Accelerating Information Technology Innovation

http://aiti.mit.edu/program/philippines-summer-2012/

Philippines Summer 2012
Lecture 7 –Android Activities, Event Handlers and Intents
July 12, 2012
Agenda

• Android Activities
• Event Handlers
• Intents
Android Activities
Android Activities

• Apps are composed of Activities
• Just like a Website contains many web pages, an Android app contains many Activities
• Each Activity is a “page”, or user interface view
• Activities are interconnected with one another, just like web pages contain navigation from one web page to another
• Each Activity contains a Layout View with specific view objects
Multiple Activities

• An android application may consist of one or multiple Activity objects
• Only one activity can be the main activity (i.e. the entry point to the user)

```xml
<application android:label="Snake on a Phone">
  <activity android:name="Snake"
    android:theme="@android:style/Theme.NoTitleBar"
    android:screenOrientation="portrait"
    android:configChanges="keyboardHidden|orientation">
    <intent-filter>
      <action android:name="android.intent.action.MAIN"/>
      <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
  </activity>
  define other activities here...
</application>
```
Basic Activity Lifecycle
Basic Activity Lifecycle
Destroy Application

• Two ways of destroying an application

  • `onPause()` -> `onStop()` -> `onDestroy()`
    – Most common. Most cleanup done on `onPause()` and `onStop()`
    – Kill background threads in `onDestroy()`

  • `onCreate()` -> `finish()` -> `onDestroy()`
    – Activity operates as a temporary decision maker. Destroy immediately after being created.
Basic Activity Lifecycle
Actions of onPause()

• Stop animations or other ongoing actions that could consume CPU.

• Commit unsaved changes, but only if users expect such changes to be permanently saved when they leave (such as a draft email).

• Release system resources, such as broadcast receivers, handles to sensors (like GPS), or any resources that may affect battery life while activity is paused and the user does not need them.

• Keep it simple – avoid intensive computation. User may come back to it soon.
Basic Activity Lifecycle

- **Created**
  - `onCreate()`

- **Started** (visible)
  - `onStart()`
  - `onStart()`

- **Resumed** (visible)
  - `onResume()`

- **Paused** (partially visible)
  - `onPause()`
  - `onStop()`

- **Stopped** (hidden)
  - `onStop()`
  - `onDestroy()`

- **Destroyed**
Actions of onResume()

• Called from Started and Paused states
• Initialize components that you release during `onPause()` and perform any other initializations that must occur each time the activity enters the Resumed state (such as begin animations and initialize components only used while the activity has user focus).
• Counterpart of `onPaused()`
Save your Activity State

- Additional methods. Allows you to specify additional state data you would like to save in case the Activity instance must be recreated.
- System calls these methods as an Activity is being destroyed or restored (recreated).
Example: onSaveInstanceState()

```java
static final String STATE_SCORE = "playerScore";
static final String STATE_LEVEL = "playerLevel";
...

@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    // Save the user's current game state
    savedInstanceState.putInt(STATE_SCORE, mCurrentScore);
    savedInstanceState.putInt(STATE_LEVEL, mCurrentLevel);

    // Always call the superclass so it can save the view hierarchy state
    super.onSaveInstanceState(savedInstanceState);
}
```
Recreate an Activity

• Use `onCreate()` – check if bundle is not empty

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); // Always call the superclass first

    // Check whether we're recreating a previously destroyed instance
    if (savedInstanceState != null) {
        // Restore value of members from saved state
        mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
        mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
    } else {
        // Probably initialize members with default values for a new instance
    }
    // Probably initialize members with default values for a new instance
}
```
Recreate an Activity

- Better way – use onRestoreInstanceState()
- No need to check if Bundle is empty
- Called by system after the onStart() method

```java
public void onRestoreInstanceState(Bundle savedInstanceState) {
    // Always call the superclass so it can restore the view hierarchy
    super.onRestoreInstanceState(savedInstanceState);

    // Restore state members from saved instance
    mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
    mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
}
```
Resources

• Managing the Activity Lifecycle

• Images and source code examples taken from site above
Event Handlers
Handling events

• Listen to events using callback methods:
  – onClick()
  – onLongClick()
  – onFocusChange()
  – onKeyDown()
  – onTouch()
  – onCreateContextMenu()
Example: Event-handling with Buttons

Method 1: Define callbacks using code

