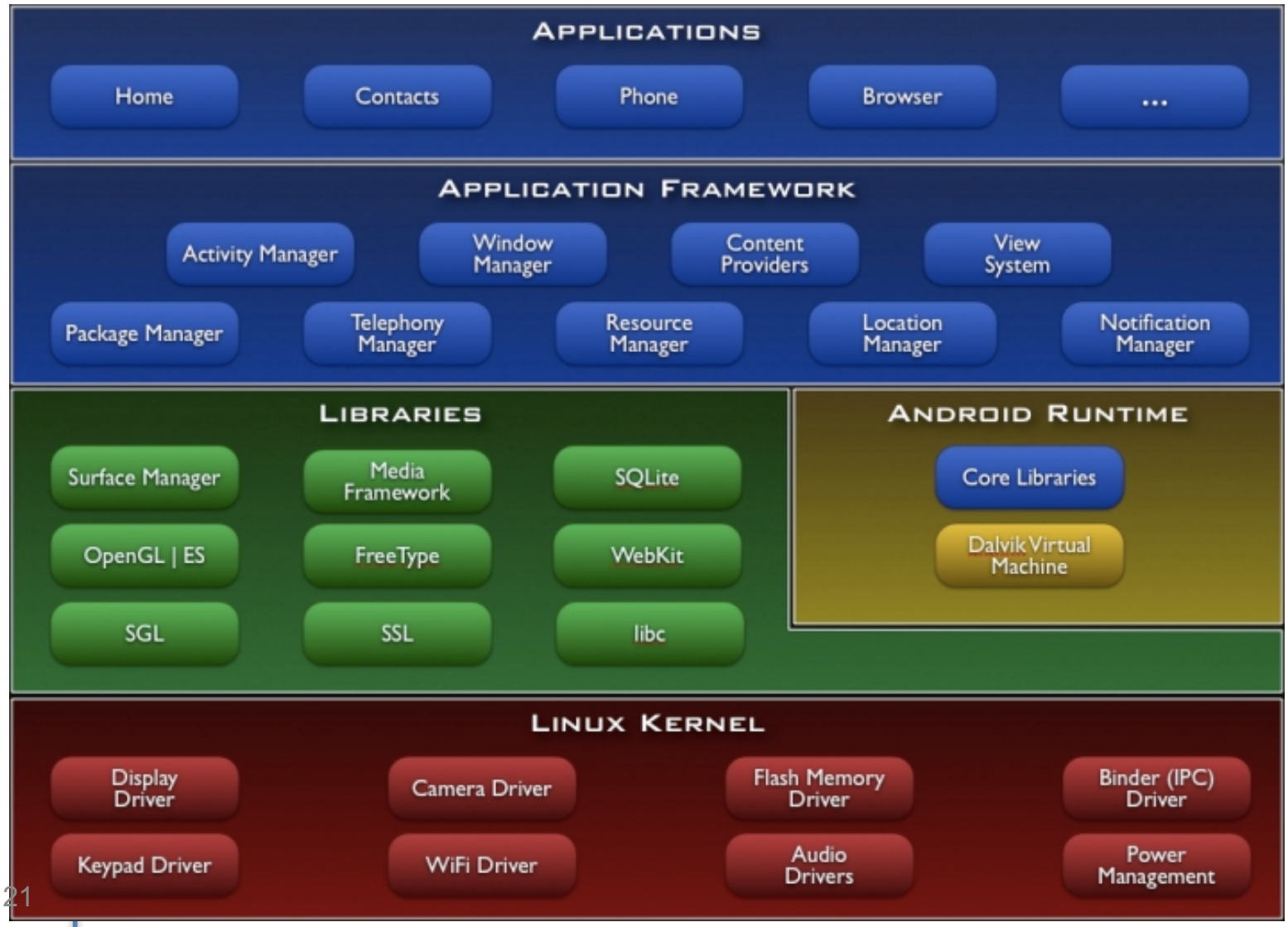
- Applications provide multiple entry points; they're distinct components that can be invoked individually.
  - Applications are typically composed of individual, standalone screens.
  - Explicitly need to pass data between screens (reduces overhead/memory).
- Applications are dynamic
  - You should provide different layouts (XML layouts, and resources) based on device characteristics.
  - Applications should dynamically adjust based on device screen size and orientation.

