ECE 1778:
Creative Applications for Mobile Devices

Lecture 4

January 30, 2013

Today

1. Logistics – Plan, Assignments
2. Recon Instruments Android Goggles
3. Some Notes on Programming Android
   - Life cycle
   - Sensors
   - Debugging
4. Proposal Discussions

Logistics

Assignments

- A2 and P2 were due yesterday
  - 2 people who are in groups did not hand these in (and didn't notify me)
- A3 and P3 are ready, and posted
  - But are **not due** for 2 weeks,
  - To give you time to work on your plans, due next week.
- A1 and P1 have been graded, with comments online
  - Generally well done; TAs have provided some specific feedback where warranted
  - See blackboard portal

Project Stages

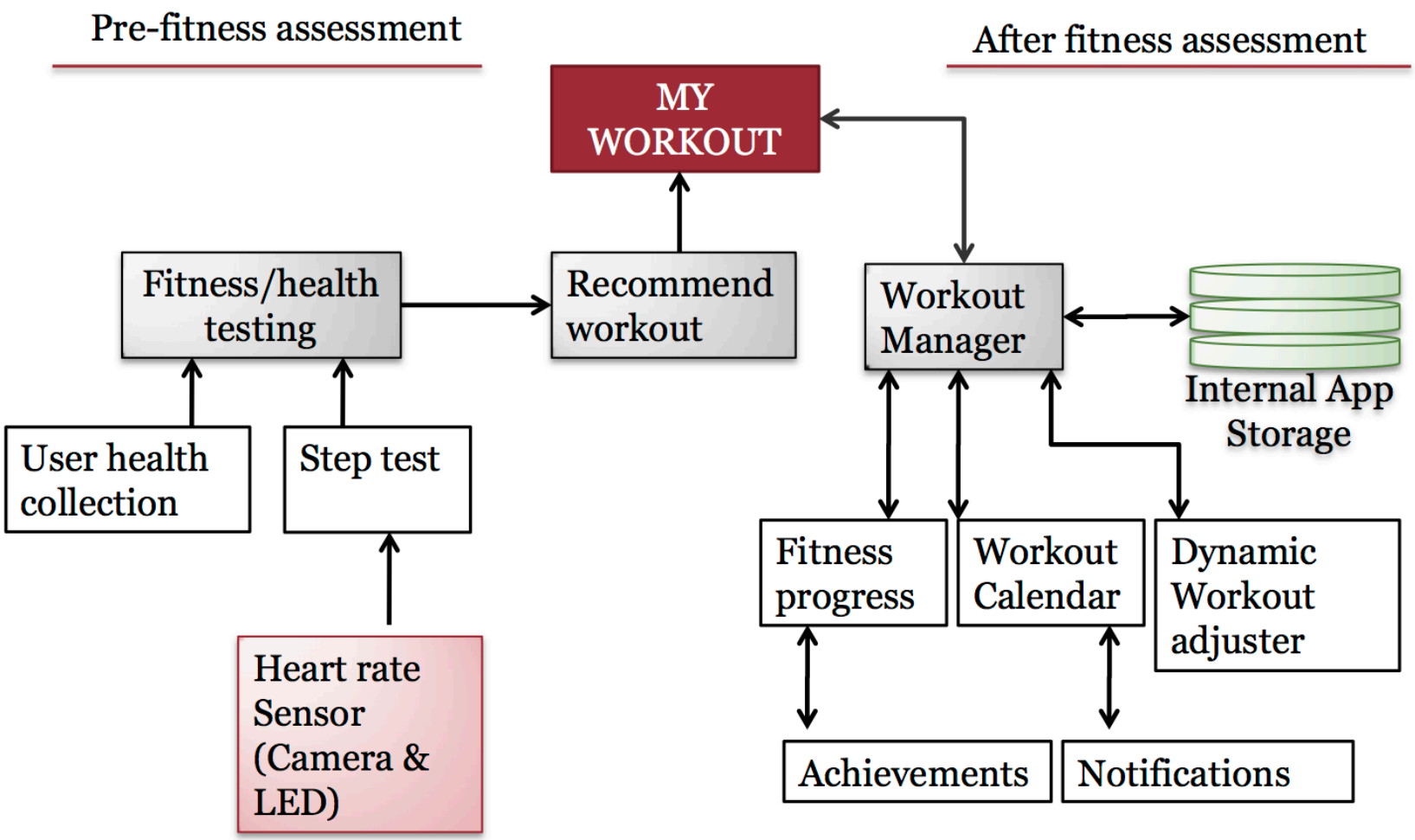
1. **Forming Groups**
   - All groups are formed, but some lack 2nd programmer; have 1
2. **One-Page Proposal**
   - I have responded to all sent yesterday; will do rest today
3. **Project Plan**
   - Due **Next week Feb 6th**
4. **Proposal & Plan Presentations**
   - February 11 & 13
   - **NOTE EXTRA LECTURE Monday Feb 11, 6-8pm, MP 137**
5. **Spiral 2 & Spiral 4 Presentations**
   - 2: March 6/13 4: March 20/27
6. **Final Presentations**
   - Weeks of April 3 & 10
7. **Final Report Due April 12th**

Plan Due Next Week: Feb 5 @ 6pm

1. Reprise Goal, make more precise
2. Rough design of what the user sees
   - Mock-ups of screens
   - https://gomockingbird.com
   - Any drawing package will do
3. Block Diagrams overview of planned code
   - Implies a top down, divide & conquer approach
   - With short prose description of each
   - Should relate to the screens given above
   - Example: see Figure 1 on page 4 of:

   http://www.eecg.utoronto.ca/~jayar/ece1778.2012/fitformula.pdf

Fit Formula's Final Block Diagram



Plan, continued

4. Statement of Risks/Issues
   - What roadblocks/issues/challenges do you foresee?
   - App-wise, programming-wise, hardware-wise, ethics-wise
5. What do you need to learn that you don't know
   - all members
6. **Important:** Appers
   - Submit a separate essay on how App relates to field of Apper, and **how the Apper will contribute to project**
   - 500 words

Plan Document

- Plan length: 1500 words max
  - Not including 500 word Apper essay (#6)
  - Should include pictures
  - Include word count, penalty for overage.

