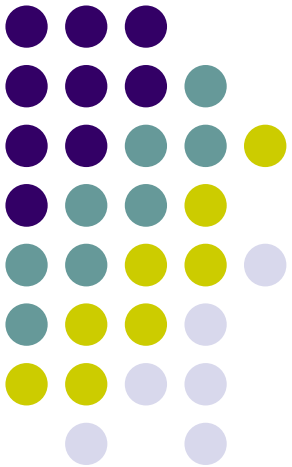
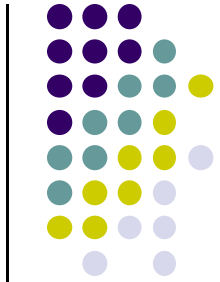
CS 403X Mobile and Ubiquitous Computing

Lecture 5: Web Services, Broadcast Receivers, Tracking Location, SQLite Databases

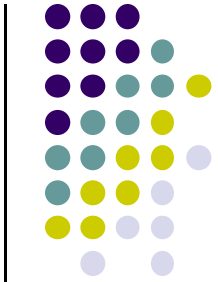
Emmanuel Agu





Web Services

What are Web Services?

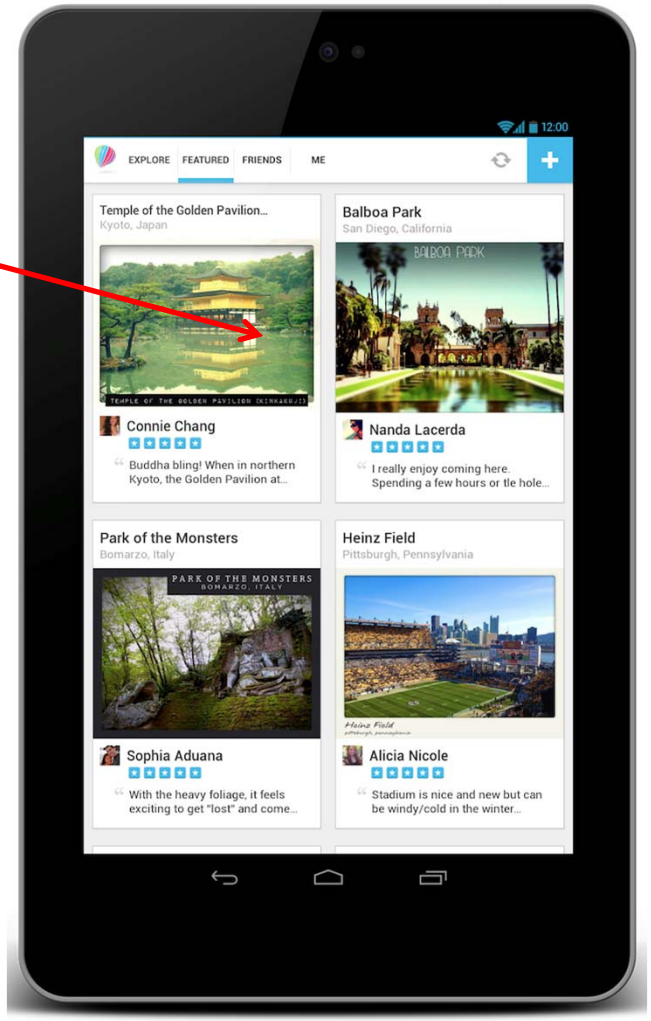


- Means to call a remote method or operation that's not tied to a specific platform or vendor and get a result."
  - Android in Action 3rd edition

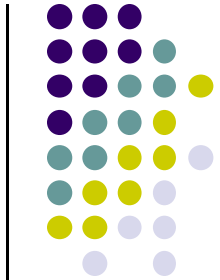
- E.g. Server may be running Linux, client running Windows

What Does that mean?

- Embed in your app third party:
  - **Pictures** from Flickr
  - **Videos** from Youtube
  - **Goods** sold on Amazon
  - **Maps** from Google
  - … etc



Web Services Sources

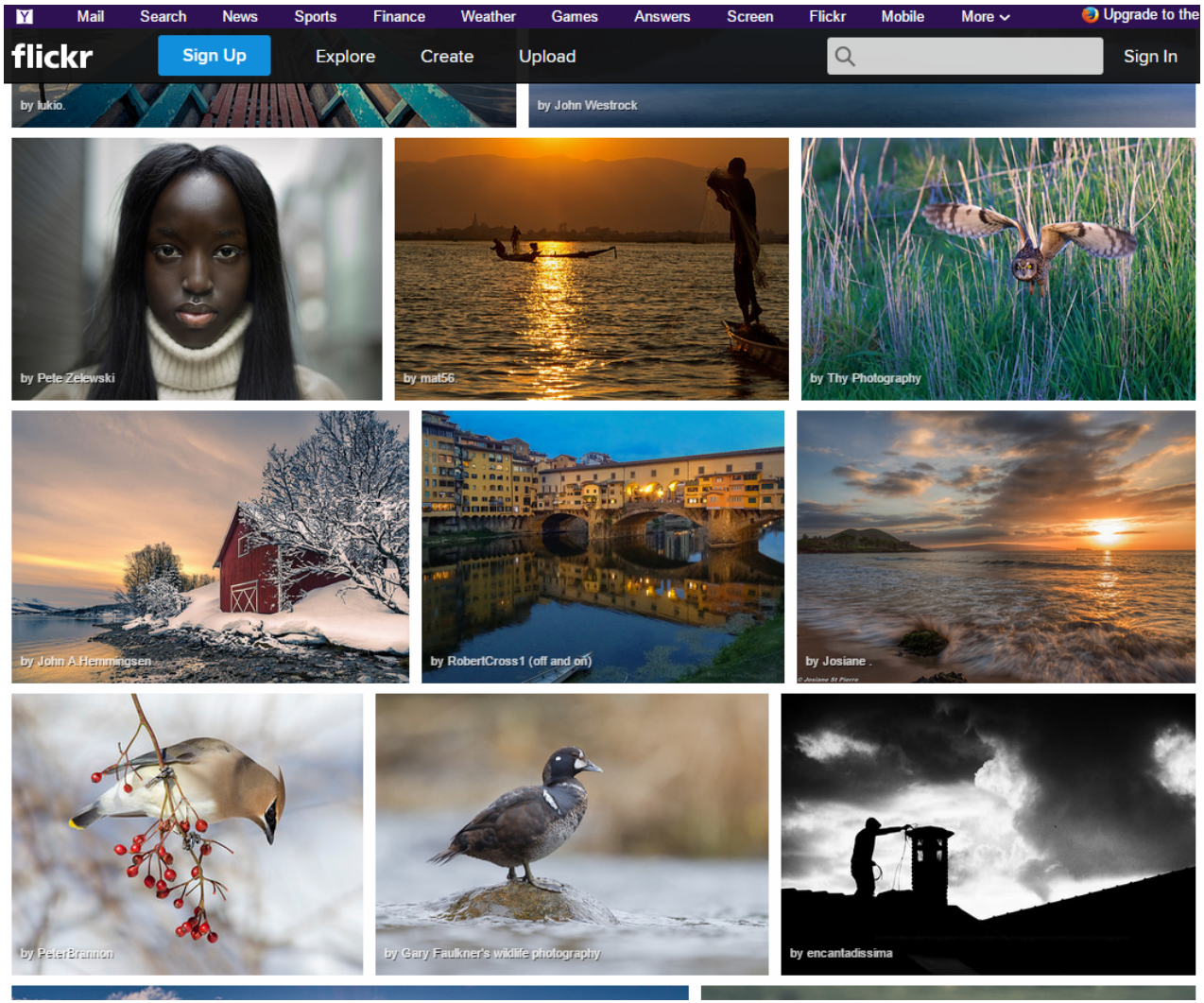


- Free third party content to enrich your android apps
- http://www.programmableweb.com/category/all/apis

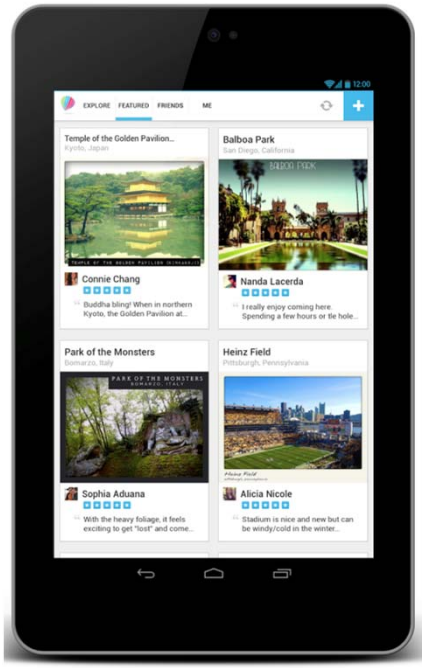
API	Description	Category
Google Maps	Mapping services	Mapping
Twitter	Microblogging service	Social
YouTube	Video sharing and search	Video
Flickr	Photo sharing service	Photos
Amazon eCommerce	Online retailer	Shopping
Facebook	Social networking service	Social
Twilio	Telephony service	Telephony
eBay	eBay Search service	Search
Last.fm	Online radio service	Music
Google Search	Search services	Search
Microsoft Bing Maps	Mapping services	Mapping
Twilio SMS	SMS messaging service	Messaging
del.icio.us	Social bookmarking	Bookmarks
Yahoo Search	Search services	Search

Flickr: Picture Sharing Website

- Website where people upload, share their pictures
- Over 5 billion free pictures!
- Pictures retrievable through web service
- Can access from Android app



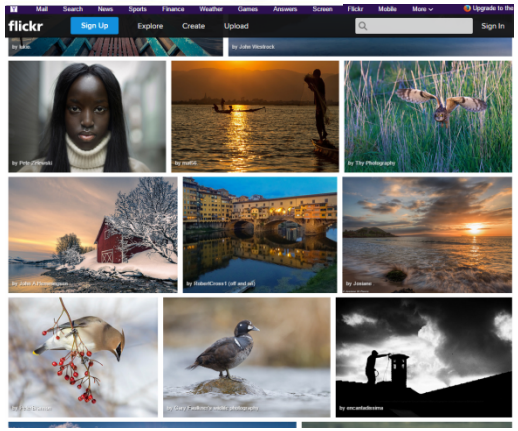
How to Embed Flickr Pictures in your App?



