Android Programming
Lecture 8:

Activities, Odds & Ends, Two New Views

9/28/2011

Return Values from Activities: Callee Side

How does the Sub-Activity send back a response?

- Create an Intent to return
- Stuff the Intent with data using putExtra
- Call setResult(Intent, int) with the Intent to return and the returnCode integer
  - These values are given to the handler in the caller Activity
  - Activity.RESULT_OK and Activity.RESULT_CANCELED are two built-in return codes one can use
- Call finish() function
  - Finish kills the activity (pops it off the Activity stack)

Return Values from Activities: Callee Side

```
public class SearchResultsActivity extends Activity implements View.OnClickListener {

        TextView labelText;
        TextView resultsText;
        Button returnButton;
        int magicNumber;
        String searchTerm;

        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.results);

            labelText = (TextView)findViewById(R.id.search_results_label);
            resultsText = (TextView)findViewById(R.id.search_results);
            returnButton = (Button)findViewById(R.id.search_return_button);

            returnButton.setOnClickListener(this);

            Intent callingIntent = getIntent();
            searchTerm = callingIntent.getStringExtra("searchTerm");
            magicNumber = callingIntent.getIntExtra("magicValue",-1);

            labelText.setText(labelText.getText() + " " + searchTerm);
            resultsText.setText(""+magicNumber);
        }

        public void onClick(View args0) {
            Intent returnIntent = new Intent();
            returnIntent.putExtra("magicValue", magicNumber+1);
            setResult(Activity.RESULT_OK, returnIntent);
            finish();
        }
```

My example returns as its return value the magic number it was sent plus one

Return Values from Activities: Callee Side

If the user hits the physical “back” button on the phone, which “goes back an activity”:

- The caller’s *onActivityResult* function is called, but it receives a null Intent object and a return code of Activity.RESULT_CANCELED

- It IS possible to catch this in the callee and to handle it accordingly (return something other than Activity.RESULT_CANCELED)
  - Override *onBackPressed()* function

Return Values from Activities: Callee Side

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

    labelText = (TextView)findViewById(R.id.search_results_label);
    resultsText = (TextView)findViewById(R.id.search_results);
    returnButton = (Button)findViewById(R.id.search_return_button);

    returnButton.setOnClickListener(this);

    Intent callingIntent = getIntent();
    searchTerm = callingIntent.getStringExtra("searchTerm");
    magicNumber = callingIntent.getIntExtra("magicValue",-1);

    labelText.setText(labelText.getText() + " " + searchTerm);
    resultsText.setText(""+magicNumber);
}

public void onClick(View args0) {
    Intent returnIntent = new Intent();
    returnIntent.putExtra("magicValue", magicNumber+1);
    setResult(Activity.RESULT_OK, returnIntent);
    finish();
}

public void onBackPressed() {
    Intent returnIntent = new Intent();
    returnIntent.putExtra("magicValue", magicNumber-1);
    setResult(2, returnIntent);
    finish();
}
```

Inter-Application Communication

Intent Actions: Lots of built-in action descriptions, stored as constants in the Intent class

These are the current standard actions that Intent defines for launching activities (usually through `startActivity(Intent)`.

- ACTION_MAIN
- ACTION_VIEW
- ACTION_ATTACH_DATA
- ACTION_EDIT
- ACTION_PICK
- ACTION_CHOOSER
- ACTION_GET_CONTENT
- ACTION_DIAL
- ACTION_CALL
- ACTION_SEND
- ACTION_SENDTO
- ACTION_ANSWER
- ACTION_INSERT
- ACTION_DELETE
- ACTION_RUN
- ACTION_SYNC
- ACTION_PICK_ACTIVITY
- ACTION_SEARCH
- ACTION_WEB_SEARCH
- ACTION_FACTORY_TEST

Constant: Intent.ACTION_VIEW

Described in full here:
http://developer.android.com/reference/android/content/Intent.html

Inter-Application Communication

Making a request from your Activity to view a webpage:

Action: Intent.ACTION_VIEW

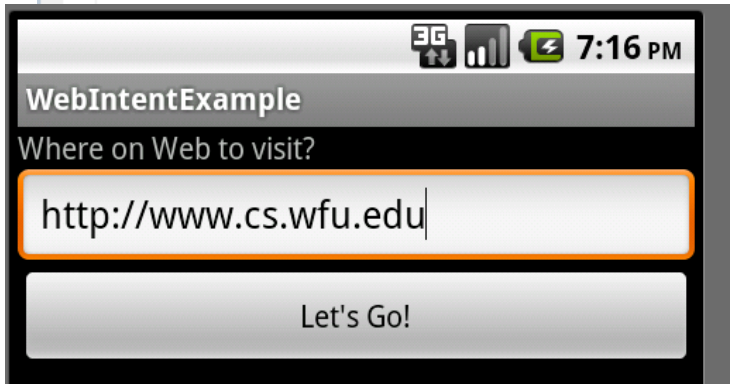
Data: Uri.parse(“http://www.cs.wfu.edu”)

(turns our string into a URI)

Inter-Application Communication Example

WebIntentExampleActivity.java