- Seeking clarity, not quantity of words
  - ‘Omit needless words’

- Submit PDF doc to Portal, look for ‘Assignment’ Plan
  - Due Tuesday February 5th at 6pm
  - Worth 10% of grade (includes presentation done following week)

Pepper Website

- We are using Pepper for Two Purposes:

1. As a class discussion board, for posting questions and answers; monitored by TAs & myself

2. As a workplace for interaction among group members
   - Every group has its own, private interaction space
   - Listed under the ‘home’ community on pepper
   - For posting information, videos, software, etc.
   - Can use as a record of work, and a holding space for work
   - Can use a medium of interaction
   - Not graded

Class Participation

Class Presentations & Participation

- A key part of what has happened in this course was the contribution students made to other's projects
- You will do many presentations in this class
  - Indeed, one side-effect of this project course is some real practice in giving high-quality, concise & clear communication
  - Most presentations will be 5 minutes in length
  - Must be geared so that most people in the class will understand
- Want everyone to come, listen & provide useful input
  - The grading scheme includes participation
  - Expectation that you'll listen and provide thoughtful feedback and suggestions to other's presentation, starting today

Grading Scheme

- Assignments: 16%
  - 4 assignments
- Project: 80%
  - Proposal 5%
  - Plan (incl presentation) 10%
  - Spiral 2 Presentation 10%
  - Spiral 4 Presentation 10%
  - Presentation/Demo 10%
  - Final Report 30%
- **Class Participation 9%**

Recon Instruments Android Goggles

Don’t Forget

- We have one pair of Recon’s instrumented ski goggles
  - Available for use in your project

- Ski Goggles with:
  - Head ‘down’ display
  - GPS
  - Full android processor with SDK
  - 9-axis accelerometer/gyroscope/compass
  - Bluetooth connection to mobile phone
  - Wrist-based selection tool for input

Features

SPEED

Calculated by GPS combined with barometric pressure data, accurately see how fast you're going as you carve the slopes.

JUMP ANALYTICS

Know your distance, height, and airtime either when pulling tricks in the park, or when hitting backcountry kickers.

VERTICAL

From the peak to the lodge, track your vertical feet by run, by day and over the course of the season.

ALTITUDE

The onboard altimeter tracks your altitude to any mountain summit. Follow your elevation up to the peak, and down to the bottom.

NAVIGATION

Find your way around resorts easily, get to know your favourite runs by name, or track down points of interest easily.

BUDDY TRACKING

Ideal for when you get separated from your group on the slopes. Now you can see where they are, and safely reunite.

SMARTPHONE CONNECTIVITY

Pair with your Smartphone to view incoming calls and read text messages immediately, as you receive them.

MUSIC

Playlist mode puts your music at your fingertips. Be in full control of what you hear as you ski or board.

- See http://www.youtube.com/watch?v=2-9UAJ4v_8M
- and http://www.youtube.com/watch?v=XpY2Oh4stM0
- and http://www.youtube.com/watch?v=VqjGsYf4upI

SDK

- Go to http://developers.reconinstruments.com
  - For tutorials, design guide and downloads

Android Essentials

- Life Cycle
- Sensors
- Debugging
- Pop-Up Messages – Toast & Alerts

Android Activity ‘Life Cycle’

Android Application Life Cycle

- Recall: Activities are screens that the user sees, and associated process
- Android manages these Activities as a **stack**.
- When a new activity is started, it is placed on the top of the stack and becomes the running activity

- The previous activity always remains below it in the stack,
  - and will not come to the foreground again until the new activity exits.

Important to Pay Attention to ‘LifeCycle’

- To ensure app behaves well in several ways, including:

1. Does not crash if the user receives a phone call or switches to another app
2. Does not consume valuable system resources when the user is not actively using your app
3. Does not lose the user's progress if they leave your app and return to it at a later time
4. Does not crash or lose the user's progress when the screen rotates between landscape and portrait orientation.

An Activity Can Be in 1 of 4 ‘States’

State 1: Active/Running

- Activity in the foreground of the screen (at the top of the stack)
- Has ‘focus’, meaning user interactions go to it.

State 2: Paused

- activity has lost focus but is still visible
- a new smaller or transparent activity has focus on top of the activity)
- A paused activity is completely alive (it maintains all state and member information and remains attached to the window manager), but can be killed by the system in extreme low memory situations.

Activity States 3 and 4

State 3: Stopped

- activity is completely obscured by another activity
- retains all state and member information
- no longer visible to the user so its window is hidden
- it will often be killed by the system when memory is needed elsewhere.

