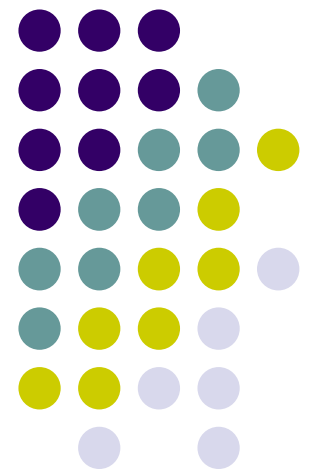
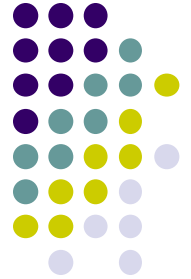


CS 528 Mobile and Ubiquitous Computing

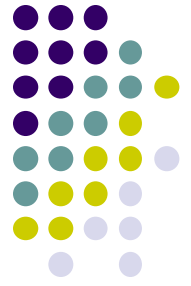
Lecture 5: Web Services, Broadcast Receivers, Tracking Location, SQLite Databases

Emmanuel Agu



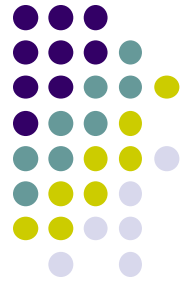


Web Services



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

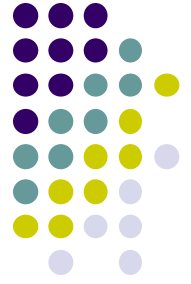


Web Services Sources

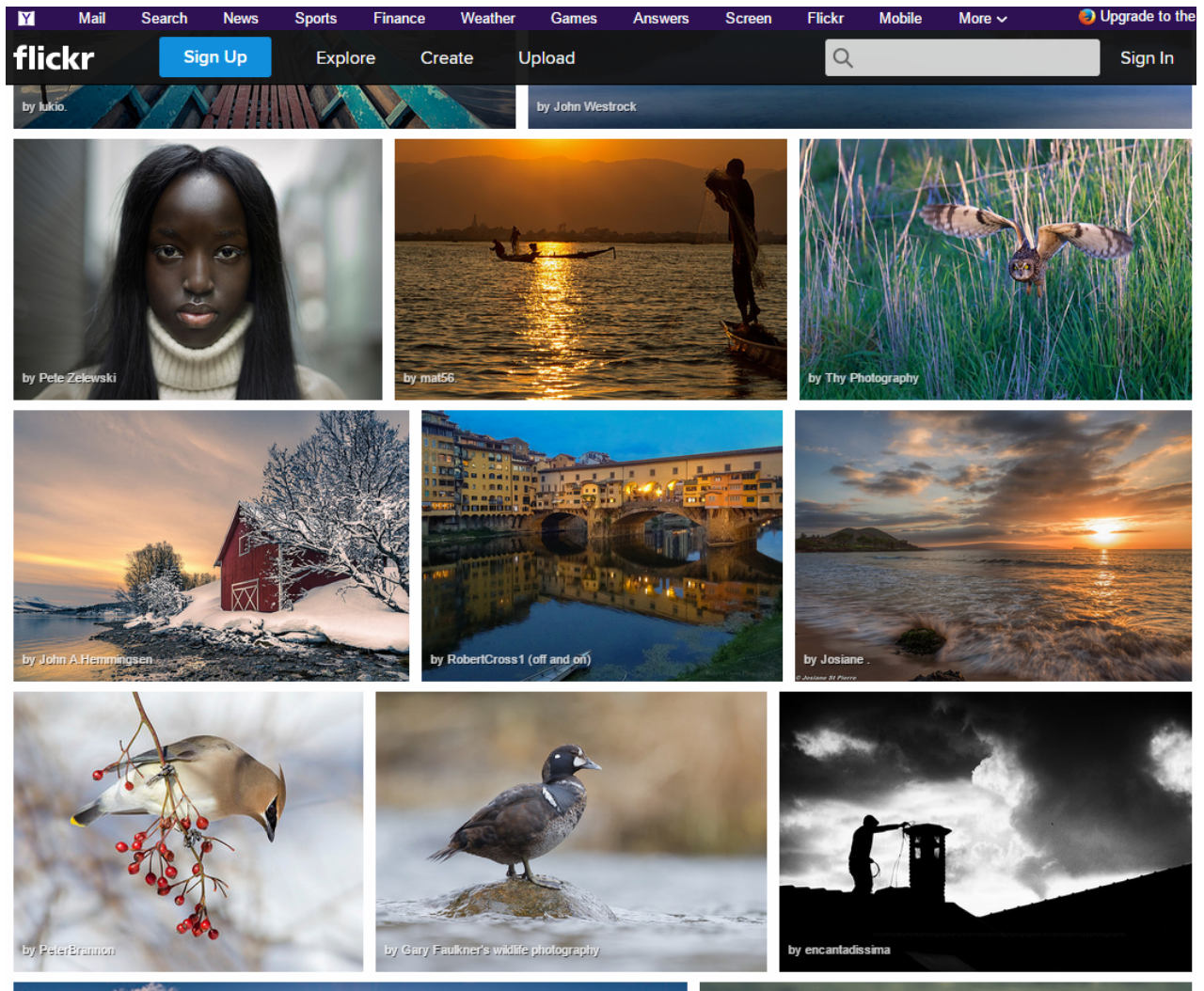
- Free third party content to enrich your android apps
- <http://www.programmableweb.com/category/all/apis>

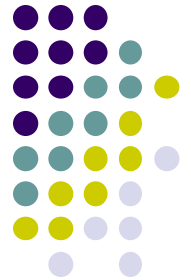
API	Description	Category
Google Maps	Mapping services	Mapping
Twitter	Microblogging service	Social
YouTube	Video sharing and search	Video
Flickr	Photo sharing service	Photos
Amazon eCommerce	Online retailer	Shopping
Facebook	Social networking service	Social
Twilio	Telephony service	Telephony
eBay	eBay 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 available as a web service
- Can access from Android app





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](#)

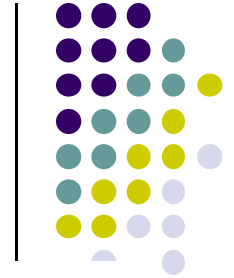
auth

- [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



API Methods

interestingness

- flickr.interestingness.getList

photos

- flickr.photos.addTags
- flickr.photos.delete
- flickr.photos.getAllContexts
- flickr.photos.getContactsPhotos
- flickr.photos.getContactsPublicPhotos
- flickr.photos.getContext
- flickr.photos.getCounts
- flickr.photos.getExif
- flickr.photos.getFavorites
- flickr.photos.getInfo
- flickr.photos.getNotInSet
- flickr.photos.getPerms
- flickr.photos.getRecent
- flickr.photos.getSizes
- flickr.photos.getUntagged
- flickr.photos.getWithGeoData
- flickr.photos.getWithoutGeoData
- flickr.photos.recentlyUpdated
- flickr.photos.removeTag
- flickr.photos.search



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)

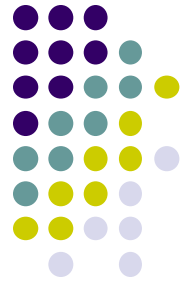
Your API application key. [See here](#) for more details.

user_id (Optional)

The NSID of the user whose 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.

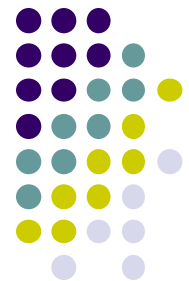
tags (Optional)

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.