http://developer.android.com/guide/index.html

Layered architecture

- Applications are sandboxed and secure.
  - Layered environment
    - Bottom tier: Linux 2.6 kernel
    - Mid tier: Android libraries/services
    - Top tier: Java applications
  - Applications run in virtual machines (VM)
    - Each application has it's own VM & address space
    - Restrictions on sharing resources and data
    - Dalvik virtual machine process

Architecture



Activities

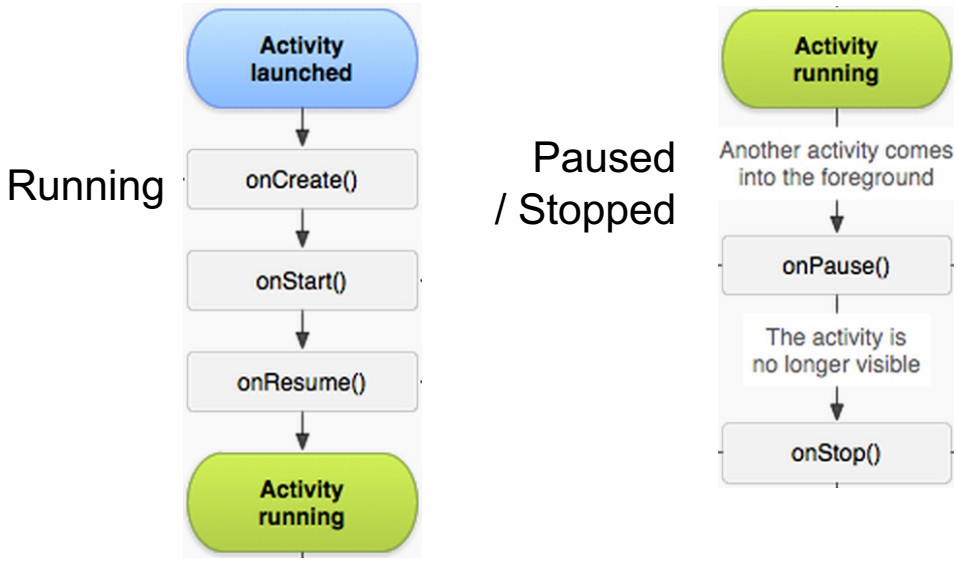
- Different types of components that we can build in Android:

Components	Description
Activity	Single-screen of an application
Service	Long-running background application
Content provider	Provides shared set of application data
Broadcast receiver	Responds to system broadcast events

- A standard application component is an **Activity**
  - Typically represents a single screen
  - Main entry point (equivalent to `main()` method)
  - For most purposes, this is your application class!