State 4: Destroyed

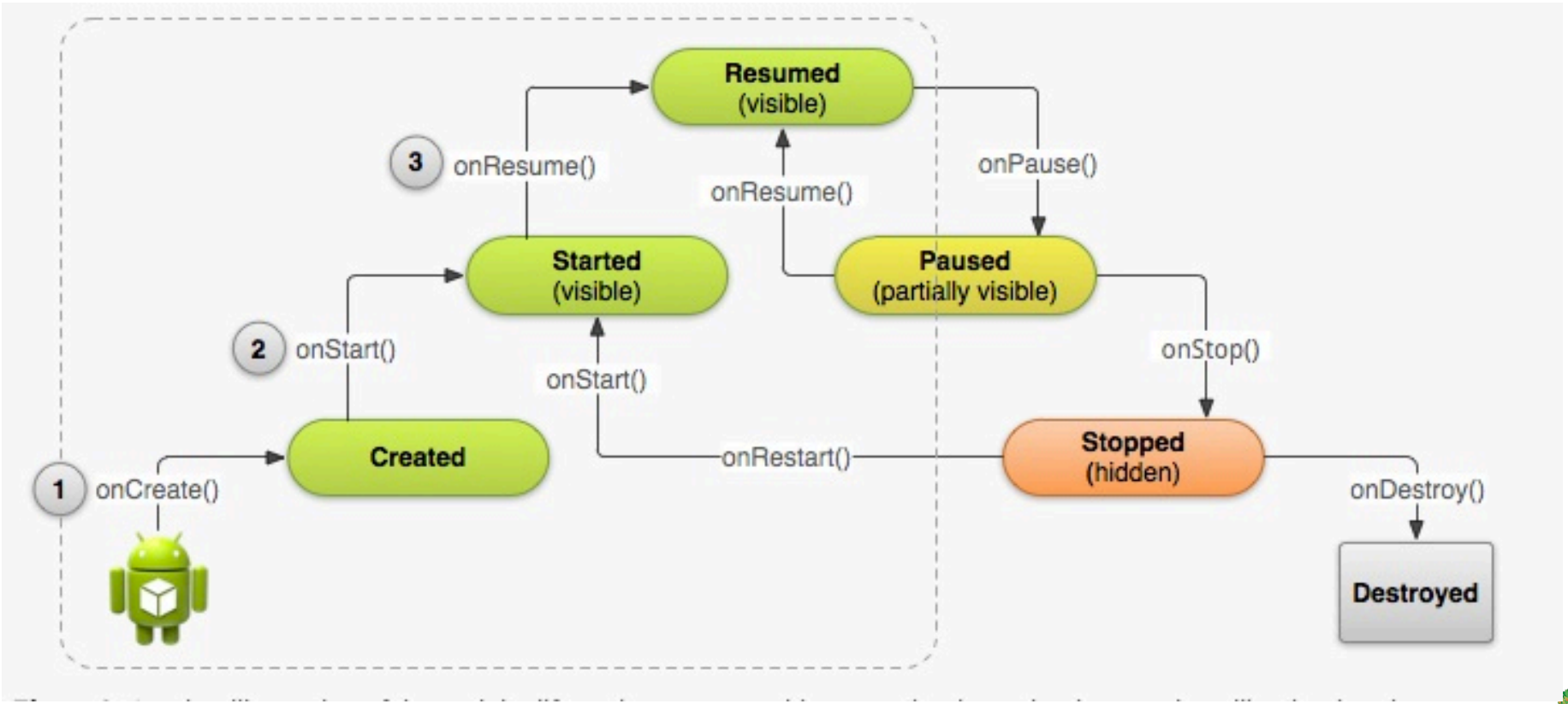
- If an activity is paused or stopped, the system can drop the activity from memory by either asking it to finish, **or simply killing its process.**
- When displayed again to the user, it must be completely restarted and restored to its previous state.

Android Talking to Your App

- The Android operating system asks (or tells) your app to go into those different states by invoking methods associated with your Activity

Methods Called By Android to Change States

- Diagram shows states and methods called to change state
  - Colours: the states
  - MethodName(): methods called by Android OS



Three Key States

- Activity can be in 1 of 3 states for long period of time:

1. Resumed
   - In this state, the activity is in the foreground and the user can interact with it. (Also sometimes referred to as the "running" state.)
2. Paused
   - In this state, the activity is partially obscured by another activity—the other activity that's in the foreground is semi-transparent or doesn't cover the entire screen. The paused activity does not receive user input and cannot execute any code.
3. Stopped
   - In this state, the activity is completely hidden and not visible to the user; it is considered to be in the background. While stopped, the activity instance and all its state information such as member variables is retained, but it cannot execute any code.

State Management

- The other states (Created and Started) are transient and the system quickly moves from them to the next state by calling the next lifecycle callback method. That is, after the system calls onCreate(), it quickly calls onStart(), which is quickly followed by onResume().

- Depending on the complexity of your activity, you probably don't need to implement all the lifecycle methods.
- However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.

References

1. The Android Documentation:

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

2. Murphy, Busy Coder's Android, Pages 275-297
   - "Handling Activity Lifecyle Events"

- Once your project gets going, it is really important to read through this and understand it
  - Previous years' students pointed out that this was the key thing they had not understood in Android, that caused the most problems

The Key 'LifeCycle' Methods

OnCreate()

- Familiar with already – brings the activity to life

OnPause()

- Another Activity has gained the 'focus'
- Should stop any background threads, release large resources (such as a camera)
- **No guarantee that OnDestroy() will be called**, so best to save **all** state here

OnResume()

- Called as activity starts, **or** is restarted from a pause
- Can recall state from file, refresh the User Interface – see example

Relating to Sensors

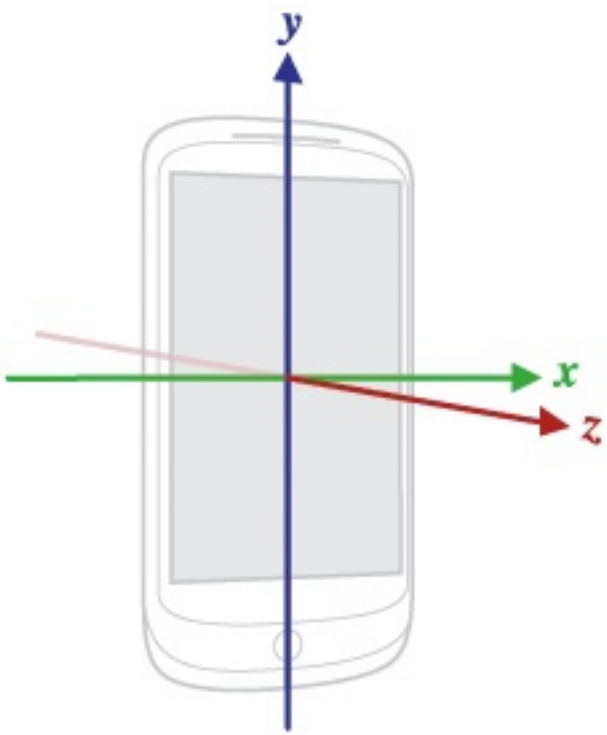
For Assignment P3 (due in 2 weeks)