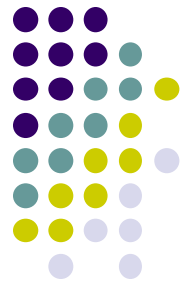
Sample Request URL

- `http://api.flickr.com/services/rest/
?method=flickr.photos.search
&api_key=754a89ad04e0b72f42ffb77f412c021e
&tags=blue,cool,pretty`
- `http://api.flickr.com/services/rest/?method=flick
r.photos.search&api_key=754a89ad04e0b72f42ff
b77f412c021e&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 (まく(まく 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 (まく(まく 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 (まく(まく 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 (まく(まく 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.(jpg)
```

* 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

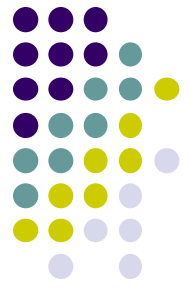
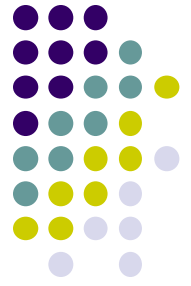


Photo URL and Result

```
<photo id="10925746854" owner="29342764@N04" secret="98695778d3" server="2816"  
farm="3" title="Beaver Moon" ispublic="1" isfriend="0" isfamily="0"/>
```

```
http://farm{farm-id}.staticflickr.com/{server-id}/{id}_{secret}.jpg  
or
```

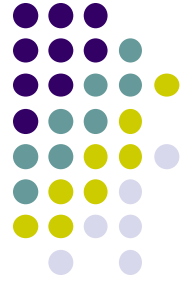
- http://farm3.staticflickr.com/2816/10925746854_98695778d3.jpg



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",
```



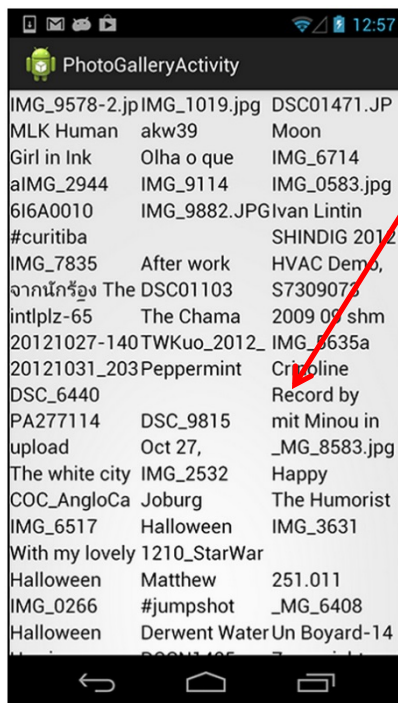
Accessing Web Services using HTTP & Background Tasks

PhotoGallery App

Ref: Android Nerd Ranch Chapter 26



- Introduces networking in Android
- Android app is client for **Flickr** photo-sharing site
- **Version 1:** Retrieve photos from Flickr using HTTP, display captions



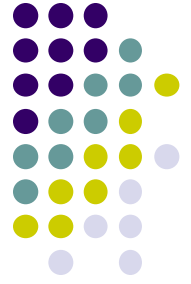
**Final
Version**



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

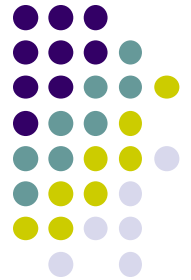
- Need to:
 - Create GridView layout in XML and inflate it
 - Create a data source (GridView is an AdapterView)
- Create a GridView

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;

...

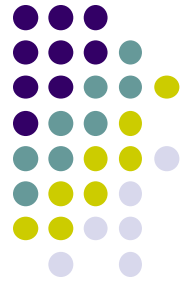
public class PhotoGalleryFragment extends Fragment {
    GridView mGridView;

    @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;
    }
}
```



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;

...

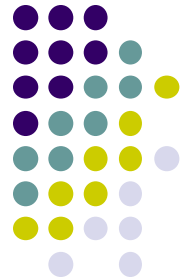
public class FlickrFetchr {
    byte[] getUrlBytes(String urlSpec) throws IOException {
        URL url = new URL(urlSpec);
        HttpURLConnection connection = (HttpURLConnection)url.openConnection();

        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));
    }
}
```



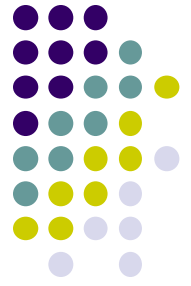
Ask App User's Permission to Network

- Modify **AndroidManifest.xml** to ask app user for permission to use 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" >

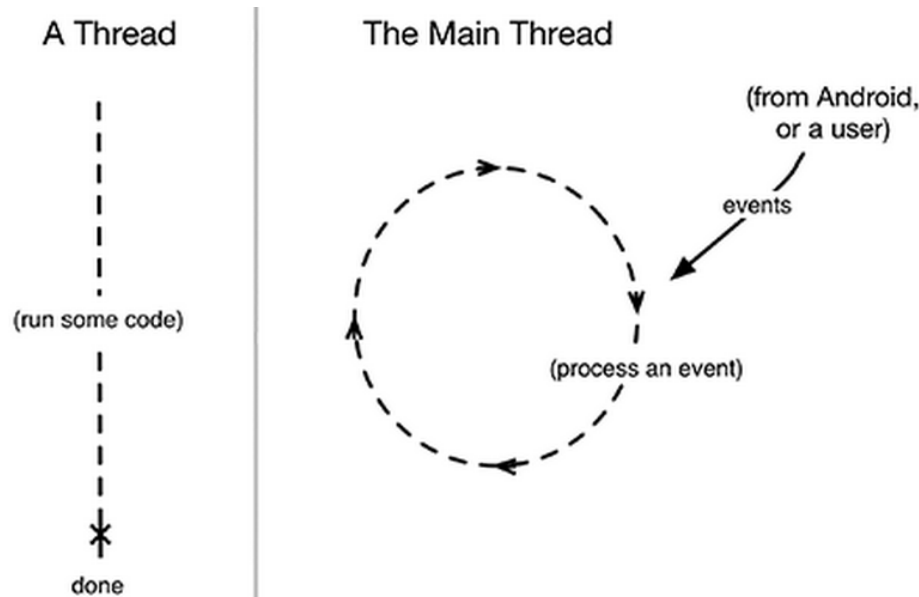
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="15" />
    <uses-permission android:name="android.permission.INTERNET" />