Your Android App

Communication using
Flickr API methods

Flickr Server



Flickr API

- http://www.flickr.com/services/api/

The App Garden

Create an App | **API Documentation** | Feeds | What is the App Garden?

The Flickr API is available for non-commercial use by outside developers. Commercial use is possible by prior arrangement.

Read these first:

- Developer Guide
- Overview
- Encoding
- User Authentication

- Dates
- Tags
- URLs
- Buddyicons

- Flickr APIs Terms of Use

Various methods for accessing pictures

API Methods

activity

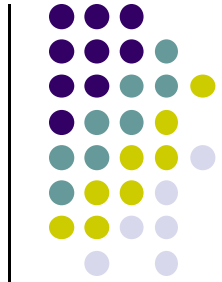
- flickr.activity.userComments
- flickr.activity.userPhotos

auth

- flickr.auth.checkToken
- flickr.auth.getFrob
- flickr.auth.getFullToken
- flickr.auth.getToken

auth.oauth

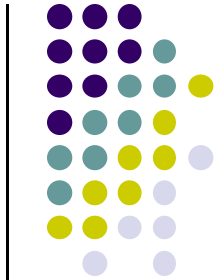
Flickr API Methods



- Create url using
  - API method name
  - Method parameters,
  - + API key
- 3 request formats
  - REST
  - XML-RPC
  - SOAP
- Many Response Formats (XML, JSON, etc

flickr.interestingness.getList

Returns the list of interesting photos for the most recent day or a user-specified date.



Authentication

This method does not require authentication.

Arguments

api_key (Required)
Your API application key. See here for more details.

date (Optional)
A specific date, formatted as YYYY-MM-DD, to return interesting photos for.

extras (Optional)
A comma-delimited list of extra information to fetch for each returned record. Currently supported fields are: `description`, `license`, `date_upload`, `date_taken`, `owner_name`, `icon_server`, `original_format`, `last_update`, `geo`, `tags`, `machine_tags`, `o_dims`, `views`, `media`, `path_alias`, `url_sq`, `url_t`, `url_s`, `url_q`, `url_m`, `url_n`, `url_z`, `url_c`, `url_l`, `url_o`

per_page (Optional)
Number of photos to return per page. If this argument is omitted, it defaults to 100. The maximum allowed value is 500.

page (Optional)
The page of results to return. If this argument is omitted, it defaults to 1.

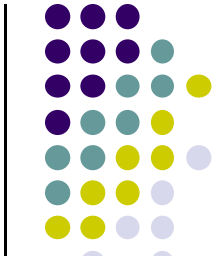
Example Response

Sample Flickr API Method

This method returns the standard photo list xml:

```
<photos page="2" pages="89" perpage="10" total="881">
        <photo id="2636" owner="47058503995@N01"
               secret="a123456" server="2" title="test_04"
               ispublic="1" isfriend="0" isfamily="0" />
        <photo id="2635" owner="47058503995@N01"
               secret="b123456" server="2" title="test_03"
               ispublic="0" isfriend="1" isfamily="1" />
        <photo id="2633" owner="47058503995@N01"
               secret="c123456" server="2" title="test_01"
               ispublic="1" isfriend="0" isfamily="0" />
        <photo id="2610" owner="12037949754@N01"
               secret="d123456" server="2" title="00_tall"
               ispublic="1" isfriend="0" isfamily="0" />
</photos>
```

Flickr API Methods



flickr.photos.search

Return a list of photos matching some criteria. Only photos visible to the calling user will be returned. To return private or semi-private photos, the caller must be authenticated with 'read' permissions, and have permission to view the photos. Unauthenticated calls will only return public photos.

Authentication

This method does not require authentication.

Arguments

`api_key` (Required)

Your API application key. See here for more details.

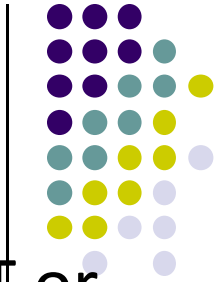
`user_id` (Optional)

The NSID of the user who's photo to search. If this parameter isn't passed then everybody's public photos will be searched. A value of "me" will search against the calling user's photos for authenticated calls.

`tags` (Optional)

A comma-delimited list of tags. Photos with one or more of the tags listed will be returned. You can exclude results that match a term by prepending it with a - character.

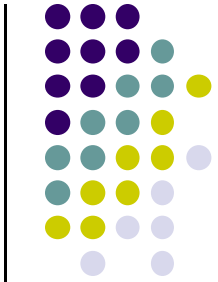
Flickr REST Request Format



- REST is simplest request format: a simple HTTP GET or POST
- E.g. Construct Flickr search request + parameters into HTTP URL

- http://api.flickr.com/services/rest/
  ?method=flickr.photos.search
  &api_key=754a89ad04e0b72f42ffb77f412c021e
  &tags=blue,cool,pretty

Sample Search Result




```
−<rsp stat="ok">
  −<photos page="1" pages="99433" perpage="100" total="9943226">
    <photo id="10925924743" owner="100272092@N07" secret="37d0e7af76" server="7455" farm="8" title="Dribblebabies Vendor Fair" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="10925643325" owner="100272092@N07" secret="016b025f56" server="3758" farm="4" title="Dribblebabies Vendor Fair" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="10925736264" owner="104102768@N06" secret="74a4bca1cf" server="7357" farm="8" title="Sky Blue Glass Evil Eye Genuine Magnetic Hematite Bracelet, Evil Eye Bracelet, #3" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="10925631155" owner="85498821@N08" secret="2af57f47e2" server="5475" farm="6" title="" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="10925746854" owner="29342764@N04" secret="98695778d3" server="2816" farm="3" title="Beaver Moon" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="10925932073" owner="100272092@N07" secret="55ab54525f" server="3828" farm="4" title="Dribblebabies Vendor Fair" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="10925625596" owner="85498821@N08" secret="1fa0e20d12" server="3832" farm="4" title="Up there" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="8059159748" owner="61674778@N07" secret="4e41dba87f" server="8457" farm="9" title="Jeannette, Haarlem 2011: Up and away OV" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="10925648603" owner="94210132@N08" secret="b5af6e53e3" server="3763" farm="4" title="franz bread delivery truck" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="10925850493" owner="104106325@N04" secret="8cbcd82b87" server="2822" farm="3" title="Beautiful 30 Strand Blue Seed Bead Necklace - Handcrafted - 30" in Length" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="10925479495" owner="15119982@N02" secret="0efcdd1441" server="5532" farm="6" title="MBlatchford-20131118-0216" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="10925523046" owner="35770825@N03" secret="bc195e7b6e" server="7322" farm="8" title="Otafukuma Pokupoku Stuffed Animal(おたふくま ぽくぽく ぬいぐるみ)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="10925575225" owner="28099933@N03" secret="da3deed406" server="7305" farm="8" title="blue pencil" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="10925530496" owner="35770825@N03" secret="037ec18650" server="5500" farm="6" title="Otafukuma Pokupoku Stuffed Animal(おたふくま ぽくぽく ぬいぐるみ)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="10310695864" owner="27278265@N02" secret="be385d5711" server="3783" farm="4" title="Un temps pour voir la beauté de la nature" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="10925755163" owner="35770825@N03" secret="2dfe2baee8" server="3761" farm="4" title="Otafukuma Pokupoku Stuffed Animal(おたふくま ぽくぽく ぬいぐるみ)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="10925625834" owner="86818001@N08" secret="35a6e46e9a" server="7304" farm="8" title="missing" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="10925611584" owner="35770825@N03" secret="bc84b0ed49" server="3675" farm="4" title="Otafukuma Pokupoku Stuffed Animal(おたふくま ぽくぽく ぬいぐるみ)" ispublic="1" isfriend="0" isfamily="0"/>
```

Parse Result to URL for Picture

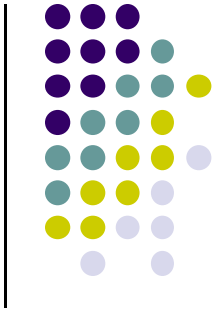


Photo Source URLs

You can construct the source URL to a photo once you know its ID, server ID, farm ID and secret, as return

The URL takes the following format:

```
http://farm{farm-id}.staticflickr.com/{server-id}/{id}_{secret}.jpg
        or
http://farm{farm-id}.staticflickr.com/{server-id}/{id}_{secret}_[mstzb]
        or
http://farm{farm-id}.staticflickr.com/{server-id}/{id}_{o-secret}_o.(jp
```

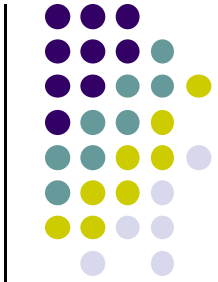
* Before November 18th, 2011 the API returned image URLs with hostnames like: "farm{farm-id}.static.flic

Size Suffixes

The letter suffixes are as follows:

s small square 75x75
q large square 150x150
t thumbnail, 100 on longest side
m small, 240 on longest side