```
package turkett.csc191;

import android.app.Activity;

public class WebIntentExampleActivity extends Activity implements View.OnClickListener {

    Button goButton;
    EditText editText;

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

        goButton = (Button)findViewById(R.id.button);
        editText = (EditText)findViewById(R.id.edit_text);
        goButton.setOnClickListener(this);
    }

    public void onClick(View args0) {
        Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(editText.getText().toString()));
        startActivity(intent);
    }
}
```



Inter-Application Communication: Return Values

Sometimes we will want other Application Activities to return values to us, such as if we select data through the activity

- Choosing an image from the image gallery
- Choosing a contact from the contacts list

- Similarities in general handling of this scenario
- A lot of specific details on how to handle the returned data for each different scenario

Inter-Application Communication: Return Values – General Ideas

Make use of *startActivityForResult* functoin

Be sure to send in a requestCode

Make use of *onActivityResult* listener function

Activity will store a URI to the selected data in the data field of the returned Intent

Handle that URI in data-specific ways

Inter-Application Communication: Return Values – Image Gallery

To request an image to be selected from the image gallery:

Action: Intent.ACTION_GET_CONTENT

Uri: None (not asking particular data to be handled)

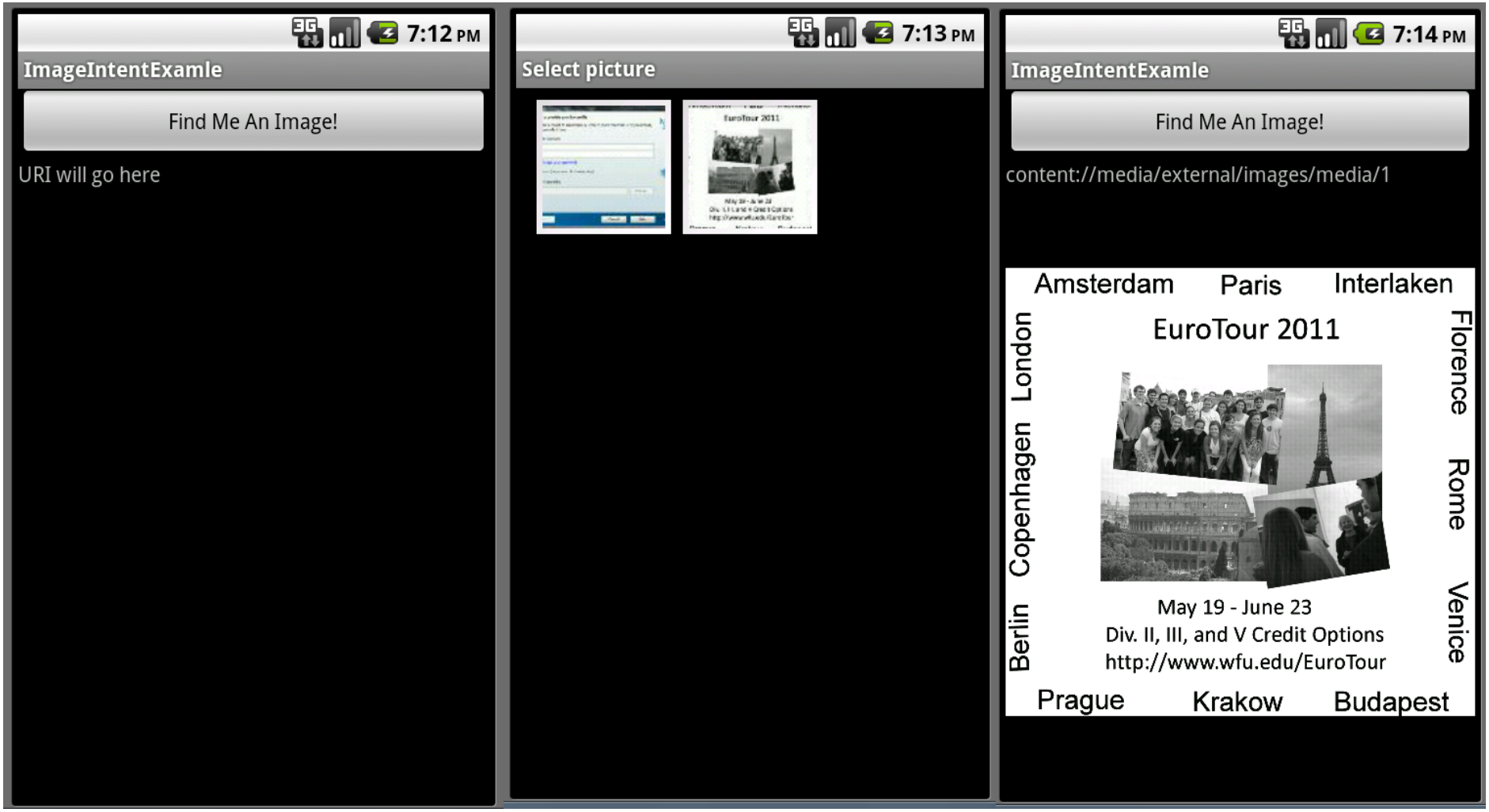
[NEW] Type: image/*

(types used here is a MIME type definition: http://en.wikipedia.org/wiki/Internet_media_type)

Inter-Application Communication: Return Values – Image Gallery