    ...
```



No Networking in Main Thread

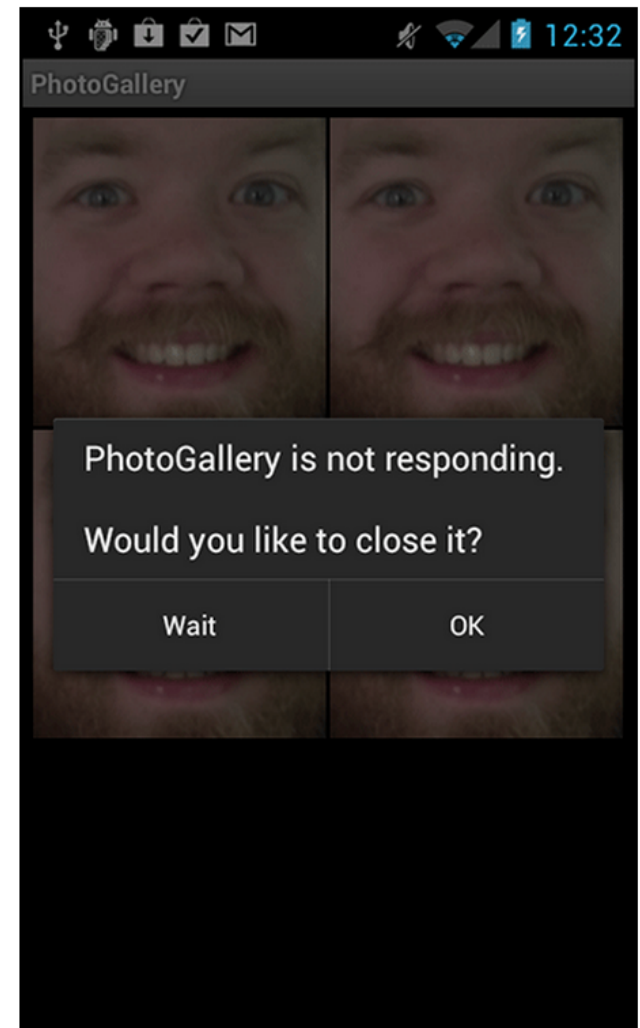
- Networking can take a long time (download, etc)
- Main thread needs to be free to respond to events
- Android throws **NetworkOnMainThreadException** if networking code is put in main thread
- Hence we network in separate background thread using **AsyncThread**





What Happens if Main Thread is Busy?

- If Flickr download takes a long time, we get **Application Not Responding**



Use AsyncTask to Fetch on Background Thread



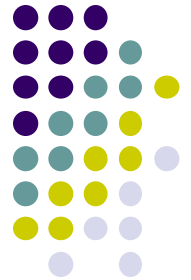
- Doing networking on main thread will freeze user interaction
- Utility class **AsyncTask** creates background thread
 - Runs code in **doInBackground(..)** method on that thread
- Add new inner class called **FetchItemsTask** to **PhotoGalleryFragment**

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

    GridView mGridView;

    ...

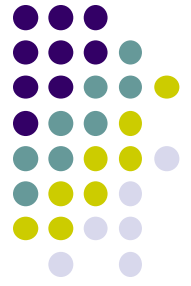
    private class FetchItemsTask extends AsyncTask<Void,Void,Void> {
        @Override
        protected Void doInBackground(Void... params) {
            try {
                String result = new FlickrFetchr().getUrl("http://www.google.com");
                Log.i(TAG, "Fetched contents of URL: " + result);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



Execute FetchItemsTask

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

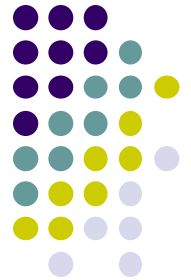
```
public class PhotoGalleryFragment extends Fragment {  
    private static final String TAG = "PhotoGalleryFragment";  
  
    GridView mGridView;  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
  
        setRetainInstance(true);  
        new FetchItemsTask().execute();  
    }  
  
    ...  
}
```



Fetching XML from Flickr

- Flickr offer XML API, some simple through **REST API**
- Details at: www.flickr.com/services/api/
- **getRecent** method returns list of latest public photos on Flickr
- Requires getting an API key
- First add some constants to our class **FlickrFetchr**

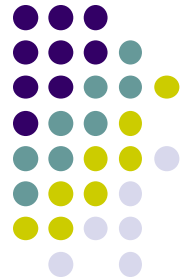
```
public class FlickrFetchr {  
    public static final String TAG = "FlickrFetchr";  
  
    private static final String ENDPOINT = "http://api.flickr.com/services/rest/";  
    private static final String API_KEY = "yourApiKeyHere";  
    private static final String METHOD_GET_RECENT = "flickr.photos.getRecent";  
    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;  
    }  
}
```

Create GalleryItem() Class

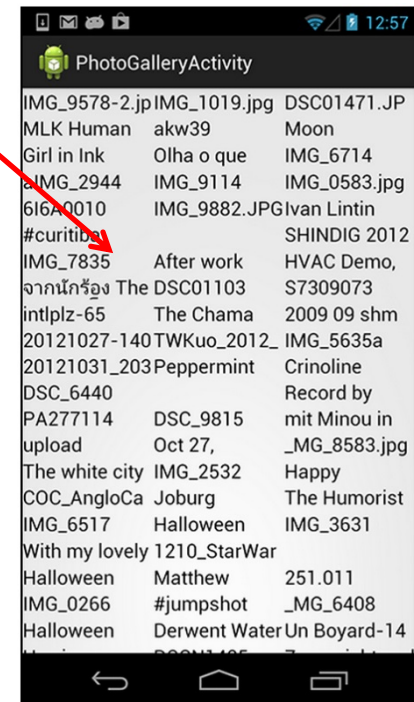
- Each item in the GridView is a **GalleryItem**

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

public class GalleryItem {
    private String mCaption;
    private String mId;
    private String mUrl;

    public String toString() {
        return mCaption;
    }
}
```

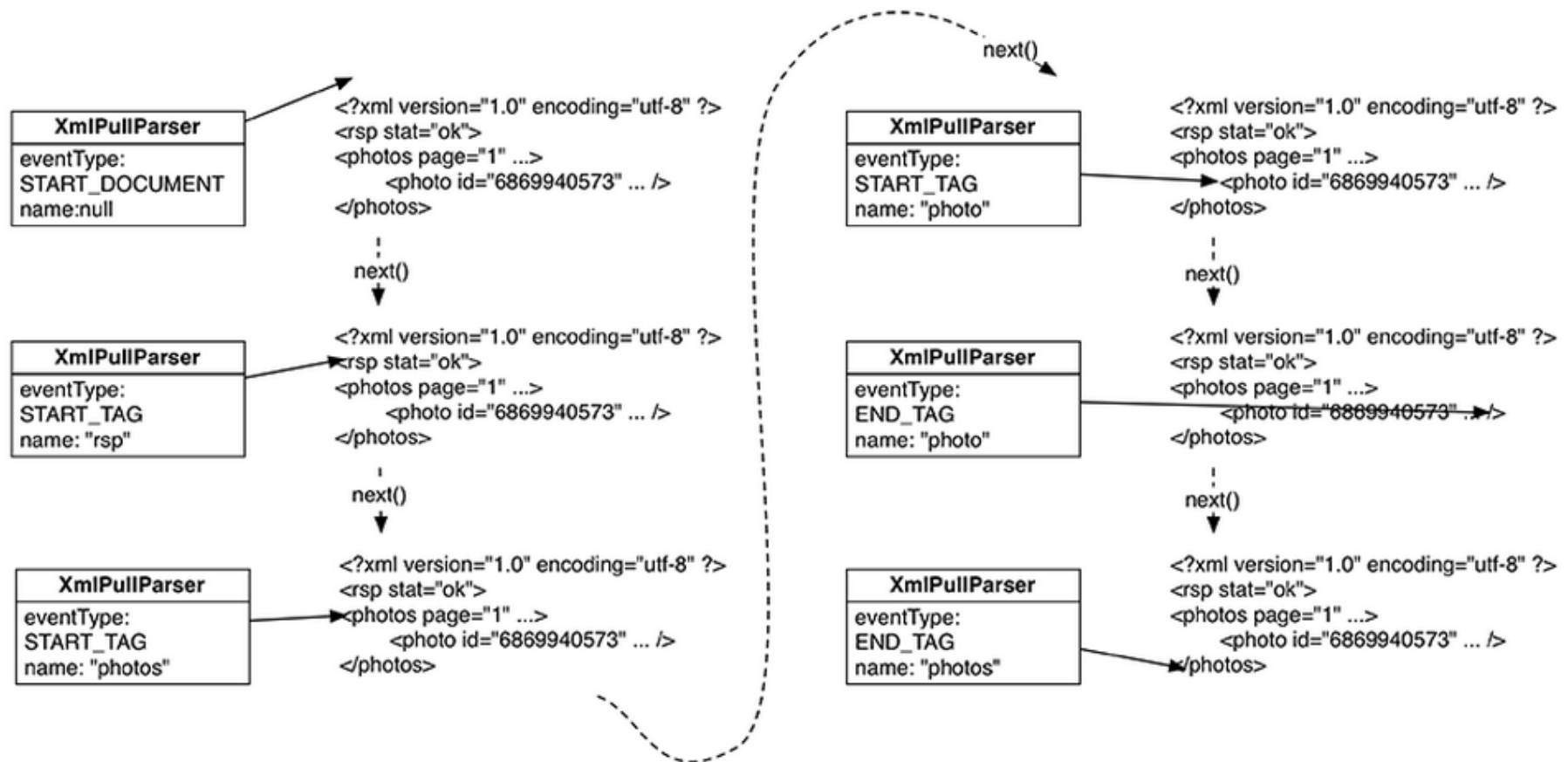
- Remember:** Version 1 displays only captions
- Next, use interface **XMLPullParser**, to pull **GalleryItem** objects