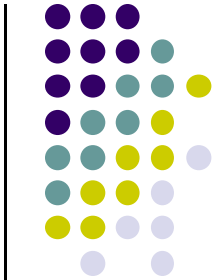
Flickr URL to Specific Picture



Example

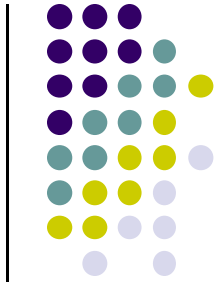
```
https://farm1.staticflickr.com/2/1418878_1e92283336_m.jpg

farm-id: 1
server-id: 2
photo-id: 1418878
secret: 1e92283336
size: m
```

JSON Format



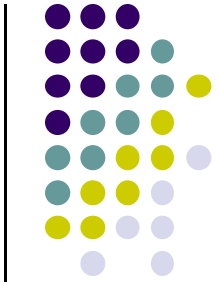
- Flickr allows JSON format
- **JSON** (JavaScript Object Notation) is a lightweight data-interchange format.
- Easy for humans to read and write, and for machines to parse and generate.


```
jsonFlickrApi({"photos":{"page":1, "pages":99436, "perpage":100,
"total":"9943595", "photo":[{"id":"8987334232",
"owner":"95960745@N05", "secret":"8ea7fa4884", "server":"3700",
"farm":4, "title":"ladder", "ispublic":1, "isfriend":0,
"isfamily":0}, {"id":"10922370846", "owner":"67877697@N07",
"secret":"9ac9663673", "server":"3670", "farm":4, "title":"Underneath
U.F.O.s", "ispublic":1, "isfriend":0, "isfamily":0},
{"id":"10926178565", "owner":"34128007@N04", "secret":"f4212fe642",
"server":"2875", "farm":3, "title":"Los Angeles City Hall",
"ispublic":1, "isfriend":0, "isfamily":0}, {"id":"10926310046",
"owner":"78618023@N04", "secret":"bc70fd4960", "server":"3678",
"farm":4, "title":"DSC_9235", "ispublic":1, "isfriend":0,
"isfamily":0}, {"id":"10926152545", "owner":"26870747@N04",
"secret":"80ab7da855", "server":"7328", "farm":8, "title":"Sunday
Sunrise", "ispublic":1, "isfriend":0, "isfamily":0},
{"id":"10926228834", "owner":"69774579@N07", "secret":"aaf5eeced1",
"server":"5533", "farm":6, "title":"Production of light",
```

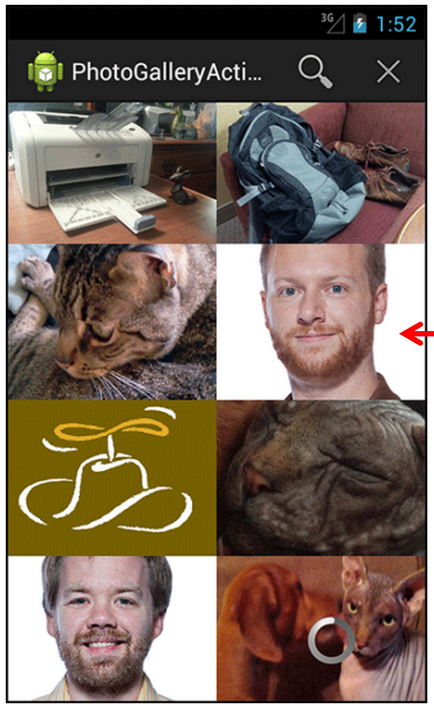


Accessing Web Services using HTTP & Background Tasks

PhotoGallery App
Ref: Android Nerd Ranch Chapter 26

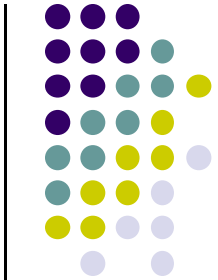


- Introduces networking in Android
- Android app is **Flickr** client
- **Version 1:** Retrieve photos from Flickr using HTTP, display captions
- Later: retrieve photos



Final Version

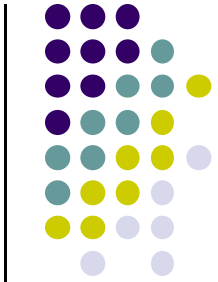
Create Blank Activity



- App will have Activity **PhotoGalleryActivity.java**
- **PhotoGallerActivity** will contain single fragment **PhotoGalleryFragment**
- Use **SingleFragmentActivity**

```
public class PhotoGalleryActivity extends Activity {
public class PhotoGalleryActivity extends SingleFragmentActivity {

    /* Auto-generated template code */

    @Override
    public Fragment createFragment() {
        return new PhotoGalleryFragment();
    }
}
```

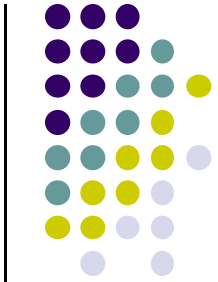
Display Results as GridView



- Need to:
  - Create GridView layout in XML and inflate it
  - Create a data source (GridView is an AdapterView)
- So, Create a GridView

```
GridView
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/gridView"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:columnWidth="120dp"
android:numColumns="auto_fit"
android:stretchMode="columnWidth"
```

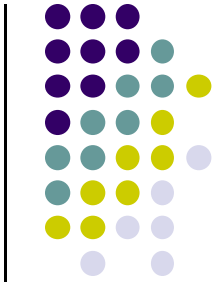
Inflate GridView + Get Handle in PhotoGalleryFragment



```
package com.bignerdranch.android.photogallery;

...

public class PhotoGalleryFragment extends Fragment {
    GridView mGridView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setRetainInstance(true);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_photo_gallery, container, false);

        mGridView = (GridView)v.findViewById(R.id.gridView);

        return v;
    }
}
```

If host Activity is stopped and resumed, fragment is also recreated.

Create a Class to Handle Networking



- **FlickrFetchr** class to handle networking in the app

- 2 methods
  - **getUrlBytes:** fetches raw data from a URL, returns it as array of bytes
  - **getUrl:** converts results of **getUrlBytes** to a **String**

```
package com.bignerdranch.android.photogallery;

...

public class FlickrFetchr {
    byte[] getUrlBytes(String urlSpec) throws IOException {
        URL url = new URL(urlSpec);
        HttpURLConnection connection = (HttpURLConnection)url.openConnection();

        try {
            ByteArrayOutputStream out = new ByteArrayOutputStream();
            InputStream in = connection.getInputStream();

            if (connection.getResponseCode() != HttpURLConnection.HTTP_OK) {
                return null;
            }

            int bytesRead = 0;
            byte[] buffer = new byte[1024];
            while ((bytesRead = in.read(buffer)) > 0) {
                out.write(buffer, 0, bytesRead);
            }
            out.close();
            return out.toByteArray();
        } finally {
            connection.disconnect();
        }
    }

    public String getUrl(String urlSpec) throws IOException {
        return new String(getUrlBytes(urlSpec));
    }
}
```

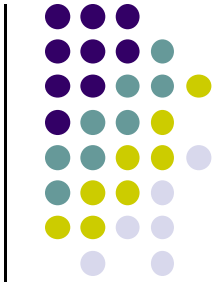
Make HTTP connection

Read bytes of data

Disconnect HTTP connection

Convert bytes returned to string

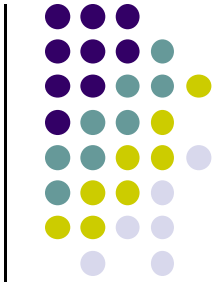
Ask App User's Permission to Network



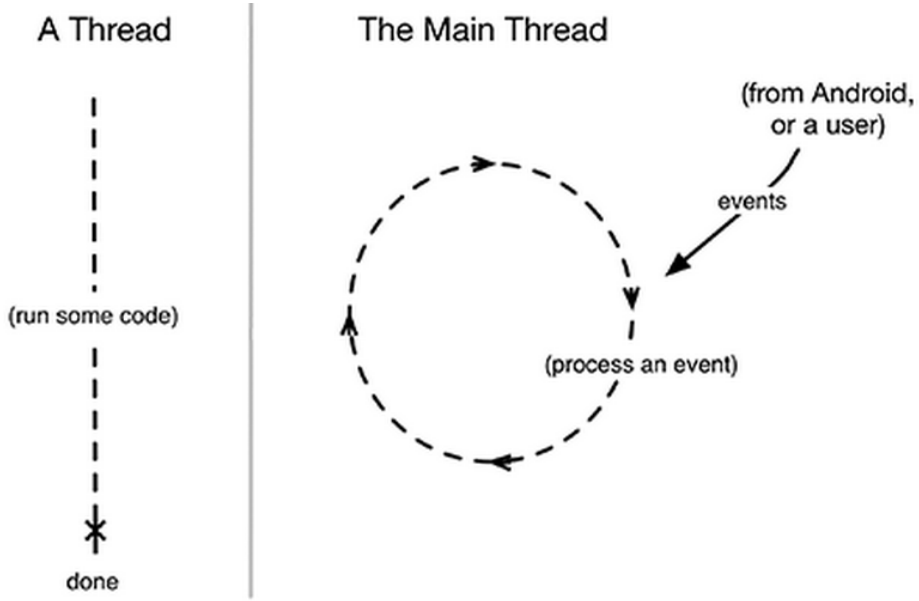
- Modify **AndroidManifest.xml** to ask app user for permission to user their Internet connection

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.bignerdranch.android.photogallery"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="15" />
  <uses-permission android:name="android.permission.INTERNET" />