```java
// Create an anonymous implementation of OnClickListener
private OnClickListener mCorkyListener = new OnClickListener()
{
    public void onClick(View v)
    {
        // do something when the button is clicked
        
    }
};

protected void onCreate(Bundle savedInstanceState)
{
    // Capture our button from layout
    Button button = (Button)findViewById(R.id.corky);
    // Register the onClick listener with the implementation above
    button.setOnClickListener(mCorkyListener);
}
```

Method 2: Define callbacks in Layout XML files

```
<Button android:id="@+id/button1" android:layout_width="80px"
        android:layout_height="fill_parent" android:onClick="clickhandler"
        android:text="1">
</Button>
```

```java
public void clickhandler(View clickedobject) {
    int idofclickedobject = clickedobject.getId();
    switch (idofclickedobject) {
    case R.id.button1:
        //do something
        break;
    }
}
```
Multiple Activities

• An android application consists of multiple Activity objects
• Each Activity is like one “page” of the app
• Only one activity can be the main activity

```xml
<application android:label="Snake on a Phone">
    <activity android:name="Snake"
        android:theme="@android:style/Theme.NoTitleBar"
        android:screenOrientation="portrait"
        android:configChanges="keyboardHidden|orientation">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    define other activities here...
</application>
```
Multiple Activities, example:

Main Activity (first thing you see when App starts)

Second Activity (clicking on a button on the Main Activity brings user to this one)
Switching between Activities

Step 1: Define all Activities in your App in the AndroidManifest.xml file

```xml
<application android:name=".MyApplication" android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".OneActivity" android:label="@string/app_name">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".AnotherActivity" android:label="picture capture" />
</application>
```

Step 2: Switch from Main Activity to the activity defined in `AnotherActivity.class`, using Intent objects.

```java
Intent intent = new Intent(this, AnotherActivity.class);
startActivity(intent);
```
Passing data between Activities

in your current activity, create an intent

```java
Intent i = new Intent(getApplicationContext(), ActivityB.class);
i.putExtra(key, value);
startActivity(i);
```

then in the other activity, retrieve those values.

```java
Bundle extras = getIntent().getExtras();
if(extras !=null) {
    String value = extras.getString(key);
}
```

Note: you can use the putExtra method to add data in key value pairs to the Intent. The key must be a String object but the value can be any of the following: integer, integer[], float, float[], double, double[], String, String[], etc…

Then, you fetch that data in the second activity using the `.getExtras().getString(key)` approach.
Other ways to exchange data between Activities

- Intent approach is best for *primitive* data types that don’t need to last forever (i.e. they are *not persistent*).
- For *primitive* types that need to last forever (i.e. *persistent* objects), use Preferences.
- For *non-primitive* types that are *not persistent*:
  - Public Static Fields
  - Maintain global application state in the Application class (all Activity objects have access to this).
- For non-primitive types that are *persistent*:
  - Use ContentProvider, SQL Database on the phone, Files, etc.
Intents
Intent

• An object that provides runtime binding between separate components (such as two activities).

• The **Intent** represents an app’s "intent to do something."

• You can use an **Intent** for a wide variety of tasks, but most often they’re used to start another activity.

• Intents can be **implicit** or **explicit**
Explicit Intent

• Specifies the exact recipient activity
• Add additional information to the intent
• Use `startActivity()` to send the intent

```java
/** Called when the user selects the Send button */
public void sendMessage(View view) {
    Intent intent = new Intent(this, DisplayMessageActivity.class);
    EditText editText = (EditText) findViewById(R.id.edit_message);
    String message = editText.getText().toString();
    intent.putExtra(EXTRA_MESSAGE, message);
    startActivity(intent);
}
```
Implicit Intent

• Send a request to open an activity based on an "action" it would like to perform.
• Specify the action to perform, not the activity to invoke.
• Uri – Uniform Resource Identifier

```java
Uri webpage = Uri.parse("http://www.android.com");
Intent webIntent = new Intent(Intent.ACTION_VIEW, webpage);
```
Implicit Intent (cont)

• Example: View a Map