```
public class ImageIntentExamleActivity extends Activity implements View.OnClickListener {

    Button goButton;
    TextView uriView;
    ImageView imageSpace;

    int IMAGE_PICKER_REQUEST;

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

        goButton = (Button)findViewById(R.id.button);
        goButton.setOnClickListener(this);
        uriView = (TextView)findViewById(R.id.text_view);
        imageSpace = (ImageView)findViewById(R.id.image_view);
    }

    public void onClick(View arg0) {
        if (arg0.getId() == R.id.button) {
            Intent imageIntent = new Intent(Intent.ACTION_GET_CONTENT);
            imageIntent.setType("image/*");
            startActivityForResult(imageIntent, IMAGE_PICKER_REQUEST);
        }
    }

    public void onActivityResult(int requestCode, int resultCode, Intent returnedIntent) {
        super.onActivityResult(requestCode, resultCode, returnedIntent);
        if ((requestCode == IMAGE_PICKER_REQUEST) && (resultCode == RESULT_OK)) {
            Uri returnedImageURI = returnedIntent.getData();
            uriView.setText(returnedImageURI.toString());
            imageSpace.setImageURI(returnedImageURI);
        }
    }
```

Inter-Application Communication: Return Values – Image Gallery



Inter-Application Communication: Return Values – Contacts List

To request a contact be selected from the contacts list:

Action: Intent.ACTION_GET_CONTENT

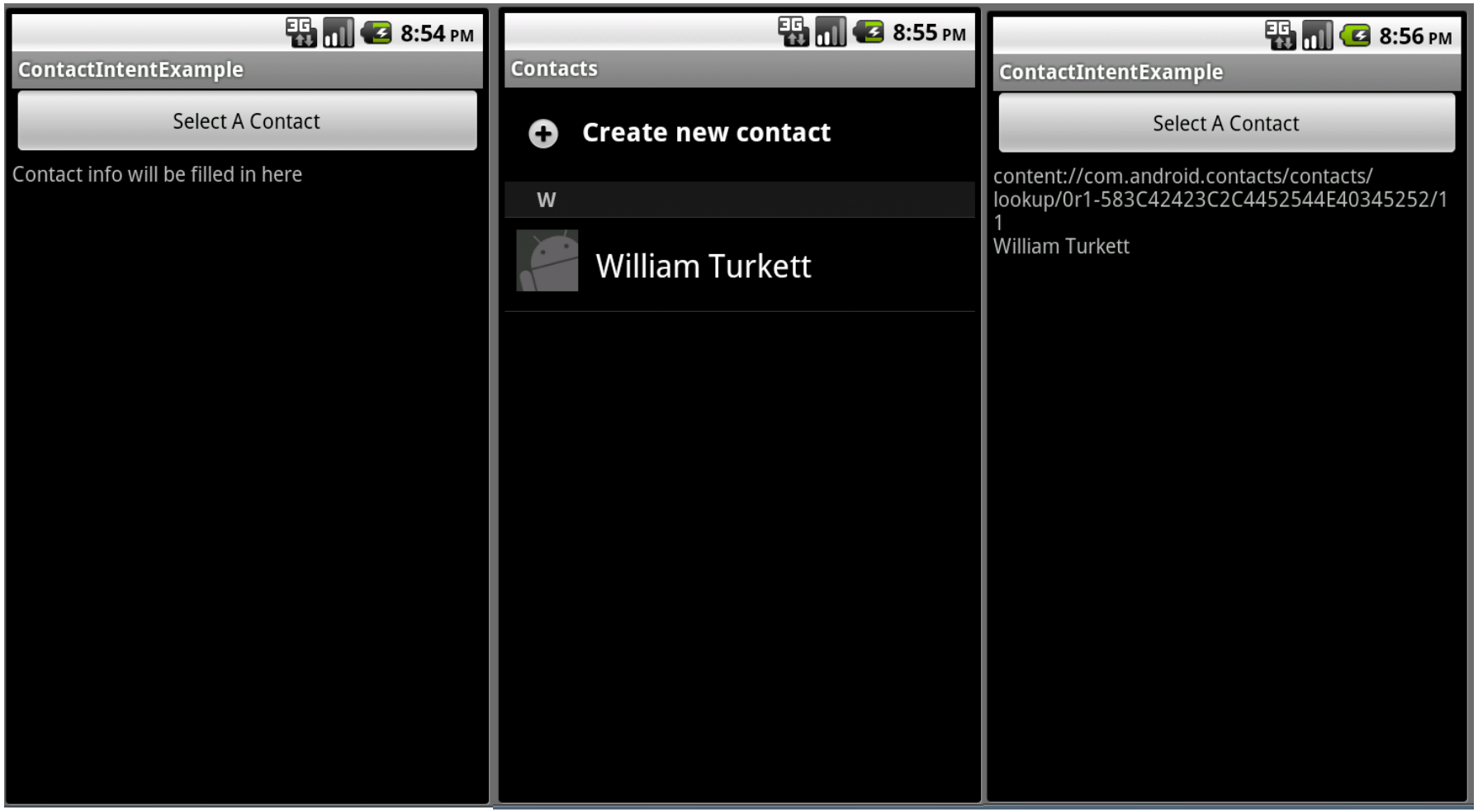
Uri: None (not asking particular data to be handled)

Type: ContactsContract.Contacts.*CONTENT_ITEM_TYPE*

Inter-Application Communication: Return Values – Contacts List

```
public class ContactIntentExampleActivity extends Activity implements View.OnClickListener {