  ...
```

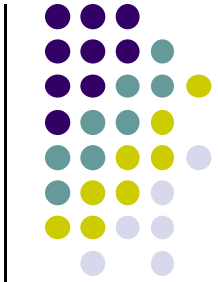
No Networking in Main Thread



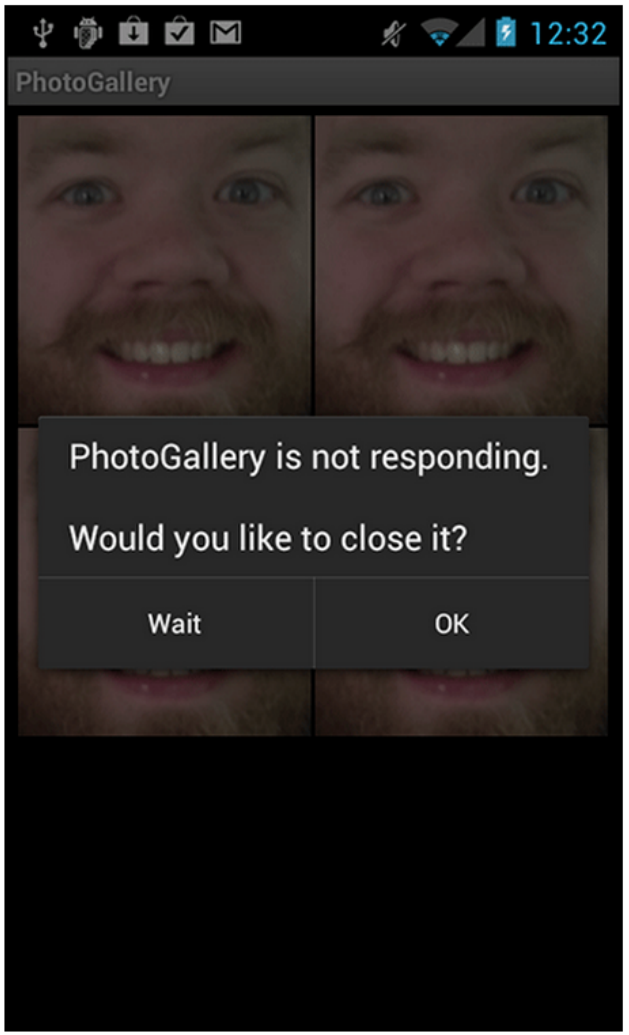
- Networking can take a long time (download, etc)
- Main thread needs to be free to handle UI, respond to events
- Android throws **NetworkOnMainThreadException** if networking code is put in main thread
- Hence we network in separate background thread using **AsyncThread**



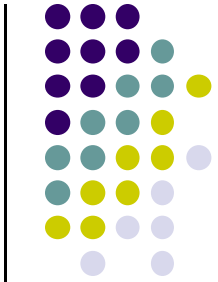
What Happens if Main Thread is Busy?



- Doing networking on main thread will freeze user interaction
- If Flickr download takes a long time, we get **Application Not Responding**
- **Not good!**



Use AsyncTask to Fetch on Background Thread



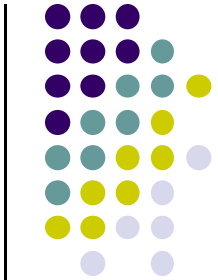
- Utility class **AsyncTask** creates background thread
  - Runs code in **doInBackground(..) method** on that thread
- Add new inner class called **FetchItemsTask** to **PhotoGalleryFragment**

```
public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";

    GridView mGridView;

    ...

    private class FetchItemsTask extends AsyncTask<Void,Void,Void> {
        @Override
        protected Void doInBackground(Void... params) {
            try {
                String result = new FlickrFetchr().getUrl("http://www.google.com");
                Log.i(TAG, "Fetched contents of URL: " + result);
            } catch (IOException ioe) {
                Log.e(TAG, "Failed to fetch URL: ", ioe);
            }
            return null;
        }
    }
}
```

Creates a background thread

code put here in doInBackground is run on background thread

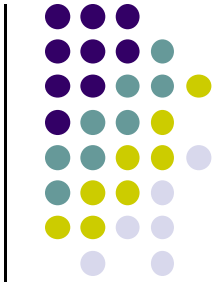
Execute FetchItemsTask



- Execute **FetchItemsTask** inside **PhotoGalleryFragment.onCreate( )**

```
public class PhotoGalleryFragment extends Fragment {
    private static final String TAG = "PhotoGalleryFragment";

    GridView mGridView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setRetainInstance(true);
        new FetchItemsTask().execute();
    }

    ...

}
```

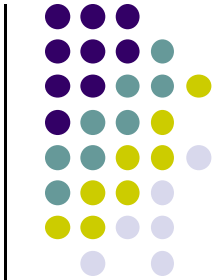
Fetching XML from Flickr



- Flickr **getRecent** method returns list of latest public photos on Flickr
- Requires getting an API key
- First add some constants to our class **FlickrFetchr**

```
public class FlickrFetchr {
    public static final String TAG = "FlickrFetchr";

    private static final String ENDPOINT = "http://api.flickr.com/services/rest/";
    private static final String API_KEY = "yourApiKeyHere";
    private static final String METHOD_GET_RECENT = "flickr.photos.getRecent";
    private static final String PARAM_EXTRAS = "extras";

    private static final String EXTRA_SMALL_URL = "url_s";
```

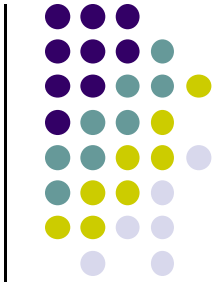
Add fetchItems( ) method



- Use constants to write **fetchItems( )** method that builds appropriate URL and fetches its content

```
public class FlickrFetchr {

    ...

    String getUrl(String urlSpec) throws IOException {
        return new String(getUrlBytes(urlSpec));
    }

    public void fetchItems() {
        try {
            String url = Uri.parse(ENDPOINT).buildUpon()
                    .appendQueryParameter("method", METHOD_GET_RECENT)
                    .appendQueryParameter("api_key", API_KEY)
                    .appendQueryParameter(PARAM_EXTRAS, EXTRA_SMALL_URL)
                    .build().toString();
            String xmlString = getUrl(url);
            Log.i(TAG, "Received xml: " + xmlString);
        } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch items", ioe);
        }
    }
}
```

Modify AsyncTask to Call New fetchItems( )

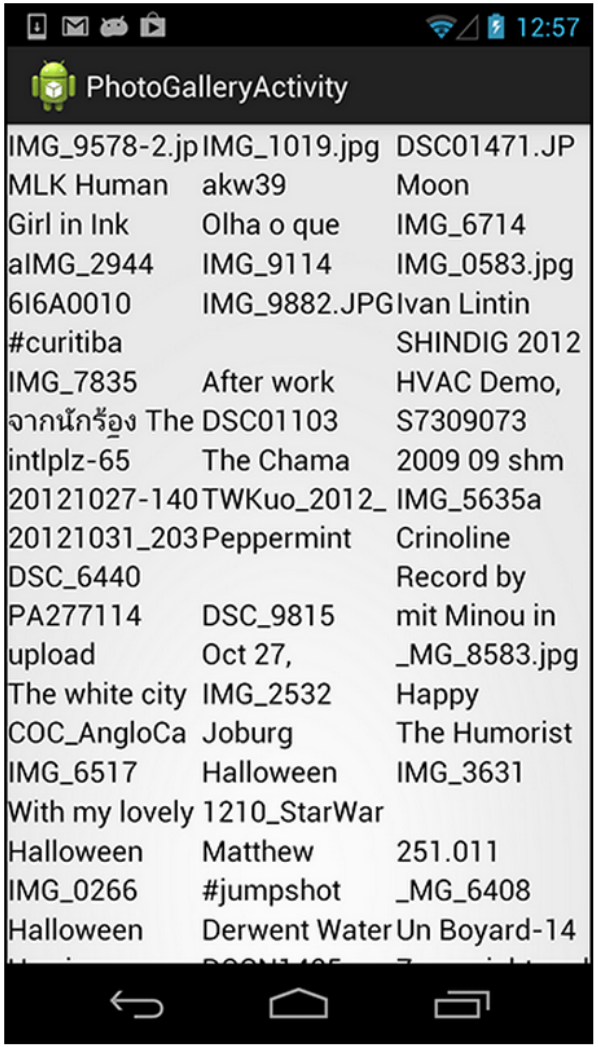


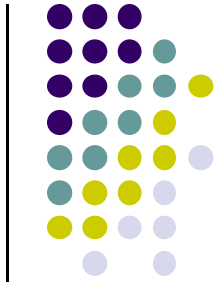
```
private class FetchItemsTask extends AsyncTask<Void,Void,Void> {
    @Override
    protected Void doInBackground(Void... params) {
        try {
            String result = new FlickrFetchr().getUrl("http://www.google.com");
            Log.i(TAG, "Fetched contents of URL: " + result);
        } catch (IOException ioe) {
            Log.e(TAG, "Failed to fetch URL: ", ioe);
        }
        new FlickrFetchr().fetchItems();
        return null;
    }
}
```

Builds HTTP URL, fetches items

Results?

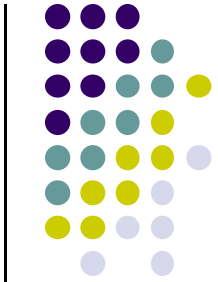
- Flickr Item Captions displayed





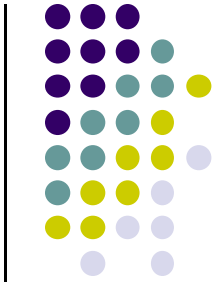
Broadcast Receivers

Broadcast Receivers



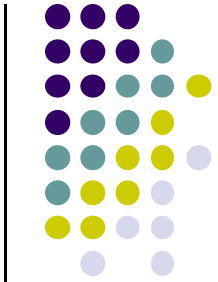
- Another main application component
- "A *broadcast receiver* is a component that responds to system-wide broadcast announcements."
- Android system sends many kinds of broadcasts
  - screen turned off,
  - battery low,
  - picture captured,
  - SMS received,
  - SMS sent

Broadcast Receivers



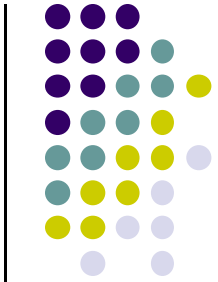
- Your app may want to listen for particular broadcasts
  - Text message received
  - Boot up complete
  - Shutting down
- App creates and registers a Broadcast Receiver
- Can create/register broadcast receiver in XML or in Java code

Declaring Broadcast Receiver in AndroidManifest.xml



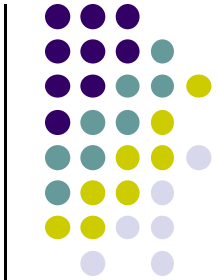
```
<application
    android:icon="@drawable/icon"
    android:label="@string/app_name" >
    <activity
        android:name=".ServiceConsumerActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