And general sensor stuff

The Accelerometer

The Accelerometer

- The Accelerometer in phones is quite an exciting sensor!
- It can be used to feel motion in three dimensions
  - It samples the acceleration at least 100 times/second
  - But not reliably!
  - It is very sensitive
- It measures acceleration in **m/s^2**



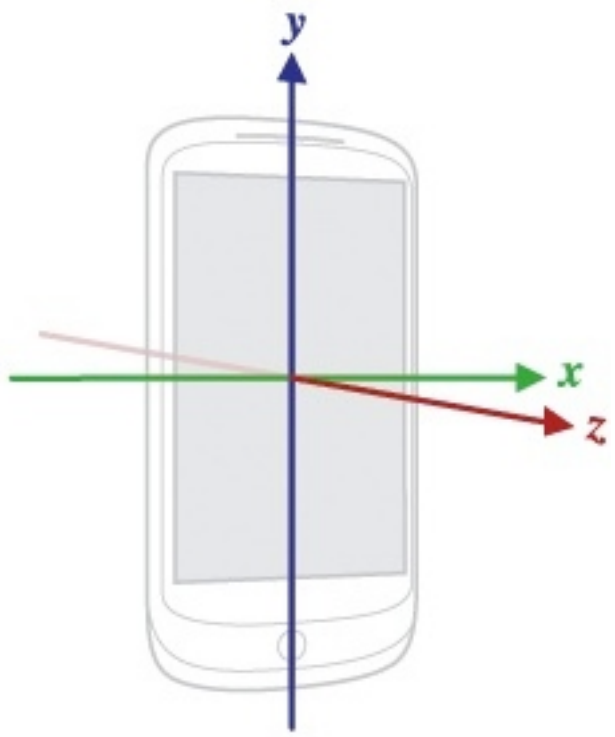
Gravity

- **Recall**: acceleration due to gravity is 9.8 m/s^2
- When the phone isn't moving, one or more of the axes will 'feel' this acceleration due to gravity or part of it.
- A phone that is falling, will feel 0 acceleration on all three axes

- Online Android documentation describes how to isolate out both gravity (to figure out which axes it is on) and motion aside from gravity:
  - http://developer.android.com/reference/android/hardware/Sensor.html

The Accelerometer Coordinate Space

- Coordinate-system is defined relative to the screen of the phone in its default orientation.
  - axes are **not** swapped when the device's screen orientation changes.
- X axis is horizontal & points right
- Y axis is vertical & points up
- Z axis points towards the outside of the front face of the screen.
  - coordinates behind the screen have negative Z values.



Using the Accelerometer

- Bones of an application to
  - Read the accelerometers every time they change
  - Output the raw values in each dimension
  - Will include gravity

To Access a Sensor

Three key classes:

1. SensorManager
2. Sensor
3. SensorEvent

- Need to create a SensorManager class

SensorManager

- Is a class that lets you figure out what sensors are on the device:
  - produces a list of all the sensors available,
  - your program must need to check and see if the one you want is actually on the device:
- First, create the general sensor manager

```
myManager = (SensorManager) getSystemService
        (Context.SENSOR_SERVICE);
```

- Then, ask if the sensor you want is on the phone, e.g. the Accelerometer:

```
sensors = myManager.getSensorList
        (Sensor.TYPE_ACCELEROMETER);
```

SensorEvent Class

- ‘event’ is signaled when there is a new reading
- SensorEvent class contains the reading of the sensor, including:
  1. Accuracy of the measurement
  2. Sensor that generated the event
  3. Timestamp
     - The time in nanoseconds at which the event happened
     - Can check the spacing, in time, of your readings!
  4. Values of the reading itself
     - e.g the three values of the acceleration along each axis

Declarations & usual

```
public class readaccel2 extends Activity {
    private TextView accText;
    private SensorManager myManager;
    private List<Sensor> sensors;
    private Sensor accSensor;
@Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);
       accText = (TextView)findViewById(R.id.accText);
```

Set-up Sensor

```
 // Set Sensor + Manager
myManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);


sensors =
myManager.getSensorList(Sensor.TYPE_ACCELEROMETER);
      if(sensors.size() > 0)
       {
          accSensor = sensors.get(0);
        }
   }
```

Listening, Reading, Output

```
private final SensorEventListener mySensorListener = new
SensorEventListener()  {
    public void onSensorChanged(SensorEvent event)  {
        float x = event.values[0];
        float y = event.values[1];
        float z = event.values[2];
        String output = String
         .format(”x: %f / y: %f / z: %f", x, y, z);
         accText.setText(output);  }
public void onAccuracyChanged(Sensor sensor, int
accuracy) {}};
```

Registering the Listener

```
@Override
    protected void onResume()
    {
     super.onResume();
     myManager.registerListener(mySensorListener,
     accSensor, SensorManager.SENSOR_DELAY_GAME);
    }
```

- Last parameter sets the frequency of updates

OnPause – Turn off accelerometer

```
@Override
    protected void onPause()
    {
     myManager.unregisterListener(mySensorListener);
     super.onStop();
    }
}
```

- i.e. unregister the listener

To Use the Compass (Magnetometer)

- Change:

```
sensors = myManager.getSensorList
                    (Sensor.TYPE_ACCELEROMETER);
```

- To:

```
sensors = myManager.getSensorList
                    (Sensor. TYPE_MAGNETIC_FIELD);
```

The Other Sensors

There are quite a few Sensors!