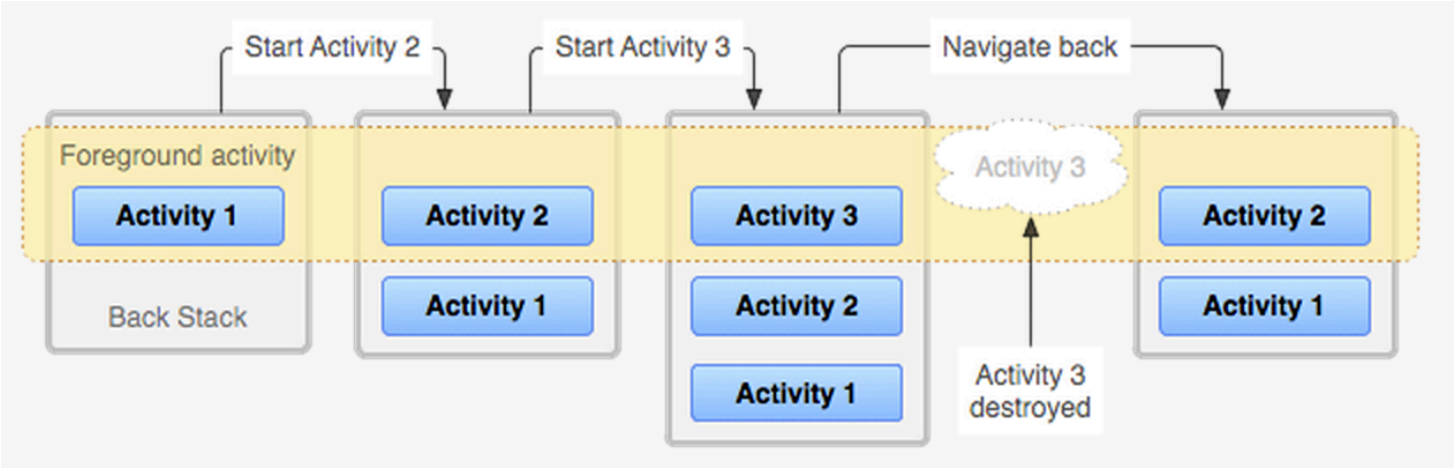
Activity Lifecycle

- Activities have an explicit lifecycle
  - One activity runs at a time, others are paused in the background.
  - As users navigate through your application, they switch activities.
- Every activity has a state*: run*, *paused*, or *stopped.*
  - Changing state fires a corresponding activity method for that state.



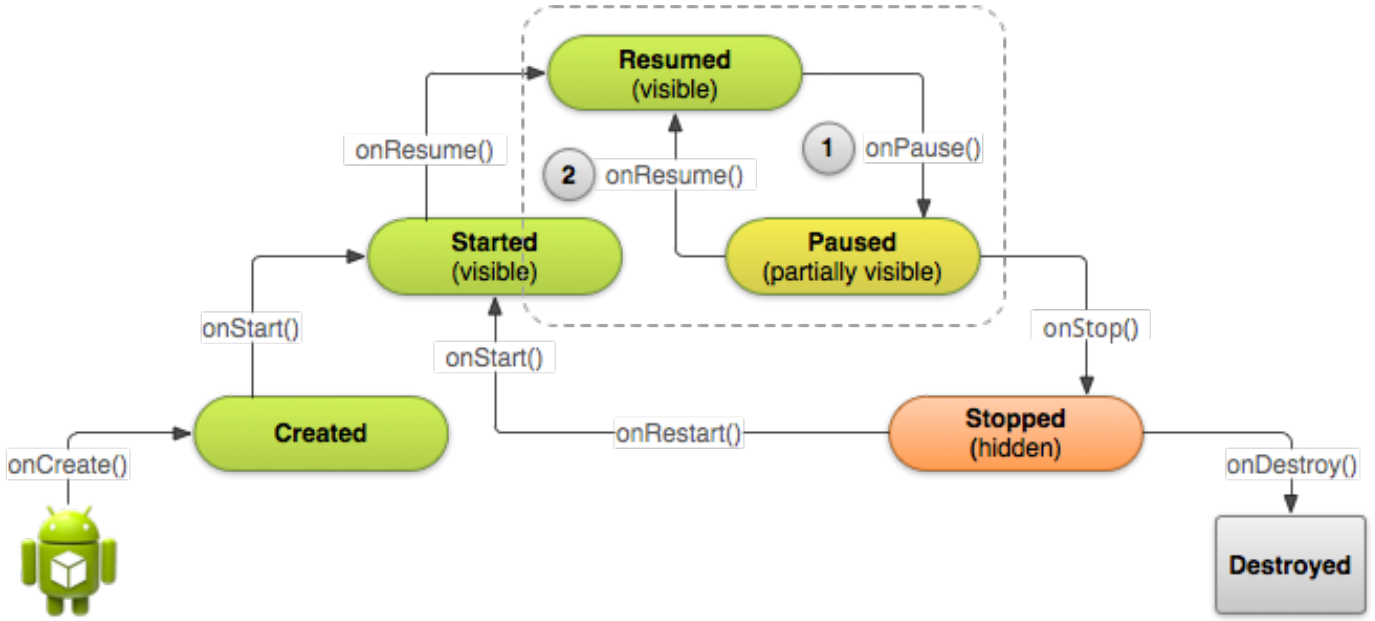
Activity Lifecycle

- Common to switch between different activities/screens.
  - Activities can create other activities (“Back stack”)
  - Navigation forward/back through activities is typically triggered by user actions