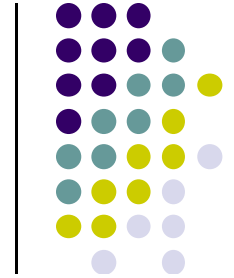




How XmlPullParser Works

- Walks line by line through XML stream, interpreting tags





Parse Flickr Photos

- Write **parseItems** method to make **GalleryItem** of each photo caption
- Add photo to **ArrayList**
- Uses **XMLPullParser**

```
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 = "your API key";
    private static final String METHOD_GET_RECENT = "flickr.photos.getRecent";

    private static final String XML_PHOTO = "photo";

    ...

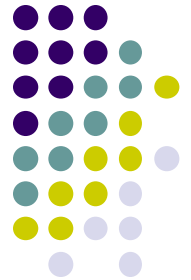
    public void fetchItems() {
        ...

        void parseItems(ArrayList<GalleryItem> items, XmlPullParser parser)
            throws XmlPullParserException, IOException {
            int eventType = parser.next();

            while (eventType != XmlPullParser.END_DOCUMENT) {
                if (eventType == XmlPullParser.START_TAG &&
                    XML_PHOTO.equals(parser.getName())) {
                    String id = parser.getAttributeValue(null, "id");
                    String caption = parser.getAttributeValue(null, "title");
                    String smallUrl = parser.getAttributeValue(null, EXTRA_SMALL_URL);

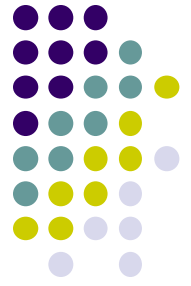
                    GalleryItem item = new GalleryItem();
                    item.setId(id);
                    item.setCaption(caption);
                    item.setUrl(smallUrl);
                    items.add(item);
                }

                eventType = parser.next();
            }
        }
    }
}
```



Modify fetchItems() to Return ArrayList

```
public void fetchItems() {  
public ArrayList<GalleryItem> fetchItems() {  
    ArrayList<GalleryItem> items = new ArrayList<GalleryItem>();  
  
    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);  
        XmlPullParserFactory factory = XmlPullParserFactory.newInstance();  
        XmlPullParser parser = factory.newPullParser();  
        parser.setInput(new StringReader(xmlString));  
  
        parseItems(items, parser); ← Parse into ArrayList here  
    } catch (IOException ioe) {  
        Log.e(TAG, "Failed to fetch items", ioe);  
    } catch (XmlPullParserException xppe) {  
        Log.e(TAG, "Failed to parse items", xppe);  
    }  
    return items;  
}
```



Return to Main Thread, Display Captions

- Set up ArrayAdapter that uses simple Android layout

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

    GridView mGridView;
    ArrayList<GalleryItem> mItems;

    ...

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

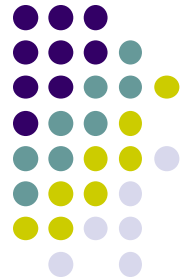
        mGridView = (GridView)v.findViewById(R.id.gridView);

        setupAdapter();

        return v;
    }

    void setupAdapter() {
        if (getActivity() == null || mGridView == null) return;

        if (mItems != null) {
            mGridView.setAdapter(new ArrayAdapter<GalleryItem>(getActivity(),
                android.R.layout.simple_gallery_item, mItems));
        } else {
            mGridView.setAdapter(null);
        }
    }
}
```



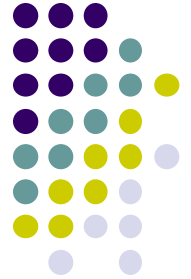
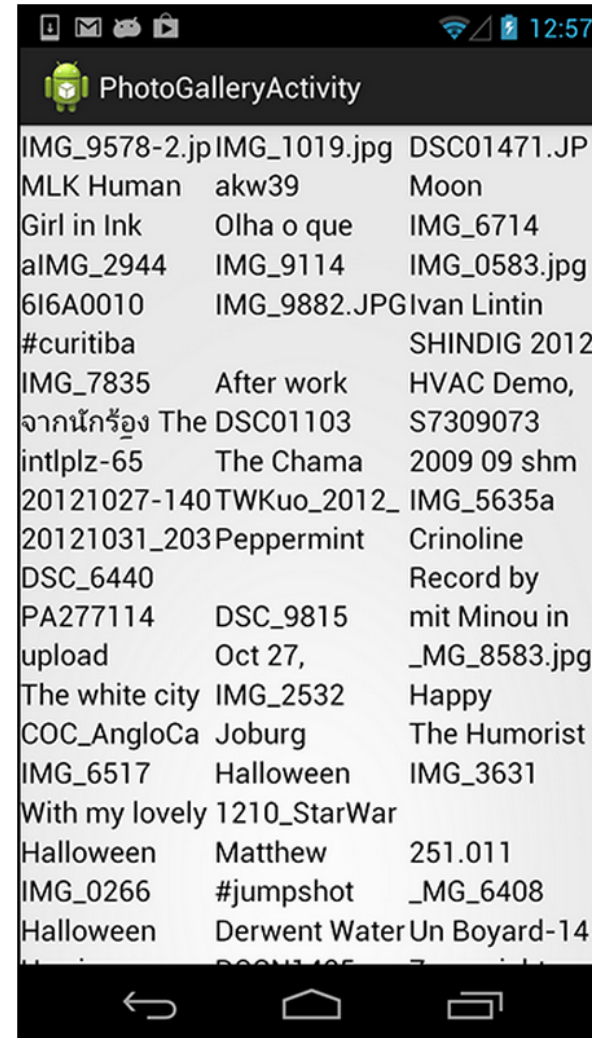
Use onPostExecute() to update GridView

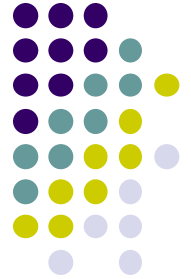
- Better to update **GridView's** adapter by overriding another method of **AsyncTask** called **onPostExecute()**
- **onPostExecute()** run after **doInBackground()**

```
private class FetchItemsTask extends AsyncTask<Void,Void,Void>{  
private class FetchItemsTask extends AsyncTask<Void,Void,ArrayList<GalleryItem>> {  
    @Override  
    protected Void doInBackground(Void... params){  
    protected ArrayList<GalleryItem> doInBackground(Void... params) {  
        new FlickrFetchr().fetchItems();  
        return new FlickrFetchr().fetchItems();  
        return null;  
    }  
  