    <receiver android:name="MyScheduleReceiver" >
        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED" />
        </intent-filter>
    </receiver>
    <receiver android:name="MyStartServiceReceiver" >
    </receiver>
</application>
```

Broadcast Receivers



- Apps can initiate broadcasts to inform other applications of status or readiness
- Broadcast Receivers don't display UI
  - may create status bar notifications
- Broadcasts are delivered to interested apps as Intents
- intents sent using **sendBroadcast()** method
- Can use **LocalBroadcastManager** to send Broadcasts within your application only

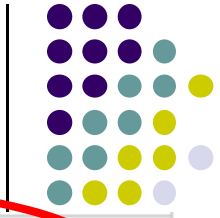
BroadcastReceivers



- What broadcasts are available?
- Check the Intent class
- http://developer.android.com/reference/android/content/Intent.html
  - search for "Broadcast Action"

- Also look in android-sdk\platforms\<number>\data\broadcast_actions.txt

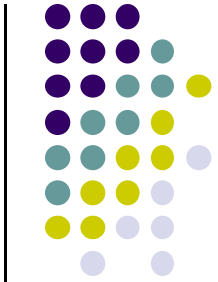
Broadcasts



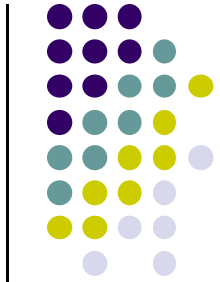
String	ACTION_CAMERA_BUTTON	Broadcast Action: The "Camera Button" was pressed.
String	ACTION_CHOOSER	Activity Action: Display an activity chooser, allowing the user to pick what they want to before proceeding.
String	ACTION_CLOSE_SYSTEM_DIALOGS	Broadcast Action: This is broadcast when a user action should request a temporary system dialog to dismiss.
String	ACTION_CONFIGURATION_CHANGED	Broadcast Action: The current device `Configuration` (orientation, locale, etc) has changed.
String	ACTION_CREATE_SHORTCUT	Activity Action: Creates a shortcut.
String	ACTION_DATE_CHANGED	Broadcast Action: The date has changed.
String	ACTION_DEFAULT	A synonym for `ACTION_VIEW`, the "standard" action that is performed on a piece of data.
String	ACTION_DELETE	Activity Action: Delete the given data from its container.
String	ACTION_DEVICE_STORAGE_LOW	Broadcast Action: A sticky broadcast that indicates low memory condition on the device This is a protected intent that can only be sent by the system.

Broadcasts

- from broadcast_actions.txt in sdk files



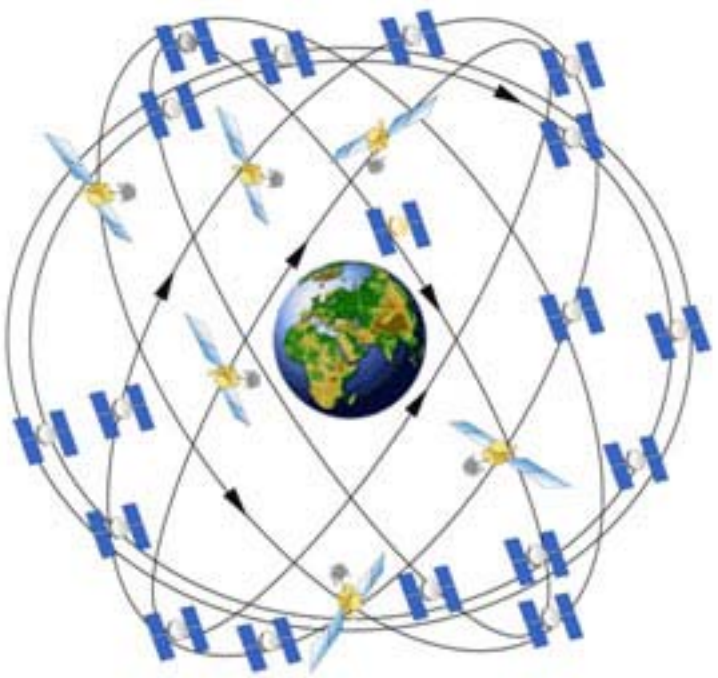
```
android.intent.action.TIME_SET
android.intent.action.TIME_TICK
android.intent.action.UID_REMOVED
android.intent.action.USER_PRESENT
android.intent.action.WALLPAPER_CHANGED
android.media.ACTION_SCO_AUDIO_STATE_UPDATED
android.media.AUDIO_BECOMING_NOISY
android.media.RINGER_MODE_CHANGED
android.media.SCO_AUDIO_STATE_CHANGED
android.media.VIBRATE_SETTING_CHANGED
android.media.action.CLOSE_AUDIO_EFFECT_CONTROL_SESSION
android.media.action.OPEN_AUDIO_EFFECT_CONTROL_SESSION
android.net.conn.BACKGROUND_DATA_SETTING_CHANGED
android.net.wifi.NETWORK_IDS_CHANGED
android.net.wifi.RSSI_CHANGED
android.net.wifi.SCAN_RESULTS
android.net.wifi.STATE_CHANGE
android.net.wifi.WIFI_STATE_CHANGED
android.net.wifi.p2p.CONNECTION_STATE_CHANGE
android.net.wifi.p2p.PEERS_CHANGED
android.net.wifi.p2p.STATE_CHANGED
android.net.wifi.p2p.THIS_DEVICE_CHANGED
android.net.wifi.supplicant.CONNECTION_CHANGE
android.net.wifi.supplicant.STATE_CHANGE
android.provider.Telephony.SIM_FULL
android.provider.Telephony.SMS_CB_RECEIVED
android.provider.Telephony.SMS_EMERGENCY_CB_RECEIVED
android.provider.Telephony.SMS_RECEIVED
android.provider.Telephony.SMS_REJECTED
android.provider.Telephony.WAP_PUSH_RECEIVED
android.speech.tts.TTS_QUEUE_PROCESSING_COMPLETED
android.speech.tts.engine.TTS_DATA_INSTALLED
```



Tracking the Device's Location

Global Positioning System (GPS)

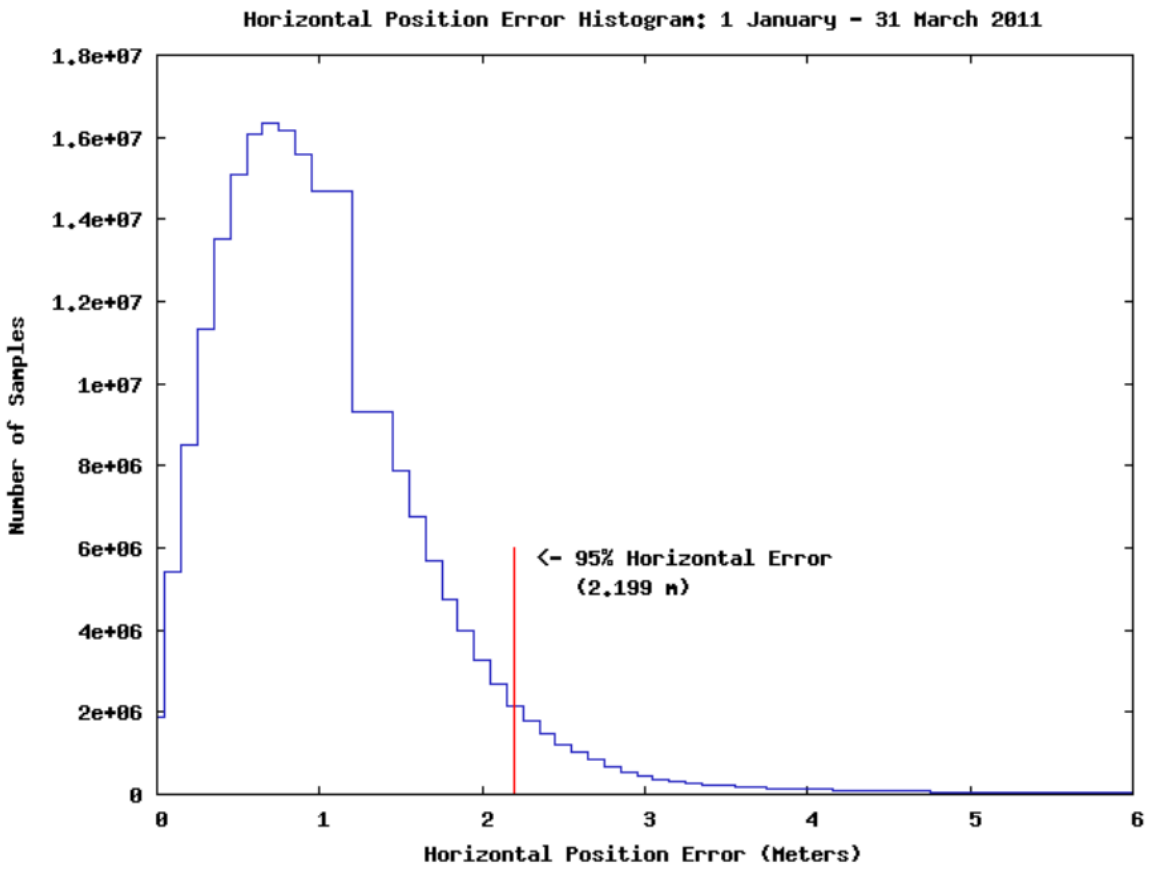
- 24 core satellites
- medium earth orbit, 20,000 km above earth
- 6 orbital planes with 4 satellites each
- 4 satellites visible from any spot on earth



- Recently upgraded to 27 sats

GPS User Segment

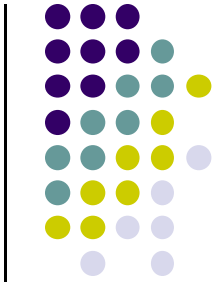
- Receiver calculates position and course by comparing time signals from multiple satellites with based on known positions of those satellites
- Accuracy normally within 5 - 10 meters



Android and Location

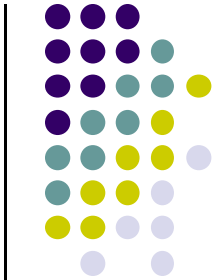
- Obtaining User Location
- GPS most accurate but
  - only works OUTDOORS
  - quickly consumes battery power
  - delay in acquiring satellites or re-acquiring if lost
- Can use Wi-Fi in some situations
- Map device's map locations based on combination of wi-fi access points (known location) seen

Tracking Device's Location
Android Nerd Ranch Ch 33



- **Goal:** Create new application called **runTracker**
- **runTracker** uses phones GPS to record and display user's travels (walking in woods, car ride, ocean voyage, etc)
- First version of **runTracker** gets GPS updates, displays current location on screen
- **Later:** Show map that follows user in real time

Create RunActivity



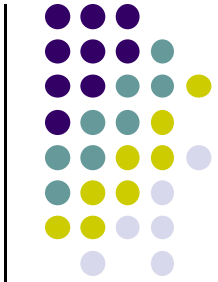
- **Compile using:** Google APIs, minimum SDK = 9
- Create **RunActivity**, subclass of **SingleFragmentActivity**

