Android Programming
Lecture 9:
Two New Views

9/30/2011
ListView View

- Using ListViews is very much like using Spinners
  - Build off an array of data
  - Events on the list happen at a particular position

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    
    <TextView android:id="@+id/text_view"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Favorite team:"
    />

    <ListView android:id="@+id/list_view"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
    />
</LinearLayout>
```
ListView View

```java
public class ListViewExampleActivity extends Activity implements AdapterView.OnItemClickListener {

    TextView textView;
    ListView listView;
    Resources resources;
    String[] acc_teams;
    ArrayAdapter<String> adapter;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        textView = (TextView) findViewById(R.id.text_view);
        listView = (ListView) findViewById(R.id.list_view);
        resources = getResources();
        acc_teams = resources.getStringArray(R.array.acc_teams);
        adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, acc_teams);
        listView.setAdapter(adapter);

        listView.setOnItemClickListener(this);
    }

    public void onItemClick(AdapterView<?> parent, View view, int pos, long id) {
        Log.v("LVA", "+pos);
        textView.setText("Favorite team: " + acc_teams[pos]);
    }
}
```

Employs `OnItemClickListener` with `onItemClick` method instead of `OnItemSelectedListener` (OnItemSelectedListener works on lists, but returns when arrow keys are moving you up & down entries in list)
MapView

- GoogleMaps
- Supports zooming
- Supports satellite mode
- Supports overlays (route drawing, markers)

- First things first: Ask Google to use it!
MapView

- Google will give you a map key in exchange for your application signing key
  - Anything published in the Android Market has to have a signing key
    - Proof of authorship
  - A separate debug key is automatically setup by Eclipse
    - Can’t use debug key when publishing in Market
    - We will use this for now!!!
    - Also requires a Google account
Getting a Map Key

Getting the MD5 Fingerprint of the SDK Debug Certificate

While you are developing and debugging your application, you will likely be signing your application in debug mode – that is, the SDK build tools will automatically sign your application using the debug certificate. To let your MapView elements properly display Maps data during this period, you should obtain a temporary Maps API Key registered to the debug certificate. To do so, you first need to get the MD5 fingerprint of the debug certificate. When you are ready to release your application, you must register your release certificate with the Google Maps service and obtain a new Maps API Key. You must then change the MapView elements in your application to reference the new API key.

To generate an MD5 fingerprint of the debug certificate, first locate the debug keystore. By default, build tools create the debug keystore in the active AVD directory. The location of the AVD directories varies by platform:

- Windows Vista: C: \Users\<user>\.android\debug.keystore
- Windows XP: C: \Documents and Settings\<user>\.android\debug.keystore
- OS X and Linux: ~/.android/debug.keystore

If you are using Eclipse/ADT and are unsure where the debug keystore is located, you can select Windows > Pref > Android > Build to check the full path, which you can then paste into a file explorer to locate the directory containing the keystore.

Once you have located the keystore, use this Keytool command to get the MD5 fingerprint of the debug certificate:

```
keytool -list -alias androiddebugkey 
-keystore <path_to_debug_keystore>.keystore 
-storepass android -keypass android
```

keytool is in your Java SDK bin directory

Registering the Certificate Fingerprint with the Google Maps Service

When you are ready to register for a Maps API Key, load this page in a browser:

http://code.google.com/android/maps-api-signup.html

To register for a Maps API Key, follow these steps:

1. If you don’t have a Google account, use the link on the page to set one up.
2. Read the Android Maps API Terms of Service carefully. If you agree to the terms, indicate so using the checkbox on the screen.
3. Paste the MD5 certificate fingerprint of the certificate that you are registering into the appropriate form field.
4. Click “Generate API Key”

The server will handle your request, associating the fingerprint with your developer identity and generating a unique Maps API Key, and then will return a results page that gives you your Key string.

To use the Maps API Key string, copy and paste it into your code as described in the next section.

Note: If you happen to forget your Maps API Key, you can generate a fingerprint for your certificate and register it again. The registration server will give you the same Maps API Key for the specified certificate fingerprint.

http://code.google.com/android/add-ons/google-apis/mapkey.html, pg 261 of textbook
MapView

Updating Application Manifest

When you create a new Android project, make sure you choose the Google API version of a Build Target:

May also need to setup your emulator (or create a new one) to support Google API.

In app manifest (above right [AndroidManifest.xml]), need to indicate:

- Use of Google maps library (as part of application description in manifest)
- Requires permission to use the Internet (as part of manifest)
Using a Map Key: Adding View to Layout

MapViews can only be used within a MapActivity, so you should extend MapActivity.