    @Override  
    protected void onPostExecute(ArrayList<GalleryItem> items) {  
        mItems = items;  
        setupAdapter();  
    }  
}
```

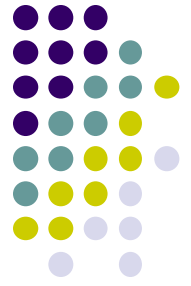
Results?

- Flickr Item Captions displayed



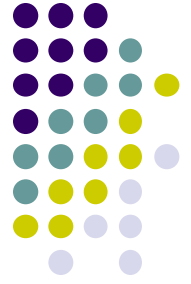


Broadcast Receivers



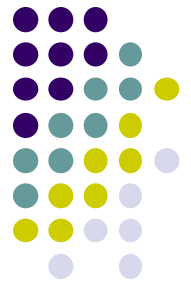
Broadcast Receivers

- 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
- Application component creates and registers a Broadcast Receiver
- Can be done in XML or in Java code

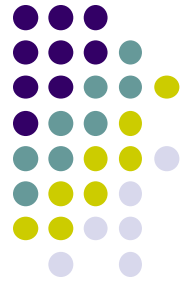


Declaring Broadcast Receiver in AndroidManifest.xml

```
<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" />

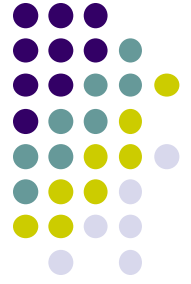
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>

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



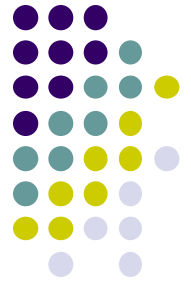
Broadcast Receivers

- Apps can initiate broadcasts to inform other applications of status or readiness
- Broadcast Receivers don't display UI
 - may create status bar notifications
- Typically used as gateway by other components and does very minimal work
 - initiate service based on some event
- Broadcasts are delivered to interested apps as Intents



Broadcast Receivers

- intents sent using **sendBroadcast()** method
- Can use **LocalBroadcastManager** to send Broadcasts within your application only



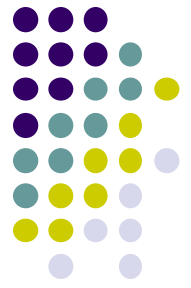
BroadcastReceivers

- 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



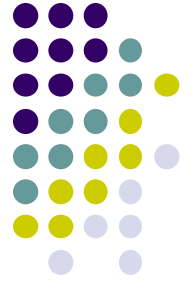
String	<code>ACTION_CAMERA_BUTTON</code>	Broadcast Action: The "Camera Button" was pressed.
String	<code>ACTION_CHOOSER</code>	Activity Action: Display an activity chooser, allowing the user to pick what they want to before proceeding.
String	<code>ACTION_CLOSE_SYSTEM_DIALOGS</code>	Broadcast Action: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



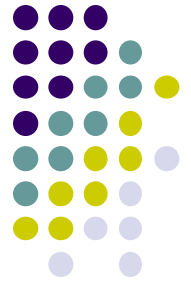
Broadcasts

- 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.suppliment.CONNECTION_CHANGE
android.net.wifi.suppliment.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
```

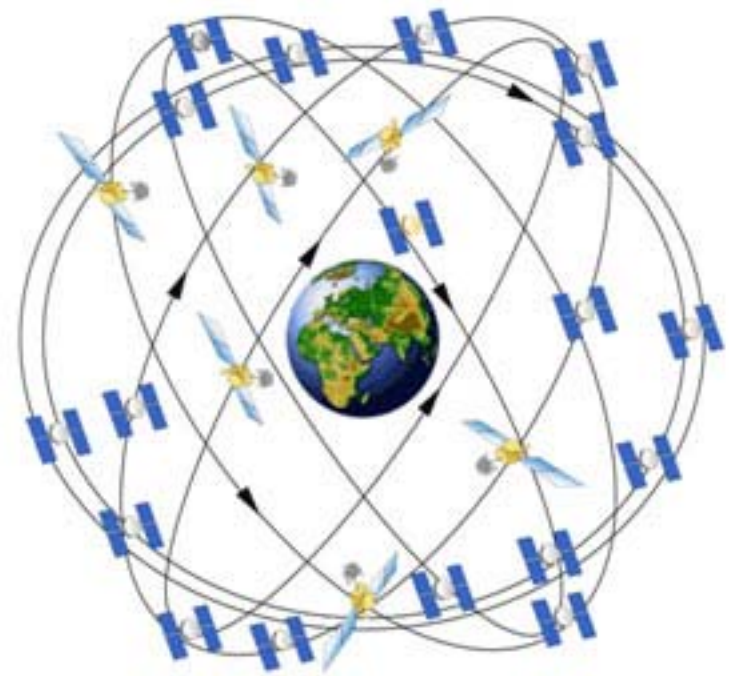


Tracking the Device's Location



Global Positioning System (GPS)

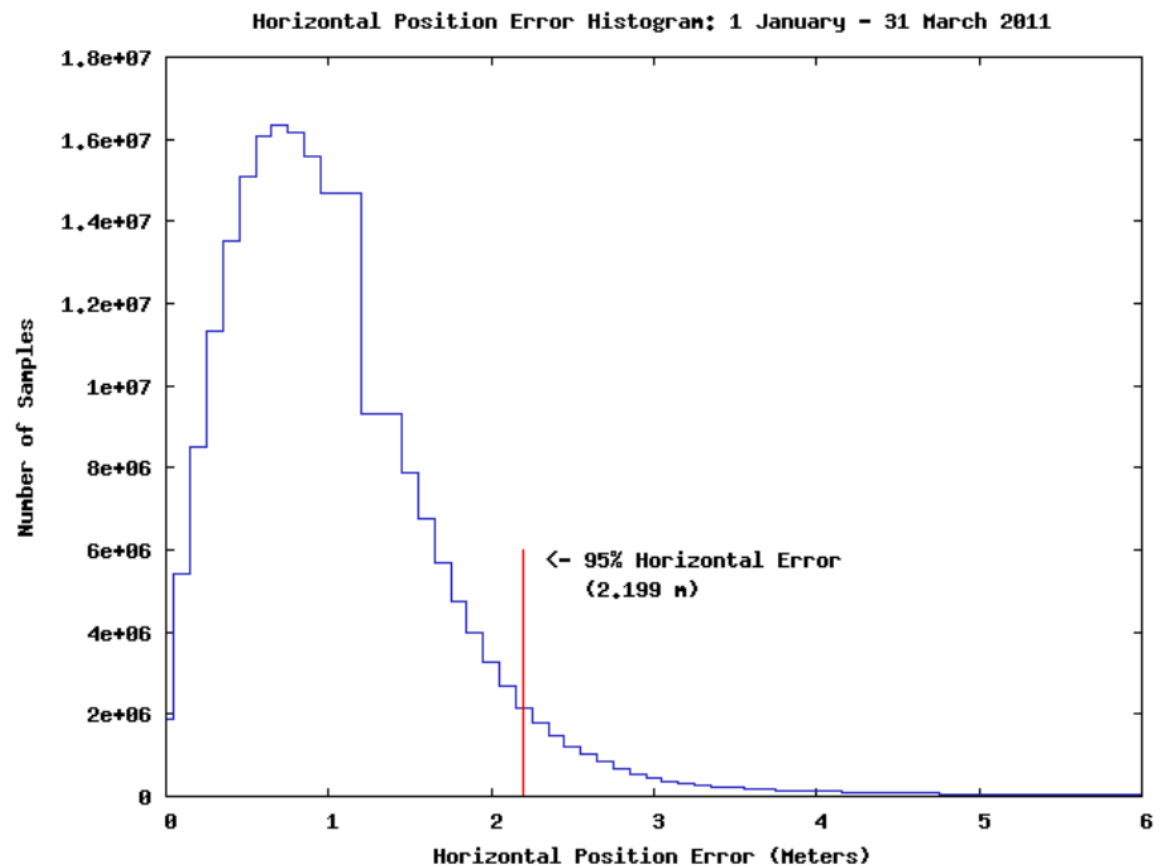
- 24 core satellites
- medium earth orbit, 20,000 km above earth
- 6 orbital planes with 4 satellites each
- 4 satellites visible from any spot on earth
- Recently upgraded to 27 sats



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

Ref: Android Nerd Ranch Chapter 33



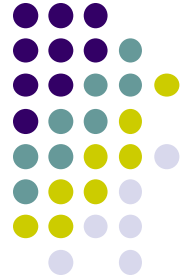
- **Goal:** Create new application called **runTracker**
- **runTracker** uses phone's GPS to record and display user's travels (walking in woods, car ride, ocean voyage, etc)
- First version of **runTracker** gets GPS updates, displays current location on screen
- **Later:** Show map that follows user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



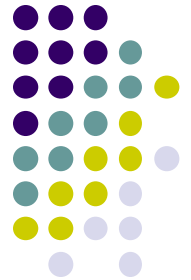


Add Strings to Strings.xml

- 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>
```