```
public class RunActivity extends SingleFragmentActivity {

    @Override
    protected Fragment createFragment() {
        return new RunFragment();
    }

}
```

Create XML for UI

- Next, create user interface (XML) and initial version of **RunFragment**
- UI will display data about current **"run"** and its location
- Has **Start**, **Stop** buttons
- Use **TableLayout** for UI

Add Strings to Strings.xml



- Add strings required for UI in **strings.xml**

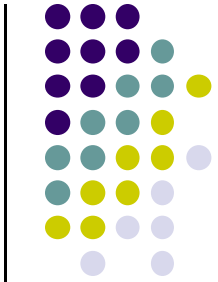
```
<resources>
  <string name="app_name">RunTracker</string>
  <string name="started">Started:</string>
  <string name="latitude">Latitude:</string>
  <string name="longitude">Longitude:</string>
  <string name="altitude">Altitude:</string>
  <string name="elapsed_time">Elapsed Time:</string>
  <string name="start">Start</string>
  <string name="stop">Stop</string>
  <string name="gps_enabled">GPS Enabled</string>
  <string name="gps_disabled">GPS Disabled</string>
  <string name="cell_text">Run at %1$s</string>
</resources>
```

Create RunFragment

- Create **RunFragment** class
- Initial version generates UI onto screen, provides access to widgets

```
public class RunFragment extends Fragment {
    private Button mStartButton, mStopButton;
    private TextView mStartedTextView, mLatitudeTextView,
        mLongitudeTextView, mAltitudeTextView, mDurationTextView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_run, container, false);

        mStartedTextView = (TextView)view.findViewById(R.id.run_startedTextView);
        mLatitudeTextView = (TextView)view.findViewById(R.id.run_latitudeTextView);
        mLongitudeTextView = (TextView)view.findViewById(R.id.run_longitudeTextView);
        mAltitudeTextView = (TextView)view.findViewById(R.id.run_altitudeTextView);
        mDurationTextView = (TextView)view.findViewById(R.id.run_durationTextView);

        mStartButton = (Button)view.findViewById(R.id.run_startButton);

        mStopButton = (Button)view.findViewById(R.id.run_stopButton);

        return view;
    }
}
```

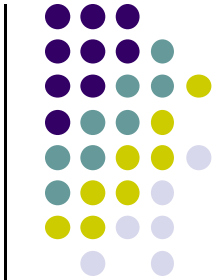
Android Location using LocationManager



- In Android, location data provided by **LocationManager** system service
- **LocationManager** provides location updates to applications that are interested
- 2 alternatives for **LocationManager** to deliver updates
  1. **LocationListener** interface: Gives location updates (via **onLocationChanged(location)**), status updates and notifications
  2. **PendingUpdate** API: Receive future location updates as **Intent**

- **runTracker** app will use **PendingUpdate** approach (second way)

Communicating with LocationManager



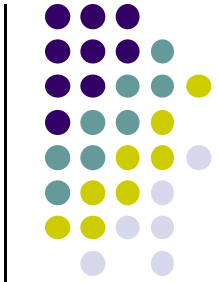
```
public class RunManager {
    private static final String TAG = "RunManager";

    public static final String ACTION_LOCATION =
        "com.bignerdranch.android.runtracker.ACTION_LOCATION";

    private static RunManager sRunManager;
    private Context mAppContext;
    private LocationManager mLocationManager;

    // The private constructor forces users to use RunManager.get(Context)
    private RunManager(Context appContext) {
        mAppContext = appContext;
        mLocationManager = (LocationManager)mAppContext
            .getSystemService(Context.LOCATION_SERVICE);
    }

    public static RunManager get(Context c) {
        if (sRunManager == null) {
            // Use the application context to avoid leaking activities
            sRunManager = new RunManager(c.getApplicationContext());
        }
        return sRunManager;
    }

    private PendingIntent getLocationPendingIntent(boolean shouldCreate) {
        Intent broadcast = new Intent(ACTION_LOCATION);
        int flags = shouldCreate ? 0 : PendingIntent.FLAG_NO_CREATE;
        return PendingIntent.getBroadcast(mAppContext, 0, broadcast, flags);
    }

    public void startLocationUpdates() {
        String provider = LocationManager.GPS_PROVIDER;

        // Start updates from the location manager
        PendingIntent pi = getLocationPendingIntent(true);
        mLocationManager.requestLocationUpdates(provider, 0, 0, pi);
    }

    public void stopLocationUpdates() {
        PendingIntent pi = getLocationPendingIntent(false);
        if (pi != null) {
            mLocationManager.removeUpdates(pi);
            pi.cancel();
        }
    }

    public boolean isTrackingRun() {
        return getLocationPendingIntent(false) != null;
    }

}
```

- Create class **RunManager** to manage communication with **LocationManager**

- Implement 3 methods for
  - Starting location updates
  - Stopping location updates
  - Getting location updates

Set frequency depending on how much delay app can withstand

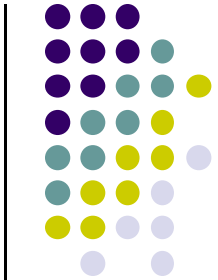
Receiving Broadcast Location Updates



- Create broadcast receiver class **LocationReceiver**
- Receives location updates whether **runTracker** is running or not
- Overrides **onReceive** method
- **LocationManager** packs intent with “extras”

```
public class LocationReceiver extends BroadcastReceiver {
  private static final String TAG = "LocationReceiver";

  @Override
  public void onReceive(Context context, Intent intent) {
      // If you got a Location extra, use it
      Location loc = (Location)intent
          .getParcelableExtra(LocationManager.KEY_LOCATION_CHANGED);
      if (loc != null) {
          onLocationReceived(context, loc);
          return;
      }
      // If you get here, something else has happened
      if (intent.hasExtra(LocationManager.KEY_PROVIDER_ENABLED)) {
          boolean enabled = intent
              .getBooleanExtra(LocationManager.KEY_PROVIDER_ENABLED, false);
          onProviderEnabledChanged(enabled);
      }
  }

  protected void onLocationReceived(Context context, Location loc) {
      Log.d(TAG, this + " Got location from " + loc.getProvider() + ": "
          + loc.getLatitude() + ", " + loc.getLongitude());
  }

  protected void onProviderEnabledChanged(boolean enabled) {
      Log.d(TAG, "Provider " + (enabled ? "enabled" : "disabled"));
  }

}
```

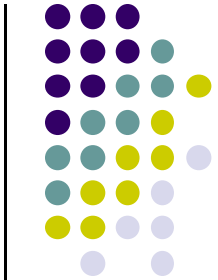
Add Location Permission



- Add **ACCESS_FINE_LOCATION** permission to AndroidManifest
- Also adds uses-feature location.gps
- Declare Location Receiver

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.bignerdranch.android.runtracker"
  android:versionCode="1"
  android:versionName="1.0">

  <uses-sdk android:minSdkVersion="9" android:targetSdkVersion="15" />
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
  <uses-feature android:required="true"
    android:name="android.hardware.location.gps"/>

  <application android:label="@string/app_name"
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:theme="@style/AppTheme">
    <activity android:name=".RunActivity"
      android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <receiver android:name=".LocationReceiver"
      android:exported="false">
      <intent-filter>
        <action android:name="com.bignerdranch.android.runtracker.ACTION_LOCATION"/>
      </intent-filter>
    </receiver>

  </application>

</manifest>
```