    Button button;
    TextView textView;
    int PICK_CONTACT_REQUEST;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        button = (Button)findViewById(R.id.button);
        textView = (TextView)findViewById(R.id.text_view);
        button.setOnClickListener(this);
    }

    public void onClick(View args0) {
        if (args0.getId() == R.id.button) {
            Intent intent = new Intent(Intent.ACTION_GET_CONTENT);
            intent.setType(ContactsContract.Contacts.CONTENT_ITEM_TYPE);
            startActivityForResult(intent,PICK_CONTACT_REQUEST);
        }
    }

    public void onActivityResult(int requestCode, int responseCode, Intent returnedIntent)  {
        if ((requestCode == PICK_CONTACT_REQUEST) && (responseCode == Activity.RESULT_OK))  {
            textView.setText(returnedIntent.getData().toString());
            Cursor cursor =  managedQuery(returnedIntent.getData(), null, null, null, null);
            while (cursor.moveToNext()) {
                   String contactId = cursor.getString(cursor.getColumnIndex(ContactsContract.Contacts._ID));
                   String name = cursor.getString(cursor.getColumnIndexOrThrow(ContactsContract.Contacts.DISPLAY_NAME));
                   textView.setText(textView.getText()+"\n"+contactId+"\n"+name);
               }
        }
    }
}
```

Inter-Application Communication: Return Values – Contacts List



Saving Killed Activities

- Changing the orientation of an Android device:
  - Causes the activity to be destroyed (activity is *killed*)
  - Then restarted

- There are other ways that Activities can be killed as well

- The GUI state of the app is automatically saved
- The data "state" of the app is not automatically saved

- Mechanisms allowing you to save the state of the Activity before death are available and employ a notion you are already aware of

Intents and Extras

- With an Intent, we could store arbitrary additional information in the Intent using the *putExtra(key,value)* function

- A *Bundle* is a class where we can store information using key-value pairs
  - We could create a Bundle of parameters for our Intent and then just associate that Bundle with the Intent

A Bundle of Extras

- Bundle *put* and *get* member functions are typed in their names:
  - *putString(String,String), getString*
  - *putDouble, getDouble*
  - …

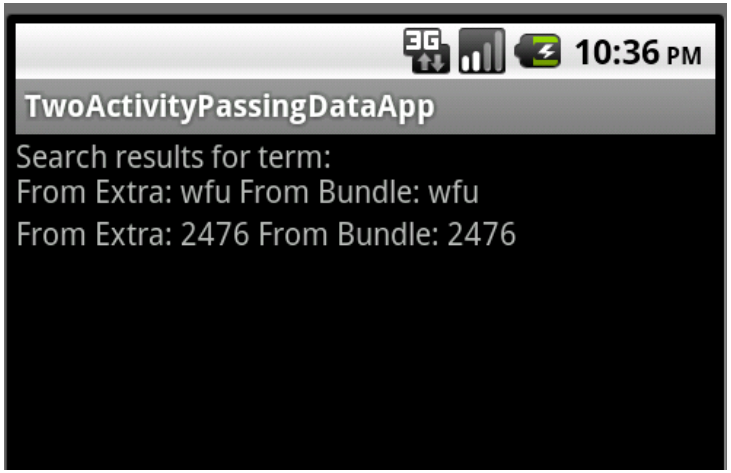
- To associate a Bundle with an Intent, use: *intentGoesHere.putExtras(bundleGoesHere);*
- To retrieve a Bundle from an Intent, use *intentGoesHere.getExtras(bundleGoesHere);*

Using Bundles with Intents: Caller (Sender) Side

```
public class TwoActivityPassingDataAppActivity extends Activity implements View.OnClickListener {

    Button searchButton;
    EditText searchTextField;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        searchTextField = (EditText)findViewById(R.id.search_field);
        searchButton = (Button)findViewById(R.id.search_button);
        searchButton.setOnClickListener(this);
    }

    public void onClick(View arg0) {
        Intent intent = new Intent(TwoActivityPassingDataAppActivity.this,
                    SearchResultsActivity.class);
        //intent.putExtra("searchTerm", searchTextField.getText().toString());
        //intent.putExtra("magicValue", 2476);
        Bundle dataBundle = new Bundle();
        dataBundle.putString("searchTerm",searchTextField.getText().toString());
        dataBundle.putInt("magicValue", 2476);
        intent.putExtras(dataBundle);
        startActivity(intent);
    }
```

Using Bundles with Intents: Callee (Receiver) Side