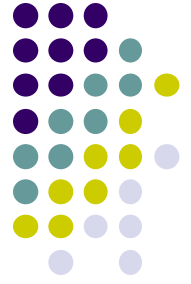


Create RunFragment

- 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 alternative 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 mContext;
    private LocationManager mLocationManager;

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

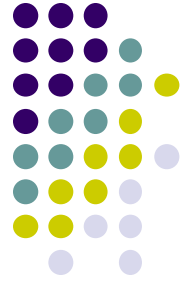
    public void stopLocationUpdates() {
        PendingIntent pi = getLocationPendingIntent(false);
        if (pi != null) {
            mLocationManager.removeUpdates(pi);
            pi.cancel();
        }
    }

    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

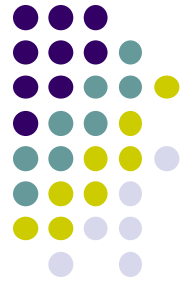
Receiving Broadcast Location Updates



- 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

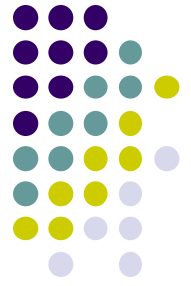
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.runtracker"
    android:versionCode="1"
    android:versionName="1.0">

    <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();
            }
        });

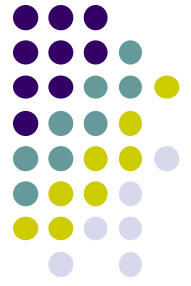
        updateUI();

        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



Displaying Updates on Screen

- Previous example displayed results as log in IDE
- To get results on app screen, implement **Run** class
- Stores start data, computes run duration

```
public class Run {
    private Date mStartDate;

    public Run() {
        mStartDate = new Date();
    }

    public Date getStartDate() {
        return mStartDate;
    }

    public void setStartDate(Date startDate) {
        mStartDate = startDate;
    }

    public int getDurationSeconds(long endMillis) {
        return (int)((endMillis - mStartDate.getTime()) / 1000);
    }

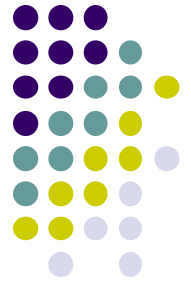
    public static String formatDuration(int durationSeconds) {
        int seconds = durationSeconds % 60;
        int minutes = ((durationSeconds - seconds) / 60) % 60;
        int hours = (durationSeconds - (minutes * 60) - seconds) / 3600;
        return String.format("%02d:%02d:%02d", hours, minutes, seconds);
    }
}
```

Update RunFragment



- New instance of **Run** is created to record a run and last location received
- Toast created to display when GPS provider is enabled or disabled

```
public class RunFragment extends Fragment {  
  
    private BroadcastReceiver mLocationReceiver = new LocationReceiver() {  
  
        @Override  
        protected void onLocationReceived(Context context, Location loc) {  
            mLastLocation = loc;  
            if (isVisible())  
                updateUI();  
        }  
  
        @Override  
        protected void onProviderEnabledChanged(boolean enabled) {  
            int toastText = enabled ? R.string.gps_enabled : R.string.gps_disabled;  
            Toast.makeText(getActivity(), toastText, Toast.LENGTH_LONG).show();  
        }  
  
    };  
  
    private RunManager mRunManager;  
  
    private Run mRun;  
    private Location mLastLocation;  
  
    private Button mStartButton, mStopButton;  
  
    ...  
  
    @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();  
                mRun = new Run();  
                updateUI();  
            }  
        });  
  
        ...  
    }  
}
```



Update RunFragment

- onStart, onStop used to register, unregister receiver when app is visible/invisible to user
- Initialize **Run** as soon as location updates start

```
@Override
public void onStart() {
    super.onStart();
    getActivity().registerReceiver(mLocationReceiver,
        new IntentFilter(RunManager.ACTION_LOCATION));
}
```

```
@Override
public void onStop() {
    getActivity().unregisterReceiver(mLocationReceiver);
    super.onStop();
}
```

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

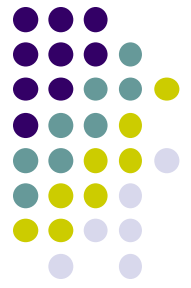
```
    if (mRun != null)
        mStartedTextView.setText(mRun.getStartDate().toString());

    int durationSeconds = 0;
    if (mRun != null && mLastLocation != null) {
        durationSeconds = mRun.getDurationSeconds(mLastLocation.getTime());
        mLatitudeTextView.setText(Double.toString(mLastLocation.getLatitude()));
        mLongitudeTextView.setText(Double.toString(mLastLocation.getLongitude()));
        mAltitudeTextView.setText(Double.toString(mLastLocation.getAltitude()));
    }
    mDurationTextView.setText(Run.formatDuration(durationSeconds));
```

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

```
}
```

```
}
```



Fast Updates with Last Known Location

- Sometimes communication with satellite slow to get updated location
- May just request last known location using **getLastKnownLocation**