- Some are just different calculations with same sensor

1. Accelerometer:
   - Sensor.TYPE_ACCELEROMETER
2. Gravity
   - The accelerometer with non-gravity acceleration filtered out?
   - Sensor.TYPE_GRAVITY
   - Only available on Android 2.3 and later
3. Linear Acceleration
   - The accelerometer with gravity removed (filtered)
   - Sensor.TYPE_LINEAR_ACCELERATION
   - Only available on Android 2.3 and later

Compass/Magnetic field

- Sensor.TYPE_MAGNETIC_FIELD:
- Measures the magnetic field in three dimensions,
  - Measured in micro Tesla
- So, just Change:

```
sensors = myManager.getSensorList
                    (Sensor.TYPE_ACCELEROMETER);
```

- To:

```
sensors = myManager.getSensorList
                    (Sensor. TYPE_MAGNETIC_FIELD)
```

Gyroscope

- Sensor.TYPE_GYROSCOPE:
- Gives pitch, roll and yaw in radians/second
  - Measures motion more directly

Light Sensor

- Used for measuring ambient light to set screen brightness
- Measures the light, in Lux, coarsely
  - Seem like roughly 8 different values, maybe, when I tried it on the Nexus One
- Sensor. TYPE_LIGHT
- Available on Nexus S
- DEMO

Proximity Sensor

- Used for measuring how close the phone is to person's ear
- To turn off the touch screen
- Usually binary: **close** or **far**
- Sensor.TYPE_PROXIMITY

- DEMO

Useful Method – sensor maximum

- MaxRange = accSensor.getMaximumRange();
- Tells the largest value a sensor can deliver on a device

Using the Debugger in Eclipse/Android

Debugging

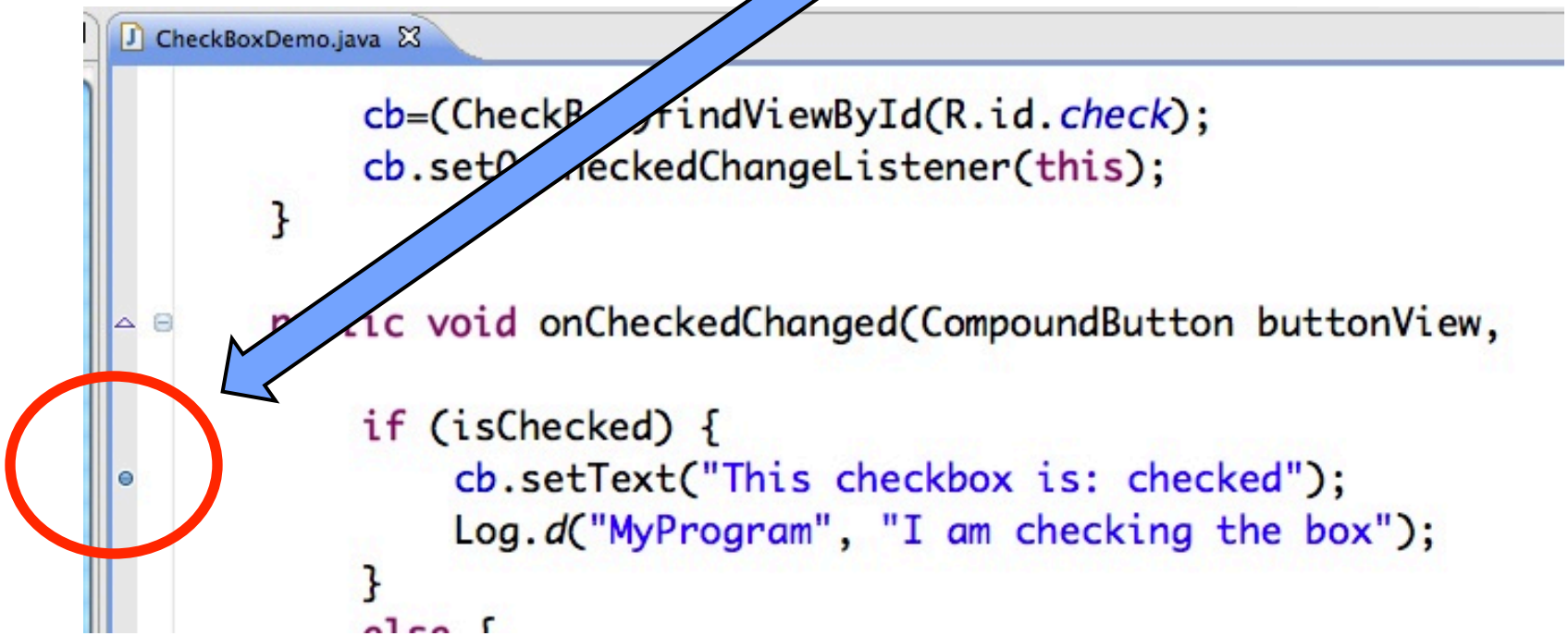
- Simple debug: use Log.d(TAG, “String”);
- Better: use the debugger in Eclipse/Android

- First, make sure your phone or emulator is set to enable ‘USB debugging’
- From the home screen:
  - Settings->Applications->Development
  - Check USB Debugging (have to do that 7-times magic incantation)