```
public class SearchResultsActivity extends Activity {

        TextView labelText;
        TextView resultsText;

        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.results);

            labelText = (TextView)findViewById(R.id.search_results_label);
            resultsText = (TextView)findViewById(R.id.search_results);

            Intent callingIntent = getIntent();
            String searchTerm = callingIntent.getStringExtra("searchTerm");
            int magicNumber = callingIntent.getIntExtra("magicValue",-1);


            labelText.setText(labelText.getText() + "\n" + "From Extra: " + searchTerm);
            resultsText.setText("From Extra: "+magicNumber);

            Bundle dataBundleReceived = callingIntent.getExtras();
            searchTerm = dataBundleReceived.getString("searchTerm");
            magicNumber = dataBundleReceived.getInt("magicValue",-1);
            labelText.setText(labelText.getText() + " " + "From Bundle: " + searchTerm);
            resultsText.setText(resultsText.getText() + " " + "From Bundle: " +magicNumber);

        }
```



Can intermingle use of *getXExtra* directly from Intent and *getX* from Intent's Bundle

Saving Killed Activities

- Before an Activity is destroyed:
  - *onSaveInstanceState(Bundle bundle)* is called, providing a Bundle to save state information in
- When an Activity is started, that saved Bundle is passed to
  - *onCreate(Bundle bundle)*

*** These do not work when <u>you</u> do a Force Kill ***

Saving Killed Activities

```
public class SavingTwoActivityReturnValueActivity extends Activity implements View.OnClickListener {

    Button searchButton;
    EditText searchTextField;
    int magicValue;

    // unique id for sub-activity
    private int RESULTS_ACTIVITY = 1;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        searchTextField = (EditText)findViewById(R.id.search_field);
        searchButton = (Button)findViewById(R.id.search_button);
        searchButton.setOnClickListener(this);
        if (savedInstanceState != null) {
            Log.v("Turkett", "Saved State Available!");
            magicValue = savedInstanceState.getInt("magicValue");
            searchTextField.setText(savedInstanceState.getString("searchTerm"));
        }
        else {
            Log.v("Turkett", "Saved State Unavailable!");
            magicValue = 2476;
        }
    }

    public void onClick(View arg0) {...}

    public void onActivityResult(int requestCode, int resultCode, Intent returnData) {...}