```java
public class StartActivity extends MapActivity {
    // Code
}
```

```xml
<com.google.android.maps.MapView android:id="@+id/map_view"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:clickable="true"
    android:apiKey="0pDANALRKj2KTWyQhmpkm0y3IdwflwHRElsKdMA"/>
```

Clickable allows direct interaction with map (zooming/etc) – off by default
You can use this map apiKey in multiple apps.
MapActivity

• You **have** to implement two functions as a subclass of MapActivity
  – They were left “abstract” in MapActivity
  – They are for Google accounting purposes, asking:
    • `boolean isRouteDisplayed()`
      Is route information being shown as part of the map?
    • `boolean isLocationDisplayed()`
      Is location information being shown as part of the map?
  • Implement these with `return true` or `return false` as appropriate for your app
    – See implementation on next few slides
MapView

MapView has functions to allow setting and querying general map appearance features

Setting the appearance:

- void setSatellite(boolean) – turn satellite mode on/off
- void setStreetView(boolean) – turn on street view blue lines
  – Deprecated (doesn’t appear to work)?
- void setTraffic(boolean) – turn on traffic estimate lines
  – Traffic and StreetView are mutually exclusive

Query the appearance:

- boolean isSatellite(), boolean isStreetView(), boolean isTraffic()
- GeoPoint getMapCenter()

Allow built-in (touch sensitive) zoom controls:

- void setBuiltInZoomControls(boolean)
MapView

```java
package turkett.csc191;

import com.google.android.maps.MapActivity;

public class ExampleMapActivityActivity extends MapActivity {

    MapView mapView;
    TextView textView;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mapView = (MapView) findViewById(R.id.map_view);
        textView = (TextView) findViewById(R.id.text_view);

        mapView.setTraffic(true);
        mapView.setBuiltInZoomControls(true);

        textView.setText("Map: satelliteOn=\"+mapView.isSatellite() +
                       \" streetViewOn=\"+mapView.isStreetView() + \" trafficOn=\"+mapView.isTraffic());

        protected boolean isRouteDisplayed() { return false; }
        protected boolean isLocationDisplayed() { return false; }
    }
```
MapView and MapController

To change the location being displayed, need access to the MapView’s `MapController`.

```java
MapController mapController = mapViewHere.getController();
```

![Diagram showing the relationship between Model, View, Controller, and data flow](image-url)

- **Data**: Read data
- **View**: Display (can read data)
- **Controller**: User interaction, Requests to modify data or change view
GeoPoints

MapView deals primarily with GeoPoints

```
com.google.android.maps

Class GeoPoint

com.google.android.maps.GeoPoint

java.lang.Object
    com.google.android.maps.GeoPoint

public class GeoPoint
    extends java.lang.Object

An immutable class representing a pair of latitude and longitude, stored as integer numbers of microdegrees.

Constructor Summary

GeoPoint(int latitudeE6, int longitudeE6)
Constructs a GeoPoint with the given latitude and longitude, measured in microdegrees (degrees * 1E6).
```

GPS for WFU: (latitude = 36.131378, longitude= -80.283661)
GeoPoint: 36131378, -80283661

(Often do this multiplication/integer conversion on fly – see examples)
Latitude/Longitude

WFU (36.13, -80.28)

Latitude: angular distance north (+) or south (-) of equator
Longitude: angular distance east(+) or west(-) of prime meridian
MapController

With the MapController, we can move to particular points, and zoom in and out.

```java
void mapController.setCenter(GeoPoint target) – jump to target location
void mapController.animateTo(GeoPoint target) – animate to target location
void mapController.setZoom(int zoomLevel) – programmatically zoom
```

- 1 is widest view
- 21 is nearest view (hard limit; actual limit may be less depending on imagery taken)
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mapView = (MapView) findViewById(R.id.map_view);
    mapView.setBuiltInZoomControls(true);
    mapController = mapView.getController();

    latText = (EditText) findViewById(R.id.lat_text);
    longText = (EditText) findViewById(R.id.long_text);
    goButton = (Button) findViewById(R.id.goButton);

    goButton.setOnClickListener(this);
}

protected boolean isRouteDisplayed() { return false; }
protected boolean isLocationDisplayed() { return false; }

public void onClick(View arg0) {
    if (arg0.getId() == R.id.goButton) {
        String latString = latText.getText().toString();
        String longString = longText.getText().toString();

        Double latitudeScaled = new Double(latString) * 1E6;
        Double longitudeScaled = new Double(longString) * 1E6;

        int latitudeInt = latitudeScaled.intValue();
        int longitudeInt = longitudeScaled.intValue();

        GeoPoint targetPoint = new GeoPoint(latitudeInt, longitudeInt);
        mapController.setCenter(targetPoint);
        mapController.setZoom(21);
    }
}
GeoPoints: Converting strings to integer microdegrees

Sources of lat/long information:
- GPS sensor: doubles → 36.131378
- EditText: Strings → “36.131378”
- From the Internet: Strings → “36.131378”

To convert to microdegrees:
Convert String or double to Double:

```java
Double latitudeScaled = new Double(input) * 1E6;
```
Get the int value associated with that double:

```java
int latitudeInt = latitudeScaled.intValue();
```
Geocoder

Geocoder: translation back and forth between latitude/longitude pairs and addresses

- *Forward geocoding*: Find the latitude and longitude of a given address
- *Reverse geocoding*: Find the address of a given latitude/longitude pair

The work is all actually done at Google – this class essentially passes the request over the network!

Also, a bug exists in the Emulator such that Geocoding doesn’t work in the Emulator! [does work on devices]
Geocoder

Geocoder

Needs an application context
Activity it is running in
and a Locale
Info about locale/language
Appropriate for our class
Locale.getDefault() or
Locale.English (English, USA)

Geocoder gc = new Geocoder(this, Locale.English);
Geocoder

Two main methods to request geocoding:

Forward geocoding:

List<Address> getFromLocationName(String address, int maxHits)

Reverse geocoding:

List<Address> getFromLocation(double lat, double long, int maxHits)

• If no matches, functions return null (which is not an empty list!).
• Object get(int x) is the appropriate List function to retrieve an address at position x from the list returned by these functions.
• Have to wrap in try-catch block to catch potential errors
List of Addresses

Address: Represents a street Address

- String getLocality() → City
- String getPostalCode() → Zip Code
- String getCountryName() → Country

- int getMaxAddressLineIndex() → number of lines in address
- String getAddressLine(int) → components of full address

- double getLatitude() → latitude
- double getLongitude() → longitude
Try-Catch Blocks

Java error ("exception")
detection and management

```java
try {
    // something risky
}

catch (Exception e) {
    // something went wrong
    // 'e' is a description of such
    // maybe I can work around it
}
```

Common uses of try-catch:
- File I/O
- Network I/O
Forward and Reverse Geocoding

Goals:

• If an address is entered, forward-geocode it and zoom the map to the appropriate coordinates

• If the map is centered on a spot, reverse-geocode the address from those coordinates
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mapView = (MapView)findViewById(R.id.map_view);
    mapView.setBuiltInZoomControls(true);
    mapController = mapView.getController();

    addressText = (EditText)findViewById(R.id.address_text);
    atocButton = (Button)findViewById(R.id.atoc_button);
    ctoaButton = (Button)findViewById(R.id.ctoa_button);
    atocButton.setOnClickListener(this);
    ctoaButton.setOnClickListener(this);

    geoCoder = new Geocoder(this, Locale.ENGLISH);
}

public void onClick(View arg0){
    if (arg0.getId() == R.id.atoc_button) {
        Log.v("Geocoder","a_to_c");
        try {
            List<Address> addresses = geoCoder.getFromLocationName(addressText.getText().toString(),1);
            Address address = addresses.get(0);
            Double latitudeScaled = new Double(address.getLatitude()) * 1E6;
            Double longitudeScaled = new Double(address.getLongitude()) * 1E6;

            int latitudeInt = latitudeScaled.intValue();
            int longitudeInt = longitudeScaled.intValue();

            GeoPoint targetPoint = new GeoPoint(latitudeInt, longitudeInt);
            mapController.setCenter(targetPoint);
            mapController.setZoom(21);
        } catch (Exception e) {
            Log.v("Geocoder", e.toString());
        }
    }
}
Reverse Geocoding

This is the other half of the `onClick` function on the previous slide.

```java
else if (arg0.getId() == R.id.ctoa_button) {
    Log.v("Geocoder","c_to_a");
    GeoPoint mapCenter = mapView.getMapCenter();
    double latitude = (mapCenter.getLatitudeE6() * 1.0) / 1E6;
    double longitude = (mapCenter.getLongitudeE6() * 1.0) / 1E6;
    try {
        List<Address> addresses = geoCoder.getFromLocation(latitude, longitude, 1);
        Address address = addresses.get(0);
        int numberOfAddressLines = address.getMaxAddressLineIndex();
        String theAddressString = ""
        for (int i = 0; i < numberOfAddressLines; i++)
        {
            if (i == (numberOfAddressLines-1))
                theAddressString += address.getAddressLine(i);
            else theAddressString += address.getAddressLine(i) + ", ";
        }
        addressText.setText(theAddressString);
    }
    catch (Exception e)
    {
        Log.v("Geocoder", e.toString());
    }
}
```