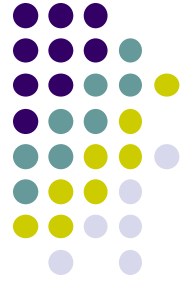
```
public void startLocationUpdates() {
    String provider = LocationManager.GPS_PROVIDER;

    // Get the last known location and broadcast it if you have one
    Location lastKnown = mLocationManager.getLastKnownLocation(provider);
    if (lastKnown != null) {
        // Reset the time to now
        lastKnown.setTime(System.currentTimeMillis());
        broadcastLocation(lastKnown);
    }

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

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

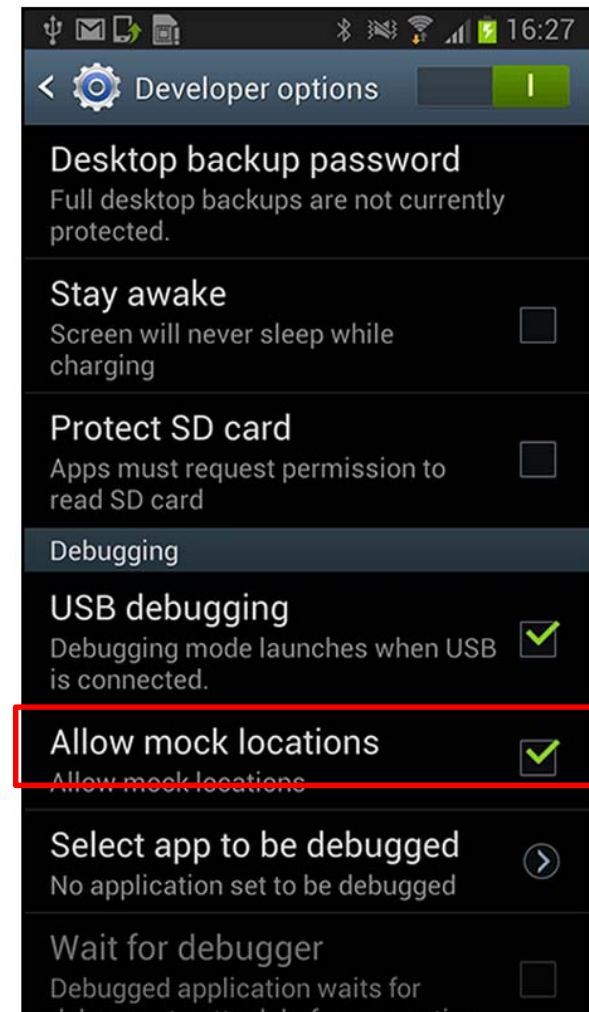
Testing Locations on Real vs Virtual Devices

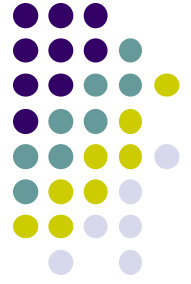


- Can be tricky to test if locations received are correct
- Can test by manually setting **LocationManager** locations in emulator (either one by one or series of locations in a file)
- Steps to test on a real device a bit more difficult
 1. Request the `ACCESS_MOCK_LOCATION` permission.
 2. Add a test 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 provided by Book (Android Nerd Ranch)

Allow Mock Locations

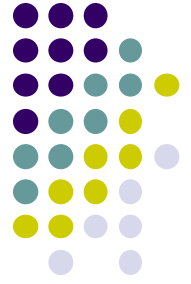
- For **TestProvider** class provided by text to work, turn on **Allow mock locations** in **Developer options** menu



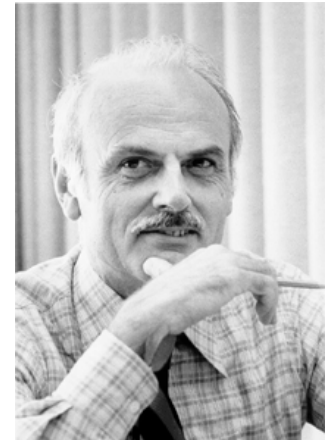


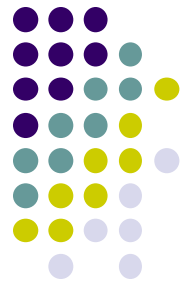
Local Databases with SQLite

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

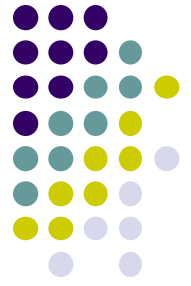
Winery Table

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

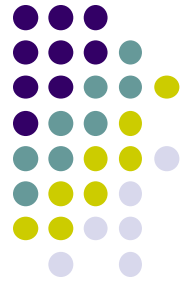
- Data in different tables can be related

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

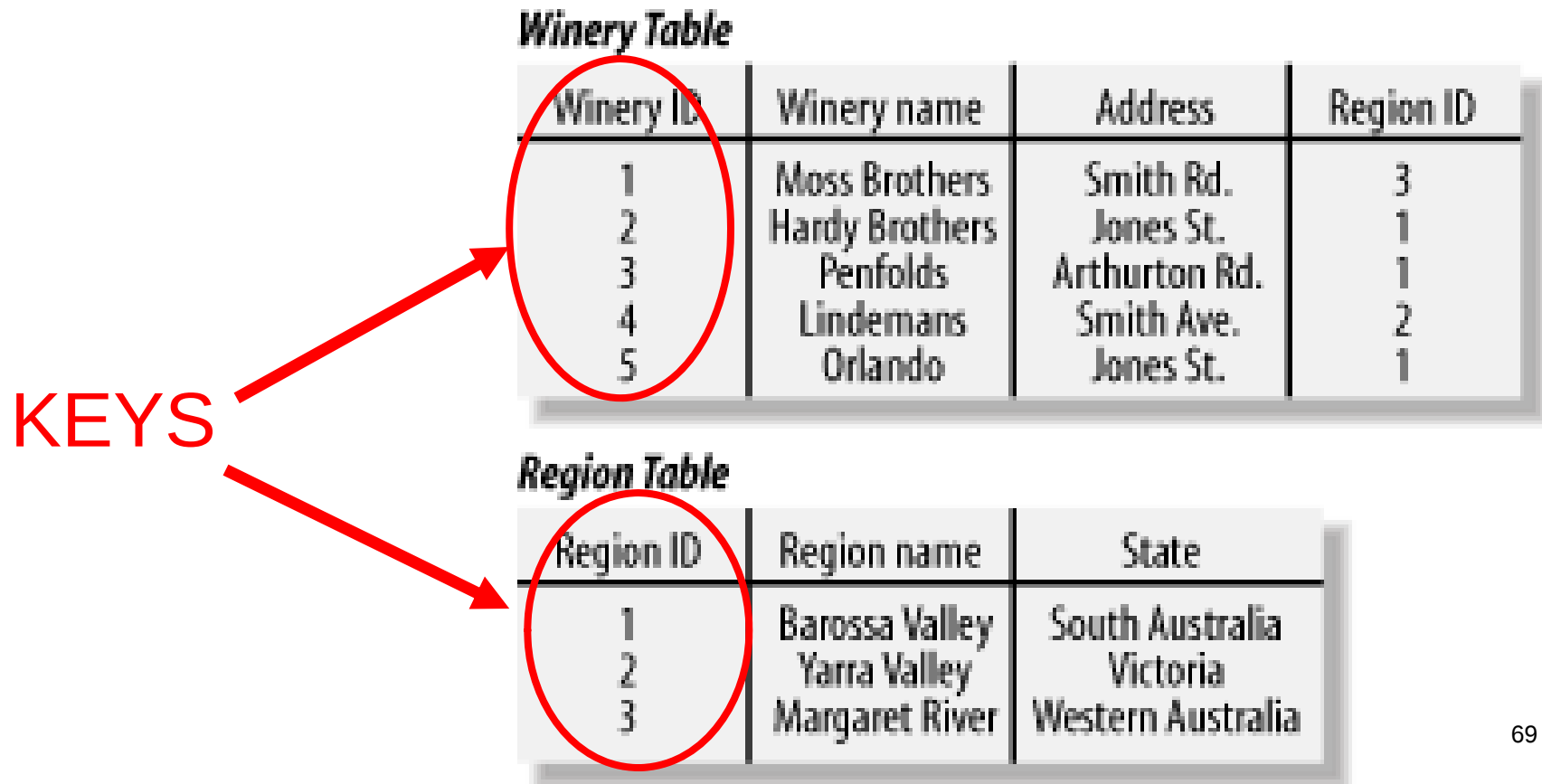
Region Table

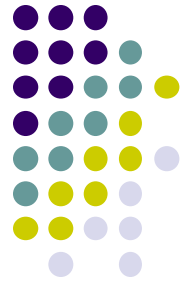
Region ID	Region name	State
1	Barossa Valley	South Australia
2	Yarra Valley	Victoria
3	Margaret River	Western Australia



Keys

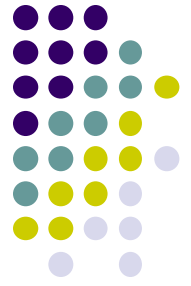
- Each table has a key
- Column used to uniquely identify each row





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

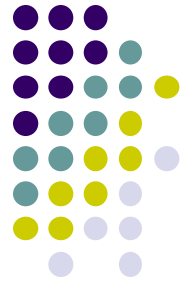


SQL Commands

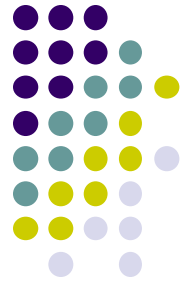
- **SELECT** - get data from a database table
 - **UPDATE** - change data in a database table
 - **DELETE** - remove data from a database table
 - **INSERT INTO** - insert new data in a database table
-
- SQLite implements most, but not all of SQL
 - <http://www.sqlite.org/>

Why Local Databases?

Ref: Android Nerd Ranch Chapter 34

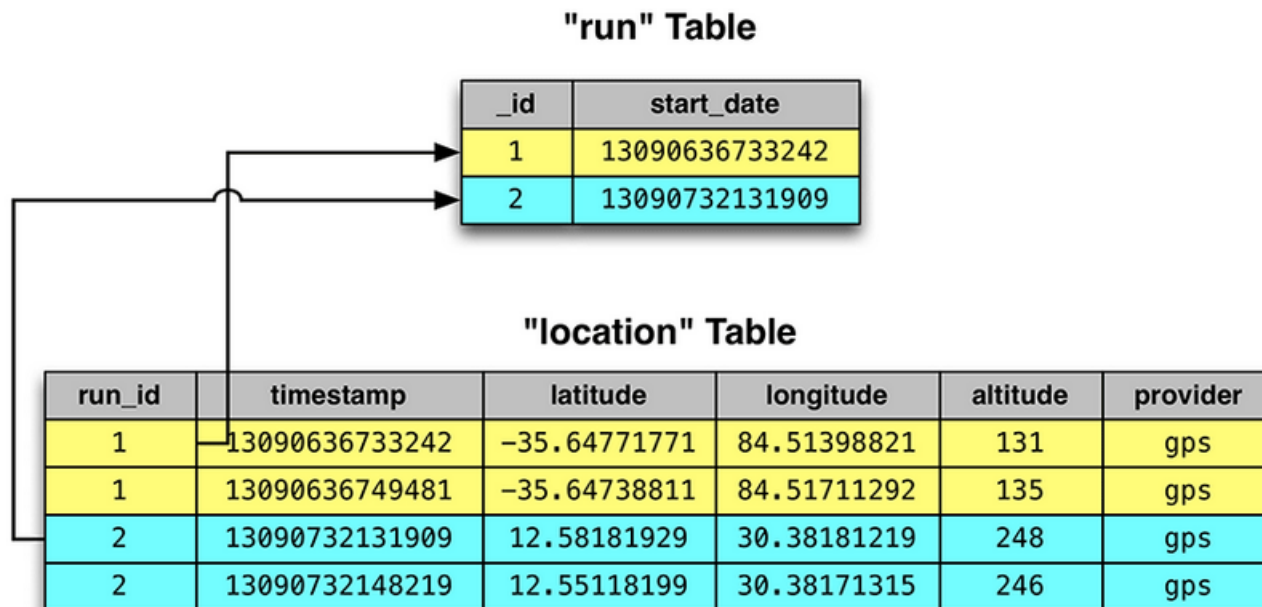


- User may track their runs forever
- Lots of data
- **Solution:** Store runTracker runs and locations in SQLite database
- **SQLite** is open source relational database
- **SQLiteOpenHelper** encapsulates database creation, opening and updating
- In **runTracker**, create subclass of **SQLiteOpenHelper**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





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 constructor
- 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;

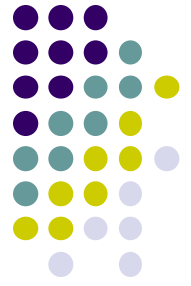
    private static final String TABLE_RUN = "run";
    private static final String COLUMN_RUN_START_DATE = "start_date";

    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 massge 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 **RunManager** to use new database

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

    private static final String PREFS_FILE = "runs";
    private static final String PREF_CURRENT_RUN_ID = "RunManager.currentRunId";

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

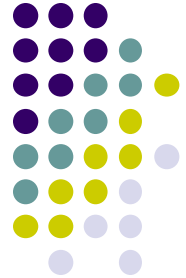
    private static final String TEST_PROVIDER = "TEST_PROVIDER";

    private static RunManager sRunManager;
    private Context mContext;
    private LocationManager locationManager;
    private RunDatabaseHelper mHelper;
    private SharedPreferences mPrefs;
    private long mCurrentRunId;

    private RunManager(Context appContext) {
        mContext = appContext;
        locationManager = (LocationManager)mContext
            .getSystemService(Context.LOCATION_SERVICE);
        mHelper = new RunDatabaseHelper(mContext);
        mPrefs = mContext.getSharedPreferences(PREFS_FILE, Context.MODE_PRIVATE);
        mCurrentRunId = mPrefs.getLong(PREF_CURRENT_RUN_ID, -1);
    }

    ...
}
```