AndroidManifest.xml



- User Permission in manifest

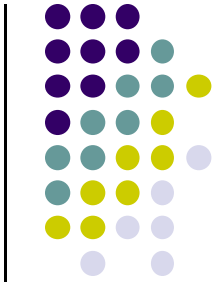
```
<manifest ... >
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    ...
</manifest>
```

- **Options:** ACCESS_FINE_LOCATION or ACCESS_COARSE_LOCATION

- ACCESS_COARSE_LOCATION: use cell-ID and Wi-Fi

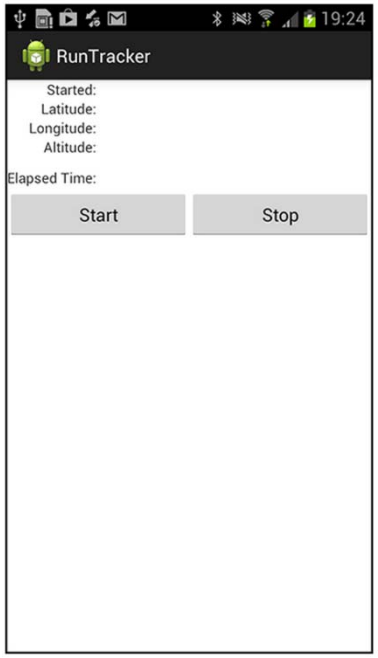
- ACCESS_FINE_LOCATION: use GPS

Add Click Listeners to Start/Stop Buttons

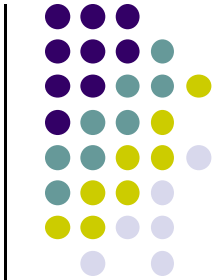


```
public class RunFragment extends Fragment {

    private RunManager mRunManager;

    private Button mStartButton, mStopButton;
    private TextView mStartedTextView, mLatitudeTextView,
        mLongitudeTextView, mAltitudeTextView, mDurationTextView;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
        mRunManager = RunManager.get(getActivity());
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
            Bundle savedInstanceState) {
        ...

        mStartButton = (Button)view.findViewById(R.id.run_startButton);
        mStartButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mRunManager.startLocationUpdates();
                updateUI();
            }
        });

        mStopButton = (Button)view.findViewById(R.id.run_stopButton);
        mStopButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mRunManager.stopLocationUpdates();
                updateUI();
            }
        });

        updateUI();


        return view;
    }

    private void updateUI() {
        boolean started = mRunManager.isTrackingRun();

        mStartButton.setEnabled(!started);
        mStopButton.setEnabled(started);
    }
}
```

- Done to test its all working
- Add simple **updateUI( )** method



Testing Locations on Real vs Virtual Devices



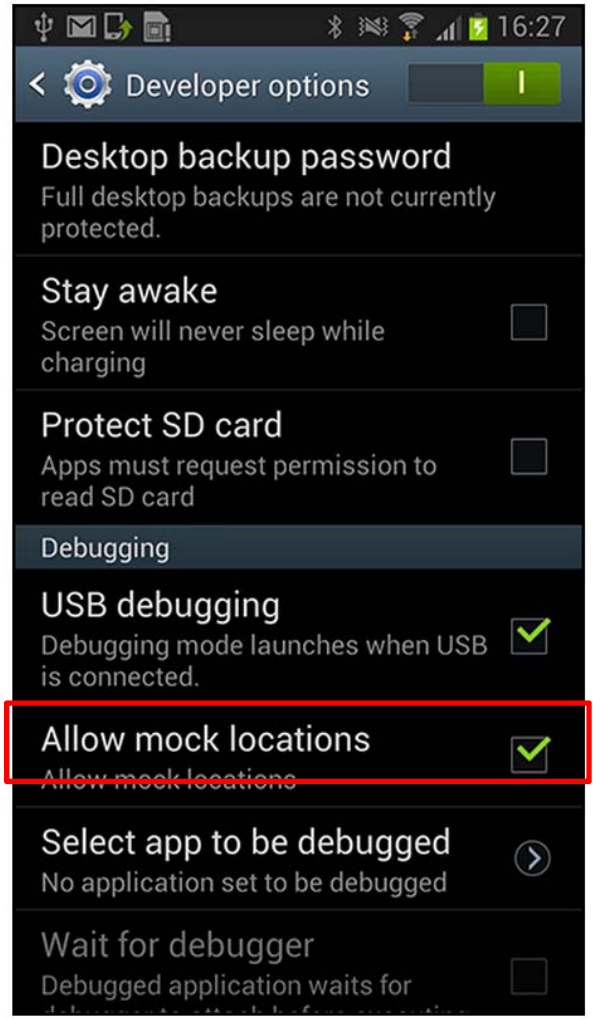
- Can be tricky to test if locations received are correct
- Can test by manually setting **LocationManager** locations in emulator (either one by one or series of locations in a file)
- Steps to test on a real device a bit more difficult

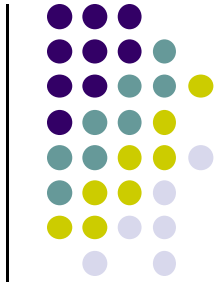
1. Request the `ACCESS_MOCK_LOCATION` permission.
2. Add a test provider via **`LocationManager.addTestProvider(…)`**.
3. Enable the provider using **`setTestProviderEnabled(…)`**.
4. Set its initial status with **`setTestProviderStatus(…)`**.
5. Publish locations with **`setTestProviderLocation(…)`**.
6. Remove your test provider using **`removeTestProvider(…)`**.

- Download and use test project (mock locations) provided by Book (Android Nerd Ranch)

Allow Mock Locations

- For **TestProvider** class provided by text to work, turn on **Allow mock locations** in **Developer options** menu

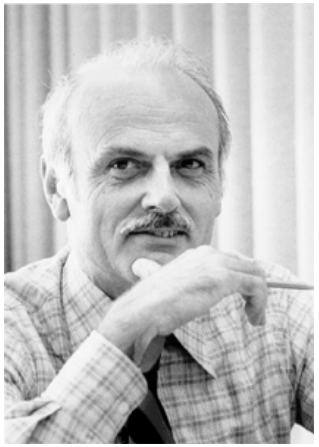




Local Databases with SQLite

Databases

- RDBMS
  - relational data base management system
- Relational databases introduced by E. F. Codd
  - Turing Award Winner
- Relational Database
  - data stored in tables
  - relationships among data stored in tables
  - data can be accessed and viewed in different ways



Example Database

- Wines

Winery Table

Winery ID	Winery name	Address	Region ID
1	Moss Brothers	Smith Rd.	3
2	Hardy Brothers	Jones St.	1
3	Penfolds	Arthurton Rd.	1
4	Lindemans	Smith Ave.	2
5	Orlando	Jones St.	1

Region Table

Region ID	Region name	State
1	Barossa Valley	South Australia
2	Yarra Valley	Victoria
3	Margaret River	Western Australia

Ref: Web Database Applications with PHP and MySQL, 2nd Edition , by Hugh E. Williams, David Lane

Relational Data

- Data in different tables can be related

Winery Table

Winery ID	Winery name	Address	Region ID
1	Moss Brothers	Smith Rd.	3
2	Hardy Brothers	Jones St.	1
3	Penfolds	Arthurton Rd.	1
4	Lindemans	Smith Ave.	2
5	Orlando	Jones St.	1

Region Table

Region ID	Region name	State
1	Barossa Valley	South Australia
2	Yarra Valley	Victoria
3	Margaret River	Western Australia

Keys

- Each table has a key
- Column used to uniquely identify each row

KEYS

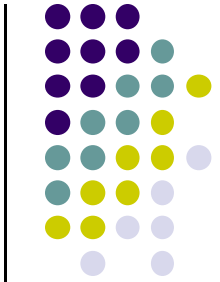
Winery Table

Winery ID	Winery name	Address	Region ID
1	Moss Brothers	Smith Rd.	3
2	Hardy Brothers	Jones St.	1
3	Penfolds	Arthurton Rd.	1
4	Lindemans	Smith Ave.	2
5	Orlando	Jones St.	1

Region Table

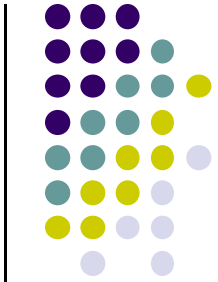
Region ID	Region name	State
1	Barossa Valley	South Australia
2	Yarra Valley	Victoria
3	Margaret River	Western Australia

SQL and Databases



- SQL is the language used to manipulate and manage information in a relational database management system (RDBMS)
- SQL Commands:
  - **CREATE TABLE** - creates new database table
  - **ALTER TABLE** - alters a database table
  - **DROP TABLE** - deletes a database table
  - **CREATE INDEX** - creates an index (search key)
  - **DROP INDEX** - deletes an index

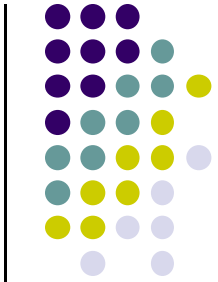
SQL Commands



- **SELECT** - get data from a database table
- **UPDATE** - change data in a database table
- **DELETE** - remove data from a database table
- **INSERT INTO** - insert new data in a database table

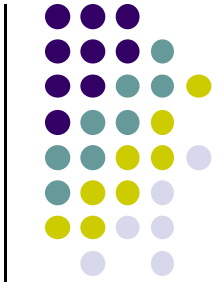
- SQLite implements most, but not all of SQL
  - http://www.sqlite.org/