Set a Breakpoint

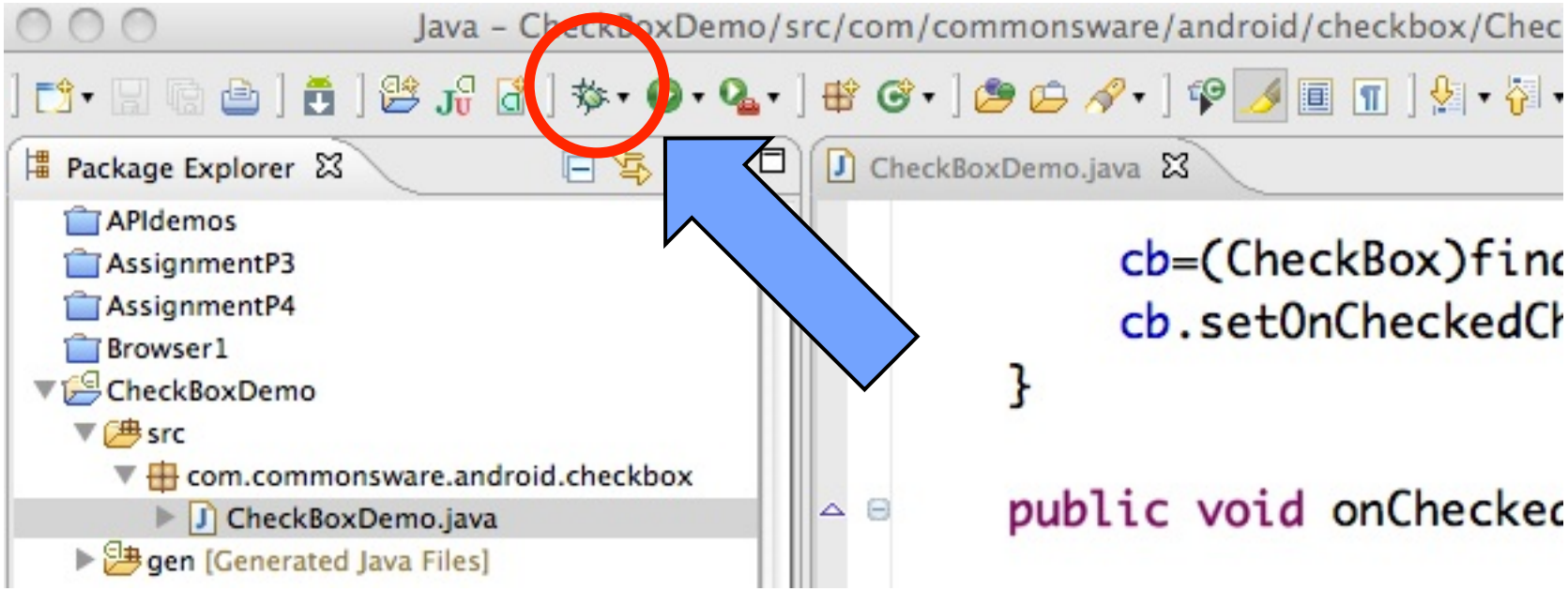
- Go to your source file,
- right click in the left-most column
- Select ‘Toggle Break Point’
  - Will set a breakpoint at that source code line (little blue dot)