Modify RunManager to Use New Database



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)

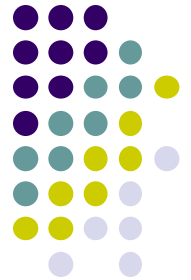
```
private void broadcastLocation(Location location) {
    Intent broadcast = new Intent(ACTION_LOCATION);
    broadcast.putExtra(LocationManager.KEY_LOCATION_CHANGED, location);
    mContext.sendBroadcast(broadcast);
}

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);

    ...

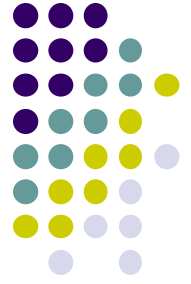
    mStartButton = (Button)view.findViewById(R.id.run_startButton);
    mStartButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mRunManager.startLocationUpdates();
            mRun = new Run();
            mRun = mRunManager.startNewRun();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

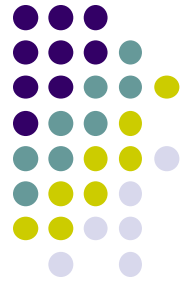


Inserting a Location for Current Run

- Add code to **RunManager** to insert a location for currently tracking run

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

public void insertLocation(Location loc) {
    if (mCurrentRunId != -1) {
        mHelper.insertLocation(mCurrentRunId, loc);
    } else {
        Log.e(TAG, "Location received with no tracking run; ignoring.");
    }
}
}
```

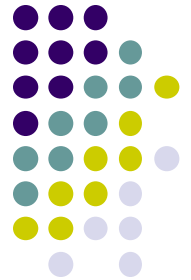


Continuously Handle Location Intents

- 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



- Make it execute in response to ACTION_LOCATION

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

    ...

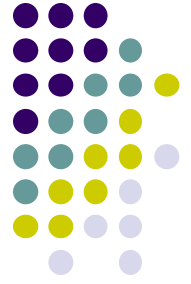
    <receiver android:name=".LocationReceiver"
    <receiver android:name=".TrackingLocationReceiver"
        android:exported="false">
        <intent-filter>
            <action android:name="com.bignerdranch.android.runtracker.ACTION_LOCATION"/>
        </intent-filter>
    </receiver>

</application>
```



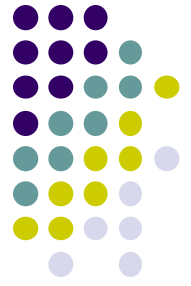
More Details in Book

- 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