Interrupted Workflow

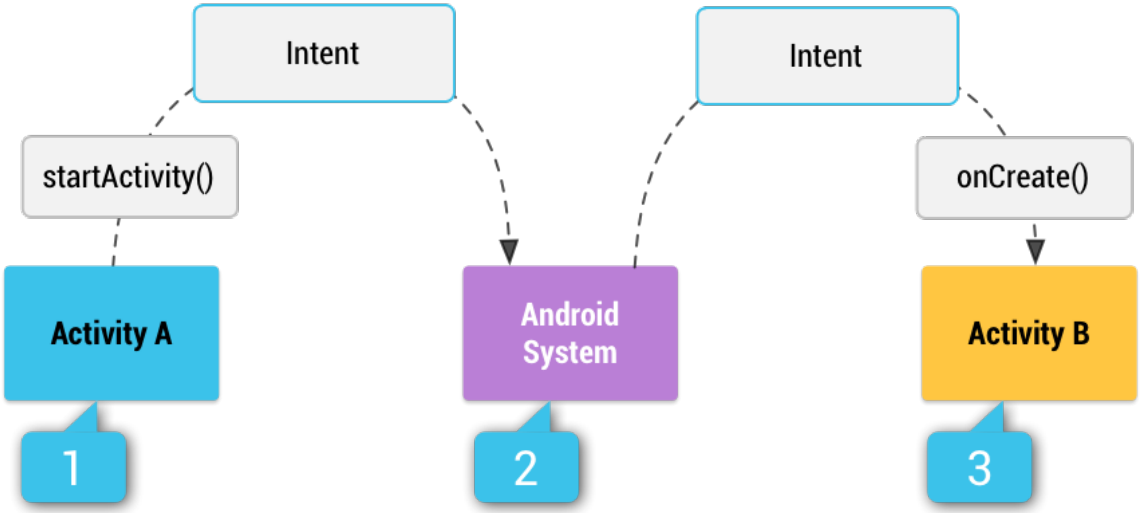
- Applications can stop at any time (i.e. user quits, OS kills it).
  - Each activity needs to manage its own state.
  - Activities have methods for saving and restoring state



http://developer.android.com/training/basics/activity-lifecycle/pausing.html

Intents

- We use **intents** to pass data between activities/screens.
  - Data structure holding an abstract description of an action.
  - Use Activity *startActivity()* method to launch with intent.
    - Explicit (named activity) vs. implicit (capabilities, e.g. camera)



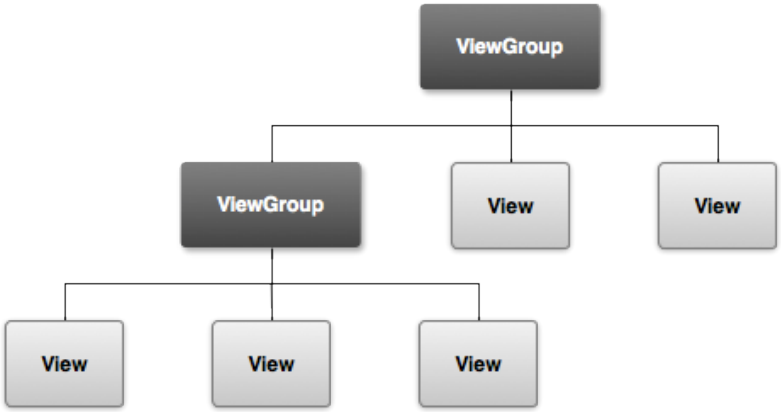
https://developer.android.com/guide/components/intents-filters.html