Right Click over here



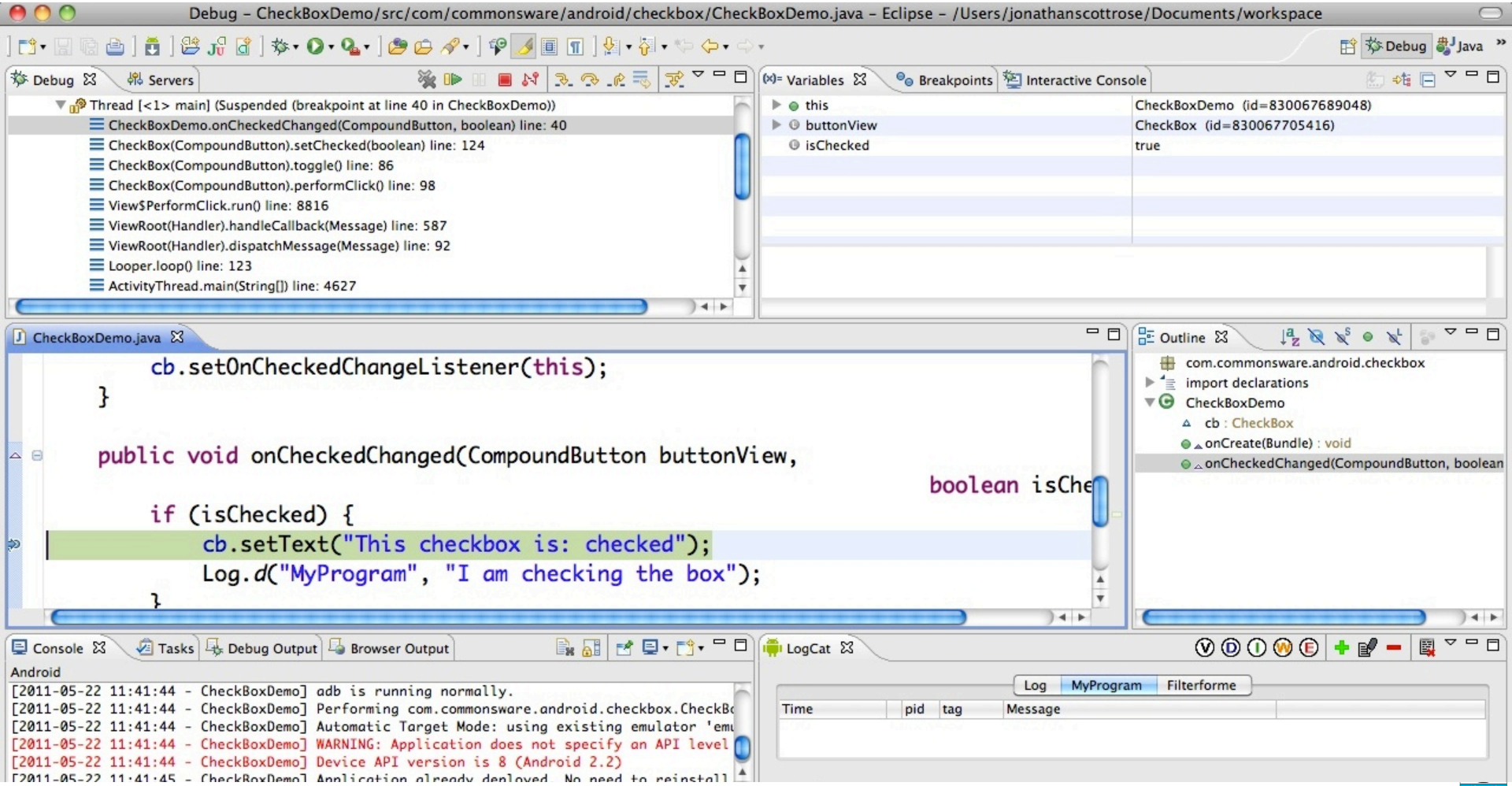
To Run the Debugger

- Click the ‘green bug’ to the left of to the ‘play’ button



Once the Breakpoint is Hit

- The whole Debugger perspective shows up:



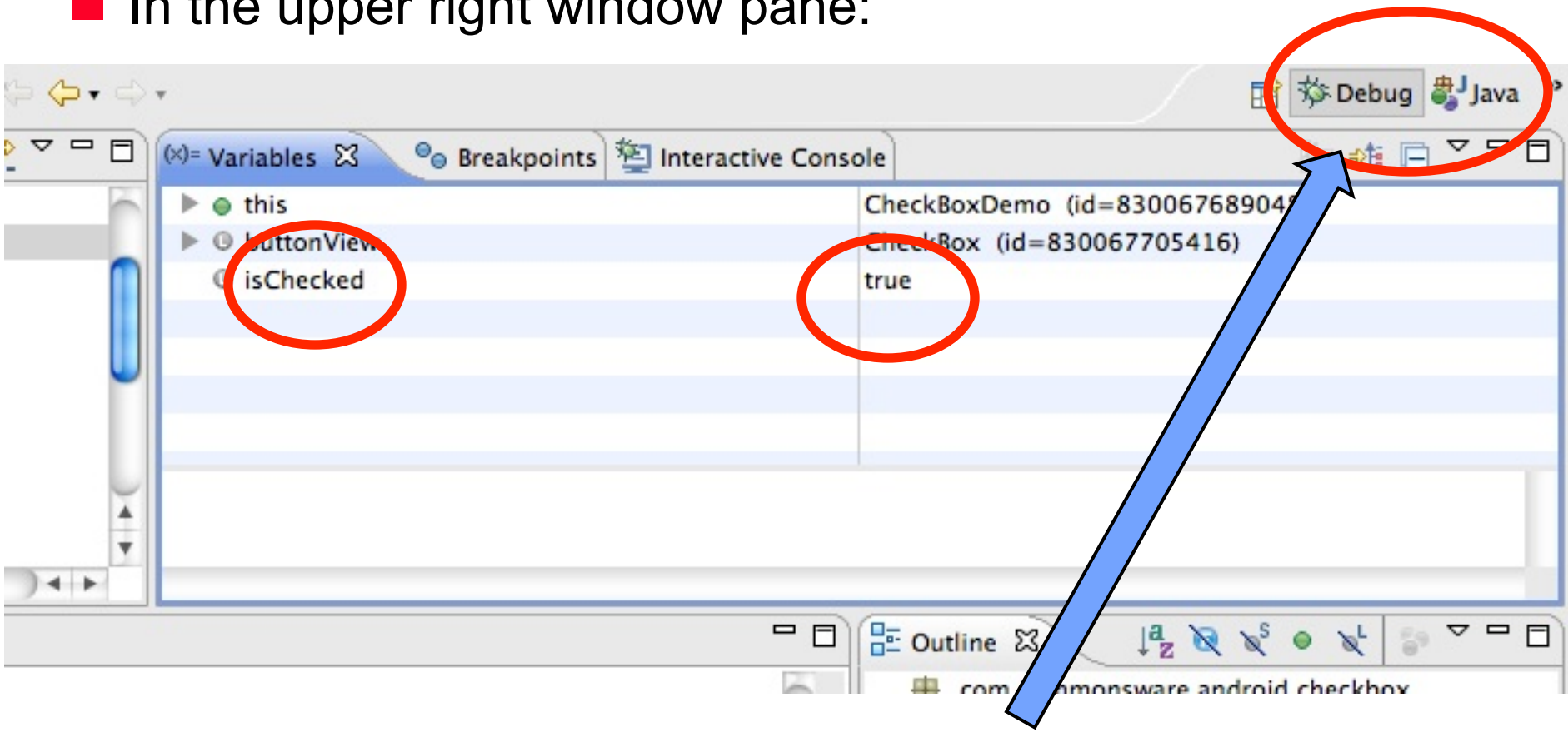
Can Single Step over, Step In, Run

- See this graphic near the top:

1. Continue running from Breakpoint
2. Step in to a function
3. Step over a line (single step)

Can View Value of Variables

- In the upper right window pane:



- To switch between regular and debug perspectives

API Demos!