    public void onSaveInstanceState(Bundle savedData) {
        savedData.putInt("magicValue", magicValue);
        savedData.putString("searchTerm", searchTextField.getText().toString());
    }
}
```

Restoring state

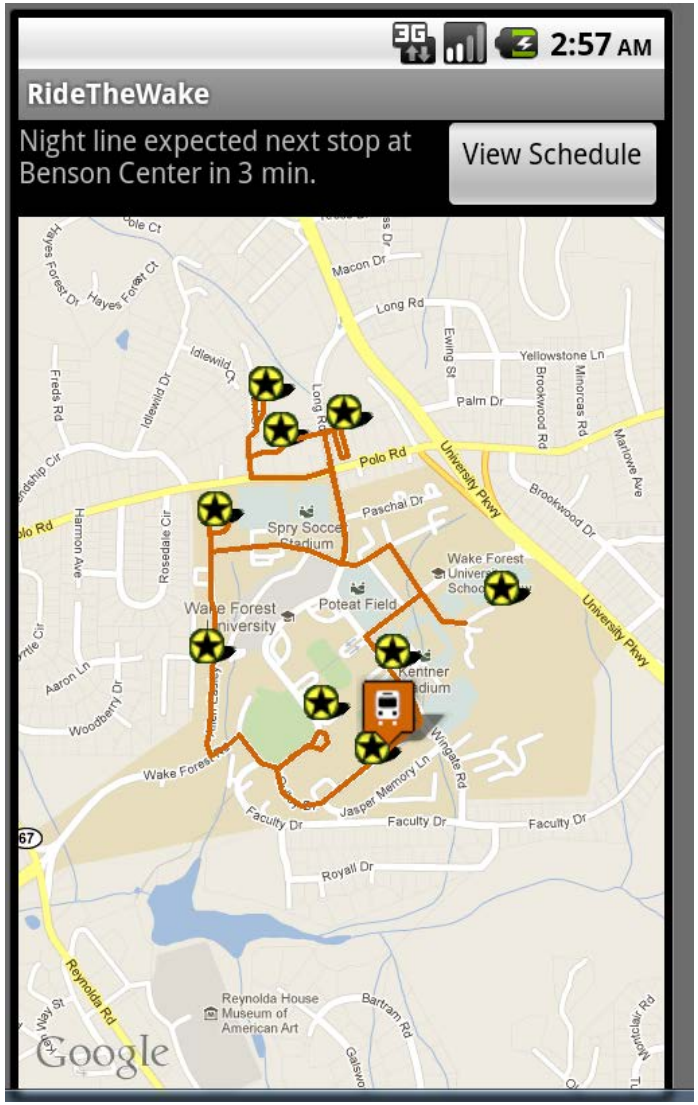
Saving state

Saving Killed Activities

- See SavingTwoActivityReturnValue example application

- Example requires setting Emulator Dev Tools to only allow 1 process (app) max running at a time

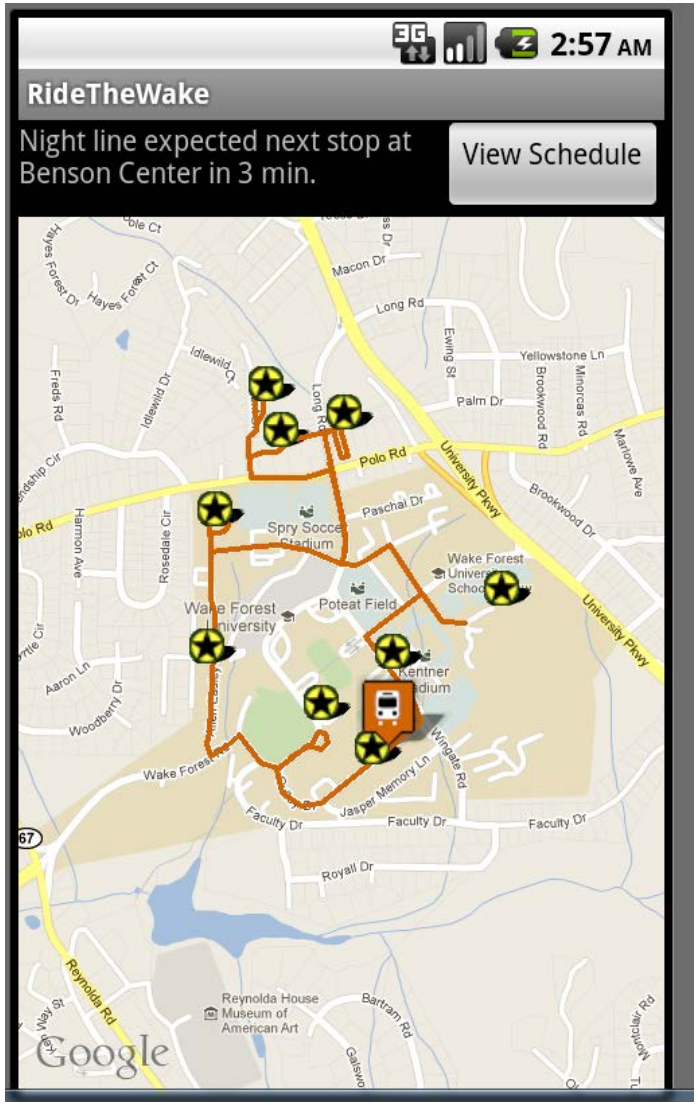
MapView



- 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 sigining 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** > **Prefs** > **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