```java
// Map point based on address
Uri location = Uri.parse("geo:0,0?q=1600+Amphitheatre+Parkway,+Mountain+View,+Cali
// Or map point based on latitude/longitude
// Uri location = Uri.parse("geo:37.422219,-122.08364?z=14"); // z param is zoom level
Intent mapIntent = new Intent(Intent.ACTION_VIEW, location);
```

• Example: Initiate a Phone Call

```java
Uri number = Uri.parse("tel:5551234");
Intent callIntent = new Intent(Intent.ACTION_DIAL, number);
```
Verify an App is available to Receive the Intent

• Your app will crash if there is no app to receive your (implicit) intent.
• Example how to check:

```java
PackageManager packageManager = getPackageManager();
List<ResolveInfo> activities = packageManager.queryIntentActivities(intent, 0);
boolean isIntentSafe = activities.size() > 0;
```
Start the Activity with an Intent

• Example:

```java
// Build the intent
Uri location = Uri.parse("geo:0,0?q=1600+Amphitheatre+Park");
Intent mapIntent = new Intent(Intent.ACTION_VIEW, location);

// Verify it resolves
PackageManager packageManager = getPackageManager();
List<ResolveInfo> activities = packageManager.queryIntentActivities(mapIntent, 0);
boolean isIntentSafe = activities.size() > 0;

// Start an activity if it's safe
if (isIntentSafe) {
    startActivity(mapIntent);
}
```
Get a Result from an Intent

• Call an Activity and get a result back
• Examples: Call Camera app and get picture taken, call Contacts app and get a certain contact
• Use `startActivityForResult()` instead of `startActivity()`
Get a Result from an Intent (cont)

• Example of `startActivityForResult()` usage

```java
static final int PICK_CONTACT_REQUEST = 1;  // The request code
...
private void pickContact() {
    Intent pickContactIntent = new Intent(Intent.ACTION_PICK, new Uri("content://com.picks/contact");  // Show user only contacts w/ phone number
    pickContactIntent.setType(Phone.CONTENT_TYPE);
    startActivityForResult(pickContactIntent, PICK_CONTACT_REQUEST);
}
```
Receive the Result

- **Use of** `onActivityResult()`
- `requestCode` = same as from start activity
- `resultCode` = `RESULT_OK`, `RESULT_CANCELED`

```java
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    // Check which request we're responding to
    if (requestCode == PICK_CONTACT_REQUEST) {
        // Make sure the request was successful
        if (resultCode == RESULT_OK) {
            // The user picked a contact.
            // The Intent's data Uri identifies which contact was selected.

            // Do something with the contact here (bigger example below)
        }
    }
}
```
Allow other Apps start your Activity

• Add an intent filter in AndroidManifest.xml

```xml
<activity android:name="ShareActivity">
  <!-- filter for sending text; accepts SENDTO action with sms URI schemes -->
  <intent-filter>
    <action android:name="android.intent.action.SENDTO"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:scheme="sms"/>
    <data android:scheme="smsto"/>
  </intent-filter>
  <!-- filter for sending text or images; accepts SEND action and text or image data -->
  <intent-filter>
    <action android:name="android.intent.action.SEND"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="image/*"/>
    <data android:mimeType="text/plain"/>
  </intent-filter>
</activity>
```
Handle the Intent in your Activity

```java
@override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);

    // Get the intent that started this activity
    Intent intent = getIntent();
    Uri data = intent.getData();

    // Figure out what to do based on the intent type
    if (intent.getType().indexOf("image/") != -1) {
        // Handle intents with image data ...
    } else if (intent.getType().equals("text/plain")) {
        // Handle intents with text ...
    }
}
```
If you want to return a result to the activity that invoked yours, simply call `setResult()` to specify the result code and result `Intent`. When your operation is done and the user should return to the original activity, call `finish()` to close (and destroy) your activity. For example:

```java
// Create intent to deliver some kind of result data
Intent result = new Intent("com.example.RESULT_ACTION", Uri.parse("content://result_uri")
setResult(Activity.RESULT_OK, result);
finish();
```
Resources

• Intents

• Intents Filters in your Activity
Lab 7

- Paper Prototyping Exercise
- Android Calculator