Events

- Android uses the Java event model, modified to support additional events.
  - **Event listener**: interface for specific type of event
  - **Event handler**: registered callback method to handle the event

Event Listener	Event Handler	Type of event
OnClickListener()	onClick()	Touch, click
OnLongClickListener()	onLongClick()	Press and hold
onTouchListener()	onTouch()	Generic touch events; can be used for touch_up, second_touch

- android.view.ViewGroup
  - Abstract container class
  - Includes layout functionality directly
  - Subclasses: FrameLayout, GridLayout, LinearLayout, RelativeLayout, Toolbar, …
- android.view.View
  - Base widget class (drawing and event handling)
  - Subclasses:
    - android.widget.Button
    - android.widget.ImageView
    - android.widget.ImageButton
    - android.widget.ProgressBar …



Layout

- Layout can be handled in one of two ways:
  1. Programmatically. You write code to instantiate ViewGroups, Views and bind them together (sim. to how you've done this in Java).
  2. Use XML to describe your layout. In-code, describe the screen elements (view groups and views) along with properties, and then tell your application to dynamically load it.

- Using XML is actually the preferred way to handle this.
  - Android Studio includes a GUI builder to make this easier!

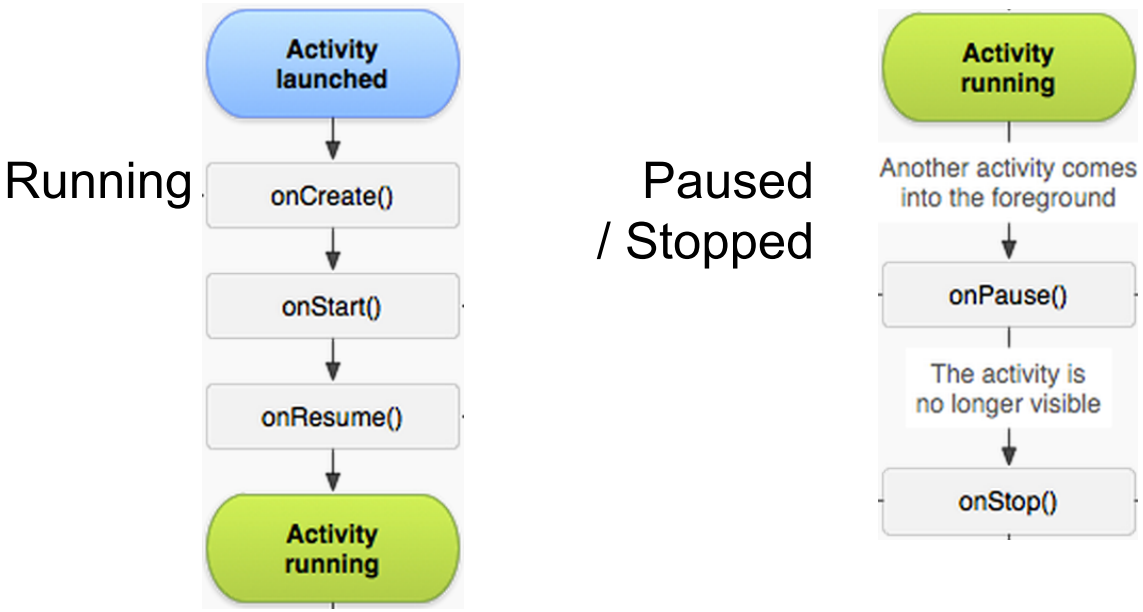
Recap: Building Applications

A typical application will include:

- Activities
  - MainActivity as your entry point
  - Possibly other activities (corresponding to multiple screens)
- Views
  - Screen layouts
  - ViewGroups containing Views (i.e. components)
- Intents
  - Only required if you need to communicate or pass data between screens

Demo

- The New-Project wizard generates “Hello World” starter code.
- Great demonstration of how the project fits together.
- Recall the application lifecycle. Event states map directly to methods.