Google's API Demos

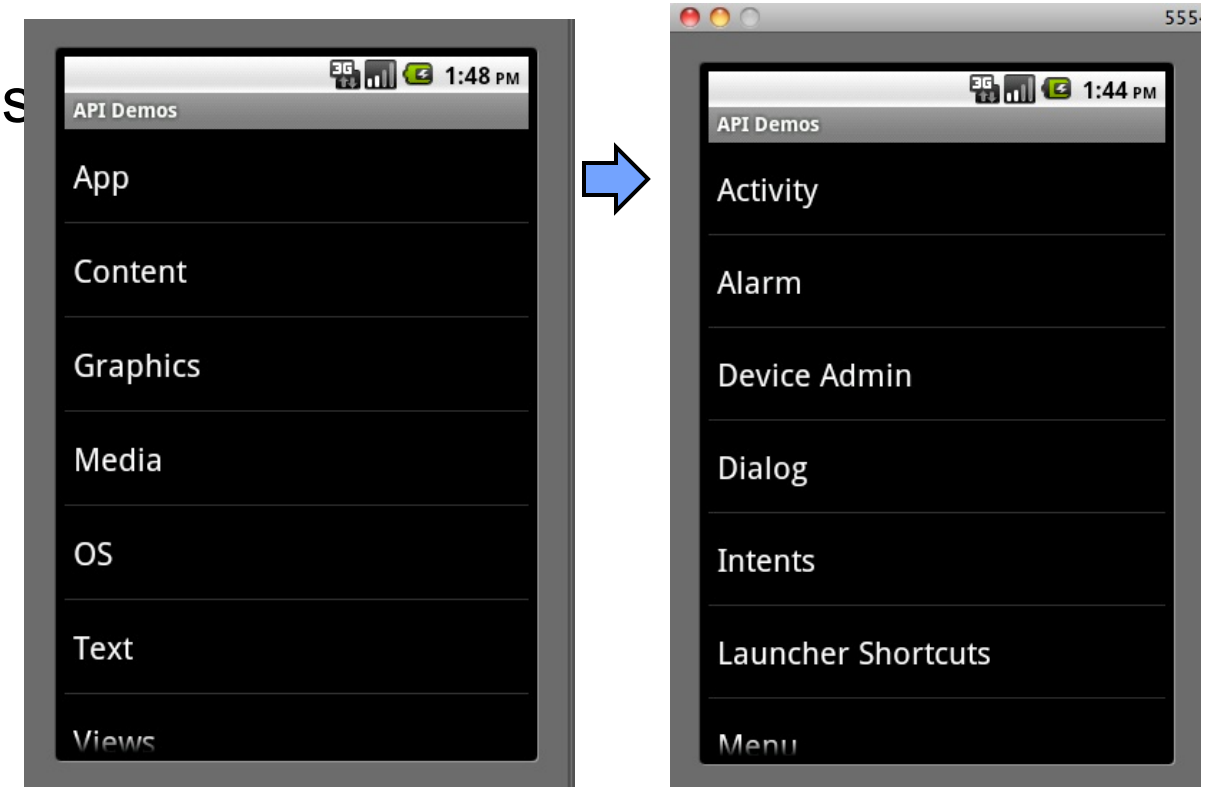
- Are great for learning everything Android
- Every feature of the phone is used, in a simple example

How:

- Be sure to have downloaded the 'Samples for SDK' from the SDK Manager available in Eclipse
- To see them, create a new project in Eclipse (not an Android Project)
  - Under 'Android Project'
  - Select 'Android Project from Existing Code'
- Load in the project from this directory:
  - adt-bundle/sdk/samples/android-N/APIdemos

Then, Experiment and Learn!

- Each feature – activity, camera, sensor, is shown as a simple example, along with code
- Lots!
- Demo



Pop-up Messages

Pop-Up Messages

- Are really handy for conveying an alert or information to the user.

- Two kinds in Android:
  1. **Toasts**
  2. **Alerts**

Toasts

- A temporary message that appears and then disappears after a fixed amount of time
  - e.g. battery low warning
  - Someone signed in to Skype
  - Won lottery
- Purpose is to be un-obtrusive
  - Doesn't take 'focus' away from your activity

Toast Code – Easy!

- Create and show, all in one:

Toast.makeText(this,
"Eureka, you win!",
Toast.LENGTH_LONG).**show**();

Three fields:

1. 'this' is the current class context
   - Can also use **getApplicationContext**() instead
2. The text to be shown
3. Length of time to show
   - LENGTH_LONG or LENGTH_SHORT constants

Alerts

- A specific message that changes focus
- User must respond by clicking

- Three parts:
  - Message title
  - Actual message
  - Up to three response buttons:
    - Positive
    - Neutral
    - Negative
- Use OnClick to respond to buttons

Alert Code – using Toast to respond

```
new AlertDialog.Builder(this)
        .setTitle("MessageDemo")
        .setMessage("How are you feeling?")
        .setPositiveButton("I'm Positive",

 new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dlg, int sumthin) {
    Toast.makeText(MessageDemo.this, "Eureka, you win!",
                    Toast.LENGTH_LONG).show();
        }
})
```

Construction of the part of the Alert

A Toast in response to the positive click

Neutral Button in Alert

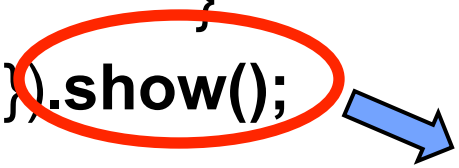
- A cascade of method calls

```
.setNeutralButton("Either Way",
    new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dlg, int sumthin) {
    Toast.makeText(MessageDemo.this, "Neutral Feeling",
                Toast.LENGTH_SHORT).show();
        }
})
```

Neutral Button in Alert

- Another cascade

```
.setNegativeButton("Either Way",
    new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dlg, int sumthin) {
   Toast.makeText(MessageDemo.this, ”I’m feeling bad",
                Toast.LENGTH_SHORT).show();
        }
}).show();
```



This shows the Alert

DEMO: MessageAlert