Why Local Databases?
Android Nerd Ranch Chapter 34



- User may track their runs forever
- Lots of data
- **Solution:** Store runTracker runs and locations in SQLite database
- **SQLite** is open source relational database
- **SQLiteOpenHelper** encapsulates database creation, opening and updating
- In **runTracker**, create subclass of **SQLiteOpenHelper** called **RunDatabaseHelper**

Use Local Databases in runTracker



- Create 1 table for each type of data
- Thus, we create 2 tables
  - **Run** table
  - **Location** table
- **Idea:** A run can have many locations visited

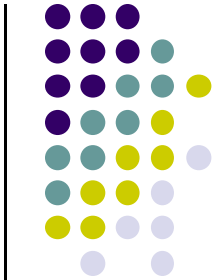
"run" Table

_id	start_date
1	13090636733242
2	13090732131909

"location" Table

run_id	timestamp	latitude	longitude	altitude	provider
1	13090636733242	-35.64771771	84.51398821	131	gps
1	13090636749481	-35.64738811	84.51711292	135	gps
2	13090732131909	12.58181929	30.38181219	248	gps
2	13090732148219	12.55118199	30.38171315	246	gps

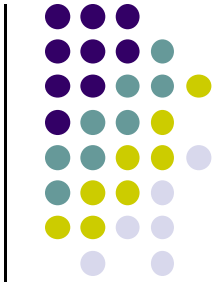
Create RunDatabaseHelper



- Need to override 2 methods: **onCreate( )** and **onUpgrade( )**
- **onCreate:** establish schema for newly created database
- **onUpgrade( ):** execute code to migrate to new version of schema
- Implement **insertRun(Run)** to write to database

```
public class RunDatabaseHelper extends SQLiteOpenHelper {
    private static final String DB_NAME = "runs.sqlite";
    private static final int VERSION = 1;

    private static final String TABLE_RUN = "run";
    private static final String COLUMN_RUN_START_DATE = "start_date";

    public RunDatabaseHelper(Context context) {
        super(context, DB_NAME, null, VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        // Create the "run" table
        db.execSQL("create table run (" +
            "_id integer primary key autoincrement, start_date integer)");
        // Create the "location" table
        db.execSQL("create table location (" +
            " timestamp integer, latitude real, longitude real, altitude real," +
            " provider varchar(100), run_id integer references run(_id))");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Implement schema changes and data massage here when upgrading
    }

    public long insertRun(Run run) {
        ContentValues cv = new ContentValues();
        cv.put(COLUMN_RUN_START_DATE, run.getStartDate().getTime());
        return getWritableDatabase().insert(TABLE_RUN, null, cv);
    }
}
```

Add ID to Run Class

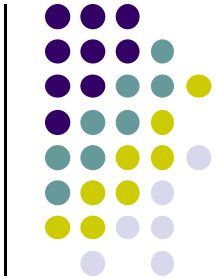


- Add ID property to **Runs** class

- ID required to
  - Distinguish runs
  - Support querying runs

```
public class Run {
    private long mId;
    private Date mStartDate;

    public Run() {
        mId = -1;
        mStartDate = new Date();
    }

    public long getId() {
        return mId;
    }

    public void setId(long id) {
        mId = id;
    }

    public Date getStartDate() {
        return mStartDate;
    }
```

Modify RunManager to Use New Database



Note: The rest of **runTracker** will use methods in **RunManager**

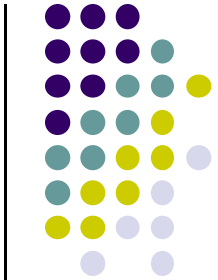
Use this method In RunFragment when a new run Is STARTED (When Start button is Pressed)

Use this method directly from RunFragment when a new run Is RESTARTED

Use this method In RunFragment Run Is STOPPED (When STOP button is Pressed)

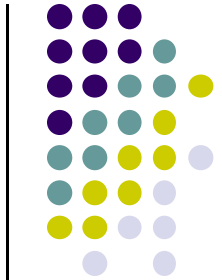
```
private void broadcastLocation(Location location) {
    Intent broadcast = new Intent(ACTION_LOCATION);
    broadcast.putExtra(LocationManager.KEY_LOCATION_CHANGED, location);
    mAppContext.sendBroadcast(broadcast);
}

public Run startNewRun() {
    // Insert a run into the db
    Run run = insertRun();
    // Start tracking the run
    startTrackingRun(run);
    return run;
}

public void startTrackingRun(Run run) {
    // Keep the ID
    mCurrentRunId = run.getId();
    // Store it in shared preferences
    mPrefs.edit().putLong(PREF_CURRENT_RUN_ID, mCurrentRunId).commit();
    // Start location updates
    startLocationUpdates();
}

public void stopRun() {
    stopLocationUpdates();
    mCurrentRunId = -1;
    mPrefs.edit().remove(PREF_CURRENT_RUN_ID).commit();
}

private Run insertRun() {
    Run run = new Run();
    run.setId(mHelper.insertRun(run));
    return run;
}
}
```

Use RunManager in RunFragment



```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_run, container, false);

    ...

    mStartButton = (Button)view.findViewById(R.id.run_startButton);
    mStartButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mRunManager.startLocationUpdates();
            mRun = new Run();
            mRun = mRunManager.startNewRun();
            updateUI();
        }
    });

    mStopButton = (Button)view.findViewById(R.id.run_stopButton);
    mStopButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mRunManager.stopLocationUpdates();
            mRunManager.stopRun();
            updateUI();
        }
    });

    updateUI();

    return view;
}
```

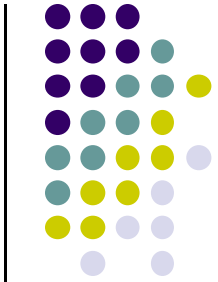
Inserting Locations into Database



- Similar to inserting runs, need to insert locations when **LocationManager** gives updates
- Add **insertLocation** method

```
public class RunDatabaseHelper extends SQLiteOpenHelper {
    private static final String DB_NAME = "runs.sqlite";
    private static final int VERSION = 1;

    private static final String TABLE_RUN = "run";
    private static final String COLUMN_RUN_START_DATE = "start_date";

    private static final String TABLE_LOCATION = "location";
    private static final String COLUMN_LOCATION_LATITUDE = "latitude";
    private static final String COLUMN_LOCATION_LONGITUDE = "longitude";
    private static final String COLUMN_LOCATION_ALTITUDE = "altitude";
    private static final String COLUMN_LOCATION_TIMESTAMP = "timestamp";
    private static final String COLUMN_LOCATION_PROVIDER = "provider";
    private static final String COLUMN_LOCATION_RUN_ID = "run_id";

    ...

    public long insertLocation(long runId, Location location) {
        ContentValues cv = new ContentValues();
        cv.put(COLUMN_LOCATION_LATITUDE, location.getLatitude());
        cv.put(COLUMN_LOCATION_LONGITUDE, location.getLongitude());
        cv.put(COLUMN_LOCATION_ALTITUDE, location.getAltitude());
        cv.put(COLUMN_LOCATION_TIMESTAMP, location.getTime());
        cv.put(COLUMN_LOCATION_PROVIDER, location.getProvider());
        cv.put(COLUMN_LOCATION_RUN_ID, runId);
        return getWritableDatabase().insert(TABLE_LOCATION, null, cv);
    }
```

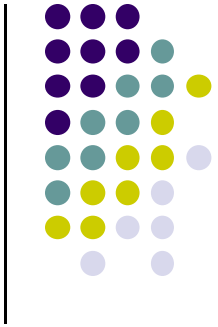
Continuously Handle Location Intents



- System will continuously give updates
- Need to receive location intents whether app is visible or not
- Implement **dedicated Location Receiver** to insert location
- Inserts location into run whenever new location is received

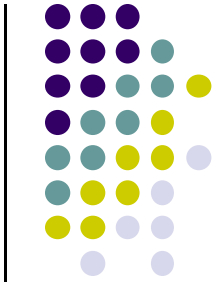
```
public class TrackingLocationReceiver extends LocationReceiver {

    @Override
    protected void onLocationReceived(Context c, Location loc) {
        RunManager.get(c).insertLocation(loc);
    }

}
```

Register Location Receiver in AndroidManifest.xml



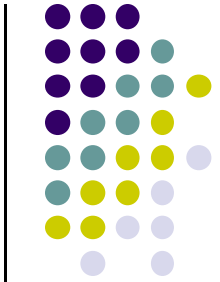
```
<application
  android:allowBackup="true"
  android:icon="@drawable/ic_launcher"
  android:label="@string/app_name"
  android:theme="@style/AppTheme">

  ...

  <receiver android:name=".LocationReceiver"
  <receiver android:name=".TrackingLocationReceiver"
    android:exported="false">
    <intent-filter>
      <action android:name="com.bignerdranch.android.runtracker.ACTION_LOCATION"/>
    </intent-filter>
  </receiver>

</application>
```

More Details in Book



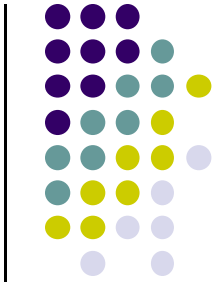
- Example in book also describes more features
  - Querying runs database
  - Displaying a list of runs
  - Creating an options list for the runs list
  - Querying a single run
  - Querying last location of a run

Alternatives to sqlite



- SQLite is low level ("Down in the weeds“)
- Various alternatives to work higher up the food chain
- Object Relational Mappers - ORM
- Higher level wrappers for dealing with SQL commands and sqlite databases
- Many ORMs exist

References



- Busy Coder's guide to Android version 6.3
- CS 65/165 slides, Dartmouth College, Spring 2014
- CS 371M slides, U of Texas Austin, Spring 2014