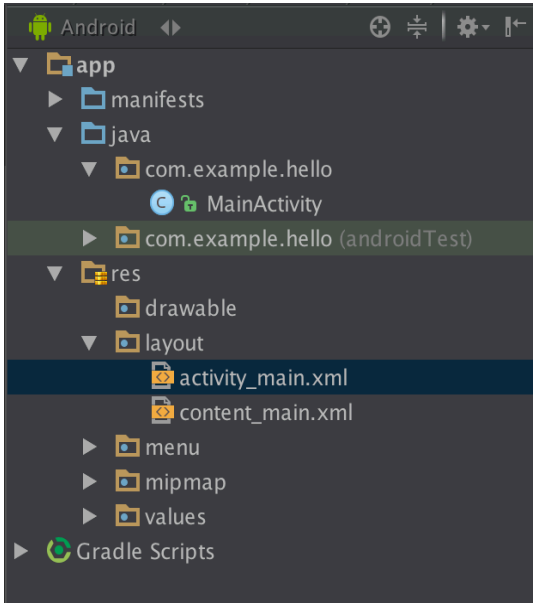
“Hello World”

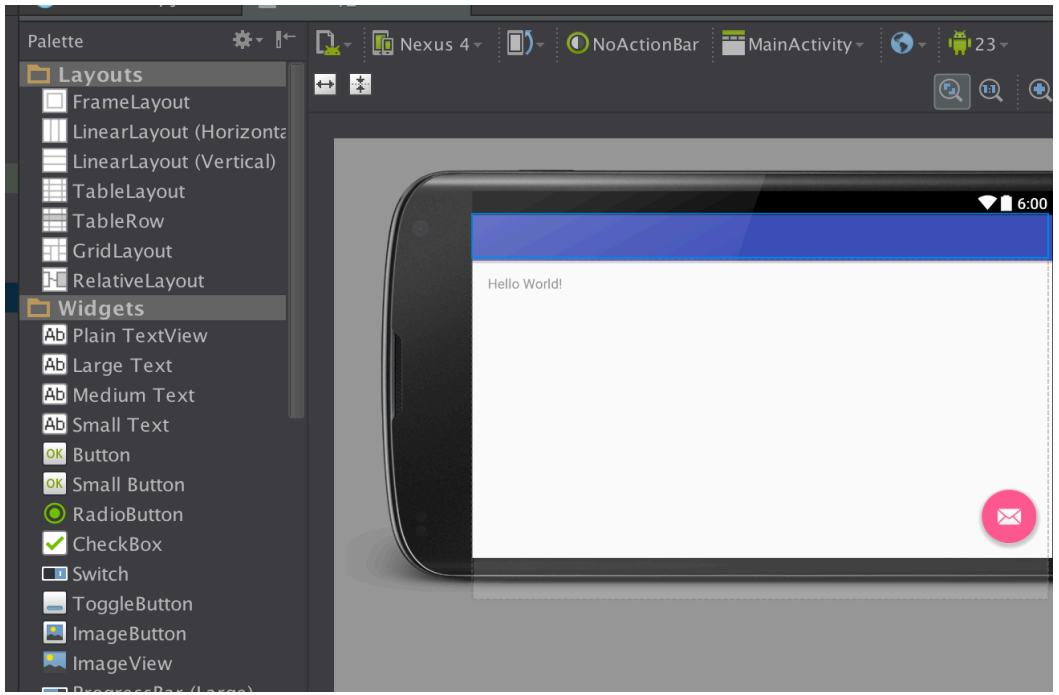
```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);  // points to resource/layout
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar); // default toolbar

        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
                        .setAction("Action", null).show();
            }
        });
    }
```

- OnCreate() - executed when the app runs
- SetContentView(R.layout.activity_main) - Uses layout from **res/layout/activity_main.xml**

"Hello World"

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);  // points to resource/layout
```





Resources

- Android Developer http://developer.android.com/
- Android Developer Channel @YouTube
  - http://www.youtube.com/user/androiddevelopers
- Android Open Source Project https://source.android.com
- Recommended (Optional!) Books
  - Phillips & Hardy, Android Programming: The Big Nerd Ranch Guide
  - Mednieks et al., Programming Android